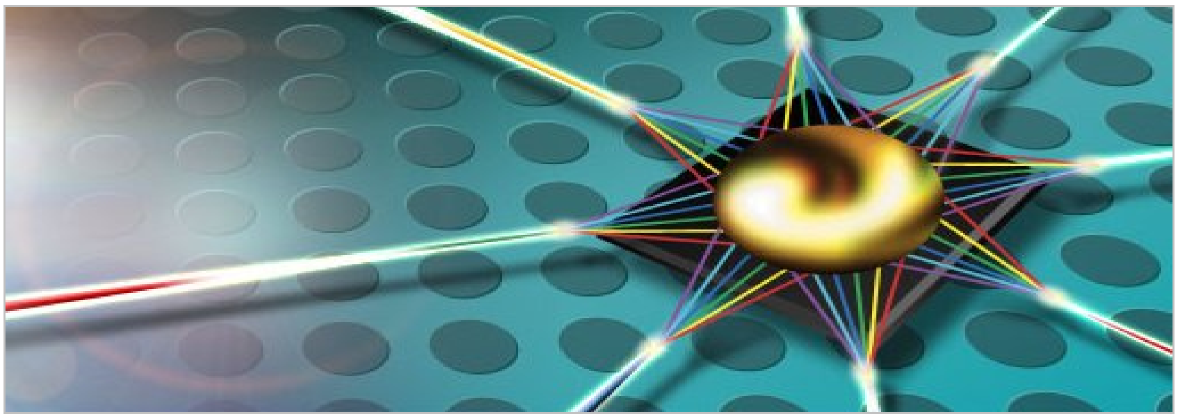
NPS-400 Architectural Specifications

Network Processor for Smart Networks

Document Version 1.8



Document Number: 27-8101-08

EZchip Semiconductor Ltd.
1 Hatamar Street, P.O. Box 527
Yokneam 20692, Israel

EZchip Semiconductor Inc.
2700 Zanker Road, Suite 150
San Jose, CA 95134, USA

Preface

EZchip Semiconductor Inc.
2700 Zanker Road, Suite 150,
San Jose, CA 95134, USA
Tel: (408) 520-3700, Fax: (408) 520-3701

EZchip Semiconductor Ltd.
1 Hatamar Street, PO Box 527,
Yokneam 20692, Israel
Tel: +972-4-959-6666, Fax: +972-4-959-4166

Email: info@ezchip.com, Web: www.ezchip.com

Please send comments regarding the documentation and/or product to supportNP@ezchip.com.

About this Manual

This document is intended to provide an architectural overview and detailed explanation of the NPS-400 service capable network processor.

Related Documentation

The following documents contain supplementary information:

DOCUMENT	***CONTENT***
NPS-400 Hardware Specifications	Hardware specifications for the NPS-400, including pinout, electrical specifications, mechanical specifications and timing.
NPS-400 Programming Manual	Offers programmers with the basic information required to code a data plane processing application for the EZchip NPS-400 network processor.
CTOP Architecture Manual	Architecture, programming and instruction set for NPS C-programmable Task Optimized Processors (CTOPs).

Terminology and Conventions

General

The following terminology is used throughout this document:

TERM	***DESCRIPTION***
NPS	Refers to the EZchip NPS-400 network processor device and/or software simulator.

This Document

The following is a brief description of the contents of each chapter:

SECTION	NAME	DESCRIPTION
Section 1	**Product Overview**	Overview of the NPS-400 network processor, product features and highlights. It includes an overview of the software architecture, which provides a flexible programming environment for a wide variety of SW architecture implementations.
Section 2	**Physical Description**	Describes the NPS-400 device interfaces, MACs, and port mapping.
Section 3	**Data Flow**	Describes the main building blocks and high-level architectural overview of the data flow through the NPS. It includes an overview of several of the data structures used in the NPS that affect packet processing or programming, e.g. job, frame, and frame buffers.
Section 4	**Network Processor Cluster (NPC)**	High-level architectural overview of the Network Processor Clusters (NPC) inside the NPS.
Section 5	**C-programmable Task Optimized Processors (CTOP)**	High-level architectural overview of the C-programmable Task Optimized Processors (CTOP) inside the NPS.
Section 6	**Memory Architecture**	Describes the physical memory components that comprise the NPS memory architecture.
Section 7	**Input Classification Unit (ICU)**	Describes the hardware-based classification for incoming packets.
Section 8	**Processor Management Unit (PMU)**	High-level architectural overview of the Processor Management Unit (PMU) inside the NPS.
Section 9	**Buffer Management Unit (BMU)**	High-level architectural overview of the Buffer Management Unit (BMU) inside the NPS.
Section 10	**Traffic Management**	Describes the NPS's queuing and service provisioning features.
Section 11	**Statistics Management**	Describes the implementation of statistics counters.
Section 12	**Look-up Acceleration**	Describes the look-up acceleration features and search data structures offered by the NPS.
Section 13	**Deep Packet Inspection (DPI)**	Provides a high-level description of the Deep Packet Inspection (DPI) capabilities of the NPS.
Section 14	**Cryptographic Features**	Describes the cryptographic service accelerations provided by the device.
Section 15	**Atomic Operations**	Describes the available atomic operations and semaphores.
Section 16	**TCAM Interfaces**	Describes the internal and external TCAM interfaces.
Section 17	**Algorithmic TCAM Extension**	Describes an NPS-400 alternative to the conventional TCAM approach.
Section 18	**PCI Express Controller**	Describes the PCI Express Controller for the PCIe interfaces,
Section 19	**Congestion Management and Flow Control**	Describes NPS device and system resources congestion management and traffic flow control mechanisms.
Section 20	**Precision Clock Synchronization Support**	Describes how the Precision Clock Synchronization application is supported by NPS-400 using the IEEE 1588 standard.
Section 21	**General Interrupt Controller**	The general interrupt controller aggregates services that are required to signal interrupts and process interrupt acknowledges.
Section 22	**Power-up and Initialization**	Describes the NPS-400 power-up, reset sequence and device initialization steps.
Appendix A	**Acronyms and Abbreviations**	Lists the acronyms and abbreviations used in this document.

Revision History

REVISION	***DATE***	***DESCRIPTION OF MODIFICATIONS***
1.8	Jan. 12, 2016	Chapter 2 Physical Description: ▪ SerDes Allocation Scheme: PCIe interface updated. ▪ Change in number of lanes supported on ILKN-LA Tx interface: 24, 16, 12, 8 or 4 lanes starting at SerDes lane 47. ▪ A configuration of both ILKN-LA and PCIe (lanes 40-47) interfaces on the same side is not supported. ILKN-LA can be used together with the x1 PCIe interface (lane 48). ▪ SerDes lane 49 may be used as SGMII Debug port. ▪ SerDes overclocking to 12.5 Gbps is not supported on XFI / XLAUI / CAUI interfaces. ▪ Interlaken and GCI interfaces are rate limited to 12.5Gbps, instead of 15Gbps. Chapter 3 Data Flow ▪ Update to descriptions of several of the data structures used in the NPS that affect packet processing or programming, e.g. job, frame, and frame buffers. Chapter 11 Statistics Management: ▪ Supports 512M on-demand counters (updated and collected by the application); 256M of these are allocated for EMEM buffer multicast counters. Chapter 12 Look-up Acceleration: ▪ IMEM results are limited to 32 bytes maximum. Chapter 18 PCI Express Controller: ▪ Update to PCIe interface allocation on SerDes lanes.
1.7	Aug. 18, 2015	Package: HFCBGA ball size is 0.64 mm. Chapter 2 Physical Description and Chapter 16 TCAM Interfaces: Removed limitation that ILKN-LA interface was only supported on one side of the device. Chapter 3 Data Flow: updated overview of jobs, frames and buffers. Chapter 8 Processor Management Unit (PMU): Gross Checksum field was returned to the Job Descriptor (JD) structure. Erroneously removed from last release of document. Field exists in Rx frames but not in replicated frames. Chapter 10: Traffic Management: De-featured TM directly attached flows/classes/ subports. (Does not affect the fast queues.) Chapter 11 Statistics Management: ▪ Hierarchical Token Buckets added. ▪ OAM Watchdog counter modes (accumulative window and sliding window) were re-introduced. Chapter 14 Cryptographic Features: The features outlined in this chapter are relevant only for NPS-400 models that support cryptographic features (order no. 207850xx). Chapter 16 TCAM Interfaces: Corrections to ILKN-LA interfaces feature list. Chapter 19 Congestion Management and Flow Control: typical latency added for Input FIFO based Link Level FC Generation; typical response time added for Link Level Flow Control.
1.6	May 19, 2015	Updates throughout the document including but not limited to: Chapter 2 Physical Description: ▪ Clarifications regarding interface engines. Each IFE serves 12 SerDes lanes and there are four IFEs on each device side. ▪ SerDes Allocation Scheme updated: 40GbE option B added. ▪ Ethernet MAC Supported Configurations section added. ▪ Enhancements to interface descriptions of their supported features/ configurations. ▪ Interlaken interface supports 128 channels per side; does not support channel interleaving, no segment mode. ▪ A configuration of both ILKN-LA and GCI interfaces on the same side is not supported. Chapter 3 Data Flow: Corrected Figures 3-5, 3-6 and 3-7 on data flow to show the

REVISION	DATE	DESCRIPTION OF MODIFICATIONS
		ICU after the RxIF. Chapter 6 Memory Architecture: EBP over LLP Protection of Frame Buffer added. Chapter 7 Input Classification Unit (ICU): Per Port ID configurations updated. Chapter 8 Processor Management Unit (PMU): Gross Checksum field removed from Job Descriptor (JD) structure format. Application Aware Job Scheduling expanded. Chapter 9 Buffer Management Unit (BMU): SW pool updates. Chapter 11 Statistics Management: OD watchdog scan support, WD messages and watchdog RTC mechanism have been removed. Chapter 14 Cryptographic Features chapter: Support for RC-4 removed. Chapter 20 Precision Clock Synchronization Support: Table 20-2 Header for RTC time stamping (2-step format) updated with HOFS[7:0] field in bits 55:48. RTC Synchronization Modes expanded.
1.5	Nov. 25, 2014	New chapters added: ▪ Algorithmic TCAM Extension. ▪ Precision Clock Synchronization Support. ▪ Power-up and Initialization Changes to interfaces: ▪ ILKN1 interface removed. ▪ New 12-lane CAUI mode is used together with 12-lane XFI mode. ▪ Ethernet Fabric Adapter (EFA) removed. ▪ 2x remote management ports support 1GbE SGMII/1000BASE-X only; 10GE SFI/XFI Host port is obsolete. 1GbE Host port is supported in SerDes lane 49 only. Package is HFCBGA 2892 pins. Chapter 6 Memory Architecture: Added “When the data cache is configured to be shared by all threads, if PDEMEM memory space is used it must be programmed to be non-cached.” Chapter 12 Look-up Acceleration: sections on Longest Prefix Match and UltraIP were improved. PCI Express Registers Appendix was moved to the *NPS-400 Configuration Registers Reference Manual*.
1.4	Sept. 8, 2014	Updates throughout the document including but not limited to: Chapter 2 Physical Description: RXAUI Host port removed. Chapter 4 Network Processor Cluster (NPC): Protocol Decoder added. Chapter 8 Processor Management Unit (PMU): NPS resource grouping removed. Chapter 10 Traffic Management: Interface Mapping Table updated; TM Statistics section expanded. Chapter 12 Lookup Acceleration: general updates. Chapter 13 Deep Packet Inspection: completely revised. Chapter 17 PCI Express Controller: Clarification added: Only PCIe on the west side can access the NPS device SoC configuration and management address space. The control plane CPU must be connected on the west side. Chapter 21 General Interrupt Controller: new chapter added.
1.32		Updates throughout the document including but not limited to: DDR4 DRAM capacity increased from 48GB to 96GB. Chapter 8 Processor Management Unit (PMU): SW bypass option removed. Chapter 11 Statistics Management chapter: completely updated; removed limitation of 16M watchdog counters. Chapter 17 and Appendix B: fixed inconsistencies. NPS supports 2 Virtual Channels (VCs), 4 Physical Functions with 128 Virtual Functions (VFs).
1.31	May 4, 2014	Updates to Chapter 19: Congestion Management and Flow Control.
1.3	April 30, 2014	Updates throughout the document including but not limited to: Interface updates: ▪ 1G ports (SGMII/QSGMII) removed. ▪ One CGI port per side; 4 or 8 lanes. ▪ One PCIe port per side; up to 8 lanes. ▪ ILKN-LA is supported only on either side of chip; not both sides.

REVISION	DATE	DESCRIPTION OF MODIFICATIONS
		Chapter 10: Traffic Management: ▪ L4 Flows increased to 2 x 0.5M or 1 x 1M; L3 Classes increased to 2x 32K. ▪ New sections added: Congestion Change based Probabilistic Report Generation Method, Congestion Segmentation Method. ▪ Changed to 144 TM output queues (128 regular +16 special). New chapters added: ▪ Chapter 9: Buffer Management Unit (BMU) ▪ Chapter 14: Cryptographic Features ▪ Chapter 17: PCI Express Controller and Appendix B: PCI Express Registers. ▪ Chapter 18: Ethernet Fabric Adapter. ▪ Chapter 19: Congestion Management and Flow Control. General updates to chapters 1, 2, 7, 8 and 15.
1.2	Aug. 21, 2013	Number of pins changed to 2896. Chapter 1 Product Overview: general updates; power consumption is 80W typical; EZdk Software Development Kit improved; GigaChip Interfaces (GCI) added for Bandwidth Engine memory; Security and encryption/decryption standards updated. Chapter 2 Physical Description: SerDes lane 49 and RXAUI host ports added. Chapter 11 Lookup Acceleration: expanded and improved.
1.0	Feb. 21, 2013	Initial release - preliminary draft.

Contents

List of Figures

List of Tables

1. Product Overview

EZchip's NPS-400 is a game changing network processor that merges NPU performance with CPU flexibility and ease of programmability. The NPS-400 C-programmable packet processor at ultra-high 400-Gigabit throughput, supports all 7-layers with integrated traffic management to enable the next wave of high-performance intelligent carrier routers, data-center network equipment, and accelerates virtualized functions in emerging SDN and NFV networks. Listed below are highlights of the features offered by the NPS-400.

1.1 Feature List

1.1.1 Highlights

- Revolutionary – C programmable 400Gbps Network Processor for Smart Networks
- Technology – based on 10 times smaller and faster C programmable Task Optimized Processors (CTOPs)
- Integration – NPU with 256 CTOPs, 4K threads, Traffic Manager, DPI, security, search engine and 960Gbps I/O in one chip
- Performance – 400-Gigabit wire-speed packet processing
- Flexibility – advanced services and full layers 2-7 stateful processing
- Simplicity – easy to use C programmable and Linux® operating system
- Versatility – addressing carrier, cloud and data-center networking equipment
- Applications – switching, routing, SDN/ OpenFlow and virtualization of network elements, load balancing, DPI, TCP offload, firewall, VPNs, intrusion detection/prevention, network analytics and more

1.1.2 Featuring

- 600Mpps wire speed, application dependent
- 256 innovative CTOP processors, 16 threads each, for a total of 4K virtual processors, in an array with dedicated HW accelerators
- Symmetric multi-processing (SMP) Linux optimized for data plane applications
- Hardware-based traffic manager (TM) up to 600Mpps, 1 million queues and 5-level hierarchical scheduler
- Up to 200Gbps security HW acceleration
- Up to 200Gbps DPI
- High-touch value L4-L7 services can be performed in-line on the line card in addition to the classic L2/L3 processing
- Lookup acceleration for high-speed algorithmic searches
- Integrated TCAM with algorithmic extension to external DRAM
- Autonomous CPU operation capability
- 10/40/100GbE, Interlaken, Interlaken-LA, GCI, and PCIe Gen 1/2/3 ports for network/ fabric interfaces
- 96 SerDes lanes supporting 2.5/5/6.25/8/ 10.3125/12.5Gbps
- 4 SerDes lanes (2 lanes on each side):
  - Each lane supporting 10.3125 Gbps
  - One lane per side configurable to 1.25Gbps for debug LAN
  - One lane per side configurable to 1x PCIe supporting 2.5/5/8Gbps

1.1.3 Target Applications

- Carrier Ethernet switch router line cards and services cards
- Data center and carrier appliances
- Mobile packet network infrastructure – evolved packet core (EPC) routers
- Software defined networking (SDN), OpenFlow and virtual networking
- Load balancing
- Firewall and intrusion detection and prevention systems (IDS/IPS)
- IPsec and SSL VPN gateways
- TCP offload
- Traffic analytics and network monitoring
- WAN optimization
- DPI and lawful interception
- Session border controllers

1.1.4 Detailed Feature List

Interface Sides (see Figure 2-2 & Table 2-1)

- Two identical interface sides:
  - 2x 50 SerDes lanes (48x Network IFs and 2x Management ports)
  - Flexible lane assignment to 10/40/100GbE MACs, Interlaken MAC, Interlaken-LA, GCI or PCIe
- Network port options (up to)
  - 2x 48x 10GbE XFI/SFI/10GBASE-KR
  - 2x 12x 40GbE XLAUI/40GBASE-KR4
  - 2x 4x 100GbE CAUI
  - 2x 48-lane Interlaken
  - 2x 24-lane Interlaken-LA for external TCAM knowledge-based processor at 6.25/10.3125/12.5Gbps
  - 2x 4- or 8-lane GCI at 12.5Gbps
  - 2x 8-lane PCIe Gen 1/2/3 at 2.5/5/8Gbps
    - Endpoint functionality with DMA capability into shared memory space
- 400GE interface support via Interlaken interface
- SerDes properties
  - Speed selection per pair of lanes
  - Supporting 1.25[1]/2.5/5/6.25/8/10.3125/12.5Gbps
  - 10GBASE-KR/40GBASE-KR4 for 10/40GbE backplane applications
  - Auto-negotiation for backplane Ethernet (IEEE 802.3 Clause 73)
  - Receive lane order swap according to IEEE 802.3ba Clause 82
  - Hot swap support for interface modules and line cards
  - Forward error correction (FEC)
  - Frame sizes from 1B to 12KB supported
- Data center bridging (DCB) support:
  - 802.1Qaz enhanced transmission selection (ETS) by TM abilities
  - 802.1Qbb priority flow control (PFC)
  - 802.1Qau quantized congestion notification (QCN) and congestion notification (CN) flow control with lossless operation

[1] 1.25Gbps is supported on debug ports only; not on network interfaces.

Management Ports

- 1-lane PCIe Gen1/2/3 at 2.5/5/8Gbps
  - Available after reset
  - Serves as endpoint
  - External host configuration of the NPS
- 2x 1GbE SGMII/1000BASE-X debug ports

Memory Interfaces

- Support for DDR3 (including low voltage) or DDR4 memory devices
  - 12x DDR3/DDR4 32b interfaces (24 DRAM controllers)
  - Up to 96GB using 24x16-bit or 48x8-bit DDR3/DDR4
  - 1066MHz DDR3 or 1333MHz DDR4 clock rate
  - Up to 3.2 billion accesses per second for DDR3 and 3.9 billion for DDR4 (16 bit)
  - 800Gbps bandwidth for DDR3 and 1Tbps for DDR4
  - Unified memory management: Frame memory, TM control memory, search (lookup) memory, policers, statistics counters, and CTOPs
  - Per data structure configurable dedicated banks option
  - Lookup structures auto replication
  - Supports clam-shell board topology
- Supports GigaChip™ interfaces
  - Allocate up to 8 SerDes for GCI per side
  - Supports a x4/x8 lane GCI host interface per side to Bandwidth Engine-3 capable memory device
  - BE-3 compatible transaction layer supporting efficient short transactions, atomic operations and statistics, traffic management policing and user defined subroutines
  - Supports 12.5Gbps per lane
  - Two scheduling domains per GCI interface
  - Up to 0.8 billion/sec. 16B reads per x8 GCI interface, or 1.6 billion/sec. 16B reads over two interfaces
  - 90% BW efficiency for short transactions
  - Serves search/control-structures and metering/statistics policers/counters
  - Search structures DRAM replica auto-access when GCI is congested
  - Support duplications of search structures between GCI interfaces and within the same GCI interface

- Internal unified 16MB SRAM memory
  - Flexibly allocated to frame data, search structures, packet context, statistics and TM queuing control
  - 256 banks, non-blocking access
  - Up to 256B accesses per second
- Error-correcting code (ECC) protected internal and external memories
  - In-band ECC for external memory, saving pins and memory devices

Internal TCAM

- 2x 1.25Mb TCAM
- 10/20/30/40/50/60/70/80B key lookups
- 64 tables, 8 global mask registers
- High speed lookups, one per clock, with 4 results per lookup
- Pipelined operation to reduce power. Lookups arranged in a pipeline (i.e. lookup the first bank then the second, third, etc.). Stop pipeline after first match.
- TCAM algorithmic emulation increasing effective TCAM size using external memory. Ideal for applications requiring very large TCAM tables and classification lookups per session instead of per packet, such as OpenFlow.

External TCAM Knowledge-based Processor

- Interlaken-LA interface to connect to off-the-shelf external TCAM/ knowledge-based processor
- Interlaken-LA master mode:
  - Up to 80B key lookups, up to 4 results per key
  - Dynamic TCAM context allocation
  - Data cache

CTOPs

- CTOPs (C-programmable Task Optimized Processors) are ARC®-based processors running at 1GHz and augmented with EZ-ISA (EZchip Instruction Set Architecture), optimized for packet processing
- EZnet, cross chip fabric, connects an array of 16 NPCs (Network Processor Clusters), each one is an array of 16 CTOPs, providing a total of 256 CTOPs
- 7-stage scalar, fully interlocked instruction pipeline resolving data hazards in HW
- Multi-threading: 4K HW threads (16 threads per CTOP), HW and SW scheduled
- Dynamic branch prediction
- Linux OS support
  - User and kernel modes
  - Precise exceptions
  - MMU (Memory Management Unit)
- Secure processing
  - Limited privileges for user tasks
  - Exception on illegal instruction
- Multi-processing support
  - Atomic operations, such as exchange and cmpxchg
  - EZ-ISA atomic operations, e.g. bit manipulation, counting semaphores
  - Fast messaging in user mode
  - Shared and private memory regions
- Memory per CTOP
  - Code and data space only limited by external DRAM size
  - 64b data bus width
  - 32b virtual address and 44b physical address
  - Total of 16KB SRAM partitioned between threads
    - Size of private D-Cache and CMEM (core local memory) are configurable per thread
    - Configurable shared CMEM size
  - 8KB instruction cache, 2-way set associative
  - Single error correction double error detection (SECDED) ECC protection
  - 32 general purpose registers
  - Misaligned memory accesses
  - Data plane processing dedicated memory space FMT (Fixed Mapping Table)
- Configurable interrupt controller
  - 8 IRQs
  - Two level interrupts
- ARCompact™ ISA
  - Native support for 16b and 32b instructions for high code density
  - Zero overhead switching between 16b and 32b
  - Floating point single precision acceleration
- EZ-ISA packet processing extensions
  - 16/32/48/64b instructions
  - Built-in packet processing ALU to accelerate bit manipulation

  - Embedded acceleration for case/switch control instructions
  - Bit granularity for arithmetic and logic operations
  - Hardware network protocol decoder
  - BCAM, TCAM
  - Lookup instructions
- DPI extensions
  - EZ-ISA instruction with HW acceleration for DPI algorithms
- Supports both run to completion and pipelined programming models
- Supports Linux in both control and data plane
- Support for third party Linux applications
- Autonomous CPU operation capability
  - One or more of the CTOPs' HW threads can replace an external host CPU
  - Boots from external flash
  - Suitable for light control plane processing
  - Optional external host for demanding control applications

Integrated Traffic Management

- Two traffic managers up to 480Gbps each, one for each interface side
  - 600Mpps WRED and peak bandwidth scheduling
  - Flexible connection to any physical interface port
- Combines both ingress and egress functionality
- External SDRAM packet buffering
  - Up to 64GB addressable memory space; up to 256M frames
  - Frame sizes from 1B to 12KB
- Per flow queuing with 5-level hierarchical scheduling:
  - 2x 0.5M or 1x1M TM flows
  - 2x 32K classes/users
  - 2x 4K subports
  - 2x 0.5K ports
  - 2x 128 interfaces
- Dynamic hierarchy mapping and hitless resource allocation and reconfiguration
- Configurable policing/dropping schemes between all WRED hierarchy levels
  - Configurable WRED profiles
  - Buffer reservation for guaranteed memory resources per entity
  - Per flow per color WRED statistics
- Policing: Per-flow metering, marking and policing for millions of flows
- Shaping: Single and dual token bucket on committed/peak rate/bursts (CIR, CBS, PIR, PBS), with inter frame gap (IFG) emulation for accurate rate control
- Scheduling: WFQ and priority scheduling at each hierarchical level
  - Eight priority levels for queues
  - Work conserving and non-work conserving schedulers
  - Color aware weights
- Per frame timestamp and timeout drop
- Hardware and software flow control of all hierarchical entities
- Flexible and dynamic mapping of TM hierarchies onto the physical underlying ports and interfaces
- 8 strict priority propagation thru the TM hierarchy with optional priority remapping
- Per CoS scheduling
- 400GE interface support
- Per packet physical transmit interface assignment
  - Decoupling of TM scheduling port entity from physical interface
  - Resilient network engineering by fast re-route on path failure
  - Link aggregation group (LAG) shaping
- Flow control scheme based on TM source and destination resources congestion
- Class-based flow control
- Based on EZchip's field-proven NP-5 TM

Statistics and Counters

- Per-flow statistics for programmable events, traffic metering, policing and shaping
- Counters can reside in 16MB unified internal memory, dedicated 128KB internal memory, or external SDRAM
- On-chip memory containing 512K policers status for cache acceleration
- 64-bit counters in both internal and external memory with on-chip caching
- Dynamic allocation, recycle and auto association between counters and flows. Counters are automatically recycled when a flow is deleted or aged
- Programmable threshold settings and threshold exceeded notification

- On-demand and posted counters:
  - 512M on-demand counters (updated and collected by the application); 256M of these are allocated for EMEM buffer multicast counters
  - 256M posted counters (updated by application and collected by the control plane application)
- Up to 6 billion statistic operations per second, peaking at 4 billion per second for each of on-demand and 2 billion posted statistics operation per second
- Auto implementation of token bucket policers per flow (srTCM, trTCM or MEF5)
  - Coupled policers scheme

Security and Encryption/Decryption

- Network processor cluster (NPC) natively supports crypto instruction acceleration
- 200Gbps IPsec support (100 Gbps full duplex both encryption & decryption simultaneously)
- Supports NIST Suite B ciphers
- AES 128/192/256 encryption/decryption, ECB/CBC/CFB/OFB/CTR/CCM/GCM chaining modes. AES-CMAC/AES-GMAC
- DES/3DES 56/112/168 ECB/CBC/CFB/OFB/ CTR chaining modes
- MD-5, SHA-1, SHA2-224/256, SHA2-384/512 hashing/authentication
- Relevant only for NPS-400 models that support cryptographic features (order no. 207850xx)

Deep Packet Inspection

- Up to 800Gbps DPI-based application recognition
- Up to 200Gbps DPI content filtering, with at least 1 Gbps single-thread DPI filtering per CTOP
- Assignment of processing cycles to DPI is configurable on a per CTOP HW thread level. A single CTOP can simultaneously process DPI and run other packet processing or control plane code on different HW threads.
- Stateful cross-packet inspection, i.e. signature matches start and end points can be in different packets
- Easily-customized, SW-based DPI implementation, augmented with ISA extensions and HW accelerators
- Application recognition solution:
  - Recognizes up to 1500 different applications organized into categories
  - Signatures are *compiled* on an external CPU and loaded into the NPS
  - Identification signatures based on multiple technologies (e.g. RegExp, Behavioral)
  - Full application recognition performed in NPS
  - Early bypass of DPI engine following classification
  - DPI application ID stored in the Stateful Flow Table (SFT)
- Bloom filter-based content filtering solution:
  - Parallel matching of hundreds of thousands Perl compatible regular expression (PCRE) rules
  - EZregex, EZchip PCRE compiler, splits PCRE matching between NPS and corresponding external management CPU
  - External management CPU selective PCRE matching triggered by NPS sub-string matches, using pre-compiled suitable deterministic/nondeterministic finite automaton (DFA/NFA)
  - External management CPU SW can be executed on control-plane CTOPs or on external CPU
  - NPS offloads external CPU by filtering incoming wire-speed traffic, so that only a few percent of the traffic needs inspection by external CPU
  - On-chip zero false-negative and controlled false-positive scanning of strings and fixed length patterns anywhere in packet payload, including fragmented packets
  - Search sub-string can be anywhere within the PCRE (optionally case sensitive)
- NPS character-set and sub-string matching using Bloom filter, saving memory space and execution time
- On-the-fly detection of UTF8 (non-ASCII) encoding
- Supports in-service software update (ISSU) including the PCRE database
- Stateful flow awareness as an infrastructure for the DPI and other clients
  - Flow is defined as a bi-directional 6 tuple entity
  - Includes support of tunnel stripping, NAT awareness, IPv4 and IPv6
- EZdpi, DPI application SW package, supplied by EZchip including:
  - NPS-based application recognition solution

  - Content filtering solution for both NPS and management CPU

Synchronization

- On-chip IEEE 1588v2 1-step and 2-step hardware time stamping
- Support for synchronous Ethernet ITU-T G.8261
  - Selectable receive clock recovery per side
  - Selectable transmit reference clock

OAM Processing

- KeepAlive frame generation for precise and accurate session maintenance operations
- KeepAlive watchdog timers for fast detection of loss of connectivity
- IEEE 802.1ag/ITU-T Y.1731 compliant message generation/termination offload
- Per OAM session state tracking and reporting
- Flexible statistics and performance monitoring

Power Management

- Per memory and data interface power-up/ power-down
- Automatic voltage scaling (AVS) to minimize the average power consumption
- Temperature monitoring for dynamic selection of power profiles allowing relaxed temperature and power design constraints
- On-chip thermal diode for temperature monitoring
- Configurable number of active CTOPs cores for best power optimization per application

High Reliability

- Comprehensive on-chip diagnostic hardware support
  - Trace and debug
  - Built-in performance counters for system monitoring
  - ECC SECDED on internal and external memory

Software Development

- EZdk is a comprehensive set of design and testing software tools for C-programming of CTOPs for data plane and control plane
- EZ-ISA instruction set provides C programmed packet processing
- SMP Linux kernel 3.x supporting control and data plane, including boot loader
- Eclipse™-based integrated development environment (IDE) enabling simultaneous multi-core debugging
- Industry-standard GNU toolchain
- Instruction accurate simulator for rapid prototyping of applications
- C-language applications including a sample reference L2-L3 switching and routing package, security and regex package

Physical Specifications

- Package: HFCBGA, 2892 pins, 55x55 mm, 1mm ball pitch, 0.64mm ball size
- Process: TSMC 28nm
- Core power supply voltage: 1.0V
- Power consumption: 100W typical (estimated)

Ordering Information

NPS MODEL	*ORDER NO.*
NPS-400-RoHS with crypto features	207850xx
NPS-400-RoHS without crypto	207857xx

1.2 Product Description

EZchip's NPS (Network Processor for Smart Networks) is a revolutionary network processor unit (NPU) that enables the next generation of smart high-performance carrier and data-center equipment. Through its breakthrough architecture the NPS boosts the flexibility and performance of traditional NPUs and CPUs to enable advanced baseline features for carrier equipment and to scale the performance levels of data-center equipment. NPS provides equipment vendors with a feature-proof, fully programmable, high performance solution.

The demand for network bandwidth and intelligence produces an insatiable appetite for processing power in networking equipment. Network equipment vendors essentially have had a choice of two types of processors: NPUs that provide fast layer 2-3 processing however with complex microcode programming, and CPUs (single or multi-core) that provide flexible C-programmable layer 4-7 processing however at reduced performance and power efficiency. Through its breakthrough architecture, NPS removes the barriers imposed by traditional NPUs and CPUs. It enables extremely high-performance, C-programmable, layer 2-7 processing, and provides networking vendors with an architecture that scales as more advanced services at higher speeds are required over time.

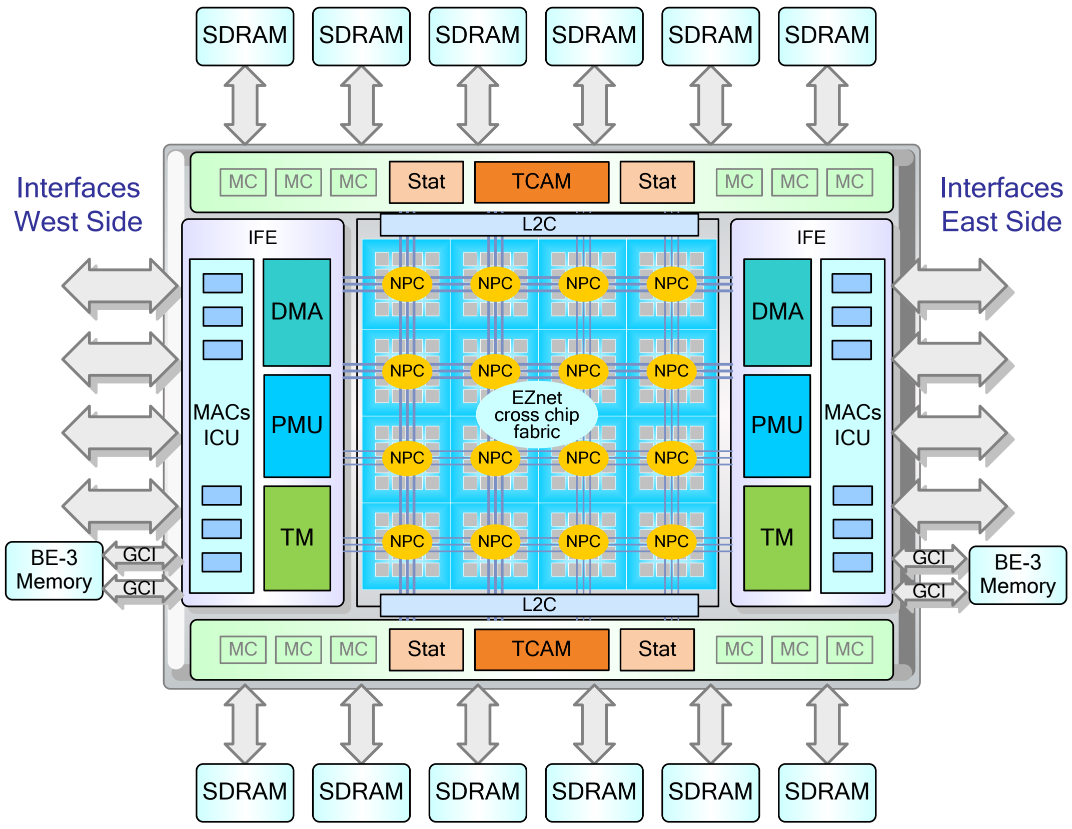
For carrier equipment, the NPS enables the migration of advanced layer 4-7 features from specialized services cards to common line cards and enables the line card with new baseline features such as application recognition and IPSec VPNs, as well as greater velocity for adding new features. For data-center equipment, the NPS allows scaling their performance to the required loads of the evolving data center for greater layer 2-7 capabilities within constrained power and space. It empowers data-center appliances as well as carrier appliances with line-rate performance for applications such as load balancing, firewall and OpenFlow/SDN (Software Defined Networks) and network virtualization.

The NPS is uniquely designed for data-plane processing where packet processing and wire-speed forwarding prevail, unlike CPUs that are designed for general processing where management functions and control-plane processing prevail. Central to the NPS are its innovative CTOPs (C-programmable Task Optimized Processors). These processing engines build on EZchip's extensive NPU experience and are designed specifically for data-plane processing. The optimized design allows the integration of 256 such processors, each with 16 threads, for a total of 4K virtual processing engines. This vast number of engines is mandatory for high-speed data-plane processing where packets are arriving at an extremely high rate and every packet is processed. The NPS architecture is in contrast to CPUs that have an order of magnitude fewer engines integrated into a chip because they are based on much larger general-purpose cores, have large memory caches, designed to execute millions lines of code, and target both data and control processing. The NPS' efficient design incorporates into the chip not only a great number of cores but also EZchip's market-proven hardware traffic manager, a task optimized memory architecture, and numerous hardware accelerators for efficient table lookups, application recognition and security. The end result is a very powerful network processor with superior power efficiency and integration.

The NPS provides great packet processing simplicity and flexibility through C-based programming, a standard toolset, support of the Linux® operating system, large code space, and a run-to-completion or pipeline programming style. A comprehensive library provides source code for a variety of applications to speed customer's design cycle. The NPS features cores that are highly optimized for packet processing and leverage EZchip's vast packet processing and applications experience, a market-proven traffic manager, hardware accelerators for security and DPI (Deep Packet Inspection) tailored for efficiency and performance, on-chip search engines including TCAM with scaling through algorithmic extension to external low-cost, low-power, DRAM memory, GigaChip interfaces to Bandwidth Engine memory devices, a fabric adaptor to enable direct connection to a chassis backplane and switch fabric, and a multitude of interfaces providing an aggregated bandwidth of 960-Gigabits per second including 10-, 40- and 100-Gigabit Ethernet, Interlaken and PCI Express interfaces.

1.2.1 Main Blocks

Figure 1-1. NPS-400 high level block diagram



NPS-400 includes the following main blocks (see Figure 1-1):

IFEs (Interface Engines) integrate all the I/O interfaces, MACs, input classification units (ICU), DMAs, TMs and Processor Management Units (PMU). Each IFE serves 12 SerDes lanes and there are four IFEs on each device side.

ICUs (Input Classification Units) classify the packet into a profile which sets the behavior of the packet in terms of CoS, buffer management (internal or external memory), thresholds and early drops.

DMAs (Direct Memory Access) are responsible for putting/taking the packet in/out of the memory. The DMA creates a job descriptor which is queued into the PMU.

PMUs (Processor Management Units) are responsible for handling the queues of jobs to be given to the array of CTOPs.

TMs (Traffic Managers) are 5-level hierarchical QoS TMs with a total of 1M physical queues that provide QoS for individual users and applications, crucial for enabling services over Ethernet networks.

NPCs (Network Processing Clusters) consist of an array of 16 CTOPs with local memory, L2 cache for look-up data and a bus interface unit to the EZnet and the PMU. Each CTOP has 16 HW threads. Sixteen NPCs are organized in an array, for a total of 256 CTOPs and 4096 HW threads.

EZnet is a highly scalable cross chip fabric to connect CTOPs, TMs, memories and interface units. EZnet allows the NPS-400 to scale to hundreds of CPU cores, with close to linear performance scaling.

L2C (Level 2 Cache) manages EZnet external memory traffic and optimizes the DDR3/DDR4 SDRAM performance. It selectively stores high value data for improved access latency, or controls transactions of non-cached data to external memory. The distributed L2 cache system aggregates 1MB of on chip cache for data or code and balances external memory traffic loads among multiple SDRAM memory controllers. Transactions are optimized to get efficient SDRAM memory utilization. L2C also manages load balancing between DRAMs on duplicated search structures.

Stat statistics acceleration block offloads the CTOPs for statistics and token bucket operations.

MC (Memory Controller) is a DDR memory controller accessed through L2 cache with or without caching and optimization for memory bandwidth utilization.

TCAM is an integrated TCAM.

GCI is a GigaChip serial interface to a Bandwidth Engine memory device.

1.2.2 CTOPs

The NPS provides C-programming capabilities and a standard Linux OS for both the data plane and the control plane. CTOPs offer the benefit of a C-based programming environment, while retaining all the benefits of key TOP-specialized instructions, which are accessed thru in-line functions. Unlimited code space for control plane and debugging features, a Linux environment that offers access to all the standard toolchain, debugger and other features familiar to programmers.

The NPC architecture is built on an array of symmetrical processors, CTOPs, which run software in a run to completion model. The core hardware multi-threading conceals the latency of memory accesses.

EZ-ISA instruction set, which includes legacy TOP instructions, provides a smooth transition from the TOP to the CTOP architecture.

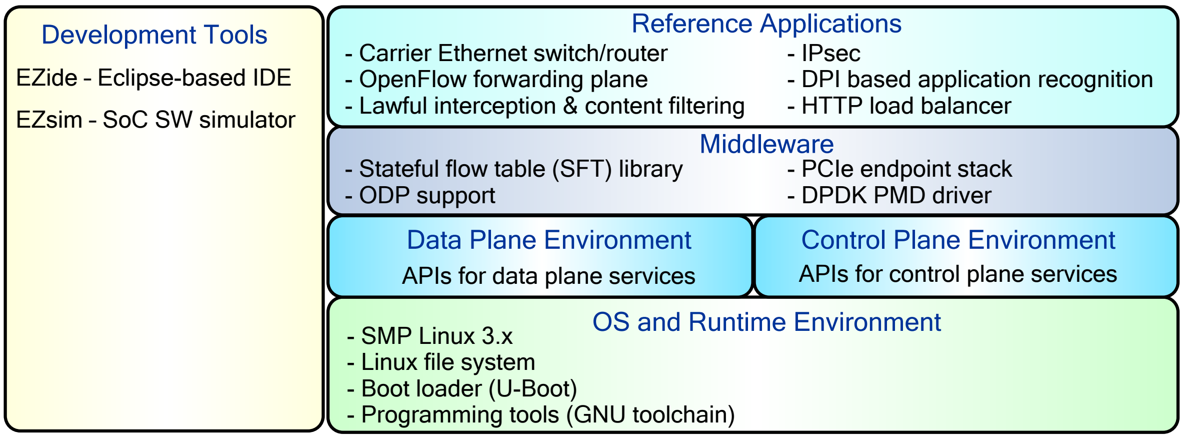
1.3 NPS Software Architecture

The NPS provides great packet processing simplicity and flexibility through C-based programming, a standard toolset, support of the Linux® operating system, large code space, and a run-to-completion or pipeline programming style. A comprehensive library provides source code for a variety of applications to speed customer's design cycle.

1.3.1 EZdk Software Development Kit

EZchip's Software Development Kit (EZdk) is a comprehensive set of design and runtime tools for developers, enabling a short time-to-market of new designs based on EZchip's C-programmable NPS. EZdk allows developers to design, implement and verify both control-plane and data-plane applications for the NPS. EZdk provides an Eclipse™-based IDE (Integrated Development Environment), optimized GNU toolchain and Linux kernel, control and data plane software development kits, reference applications, and software simulator. See Figure 1-2.

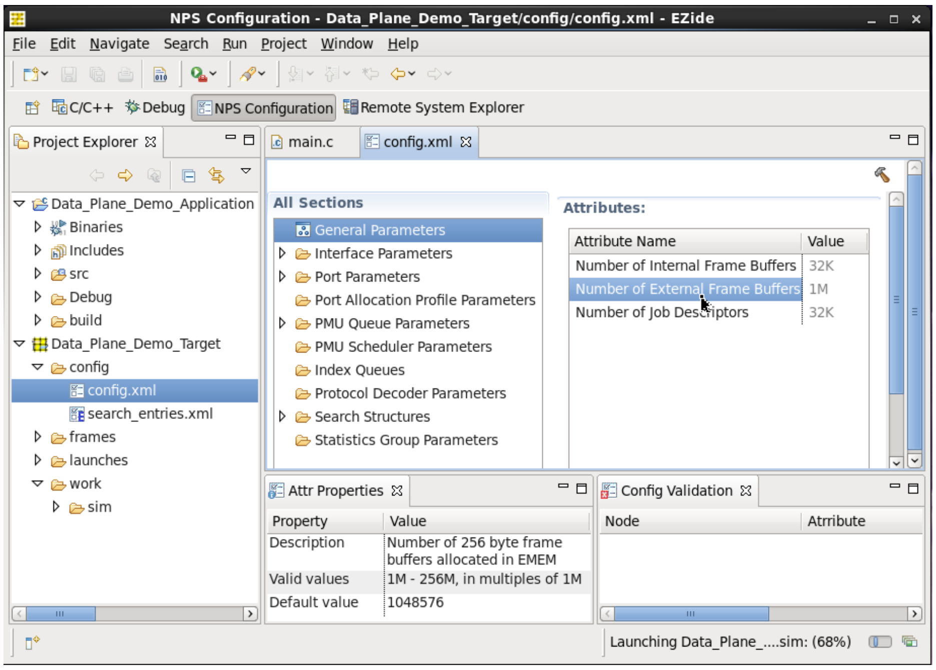
Figure 1-2. EZdk software development kit


Development Tools
EZide - Eclipse-based IDE
EZsim - SoC SW simulator
Reference Applications
- Carrier Ethernet switch/router
- OpenFlow forwarding plane
- Lawful interception & content filtering
- IPsec
- DPI based application recognition
- HTTP load balancer
Middleware
- Stateful flow table (SFT) library
- ODP support
- PCIe endpoint stack
- DPDK PMD driver
Data Plane Environment
APIs for data plane services
Control Plane Environment
APIs for control plane services
OS and Runtime Environment
- SMP Linux 3.x
- Linux file system
- Boot loader (U-Boot)
- Programming tools (GNU toolchain)

1.3.2 Development Tools

- **EZide** – Eclipse™-based Integrated Development Environment
  - Industry standard-based symbolic software debugging and profiling tools (GDB, perf) with multi-core support
  - System on a chip (SoC) level performance profiling and hot-spot analysis tools, allowing detection of system-wide bottlenecks (DDR, etc.)
  - Hardware level debugging via JTAG interface
  - GUI-based editor for generating NPS device configurations
  - GUI-based interface for exploring and controlling NPS device status
  - Enables operation with the real hardware targets and the software simulation tools
  - Supported on x86-64 development stations running Linux
- **EZsim** – Instruction accurate simulation of the CTOPs and the NPS SoC supporting the running of full multi-core data plane and control plane applications, including SMP (Symmetric Multi-processing) Linux.

Figure 1-3.Sample screen from EZide development environment



Attribute Name	Value
Number of Internal Frame Buffers	32K
Number of External Frame Buffers	1M
Number of Job Descriptors	32K

Property	Value
Description	Number of 256 byte frame buffers allocated in EMEM
Valid values	1M - 256M, in multiples of 1M
Default value	1048576

- **EZspy** – Data plane troubleshooting tool that assists in identifying obstructions in the data path by collecting and analyzing NPU status data and resource utilization.

Operating System

- SMP Linux 3.x kernel ported to NPS architecture to run on the CTOPs
- Zero Overhead Linux™ (ZOL) supporting bare-metal performance with the same familiar and convenient Linux user-space development environment
- Linux file-system and standard runtime libraries
- Device drivers for the NPS hardware peripherals, such as the Ethernet network interfaces and the UART interface for console services
- Boot loader (U-Boot)
- Industry standard GNU toolchain

Data Plane Environment Libraries

- Data-plane services libraries providing a C-based application programming interface (API) for data-plane applications running on the NPS, abstracting the complexities of the underlying CTOP core instruction set and various hardware accelerators.

Control Plane Environment Libraries

- Control-plane services libraries providing a C-based API for control-plane applications for the NPS, abstracting the complexities of the underlying hardware interface. The control-plane service libraries are used for configuration and management of the NPS, including its network and memory interfaces, embedded traffic manager and statistics acceleration block, as well as for creation and management of lookup structures.
- Access control list compiler for EZchip's algorithmic TCAM accelerators
- PCRE (Perl Compatible Regular Expression) rule compiler for EZchip's DPI (Deep Packet Inspection) accelerators
- The control plane environment libraries may be run on an external control CPU or on one or more of the NPS HW threads implementing the control CPU functionality, for fully autonomous operation without requiring an external control CPU.

Software Libraries

EZchip provides a rich set of software middleware and reference applications. Middleware is comprised of reusable building blocks, modules or stacks that build on the basic data and control plane API to facilitate the fast and simple creation of applications.

- **Stateful Flow Table (SFT)** provides bi-directional stateful flow awareness at wire speed in a client agnostic implementation. Among the clients using the SFT are the DPI-based Application Recognition, Lawful Interception and HTTP Load Balancer solutions (described below).
- **OpenDataPlane (ODP)** is a cross-platform open-source initiative. EZchip is a contributing member of ODP driving the SDK definition to support hardware accelerated high end NPUs.
- **PCIe Endpoint Stack** provides a framework for efficient communication of the NPS data plane with host-based control or data plane applications over the PCIe interface.
- **DPDK PMD Driver**: The virtual Ethernet port switch/aggregator builds upon the PCIe Endpoint Stack to implement a fully featured PCIe smart NIC engine.

Reference Applications

EZchip provides reference applications to assist developers in their designs.

- **Carrier Ethernet Switch/Router** reference application is a full featured carrier Ethernet switch router implemented using the NPS data plane APIs.
- **OpenFlow Forwarding Plane** reference application provides an OpenFlow™ forwarding plane.
- **Lawful Interception and Content Filtering** reference application provides hardware-accelerated, flow-based, cross-packet, Bloom filter-based content filtering. A PCRE compiler identifies key sub strings which are loaded into the Bloom filter. Suspected signatures are then inspected, and residual inspection is offloaded to a host.
- **IPsec** reference application demonstrates the SW and HW abilities of the NPS-400 that enable high-speed and high-performance for IPsec traffic..
- **DPI-based Application Recognition** reference application provides up to 800 Gbps application recognition. A DPI engine is loaded with up to 1500 compiled signatures. Packets are processed by the engine after they have been mapped to flows by the SFT. Application IDs for the classified flows are stored in the SFT. The DPI engine is bypassed for all packets belonging to classified flows.
- **HTTP Load Balancer** provides a reference implementation for a layer 7 content-aware HTTP load balancer. The application transparently intercepts HTTP requests to server IPs, answers as a proxy for the server, and based on the request of the client transparently redirects the connection to an available web server out of a pool to service it. The solution utilizes the SFT and DPI AR's HTTP parser as its infrastructure.

For more information refer to the *NPS Software Libraries Product Brief.*

1.3.3 Zero Overhead Linux™ (ZOL) Environment

EZchip provides an integrated SMP Linux runtime environment for both control and data plane applications running on the NPS CTOP cores.

To achieve the performance requirements for data plane applications, EZchip's optimized Linux kernel provides deterministic performance for data plane applications on par with a bare metal application, while enjoying the benefits of a standard OS support.

The data plane application processes run on dedicated hardware threads, and are isolated from all uncontrolled interferences. They do not perform context switches unless explicitly requested by the application or the debugger, and are isolated from all interrupt sources, including timer interrupts. The data plane applications access all performance-critical code and data through a fixed memory mapping table, avoiding any costly TLB misses. The data plane applications interface directly with the hardware services through the data plane user-space library, removing the need for context switching between user and kernel space modes.

The data plane applications are developed, controlled and debugged as standard Linux processes. They can be started/killed thru a Linux shell, viewed using the standard "ps" or "top" commands and debugged via the standard GDB GNU debugger. Furthermore, the applications can perform any standard Linux library and system call (e.g., "printf"). The Linux kernel still provides the initialization, exception handling and debug services for the data plane applications, but will not involuntarily interrupt these applications unless there is an exception (e.g. divide by zero) or when requested by a debug process.

1.3.4 Flexible Software Architecture Design

This section provides an overview of the software architecture of the NPS. The NPS provides a highly flexible programming environment enabling a wide variety of SW architecture implementations.

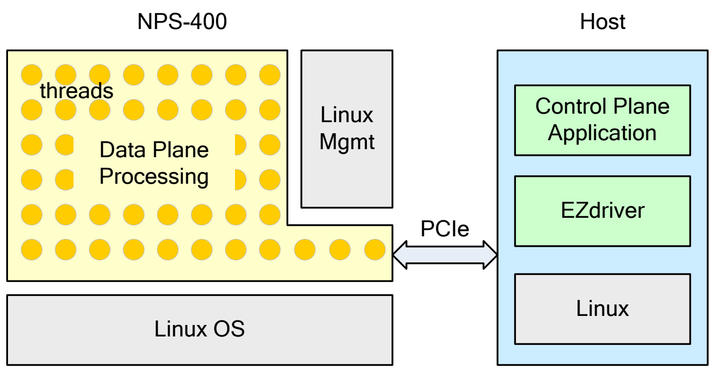
1.3.4.1 Using NPS as a Data-plane Processor with an External Control CPU

In applications that require heavy control plane processing, an external control plane CPU may be required. The control plane CPU communicates with the NPS via the PCIe interface which is used as both a data and control interface.

The external control plane CPU is likely to be an X86 type CPU and will likely run the Linux® operating system and run the EZcp SW library that enables the control plane application to communicate with the data plane running on the NPS.

The NPS runs Linux SMP kernel on 4096 threads. At least one thread is dedicated to running the Linux kernel management while all the other threads can be dedicated to running the data plane processing.

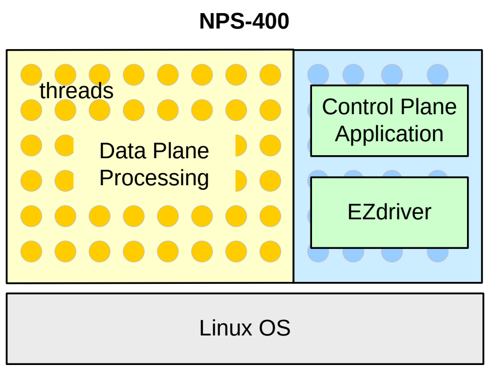
Figure 1-4. Data-plane processing and an external control CPUSW architecture overview



1.3.4.2 Using NPS as a Data-plane Processor with a Local Host

For applications that do not require heavy control plane processing, the option of running combined data plane and control processing inside the NPS can be considered. In this case, a number of threads (typically a few) can be dedicated to control processing while the rest can perform the data plane processing.

Figure 1-5. Combined data plane and control processing

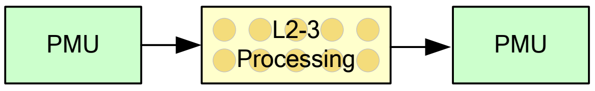


1.3.4.3 Data Plane Programming Models

1.3.4.3.1 Basic Run to Complete Programming Model

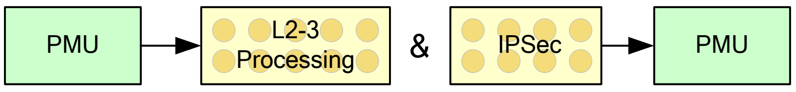
In this model, all the packet processing is done a single pass through the NPS where each thread runs the data plane application from beginning (getting the packet from the PMU) to the end (passing the packet back to the PMU) so that it is scheduled into the Traffic Manager for transmission on the interface. This is the typical programming model for a classic L2/3 packet processing function.

Figure 1-6. Simple run to complete model



This model can be extended to incorporate new features supported by NPS, for example IPSec, where by the complete data processing can be executed in a run to complete programming model including L2/3 and IPSec on each of the HW threads running the SW.

Figure 1-7. Run to complete model

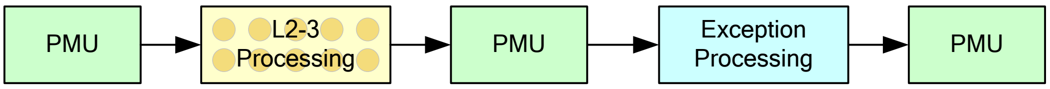


1.3.4.3.2 Combined Run to Complete and Pipeline Programming Models

In some cases, it can beneficial to combine the classic run to complete model with the pipeline models in order to split the packet processing.

For example, the L2/3 processing can be optimized by removing the exception processing which can be dealt with in a second pass. This optimization unblocks packet order delays. In addition, it enables to break the packet ordering so that packets arriving after a packet that requires exception processing (which typically takes longer) do not get delayed.

Figure 1-8. Pipeline programming model



Another option is to break-up the SW processing into stages and to allocate a specific number of HW threads per stage to perform a specific packet processing function. This enables controlling the amount of HW resources (i.e. threads) that can be consumed by a specific packet processing function. For example, in the case of a combined L2/3 and IPSec packet processing function, it might be beneficial to control the amount of resources that are dedicated to performing IPSec encryption/decryption functions.

By using the pipeline model, it is possible to allocate X dedicated threads, for example, to run the L2/3 packet processing and Y dedicated threads for performing the IPSec functions. This segregation can be configured whereby a specific set of threads are associated with a given application, or it can be made more flexible whereby each application can run on all the threads allocated to the data plane processing, but it is possible to prioritize the L2/3 processing over the IPSec application.

Figure 1-9. Segregation

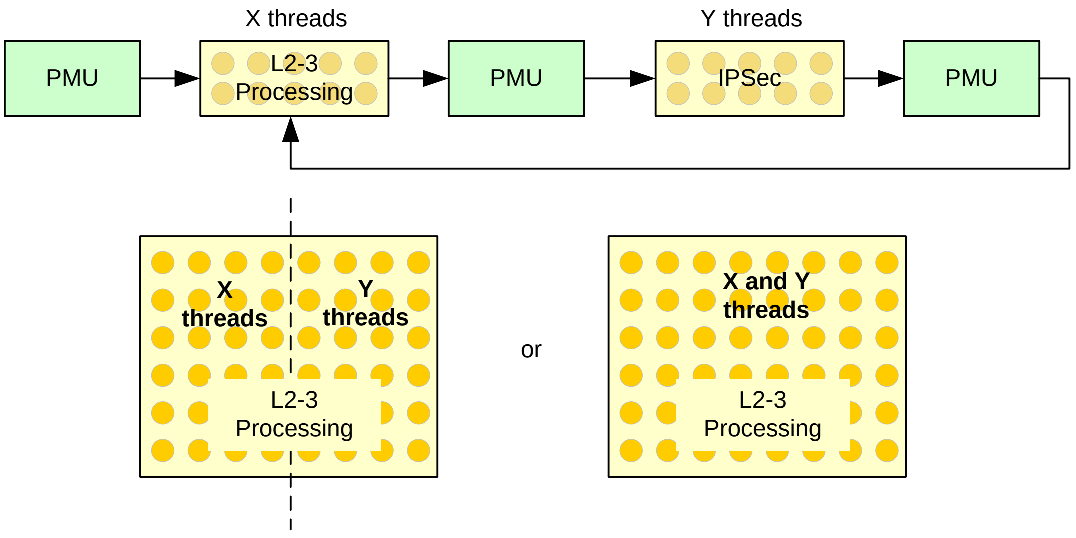
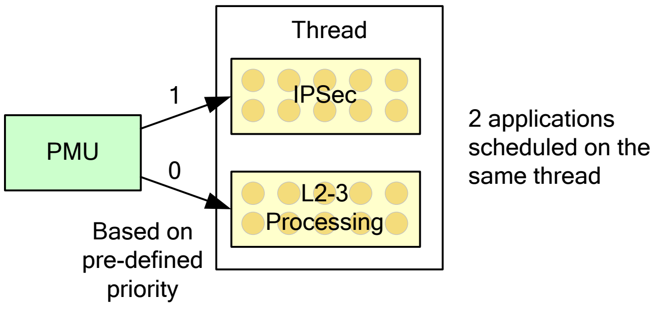


Figure 1-10. Application selection


Thread
IPSec
L2-3 Processing
PMU
1
0
Based on pre-defined priority
2 applications scheduled on the same thread

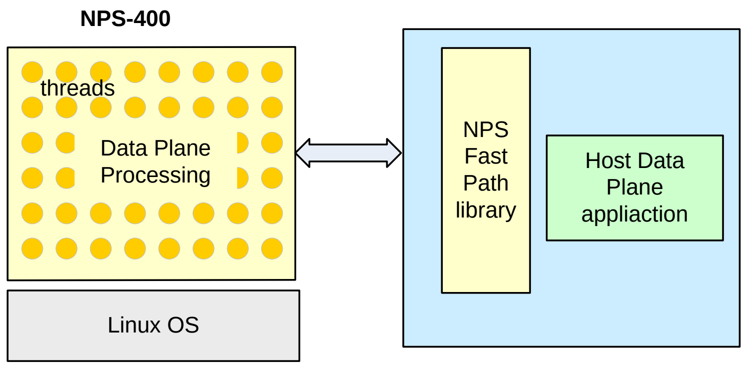
1.3.4.4 Distributed Data Plane

In some cases there may be a need for a distributed data plane where the NPS data plane functions as an offload or fast path acceleration for the data plane service located at a host. One example for this is an application where some of the packets require heavy processing while the rest of the packets can enjoy the NPS data plane's superior performance. Another example for this is an NFV solution where the VNFs are able to offload much of the computation from the NPS data plane.

In these cases, the NPS may be responsible for the forwarding of packet, traffic management and flow and DPI classification. The host-based services would receive the packets with metadata that includes the classification results. These host-based applications would use a host located NPS fast path library to offload flow and packet computation. Such a library may include APIs such as *drop_flow()* and *shape_flow()*.

The distributed data plane would typically involve run-to-completion NPS processing. The driver and control plane location will vary as explained in previous sections.

Figure 1-11. Distributed Data Path (CP not depicted)



1.3.5 Reference Designs

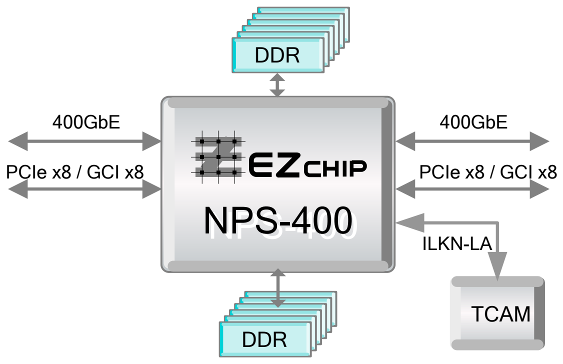
EZchip also offers an NPS Evaluation System, a compact hardware platform, which can be used to accelerate the development of the target hardware and software applications.

2. Physical Description

2.1 Device Interfaces

The NPS-400 processor provides connectivity to the west interface side 0, and east interface side 1, control CPU, external memories and TCAM device. Traffic is flexibly switched/routed between all NPS-400 traffic interfaces.

Figure 2-1. NPS-400 system connectivity



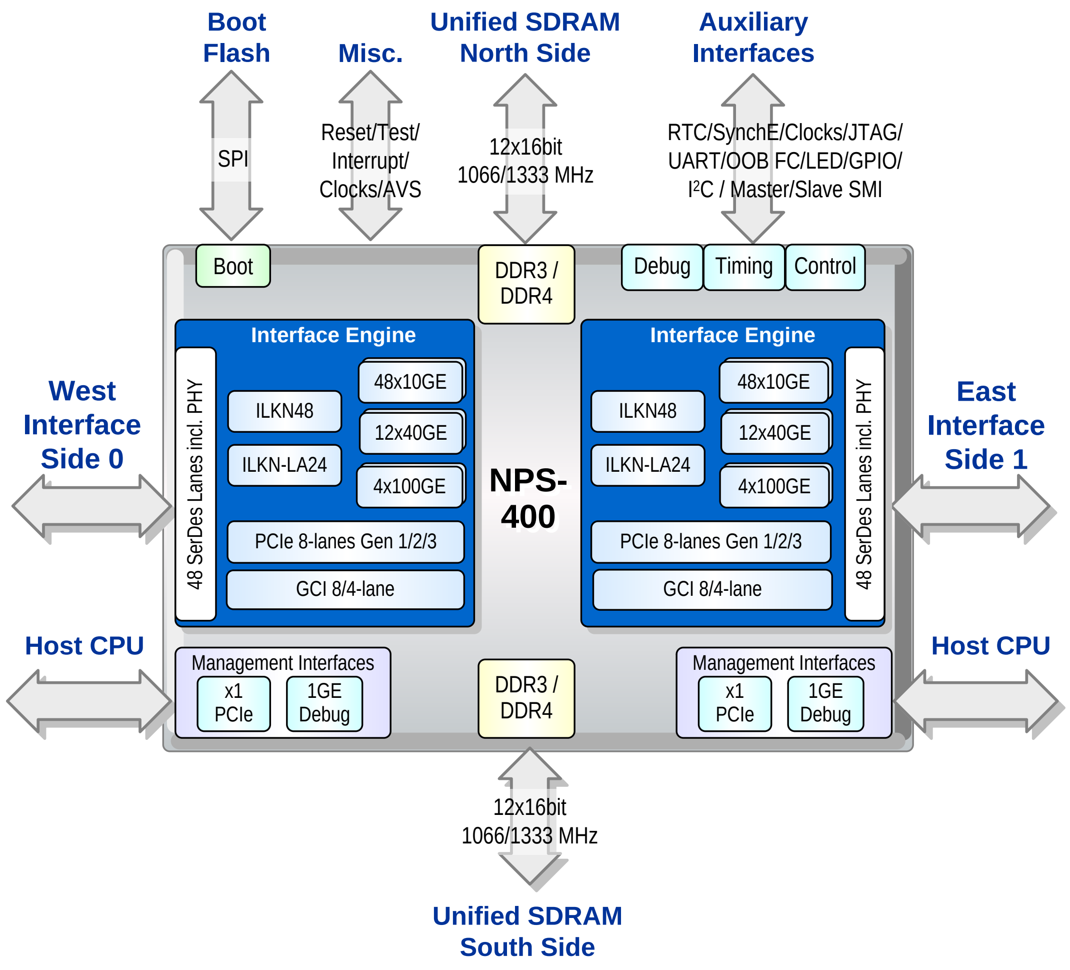
The interface to an external switching fabric enables scaling of the number of system ports and enables interfacing the NPS-400 to other system processors and line cards.

DDR3/DDR4 is used as external memory for search and statistics structures, traffic buffering, TM control memory and embedded CPU memory. All internal and external memory data is ECC protected. An external TCAM interface is optionally used for extending access control lists and knowledge-based processor attachment.

The GCI interfaces to a Bandwidth Engine memory device allowing for high-bandwidth, low-latency memory for search/control-structures and metering/statistics policers/counters. When the GCI interface is congested, the search structure access is automatically redirected to a DRAM replica.

The figure below illustrates the NPS-400 interfaces. The two interface sides are identical but located physically on the west and east side of the chip. This provides flexibility in the design of line cards and can help reduce the number of PCB layers required for routing.

Figure 2-2. NPS-400 interface diagram



Legend:	GCI	GigaChip Interface
	OOB FC	Interlaken Out-of-Band Flow Control
	PCIe	PCI Express
	RTC	Real time clock for IEEE 1588v2
	SMI	Serial Management Interface (MDC/MDIO)
	SynchE	ITU-T G.8262/Y.1362 Synchronous Ethernet clock recovery & reference

2.1.1 SerDes Interfaces

The NPS-400 provides 100 SerDes lanes (up to 12.5Gbps) that can be assigned to variety of Ethernet and Interlaken MACs for network interconnect, host and switch fabric connectivity, packet connectivity and PCIe for CPU sub-system or GCI for Bandwidth Engine memory devices.

- West interface (side 0) has 50 SerDes lanes:
  - 48 lanes for network interfaces (four interface engines of 12 lanes each)
  - Host CPU interface – 1-lane PCIe interface (lane 48) with access to NPS configuration registers
  - Debug port – SGMII/1000BASE-X (lane 49)
- East interface (side 1) has 50 SerDes lanes:
  - 48 lanes for network/fabric interfaces (four interface engines of 12 lanes each)
  - Host CPU interface – 1-lane PCIe interface (lane 48)
  - Debug port – SGMII/1000BASE-X (lane 49)

Interfaces support

- In-band and out-of-band class-based flow control
- Packet-based channelized operation with per channel flow control for ILKN (total of 128 channels per side)
- Support for IEEE 802.1au congestion notification (CN) and 802.1Qbb congestion management
- System-wide traffic management with end-to-end QoS
- Minimum IPG and software-controlled preamble and headers provided to reduced overhead

The NPS-400 interfaces may be configured as shown in Table 2-1 SerDes Allocation Scheme below.

2.1.1.1 Network Interface Testability Features

- IEEE 802.3 Clause 49.2.8 test pattern support for 10GbE
- Test patterns PRBS31, PRBS9, and PRBS7 are supported
- PRBS lock/sync status indication
- Count of PRBS RX errors
- Loopbacks support, all loopbacks are user/software configurable
- PCS loopback TX to RX (not for ILKN-LA)
- SerDes loopback, TX to RX and RX to TX
- On-chip analyzer/scope capability
- On-chip eye plotting test pattern support (IEEE 802.3ba Clause 82)

2.1.1.2 Network Interfaces

Two interface sides (west and east) with 48 SerDes lanes each for Ethernet, Interlaken, Interlaken-LA, PCIe, and GCI ports.

Interface types supported:

- 2x 48x 10GbE XFI/SFI/10GBASE-KR at 10.3125Gbps (assigned per group of 4 SerDes lanes)
- 2x 12x 40GbE XLAUI/40GBASE-KR4 at 10.3125Gbps (assigned per group of 4 SerDes lanes)
- 2x 4x 100GbE CAUI at 10.3125Gbps (assigned per group of 10 SerDes lanes)
- 2x 48-lane Interlaken at 6.25/10.3125/12.5Gbps
- 2x 24-lane Interlaken-LA for external TCAM / knowledge-based processor at 6.25/10.3125/ 12.5Gbps
- 2x 8-lane PCI Express (PCIe) Gen 1/2/3 at 2.5/5/8Gbps
  - Endpoint functionality with DMA capability into shared memory space
- 2x 8- or 4-lane GCI at 12.5Gbps
- 400GE interface external MAC support

2.1.2 SerDes Allocation Scheme

NPS-400 has two interface sides: west and east. The two interface sides are identical, except only the west-side PCIe interface has access to the NPS configuration registers. The table below depicts one of the two identical interface sides.

Table 2-1. NPS-400 SerDes allocation scheme for one interface side

SerDes Lane	10GbE XFI/SFI/ 10GBASE-KR	40GbE XLAUI/XLPPI/ 40GBASE-KR4	100GbE CAUI/CPPI	10GbE XFI/SFI/ 10GBASE-KR	40GbE XLAUI/XLPPI/ 40GBASE-KR4	100GbE CAUI/CPPI	ILKN	ILKN-LA	PCIe	GCI	1GbE Debug Port
	Option A: 12 lane mode			Option B:10-lane mode							
	10.3125 Gbps	10.3125 Gbps	10.3125 Gbps	10.3125 Gbps	10.3125 Gbps	10.3125 Gbps	6.25 / 10.3125 / 12.5 Gbps	6.25 / 10.3125 / 12.5 Gbps	2.5 / 5 / 8 Gbps	12.5 Gbps	1.25 Gbps
0	XFI-0	XLAUI-0	CAUI-0	XFI-0	XLAUI-0	CAUI-0	ILKN-0 1..48 lanes				
1	XFI-1			XFI-1							
2	XFI-2			XFI-2							
3	XFI-3			XFI-3							
4	XFI-4	XLAUI-1		XFI-4	XLAUI-1						
5	XFI-5			XFI-5							
6	XFI-6			XFI-6							
7	XFI-7			XFI-7							
8	XFI-8	XLAUI-2		XFI-8							
9	XFI-9			XFI-9							
10	XFI-10			XFI-12	XLAUI-3	CAUI-1					
11	XFI-11			XFI-13							
12	XFI-12	XLAUI-3	CAUI-1	XFI-14							
13	XFI-13			XFI-15							
14	XFI-14			XFI-16	XLAUI-4						
15	XFI-15			XFI-17							
16	XFI-16	XLAUI-4		XFI-18							
17	XFI-17			XFI-19							
18	XFI-18			XFI-20							
19	XFI-19			XFI-21							
20	XFI-20	XLAUI-5		XFI-24	XLAUI-6	CAUI-2					
21	XFI-21			XFI-25							
22	XFI-22			XFI-26							
23	XFI-23			XFI-27							
24	XFI-24	XLAUI-6	CAUI-2	XFI-28	XLAUI-7			ILKN-LA 4/8/12/16/24 lanes			
25	XFI-25			XFI-29							
26	XFI-26			XFI-30							
27	XFI-27			XFI-31							
28	XFI-28	XLAUI-7		XFI-32							
29	XFI-29			XFI-33							
30	XFI-30			XFI-36	XLAUI-9	CAUI-3					
31	XFI-31			XFI-37							
32	XFI-32	XLAUI-8		XFI-38						GCI lanes 7:4	
33	XFI-33			XFI-39							
34	XFI-34			XFI-40	XLAUI-10						
35	XFI-35			XFI-41							

SerDes Lane	10GbE XFI/SFI/ 10GBASE-KR	40GbE XLAUI/XLPPI/ 40GBASE-KR4	100GbE CAUI/CPPI	10GbE XFI/SFI/ 10GBASE-KR	40GbE XLAUI/XLPPI/ 40GBASE-KR4	100GbE CAUI/CPPI	ILKN	ILKN-LA	PCIe	GCI	1GbE Debug Port
36	XFI-36	XLAUI-9	CAUI-3	XFI-42	XLAUI-10 (cont.)	CAUI-3 (cont.)	ILKN-0 (cont.)	ILKN-LA (cont.)		GCI lanes 3:0	
37	XFI-37			XFI-43							
38	XFI-38			XFI-44							
39	XFI-39			XFI-45							
40	XFI-40	XLAUI-10							PCIe Up to 8 lanes	GCI lanes 7:4	
41	XFI-41										
42	XFI-42										
43	XFI-43										
44	XFI-44	XLAUI-11							PCIe Up to 4 lanes	GCI lanes 3:0	
45	XFI-45										
46	XFI-46										
47	XFI-47										
48	--	--	--	XFI-46	--	--	--	--	x1 PCIe	--	--
49	--	--	--	XFI-47	--	--	--	--	--	--	SGMII-Debug

Legend:		
	GCI	GigaChip Interface
	ILKN	Interlaken
	ILKN-LA	Interlaken Look-Aside
	PCIe	PCI Express

Notes:

(1) The XFI, XLAUI and CAUI interfaces can operate in either Option A; 12-lane mode or Option B: 10-lane mode. The configuration is global and affects the 10GbE, 40GbE and 100GbE interfaces, meaning that all these interfaces are either in one mode or the other. Each interface side (west/east) must be either Option A or Option B.

a. Option A: 12-lane mode.
XFI lanes are contiguous. XLAUI lanes are contiguous.
The 2 lanes between the CAUI interfaces may be used for XFI ports. For example, lanes 0-9 are CAUI-0 and lanes 10-11 are XFI-11 & XFI-12.

b. Option B: 10-lane mode.
XFI lanes are grouped into 10 lanes. The last two lanes (XFI-46, XFI-47) may be routed to SerDes lanes 48-49. This mode does not cause shifting. This leaves lanes 40-47 available for another interface such as PCIe.
Maximum of 8 XLAUI interfaces; the 2 lanes between the XLAUI interfaces may be used for XFI ports.
CAUI lanes are contiguous.

(2) ILKN-0 interface may be from 1-48 lanes and must start at SerDes lane 0.

(3) Supports an external 400GbE MAC/FPGA using ILKN-0 lanes 0-39.

(4) PCIe, ILKN, and ILKN-LA lane logical numbering can be swapped to ease PCB design.
E.g. a configuration of ILKN-0 that occupies lanes 0 to 7 will become lanes 7 to 0, etc.
For XLAUI and CAUI lane swapping is not required since it is done by the external receiver according to IEEE 802.3ba Clause 82.

(5) PCIE_MODES configuration register in SERDES block configures the PCIe width options and PCIe physical lane 0 mapping to SerDes lane:
- x1 on SerDes lane 48.
- x1/x2 on SerDes lanes 46-47 (SerDes lane 46 is PCIe physical lane 0). The SerDes quad cannot be used by another interface.
- x1/x2/x4 on SerDes lanes 44-47 (SerDes lane 44 is PCIe physical lane 0). The SerDes quad cannot be used by another interface.
- x1/x2/x4/x8 on SerDes lanes 40-47 (SerDes lane 40 is PCIe physical lane 0). Both SerDes quads

cannot be used by another interface.
The PCIe MAC supports full or partial lane reversal by auto-negotiation or software, enabling the connection of external host PCIe physical lane 0 to SerDes lane 47.

(6) Only the west side PCIe interface with any width and rate can be used by an external host to access the NPS configuration registers after training.

(7) Either 4- or 8-lane GCI is supported on each side (east/west):
GCI[7:0] = lanes[47:44] + [43:40] or lanes[39:36] + [35:32],
GCI[3:0] = lanes[47:44] or lanes[39:36].

(8) The ILKN-LA Tx interface can use 24, 16, 12, 8 or 4 lanes starting at SerDes lane 47, and the Rx interface can use 1 – 24 SerDes lanes starting at SerDes lane 47. E.g., for 24 lanes SerDes lane 47 is TCAM lane 0, and after lane swap SerDes lane 24 is TCAM lane 0.

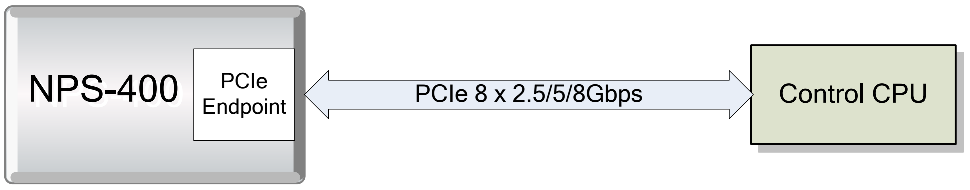
(9) A configuration of both ILKN-LA and GCI interfaces on the same side is not supported.

(10) A configuration of both ILKN-LA and x1/x2/x4/x8 PCIe (lanes 40-47) interfaces on the same side is not supported. ILKN-LA can be used together with x1 PCIe interface (lane 48).

2.1.2.1 PCIe Interfaces

- 2x 8-lane PCI Express (PCIe) Gen 1/2/3 at 2.5/5/8Gbps; one PCIe interface per side. See SerDes Allocation Scheme.
- Endpoint functionality with DMA capability into shared memory space.

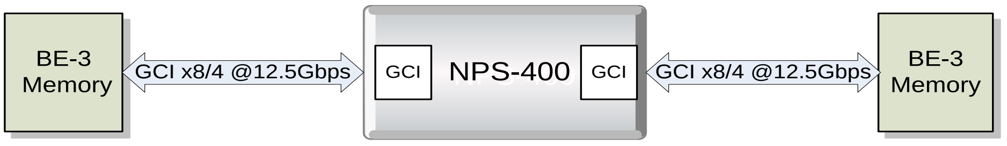
Figure 2-3. PCIe interfaces



2.1.2.2 GCI Interfaces

- Allocate up to 8 SerDes @ 12.5Gbps for a GCI interface on each side (east and west). See SerDes Allocation Scheme.
- Supports one x4/x8 lane GCI host interface per side to a Bandwidth Engine-3 capable memory device
- Four scheduling domains per GCI interface.
- 90% BW efficiency for short transactions.
- Serves search/control-structures and metering/statistics policers/counters.
- Search structures DRAM replica auto-access when GCI is congested.

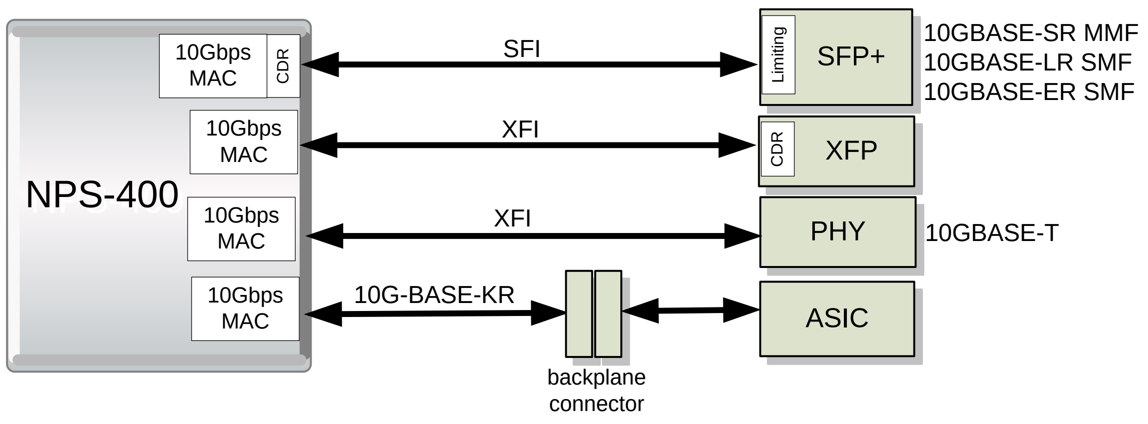
Figure 2-4. GCI interfaces



2.1.2.3 10GbE Interfaces

- The XFI/SFI/10GBASE-KR interfaces are supported on lanes 0-47. See SerDes Allocation Scheme.
- IEEE 802.3 support:
  - IEEE 802.3 Clause 49 (PCS) support
  - IEEE 802.3 Clause 51 (PMA) support
  - IEEE 802.3 Clause 72 (PMD) support of 10GBASE-KR
- Interface features
  - Polarity swapping of +/- on high-speed (programmable)
  - Internally terminated TX and RX
  - Per interface TX disable software control
  - Per interface RX power down software control
  - Meets jitter requirements for telecom/IEEE.
  - Integrated SFI Clock Data Recovery (CDR).
- 10GBASE-KR for backplane connectivity
- Supporting XFI direct connection to XFP modules
  - XFI high-speed SerDes compatibility support (INF-8077)
- Supporting SFI direct connection to SFP+ limiting modules
  - 10GBASE-SR/LR/ER SFP+ direct attachment support
  - Electronic Dispersion Compensator (EDC) is not integrated.
  - Support for SFP+ MSA SFF-8431 Rev 4.1 high-speed electrical
- Loop-timing (RX recovered) configurable
- 10GBASE-KR interfaces support FEC (Forward Error Correction)

Figure 2-5. 10GbE interfaces



2.1.2.4 40GbE and 100GbE Interfaces

- XLAUI interfaces are supported on lanes 0-47. See SerDes Allocation Scheme.
- CAUI interfaces are supported on lanes 0-45. See SerDes Allocation Scheme.
- IEEE 803.2 support
- IEEE 802.3ba Clause 82 (PCS) support
- IEEE 802.3ba Clause 83 (PMA) support
- IEEE 802.3ba Clause 84 PMD support
- Support for XLAUI interface (40G)
- Support for XLPPI interface (40G)
- Support for CAUI interface (100G)
- Support for CPPI interface (100G)
- 40GBASE-SR4 PMD direct attachment support, IEEE 802.3ba Clause 86
- 40GBASE-LR4 PMD direct attachment support, IEEE 802.3ba Clause 87
- 100GBASE-SR10 PMD direct attachment support, IEEE 802.3ba Clause 86
- Backplane Ethernet auto-negotiation support, IEEE 802.3ap Clause 73
- Polarity swapping of +/- on high-speed (programmable)
- Per SerDes power down, and global
- Per SerDes TX enable/disable, and global
- Loop-timing (RX recovered) configurable (SynchE)
- 40GBASE-KR4 interfaces support FEC (Forward Error Correction)

Figure 2-6. 40GbE interfaces

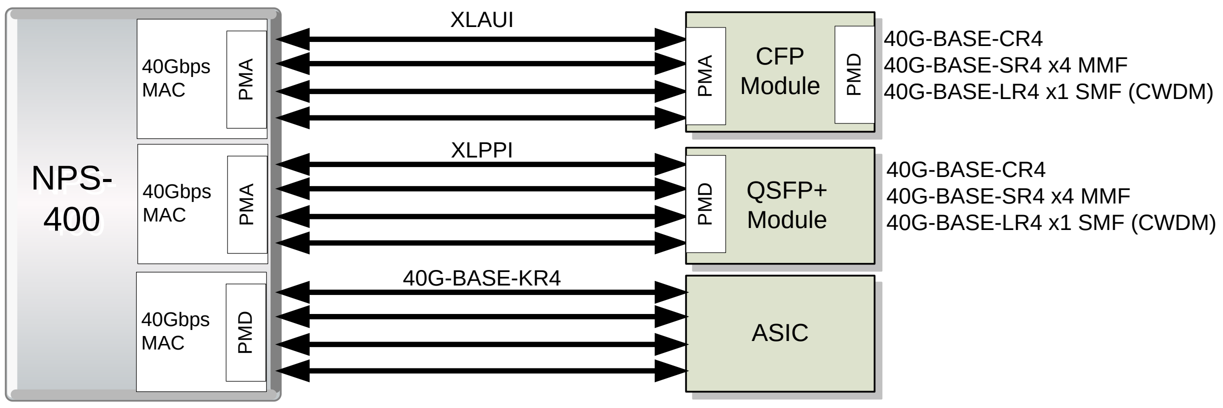
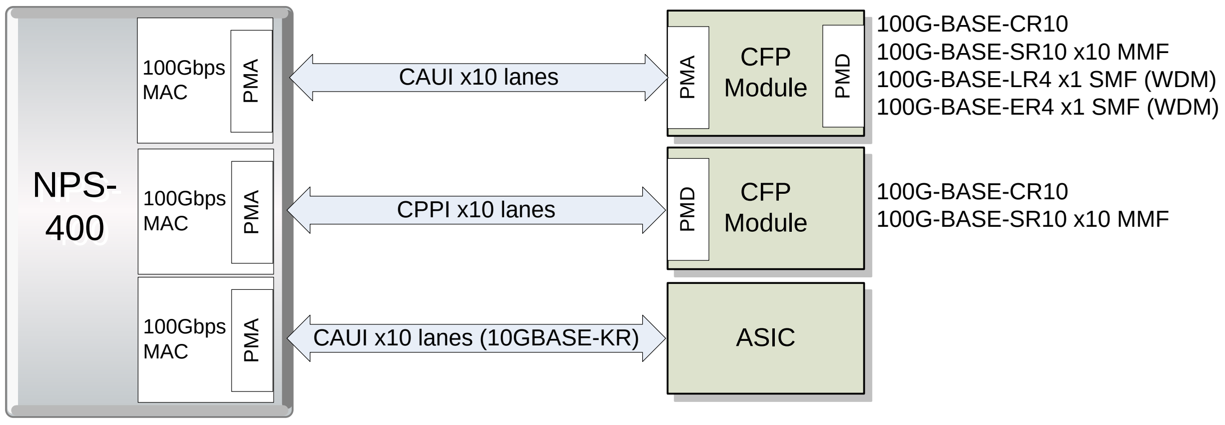


Figure 2-7. 100GbE interfaces



2.1.2.5 10GBASE-KR and 40GBASE-KR4 Interfaces

- 10GBASE-KR/40GBASE-KR interfaces are supported on lanes 0-47. See SerDes Allocation Scheme.
- IEEE 802.3ba Clause 84 (PMD) support
- IEEE 802.3ba Clause 82 (PCS) support
- IEEE 802.3ba Clause 83 (PMA) support
- IEEE 802.3 Clause 72 support
- IEEE 802.3ap Clause 73 (backplane Ethernet auto-negotiation) support
- Internally terminated TX and RX
- Polarity swapping of +/- on high-speed (programmable)
- Per interface TX disable software control
- Per interface RX power down software control
- Programmable driver swing transmit equalization (pre-emphasis) output driver slew rate control
- Programmable linear receive equalization
- Performance up to 40+ inches (improved FR4) and 2 connectors at 10.3125Gbps (nominal)
- Auto-negotiation (on any lane for 40G): None, Symmetric, Asymmetric, Asymmetric Tx only.

2.1.2.6 Ethernet MAC Supported Configurations

- MAC source address for link level flow control.
- Enable/ignore receive link level flow control. Enable/ignore receive priority level flow control.
- Enable transmit link level flow control. Enable transmit priority level flow control. Transmit FC interval.
- Receive/transmit priority flow control translate profile, mask and value.
- SW process or MAC drop received flow control frames.
- CRC: Auto add CRC on transmit. Auto strip CRC on receive. Ignore receive CRC errors.
- Ignore frame length error. Maximal frame length: 15B - 15808B.
- Enable local/remote fault detection. Unidirectional fault mode.
- Auto pad short transmitted frames to 64B (or 76B with SW preamble).
- SW preamble of 0B, 8B, 12B, 16B.
- Average transmit IPG: 8B, 12B, 16B, ..64B. LAN or WAN Phy IPG.
- Minimal receive IPG: disabled, 1B - 15B.
- Profile selector (1-8) for maximum frame length MAC statistics.
- Enable IEEE1588 receive time stamping. Enable IEEE1588 transmit time stamping.

2.1.2.7 Interlaken Interfaces

- Compatible with Interlaken protocol definition V1.2
- SerDes rate of 6.25Gbps, 10.3125Gbps or 12.5Gbps
- SerDes lane assignment options (see SerDes Allocation Scheme):
  - ILKN-0 is supported on lanes 0-47 in ascending order from SerDes-0.
- Interlaken parameters:
  - Upper limit of transmit BurstMax: 256 bytes
  - Minimum BurstShort requirement: 32 bytes for ILKN interface with up to 24 lanes, and 64 bytes for ILKN interface with more than 24 lanes
  - Transmit BurstMin: 32 bytes, 64 bytes
  - Scheduling enhancement for bandwidth efficiency
  - Meta frame length range: 64-8K of 8-byte words
  - Maximal allowed receive skew between lanes: 240 UI
  - Retransmit extension: supported
  - Lane redundancy: not supported
- Flow control:
  - Supports in-band and out-of-band flow control receive and transmit for up to 128 channels per interface side
  - Two OOB flow control full duplex buses
  - Link level and channel based calendar. Enable/disable receive link level flow control.
  - In-band receive and transmit link-level flow control can be carried in multiple use field or calendar field of control word. Receive/transmit LL FC repeated in every control word or once per calendar. Bit position of FC, 0-7 multiuse bits, 0-255 calendar bits.
  - Receive and transmit priority flow control enable/disable. Priority flow control can be carried in multiple use field or calendar field of control word. Bit position of FC, 0-7 multiuse bits, 0-248 calendar bits. Priority FC translate profile.
  - Receive/transmit per channel FC or per channel + per priority. Configurable calendar bit position.
  - Transmit rate limiting in granularity of 1Gbps up to link capacity.
  - In-band status messaging
  - Optional out-of-band status messaging (Appendix A of specification)
- Channelization:
  - Receive side:
    - Up to 128 channels per Interlaken port on each side. The 128 channel contexts are shared by all interface ports, and each Ethernet port requires one context.
    - Packet mode – one full packet at a time
  - Transmit side:
    - Packet mode only (one full packet at a time) – up to 128 channels.
- CRC control:
  - Append 4B CRC to transmitted frames. Strip 4B CRC from received frame. Ignore receive CRC error. Frame start offset for receive/transmit CRC - 0 to 120B in multiple of 8B.
- IEEE1588 timestamp control:
  - Enable receive and/or transmit time stamping. Receive time stamp frame length filter min. and max. length. TM engine select for transmit 2-step time stamp enable. Two TM L4 flows for time stamp traffic.
- Diagnostics
  - Statistic counters:
    - RX_Packets: Number of packets received (per interface/up to 64 channels)
    - RX_Bytes: Number of bytes received (per interface/up to 64 channels)
    - TX_Packets: Number of packets transmitted (per interface/up to 64 channels)

        - TX_Bytes: Number of bytes transmitted (per interface/up to 64 channels)
        - RX_Bad_Packets: Number of packets with errors (i.e. bad CRC, FIFO overflow, ERR bit set, etc., per interface)
        - RX_CRC_24_Error: Number of bursts with a detected CRC error
        - RX_CRC_32_Error: Number of packets with a detected CRC error
        - RX_BurstMax_Error: Number of bursts received longer than the BurstMax parameter
        - RX_Alignment_Error: Number of alignment sequences received in error (i.e. those that violate the current alignment)
        - RX_Alignment_Failure: Number of times alignment was lost (after four consecutive RX_Alignment_Errors)
        - RX_Word_Sync_Error: Number of times a lane lost word boundary synchronization (per lane)
        - RX_Lane_CRC_Error: Number of errors in the lane CRC (per lane)
        - RX_Bad_Control_Error: Number of words received with Control Word framing (‘x10’) that do not match one of the defined Control Words

        📖 Refer to *NPS-400 Configuration Registers Reference Manual*, ILKN_STAT SRAM in NET_ILKN block.
    - Interrupts are generated on link synchronization status change (on data path and on out of band FC interface)
- Test patterns:
    - PRBS7, PRBS23, PRBS31, two programmable patterns
- Time stamping:
    - Support for input and output time stamping

2.1.2.8 Interlaken-LA Interfaces

- Compatible with Interlaken Look-Aside Protocol Definition Revision 1.1
- SerDes rate of 6.25Gbps, 10.3125Gbps or 12.5Gbps
- SerDes lane assignment options (see SerDes Allocation Scheme):
    - ILKN-LA is supported on lanes 24-47 starting from SerDes-47
    - Supported number of Tx lanes: 24, 16, 12, 8 or 4 lanes
    - Supported number of Rx lanes: 1-24
- ILKN-LA lane swap on receive and/or transmit
    - ILKN-LA lane logical numbering can be swapped to ease PCB design, e.g. a configuration that occupies lanes 32 to 47 will become lanes 47 to 32, etc.
- Interlaken parameters:
    - Upper limit of transmit BurstMax: 256 bytes
    - Minimum BurstShort requirement: 8, 16, 32 bytes. Enable receive BurstShort check.
    - Meta frame length range: 16-16K of 8-byte words set for receive and transmit. Enable/disable include of skip word in metaframe.
    - Maximal allowed receive skew between lanes: 240 UI
    - Lane redundancy: not supported
- Flow control:
    - In-band status messaging: supported
- Diagnostics
    - Ignore CRC error (pass fault indication for SW processing)
    - Ignore receive link status
    - Statistic counters:
        - RX_Packets: Number of packets received (per interface)
        - RX_Bytes: Number of bytes received (per interface)

  - TX_Packets: Number of packets transmitted (per interface)
  - TX_Bad_Packets: Number of bad packets transmitted (per interface)
  - TX_Bytes: Number of bytes transmitted (per interface)
  - RX_Bad_Packets: Number of packets with errors (i.e. bad CRC, ERR bit set, etc., per interface)
  - RX_CRC_Error: Number of bursts with a detected CRC error
  - RX_Alignment_Error: Number of alignment sequences received in error (i.e. those that violate the current alignment)
  - RX_Alignment_Failure: Number of times alignment was lost (after four consecutive RX_Alignment_Errors)
  - RX_Word_Sync_Error: Number of times a lane lost word boundary synchronization (per lane)
  - RX_Lane_CRC_Error: Number of errors in the lane CRC (per lane)
  - RX_Bad_Control_Error: Number of words received with Control Word framing (‘x10’) that do not match one of the defined Control Words

  📖 Refer to *NPS-400 Configuration Registers Reference Manual*, Indirect in NET_ILKN block.

- Interrupts are generated on link synchronization status change (on data path and on out-of-band FC interface)

2.1.2.9 Management Interfaces

NPS-400 contains various management interfaces including LAN, UART/RS232.

2.1.3 Internal Memory

Internal unified 16MB SRAM memory.

- Flexibly allocated to frame data, search structures, packet context, statistics and TM queuing control
- 256 banks, non-blocking access, up to 256 parallel accesses within clusters, up to 128 parallel accesses between clusters
- Up to 256B accesses per second

2.1.4 Unified External Memory Interfaces

The NPS-400 can interface with external memory (DDR4/DDR3) for search structures, traffic buffering, control memory, statistics counters and CTOPs memory. The data stored in the DDR4/DDR3 is shared.

The DRAM is accessible via 12 DDR4/DDR3 interfaces (groups 0 to 11), each with a 32-bit wide data bus. Each 32-bit interface is managed by two controllers sharing the address and control and managing 16b data bus each. Each interface supports two 16-bit devices or four 8-bit devices. DDR3 bus voltage is 1.5V or 1.35 (low voltage) and DDR4 bus voltage is 1.2V. Address mirroring for “clam-shell” board topology support. The memory interface supports DDR3 data rate of up to 2133 Mbps and DDR4 devices at up to 2666 Mbps.

The external memory size is dependent on the external DRAM density, for up to 8GB per interface or a total of 96GB.

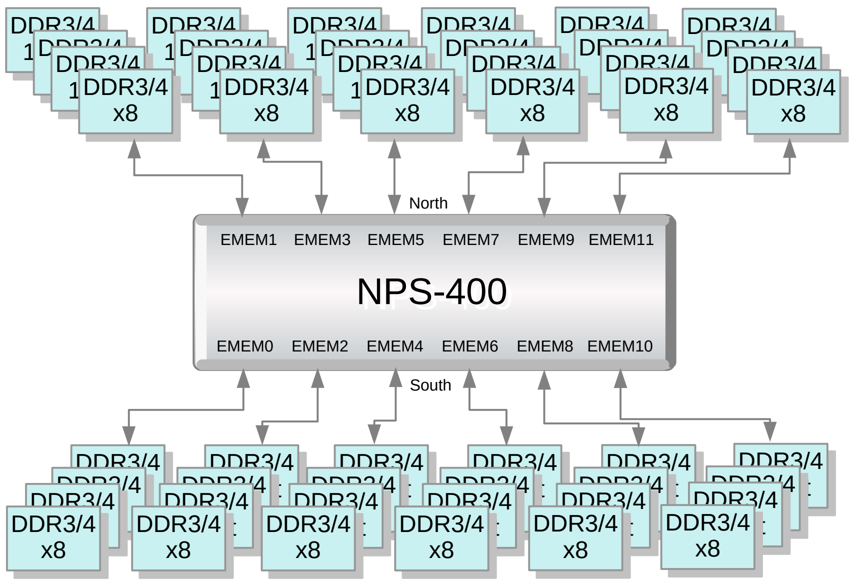
- Traffic Manager can utilize up to 16GB
- Search structures can utilize up to 16GB
- Statistics structures can utilize up to 20GB

DRAM structures are ECC protected. Memory requirements are flexible to match application requirements. Memory usage is configurable to 0, 4, 8 and 12 interfaces. Applications can share memory devices and be configured to use dedicated banks. Auto replication of structures for high access rates and redundancy is supported.

2.1.4.1 External Memory Device Examples

The following figure shows a full capacity DDR3/DDR4 with 12 interfaces supporting 4 devices x8 for a total of 48 devices. The memory interface alternates between the south and north side of the chip starting at the south side.

Figure 2-8. Maximum capacity with x8 DDR3/DDR4 devices example


DDR3/4 x8
North
EMEM1 EMEM3 EMEM5 EMEM7 EMEM9 EMEM11
NPS-400
EMEM0 EMEM2 EMEM4 EMEM6 EMEM8 EMEM10
South

The following figure shows a typical capacity DDR3/DDR4 with 12 interfaces supporting 2 devices x16 for a total of 24 devices. The memory interface alternates between the south and north side of the chip starting at the south side.

Figure 2-9. Maximum capacity with x16 DDR3/DDR4 devices example

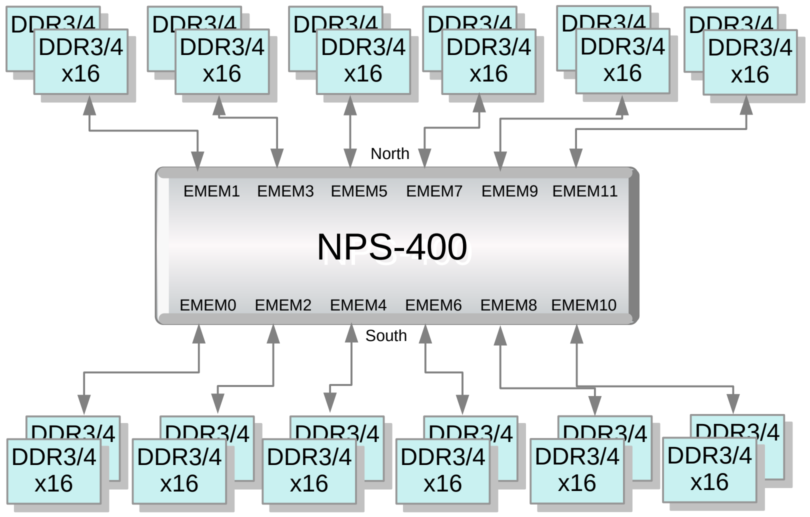

DDR3/4 x16
North
EMEM1 EMEM3 EMEM5 EMEM7 EMEM9 EMEM11
NPS-400
EMEM0 EMEM2 EMEM4 EMEM6 EMEM8 EMEM10
South

Table 2-2. Total DDR3 external memory size [Gbyte]

COMPONENT DENSITY/WIDTH (Gbit)	1Gb	2Gb	4Gb	8Gb
x16	3GB	6GB	12GB	24GB
x8	6GB	12GB	24GB	48GB

Table 2-3. Total DDR4 external memory size [Gbyte]

COMPONENT DENSITY/WIDTH (Gbit)	2Gb	4Gb	8Gb	16Gb
x16	6GB	12GB	24GB	48GB
x8	12GB	24GB	48GB	96GB

2.1.5 External Management CPU (Host) Interfaces

The NPS-400 provides a PCI Express Gen 1/2/3 endpoint interface at 2.5/5/8Gbps. The PCI Express interface is used for configuration and management of the NPS-400 as well as frame exchange between the external host and the NPS-400.

Although frames can be exchanged via the PCI Express interface (endpoint), a host CPU should also be connected to one or more of the host port(s) for faster traffic exchange. A standby host could also be connected to the other host port for redundancy of the packet exchange.

Features:

- Lane 48 on each side (east/west) can be used for a x1 PCIe connection
  - PCIe Gen1/2/3 at 2.5/5/8Gbps
  - Enables an external host to configure the NPS; only PCIe west has access to the NPS configuration registers
- Lane 49 on each side (east/west) can be used as a 1GbE SGMII/1000BASE-X debug port
  - Interrupt out/in signals
  - The port can be used as a debug port for the Linux OS, e.g. Telnet, FTP, remote GDB, etc.
  - The MAC does not support DMA and all frame transactions have to be managed by CTOP software, limiting the port throughput. The debug port shall not be used for data plane applications.

2.1.6 Auxiliary Interfaces

The NPS-400 provides auxiliary interfaces for system management:

- LED serial interface for all ports status exporting port status
- Two out-of-band bidirectional Interlaken/SPI4.2-Status channelized interfaces (one per TM entity)
  - Flow control interface for port and TM entities
  - Congestion exporting and messaging
- Side-band Interlaken MAC link-level exporting flow control status
- Inputs/outputs for system wide synchronization of on-chip real time clock (RTC) for IEEE 1588v2 support
- JTAG test and debug port
- Synch Ethernet
  - SerDes recovered and reference clocks
- MDC/MDIO Serial Management Master and Slave interfaces
- SPI supporting full-duplex serial protocol by Motorola®
- I^{2}C Rev. 4 interface
- 4 clock reference, each pair of lanes can select one reference clock

2.1.6.1 Synch Ethernet

- 2x SynchE receive recovered CDR SerDes clock outputs
  - One per interface side
  - Automatic squelch of recovered clock on link failure
- Reference clock inputs
  - Each SerDes pair selects from two SerDes reference clocks of the interface side

2.1.6.2 RTC Interface for System-wide IEEE 1588 and OAM

- RTC_CLK_OUT – Real Time Clock based clock output
- 1PPS_OUT – Real Time Clock One pulse per second output
- RTC_CLK_IN – Real Time Clock reference clock input
- 1PPS_IN – Real Time Clock One pulse per second trigger input

2.1.6.3 MDC/MDIO PHY Management Interfaces

- Slave MDC/MDIO for device management, complies to IEEE 802.3 Clause 22 standard, except for voltage changes
- 2x Master MDC/MDIO for Phy control, each complies to either the IEEE 802.3 Clause 22 or Clause 45 standard, except for voltage changes
- Optional preamble suppression to reduce management cycle from 64 clocks to 32 clocks.
- Configurable MDC clock rate based on core clock: [core clock /16] to [core clock /1024]
- External Phy link status auto scan mechanism, for configurable Phy address range. Link status change triggers interrupt generation.

2.1.6.4 Serial LED Interfaces

- The Serial LED stream contains various port/device status indications and can be connected to LED/logic through a shift register or PLD
- There is one LED interface

2.1.6.5 JTAG

- Boundary Scan Test interface compliant with the IEEE 1149.1 and 1149.6 standards. Supports mandatory and optional boundary scan instructions.
- CTOP debugger connectivity

2.1.6.6 Clocks, Reset & Interrupt

- Single reset
- Interrupt out to an external CPU
- Two interrupt inputs for embedded autonomous CPUs including any CTOP
- Two dedicated general purpose input/output (GPIO) signals + two GPIO multiplexed on M<s>_MDIO
- Operates based on single 25MHz reference, with the following PLLs:
  - Core clock
  - Embedded memory (based on internal memory speeds)
  - Embedded autonomous CPUs core clock
  - DDR3/DDR4 DRAM interface clock

2.2 Solution Examples

NPS-400 flexibility enables building platforms that deliver a wide variety of applications for layer 2-3 switching and routing, layer 4-7 stateful session processing, and packet payload manipulation. Unmatched integration allows system vendors to deliver solutions that are cost effective as well as power and board-space efficient.

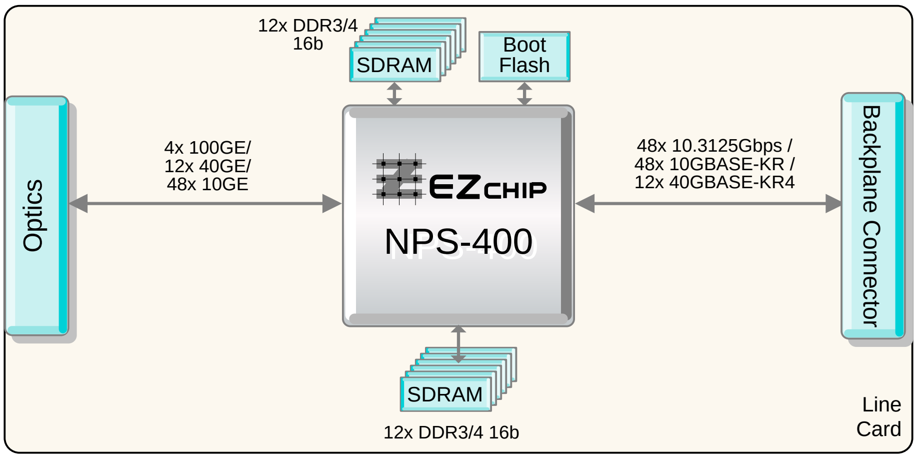
2.2.1 Line Card Solutions

The following line card examples show wire-speed as well as over-subscribed solutions with various complexities. NPS-400 supports connection to either Ethernet-based switch fabrics or proprietary switch fabrics.

2.2.1.1 L2-L7 Line Card

The line card example below shows a L2-L7 line card utilizing the NPS-400 on-chip autonomous CPU. The CPU performance is sufficient for control applications that do not require low memory latencies. The solution is very cost and power efficient, saving the need for an external host CPU and FIC (Fabric Interface Controller). Up to 480Gbps wire speed (ingress) processing is supported.

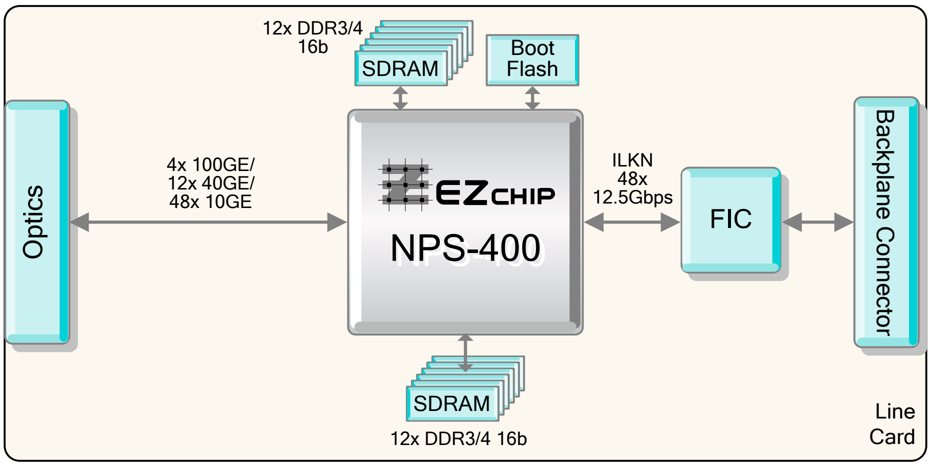
Figure 2-10. L2-L7 line card



2.2.1.2 L2-L7 Line Card with FIC

The line card figure below depicts a basic line card utilizing the NPS-400 on-chip autonomous CPU. The connection to the switch fabric is provided by an Interlaken 48-lane interface via a FIC. Up to 480Gbps wire speed (ingress) processing is supported.

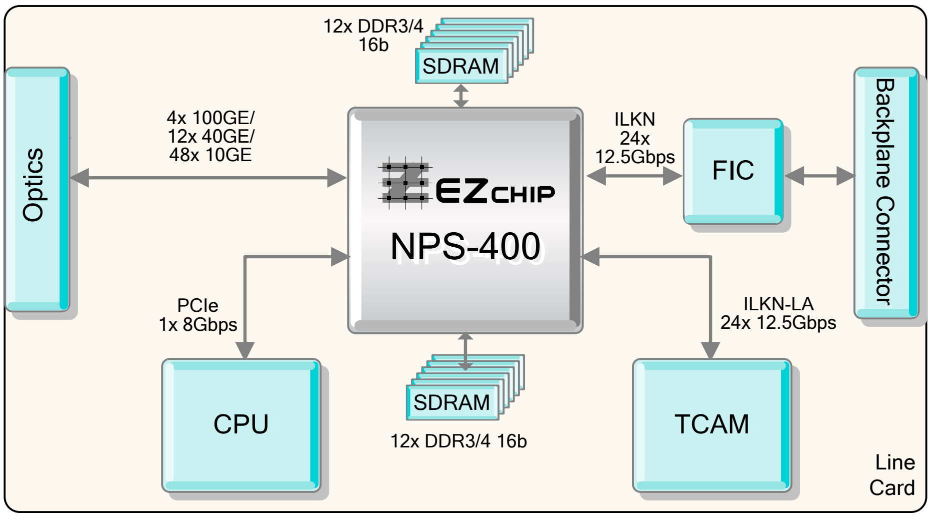
Figure 2-11. L2-L7 line card with FIC



2.2.1.3 L2-L7 Line Card with FIC, TCAM and Host CPU

The line card figure below shows an over-subscribed line card. The connection to the switch fabric is provided by Interlaken 24-lane interface via a FIC. An external TCAM for applications requiring large ACL tables is connected using an Interlaken-LA 24-lane interface. An external host CPU is connected for control applications that require low memory latencies. The CPU is connected with an 8Gbps 1-lane PCIe-host Gen 3 port that enables the CPU to access the NPU control registers after boot.

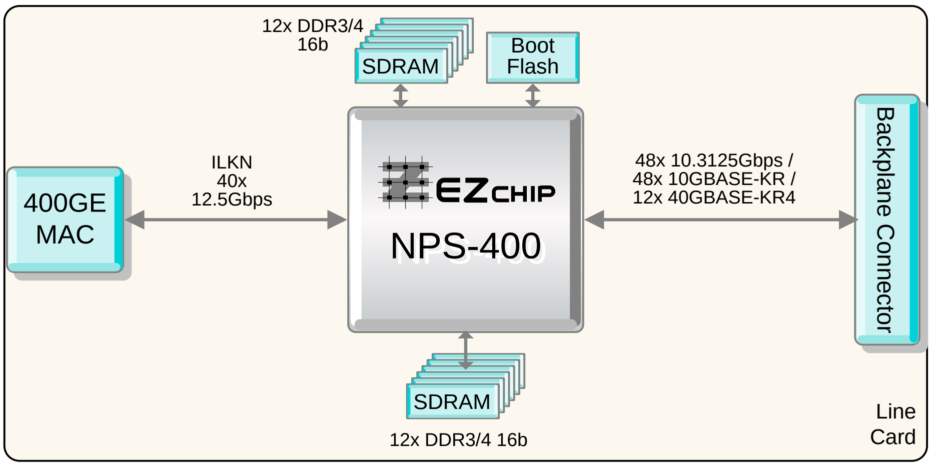
Figure 2-12. Over-subscribed L2-L7 line card with external host CPU, external FIC and TCAM



2.2.1.4 L2-L7 Line Card with 400GE External MAC

The line card figure below depicts a line card utilizing the NPS-400 on-chip autonomous CPU. An Interlaken 40-lane interface at 12.5Gbps is used to connect to an external 400GbE MAC. Up to 400Gbps wire speed (ingress) processing is supported.

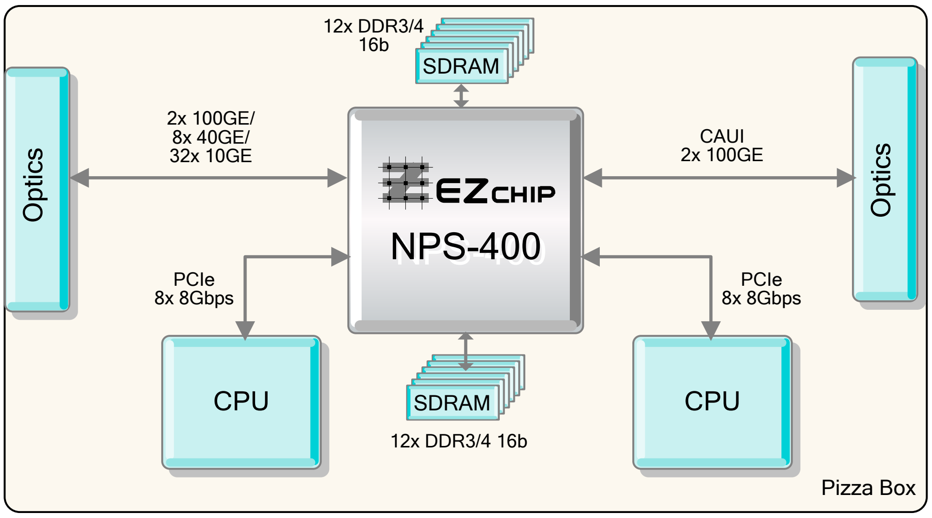
Figure 2-13. L2-L7 400Gbps line card with external 400GE MAC with on-chip autonomous CPU



2.2.2 Pizza Box Appliance with One or More CPUs for Control and Application Processing

In the figure below, the NPS-400 provides the entire network layer 2-7 processing and traffic management. The NPS offloads these functions from the CPU and because of its high throughput can serve more than one CPU. The CPUs receive selected packets from the NPS for exceptions handling, and perform all the protocol, management and application tasks. Each CPU is connected with an 8-lane PCIe Gen 3 port providing 64Gbps throughput. PCIe enables the CPU to access the NPU control registers after boot.

Figure 2-14. L2-L7 pizza box appliance



2.2.3 System Application for Multi-Tbps Operation

The following figure exemplifies a multi-Tbps (Terabit/second) system with multiple 480Gbps line cards inter-connected through *n* fabric switch cards.

The system provides:

- Multi-Tbps of non-blocking, QoS, unicast and multicast operation
- Multiple Ethernet switches
- Multiple NPS-400 devices
- Multi-terabit chassis for mega data centers, enterprise and service provider applications

The fabric switches support Data Center Bridging (DCB). The DCB requires:

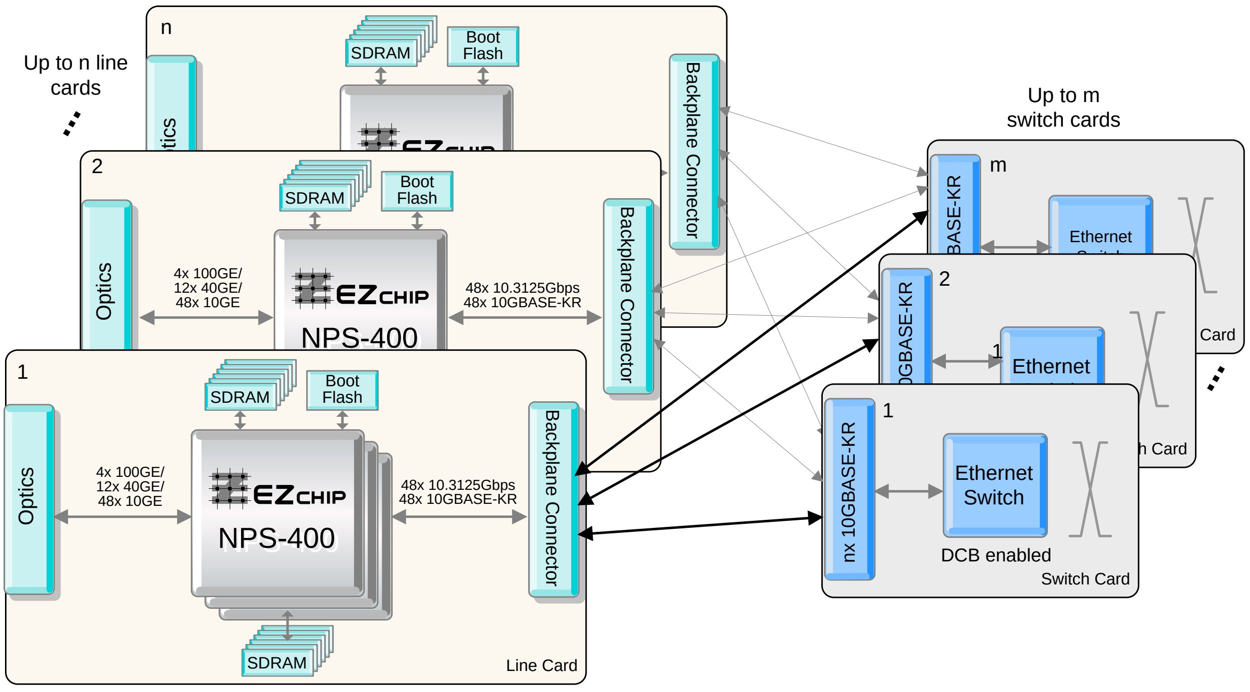
- IEEE 802.1Qbb – Priority-based flow control (PFC). NPS-400 supports PFC on all Ethernet links.
- IEEE 802.1Qau – (Quantized) Congestion Notification (CN and QCN). NPS-400 application supports the standard form and E2E CN proprietary messages.
- IEEE 802.1Qaz – Enhanced Transmission Selection by runtime configuration of TM via one of the CTOPs.

Each NPS allocates 48x 10GBASE-KR lanes for fabric connectivity.

The line card distributes the fabric traffic for even load distribution over the backplane fabric lanes.

Each line card maintains virtual output queuing (VoQ) per system output port CoS. The in-band signaling of end-to-end QCN messages maintains the CoS aware fabric traffic engineering scheme.

Figure 2-15. Multi-terabit chassis application for Multi-Tbps operation



Typical number of switch cards (m) is 4 (with 100GbE), 12 (with 40GbE), 48 (with 10GbE).

3. Data Flow

This section is intended to provide a high-level architectural overview of the data flow through the NPS-400. The section highlights the data path functional blocks and explains the network frames storage descriptor handling and associated job processing handling. The section also covers the functionality of the Network DMAs, Buffer Management Unit, Processor Management Unit. The section further describes the main data flow including the unicast and multicast frame processing.

3.1 High-level Overview

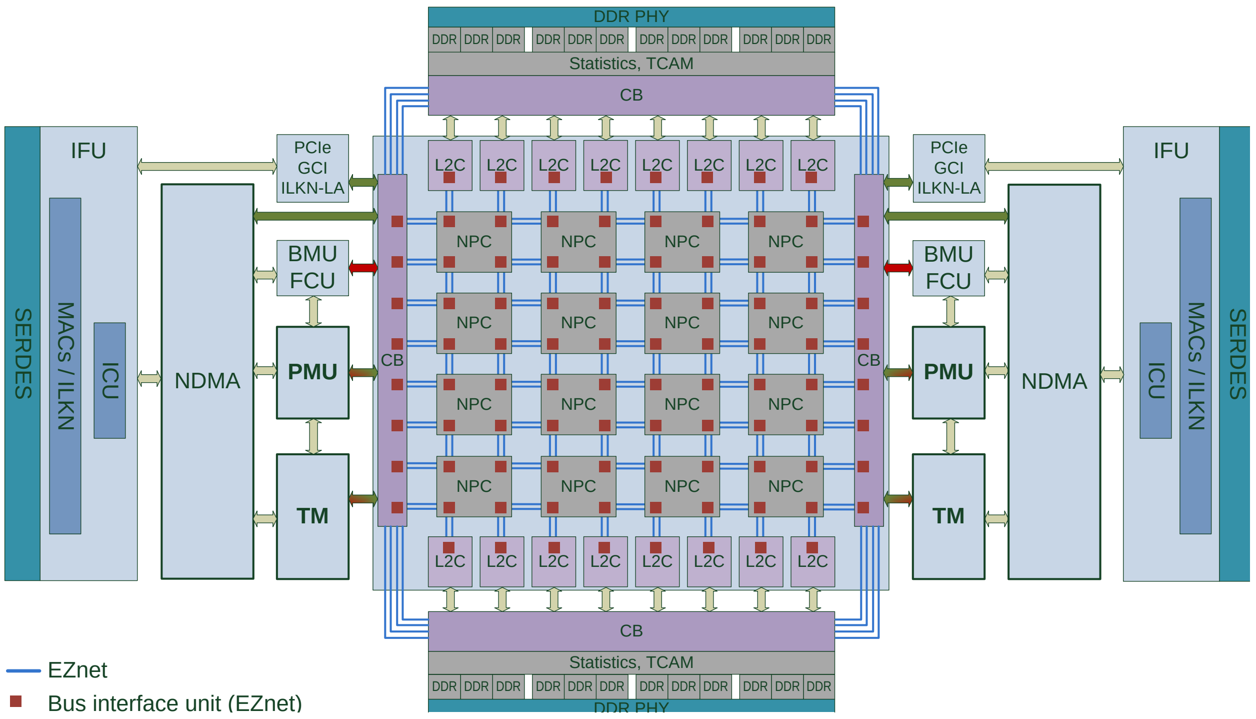
The NPS-400 data path is designed to meet 400Gbps (600Mpps) processing rates. The NPS-400 can receive and HW classify traffic up to entire network interface capacity of 960Gbps enabling smart over-subscription. The NPS-400 blocks are able to move traffic from external interface ports to the packet memory (combination of IMEM and EMEM), to the different execution units and from the packet memory to the external interface. In a computing intensive application, the execution units may not keep the traffic pace and low priority traffic may be dropped based on the (ICU block) HW classified incoming traffic priority.

Packets entering the NPS-400 may traverse the CTOPs (C-language Task Optimized Processors) that perform packet processing, and the Traffic Managers (TMs) that perform the queuing and service provisioning. Packets may arrive on any input interface and packets can be sent to any of the output interfaces.

Packet ordering and scheduling is performed at the Processor Management Unit (PMU).

The NPS-400 is comprised of these main functional blocks:

Figure 3-1. Main functional blocks



- **Interface Units (IFU)** – Configurable interface units which allow a variety of MAC selections with an Interface Classification Unit (ICU) which classifies the incoming traffic into 128 queues. Packets may arrive from any input interface and packets can be sent to any of the output interfaces.
- **Network DMA (NDMA)** – DMA engines which acquire packet buffers from buffer pools and move packet data from/to IFU/EZnet and to/from the NPS-400 memory subsystem.
- **Buffer Manager Unit (BMU)** – Manages pools of buffers, supports buffer allocation and release.
- **Traffic Managers (TM)** – Advanced, configurable, flow-based, hierarchical QoS and bandwidth control.
- **Processor Management Unit (PMU)** – Manages, orders and schedules packet processing jobs and manages message passing between different executing units.
- **Network Processing Cluster (NPC)** – Consists of an array of 16 CTOPs with local memory, L1 cache for look-up data and a data bus interface unit to the EZnet and the PMU. Each CTOP has 16 HW threads. The 16 NPCs are organized in an array, for a total of 256 CTOPs and 4096 HW threads.
- **EZnet** – A highly-scalable cross chip fabric to interconnect CTOPs, TMs, memories and Interface Units. EZnet allows NPS-400 to scale to hundreds of CPU cores with close to linear scaling.

The **Interface Unit (IFU)** integrates MACs, including Ethernet MACs (mixtures of XFI, XLAUI and CAUI interfaces, aggregating more than 480Gbps IO bandwidth). Additionally it may connect the PCI-Express, Interlaken and Interlaken-LA blocks to the SerDes.

The **Buffer Manager Unit (BMU)** provides buffer index pools that dynamically partition IMEM or EMEM spaces into equally sized buffers. It can allocate or release buffers either through a direct interface to side peripherals or through EZnet messages.

The **Processor Management Unit (PMU)** serves as a high-speed scheduler for tasks (jobs) that are being processed in the pipeline. It is application aware and keeps track of ordering and scheduling of all jobs currently in the processing stage. It supports up to 32K of active jobs, either waiting in its physical queues or being processed by a running SW thread in one of the NPCs.

Packets that have completed processing and are waiting for transmission are queued to the **Traffic Manager (TM)** which tracks hundreds of thousands of traffic flows. It provides differentiated classes of service and rate control for traffic flows and provides flow control and packet drop mechanisms, such as WRED.

The NPS-400 is organized symmetrically such that network traffic flows through the SerDes on the east and west sides, and memory interface traffic (DDR3/DDR4) flows through the north and south.

The **Network DMAs (NDMA)** are responsible for storing the incoming Rx traffic into IMEM or EMEM using dynamically allocated buffers (from the BMU), and for reading outgoing packets from the IMEM/ EMEM for transmission on interfaces. It provides an automatic buffer release mechanism for transmitted frames.

Incoming packets undergo initial classification in the **Input Classification Unit (ICU)** and are separated by class of service and other initial parsing results into a few main flows that do not have order dependency on each other. Each such flow may be mapped by the NDMA to an initial physical queue in the PMU. The NDMA allocates 256B buffers to store the incoming frame. The header buffer typically gets an IMEM buffer for efficient processing, while the rest of the data buffers (on longer frames) are typically allocated in EMEM. When frame reception is complete, a Job descriptor is dynamically allocated by the PMU and stays associated with this frame as long as it is in the processing stage. The PMU writes 32B Job Descriptors to predefined locations in IMEM (every allocated job has an associated job descriptor buffer). The PMU uses **EZnet** to communicate with **Messaging and Scheduling Units (MSU)** in each of the NPCs, where the jobs are allocated to the specific core and SW threads that process it. Each NPC holds 16 CPU cores for a total of 256 cores. Each CPU core can run up to 16 SW threads, totaling 4096 SW concurrent threads for packet processing.

The MSU in each NPC collects job-done indications from its associated threads and sends completion messages to the PMU. A job may be forwarded to another processing stage (pipelined application model), or forwarded to the TM for transmission after a single processing stage (run to completion model). The PMU tracks the original order in the physical queue from which the frame arrived and on any forwarding operation it can enforce the same order before the frame is queued to the next stage (either a physical queue or TM). When a frame is forwarded to the TM, its job ID and associated job descriptor are recycled and may be used for another incoming job. This enables holding a limited on-device amount of open jobs currently being processed by the NPC cores and holding very large amount of processed frames in TM queues. TM queues may also extend the PMU job queues using the loopback path, where the TM sends a frame back to PMU for additional processing. In this flow, the frame gets a new job ID when it re-enters the PMU for the second time. Frame context or other means may be used to associate the looped-back frame with this flow and determine the next processing step required.

3.2 Frame Storage and Data Structures

Frames are stored in specific formats utilizing both the IMEM and the EMEM. The same format is used for both the processing stage and the traffic management stage.

This section provides an overview of several of the data structures used in the NPS that affect packet processing or programming.

Specifically included are:

- Working with jobs
    - Job ID (JID)
    - Job descriptor (JD)
- The frame
    - Frame buffers
    - Buffer descriptors
        - Linked Buffers Descriptors
    - Frame descriptor
    - Frame types

For more detailed information on data structure formats refer to the *NPS-400 Programming Manual* and the *EZdp Reference Manual*.

This section does not include information on lookup structures, such as hashes and tables, for these refer to the *NPS-400 Programming Manual*.

3.2.1 Working with Jobs

The NPS boasts a multi-core architecture that supports processing of many concurrent tasks on multiple processors in the system. Each such task, called a *job*, represents a single quantum of work to be performed by the NPS.

The NPS supports the following job types:

- **Interface job** – a job containing a frame received from one of the NPS external network interfaces. This is the most common job type.
- **Loopback job** – a job containing a frame which was created in one of the NPS internal hardware engines, such as one replica of a multicast frame.
- **Confirmation job** – a job containing a confirmation message of the transmission of a single frame, such as those used in the IEEE 1588 network time synchronization protocol.
- **Timer job** – a job signifying a hardware timer has elapsed.
- **User Information job** – a job containing a programmer defined software message.

In order to track and schedule the different jobs being processed in the system at any given time, the NPS is equipped with a hardware block called the Processor Management Unit, or PMU. The PMU is responsible for orchestrating the assignment of jobs for processing to cores or other hardware processing blocks throughout the system, in a fair and prioritized manner.

3.2.1.1 The Job ID

Each job in the system has a globally unique identifier called the Job Identifier, or *job ID* (JID). A job ID is the token by which one references a single job.

It is important to note that a job ID is just a handle: an opaque number with no inherent meaning which is used to address a specific job in the system. Some of the operations done on a job by the programmer may change, or reassign, the job ID of a specific job mid processing implicitly; however, no programmer action is ever required to modify the job ID explicitly.

3.2.1.2 The Job Descriptor

Each job in the system also has a job descriptor (JD), in addition to the job ID.

A job descriptor is a data structure, maintained cooperatively by the Processor Management Unit and the frame processing software, which tracks the state of every job that is being processed by the NPS.

The job descriptor is divided into three parts:

- **The frame descriptor** – the frame descriptor holds information about the frame which is associated with this job, if any. It is described in detail in the "**Error! Reference source not found.**The Frame Descriptor" section.
- **Receive-side job information** – the receive side job information structure holds information about the frame, the resources it uses and the conditions of the system as a whole, such as Ethernet and IP checksums, ordinal number of the frame in the processing queue, various congestion level notifications and so on.
- **Transmit-side job information** – the transmit-side job information structure holds parameters which you may set prior to sending the frame to the Traffic Manager for processing that control the QoS enforced by the Traffic Manager on the frame, transmission statistics aggregation and transmission interface selection. Information in this structure is only relevant for frames transmitted via the Traffic Manager mechanism and has no effect when other mechanisms of frame transmission are being used.

For efficiency reasons, the use of the receive information and transmit information are mutually exclusive as they are defined as fields in a C language union. Therefore, care must be taken not to set the destination information until after the last read of the source information.

3.2.2 The Frame

A frame is a single Ethernet wire transmission that has been received by the NPS or is transmitted by it. The frame data excludes the Ethernet frame preamble, start of frame delimiter and inter-frame gap and may or may not include the 4 byte frame CRC, depending on the network port configuration. All other frame data, from the MAC destination to the frame payload, is kept in the system memory.

3.2.2.1 Frame Buffers

The frame data is kept in system memory as a series of 256 byte buffers; each buffer contains a portion of the frame data. Buffers may reside in either EMEM or IMEM.

Data in a frame buffer is protected by an ECC scheme which is stored outside the buffer itself and managed automatically by the hardware.

The Buffer Management Unit hardware engine, or BMU, keeps tracks of all buffers in the system.

3.2.2.2 Buffer Descriptors

A buffer descriptor, or *BD*, is a data structure that describes a single frame buffer.

Buffer descriptors act as a pointer to buffers in memory; however, since the CTOP is a 32-bit core and the NPS supports more than 4Gb of RAM, it is not a simple pointer that may be accessed directly by load and store. Instead, special Direct Memory Access, or *DMA*, copy operations may be used in order to copy information to and from a frame buffer.

In addition, a buffer descriptor encodes additional metadata about the use of the frame buffer; whether it is used to hold frame data or metadata.

The buffer descriptor is comprised of the following fields:

- Valid data buffer – specifies whether the corresponding buffer holds frame data or not.
- Memory type – specifies whether the corresponding buffer is located in IMEM or EMEM.
- ID – an index used by the hardware engines to locate the corresponding buffer in memory.

3.2.2.2.1 Linked Buffer Descriptors

Each buffer descriptor functions as a pointer to a single 256 byte buffer. Most frames however, are composed of more than a single buffer. Therefore, an efficient representation is needed to describe a series of buffers. The Linked Buffers Descriptor, or *LBD*, is this representation. The linked buffer descriptor is a memory structure (i.e. array) that holds a series of buffer descriptors and other ancillary data.

3.2.2.3 The Frame Descriptor

As we have described in the previous section, each frame is assigned a job that is being tracked by the system. Part of the job descriptor structure is the frame descriptor structure which is a data structure in the system memory.

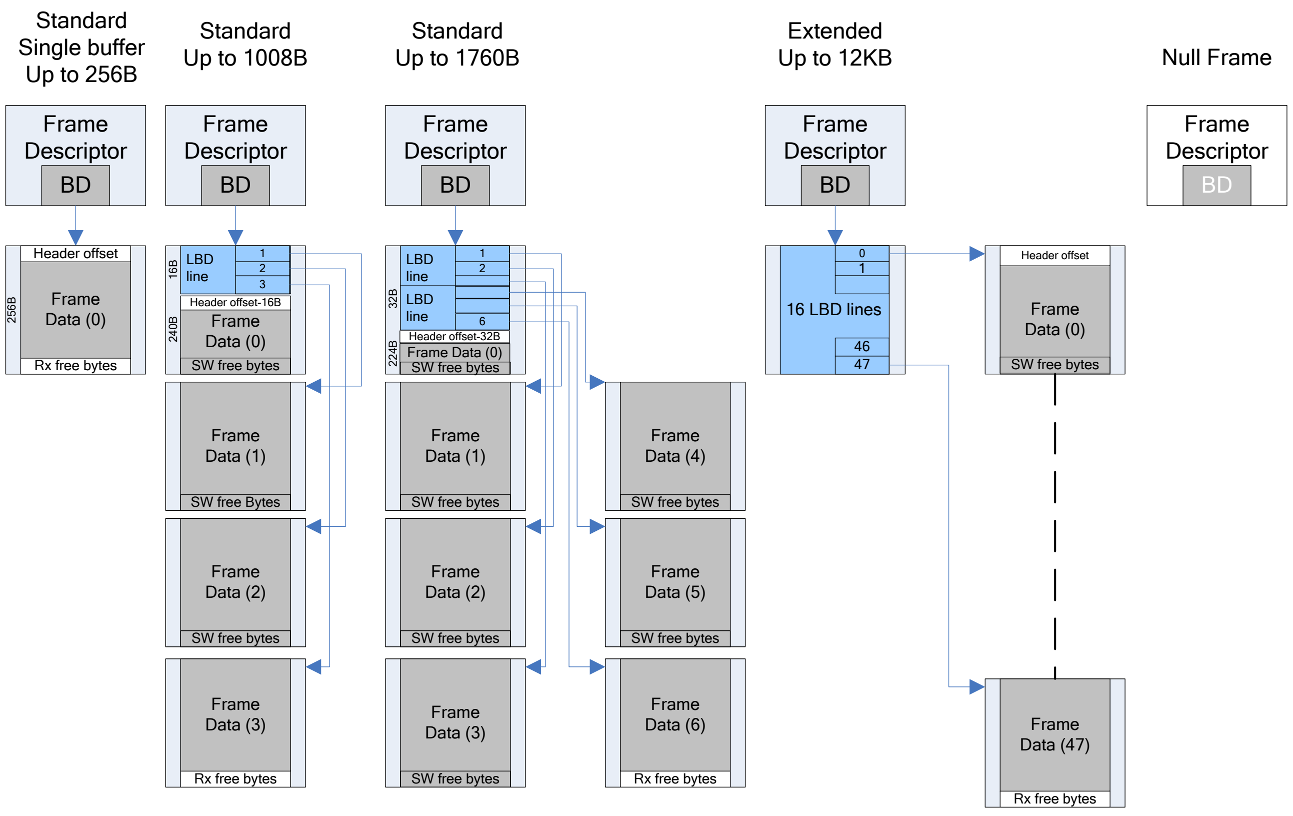
The main fields of a frame descriptor structure:

- **Type** – describes how the buffers that store the frame data are laid out in the system memory (see Frame Types below).
- **Frame Length** – total length in bytes of the frame data starting from the layer 2 frame headers. The frame length excludes all other data not received or transmitted even if it is present in the frame data buffers, such as the frame context bytes, and other bytes preceding the header offset field position.
- **Data Buffers Count** – the number of data buffers that hold frame data. This count does not include buffers containing non-frame data, such as extended linked buffers descriptor buffers or buffers that contain software contexts.
- **Buffer Descriptor** – contains a buffer descriptor for the first, and possibly only, buffer of the frame. Depending on the frame type as described in the *type* field, this buffer may contain frame data, additional buffer descriptors that contain frame data themselves, or both.
- **Header Offset** – the offset from the beginning of the first frame data buffer where frame data starts. Note that small and large linked buffers descriptors and any software context information which is not part of the frame data and are not to be transmitted by the MACs are not included in the offset. The offset may be used to add additional frame headers, where required, without the need to copy the original of the frame.
- **Logical ID** – a software logical context to designate the type of processing the frame should undergo.

3.2.2.4 Frame Types

The NPS supports three frame types which differ from one another in the way the frame data is arranged in system memory.

Figure 3-2. Frame types



These frame types are:

- **The standard frame –** the standard frame type is used to represent standard Ethernet frames (up to 1760B size). The first buffer in this frame type is a partial linked buffer descriptor, of either short (1 line) or long (2 lines) type. The linked buffer descriptor lines in the first buffer hold buffer descriptors, or pointers, to the rest of the frame buffers (up to 6 additional buffers). The remainder of the first frame buffer contains the first bytes of the frame data.
- **The extended frame** – the extended frame type is used to represent frames that require more than 6 frame buffers, such as Ethernet Jumbo frames. The extended frame has a full linked buffer descriptor as its first buffer which points to additional buffers. These buffers in turn hold the actual frame data.
- **The NULL frame** – as the name implies, frames of the NULL frame type have no frame data buffers at all. This kind of frame is useful when allocating a new frame structure for creating a new frame, or when job without associated frame is passed to SW (for example, a timer job generated by HW).

3.2.3 IMEM Frame Data Buffering

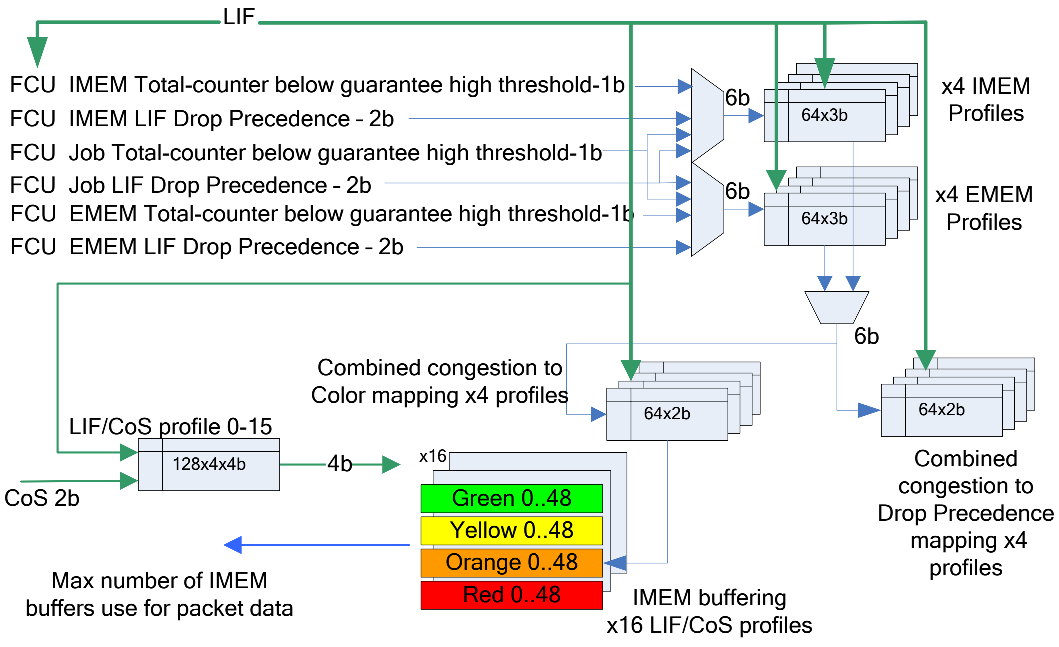
The frame can be buffered in IMEM or EMEM buffers. The buffering is indicated per frame buffer. Upon receive, the first frame buffers maybe buffered in IMEM while subsequent frame buffers reside in EMEM. The SW may chain IMEM and EMEM buffers without restriction and transmit path is capable transmitting IMEM/EMEM interleaved buffers.

3.2.3.1 Receive Frame Data Buffering

The RxNDMA decides what buffer to allocate for frame buffering. The RxNDMA consults the IMEM and EMEM resources accounting management of the FCU for accurate decision. The FCU manages the usage of IMEM, EMEM and Job resources per Logical Interface (LIF) utilizing hierarchical management with resource sharing and resource guarantee.

There is a configuration hierarchy that selects the IMEM/EMEM frame buffering profile. There are four profiles to map FCU indications to 3b IMEM/Job congestion status and 3b EMEM/Job congestion status. The FCU mapping profile is selected per LIF. The IMEM/EMEM/Job 6b combined congestion is mapped with per LIF profile to 2b Color. The 2b Color is mapped with per LIF/CoS profile to IMEM buffering usage profile. The buffering profile determines the maximal number of IMEM buffers to use per received frame, before using EMEM buffers. The number of buffers is configurable in range 0..48.

Figure 3-3. Frame buffering profiles


LIF
FCU IMEM Total-counter below guarantee high threshold-1b
FCU IMEM LIF Drop Precedence - 2b
FCU Job Total-counter below guarantee high threshold-1b
FCU Job LIF Drop Precedence - 2b
FCU EMEM Total-counter below guarantee high threshold-1b
FCU EMEM LIF Drop Precedence - 2b
6b
64x3b
x4 IMEM Profiles
6b
64x3b
x4 EMEM Profiles
6b
Combined congestion to Color mapping x4 profiles
64x2b
64x2b
Combined congestion to Drop Precedence mapping x4 profiles
LIF/CoS profile 0-15
128x4x4b
4b
CoS 2b
x16
Green 0..48
Yellow 0..48
Orange 0..48
Red 0..48
Max number of IMEM buffers use for packet data
IMEM buffering x16 LIF/CoS profiles

3.3 BMU Overview

The NPS architecture relies on the Buffer Management Unit (BMU) for the central management of memory buffer allocation and recycling. The BMU maintains lists of indexes that represent memory resources.

The BMU serves several architectural units:

- RxNDMA – The RxNDMA is served by allocating memory buffers for frame storage.
- TxNDMA – The TxNDMA is served by recycling the memory buffers after frame transmission.
- PMU – The PMU is served by allocating and recycling memory for Job descriptors.
- SW – The SW application uses the BMU for managing general purpose index lists for memory allocation and general index queue management.
- The BMU use the Flow Control Unit (FCU) to manage the resource usage accounting of IMEM frame buffers, EMEM frame buffers and PMU jobs. The FCU is updated with the allocation and recycle of resources.

There are two instantiations of BMU units – one on each device side (west/east).

Each BMU serves the other units on its side. The BMU units interconnect for resource sharing.

The resources managed by BMU are:

- EMEM buffers for frames.
- IMEM buffers for frames and Jobs.
- General Index queues for SW usage.

Refer to the *Buffer Management Unit* chapter for more details.

3.4 FCU Overview

The Flow Control Unit (FCU) is responsible for managing the resource usage counters for three budgets:

- Frame data memory buffers in IMEM.
- Frame data memory buffers in EMEM.
- Job descriptor memory buffers in IMEM.

The FCU receives resource usage updates (increments/decrements) from the BMU for frame memory buffers and from the PMU (through BMU) for Job descriptors. The application SW also performs resource budget updates.

The resource management is organized in a hierarchical manner that allows various schemes of resource sharing. The hierarchy is composed of Budget ID, Logical Interface, Group and Global levels.

Refer to the *Congestion Management and Flow Control* chapter for more details.

3.5 NDMA Overview

The Network DMAs (NDMA) are responsible for storing the incoming Rx traffic into IMEM or EMEM using dynamically allocated buffers (from the BMU), and for reading outgoing packets from IMEM/EMEM into the serial interfaces for transmission. It provides an automatic buffer release mechanism for transmitted frames.

Incoming packets pass an initial classification in the ICU (Initial Classification Unit) and are classified by class of service and other initial parsing results into a few main flows that do not have order dependency on each other. Each such flow may be mapped by the NDMA to an initial physical queue in the PMU. The NDMA allocates 256B buffers to store the incoming frame. When frame reception is complete, a Job ID is dynamically allocated by the PMU and stays associated with this frame as long as it is in processing stage.

The implementation is split into the receive path (RxNDMA block) which handles frames received from the network interfaces, and the transmit path (TxNDMA block) which handles frames to be transmitted out by the network interfaces.

The NDMA blocks use the Buffer Manager Unit block (BMU) for memory buffer management services, e.g. allocation and recycling of frame memory buffers.

3.5.1 Receive DMA (RxNDMA)

On the receive path (Rx), the NDMA receives a stream of frame data from the network interface units (IFU) and builds frames in NPS memory format (frame descriptors, link buffer descriptors, data buffers) in IMEM and/or EMEM. The NDMA passes the generated frame descriptors representing the frames to the PMU block.

The NDMA allocates data buffers and Link Buffer Descriptors (LBDs) from the BMU as needed and fills the data buffers.

For each frame, the NDMA is typically configured by the user to allocate the first frame data buffer and link buffer descriptor (if used) from IMEM, and the rest from EMEM. Based on Logical IF configuration and IMEM and EMEM resources availability, NDMA allocates a buffer from the EMEM.

3.5.2 Transmit DMA (TxNDMA)

On the transmit path (Tx), the NDMA receives frame descriptors from the PMU/TM destined for specific output ports. It reads the frame data from the (IMEM and/or EMEM, and passes the read data to the network interface unit (IFU) for transmission as a stream of data.

The transmit NDMA capable of handling any mixture of IMEM and EMEM frame data buffers. Any frame data buffer can have free bytes at the end. The first buffer usually has some Header offset .

Based on user configuration, the NDMA recycles any LBD buffers or frame buffers by passing the pool ID and the index to the BMU block. The TxNDMA can generate a **Transmit Confirmation Flag** back to SW by sending the FD back to the PMU.

The frame descriptor flag **Transmit Keep Buffers Flag** indicates a request to avoid freeing the frame buffers automatically by the NDMA hardware engine. It is, therefore, the software's responsibility to free the frame buffers when the frame has been sent. For this purpose, the flag is echoed on frame transmission confirmation responses.

3.6 PMU Overview

The PMU serves as a task scheduler for jobs. Jobs are packet processing tasks or SW initiated task.

A job is a processing element tracked by the PMU, which has unique Job Identifier (JID) that stays stable through the entire period it is being processed.

A new job is created for a received packet, for a TM looped back packet, or as initiated by SW. A job is ended (terminated) by forwarding outside of the PMU (to TM) or by SW initiated termination without forwarding.

Each job is described by a Job Descriptor (JD) stored in IMEM and indexed by a JID. Up to 32K Job Descriptors are managed by the PMUs. The *job* list can be further extended to DRAM stored queues of packet descriptors by the Traffic Manager for long term processing. TM long term queuing does not utilize PMU JID resources.

The PMU manages job queues (PQ-PMU queues). When required, the PMU schedules jobs according to a defined order relative to other queued jobs.

The PMU schedules jobs for SW processing (job dispatch) or forwards jobs for Traffic Manager.

The PMU may also schedule job for re-queue to itself or the other PMU unit.

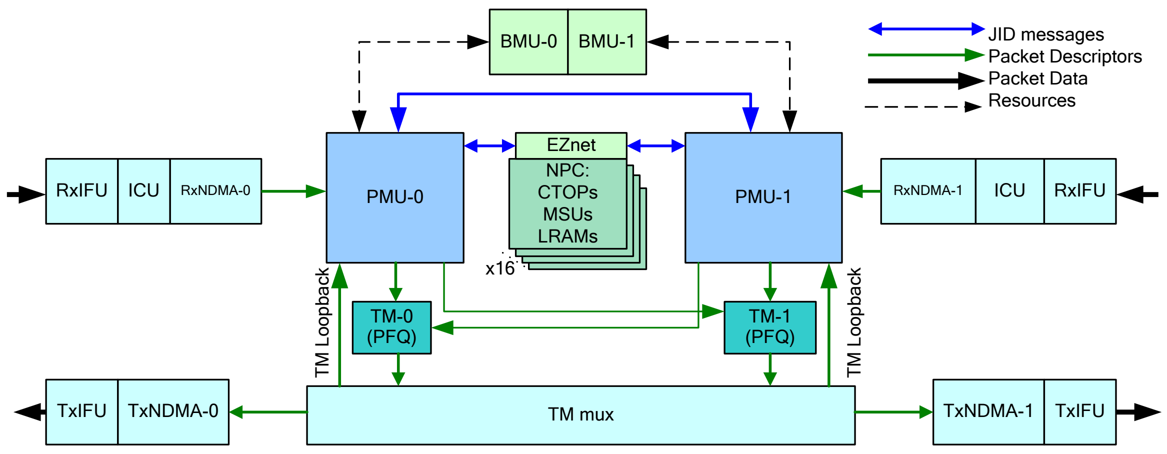
The PMU job dispatch for processing, distributes the jobs to CTOPs to achieve load balancing and utilization of resources. The PMU dispatch is application aware, assigning a job to designated pool of CTOP resources and accelerator resources.

The PMU scheduling is application SLA aware utilizing priority and WRR application arbitration scheduling. The PMU assigns jobs to the MSU which represents the entire NPC cluster, based on load cluster level metrics. The MSU spreads the jobs to threads in the cluster across 16 CTOP cores.

The PMU keeps track of job ordering during SW processing, scheduling to the TM and re-scheduling.

The PMU supports Job replication by SW or by the Traffic Manager loopback path via Multicast Replicator (MR).

Figure 3-4. East and west PMU modules



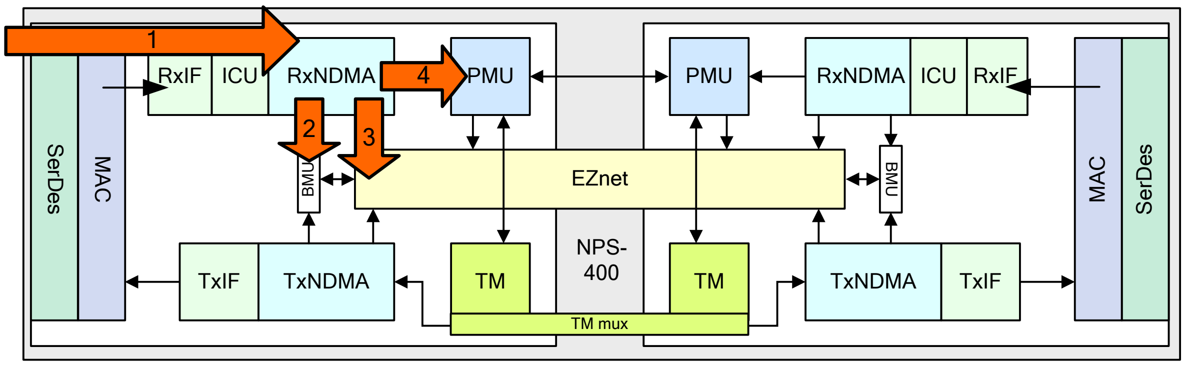
For more information see the *Processor Management Unit (PMU)* chapter.

3.7 Data Flow Overview

The sections that follow describe significant portions of the data flow through the NPS.

3.7.1 Frame Reception and Queuing in Memory

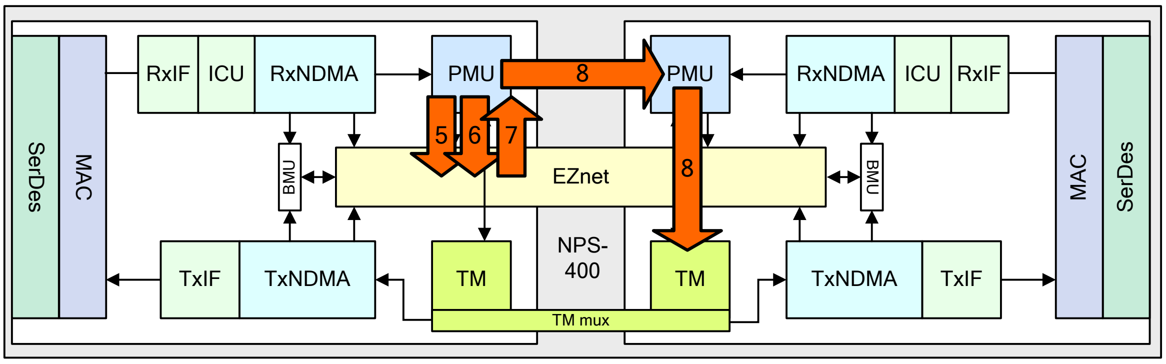
Figure 3-5. Data flow: frame reception and queuing in memory



1. The frame is received by the input MAC and forwarded to the RxNDMA.
2. The NDMA allocates buffers located in IMEM and EMEM from the BMU. The ICU selects a Physical Queue in the PMU.
3. The frame is stored in the memory.
4. The Frame Descriptor (FD) is passed to the PMU.

3.7.2 Frame Processing

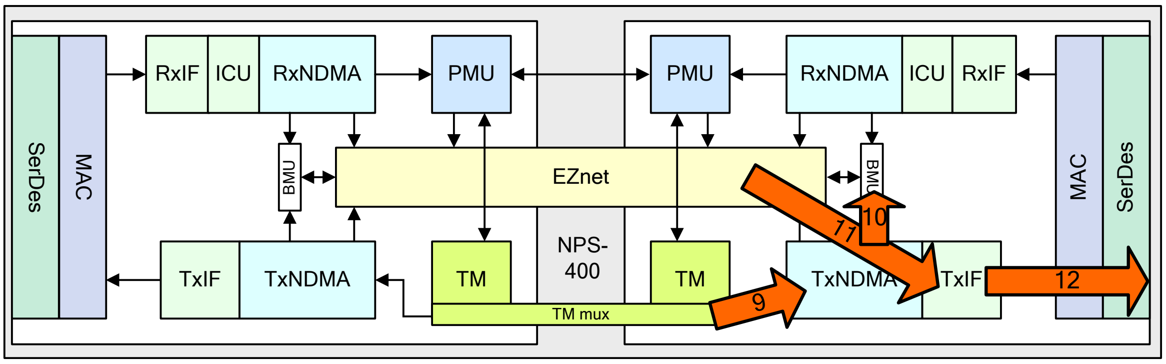
Figure 3-6. Data flow: frame processing



5. The PMU writes a job descriptor to IMEM (including FD and other fields).
6. The PMU sends the Job for processing by a specific NPC.
7. SW processes the frame and sends “Done + next destination” to the PMU.
8. The PMU restores frame order in its queue and forwards the frame to the next destination. (PMU for additional SW processing or TM for transmission.)

3.7.3 Frame Transmission

Figure 3-7. Data flow: frame transmission



9. The TM schedules the frame for transmission and delivers the Frame Descriptor to TxNDMA.
10. The TxNDMA reads the frame from memory and writes the transmit data to the TxIF.
11. The TxNDMA releases buffers to the BMU.
12. The frame is transmitted.

3.8 Unicast Flow Processing Example

The frame is received by the MAC, forwarded to the RxIF block and to the RxNDMA. The RxNDMA stores the frame in IMEM and EMEM buffers, according to its configured settings (header offset, number of buffers per frame stored in IMEM, etc.)

The RxNDMA passes the Frame Descriptor to the PMU Physical Queue (PQ) based on the configuration of the QoS result of the ICU (Input Classification Unit), the PMU budget and the load-balancing function results in the ICU. The RxNDMA decrements the relevant budgets in the BMU/FCU.

The PMU allocates a 32-byte buffer (JD) from the BMU and writes the Job Descriptor into the IMEM. Then the PMU then pushes the JID into to the selected PQ in order.

The PMU schedules the JID to CTOP/MTM for SW for processing of the frame. SW selects the “next destination” and updates the JD (if needed). For example, the next destination could be a PQ on the other PMU. The “Done message” is sent to the PMU indicating the “Next destination”.

The PMU restores order and forwards the JID to the relevant PMU. The JD is pushed to the new PQ in order. The job gets scheduled, dispatched and goes through the SW process. The SW selects the TM as the next destination, writes TM parameters to the Job Descriptor. SW allocates EMEM buffers, and activates the Cluster DMA to copy the IMEM buffers to EMEM buffers. SW updates the frame descriptor.

A “Done message” is sent to the PMU. The PMU restores order and reads the Job Descriptor from IMEM and pushes the FD + TM parameters to TM. The PMU releases the Job Descriptor buffer and decrements relevant PMU budget counters.

The TM schedules the frame for transmission. The TxNDMA reads the data and transmits the frame.

3.9 Multicast and Replication Processing Overview

A common requirement of many network processing applications is the necessity to send copies of a frame to multiple destinations.

NPS provides the ability to share frame buffers by multiple frame replicas and supports two methods for multicasting: lightweight multicast (SW based) and heavyweight multicast (TM loopback based).

- Lightweight multicast
  - Can be used to create up to seven replicas of the frame. In this method the SW is responsible for creating a batch of jobs (packed into a Job Container) and sending the Job Container to the PMU. The PMU unpacks the Job Container and creates a Job per replica.
    - SW identifies lightweight multicast and immediately replicates the frame.
    - SW needs to allocate a new Job ID per replica.

  For a detailed description see the "Lightweight Multicast Processing" section below.
- Heavyweight multicast
  - Allows sending a frame to one of the TM loopback interface channels and requesting that a designated number of frame replicas, or copies, be looped back by the TM into the PMU queues.
  - The replication is logical (i.e. nonphysical) – all the returned frame replicas reference the same frame buffers, which are not copied.
    - SW identifies heavyweight multicast option.
    - The Frame Descriptor is sent to the TM. The TM manages the rate of the replication process.
    - The TM schedules frame to the PMU.
    - The PMU replication unit replicates the FD with a unique replication ID and pushes the frame to the Physical Queue (PQ).

  For a detailed description, see the "Heavyweight Multicast Processing" section below.

In addition, the NPS provides two features in support of this requirement: multicast reference counters and TM loopback frame replication.

For additional information refer to the *NPS-400 Programming Manual*.

3.9.1 Multicast Reference Counters

Normally, a frame buffer is only referenced by a single frame at a time. This statement ceases to be true, however, when dealing with multicast frames. Multicast frames are frames that need to be replicated to service a network broadcast or multicast service. Often, the different replicas of the frame have the same data with the exception of a few header fields.

In order to support multicast frames in an efficient manner, without requiring the creation of a separate copy of all the buffers for each frame replica, the NPS supports multicast reference counters.

Each NPS frame buffer has a reference counter associated with it. When frames are transmitted or discarded, the network DMA engine atomically checks and decrements the multicast reference counters of each frame buffer that may be in use by more than one frame. These buffers will only be released when that multicast reference counter indicates that the frame buffer is no longer used by any frame.

As most frames are not multicast frames, the NPS provides the option of marking frames as multicast frames and, therefore, having their frame buffer multicast reference counters checked and updated via the frame multicast control status.

3.9.2 Multicast Buffer Structure

Each 256B buffer has a pre-allocated multicast counter (MC counter) that indicates the number of replications expected from this buffer before it will get released.

Multiple frame descriptors can point to the same buffer.

The MC counter should be initialized with the number of expected copies. The release of the buffer is performed by the TxNDMA which decrements the MC counter for each replica. The buffer is released to the BMU once the counter reaches zero.

3.9.3 Lightweight Multicast Processing

Provided below are the steps to be performed to implement lightweight multicast processing:

1. SW gets the Job to process.
2. SW identifies that the application frame flow requires a lightweight multicast replication.
3. SW allocates resources for each replication:
   - 3.1. Job descriptor buffer
   - 3.2. PMU budget
   - 3.3. BMU buffers (buffers not shared by replicas)
4. SW creates multiple copies of the frame:
   - 4.1. Creates a physical copy of the first buffer(s) of each frame.
   - 4.2. Shared buffers: Set Multicast counter of the buffers(s) including the first buffer.
5. SW modifies the JD (32 bytes) to be a multicast container that points to all JDs. (MC containers can be linked.)
6. SW sends a “done” indication” for each Job completed (each replicated frame).
7. The PMU sends the frame to the TM or back to the PMU for more SW processing and then to the TxNDMA.

3.9.4 Heavyweight Multicast Processing

Provided below are the steps to be performed to implement heavyweight multicast processing:

1. SW gets the Job to process.
2. SW identifies that this case requires heavyweight multicast replication.
3. SW sets the Multicast bit in the frame descriptor and sets the MC replication number in the MC counter in all the buffers (or in the JD).
4. SW sends a “done” indication with the next destination as TM loopback.
5. The PMU sends the frame to the TM in order.
6. The TM sends the frame to the PMU to perform replication in HW.
7. PMU replication:
   - 7.1. Reads the replication number from the first buffer’s MC counter.
   - 7.2. Replicates the FD according to the replica number, and attaches a unique replication ID to each Frame Descriptor.
   - 7.3. Pushes the FD to the PMU as new job (allocate JID, write to IMEM).
   - 7.4. Budget based backpressure is implemented to the replication unit to prevent flooding.
8. Each Job is dispatched to SW for processing.

9. SW processes each replication based on its “replication ID”. SW needs to allocate a new IMEM buffer and copy each buffer it modifies (typically the first buffer) and decrement the MC counter.
10. SW sends a “done” indication” for each job completed.
11. The PMU sends the frame to the TM and then to TxNDMA (similar to the lightweight multicast).

4. Network Processor Cluster (NPC)

This section is intended to provide a high-level architectural overview of the Network Processor Clusters (NPC) inside the NPS-400.

4.1 NPC Overview

The NPCs are the heart of the NPS-400's packet processing capabilities. The 16 NPCs interconnect through the EZnet fabric. Each NPC serves 16 CTOPs. Each CTOP is equipped with a Multi Thread Manager (MTM) for scheduling the 16 threads for each CTOP.

The NPC hosts 1MB local RAM that is a portion of the NPS-400 IMEM memory region.

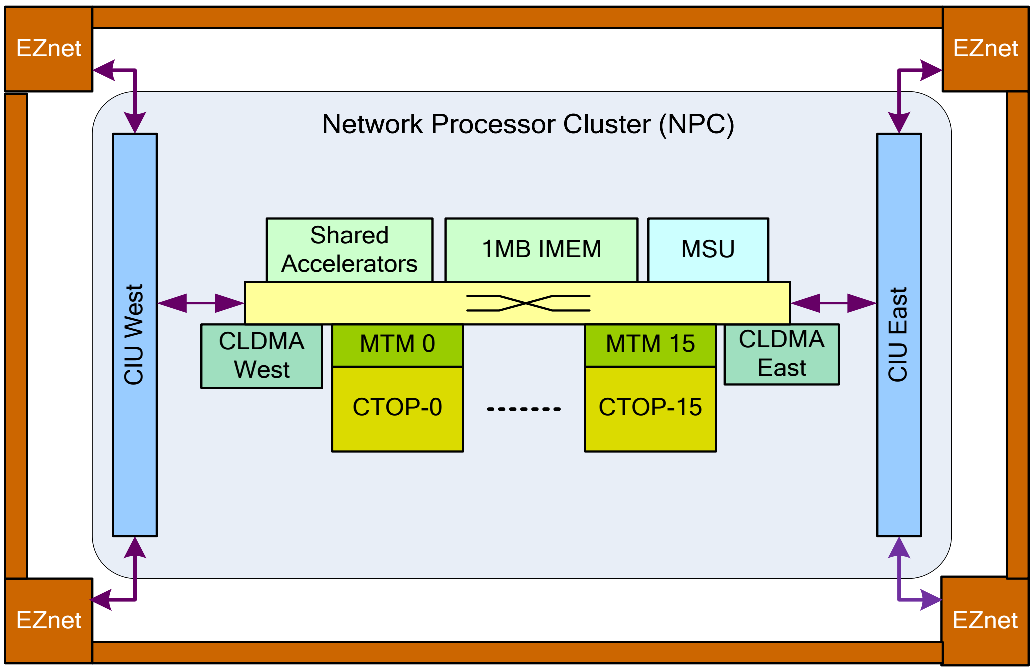
The NPC is served by a shared Messaging and Scheduling Unit (MSU) that is responsible for system message communication (i.e. job dispatch and inter-processor interrupts/messages).

The CTOPs utilize shared HW accelerator engines and Cluster DMA units.

All of NPS' functional elements are interconnected through a local fabric enabling any element to access any other. The two Cluster Interconnect Units (CIU) connect the NPC to EZnet system level interconnect.

The figure below shows a block diagram of a Network Processor Cluster (NPC):

Figure 4-1. Network Processor Cluster (NPC) block diagram

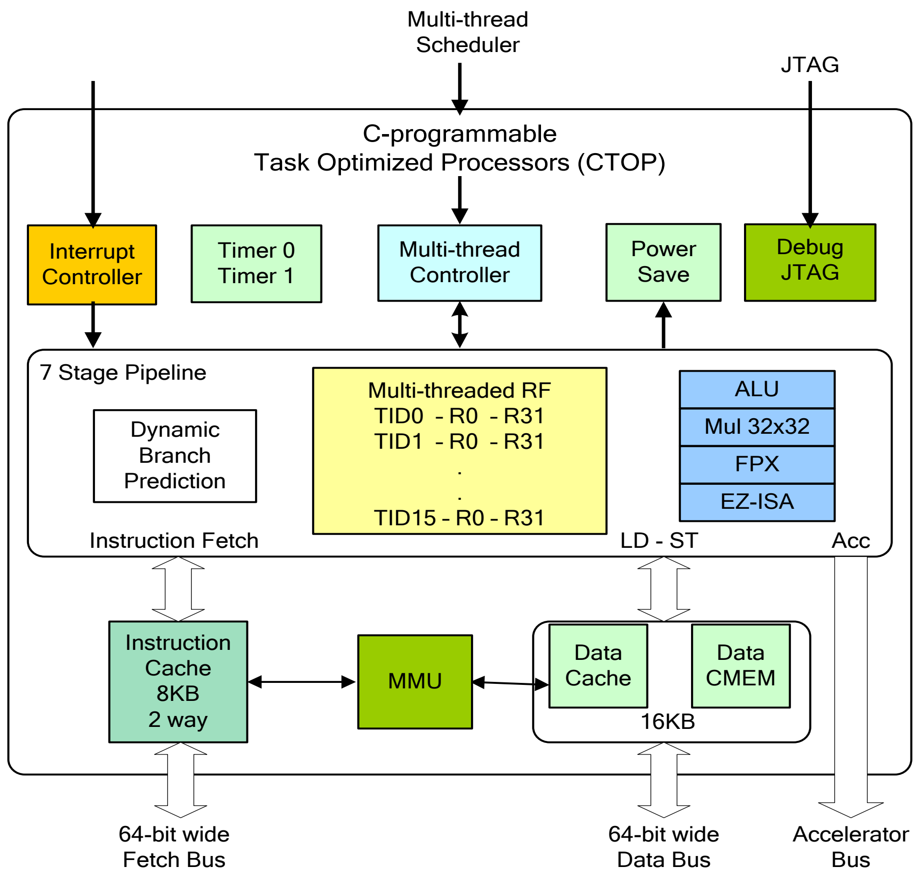


4.2 C-programmable Task Optimized Processors (CTOP)

A CTOP is a 32-bit RISC processor implementing ARC™ ISA with extended support for large scale multithreading and EZ-ISA extensions for EZchip's proprietary packet-optimized processing instructions. CTOPs provide a C-programming environment under an operating system memory management while retaining the strong network programming capabilities of the network Task Optimized Processors available in EZchip's NP line of network processors.

The CTOPs offer a 7-stage execution pipeline with dynamic branch prediction. In order to support the 16 HW threads, 16 sets of register files are implemented as well as associated processor state context. Each thread is logically viewed as a virtual core. Some of the HW accelerators are built into the CTOP core as part of the EZ-ISA and others are provided through a shared accelerator block accessed by a dedicated Accelerator Bus. The Interrupt controller provides two levels of interrupts (high and low) and two dedicated timers. The CTOP Memory Management Unit (MMU) provides virtual memory addressing, memory protection and cache control.

Figure 4-2. CTOP block diagram



The CTOP has the following features:

- High performance, non-blocking, 7-stage pipeline
- 1.5 DMIPS/MHz for a single thread. 1GHz core clock speed.
- Multi-threaded design with 16 HW threads with a dedicated register file and processor state context per thread.
- ARCompact ISA compatible freely mixing 16/32 bit instructions. Using 32-bit long immediate data (LIMM) also provides 48-bit and 64-bit instructions.
- The CTOP Floating Point Acceleration (FPX) supports IEEE-compliant instructions and library functions for high performance single-precision floating point operations.
- Networking ISA extensions
  - CRC, hash, fast checksum instructions
  - Bit manipulation instructions
  - Fast classification
  - Messaging
  - Buffer/index allocation
  - Search/lookup macros
  - DMA controls
  - Wide load/stores
- Deep Packet Inspection ISA extensions for intrusion detection and prevention networking applications.
- Networking extensions by shared accelerators
  - Security block ciphers
  - Search functions
  - Protocol decoding
- Out of order load/store interface
- Out of order variable delay accelerator activation interface
- Up to four open variable non-posted delay transactions (load/store or accelerators) per thread
- Misaligned load/store operations
- 8KB 2-way instruction cache with 32B cache line shared by all the threads
- Up to 16KB 2-way Data Cache (shared memory with CPMEM/CMEM); either shared by all threads or partitioned for 1/2/4/8/16 threads
- Configurable size thread private Scratchpad Memory (CPMEM) per thread and core shared Scratchpad Memory (CMEM)
- Lightweight MMU support combined with Fixed Mapping Table:
  - Enabling Linux operating system to run natively on each thread
  - Combining data plane applications using Fixed Mapping Table with Linux operating system services and debug using the TLB based MMU
  - Four entry instruction micro-TLB (fully associative)
  - Eight entry data micro-TLB (fully associative)
  - Joint Translation Lookaside Buffer (JTLB) x2 way with 256 entries.
  - Dynamic branch prediction
    - Branch History Table
    - Subroutine call return detection

4.3 Messaging and Scheduling Unit (MSU)

The MSU provides low latency and high throughput message management for the CTOPs clusters.

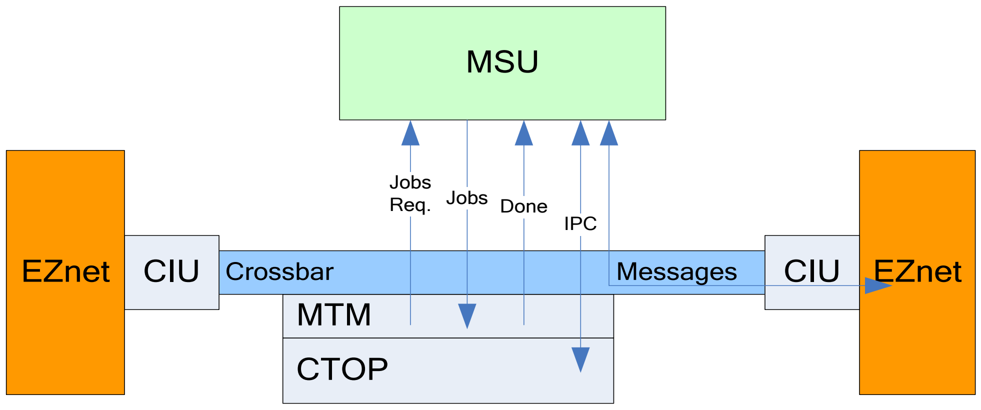
The messages are exchanged over the EZnet messaging interface. Interrupts, processor to processor communication and accelerator related commands are all passed as messages.

The CTOP communicates via messages with list of functional units:

FUNCTIONAL UNIT	***MESSAGING PUPOSE***
PMU	Jobs distribution and management: MSU sends: ▪ Job request ▪ Threads status ▪ Job done, Job terminate, Job forward MSU gets from PMU: ▪ Job dispatch
BMU	Buffer management: MSU sends: ▪ Packet internal/external buffer pre-fetch request. ▪ Packet internal/external buffer release. ▪ Allocate/release buffers from 64 generic pools.
CTOPs of other clusters	Inter process interrupt and messaging. Sent/received interrupt messages from Global Interrupt Manager to/from CTOPs
TCAM search	Lookup key and result transactions

The message exchange is managed by SoC end-to-end flow control.

Figure 4-3. CTOP messaging and scheduling



The MSU interfaces with the cluster MTM for job distribution, job status update and job termination.

The MSU interfaces with the CTOPs cluster for BMU buffer allocation and release services and inter-process communications and interrupts.

4.3.1 PMU Job Management

The MSU in each NPC communicates with the two PMUs in order to distribute the jobs to the NPCs and CTOPs.

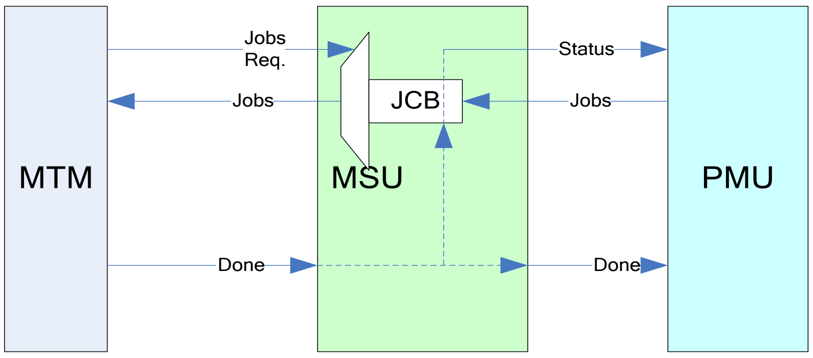
The MSU is responsible for job distribution between threads inside a cluster with load balance considerations. Each job can be distributed for processing to 1 of 256 threads (16 threads x 16 CTOPs).

The MSU to PMU messages are:

- Job request
- Threads status
- Job done

The MSU gets from PMU messages:

- Job dispatch



4.3.1.1 Job Management

The jobs from the PMU arrive in job containers, with up to four jobs per container.

The MSU temporarily aggregates PMU dispatched jobs in a buffer (JCB), buffering 8 jobs for each PMU.

The MSU manages job statistics:

- Free jobs – The number of free entries in the JCB to receive job-IDs from the PMU.
- Reserved jobs – Threads Status Message has been sent to PMU and there are entries reserved in the JCB for job-IDs that need to arrive.
- Busy jobs – Indicates the number of occupied entries in the JCB.
- Jobs being processed – Jobs forwarded to CTOP and not yet completed processing.

When a Job Dispatch Message arrives from the PMU containing N job-IDs (4 or less) then the Free and Reserved Jobs will decreased by N and the Busy Jobs (JCB occupied) will increased by N.

Job Request messaging to PMUs is based on availability of buffers and threads.

The MSU packages up to four MTM Job Done indications into a single Job-Done message.

The message is sent as soon as four free places are accumulated in JCB or at time-out since reception of first indication from MTM. Along with the job request message, the MTM sends updated load statuses to the PMU schedulers.

Configuration:

- Period for the time-out timers of the PMU Threads Status and Job Done messages.

4.3.1.2 Job Distribution

The MSU manages a CTOP load balancing table. The jobs are distributed according to CTOP's thread status in the load balancing table. The least busy CTOP will be assigned to a new job. The new job is assigned to a CTOP with the maximum available threads. The CTOP thread status indicates the number of available threads. The MSU maintains 16 available-thread counters, one per CTOP.

Arbitration among MSU jobs awaiting dispatch to MTM is according to:

- CoS
- 4 types of jobs

The MSU gets a job request from the MTM and responds with a job dispatch to the MTM with Job-ID.

The MSU gets a job done indication from the MTM and generates a job done message to the PMU.

Configuration:

- Enable/disable per PMU for job container transfer. Either PMU, neither PMU, or just one PMU can be enabled.
- Enable/disable per thread to participate in the job dispatch targets.

4.3.1.3 Threads and Jobs Status

The MSU sends Thread Status to PMUs. The status message conveys CTOP cluster's available threads.

This status messages enable the PMUs to control cluster level job distribution balancing.

The time interval between status messages is configurable to allow message rate control.

The various statuses managed by the MSU:

- Threads status (per 256 threads) of load metrics. The metrics measure threads' status (Not assigned, Busy, Waiting for new job).
  - Configure IPC to generate an interrupt (IPI) or work in user mode (IPC).
- Cluster status: Number of available (not assigned) threads in the cluster 0-256.
- Processed jobs status: Per PMU number of jobs being processed.
- Pending jobs status: Number of pending jobs in MSU (max. 16) waiting to be dispatched to the MTM.
- Adding/removing jobs from PMU job budgets.

4.3.2 BMU Management and Messages

The MSU converts MTM requests from the BMU to messages to the BMU.

The MSU converts BMU messages to indications to the MTM.

Buffer pre-fetching of 8 internal/external buffer indexes from BMU to speed up buffer allocations by SW.

Allocating/releasing software generic indexes from up to 64 BMU SW buffer pools.

Accumulation of up to 4 BMU addressed buffer releases to reduce message traffic.

Configurations

- Time-out timer value of the BMU messages.

4.3.3 Processor Communication/Interrupts

Managing IPC (Inter Processor Communication) and IPI (Inter Processor Interrupts) for 256 NPC local threads and threads on other NPCs. Driving IPI enable to threads according to incoming IPI messages, and transmitting IPI messages to other threads according to thread requests.

- IPI (Inter Processor Interrupt) – Transfers incoming and outgoing interrupts to and from the cores.
- IPC (Inter Processor Communication) – Transfers incoming and outgoing messages to and from the cores. Either in user mode (no interrupt generation) or kernel mode (operating as IPI).

IPI message generation request is driven by the MTMs. The request indicates a single/unicast message or multicast messages. Single (unicast) messages are destined to addressed CTOP/threads. Multicast messages can be replicated to all other 15 clusters and to the 256 threads of each cluster.

The multicast groups are:

- The 16 threads of the target CTOP.
- The 256 threads of the target cluster.
- All 4K threads of all CTOPs.

Configurations

- Enable/disable general cluster IPI.
- Configure IPC to generate an interrupt (IPI) or work in user mode (IPC).

4.4 Multi Thread Manager (MTM)

The CTOP implements a coarse-grained, non-interleaved, multi-thread methodology that allows the CTOP to time share a single CTOP execution unit with up to 16 hardware threads.

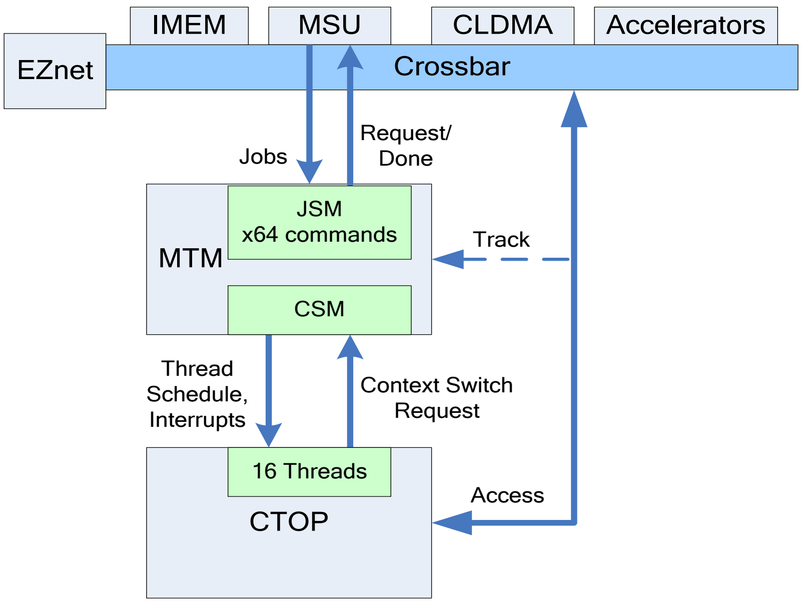
The Multi Thread Manager (MTM) is the CTOP HW threads scheduler. Multi-thread programs tolerate memory and accelerator latency by overlapping the long latency access or accelerator activation delays of one thread with the execution of the other threads while they wait. Overall, it maximizes the utilization of the CTOP execution unit.

Each MTM works closely with a single CPU. The MTM enables the CPU to work on many threads with only partial awareness of its multi-threaded nature.

The MTM is responsible for handling the following operations:

- Dispatch new jobs from the MSU to the CTOP.
- Issue the 'job done' response from the CPU to the MSU and then back to the PMU.
- Track eligibility and enabling status of the running jobs/threads.
- Track up to 64 commands that await responses (new job, load/store, DMA transaction, accelerators, etc.) up to 4 commands per thread. Posted store instructions do require MTM tracking and do not stall execution.
- Issue interrupts and messages to the CPU.

Figure 4-4. Multi Thread Manager (MTM) flow



4.4.1 MTM Functions

- Job Status Manager (JSM)
    - Holds status information for 64 entries (pending, new, execute, tag) in scoreboard
    - Holds tracking of transactions (load, store, DMA…), up to 4 transactions per thread
- Context Switch Manager (CSM)
    - Create eligibility list according to scoreboard info.
    - Select job to be dispatched according to round robin scheduling algorithm
    - Receives context switch request from the CTOP to switch from “hot” to “warm” thread
    - Swap warm thread with one of the eligible cold threads.
    - Update job status (scoreboard) when selected for eligibility
- Interrupt Controller (IC)
    - Interrupts originate from MSU (messages)
    - Clears/sets timer status bit according to timer indication
    - Set IPI/IPC status bits according to MSU indication
    - Dispatch an interrupt to the "hot" thread at the CTOP or advance a cold thread which got an interrupt to be serviced by the core.
    - Get thread interrupt mask from the CTOP

MTM Transaction Scoreboard

The MTM keeps track of 64 CTOP pending transactions/commands. The transactions’ status is marked in a scoreboard. Up to four pending Load transactions are allowed per 16 threads. Up to a total of 64 Load/Store pending transactions are allowed for all threads.

The transaction status can be:

- Invalid (free to allocate)
- Valid – pending transaction. The transaction may be blocking or non-blocking as determined by CTOP execution control and scheduling instructions.
- Valid – ready (transaction completed).

The thread status can be:

- Disabled (after reset, a single thread is enabled and held at halt)
- Halt – The thread awaits boot-loader initialization
- Cold – Not eligible for scheduling because not all pending blocking transactions are ready
- Cold awaits trigger (or new Job) – The only pending command can be new Job request. Thread interrupt activation is enabled
- Cold – Eligible for scheduling because no valid blocking transactions are pending
- Warm – Next thread to execute
- Hot – running. In parallel some transactions could be pending and blocking the thread’s next scheduled event
- Hot (pipeline stalled) – Some pending transactions are blocking.

The MTM is responsible for switching the threads to disabled/halt/cold/warm state. The CTOP switches the thread from “warm” to “hot” and from “hot” to “cold”.

The MTM selects next the “warm” thread from all the “cold” eligible threads based on a round robin algorithm. The MTM tracks the order of thread execution according to thread suspended from “hot” state by the CTOP. A cold thread which gets an interrupt pending, gets an upgraded priority in MTM scheduling to minimize its service latency.

All pending transactions are bound to terminate in a due time interval. The only commands that have unbounded wait time are the new Job request and the wait for IPC.

4.4.2 Process Flow

The MTM enables CTOP thread execution. The CTOP active flow requests a new job from the MTM `(ezdp_receive_job(&job, &job_desc))`. Usually the job request is followed by a 'Schedule wait for trigger' command that causes the thread to suspend. The request contains the job handle `( &job)`;

The MTM allocates a free entry in the active transaction scoreboard for the new job request.

The MTM sends the new job request to MSU. The MSU responds to the new job request with a Job-ID.

The MTM stores the Job-ID in the CPMEM scratchpad address `(&job)`.

The MTM marks the thread as eligible for scheduling in the scoreboard – "cold".

The CTOP requests thread scheduling.

The MTM schedules the thread as "warm".

The CTOP switches to execute the thread.

When the CTOP accesses a remote resource (IMEM, EMEM) or activates a HW accelerator or CLDMA, the CTOP performs thread Wait and Schedule instruction (schd.*). The MTM suspends the thread to "cold" state until all pending transactions/commands are terminated.

4.4.3 CTOP Thread Scheduling Instructions

The table below lists CTOP instructions controlling thread scheduling and thread execution.

Table 4-1. CTOP instructions controlling thread scheduling and thread execution

INSTRUCTION	*DESCRIPTION*
schd.rw	Schedule Sync Read/Writes. Perform a schedule operation. Thread becomes eligible for execution by the threads scheduler after all pending read/write transactions are completed. Posted store instructions need not be completed in order for the thread to become eligible.
schd.wft	Schedule Wait For Trigger. Perform a schedule operation. Thread becomes eligible for execution by the threads scheduler after the trigger has arrived. Trigger is defined as an interrupt or a new job request response.
schd.rd	Schedule Sync Reads. Perform a schedule operation. Thread becomes eligible for execution by the threads scheduler after all pending read transactions are completed. Posted store instructions need not be completed in order for the thread to become eligible.
hwschd.off hwschd.restore	HW schedule mechanism control instruction. Used to enable/disable HW scheduling. HW thread schedule is derived from a HW based event. When the HW event occurs, the CPU will schedule the thread and perform context switch (depends of the state of internal hwschd bit). The programmer can control whether the HW schedule will take place or not, using the *hwschd* instruction which affects internal hwschd status bit. HW Schedule Event Off - HW schedule event suppressed. State of hwschd bit is written to destination General Purpose Register. HW Schedule Restore state - State of hwschd bit is restored according to the value at source GPR.
sync.rd	Pipeline control. Wait for all write or read data memory (including accelerator read) transactions to complete. Posted writes are not tracked to be completed.
sync.wr	Pipeline control. Wait for all write and read data memory transactions (including accelerator write operations) to complete. Posted writes are not tracked to be completed

4.4.4 Interrupt Servicing

The MTM presents to the CTOP up to three pending interrupts per thread. The CTOP interrupt mask register is used to mask out the interrupts.

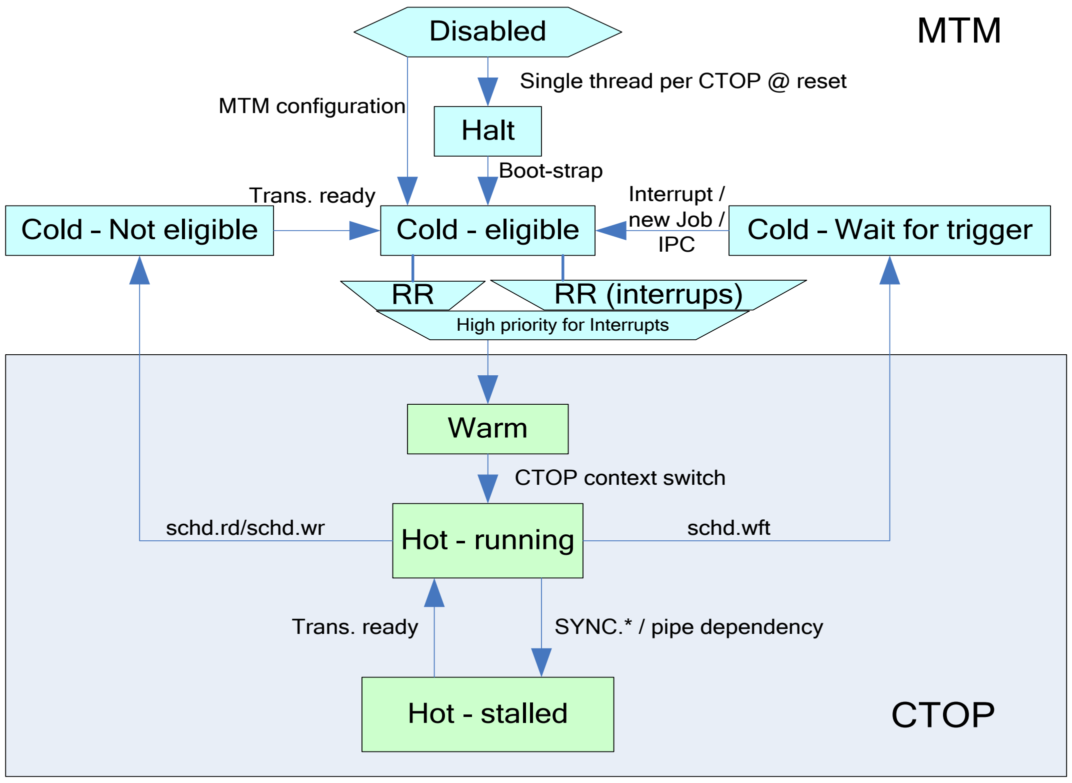
Any enabled thread can get an interrupt pending. A cold thread that gets an interrupt has to wait until all its pending transactions are completed and it is eligible for execution. Cold threads become eligible to service the interrupt only after completion of their time-bonded events.

A cold thread that waits for a trigger is not time bounded.

The “cold wait for trigger” thread is scheduled as “warm” with Job valid/invalid indication. Upon interrupt, the SW checks the Job validity to resolve the reason the thread was awakened – interrupt or pending job. Having a pending interrupt upgrades the cold thread’s priority in MTM schedulers.

The figure below shows the flow of thread states.

Figure 4-5 .Thread state flow chart



4.5 Cluster Interface Unit (CIU)

The CIU deals with incoming requests and outgoing responses from the EZnet to/from the IMEM portion located in the NPC:

- Converts EZnet incoming burst transactions to a single transaction towards the local NPC IMEM banks.
- Selects bank according to load-balancing hash function to get uniform distribution across banks.

The CIU deals with incoming requests and outgoing responses from the NPC (MTM & CLDMA) to the EZnet:

- Performs address translation according to FMT/MSID tables.
- Performs write protection for erroneous addresses.
- Memory region checks and access privilege enforcement.
- Receives and transmits messages between the EZnet and the MSU.
- Connects MSU to the EZnet messaging network (MSN).

4.6 Cluster Level DMA (CLDMA)

There are two CLDMAs in each cluster. The Cluster DMA provides data transaction services to all NPC threads (each CLDMA serves 8 CTOPs' 128 threads). After activating the DMA transaction, the CTOP thread can be suspended until the transaction terminates and awakened by the MTM. The MTM schedules other threads while the suspended thread waits for CLDMA termination.

The CLDMA supports data copy between the following memories:

- CTOP CMEM and CPMEM regions.
- Own sub-cluster's IMEM portion
- Local cluster IMEM portion (comprising two sub-clusters)
- Multiple clusters shared IMEM regions (including own cluster portions)
- EMEM memory spaces

The CLDMA enable any alignment on source data and any alignment on destination data as well as byte resolution transaction length. Frame data structure may be spread in EMEM or in IMEM in a multi-buffer format. The thread SW tracks the frame buffers and operates CLDMA as required by each buffer.

Using its threads it can fetch frame header data as well as data to feed shared accelerators into the thread's CPMEM/CMEM scratchpad, while selecting convenient data alignment for the required service.

The CLDMA can also accelerate some bump-on-the-wire operations such as checksum and CRC calculations.

- Each CLDMA can accept up to 256 DMA commands from 128 HW threads.
- CLDMA transfer command data copy size may be any value from 1 byte to 64KB.
- Slices the transfer to multiple transactions of a size (up to 128B) that is adequate and optimized to the source/target. Manages up to 128 outstanding transactions. Multiple streams long term transactions are scheduled with burst interleaving.
- Manages transfer data alignment according to the CPU read/write commands.

4.7 Shared Accelerators

The HW accelerators are located within the CTOP cluster and are accessed via EZ-ISA extensions to the instruction set. For more information on the ISA refer to the *CTOP Programmer's Manual*.

The HW accelerators supported are:

- Protocol decoder for packet header parsing
- Network information database structures lookup – refer to the *Look-up Acceleration* chapter.
- Wide hash function calculation – refer to the *Look-up Acceleration* chapter.
- Cryptographic engines – refer to the *Cryptographic Features* chapter.

4.7.1 Protocol Decoder

The CTOP cluster contains a shared protocol decoder accelerator. The MTM manages the accelerator sharing among the CTOP cores. The CTOP supports a set of dedicated extended commands to operate the accelerator. The decoder parses the packet header and replies with identified error codes and decoded network protocol fields. The packet header for decoding is prepared in the CTOP core scratchpad memory (CMEM). The pointer to the packet in CMEM and header size and packet size are passed as command parameters via CTOP registers. An additional command parameter is a CMEM pointer for command return values. The decoder command execution returns the decoded error codes and network protocols fields in CMEM. The return information can range from 4B to 12B. The returned 4B containing decoded errors is also returned in CTOP register.

The decoder accelerator performance is decoding of 8B per core clock. The command parameters passed to the decoder consume two core clocks. Thus, the core clock cycles for decode are:

2 + [header size bytes / 8].

While decode command is executed, the CTOP thread is suspended.

Table 4-2. Summary of accelerator decoded protocols

PROTOCOL	COMMAND	DECODE	ERRORS	FEILDS DECODED
Ethernet Layer 2 - MAC	dcmac	Decode SA, DA and Ethertype. DA compares	MAC is 0, DA=SA.	Ethertype protocol Hash (DA, SA), L2 size
IPv4	dcipv4	IP header DIP compare, MC SIP compare Next protocol, Control	Header length, Total length, Version, Checksum, SIP, SIP=DIP	DIP compare to ranges Next protocol
IPv6	dcipv6			
MPLS 4 Labels	dcmplsl	Decode up to 4 labels	Labels TTL value.	
MPLS Label	dcsmpls	Decode single label	Label TTL and value	EOS, Exception, Stop bit

The TCP, UDP and GRE decoders have dedicated instructions but do not use shares accelerator.

The command format is similar to all decode commands:

Command DST,[cm1: SRC1], [cm2: SRC1], [SRC2], size

DST – Register to hold first word of command return value.

SRC1 – Register that packs two pointers to CMEM. Alternatively each can be pointer immediate value:

[cm1: SRC1] - CMEM destination for command return value of up to 16B.
[cm2: SRC1] - CMEM address for packet header to be decoded.

SRC2 – (optional) Packet total length for protocol field validation.

Size – Packet header size in CMEM to decode; can be passed as immediate value or register.

4.7.1.1 Ethernet Layer 2 MAC Decoder

The decoder decodes the Ethernet Layer 2 format. The decoder parses MAC DA, SA and Ethertype. The decoder sets report bit fields per validity check. The header pointer should point to DA.

Table 4-3. Ethernet layer 2 MAC decoder

DECODE TYPE	***DECODE RESULT FIELD***	***BITS***	***DESCRIPTION***
IEEE Std 802.1D and IEEE Std 802.1Q Reserved addresses	01-80-C2-00-00-0x	1	IEEE Std 802.1D and IEEE Std 802.1Q reserved addresses
	01-80-C2-00-00-1x	1	IEEE Std 802.11aa/802.1B/802.1D/ 802.1E/1905.1. ISO/IEC 10589/10030/9542
	01-80-C2-00-00-2x	1	Multiple Registration Protocol (MRP) applications
	01-80-C2-xx-xx-xx	1	
VRRP	00-00-5E-00-01-xx	1	Virtual Router Redundancy Protocol
Multicast Group Destination Address (GDA)	01-00-5E-xx-xx-xx	1	Group Destination Address (GDA) for mapping between Ipv4 address and MAC address.
	33-33-5E-xx-xx-xx	1	Group Destination Address (GDA) for IPv6 Multicast addresses
MAC DA range	MAC DA-0	1	Configured DA with bit Mask
	MAC DA-1	1	Configured DA with bit Mask
	MAC DA-2	1	Configured DA with bit Mask
	MAC DA-3	1	Configured DA with bit Mask
SA MAC Error	SA MAC not Unicast	1	Set when SA MAC is not unicast
	00-00-00-00-00-00	1	Set when SMAC is 00-00-00-00-00-00
SA and DA error	SA = DA	1	
PPPoE validity	IP version error	1	IP version error in PPPoE. Set when Ethertype is PPPoE (0x8864) and the PPP protocol ID is IPv4 (0x0021) but the version in IP header is not 4. Set when Etherype is PPPoE (0x8864) and the PPP protocol ID is IPv6 (0x0057) but the version in IP header is not 6.
	IPv4 in PPPoE/PPP	1	PPPoE/PPP protocol ID is IPv4 (0x0021)
	IPv6 in PPPoE/PPP	1	PPPoE/PPP protocol ID is IPv6 (0x0057)
Ethertype compare	Number of VLAN Tags	3	Number of nested VLAN Tags per (3) configured VLAN_Ethertype values
	Ethertype 1	8	Enumerate compare of Ethertype (16 bit): 0x0 - 0x0800 (IPv4) 0x1 - Configured VLAN_Ethertype_0 0x2 - Configured VLAN_Ethertype_1 0x3 - 0x0806 (ARP) 0x4 - 0x8847 (MPLS) 0x5 - 0x8848 (MPLS) 0x6 - 0x86DD (IPv6) 0x7 - Less or equal to 0x0600 0x8 - Configured Ethertype_0 0x9 - Configured Ethertype_1 0xA - Configured Ethertype_2 0xB - Configured Ethertype_3 0xC - 0x8864 (PPPoE Session Stage) 0xD - 0x8863 (PPPoE Discovery stage) 0xE - Configured VLAN Ethertype 2 0xF - Other
	Ethertype 2	8	
	Ethertype 3	8	
	Last Ethertype (next protocol)	8	
Hash	Hash(DA,SA)	16	Used for L2 based load balancing
L2 size	Layer 2 packet size	8	L3 offset from DA after skipping all L2 fields.

4.7.1.2 IPv4 Protocol Decoder

The IPv4 decoder decodes IPv4 header format. The decoder sets report bit fields per validity check. The header pointer should point to IPv4 header. The protocol decode size should be 20B.

Table 4-4. IPv4 protocol decoder

DECODE TYPE	***DECODE RESULT FIELD***	***BITS***	***DESCRIPTION***
Header	Version error	1	Incorrect version (Header Version[3:0] is not equal to 4)
	Header length > 5	1	Compares Header length field. If IPv4 header length > 5, i.e. the header length is greater than 20 bytes.
	Header length < 5	1	
	Header length > frame	1	Compare header length (IHL*4B) field to (frame length -L2 header)
	IPv4 total length > frame	1	Header Total Length > (frame length - L2 header)
	First fragment	1	Fragment offset is 0 (first fragment).
	Header checksum error	1	Validates header checksum (without options).
Control protocol	ICMP	1	Next protocol = 1
	IGMP	1	Next protocol = 2
SIP check	DIP = SIP	1	DoS rule
	Known SIP range	1	SIP & Configured SIP-Mask = Configured SIP
DIP check	E0-00-00-xx	1	224.0.0.0 to 224.0.0.255 multicasting on the local subnet.
	E0-00-01-xx	1	224.0.1.0 to 224.0.1.255 multicasting Internetwork Control Block
	Known DIP1	1	DIP & Configured DIP-Mask1 = Configured DIP1
	Known DIP2		DIP & Configured DIP-Mask2 = Configured DIP2
	Known DIP3	1	DIP & Configured DIP-Mask3 = Configured DIP3
Next Protocol	TCP	1	Next protocol = 6
	UDP	1	Next protocol = 17
	MPLS	1	Next protocol = 137
	GRE	1	Next protocol =47
	IPv4	1	Next protocol = 4
	IPv6	1	Next protocol = 41
	ICMP/IGMP	1	Next protocol = 1 or 2
	Other	1	Next protocol non of the above.
Hash	Hash(DIP,SIP)	16	Used for L3 based load balancing

4.7.1.3 IPv6 Protocol Decoder

The IPv6 decoder decodes IPv6 header format. The decoder sets report bit fields per validity check. The header pointer should point to IPv6 header. The protocol decode size should be 40B.

Table 4-5. IPv6 protocol decoder

DECODE TYPE	DECODE RESULT FIELD	BITS	DESCRIPTION
Header	Options Header exists	1	
	Length is 0 and Next Header is not "Hop by Hop"	1	
SIP	SIP=0	1	::/128 -Unspecified address.
	SIP=1	1	::1/128 - The localhost loopback address
	SIP is Multicast	1	Multicast addresses in IPv6 have the prefix ff00::/8
DIP	ff02::XX	1	Link-Local Scope Multicast Addresses
	ff02::01XX	1	
	ff02::01:ffXX:XXXX	1	Solicited-Node multicast address
	DIP multicast	1	Multicast addresses in IPv6 have the prefix ff00::/8
	DIP Well Known multicast	1	
	DIP = 0	1	::/128 — The default unicast route (unspecified) address
	DIP = 1	1	::1/128 — The localhost loopback address
SIP/DIP scope	SIP or DIP Link local scope	1	
	SIP or DIP Site local scope	1	
	SIP and DIP Global scope	1	
	SIP = DIP	1	
Next Protocol	TCP	1	
	UDP	1	
	MPLS	1	
	GRE	1	
	IPv4	1	
	IPv6	1	
	ICMPv6	1	
	Other	1	
Hash	Hash(DIP,SIP)	16	Used for L3 based load balancing

4.7.1.4 MPLS Protocol Decoder

The MPLS decoder decodes MPLS header format for up to 4 labels.

Table 4-6. MPLS protocol decoder (up to 4 labels)

DECODE TYPE	*DECODE RESULT FIELD*	*BITS*	*DESCRIPTION*
Label stack	Number of labels>4	1	When S *bottom of stack* flag is not found in first four labels.
	Last entry in the label stack	2	Values 0,1,2,3
Labels TTL	Label 1 TTL value = 0	1	
	Label 2 TTL value = 0	1	
	Label 3 TTL value = 0	1	
	Label 4 TTL value = 0	1	
	Label 1 TTL value = 1	1	
	Label 2 TTL value = 1	1	
	Label 3 TTL value = 1	1	
	Label 4 TTL value = 1	1	

4.7.1.5 MPLS Single Label Protocol Decoder

The MPLS decoder decodes MPLS header format.

Table 4-7. MPLS single label protocol decoder

DECODE TYPE	*DECODE RESULT FIELD*	*BITS*	*DESCRIPTION*
	End Of Stack	1	
	Null label (value<=15)	1	
TTL	TTL = 0	1	
	TTL =1	1	
Label value	Label value = Configuration 1	1	
	Label value = Configuration 2	1	
	Label value = Configuration 3	1	
	Label value = Configuration 4	1	
	Exception bit		Or-reduce (bits[7:0] xor Invert) && mask
	Stop Bit	1	Exception bit or (not exception bit && not bit[1])

5. C-programmable Task Optimized Processors (CTOPs)

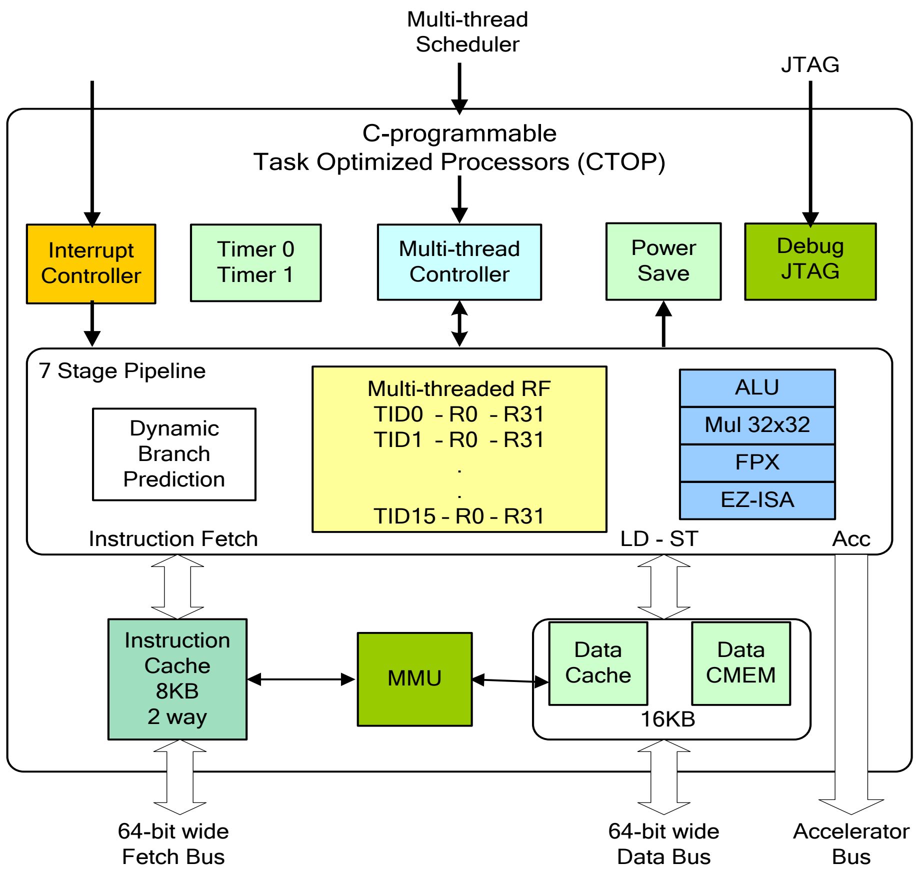
This section is intended to provide a high-level architectural overview of the C-programmable Task Optimized Processors (CTOPs) inside the NPS-400.

5.1 CTOP Overview

A CTOP is a 32-bit RISC processor implementing ARC™ ISA with extended support for large scale multithreading and EZ-ISA extensions for EZchip's proprietary packet optimized processing instructions. CTOPs provide a C-programming environment under an operating system memory management while retaining the strong network programming capabilities of the network Task Optimized Processors available in EZchip's earlier network processors.

The CTOPs offer a 7-stage execution pipeline with dynamic branch prediction. In order to support the 16 HW threads, 16 sets of register files are implemented as well as associated processor state context. Each thread is logically viewed as a virtual core. Some of the HW accelerators are built into the core as part of the EZ-ISA and others are provided through a shared accelerator block accessed by a dedicated Accelerator Bus. The Interrupt controller provides two levels of interrupts (high and low) and two dedicated timers. The Memory Management Unit (MMU) provides virtual memory addressing, memory protection and cache control.

Figure 5-1. CTOP block diagram



The CTOP has the following features:

- High performance, non-blocking, 7-stage execution pipeline
- 1.5 DMIPS/MHz for a single thread. 1GHz core clock speed.
- Multi-threaded design with sixteen HW threads with a dedicated register file and processor state context per thread.
- ARCompact ISA compatible freely mixing 16/32 bit instructions. Using 32-bit long immediate data (LIMM) also provides 48-bit and 64-bit instructions.
- The CTOP Floating Point Acceleration (FPX) supports IEEE-compliant instructions and library functions for high performance single-precision floating point operations.
- Networking ISA extensions
  - CRC, hash, fast checksum instructions
  - Bit manipulation instructions
  - Fast classification
  - Messaging and IPC/IPI
  - Buffer/Index allocation
  - Search/lookup macros
  - DMA controls
  - Wide load/stores
- Deep Packet Inspection ISA extensions for intrusion detection and prevention networking applications
- Networking extensions by shared accelerators
  - Security block ciphers
  - Search functions
  - Protocol decoding
- Out of order load/store interface
- Out of order variable delay accelerator activation interface
- Up to four open variable non-posted delay transactions (load/store or accelerators) per thread
- Misaligned load/store operations
- 8KB 2-way instruction cache with 32B cache line shared by all the threads
- Up to 16KB 2-way Data Cache (shared memory with CPMEM/CMEM); either shared by all threads or partitioned for 1/2/4/8/16 threads
- Configurable size thread private Scratchpad Memory (CPMEM) per thread and core shared Scratchpad Memory (CMEM)
- Standard MMU support combined with Fixed Mapping Table (FMT):
  - Enabling Linux operating system to run natively on each thread
  - Combining data plane applications using Fixed Mapping Table with Linux operating system services and debug using the TLB based MMU
  - Four entry instruction micro-TLB (fully associative)
  - Eight entry data micro-TLB (fully associative)
  - Joint Translation Lookaside Buffer (JTLB) x2 way with 256 entries
  - Dynamic branch prediction
    - Branch History Table
    - Subroutine call return detection
- JTAG debug interface

For more information refer to the *CTOP Programmer's Manual.*

5.2 JTAG Interface

The JTAG interface and protocol are according to IEEE 1149.1 standard. The module contains logic for communicating with each CTOP processor and its memory system, providing the host with a high level role where transaction parameters are simply specified.

Via standard 4-wire JTAG, the host device communicates with the JTAG module via four interface signals. These interface signals provide the host with the ability to control and serially pass data in and out of the module:

- TCK – Test Clock
- TMS – Test Mode Select
- TDI – Test Data In
- TDO – Test Data Out

An optional JTAG interface signal, Test Reset (TRST*) is provided to allow asynchronous initialization of the JTAG port without supplying a clock. Its use is necessary in simulation, but in actual operation it may be tied high. If the JTAG emulator chooses to take advantage of it, by using it to clock in TDO, it can compensate for the cable, board, and I/O pad delays to increase the speed at which TCK may be run.

6. Memory Architecture

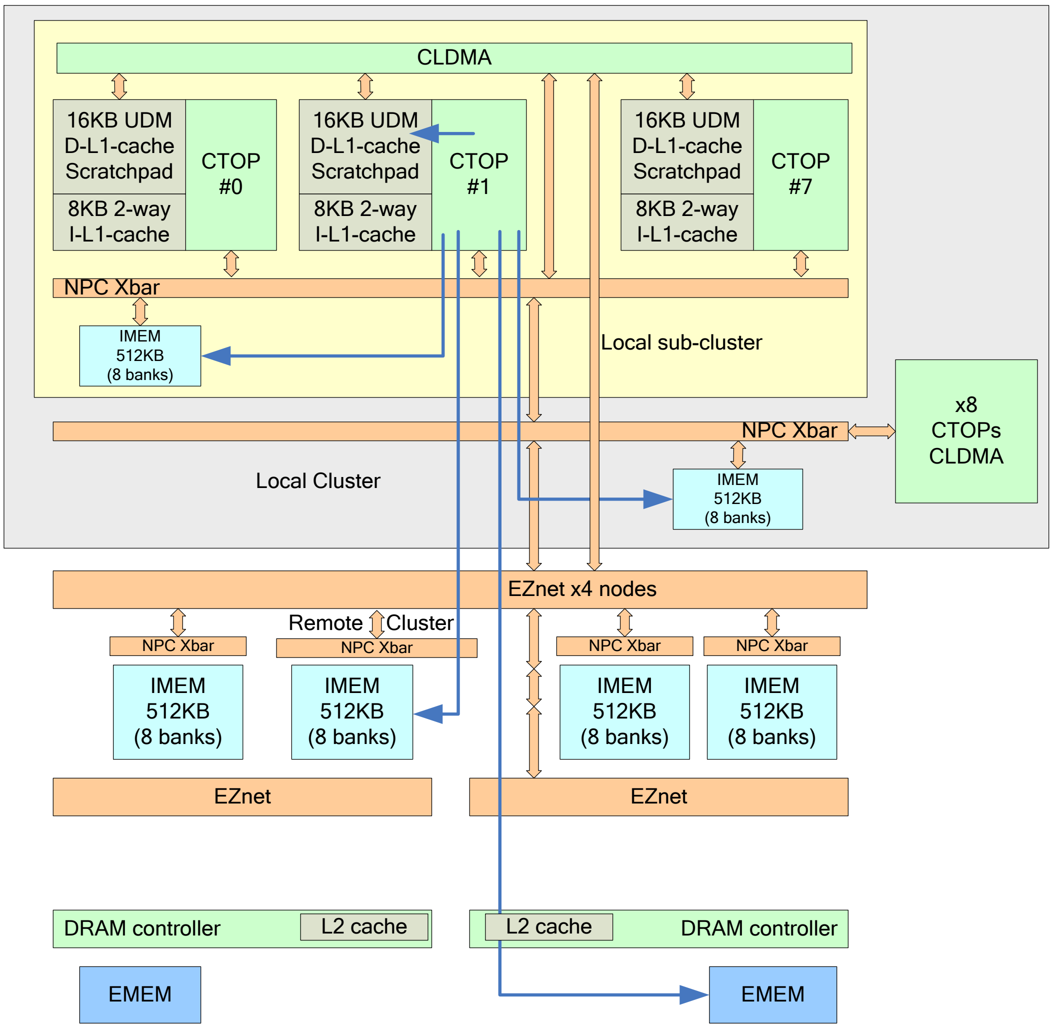
This section describes the components that comprise the NPS memory architecture.

- Each CTOP cluster of 16 CTOPs includes:
  - NPC crossbar interconnect of two sub-clusters.
  - Cluster EZnet router interconnects other four neighboring cluster nodes, two 128-bit channels per node. All channels are full duplex. Two memory request and two memory response channels.
  - HW accelerators for network protocol decoding, hash functions, lookup accelerators and cryptography.
  - MSU - Messaging unit and job scheduling
- Each CTOP sub-cluster of 8 CTOPs include:
  - Local portion of IMEM of 512KB comprised of 8 parallel accessible banks. The sub-cluster LRAM access latency is ~10 core clocks. Configured portion of the local IMEM memory is set to be shared at the sub-cluster level while other portions of it are system level shared. Thread private IMEM data is also managed in the sub cluster IMEM portion.
  - Cluster DMA (CLDMA) engine.

Each CTOP has dedicated:

- Instruction L1-cache of 8KB 2 way associative set. Zero wait state access (single core clock access latency).
- Unified Data Memory (UDM) of 16KB that can be shared by Data L1-cache (Tags), thread private CMEM for thread private scratchpad, and core shared CMEM for thread shared scratchpad. Zero wait state access (single core clock access latency).

Figure 6-1. NPS memory interconnect architecture



The following table lists CTOP access to memory and memory mapped IOs according to access latency:

Table 6-1. CTOP access to memory and memory mapped IOs according to access latency

MEMORY COMPONENT	*MEMORY CAPACITY*	*ACCESS PATH*	*READ ACCESS LATENCY*
CTOP instruction L1 cache	8KB 2-way	Closely coupled	Zero waits. Single core clocks
CTOP UDM (data L1 cache, private CMEM, shared CMEM)	16KB data (0/1KB/4B/8KB/16KB) cache 2-way tag coverage	Closely coupled	Zero waits. Single core clocks
Sub-cluster IMEM	Up to 512KB	Sub-cluster crossbar	~10 core clocks
Neighbor sub-cluster IMEM	Up to 512B	1-3 hops cluster crossbar	~14 core clocks
Neighbor clusters IMEM	Up to 4MB	One EZnet hop	~20 core clocks
All 16 far clusters LRAMs	16MB IMEM	1-14 EZnet hops	~40 core clocks
EMEM L2 cached data/code		1-15 EZnet hops	~50 core clocks
EMEM uncached data/code	4TB	1-14 EZnet hops, EMEM crossbar and DRAM access queues	~500 core clocks

6.1 CTOP Memory Management Unit

Multi-core Symmetric Multi Processor Linux requires support of shared coherent memory management.

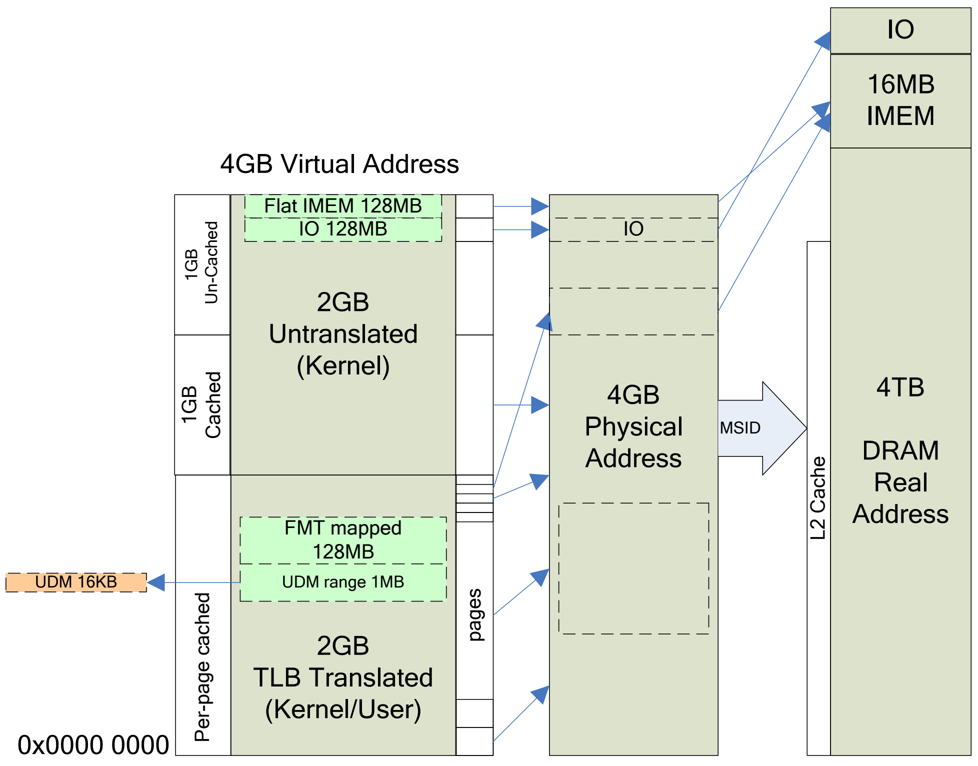
The CTOP supports virtual memory addressing via the Memory Management Unit (MMU). The MMU is managed by the Linux operating system kernel. The data uncached region is always active even when the MMU is disabled.

The MMU features a Translation Lookaside Buffer (TLB) for address translation and protection of 8 KB memory pages, and fixed mappings of un-translated memory.

The upper half (1GB) of the untranslated memory section is uncached (for IO usage) and the lower half (1GB) of the untranslated memory section is cached (for the Linux kernel).

The 32-bit CTOP architecture features a 32-bit virtual address space extended by an 8-bit selectable Memory Space Identifier (MSID) and a 32-bit physical address space translated to a 40-bit real address space. Additionally all MMU pages have an 8-bit Address Space Identifier (ASID) enabling virtual address space separation between user applications running under the Linux kernel supervision.

Figure 6-2. CTOP Memory Management Unit (MMU)



When the MMU is active, the CTOP defines a common address space for both instruction and data accesses. The memory translation and protection systems can be arranged to provide separate non-overlapping protected regions of memory for instruction and data access within a common address space.

The CTOP address space is unified. Separate memory spaces for code and data can be accessed through a fixed mapping table (FMT) of 8MB configurable pages.

The MMU pages are configured by OS to distinguish page access permissions for code and data.

The programming interface to the MMU has been designed to be independent of the configuration of the TLB in terms of the associativity or number of entries.

Virtual addresses in the range of 0x00000000 to 0x7FFFFFFF (except FMT enabled pages and CMEM/ private CMEM) are mapped by software managed page tables using hardware translation look-aside buffers (TLB). Private and shared CMEM ranges are 64KB that are limited to the actual UDM size of 16KB.

The 128MB of virtual translated address range (0x58000000-0x5FFFFFFF) may be translated via a Fixed Mapping Table (FMT) with an 8MB page size (total of 16 FMT pages). The FMT translation provides fixed performance behavior required by data plane applications. The FMT translation is independent of translation caching and associated cache-miss handling by the OS.

The 1MB range, 0x57F00000 to 0x57FFFFFF, is allocated for the CMEM range. This address range is mapped to CTOP 16KB UDM RAM.

From this 1MB range, the first 256KB is mapped to thread private memory, CMEMP, and the second 256KB is mapped to CMEM (shared between all threads).

A portion of UDM can be allocated for L1-Data-cache (shared or per thread) – see the Unified Data Memory section. The UDM can be partitioned by configuration to use part or all of its RAM as data cache instead of scratchpad.

The upper 128MB of the untranslated virtual range is mapped to flat IMEM view. Flat view is an additional way to access IMEM locations regularly accessed through FMT mapping.

The 128MB Kernel untranslated virtual address range 0xF0000000 to 0xF7FFFFFF is mapped to uncached/untranslated IO access.

6.1.1 Page Table Lookup

In a demand-paged virtual memory operating system (e.g. Linux), the OS maintains one or more page tables containing details of each translated page in each separate translated address space. This can be a very large number of pages.

The OS uses the MMU so that user programs can execute as if they had the machine to themselves – each user program runs in its own translated address space. The OS manages how chunks of physical memory are mapped into these translated address spaces, and sets access permissions to prevent malfunctioning or malicious code from making the whole system unstable, and to protect one application's data from others.

The CTOP allows for 256 separate virtual address spaces using an 8-bit address space identifier (ASID).

The 8-bit address space identifier (ASID) can be considered as an attribute of the virtual address. The ASID separates MMU pages within the lower 2GB address space between different user applications selected by the OS, or unifies them on shared MMU pages.

The page can be:

- Global, appears in all virtual address spaces – ASID bits for this entry are ignored, and must be set to zero.
- Page appears in a shared library as described by the SASID register. ASID[4:0] is compared against the corresponding bit position in the SASID.
- Page appears in a single virtual address space as qualified by the ASID bits A[7:0]. The ASID bits are tested against the current task's ASID from the machine status register (PID) to determine whether a TLB match has taken place. With an 8-bit ASID, it follows that the CTOP user tasks running under kernel MMU supervision support up to 256 concurrent virtual address spaces.

The CTOP Memory Management Unit (MMU) acts as a software-controlled cache into the OS page table, and performs hardware address translation and checks access permissions. Two levels of cache are provided:

- The first level consists of micro TLBs (or μTLBs). These are very small fully-associative caches into the second level of the MMU cache. They allow for single-cycle translation and permission checking in the processor pipeline. The uTLBs are updated automatically from the second level of the cache. A least-recently-used (LRU) algorithm is used to decide which uTLB entry to replace.

- The second level of the cache consists of the main TLB, or joint TLB (JTLB). This is a RAM-based 2-way associative structure, which is loaded by special kernel mode handlers known as TLB miss handlers. Separate vectors are provided for different instruction and data TLB miss handlers. A number of registers provide commands to control the reading and writing of TLB contents from the miss handlers.
- The final level of the hierarchy is the OS page table itself. This contains the complete details of each page mapped for use by kernel or user tasks. The uTLBs, main TLB and miss handlers combine to implement cached access into the OS page table.

Code running on the processor views the OS page table through the lens of the multi-level cache system implemented using the uTLBs, main TLB and TLB miss handlers. The OS is responsible for ensuring that the page table entries loaded into the MMU are always in sync with the OS page table in memory. This must be taken into account when loading entries into the MMU on a TLB miss, and when removing entries from the page table in memory.

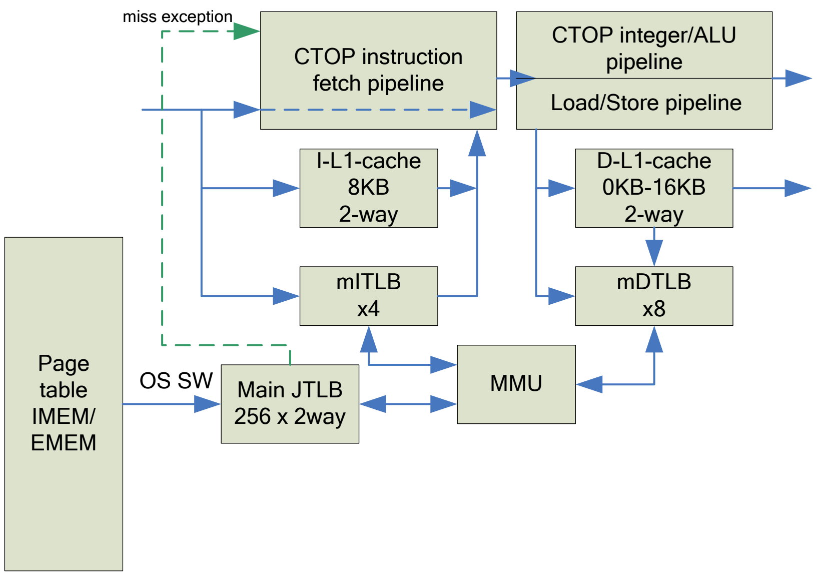
6.1.2 Translation Look-aside Buffers

To provide fast translation from virtual to physical memory the MMU contains Translation Look-aside Buffers (TLBs). The MMU can be thought of as a two level cache for page descriptors: The µITLB and µDTLB at level one, and the main (or Joint) TLB at level two. The µITLB and µDTLB contain copies of the content in the Joint TLB. The µTLBs may have descriptors not contained in the main TLB.

The µTLB miss results in checking if the page exists in JTLB and subsequent auto-copying of that page to the selected µTLB entry. In case JTLB gets page miss as well, a SW exception is triggered to fill the JTLB with the page from the main page table.

In addition to providing address translation, the TLB system also provides cache control and memory protection features for individual pages.

Figure 6-3. Translation Look-aside Buffers (TLBs)



The µITLB and µDTLB are fully associative and physically located alongside the L1 instruction cache and L1 data cache, respectively, where they perform the virtual and physical address translation. The µITLB and µDTLB are hardware managed. On a µITLB (or µDTLB) page miss, the hardware fetches the missing page from the main TLB.

The Main TLB consists of two-way set associative Joint Translation Lookaside Buffers (JTLB), with 256 entries. The Joint TLB is software managed. On a joint TLB page miss, the operating system has to fetch the missing page descriptor from memory and store it into the Joint TLB. No part of the MMU has direct access to the main memory. The Joint TLB is filled by software through an auxiliary register interface. The instruction that caused the µTLB miss is retried while the JTLB is interrogated.

6.1.3 Linux Kernel Fixed (Untranslated) Mapping

The upper 2GB of virtual address space, 0x80000000 to 0xFFFFFFFF, is mapped to physical address space 1:1 fixed mapping according to ARC memory architecture.

The upper 1GB of virtual address space, 0xC0000000 to 0xFFFFFFFF, is uncached by L1 CTOP I/D caches.

The lower 1GB of untranslated virtual address space, 0x80000000 to 0xBFFFFFFF, cached by L1 CTOP I/D caches per global configuration Caching is conditioned by the instruction cache or data cache global enable bit, respectively.

The untranslated 2GB range is configured to use the default MSID for real address mapping. The default MSID is shared with TLB translated address range – see Memory Space Identifiers for MSID definition.

The default MSID is used to access either the EMEM/IO or IMEM/IO. The IO access permission can be enabled per default MSID (virtual range 0xF7000000 – 0xF8FFFFFF). For IO register access permission, the default MSID should be configured in the Cluster Interface Unit to enable configuration access – in this case, addresses 0xF7000000-0xF8FFFFFF are assigned to the SOC configuration area (total of 32MB).

The upper 128MB of the untranslated virtual range is mapped to a flat view of the IMEM, where the lower 16MB of this range covers implemented IMEM.

The 128MB Kernel untranslated virtual address range 0xF0000000 to 0xF7FFFFFF is mapped to memory-mapped IO covering the entire device configuration address space.

6.1.4 Fixed Mapping Table Address Translation

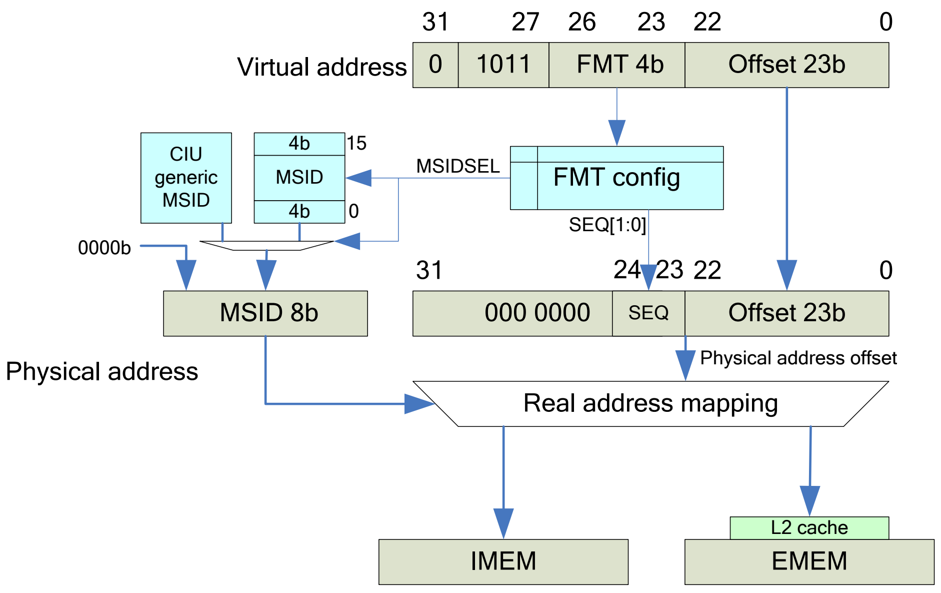
The Fixed Mapping Table (FMT) is mainly defined to service data plane applications. Data plane applications are expected to work with private and shared data. Shared data such as lookup structures require frequent DP access but a relatively low update rate, therefore can utilize a simple SW coherency scheme. The data plane application may implement SW controlled coherency protocols when pipelining execution data to the other core or can refrain from caching shared data.

The FMT is defined for CTOP virtual address translation in addition to TLB.

The FMT translates up to sixteen 8MB pages (128MB) of the lower 2GB virtual address range, 0x58000000 to 0x5FFFFFFF. Enabled FMT pages have priority over MMU pages on the sane address range.

The FMT offers large translation pages of 8MB. The 128MB virtual address range is covered by 16 pages of 8MB. Virtual address bits [26:23] selecting the FMT page.

Figure 6-4. FMT Address Translation



The MSID 8b value range is 0-255. The FMT page 4b selects MSID value 0-15.

The MSID values 0-15 have predefined functionality of address remapping to IMEM/EMEM.

There are eight predefined MSIDs for IMEM application data.

There are six predefined MSIDs for IMEM application code.

There is a single predefined MSID for EMEM application code.

There is a single predefined MSID for IMEM/EMEM combined region for application data.

The FMT configuration generates 5-bit MSID selection, where the lower 16 values refer to MSID values 0-15, and the upper 16 values are mapped by the CIU to any of the generic MSID values (see MSIDSEL[5]=1 in FMT Configuration below).

Up to four 8MB pages can be concatenated using 2-bit SEQ number to form a 32MB region of the same FMT selected MSID.

6.1.4.1 FMT Configuration

The configuration of each FMT memory region is:

REGION	***NAME***	***DESCRIPTION***
V	Valid bit	When this bit is clear the FMT page does not exist and the address range can be mapped by MMU TLB pages (using CTOP default MSID).
MSIDSEL[5:0]	MSID Select	Lower four bits of the MSID (the upper bits are implicitly zero). Assigning an aligned 8MB FMT page address range on this application to one of MSID0 to MSID15. More than one FMT can share an MSID value to cover >8MB memory space or have different access attributes with the same MSID. When MSIDSEL[5]=0 MSIDSEL[3:0] choose one of 16 FMT spaces, When MSIDSEL[5]=1 MSIDSEL[4:0] choose one of general MSIDs configured in the CIU. The FMT configuration includes only MSIDSEL[3:0]. MSID[5] and MSID[4] are selected through special CTOP Lookup Descriptor addressing modes.
C	L1 Cached	In data spaces this bit controls enabling or bypassing the L1 data cache on load/store accesses to/from this region. The data cache coherency is supported only per CTOP. There is no coherence among CTOPs on shared address spaces. In code spaces the bit controls enabling or bypassing the instruction cache.
UW	User-Writable	User writeable memory region. This bit enables or disables user mode program to access to the region on store operations. Kernel mode program can store to any valid range regardless of the bit state. Kernel mode may change this bit on the FMT to allow user code store to the page. A privilege violation causes a precise exception. The per MSID read/write attributes do not affect FMT mapped MSIDs.
SEQ[1:0]	Sequence	Order identifier within the selected MSID address range. In case the IMEM/EMEM address range is larger than 8MB, each 8MB block gets a different order identifier and sequential FMT entries are programmed together to combine a continuous address range. The last region may not be full. For example, a 10MB address range can be selected by combining an 8MB range with a 2MB range in two FMT entries.
SIZE[3:0]	IMEM Size	Four bits selecting power of two IMEM physical size for the private data region (configured per core/sub-cluster), in 256B granularity (0=2^{0+8}=256B, 1=2^{1+8}=512B up to 15=2^{15+8}=8MB). The max value covers 8MB IMEM page granule. Additional FMT entries should be used to cover more than 8MB range. In MSID regions that access EZnet shared data, these bits are not used and replaced by MSID size configuration in the CIU which works in round granules (full round memory size is 128B x number of EZnet nodes). A transaction that exceeds the configured memory space range causes an imprecise memory error exception.
BASE[10:0]	Address	The base line is selecting the first line in the physical IMEM array that serves a single EZnet node. According to the MSID spreading the number of nodes per round is determined: two nodes for sub cluster, four nodes for single cluster, and eventually 64 nodes for all 16 clusters. The base line is always normalized to be the first lines in all arrays that serve this memory space. With the full IMEM size of 16MB and 64 EZnet nodes, each node maps to 256KB, which is 2048 x 128B lines. Therefore, 11 bits select the first line, in a normalized way for any MSID, and the IMEM is divided in granules of 1/2048 of its side between FMT based MSIDs.

Access range for internal cluster access is controlled in the core, before entering the local crossbar. Access range of multi-cluster IMEM access is controlled in the CIU, before sending the transaction to EZnet. Any access within the 8MB page that falls beyond the configured MSID size is detected and causes an imprecise memory error exception.

6.1.5 Memory Space Identifiers

The Memory Space Identifiers (MSIDs) are used to identify memory regions with common attributes. The MSID extends the physical address.

The MSID are EZnet global values in range 0-255.

The MSID value is assigned to each memory transaction:

- Range of 16 predefined MSIDs (values 0-15) assigned per CTOP physical address range by FMT. Each FMT/MSID entry may address up to 8MB virtual space. Multiple sequential entries cover up to 32MB virtual space. FMT/MSID is pre-configured way for IMEM or combined IMEM/EMEM access.
- Range of 24 sub-cluster general MSIDs (values 16-255) assigned per eight CTOPs default or HW accelerator engine. No pre-configured address range limitations besides the granularity of 1MB on address range sizes. The general MSIDs are mainly for EMEM and Statistics access.

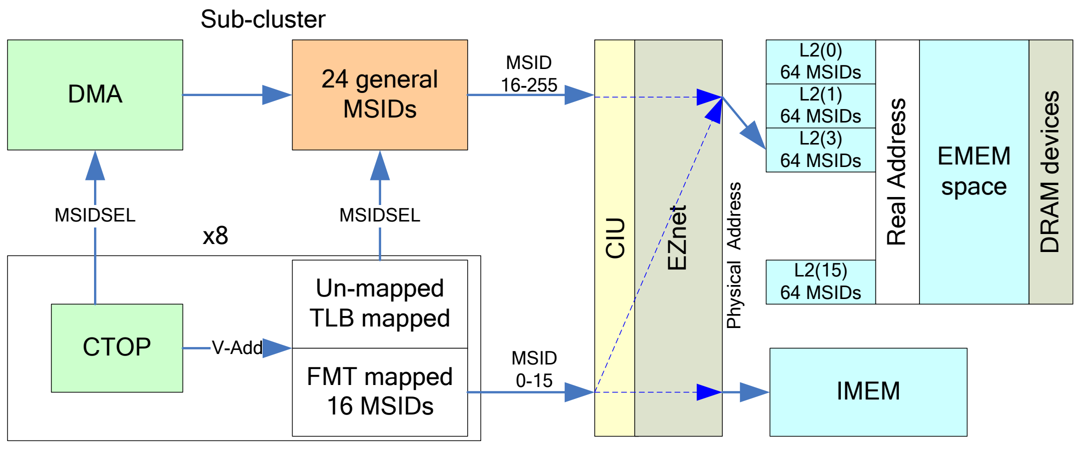
The MSID attributes are assigned per CTOP sub-cluster (CIU).

The MSID attributes specify:

- Address range cacheability
- Address range sharing
- Access privilege rights
- Target EMEM L2-cache spreading profile.
- ECC protection (see details in the In-band ECC/Parity Protection section):
  - No Protection (raw data)
  - LLP (Linear Line Protection, 7x16B ECC+ 7x128B data line structures)
  - EBP (Explicit Burst Protection, 1B checksum +15B data structures)
  - Protection by duplication (126-bit + 2-bit and recover from duplicated table).
- Control of duplications of lookup structures' number of copies in L2C, GCI or combinations.
- Association to PCI Express ports.

The MSIDs/PADDR combination, addressing EMEM, is used by the EMEM L2 cache for physical to real address translation and maintaining memory consistency without blocking independent accesses. The MSID configuration is local to each L2-cache instance (there are 16 L2-cache instances). Each L2-cache instance supports up to 64 MSIDs.

Figure 6-5. MSID



DMA gets MSIDSEL from the CTOP that activated it and passes that to the general MSID table in CIU.

6.1.5.1 CTOP Pre-configured MSIDs

There are 16 MSID values (0-15) that are globally pre-configured. The MSID is selected by CTOP per FMT virtual address translation (see Fixed Mapping Table Address Translation).

Each of the pre-configured MSIDs has dedicated functionality.

MSID VALUE	NAME	MAX USAGE	SHARING	ADDRESS CALCULATION METHOD
0	PDMEM (IMEM)	Data 512KB	Per thread	Per thread private RAM accessed through NPC local sub-cluster local crossbar, to a close IMEM array. Each thread sees a private non-overlapping memory slot. The region size seen by each thread is from 256B up to the entire sub-cluster memory size divided by number of threads.
1	PDEMEM (IMEM, (EMEM)	Data 8MB	Per thread	PDEMEM is an extension to PDMEM with capability to access/cache private data on EMEM with programmed window size mapped to IMEM, providing low miss penalty on selected area. This memory space is typically used for thread private variables and stack that are typically cached. The entire region size covers the unified EMEM. IMEM window size is programmed as well to hide a small portion of the EMEM range. The IMEM window may be disabled, directing all cache misses to EMEM. The stack pointer is typically initialized to an offset in the IMEM window. As stack grows backwards due to push operations it may spill into EMEM range.
2	LDMEM (IMEM)	Data 512KB	Sub-cluster	Access goes to the NPC sub-cluster local crossbar to the local IMEM array without using EZnet resources. Same shared address range is seen by all NPC threads (up to 128 threads per sub-cluster).
3	LCMEM (IMEM)	Code 512KB		
4	IDMEM1 (IMEM)	Data 1MB	Cluster	Access goes to the NPC local crossbars to the local IMEM array without using EZnet resources, but may cross between east and west sub-crossbars. Same shared address range is seen by all NPC threads (up to 256 threads).
5	ICMEM1 (IMEM)	Code 1MB		
6	IDMEM2 (IMEM)	Data 2MB	2x clusters	Access goes through EZnet to one of two neighboring NPCs (including self) organized vertically (1x2). Same memory contents are seen in these spaces by all threads in the two clusters.
7	ICMEM2 (IMEM)	Code 2MB		
8	IDMEM4 (IMEM)	Data 4MB	4x clusters	Access goes through EZnet to one of four neighboring NPCs (including self) organized (2x2). Same memory contents are seen in these spaces by all threads in the four clusters.
9	ICMEM4 (IMEM)	Code 4MB		
10	IDMEM16 (IMEM)	Data 8MB	All	Access goes through EZnet to one of sixteen neighboring NPCs (including self) organized (4x4). Same memory contents are seen in these spaces by all threads in the sixteen clusters. Multiple FMT entries using the SEQ bit can cover larger FMT area.
11	ICMEM16 (IMEM)	Code 8MB		
12	IDMEMG (IMEM)	Data 8MB	All	Access goes through EZnet to all NPCs (including self). Same address range is seen by all threads. This region is not accessible by IO peripherals. Multiple FMT entries using the SEQ bit can cover a larger FMT area. All NPCs sharing is equivalent to spreading over 16 NPCs. Applications that need to spread on the entire SOC should use global mapping for forward compatibility.
13	ICMEMG (IMEM)	Code 8MB		

MSID VALUE	NAME	MAX USAGE	SHARING	ADDRESS CALCULATION METHOD
14	IDMEMGIO (IMEM)	Data 8MB	All	Shared CTOP data and IO peripherals. Same as IDMEMG except that IO peripherals have read/write access privileges to this region. Multiple FMT entries using the SEQ bit can cover a larger FMT area.
15	EDMEMG (EMEM)	Data 8MB	All	This address space provides up to 8MB per FMT page holding shared data residing in external memory. Access goes through EZnet to nodes connected L2 caches and through there may continue to the appropriate DRAM storage location. Same memory contents are seen in these spaces by all the threads. Multiple FMT entries using the SEQ bit can cover a larger FMT area.

6.1.5.1.1 Private Data Memory Space – PDMEM (MSID=0)

The private data memory accesses IMEM instances are directly attached to the sub-crossbar of the core. Each eight cores reside on the same sub-crossbar and there are two sub-crossbars in one NPC, one on the east and one on the west. Each sub-crossbar holds half of the NPC memory resources (0.5MB). PDMEM may be cached in the core's L1 data cache without need to maintain multi-core coherency protocols. When the data cache is configured to be shared by all threads, if PDMEM memory space is used it must be programmed to be non-cached.

PDMEM allows only data accesses. Instruction fetch from this space is detected in the core and causes precise privilege violation exception. User code privilege to store into this space is controlled by the SW FMT CTOP configuration and may be set to read-only or read/write. User code attempting to store into a read-only space causes a precise privilege violation exception. Any access within the 8MB page that falls beyond the configured MSID size (power of two size per thread) causes an imprecise memory error exception.

A configured portion of the sub-crossbar IMEM arrays is partitioned per thread to hold private data. Eight cores and 128 threads can share this range. Each thread sees a private non-overlapping memory portion of the local IMEM instances attached to the sub-cluster.

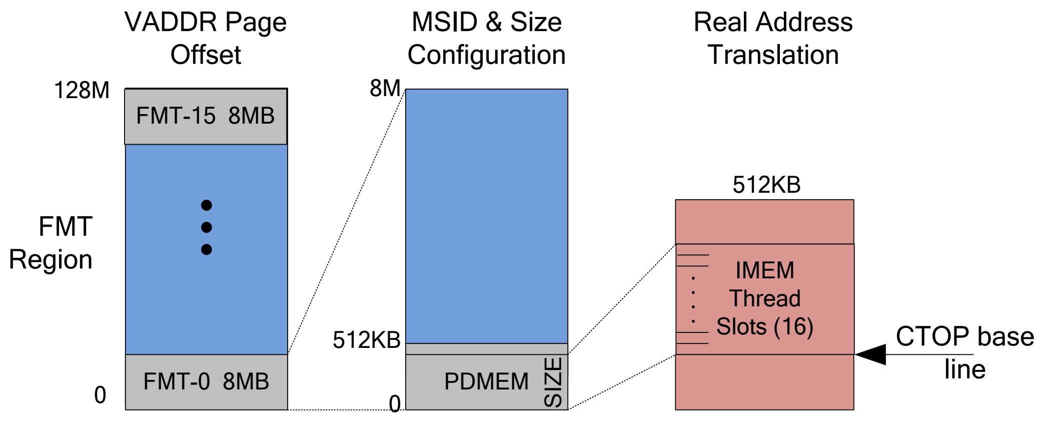
Each CTOP MTM configures a separate base address (sub-cluster array first 128B line) and region size (number of lines which is power if two). This enables putting combinations of private thread memory regions per CTOP. For example: one CTOP with 16 threads in one place and another CTOP with two threads in another place. The local IMEM array start offset and region size may be configured uniformly for the entire cluster, creating a symmetrical partition.

Start offset is the number of the 128B line within each array and region size is the power of two number of 128B lines used by each thread. The base address configuration selects the first IMEM line with 128B granularity, and gets added to the calculated line number.

For each CTOP the configured base address replaces the core ID in the address, and added in the MTM to an offset which multiplies thread by the lines per thread (shift left due to power of two sizes). In a symmetrical case, NPS-400 supports from a minimum of two lines per thread (256B) up to 32 lines per thread (4KB) which together cover an entire sub cluster LRAM (128 x 4KB = 512KB). Each core MTM is configured to a non-overlapping location through a different base.

The configured region size seen by each thread ranges from 256B up to 4KB of the sub-cluster local IMEM array sizes, in power of two steps. For example, when 128 threads symmetrically share 512KB sub-cluster RAM, configuration may cover up to 4KB per thread, which is the entire array, and FMT size configuration is from 0 to 4 (0=2^{0+8}=256B, 1=2^{1+8}=512B …, 4==2^{4+8}=4KB). On cores with unified threads (e.g. 1, 2, 4 or 8 threads) each thread can get the relative proportion of the unused threads. For example, on a core running eight threads instead of sixteen, each thread gets twice the memory (threads 0-7 get two accesses per round through slots 0-15).

Figure 6-6. Private Data Memory Space



6.1.5.1.2 Private Data External Memory Space – PDEMEM (MSID=1)

PDEMEM is an extension to PDMEM with capability to cache private data on EMEM and cover some portion of it in PDMEM window.

On PDEMEM range only data accesses are allowed. Instruction fetch from this space is detected in the core and causes precise privilege violation exception. User code privilege to store into this space is controlled by the UW FMT configuration and may be set to read-only or read/write. User code attempt to store into read-only space causes precise privilege violation exception.

The region size configuration defines the EMEM range. EMEM access within the 8MB page that falls beyond the configured MSID size is detected in the CIU and causes an imprecise memory error exception.

The MSID=1 has additional configurable MSID 8b value for EMEM access and additional space size for EMEM window. The EMEM MSID also configure with L2-cache usage profile (similar to general MSIDs).

The PDMEM window maps a configured portion of the page to sub-cluster IMEM. The IMEM window size is a power of two and it is placed in a naturally aligned offset within the PDEMEM page. This is configured by selecting number of base bits and their value within the 8MB page.

Each CIU sets a separate base address in EMEM and size per CTOP within its sub-cluster. This way each sub cluster is configured to a separate non-overlapping EMEM region and there is no constraint for symmetry between all sub-clusters.

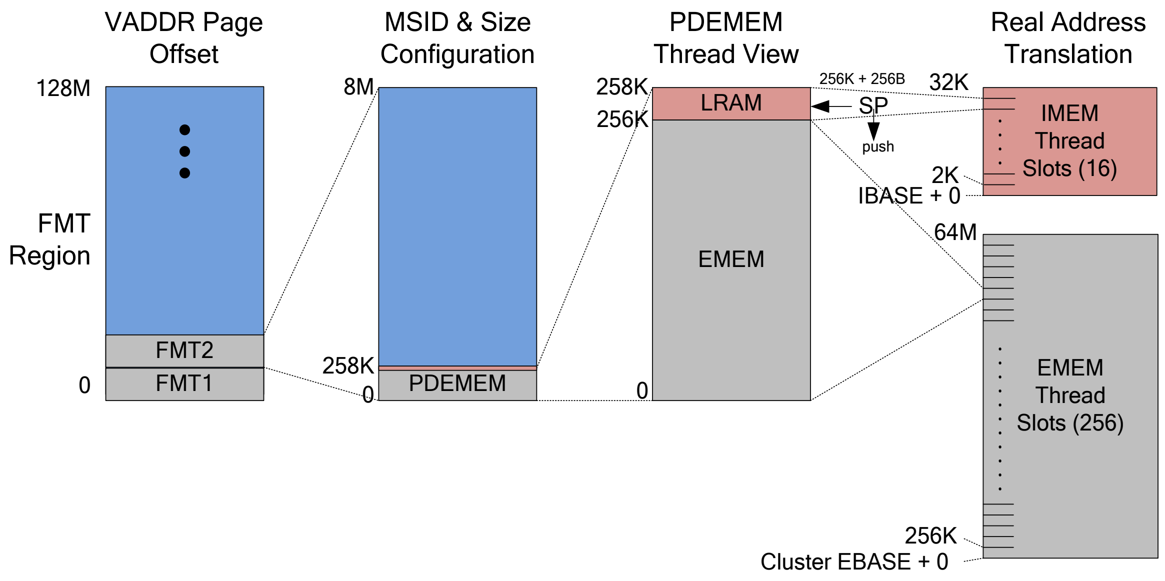
If the window overlaps EMEM range, it hides that portion of the EMEM. This scheme also enables placing the window immediately after the EMEM range achieving both continuous stack starting from IMEM and spilling to EMEM while keeping full utilization of EMEM storage.

This region can be used for caching large private data region in EMEM while keeping local window for critical data accessing the low latency IMEM. The IMEM window addressing is typically contiguous with the EMEM addressing. A stack pointer, for example, can be mapped to start in the IMEM window and continue with EMEM when stack depth increases due to repeated push operations (push on to stack decrements the stack pointer). When the region is cached, the beginning of the stack pointer is cached in the IMEM window with reduced cache miss latency. As the stack grows it is still cached, but cache miss results in EMEM access latency. On other portion of the IMEM window (above the stack pointer initial address) the application may place critical variables with reduced cache miss penalty. When the data cache is configured to be shared by all threads, if PDEMEM memory space is used it must be programmed to be non-cached.

Figure 6-7 shows a symmetric example of PDEMEM cached page. The EMEM region size is configured to be 256KB of thread private data. Assuming EMEM interfaces are equally shared by 4096 threads the actual EMEM covered region size holds 256KB slot per thread totaling 256KBx4096=1GB. The IMEM window is 2KB that get mapped into close IMEM array with slots per 128 threads, totaling 2KBx128=256KB. In this example the IMEM window is configured to start continuously following the end of the PDEMEM range.

Each thread sets its stack pointer to start at 256B offset within its IMEM window such that the first 256B data are pushed into the window (cached from IMEM). The stack expands outside the window when the stack grows where it gets cached from EMEM. SW may further optimize the beginning of the stack by locking few of the first 128B lines in the data cache and eliminating their eviction.

Figure 6-7. PDMEMC Arrangement Example



As seen in the example, the real memory address is separated per thread such that all threads using the same address range access non-overlapping data. Each CTOP selects a base address and concatenates four address bits (representing 16 threads) above this base address, mapped to the non-overlapping sub-cluster IMEM region.. The configured window size seen by each thread ranges from 256B up to 1/128 of the sub-cluster IMEM array sizes, in power of two steps. For example, when 128 threads share 512KB sub-cluster RAM, configuration may cover up to 4KB per thread, which is the entire array. The window size configuration for the FMT entry is therefore from 0 to 4 ($0=2^{0+8}$=256B, $1=2^{1+8}$=512B …, $4==2^{4+8}$=4KB). Other accesses are forwarded to the CIU for further analysis according to the entire region size (and the CIU enforces the entire window size).

In the EMEM region (any range within the memory space size not covered by the IMEM window), each CIU sets a unique EMEM base address for its sub cluster in equal steps such that the EMEM address range for the entire NPC looks continuous between the clusters, cores and threads.

EMEM accesses are routed through selected L2 cache instances to the target DRAM memory interfaces. For unified EMEM access, one memory slot is used per thread with an architectural limit to the page size (8MB per page, or 16MB for two sequential pages), or the EMEM size (number of threads multiplied by EMEM private region size).

With 16 NPC instances per SOC (4096 threads) DRAM space per page is from 1MB (4096*256B) to 32GB (4096*8MB).

6.1.5.1.3 Local Sub-Cluster Data/Code Spaces - LDMEM and LCMEM (MSID=2, 3)

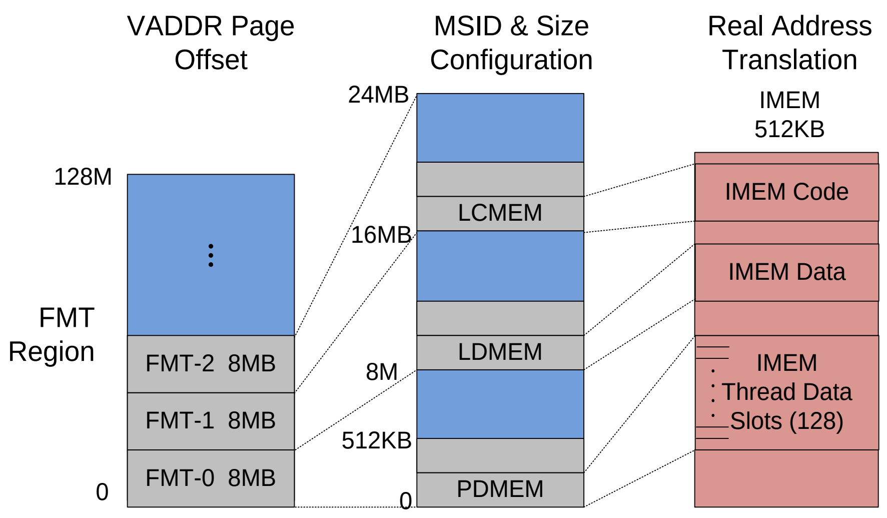
The Local Data Memory (LDMEM) and Local Code Memory (LCMEM) are both accessing IMEM arrays located in the current NPC sub-cluster. The IMEM array offset and size are configured for the sub-cluster. The memory reference does not reach the CIU and does not cross to the other sub-cluster. LDMEM/LCMEM spaces provide minimum access latency for critical data and code.

The actual IMEM array offset is determined by the base line. MSID base address is configured with 256B (2-line) granularity. LDMEM/LCMEM accesses are virtually spread between two EZnet nodes of the sub-cluster (north and south nodes).

In LDMEM range only data accesses are allowed. Instruction fetch from this space is detected in the core and causes a precise privilege violation exception. User code privilege to store into this space is controlled by the FMT configuration and may be set to read-only or read/write. User code attempts to store into read-only space cause a precise privilege violation exception. The region may also be defined as cached in L1 or bypassing it.

LCMEM range is used for critical code accesses which may be duplicated in all SOC sub-clusters (32 sub-clusters). Instruction fetches may be cached in the core instruction cache or bypassed according to the FMT cache configuration. User code privilege to store into instruction space is controlled by the FMT configuration and may be set to read-only or read/write (typically the kernel would configure FMT to disable user code from overwriting code space). User code attempts to store into read-only code space are detected in the core and cause a precise privilege violation exception.

Figure 6-8. Local Sub-Cluster Data/Code Spaces



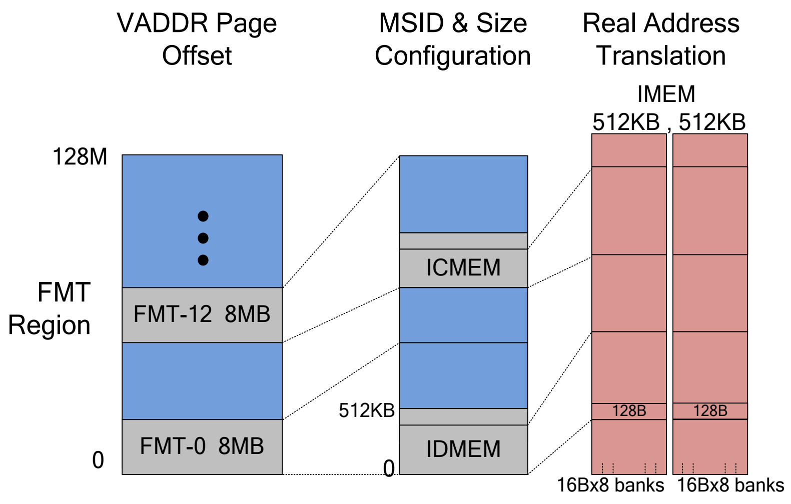
6.1.5.1.4 Single Cluster Data/Code Spaces – IDMEM1 and ICMEM1 (MSID=4, 5)

The local NPC data memory (IDMEM1) and code memory (ICMEM1) are both accessing IMEM arrays located in the current NPC. Each NPC holds 16 cores (256 threads) and 16 IMEM banks (eight banks per sub-cluster). NPC internal accesses do not use EZnet resources.

The MSID base address is configured with 512B (4-line) granularity. The address space is virtually spread between four EZnet nodes of the cluster. IDMEM1/ICMEM1 access privileges and enforcement are similar to LDMEM/LCMEM. Similarly, L1 cached shared data coherency should be preserved by SW.

Sixteen cores and 256 threads per NPC cover up to 1MB. Any access within the 8MB page that falls beyond the configured MSID size is detected locally by the MTM and causes an imprecise memory error exception. Size configuration selects number of lines within each local IMEM physical bank.

Figure 6-9. Single Cluster Data/Code Spaces



6.1.5.1.5 Multi-Cluster Data/Code Spaces – IDMEM2/4/16 and ICMEM2/4/16 (MSID=6-11)

The IDMEM2/4/16 and ICMEM2/4/16 are designated for multiple NPC shared data/code. Each NPC instantiates 16 cores and each core runs 16 threads, totaling 4096 threads. The multi-cluster memory spaces define different sharing levels with localization of neighboring clusters.

The number of threads sharing data in each memory space is 256 threads multiplied by the number of NPCs in the memory space.

Memory spaces are applicable on devices according to their implemented topology which is dictated by the horizontal and vertical dimension sizes. Multi-cluster spaces use EZnet resources to route transactions from one NPC to another.

The size configuration granularity is always line granularity multiplied by number of EZnet nodes (1KB for IDMEM2/ICMEM2 ... 8KB for IDMEM16/ICMEM16).

Figure 6-10. IDMEM2 / ICMEM2 1x2 cluster sharing

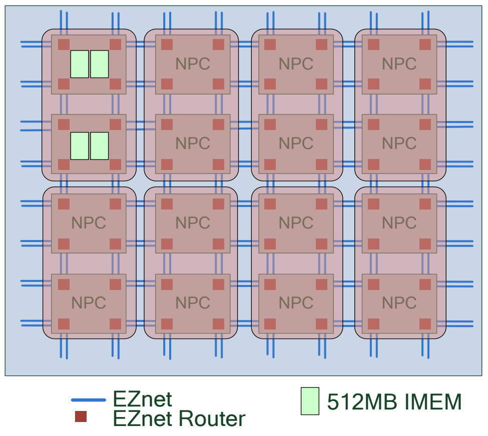


Figure 6-11. IDMEM4 / ICMEM4 2x2 cluster sharing

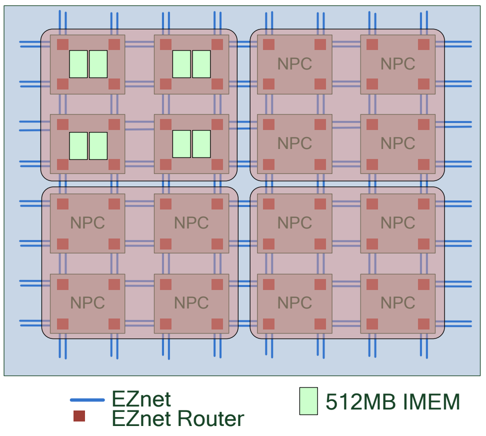
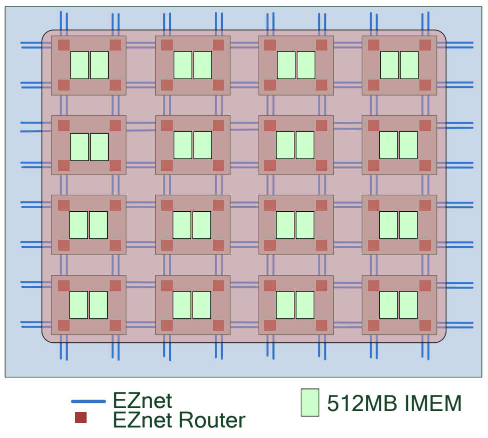


Figure 6-12. IDMEM16 / ICMEM16 4x4 cluster sharing



6.1.5.1.6 Global Data/Code Spaces – IDMEMG and ICMEMG (MSID=12, 13)

The global memory spaces have similar memory access attributes as the multi-cluster MSID except that they always spread data over the distributed IMEM banks. In NPS-400, the IDMEMG (MSID=12) is equivalent to IDMEM16 (MSID=10) and ICMEMG (MSID=13) is equivalent to ICMEM16 (MSID=11).

The same memory contents are seen by all SOC threads and size configuration of each page is limited to the FMT page size (8MB). Configuration enables non-power of two sizes for selecting the number of sub-cluster IMEM lines from two EZnet nodes (256B per sub-cluster), or one-line granularity per EZnet node, up to the number of lines per array that covers 8MB page size (1024 lines) per EZnet node. Consecutive FMT pages combined with the SEQ attribute can map the entire IMEM (two 8MB pages can cover 16MB IMEM).

The IDMEMG/ICMEMG spaces are not accessible by IO side peripherals, such as NDMA and PCIe, to provide proper protection of SW resources from HW IO blocks.

6.1.5.1.7 Global IO Data Space – IDMEMGIO (MSID=14)

The global IO data space has the same spreading algorithm, access privileges and base/size configurations for the CTOP cores as IDMEMG, except that this region is also accessible by peripheral IO devices such as NDMA and PCIe. It is typically used for storing shared data that needs IO treatment, such as frame data internal buffers, jobs descriptors and downloadable internal lookup tables.

Separation of IDMEMG from IDMEM provides protection for SW related global data that should reside in IDMEM from IO related data that could be read or written by peripherals and accelerators working on the shared data. When a peripheral tries to access one of the other predefined MSIDs, a privilege violation interrupt is generated to the device’s Global Interrupt Manager (GIM), which could steer it to any selected core/thread for further debug and analysis.

6.1.5.1.8 Global External Data Space – EDMEMG (MSID=15)

This predefined FMT space maps 8MB pages of globally shared data residing in EMEM. All NPCs are mapped to use the same L2 cache instances and the same spreading function such that each address is accessed in one specific L2 cache instance by all cores/threads.

Memory space spreads data between power of two number of L2 cache instances in a similar manner to EMEM accesses in PDEMEM space. Address scrambling selects the L2 cache instance within the selected target group. The internal line number per instance is divided by the number of instances and added to the base line number. Typically the base address is globally configured to be the same for all NPCs such that all SOC threads see the same memory contents. However system partitioning is possible as well, selecting a different offset for each NPC or group of NPCs and targeting different L2 cache instances.

EDMEMG space has the same privilege setup as other shared data spaces. This region is not accessible by IO side peripherals such as NDMA and PCIe.

6.1.6 Sub-cluster General MSIDs

In addition to 16 FMT mapped MSIDs there are 24 general MSIDs managed per sub-cluster of eight CTOPs. The general MSIDs can access EMEM or IO configuration (the IMEM is not accessible through these spaces).

The general MSIDs are used by:

- Up to eight can be used by CTOPs distinguished default MSID
- Accelerator MSIDs (e.g. lookup with/without duplications, reporting area shared between peripherals and the cores)
- On-demand and posted statistics.
- DMA MSIDs (frame data, reporting, PCIe shared buffers, etc.)

The MSID attributes configured in the sub-cluster CIU are:

- Enable/disable
- Address space size in 16KB granularity
- MSID value 8b for L2-cache tag caching of physical address
- L2-cache usage profile 1-4.
- Address scrambling controls
- GCI/PCIe/L2C interface selection.
- Lookup duplication control and GCI offload for EMEM lookup accesses
- L2C bypass

MSID L2-cache usage defines:

- Number of L2C sides.
- Groups size per side (1/2/4/8)
- Group ID in that side which is a function of size (limited by eight L2C per side divided by the group size).

L2C and EMEM coherence is implied for the case where only one data copy exists, and managed by SW for duplicated lookup structures.

The L2 cache, the MSID defines to convert the physical address to a real address which includes DRAM interface ID and offset. L2 cache programming also defines how the data is organized in external memory and whether the data structure transparently implements in-band ECC protection in one of a few options, according to the application needs.

6.1.6.1 CTOP Default MSID

The CTOP selects default MSID value from one of a sub-cluster of 24 general MSIDs.

This MSID is used for kernel untranslated and kernel/user TLB translated memory access.

In SMP applications, all CTOPs select the same default MSID. On the other hand, each CTOP can select its own private default MSID and get mapped to its own private EMEM location. Other combinations are also possible.

6.1.7 CTOP L1 Caches

A cache is a closely attached memory that sits between the processor and the IMEM/EMEM memory system.

The CTOP utilizes the following L1-cache types:

- 8KB 2-way set-associative instruction cache. Line length 32 bytes (8 words).
- Up to 16KB (configurable size) 2-way set-associative data cache. Line length 32 bytes (8 words).

6.1.7.1 Instruction Caches

When a processor requests an instruction from main memory, the request is passed to the cache. The cache essentially contains closely attached RAM that is designed to store frequently requested long-words. Upon presenting a request to the cache, a lookup is performed in the cache RAM contents for the instruction that the processor had requested. If the cache contains the requested instruction, it is fetched from the cache RAM and supplied to the processor. If however, the cache does not contain the instruction that has been requested, the cache stalls the processor until the requested instruction is fetched from main memory and written into local cache memory. The cache controller performs the process of fetching a missing long-word from main memory.

When the processor requests an instruction and the instruction resides in the cache RAM, the successful access is known as a cache *hit*. If however, the requested instruction is not in the cache RAM, the condition is referred to as a cache *miss*.

- Hit – When the requested instruction is present in the cache RAM.
- Miss – When the requested instruction is not present in the cache RAM.

Each cache *hit* takes a single cycle for the instruction long-word to return to the processor that requested it. This ensures that the processor is provided with a steady cycle-by-cycle instruction stream. There is significant performance degradation when a cache *miss* occurs, since the instruction cache halts the processor whilst it accesses main memory.

The CTOP instruction cache includes a pre-fetch mechanism for the warm thread to be executed next. It checks for the existence of the warm thread instructions in cache, and pre-fetches it to the cache in the background when necessary.

6.1.7.2 L1 Data Caches

The CTOP L1 data cache is implemented in Unified Data Memory.

The L1 cache is allocated by configuring UDM RAM to serve as either data cache, or thread private/ shared scratchpad area. The data cache size is selectable in power of two jumps from 128B up to the entire 16KB. Additionally, the data cache can be configured to be split per active thread, or unified for all the active threads (active threads are selected in power of two steps from single thread, two threads up to all 16 threads).

Only reads and writes to cacheable regions are presented to the data cache. The data cache is designed to store frequently requested data that can be written to or retrieved from. Upon presenting a request, the cache checks its local memory to see if the requested address location and associated data is present. If the requested entry is present, data is either written into the cache or alternatively retrieved from the cache. If however, the address being accessed is not present in cache, the cache accesses main memory to retrieve the missing entry. Upon retrieving the entry (the address and data), the processor represents the original request, this time guaranteeing a successful cache access. The load/store instructions have explicit data cache bypass control and may not be presented to the cache even in cacheable regions.

When the CTOP accesses a data word and the entry resides in the cache RAM, the access is known as a cache *hit*. If however, the requested entry is not in the cache RAM, the condition is referred to as a cache *miss*.

Unlike instruction caches, data caches allow the processor to write data into its cache RAMs.

6.1.7.2.1 L1 Data Cache Write Capability

The CTOP data cache is a non-blocking data cache that supports *hit-under-miss* and employs what is known as a *write-back, write allocate* policy that is used to define the operation of a write sequence.

Non-blocking caches are designed to allow out-of-order completion. Instructions that follow on from a data access that caused a cache *miss*, are allowed to execute whilst the cache fetches the missing data cache line. The processor pipeline continues to execute instructions only if there are no dependencies on the data that is being fetched.

If an instruction references the missing data word, the processor pipeline will stall. To further minimize processor pipeline stalls, the CTOP data cache utilizes a method known as *hit-under-miss*. Hit-under-miss is a scheme that allows the data cache to continue to supply data words if the access results in a cache *hit*, even when the cache is servicing a previous cache *miss*. This optimization in effect reduces the overall cache *miss* penalty for a given application, as the CPU is stalled for less time if there are a series of data accesses to the data cache.

Write-back (or copy-back) policies are used to reduce memory bandwidth by allowing the processor to locally modify a line if the data being accessed is cached. The processor reads or writes data that is held locally in the cache RAMs, therefore memory accesses are only limited by the speed of the RAMs rather than the large access times associated with IMEM/EMEM memory. The modified data is eventually written back to main memory only when either the programmer requests it (know as a *flush* command) or a cache *miss* occurs. If upon a cache *miss* the entry that is targeted for replacement is *dirty* (that is, previous accesses modified the contents of the line), the line is written back to main memory, therefore updating the main memory with the most recent copy of that line.

In addition to the write-back policy, a write-allocate (or fetch-on-write) scheme is used to further improve the efficiency of the cache. Upon a cache *miss* (read or write), the cache line that is missing is fetched from the main memory. The missing line is written into the cache RAMs, and the original request is re-presented to the data cache. A write-allocate scheme works on the assumption that subsequent accesses will occur from the same line that had been previously requested, therefore reducing the number of costly cache *misses*.

6.1.8 L2 Cache

In the CTOP memory page, attributes may bypass the L1-cache core and be evenly distributed in L2 cache instances, such that only one place is possible for each data copy. Therefore L2 cached shared data which is defined as uncacheable in L1 is naturally coherent. An exception to this is the case of memory spaces with duplicated lookup structures that may be configured to reside on multiple L2C instances.

The architecture defines a distributed L2 cache system that aggregates together a very large L2 cache shared by all cores. Data is never duplicated and any entry has a single L2 cache instance where it may reside. The L2 cache is distributed up to 16 cell instances (per MSID configuration). The shared L2 cache system can be flexibly partitioned between memory spaces.

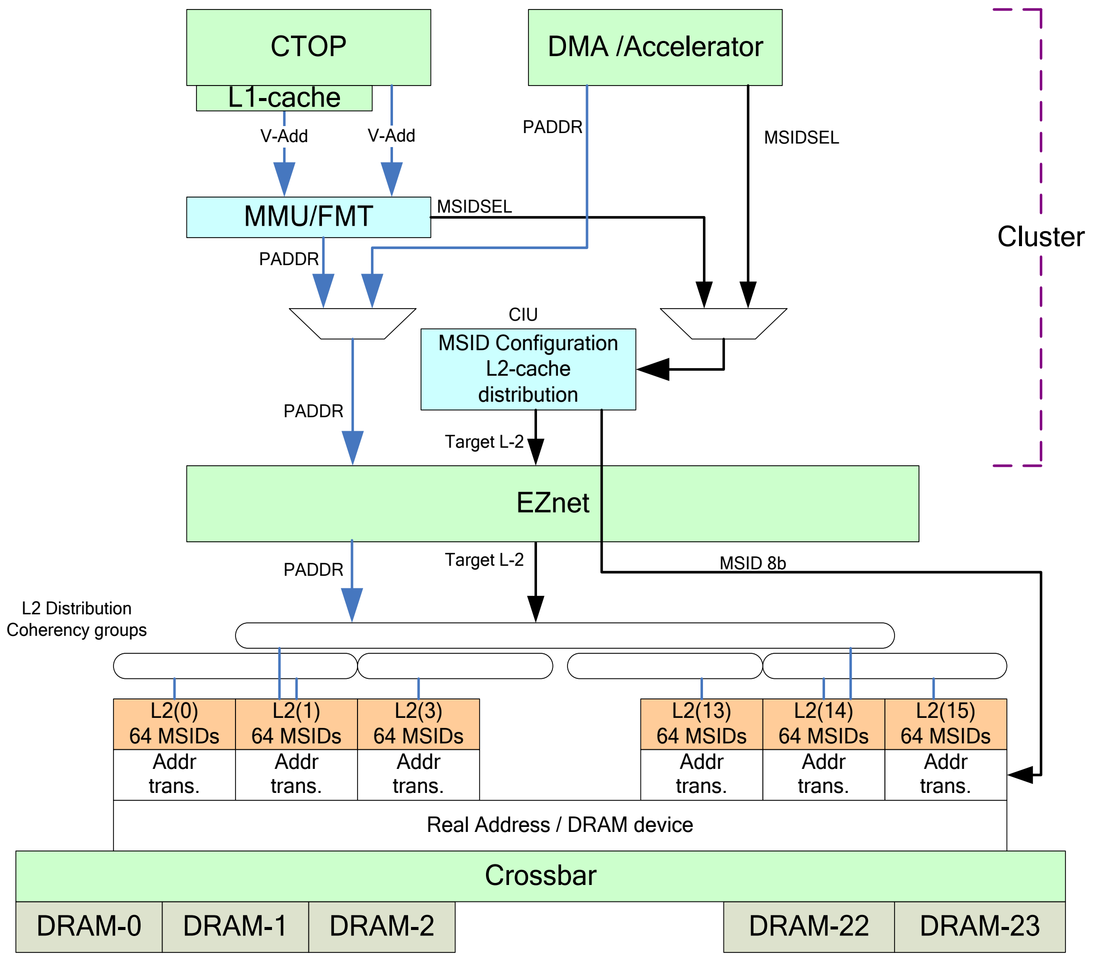
The MSID selects side, group size (number of L2C instances per side) and the group ID on that side. Therefore, continuous groups with power of two L2C instance counts are used on one or both SOC sides.

Each L2 cache instance may serve more than one memory space at time by including the MSID in its tag data.

The L2 implementation ensure data consistency (read after write, write after read, write after write) when accessing the same address locations, such that the sequence of receiving access commands shall be correctly reflected in the memory contents and read data results. Additionally the MSID can be configured to bypass L2C and memory consistency would consider the actual DRAM access as the consistency point.

The L2 cache allows non-blocking out-of-order transaction launch to non-colliding addresses while dependent addresses are tracked and serialized. The L2 cache completes the address translation process from physical address to real DRAM address, and spreads the transactions per memory space between multiple DRAM devices according to the MSID selected spreading profile. It is also responsible for the EMEM data integrity, and defines different ECC protection methods to optimize the DRAM usage by each transaction type associated with that memory space. The L2 cache also takes part in load balancing of lookup operations, by selecting one of multiple duplications of a lookup table based on per DRAM device memory interface loads.

Figure 6-13. L2-Cache



6.1.8.1 MSID Caching Attributes

The system supports 256 MSID values. Each L2-cache instance supports up to 64 MSIDs.

Each L2-cache instance maintains per MSID attributes that control DRAM access:

- Cache or Bypass cache
  - L2C bypass – completely skip the cache and also move the memory consistency point from L2C to DRAM device.
  - For the non-bypass case, the memory space may be cacheable or not cacheable.
- DRAM devices for MSID. The MSID data is spread across configured DRAMs. There eight profiles per L2-cache instance that configure DRAM spreading:
  - ID list of physical devices, mapping from logical ID (device number calculated from the address) to the physical device ID.
  - Number of devices for MSID spreading.
- Physical address to real address translation. Each MSID has a configurable offset (intermediate address) in the DRAM device in 8KB granularity.
- Each MSID has a configurable line count in the DRAM device.
- DRAM data structure and ECC protection.
  - ECC protection is included in-band with the data and organized in a way that best suits the memory space requirements. In continuous application linear memory view, the ECC structure is hidden from the application and affects under-the-hood the way a continuous physical address range gets translated to the real DRAM address range with embedded ECC protection. For more on ECC protection (Linear Line Protection ECC, etc.) see the In-band ECC/Parity Protection section.
  - Lookup structures have in-band parity protection that is checked by L2C. EMEM lookup structures support duplication count and relaxed memory consistency for high performance reads during structure update writes.

6.1.8.2 Servicing Shared MSID by L2-Cache Instances

The MSID that defines a multi-cluster shared memory space needs to have the same configuration on all clusters. The same {MSID,PADDR} transaction originated by any shared cluster is serviced by the same L2-cache instance.

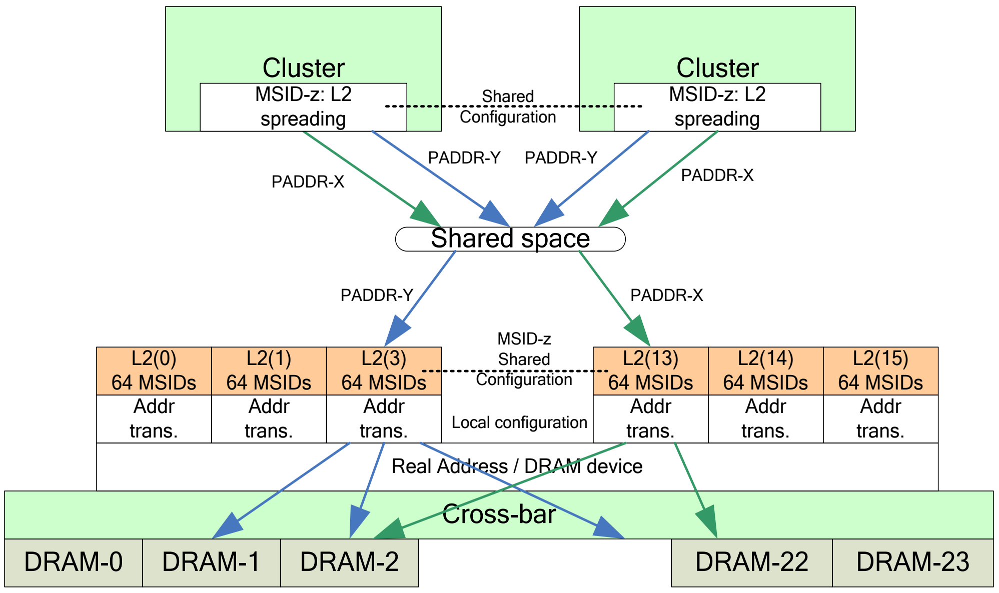
The L2-cache per MSID configuration must also be the same for all L2-cache instances for the shared space:

- Cached or Bypass.

The DRAM spreading of L2-cache instance is local and does not need to be same per shared MSID.

For a cache-enabled space, the L2-cache serves as the data consistency point. For a cache bypassed space, the DRAM serves as the data consistency point.

Figure 6-14. Servicing shared MSID by L2-cache instances



6.1.8.3 L2 Cache Bypass

When an MSID is defined to bypass the L2-cache, all DRAM transactions are routed to the nearest L2-cache instance in the direction of the requested EMEM side. The same {MSID,PADDR} transaction is serviced by multiple L2-cache instances where the L2-instance services a group of CTOP clusters.

For data sharing, all L2-instances should have same configuration for MSID, directing {MSID,PADDR} transaction to the same DRAM device. The data consistency must be ensured by SW.

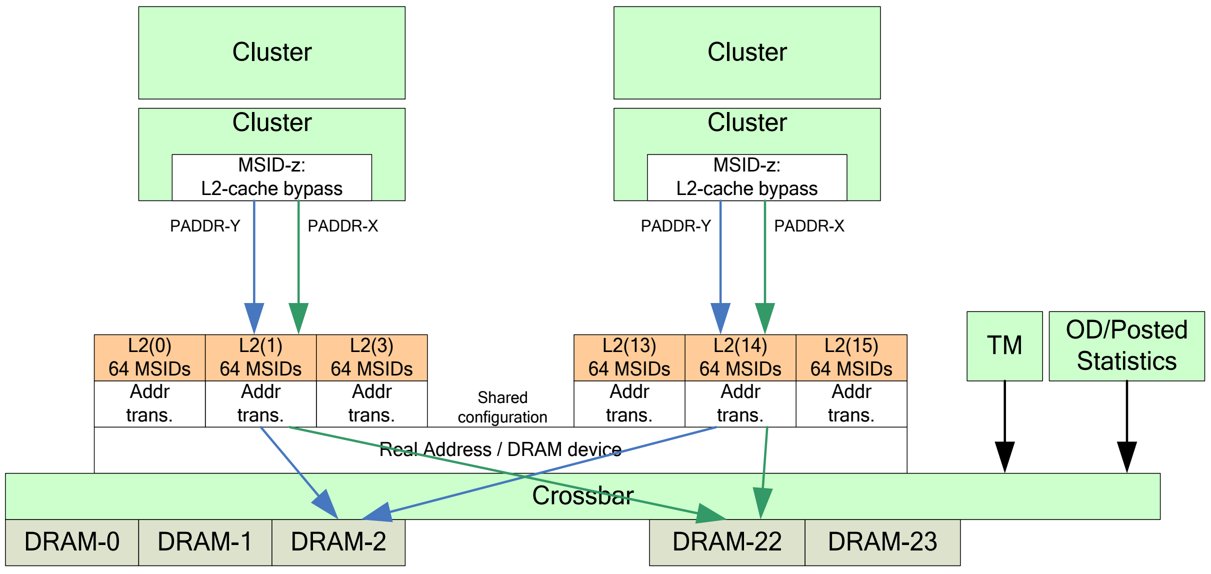
Reads and writes to the same address are not ensured to keep their order, unless SW uses data sync to ensure that all posted writes have been received by the EMEM consistency point. Similarly, HW blocks accessing EMEM through bypass avoid posted writes and ensure they are completed at the endpoint before a subsequent access to the same location is launched.

The shared DRAM serves as the data consistency point.

TM control and data structures that target EMEM also bypass the L2-cache instances directly to DRAM.

The on-demand and posted statistics structures also bypass the L2-cache instances directly to DRAM.

Figure 6-15. L2 cache bypass


Cluster
Cluster
MSID-z: L2-cache bypass
PADDR-Y
PADDR-X
Cluster
Cluster
MSID-z: L2-cache bypass
PADDR-Y
PADDR-X
L2(0) 64 MSIDs Addr trans.
L2(1) 64 MSIDs Addr trans.
L2(3) 64 MSIDs Addr trans.
Shared configuration
L2(13) 64 MSIDs Addr trans.
L2(14) 64 MSIDs Addr trans.
L2(15) 64 MSIDs Addr trans.
TM
OD/Posted Statistics
Real Address / DRAM device
Crossbar
DRAM-0
DRAM-1
DRAM-2
DRAM-22
DRAM-23

6.1.8.4 L2 Cache Features

Following are features of each L2 instance:

- L2 tag lines compare both PADDR and MSID; therefore each cache instance can serve multiple memory spaces without logical collisions. Up to 64 MSIDs are served by an L2 instance.
- The L2 cache-ability is configured per L2-instance/MSID.
- L2-cache bypassed MSIDs do not support merging writes, and L2 only provides address translation/ mapping services. L2 cache managed spaces support merging writes of either cached or uncached locations through atomic read modify write sequence.
- 8-way set associative. L2 cache instance selection by hash function that considers all useful address bits multiplies the effective associativity by the number of L2 cache instances.
  - Supports per memory space cache survival priority and minimization of thrashing between different memory spaces.
- Write-allocate write-back cache policy with optimized cache line survival policy:
  - Probability of a line staying in cache is SW controlled.
  - Safe-write supports overwriting data without read of write-allocate, or write merge (on uncached locations).
  - Two modes of operation:
    - General purpose mode (typically used by standard Linux applications). Write back policy and L2C managed full memory consistency.
    - Lookup structure mode – supporting duplication in multiple L2C instances and multiple load-balanced DRAM devices. Write-through policy and loose consistency scheme for read after write, enabling high bandwidth read accesses while update operation by write is taking place in all copies.
- Three survival priorities eliminate high priority lines from being evicted by lower priority lines.
  - Lines of the same priority use the LRU mechanism for eviction.
- 128 byte cache lines, with per 16-byte dirty and valid bit for DRAM write-back optimizations.
  - ECC data reads only once from the DRAM in one 16-byte burst (on the first cache line write miss), or with portion of the line (on first cache line read miss).
  - Cache optimization writes only dirty data to external memory in case of eviction (in 16-byte granularity, which is equal to one burst on a 16-bit DDR3 DRAM device).
  - A line can be partially valid – no need to read all 128 bytes on a miss.
  - On a 16-byte (and aligned) write-miss, the performance affect of write-allocate operation can be minimized by evicting an old clean line and filling only ECC and the relevant part of the cache line (invalidating unfilled portions of the same line).
  - Cache read optimization can read a full line or a partial line on a cache miss based on MSID configuration to optimize bandwidth for each use case.
- Address conversion and data structure support for in-band ECC on DRAM.
  - Data protected by the same ECC fields is placed on the same 1KB DRAM page. Address translation from MSID and PADDR to the actual DRAM offset reserves space for ECC without complicating the DRAM address calculation. (PADDR refers to a linear and continuous address range.)
  - On eviction of a dirty cache lines, ECC data structure enables selective writing of only modified locations without recalculating other fields. For lookup structure mode, the data modification is propagating all the way to DRAM and invalidates both local L2C copy as well as the DRAM device copy.

- Serves two channels, each having a 32-entry command queue with collision detection and out of order launch of independent transactions. The command queue is followed by a 256-entry address collision buffer per channel, tracking all launched transactions requiring L2C memory consistency management.
- Configured write allocate behavior on partial writes. With write allocate, external memory data is fetched in case of a miss and partial line write goes to the L2 cache instance. Without write allocate, a partial line write that gets a miss goes to external memory. Due to the in-band ECC such access is implemented by complex RMW DRAM access done by the DRAM controller. Write allocate operations can be done at 16-byte granularity due to multiple dirty and valid bits per line.
- Optimized write-back policy with dirty line per 16-byte granularity. Write back skips clean lines. Optimized maintenance of in-band ECC structure within the same DRAM page. All DRAM accesses are unified to one burst for maximum DRAM bus utilization. For each 128-bit line, one burst updates all 16 ECC bytes.
- Support for optimized safe writes to 16B structures or whole cache lines eliminating atomic read-modify-write merge sequences to improve performance. In safe writes, the L2 cache writes the minimal payload to EMEM and marks unwritten payload as invalid, to minimize the effective DRAM device load.
- Dirty lines can be marked by last read as don't-care, avoiding write back to DDR. This option is used on last read from dynamically allocated buffers that get recycled right afterwards and, therefore, do not carry useful data.

6.1.8.5 Increasing L2 Instance Effective Associativity

Each L2 instance can serve one or more multi-target memory spaces. It gets selected as EZnet’s target per memory space using a hash function on the group's physical address (PADDR). This architecture effectively implements very large unified L2 system, where the overall number of L2 sets (cache lines) is aggregated by all its instances. The number of ways stays the same for a single instance or multiple instances, but the depth of each way (and therefore the collision probability) is reduced. In addition, usage of the smart hash function eliminates trivial collisions caused by aliasing (for example, many data structures located in memory aligned addresses tend to access specific offsets and get aliased to the same cache lines). Instead of selecting a few low address bits for the L2 instance, all useful address bits beyond the one that selects the line are affecting the calculated cache instance. This implementation has the same effect as adding more ways to a regular cache highly-associative cache implementation.

6.1.8.5.1 DRAM Access

The DRAM does not have write strobes per byte, and therefore can only write full bursts. The memory minimum access granularity is 16 bytes, reflecting 16-bit DRAM devices with burst 8 length. Any DRAM write access to partial data requires reading of the original 16B, and writing back the merged data. This operation must be done atomically.

Each pair of 16-bit DRAM devices shares the address/control pins, and separates only the data.

The DRAM client transactions that use L2-cache are:

- CTOP SW transactions:
    - 4B single L1 uncached read
    - 1B-4B single (partial) L1 uncached write
    - 32B L1-cache line read
    - 32B L1-cache line write (all bytes valid)
    - 16B or 32B, lookup structure read accesses and update write accesses.
- Accelerators, DMAs, BMU for its own database, TM for the statistic accesses
    - (1..8)x16B read/write, 128B aligned
    - (1..8)x16B safe-write, invalidating the ECC of unwritten (1..7)x16B

The DRAM clients that use a DRAM controller directly (without L2-cache assistance):

- On-demand statistics, posted statistics
- Traffic Manager control structures and data

The tables below summarize the L2-cache resulting DRAM access for various transactions.

- *For details see In-band ECC/Parity Protection section.*

L2-cacheable space miss read transactions:

TRANSACTION	***TYPICAL USAGE***	***ECC-RAW***	***ECC- EBP 15B+1B***	***ECC LLP 128B+16B***
4B read	CTOP single. L1-uncached	16B read (allocate)	X	32B read + 16B ECC read. Allocate 128B line.
1B-4B read				
16B read	Lookup structure		16B read (allocate)	
32B read	CTOP L1-cache line. Lookup	32B read (allocate)	X	
Nx16B-128B read (N>=2)	DMAs, accelerators	Nx16B read (allocate)	Nx32B read (allocate)	Nx32B read, 16B ECC read. Allocate 128B line.

L2-cacheable space miss write transactions:

TRANSACTION	***TYPICAL USAGE***	***ECC-RAW***	***ECC- EBP 15B+1B***	***ECC LLP 128B+16B***
1B-4B write	CTOP single. L1-uncached	Read 16B. Allocate 128B cache line. Merge write to cache (one 16B chunk is dirty).	X	Line miss: 32B data + 16B ECC read. Allocate 128B cache line. Merge write to cache (one 16B chunk is dirty). Line hit / 16B chunk miss: Read 32B. Merge write.
16B (aligned) write	Structure update	Line miss: Allocate 128B cache line. Write to cache (one 16B chunk is dirty). Line hit/ 16B Chunk miss: Write to cache (one 16B chunk is dirty).		

TRANSACTION	TYPICAL USAGE	ECC-RAW	ECC- EBP 15B+1B	ECC LLP 128B+16B
32B (aligned) write	CTOP L1-cache line eviction	Allocate 128B cache line. Write to cache. Two 16B chunks are dirty.	X	Line miss: Read 16B ECC. Allocate 128B line. Line/Chunk miss: 32B write to cache (dirty).
Nx16B..128B write. (N>=2). First and last 16B chunks can be partial.	DMA	If First or Last 16B chunks are partial: Read missing chunks. Allocate 128B cache line. Merge write to cache (Nx 16B chunks are dirty).		Line miss: Read 16B ECC. Allocate 128B line. If first or last 16B chunks are partial: Read missing chunks. Nx16B write merge to cache (dirty).
Nx16B-128B write safe		Allocate 128B cache line. Merge write to cache (Nx 16B chunks are dirty).		Line miss: Allocate 128B line. Write Nx16B to cache (Nx 16B chunks are dirty). Invalidate non-written ECC chunks.
L2-cache eviction 128B		Discard clean or write dirty (1..8) 16B chunks up to 128B.		Discard clean or write dirty (1..8) 16B chunks up to 128B. Write 16B ECC.

L2-cache uncached read transactions:

TRANSACTION	TYPICAL USAGE	ECC-RAW	ECC- EBP 15B+1B	ECC LLP 128B+16B
4B read	CTOP single	16B read		16B read. 16B ECC read.
16B read	Lookup structure			
32B read	CTOP L1-cache line	32B read	X	32B read. 16B ECC read.
Nx16B..128B read	DMA	Nx16B read.		Nx16B read. 16B ECC read.

L2-cache uncached write transaction:

TRANSACTION	TYPICAL USAGE	ECC-RAW	ECC- EBP 15B+1B	ECC LLP 128B+16B
1B-4B	CTOP single	16B Read-Modify-Write		16B write, 16B ECC RMW
16B write (aligned)	Lookup structure	16B write		
32B write (16B aligned)	CTOP L1-cache line	32B write	X	32B write, 16B ECC RMW
Nx16B..128B write (16B aligned)	DMA	Nx 16B write		Nx 16B write 16B ECC RMW
Nx16B..128B write safe.				Nx 16B write 16B ECC write

L2-cache bypass write transaction:

TRANSACTION	ECC-RAW	ECC- EBP 15B+1B	ECC LLP 128B+16B
Nx16B..128B write safe	Nx 16B write		Nx 16B write 16B ECC write safe

Lookup structure with embedded parity protection.

TRANSACTION	TYPICAL USAGE	PARITY 3-BIT	
16B/32B read N*32B	Lookup structure read	16B/32B N*32B	Parity error does not get fixed and does not cause bus error exception - instead it informs the CTOP lookup access that SW recovery sequence is required.
16B/32B read N*32B	Lookup structure update	16B/32B N*32B	Lookup structure writes propagate all the way to DRAM and invalidates local copy in the accessed L2C instance as well as the accessed DRAM cache instance.

6.1.8.6 Address Translation from PADDR Real Address

The original physical address (PADDR) and MSID are converted into a DRAM interface ID and offset, which are used by the crossbar to steer the access toward the proper DRAM interface.

Each time there is a miss in L2 cache access and external memory has to be accessed, the L2 cache instance must select the appropriate DRAM controller target to carry out the access and its address offset. The platform physical addresses maximum range (PADDR) has a per MSID configuration.

L2-cache spreads transactions over one or more DRAM interfaces (configured per MSID) with load balance by address scrambling on each round.

Each MSID is configured per DRAM device with base address and memory size (granularity of 8KB multiplied by number of devices).

At initialization the system must be assured to avoid overlaps where inappropriate. In another scenario, different memory spaces may share access to certain memory regions, holding for example shared lookup tables, global application data, or shared code. Aliasing two memory spaces to the same location also enables combining EMEM data structures, such as frame data, each using a different ECC protection scheme and accessing a separate non-overlapping portion of 256B frame buffer.

DDR device and bank are provided explicitly in the address of dynamically managed memory spaces, facilitating load balancing in EMEM buffer allocations and usage.

6.1.8.7 In-band ECC/Parity Protection

The DRAM interface supports in-band ECC protection instead of adding a dedicated DRAM device for ECC. The ECC bits are embedded in the DRAM’s standard data stream. Burst accesses of cache misses (line evictions) require reading (or writing) the ECC bytes as part of the atomic burst.

The ECC mechanism configured per L2-cache instance/MSID:

- RAW data (protected by client). Used by statistic counter structures, bitwise operation structures, and Lookup structures. The ECC is generated and checked by the accelerator engine generating the transaction.
- EBP – Explicit Burst Protection: self contained 16B structure with ECC on MS byte and 15B of data. Used by data-plane SW-managed structures. The empty ECC byte is explicitly seen by SW as a reserved location. EBP may be mapped as an independent space, or an overlapping space with LLP (for the case of frame data). The EBP data structures enable random access to 16 byte of 16/32 byte aligned words directly in DRAM. All transactions refer to full 16B chunks and no merging is done. Full 16B chucks are written either to L2C (cached) or directly to DRAM (uncached). TM Queue Descriptors (QD) and Packet Descriptor (PD) use the EBP protection scheme.
- LLP – Linear Line Protection: virtualization of continuous and linear memory space where real memory cache lines are arranged as 16B ECC chunks and 128B lines of data, all in the same bank and same row. Each 1KB DRAM block in the same row and bank holds seven data lines and seven ECC chunks associated with those lines. Used by SW code and data memory spaces.
- EBP over LLP – self protected 16B structures with address translation of LLP structures. Used for frame first data buffer sharing with embedded LBD structure (ELBD) where both ECC protection schemes coexist on the same 128B line.
- Lookup structures are supported with 3b parity per 16B/32B entry or 3b per 32B. Parity errors do not get fixed and do not cause bus error exceptions; instead it informs the CTOP lookup access that a SW recovery sequence is required.

6.1.8.7.1 Linear Line Protection

The LLP solution defines a long cache line (128-byte) while keeping per 16-byte granularity for valid and dirty bits. Each 1KB DRAM page contains one ECC protection block and seven 128-byte cache lines. Each 16 bytes in an L2 cache line have their own valid and dirty bits and their own 9-bit ECC in the ECC chunk.

On a read-miss, the 128 bytes are fetched from DRAM and the 16 ECC bytes protecting the full 128-byte line are fetched as well, all in one atomic burst access. The read data, with the ECC validity bit set, is verified and single-bit corrected by the relevant two ECC fields and all the 16 ECC bytes are stored in L2 in the pre-existing ECC locations. The bytes that were stored in L2 from the first access are used to verify and correct the cache line data. The overhead of the ECC protection block is (16B/128B=12.5%). The 128B L2-cache line fill portion can be controlled to be more than 32B at a time (e.g. 64B).

Write miss to a cache line is handled by the write-allocate scheme. If the cache line already exists in L2 (meaning that all 16 ECC bytes are valid) but the specific address of a short write belongs to an invalid line portion, 32 bytes are fetched and the local data is updated in the L2 cache. If the write data gets a full cache line miss, an existing cache line is selected for eviction based on priority and LRU between same-priority lines. The evicted line may cause selective write-back of dirty data and includes the updated ECC block in the burst. In case there is no dirty data to update, the line is simply invalidated and gets reassigned to the new address containing the write address. Then the 16 ECC bytes and relevant 32 bytes (or other amount) of the line are fetched (the bytes that overlap all write data) and the line becomes partially valid in the L2 cache. The specific 16 bytes that hold the written data are updated with the new data (merged with original contents) and marked as valid and dirty. Bytes that are overwritten are not fetched from EMEM. Other portions of the long cache line are either marked as valid and clean (in case they were fetched) or invalid (in case they were not fetched). Write-back of partially dirty line supports atomic burst pattern with non-continuous 16B portions.

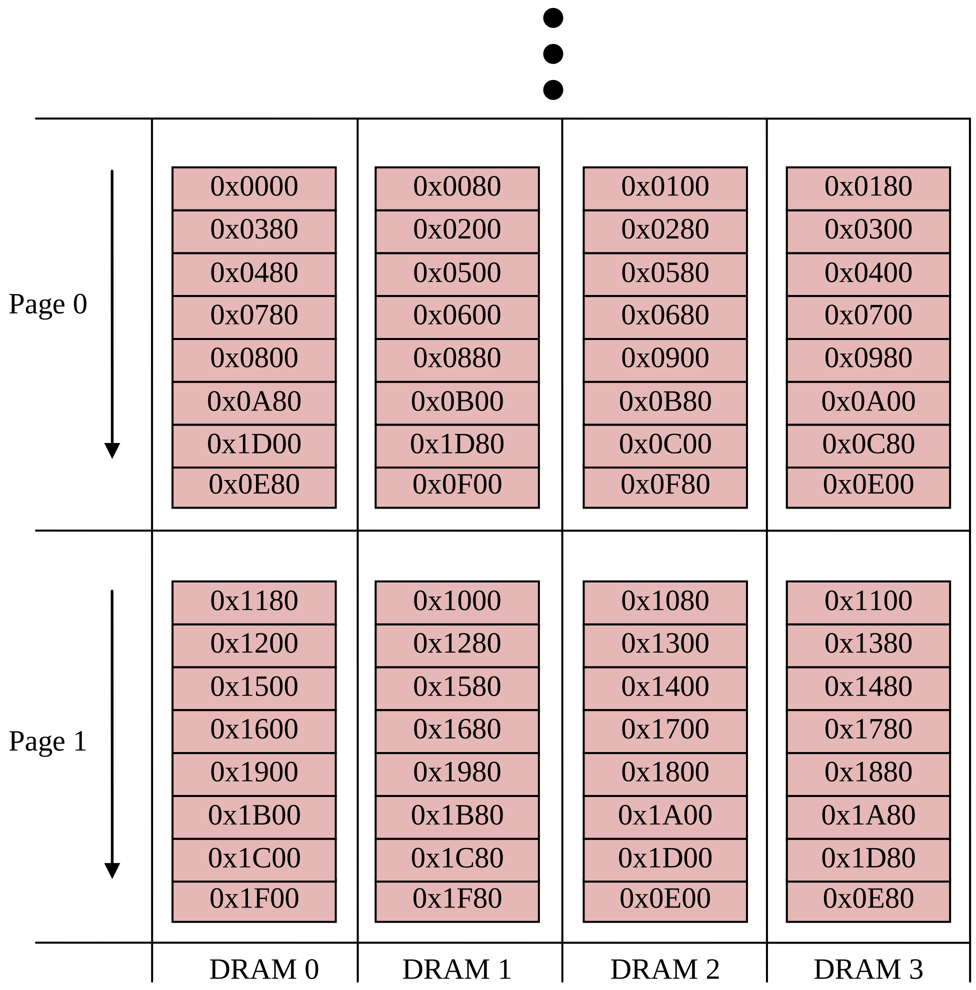
Special mapping from the MSID and the physical address offset select the external DRAM interface and its offset. It is ensured that at all cases, the same 128 bytes and their 16 bytes of ECC protection get mapped to the same DRAM device and the same row.

6.1.8.7.2 DRAM Data Organization of EBP (1B+15B) and RAW Data Formats

In the EBP data protection scheme, the ECC is embedded in 16B DRAM data structures while in RAW data format it is in the hands of the client. In either case, there is no overhead of separate ECC protection data structures. Therefore 1KB DRAM blocks can be fully utilized by the same MSID data and continuous blocks in DRAM address space may serve the same MSID without interleaving data from different MSIDs on the same DRAM blocks. One exception is where EBP protection is programmed to be aliased to the same pages as LLP for implemented frame data buffer structures. In this case, the DRAM address is mapped such that ELBD/LBD structures utilize the same buffers as frame data, using only seven lines out of eight in each 1KB block, and using the same address thresholds used to map bank and offset in the LLP DRAM device offset.

Figure 6-16 shows an example of one memory space spread over four DRAM interfaces. Each horizontal line represents continuous 128 bytes that occupy a full L2C cache line. The MSID lines fill all 1KB pages on offsets 0 to 7, and after eight rounds naturally advance to the next DRAM 1KB block. Even though a power of two number of DRAM interfaces is used, this figure does not show the results of address scrambling which may change the relative ordering of lines in each line offset (i.e. every four lines the DRAM interface internal ordering may vary and the next four lines are differently spread over the interfaces).

Figure 6-16. DRAM Page Organization Example for Raw/EBP/SSP, one MSID and four DRAM interfaces



Page 0
0x0000 0x0080 0x0100 0x0180
0x0380 0x0200 0x0280 0x0300
0x0480 0x0500 0x0580 0x0400
0x0780 0x0600 0x0680 0x0700
0x0800 0x0880 0x0900 0x0980
0x0A80 0x0B00 0x0B80 0x0A00
0x1D00 0x1D80 0x0C00 0x0C80
0x0E80 0x0F00 0x0F80 0x0E00
Page 1
0x1180 0x1000 0x1080 0x1100
0x1200 0x1280 0x1300 0x1380
0x1500 0x1580 0x1400 0x1480
0x1600 0x1680 0x1700 0x1780
0x1900 0x1980 0x1800 0x1880
0x1B00 0x1B80 0x1A00 0x1A80
0x1C00 0x1C80 0x1D00 0x1D80
0x1F00 0x1F80 0x0E00 0x0E80
DRAM 0
DRAM 1
DRAM 2
DRAM 3

6.1.8.7.3 EBP over LLP Protection of Frame Buffer

EBP over LLP self protection of 16B structures with address translation of LLP structures. It is used for the frame’s first data buffer sharing with an embedded LBD structure (ELBD) where both ECC protection schemes coexist on the same memory space. EBP is used to protect the LBD/ELBD portion of the 256B frame buffer.

The frame buffer is accessed via MSID with LLP protection. The LBD/ELBD is accessed via different MSID with EBP protection attributes in L2C address translation that points to same physical buffer.

In case of an ELBD frame, it uses the same FD pointer of the first buffer and reaches the leading 16B/32B of this buffer (holding up to 3 or 6 BDs). The same buffer is accessed twice through two MSIDs – one for writing the data and the other for writing the BDs.

In case of an LBD frame, it gets a unique pointer and fills a 256B buffer with up to 48 BDs using EBP protection.

From a memory resource perspective there is no difference between pure data buffers with LLP protection, pure LBD buffers with EBP protection or mixed first buffers (ELBD frames) with EBP over LLP. They are all equal resources allocated from the same pool.

6.1.8.8 DRAM Data Organization of Lookup Structures (3-bit Checksum per 16B/32B)

Lookup structures are organized in tables of elements with predefined sizes. Search Structure Protection (SSP) includes a 3-bit checksum located in the leading bits of 128-bit or 256-bit structures. The actual protection coverage is determined by the transaction size, assuming that 256-bit structures and above have an even number of data flits, and are accessed at their full width. Failing to match the correct element width would result in a wrong checksum on writes or false checksum errors in reads.

The 3-bit checksum is able to detect all single bit errors, and has probability of 7/8 to detect multi-bit errors in the search structure with random location distribution.

In case of SSP error detection, a non-fatal bus error is reported back to the initiator SW application, and the lookup operation is rejected based on it. Data plane threads may continue to work with other copies (in case the structure is duplicated), while one control plain thread is responsible to fix the faulty structure (either by copying the data from a good duplicate, if exists, or by recovering it from an independent database otherwise).

The implementation of the fix sequence for search structure is either based on having an alternative good copy from duplicated tables, or recovering it from another database. Where duplication exists, other threads that got the bad copy and do not participate in the fix sequence may explicitly access other good copies while one thread implements the fix.

As a general guideline for the SW fix procedure, a thread which reads corrupted data tries to lock a semaphore for fixing it. In case of success, it continues with the fix. In case of failure (meaning another thread is already fixing the entry), it may choose to explicitly read from another good copy (if exists), or retry reading again at after some delay (in hope of either being directed to another good copy), and eventually reading from the original which has already been fixed by the control plane. For data structures with high access rates this behavior enables smooth recovery of corrupted lookup structures while the system is operational using the remaining structures at some performance degradation.

During the structure update process, SW writes explicitly to all the copies of duplicated lookup structures. It must write in one burst the full width of the lookup structure. Writing to a 16B structure implements 3-bit protection over the remaining 125 bits. Writing to a 32B structure implements 3-bit protection over the remaining 253 bits in each 256-bit aligned location. Writing to a structure sized as multiple 32B portions, implements 3-bit protection per 256-bits, repeated over each 32B.

Typically search structures are arranged in a way that accesses different L2C instances and different DRAM devices for sequential structures. Additionally, reading or writing a search structure accesses exactly one element (due to being located on a separate EZnet target).

6.1.9 Data Coherency

The IMEM and L2-cache system do not duplicate data and do not use snooping mechanisms. There are two mechanisms to keep memory coherency on shared data:

- Placing shared data in the L2 cache in an address space that is not cacheable in CTOP L1-cache.
- SW based coherency in write-back write-allocate L1-cache scheme.
- Duplicated lookup structures require a sequence of writes to all duplications in L2C and all duplications in DRAM devices. The sequence is repeated twice to ensure that both hierarchies are updated.

6.1.9.1 Shared Data in L2/IMEM not Cached in L1

This scheme does not require HW coherency support for L1-caches. CTOP transactions bypass L1 cache and go directly to the shared location which is either distributed among L2 cache instances (EMEM) or LRAM instances (IMEM).

The L1-cachability is controlled by CTOP TLB and FMT address translations for user/kernel virtual space and by cacheable/uncacheable kernel spaces.

EZnet’s low latency and high bandwidth combined with core/thread tolerance to data access latency enables efficient core execution under reasonable shared data bandwidth.

L1-cache support of hit under miss further improve their performance by allowing the code to continue as long as there is no register dependency on a load result. Explicit or automated thread scheduling due to long EMEM access latency enables other threads to use the pipeline hiding the L2 cache access latency.

6.1.9.2 Software Based Coherency

With CTOP L1-data cache enabled memory space, the core data cache may also work in an incoherent write-back mode. This mode is appropriate for working on temporarily owned data (private for a certain time). At the end of the execution it is SW’s responsibility to invalidate the data and write back dirty data to IMEM or the L2 cache system. The task can move forward to the next processing stage after ensuring all data is resident in L2 and invalidated in L1 (using a data SYNC command).

6.1.9.3 L2 Shared Data Coherency

Each address in a memory space physical domain gets mapped to one and only one location in a single L2 instance assigned to it. The mapping granularity is 128B ensuring that a single L1-cache line’s data is always placed in the same L2 instance.

When shared data is located in the L2 cache and not located in the core’s L1 caches, it is by definition coherent in the eyes of all users. When data has a copy in any core's L1 cache coherency it has to be ensured by SW.

A CTOP thread that accesses shared data experiences increased latency that can be covered by other active threads, and by the core pipeline that continues executing independent commands.

Data that requires high locality for certain period can be moved to private variables during its processing time and then copied back to the shared area residing on L2 cache, to be used by other cores. Hardware accelerators and DMAs can offload the core from most of this burden by preparing the required data for the receiving core before activating the thread.

The L2C supports PREFETCH commands that explicitly cache data that is expected to be needed soon.

Additional L2C cache management commands are write-back-invalidate, and flush-invalidate.

6.1.10 Memory System Access Ordering and Consistency

EZnet interconnection to IMEM and EMEM distributed memory system defines multiple possible targets for a transaction source based on the transaction's physical address and its attributes. Access to the same address and the same group attributes is assured to target the same EZnet node and IMEM or L2 cache instance. Access requests and responses to two different addresses can be mapped to two different IMEM banks or L2 cache instances and may arrive to/from these memories out of order on any routing scheme, either deterministic or adaptive.

When addressing the same target IMEM or L2 cache instance, the deterministic EZnet route scheme assures memory consistency without requiring address collision detection at the requester side.

The sequence of access requests from one source to the same IMEM bank or L2 cache instance runs through the same network path and uses the same physical request channel, and therefore is assured to arrive in the same order at the target node. IMEM bank and L2 cache implementation assures memory consistency on colliding addresses, but is allowed to be executed out of order responses to non overlapping locations. As a result, responses from the same target may be returned to the initiator out of order. Responses can return out of order also on the same address; the order is maintained for the requests and execution due to adaptive response path.

For example, two read access requests to the same L2 instance arrive to that instance in their issuing order. However, L2 parallel banks and deeply pipelined implementation combined with a cache miss enable the second read transaction to get its read data before the first read's data is available. The CTOP and shared accelerators and DMAs support out of order transaction replies.

If in the same scenario an L2 cache detects address collision, then by definition if one access got a miss the other one which addresses the same cache line would stall for the same data to arrive. Both accesses would go to the same L2 bank and cache line and therefore are assured consistent read order. Even when the target initiates both transactions in order, EZnet's response path is adaptive (may switch to alternative physical channels) and the responses may arrive at the initiator out of their issued order.

For L2-cached data memory consistency is resolved by caching data with write-back policy. In case of partial write with cache miss, a read-allocate operation with 16B granularity fetches the line (or portions of the line), and then the partial write is merged into the cache line locally. Later on when the line is evicted to DRAM it has 16B granularity.

Out of order responses may also be the result of selecting different EZnet nodes for independent addresses. Read data from each node arrives independently to the other nodes, and therefore at any order relative to them.

Similarly two sequential write transactions from one source to the same L2 target would arrive in order and L2 implementation will use address collision detection for assuring memory consistency.

If a read command is accepted at the L2 instance after an older write command to an overlapping address, the read command will be stalled in the L2 pipeline until the write data is available. Similarly when a write transaction to EMEM arrives to the L2 cache, it does not have wait until transaction is actually received by the external memory device. It can respond immediately after accepting the request and close the write with improved latency.

The L2 architecture assures that any Write After Read (WAR), Write After Write (WRW) and Read After Write (RAW) to an overlapping address completes in correct program order at all cases. This scheme always works for the same core without requiring SYNC commands and provides high and scalable performance. Between cores using SYNC commands, the writing core waits for L2 consistency point confirmation to ensure correct order when signaling data readiness to another core through other means. (The CTOP threads use posted writes for improved performance. The data SYNC command ensures that all posted write transactions initiated by the thread are related to the thread have been accepted by the memory consistency point.)

6.1.10.1 Memory Consistency with Posted Writes and Reads

The consistency of data passed between cores/DMAs can be enforced by posted writes and reads that are synchronized via SW instructions SYNC, SYNC.RD, SYNC.WR, SCHD.RW, SCHD.RD.

The CTOP synchronization commands track execution acknowledges. The SYNC.* synchronization commands are executed by the CTOP execution pipe while SCHD.* commands are executed by the MTM thread scheduling.

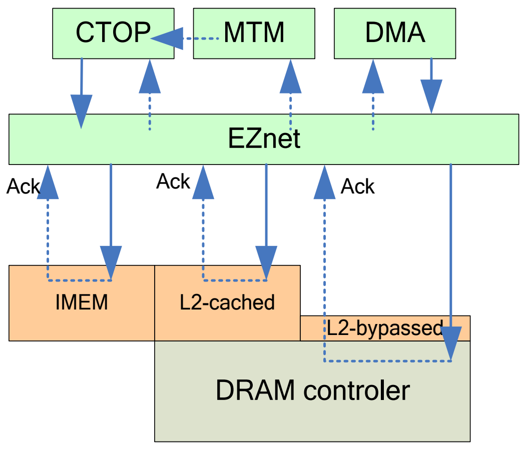
The CTOP and MTM rely on IMEM and L2-cache for command acknowledgements for consistency synchronization.

The IMEM and L2-cache implementation minimizes the latency of write acknowledgement by taking over the memory consistency immediately after accepted the write transaction. This response is used by the initiator CTOP to clear a reserved entry and minimizes the stall time involved in SYNC/SCHD commands.

The L2-cache consistency point acknowledges the write command. For cache-bypassed MSID, the consistency point is the DRAM controller and read/write ordering is not guaranteed even for the same source going to the same address unless a SYNC command is added between them. For L2-cache enabled MSIDs, the consistency point is the L2-instance.

The HW accelerators and DMAs also utilize the memory transaction acknowledgements to guarantee data consistency upon command execution.

Figure 6-17. Data consistency



SYNC.* commands:

CTOP generate posted writes and write-backs use SYNC commands for transaction completion at the consistency point. The core interface tracks all open reads/writes and keeps the core stalled in SYNC command as long as there are open read/write transactions by the SYNC requestor. In a multithreaded core implementation, the CTOP tracks open reads/writes per thread and selectively enables each thread to complete its private SYNC command according to its own open write requests.

- SYNC – CTOP execution pipe tracks incomplete reads and writes.
- SYNC.RD – CTOP execution pipe tracks incomplete reads
- SYNC.WR – CTOP execution pipe tracks incomplete writes

SCHD.* commands:

MTM execution of the SCHD.* causes the CPU to switch the executing HW thread with another thread context. In addition, the type of schedule used determines the criteria for thread's "eligibility for execution" composed by the scheduler. The thread's scheduler will set the thread to be "eligible for execution" when the requested criteria are met.

- SCHD.RW – MTM tracks incomplete reads and writes.
- SCHD.RD – MTM tracks incomplete reads.

6.2 Unified Data Memory

The UDM is mapped to the CTOP user/kernel translated region. The 1MB virtual address range, 0x57F00000 to 0x57FFFFFF, is allocated for the Unified Data Memory range. This address range is mapped to CTOP 16KB UDM. The UDM is shared by CTOP threads. The UDM is accessible by CTOPs, sub-cluster DMAs and accelerators.

The UDM is partitioned into three regions:

- Per thread private region of Data Scratchpad
- CTOP shared region of Data Scratchpad
- CTOP Data L1-cache (2 way). The UDM is used by data-cache controller for data storage. The cache tags are stored in dedicated memory.

6.2.1 UDM Partitioning

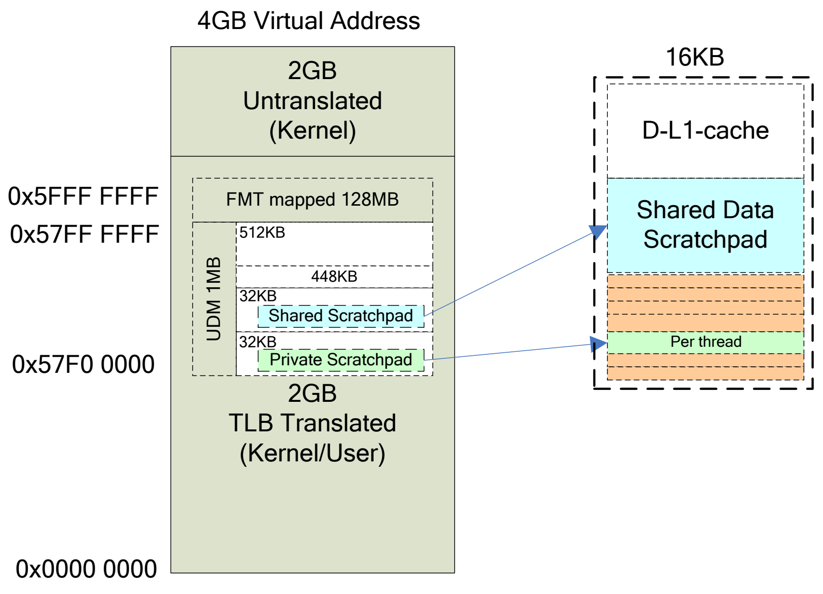
At boot, the kernel configures the UDM partitioning (via UDMC register).

The configuration options are:

- The number of UDM threads for private region and Data cache sharing is configurable: 1, 2, 4, 8, or 16.
- UDM memory allocated for the thread private region of Data Scratchpad: 0, 1KB, 2KB, 4KB, 8KB, or 16KB. This region is divided among active threads. The Data Scratchpad virtual base is 0x57F00000 for each thread. The Data Scratchpad is allocated a 64KB region.
- Data Cache thread-private/shared mode:
  - L1-Data cache can be shared among threads with affect of inter-thread trashing of the cache. The data cache size is selectable in power of two jumps from 128B up to the entire 16KB.
  - Data cache is thread private. Data cache can be configured to be split per active thread, or unified for all the active threads (active threads are selected in power of two steps from single thread, two threads up to all 16 threads).

The UDM not allocated to Cache nor private of Data Scratchpad is used as Shared Data Scratchpad. The shared region of Data Scratchpad base address is 0x57F08000.

Figure 6-18. UDM partitioning



7. Input Classification Unit (ICU)

The NPS-400 contains an Input Classification Unit (ICU) which performs hardware-based classification of each of the arriving packets, allowing the differentiation of packet priorities/CoS as well as assignment of the packet to a physical queue (PQ) in the PMU.

The ICU classified packets are scheduled to RxNDMA via per port RxFIFOs.

The RxFIFO Scheduler uses the ICU classification.

The Scheduler may drop a received packet based on source port associated resource congestion. The IMEM/EMEM/PMU resources are monitored for congestion by the FCU. The RxFIFO packet drops are performed before packet memory storage.

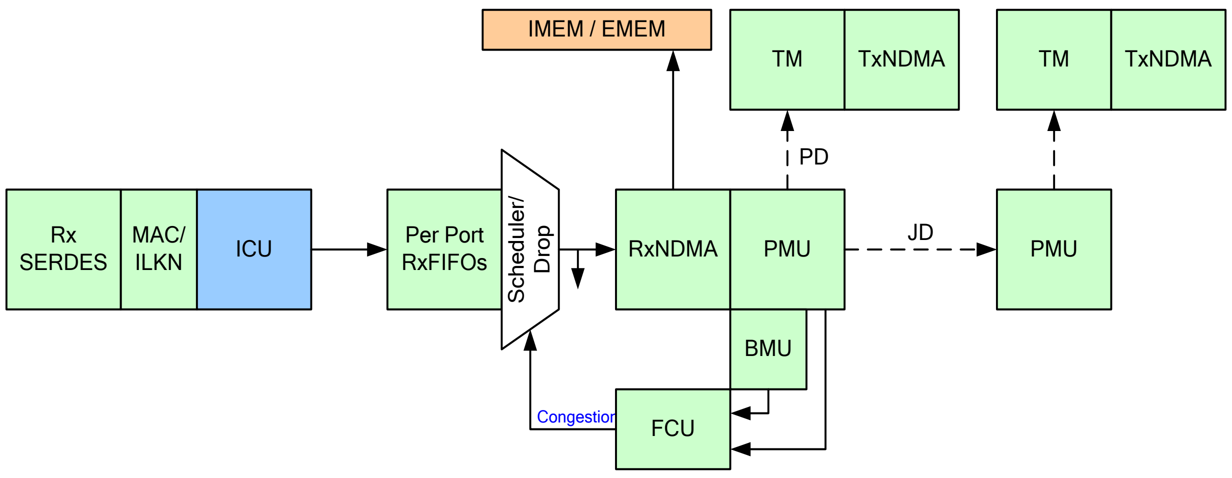
The RxNDMA subsequently buffers the packet data into internal and external memory.

The RxNDMA generates a Job Descriptor (JD) for each buffered packet.

There are six ICU units.

- West ICU 0: Interface lanes 0-23.
- West ICU 1: Interface lanes 24-47.
- West ICU 2: Interface lanes ILKN.
- East ICU 0: Interface lanes 0-23.
- East ICU 1: Interface lanes 24-47.
- East ICU 2: Interface lanes ILKN.

Figure 7-1. Incoming packet path



7.1 Packet Parsing and Classification Overview

The ICU performs classification of incoming packets based on the following parameters:

- Packet physical source port or Interlaken channel. Global and per source port configured options.
- Packet format as parsed by configurable parser.
- Optional packet extension header attached by previous device.

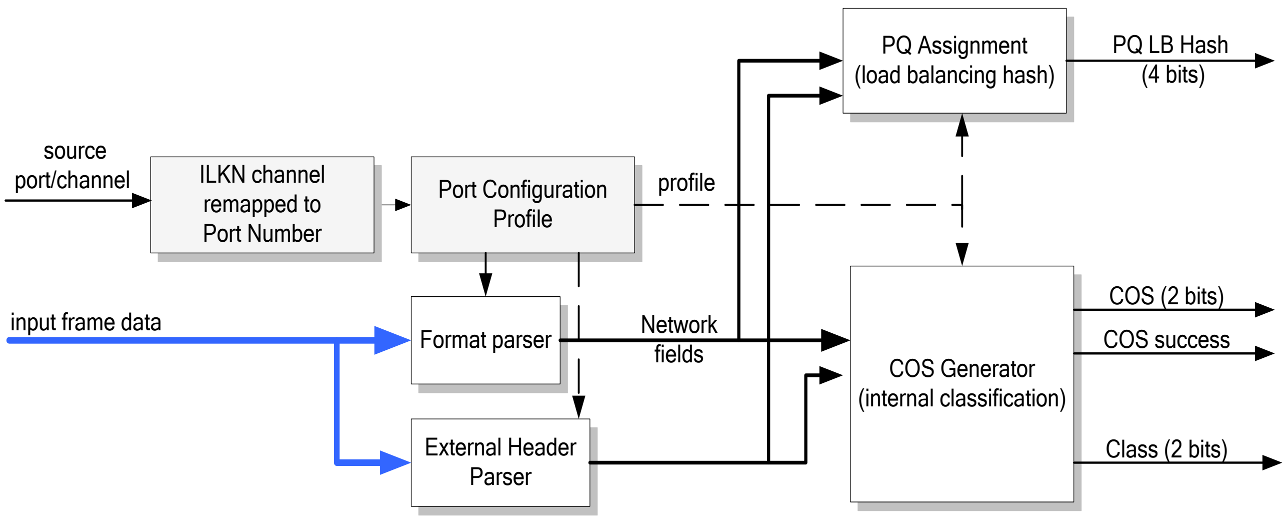
The results of classification include:

- Class of service (COS; 2 bits) – initial class of service (strict priority) assignment.
- Success of Classification – the COS classification result is non port default.
- Class (2 bits) – the Class assigned per L2 packet classification.
- PMU Physical Queue load balancing (4 bits) – the load balancing hash result.

The ICU HW Decoder supports frame data offsets of 4B, 8B, 16B, 32B, 48B and 64B to bypass fabric headers on egress line cards or other proprietary headers prepended to packets. The header offset for decoding is configurable per source port.

The following figure shows the ICU data flow:

Figure 7-2. ICU data flow



7.1.1 Class of Service (Priority) vs. Class Assignment

The ICU assigns both COS and Class.

The Class is based on source port and L2 packet classification to distinguish L2 channels, i.e. VLANs, tunnels, etc. The Class is used for resources assignment and management. The Class assignment can be derived from COS.

The assigned COS is strict priority that is assigned per packet encapsulation parsing for Priority/EXP/ DSCP and various L2/L3/L4 control protocols identification.

The COS priority is used by oversubscription management for drop precedence and various priority based scheduling decisions.

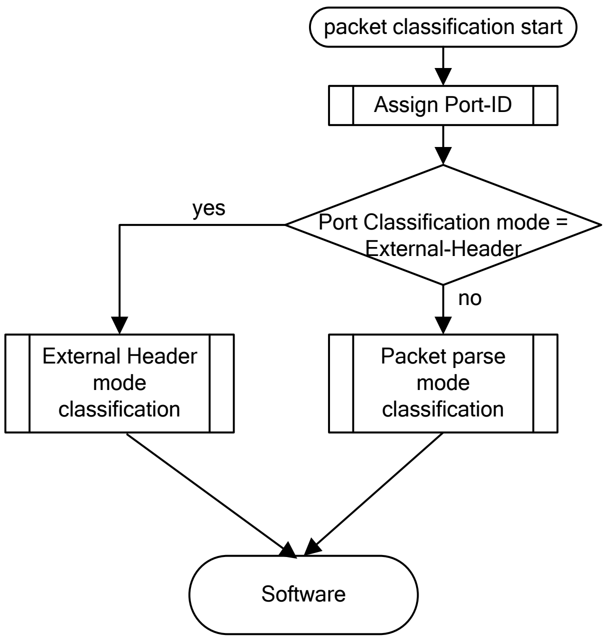
7.2 Packet Classification Modes

The source Port ID supports the selection of a packet classification mode.

The ICU enables the following classification modes:

- External header mode – Classification result are extracted from the packet extension header.
- Packet parse mode – Classification is performed by parsing packet fields. Interface number is defined by physical source port number (port mapping table is used).

Figure 7-3. Classification modes


packet classification start
Assign Port-ID
Port Classification mode = External-Header
yes
no
External Header mode classification
Packet parse mode classification
Software

The internal or external classification is configured per source interface and Interlaken channel.

- See section 7.4 for Packet parse mode Classification
- See section 7.5 for External Header mode Classification

7.3 Interlaken Interface Channel Mapping to Port Number

Interlaken interface channels (0-128) are is assigned to one of 16 configuration profiles. The profile configuration is same as per Port ID configuration.

7.4 Packet Parse Mode Classification

Classification is performed by parsing packet encapsulation network layer headers.

Port number is a function of physical source port (port mapping table is used).

7.4.1 Supported Frame Types

The frame may contain an additional proprietary (fabric) header which is skipped. Header size is configured per source port ID and can be 4, 8, 16, 32, 48 or 64 bytes.

The following types of Ethernet frames are detected:

- Layer 2 control frames.
- VLAN tagged frames. Up to 3 nested VLAN tags.
- MPLS multicast and unicast frames. Up to 3 MPLS labels stack.
- Ethernet frames without VLAN tag
- IPv4 without options
- IPv6
- TCP over IPv4
- UDP over IPv4
- Layer 3
- Layer 4

7.4.1.1 Frame Formats

Ethernet frame without VLAN tag

DA (6 bytes)	SA (6 bytes)	Type/length (2 bytes)

Ethernet frame with VLAN tag

DA (6 bytes)	SA (6 bytes)	0x8100,0x9100,0x9200,CFG	VLAN tag	Type/Length

VLAN tag 16b:

Priority (3 bits)	CFI (1 bit)	VLAN_ID (12 bit)

ARP

DA (6 bytes)	SA (6 bytes)	0x0806	ARP payload

MPLS over Ethernet

DA (6 bytes)	SA (6 bytes)	0x8847,0x 8848	MPLS header

MPLS header 32b

Label (20 bits)	EXP/COS (3 bits)	S (1 bit)	TTL (8 bits)

IPv4 header 5*32b.

Version (4 bits)	Header Length (4 bits)	Differentiated Services Code Point Explicit Congestion Notification (8 bits)	Total Length (16 bits)	
Identification (16 bits)			Flags (3 bits)	Fragment Offset (13 bits)
Time to Live (8 bits)		Protocol (8 bits)	Header Checksum (16 bits)	
Source IP Address (32 bits)				
Destination IP Address (32 bits)				

- The ICU validates the IPv4 Version and uses the Differentiated Services Code Point Explicit Congestion Notification.
- All other fields are ignored.

IPv4 TCP/UDP header

Source Port (16 bits)	Destination Port (16 bits)

- Other fields are ignored including checksum.

IPv6

Version (4 bits)	Traffic Class (8 bits)	Flow Label (20 bits)

- The ICU validates the IPv6 Version and uses the Traffic Class for COS assignment.
- Other IPv6 header fields are ignored.

7.4.2 Frame Format Parsing for COS Assignment

The ICU supports parsing of the frame formats listed below:

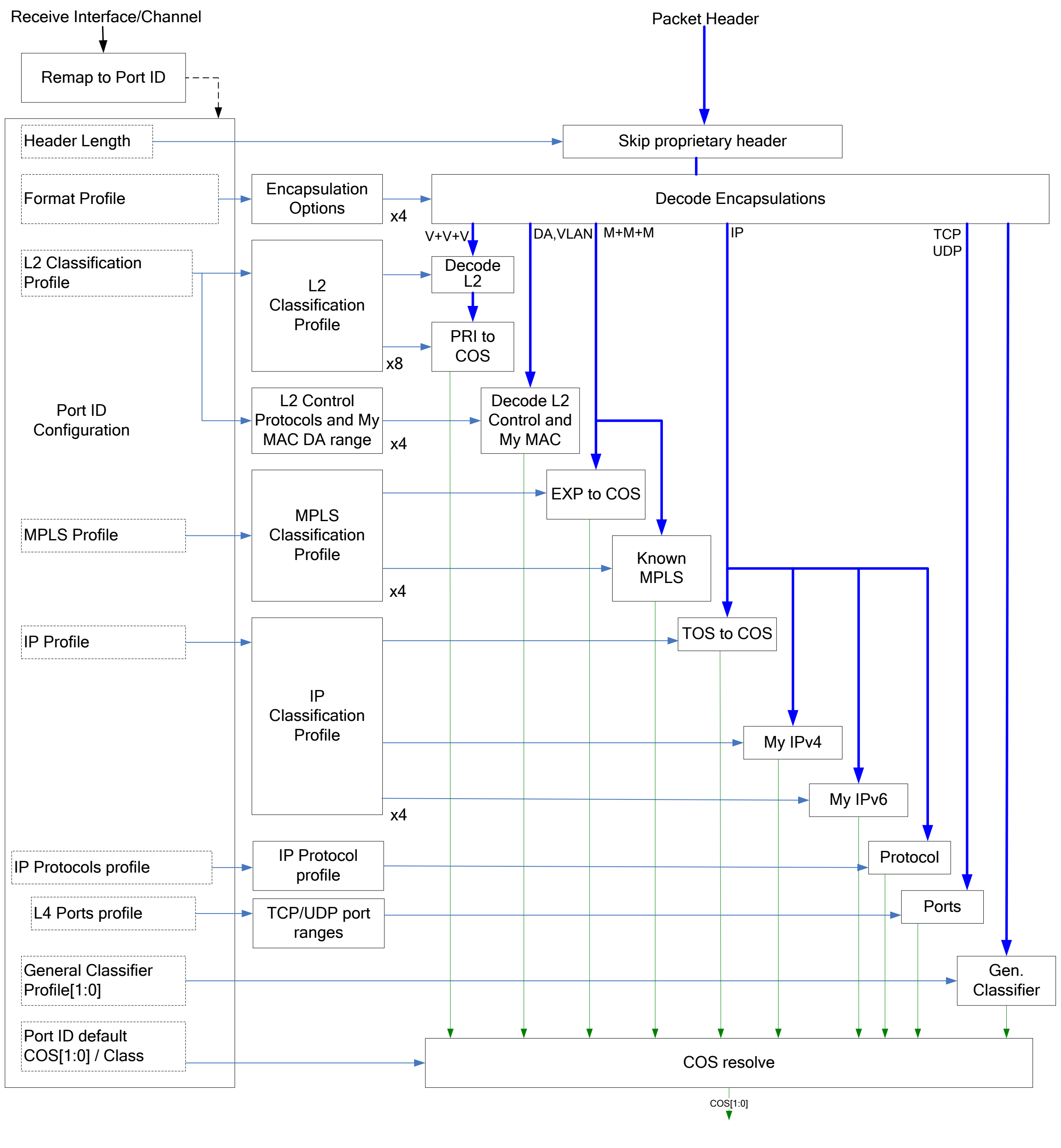
FORMAT	***COS ASSIGNMENT***
Ethernet (untagged)	Per known Ethertype configuration or Default per source port ID Override by general rules match.
ARP	
Ethernet VLAN tag	User Priority (PRI)
Ethernet VLAN + VLAN	User Priority (PRI) of either tag
Ethernet VLAN + VLAN + VLAN	User Priority (PRI) of either tag
Ethernet untagged MPLS 1 label	EXP
Ethernet untagged MPLS 2 labels	EXP of either label
Ethernet untagged MPLS 3 labels	EXP of a label
Ethernet VLAN tag MPLS 1 label	User Priority (PRI) and EXP
Ethernet VLAN tag MPLS 2 labels	User Priority (PRI) and EXP of either label
Ethernet VLAN + VLAN + MPLS 1 label	User Priority (PRI) of either tag and EXP
Ethernet VLAN + VLAN + MPLS 2 labels	User Priority (PRI) of either tag and EXP of either label
Ethernet (untagged) IPv4	DiffServ TOS
Ethernet VLAN tag IPv4/IPv6	User Priority (PRI) and DiffServ/TOS
Ethernet VLAN + VLAN + IPv4/IPv6	User Priority (PRI) of either tag and DiffServ/TOS
Ethernet VLAN + VLAN + VLAN + IPv4/IPv6	

7.4.3 Internal Classification Flow

The figured classification flow resolves packet header encapsulation based on COS assignment.

The figured configuration tables and profiles range are for a single ICU.

Figure 7-4. Internal classification flow



7.4.4 Per Port ID Configurations

The Port Configuration table contains per Port ID configurations. Each ICU supports 80 configuration entries (one entry per assigned Port ID). The table is run-time configurable.

Each entry contains the following fields:

Table 7-1. Per Port ID configurations

USAGE	*NAME*	*# BITS*	*DESCRIPTION*
Header parsing and skip	Mode	1	Classification mode: 0 – Packet parser; 1 – External header
	Header length	3	Header length: 000 – no header 100 – 16 bytes 001 – reserved 101 – 32 bytes 010 – 4 bytes 110 – 48 bytes 011 – 8 bytes 111 – 64 bytes
	External header classification profile	2	External header classification profile
	Default COS	2	COS default value
	Use default COS	1	Use only default COS (disable parsing)
	Default Class	2	Class default value
	Use default Class	1	Use only default Class (disable parsing)
	COS to Priority profile	2	COS mapping to priority profile
Format parsing for COS assignment	Format Profile	2	Format profile for encapsulation options. Refer to Format Encapsulation Options section.
	L2 Classification Profile	3	L2 Ethertype and known VLAN ID. Refer to Layer 2 Classification Profiles section.
	L2 My MAC Enable	6	L2 MY MAC DA range enable bitmap. Refer to My MAC frame detection section
	MPLS Profile	2	Configuration profile used for MPLS parsing. Refer to MPLS Classification Profiles section.
	IP Profile	2	Configuration profile for known IPv4/IPv6 destination addresses range.
	L3 Control profile	2	L3 Control packets profile
	L4 Control profile	2	L4 Control packets profile
General Classifier	General Purpose Classifier Enable	1	Enable general classifier TCAM for last Ethertype. Refer to General Purpose Rules Classifier section.
	General Purpose Classifier Profile	2	General classifier TCAM profile for masking rules. Refer to General Purpose Classifier Profile section.
PMU PQ load balance	PQ load balance hash size	3	Load balance hash size of 0 - 4 bits
	PQ load balance profile	2	Profile selector for hash fields Refer to PQ Hash Profile section.
	Load balancing shim header offset and length		Load balance (LB) based on packet header selectable between 0-127 bytes offset with configurable LB field size 1 - 6 bytes.

7.4.5 Format Encapsulation Options

The profile is assigned per Port ID via Format Profile [1:0] for encapsulation options.

The profile enables various encapsulation options ('COS Enable' column) and selects COS mapping profiles, Layer-2 classification, MPLS classification and IP classification profiles.

- The 'COS Enable' column determines which encapsulation affects the COS result.
- The 'COS Mapping Profile' column indicates the mapping table to use.
  The V[1:0] selects VLAN Tagged Frame COS Classification profile - see 7.4.13
  The M[1:0] selects MPLS Frame COS Classification profile – see 7.4.14
  The IP[1:0] selects IPv4/IPv6 COS Classification profile – see 7.4.15

Table 7-2. Encapsulation options

ENTRY	*ENCAPSULATION*	*COS ENABLE*				*COS MAPPING PROFILE*			
1	E								
2	V	En V1				V[1:0]			
3	V+V	En V1	En V2			V[1:0]	V[1:0]		
4	V+V+V	En V1	En V2	En V3		V[1:0]	V[1:0]	V[1:0]	
5	M	En M1				M[1:0]			
6	M+M	En M1	En M2			M[1:0]	M[1:0]		
7	M+M+M	En M1	En M2	En M3		M[1:0]	M[1:0]	M[1:0]	
8	V+M	En V1	En M1			V[1:0]	M[1:0]		
9	V+M+M	En V1	En M1	En M2		V[1:0]	M[1:0]	M[1:0]	
10	V+V+M	En V1	En V2	En M1		V[1:0]	V[1:0]	M[1:0]	
11	V+V+M+M	En V1	En V2	En M1	En M2	V[1:0]	V[1:0]	M[1:0]	M[1:0]
12	IP	En IP				IP[1:0]			
13	V+IP	En V1	En IP			V[1:0]	IP[1:0]		
14	V+V+IP	En V1	En V2	En IP		V[1:0]	V[1:0]	IP[1:0]	
15	V+V+V+IP	En V1	En V2	En V3	En IP	V[1:0]	V[1:0]	V[1:0]	IP[1:0]

Legend:
E – Ethernet untagged
V – VLAN tagged
M – MPLS
IP – IPv4 or IPv6
En – Enabled

7.4.6 Layer 2 Classification Profiles

There are eight profiles per ICU unit to control the Layer 2 classification.
The profile is assigned per Port ID via L2 Classification Profile [2:0].
The profile controls the MAC DA and VLAN tag parsing.

Table 7-3. Layer 2 classification profiles

FIELD	DESCRIPTION
VLAN tag Ethertype 0 value	16b configuration value
VLAN tag Ethertype 0 usage	Enable bit per tag nesting 000 – Not used 001 – For first tag only .. 111 – For all tree tags
VLAN tag Ethertype 1 value	16b configuration value
VLAN tag Ethertype 1 usage	Enable bit per tag nesting 000 – Not used 001 – For first tag only .. 111 – For all tree tags
VLAN tag Ethertype 2 value	0x8100 pre-configured constant
VLAN tag Ethertype 2 (0x8100) usage	Enable bit per tag nesting 000 – Not used 001 – For first tag only .. 111 – For all tree tags
VLAN tag Ethertype 3 value	0x9100 pre-configured constant
VLAN tag Ethertype 3 (0x9100) usage	Enable bit per tag nesting 000 – Not used 001 – For first tag only .. 111 – For all tree tags
VLAN tag Ethertype 4 value	0x88A8
VLAN tag Ethertype 4 (0x88A8) usage	Enable bit per tag nesting 000 – Not used 001 – For first tag only .. 111 – For all tree tags
VLAN tag Ethertype 5 value	0x9200 pre-configured constant
VLAN tag Ethertype 5 (0x9200) usage	Enable bit per tag nesting 000 – Not used 001 – For first tag only .. 111 – For all tree tags
Know L2 control protocols enable	20b – Enable/disable bit per L2 control protocol or known DA range – see Layer 2 Control Frame Detection per Known Ethertype. 0 - Disable 1 - Enable
Known VLAN Tag value	16b VLAN Tag = PRI[2:0], CFI, VLAN-ID[11:0]
VLAN Tag mask	16b VLA Tag mask 0 – Masked 1 - Compare to VLAN Tag value
Known VLAN ID position	3b map: 0 – Disable, 1 – Enable [2] – Third VLAN tag [1] – Second VLAN tag [0] – First VLAN tag
Known VLAN ID COS[1:0]	Reassigned as COS or Class

7.4.7 Layer 2 Control Frame Detection per Known Ethertype

A match on Ethernet DA and Ethertype fields with defined values. Each match entry can be enabled via L2 Classification Profile [1:0] field. In the case of a match, it uses the configured two bit COS value. The assigned two bit value can be used as COS or as Class.

The table below lists supported layer 2 control frames.

Table 7-4. L2 control frame types supported

NUM	PROTOCOL	DA	ETHERTYPE
0	STP	01-80-C2-00-00-00	NA
1	GMRP	01-80-C2-00-00-20	NA
2	GVRP	01-80-C2-00-00-21	NA
3	VTP	01-00-0C-CC-CC-CC	0x2003
4	LACP/802.3ah	01-80-C2-00-00-02	0x8809
5	802.1X	01-80-C2-00-00-03	0x888E
6	LLDP	01-80-C2-00-00-0E	0x88CC
7	802.1ag	01-80-C2-xx-xx-xx	configurable
8	Bridge Group	01-80-C2-00-00-10	NA
9	PAUSE	01-80-C2-00-00-01	0x8808
10	802.1ad	01-80-C2-00-00-08	TBD
11	802.1ad	01-80-C2-00-00-0D	TBD
12	Other reserved MAC	01-80-C2-00-00-0Z	NA
13	ARP	N/A	0x0806
14	RARP		0x8035
15	PPPoE Control		0x8863
16	ELMI		0x88EE
17	PTP		0x88F7
18	CFM		0x8902
19	ELOOP		0x9000
20	CLNS(ISIS)		Less than 0x5DC LLC-SNAP 0xFE-FE
21	CISCO-SNAP (CDP/CGMP/PVST/UDLD/SDP)		Less than 0x5DC LLC-SNAP: 0xAA-AA-03-00-00-0C
22	IEEE SNAP		Less than 0x5DC LLC-SNAP: 0xAA-AA-03-00-00-00-0800
23	CISCO-SNAP-BPDU		Less than 0x5DC LLC-SNAP: -x42-42-03
24	User-defined 0	Per UD: DA 48b value DA 48b mask	Per UD: Ethertype value 16b Ethertype Mask 16b DA/Ethertype or LLC/SNAP select.
25	User-defined 1		
26	User-defined 2		
27	User-defined 3		
28	User-defined 4		
29	User-defined 5		
30	User-defined 6		
31	User-defined 7		

XX – Configurable value (per ICU global configuration)
Z – Any value

7.4.8 My MAC Frame Detection

In addition to 32 predefined DAs, referred to as Layer 2 Control Frame Detection per Known Ethertype, there are six configurable DA ranges per ICU.

The configurable range is defined by maximal and minimal values of least significant byte.

The value of the last DA byte should be greater than or equal to the configured minimal value and less than or equal to the configured maximal value.

Each defined DA has a 2b COS. When classification matches the DA, the profile is used for COS assignment.

DA RANGE	***CONFIGURATION FIELDS***			***NOTES***
DA range 0	Min value[47:0]	Max value[47:0]	COS[1:0]	Use as COS or Class
…	....	....	...	...
DA range 5	Min value[47:0]	Max value[47:0]	COS[1:0]	Use as COS or Class
Predefined DAs	32x [47:0]		32 x COS[1:0]	Use as COS or Class

7.4.9 MPLS Classification Profiles

There are four profiles per ICU to control the interception of known MPLS tags for COS assignment.

The predefined known MPLS tunnels are useful for intercepting control plane traffic with pre-assigned expedited forwarding.

The profile is assigned per Port ID via MPLS Profile [1:0].

When a packet's MPLS label, EXP and S-bit match, after bit masking, the known values at a selected MPLS label stack location the configured COS[1:0] is selected.

Table 7-5. MPLS classification profiles

FIELD	***DESCRIPTION***
Known MPLS Label	20b MPLS label value
Known MPLS Label mask	20b MPLS label mask value 0 – Mask 1 – Compare to known MPLS label value
Known MPLS EXP,S bit	4 bit value
Known MPLS EXP, S bit mask	4b MPLS EXP and S-bit mask value 0 – Mask 1 – Compare to known value
Known label position	3b map: 0 – disable, 1 – enable [2] – Third label [1] – Second label [0] – First label
COS[1:0]	Assigned COS[1:0]

7.4.10 IP Classification Profiles

There are four profiles per ICU to control interception range of known IPv4 destination addresses for COS assignment.

A range of 1-256 addresses can be specified for a selected /24 IPv4 subnet and a selected /120 IPv6 subnet.

FIELD	***DESCRIPTION***
IPv4 range enable	0 – disable, 1 – enable
IPv4 subnet[31:8]	24b subnet value
IPv4 DIP LSB min[7:0]	8b value for range minimal boundary
IPv4 DIP LSB max[7:0]	8b value for range maximal boundary
IPv4 COS[1:0]	Assigned COS

FIELD	***DESCRIPTION***
IPv6 range enable	0 – disable, 1 – enable
IPv6 subnet[127:8]	120b subnet value
IPv6 DIP LSB min[7:0]	8b value for range minimal boundary
IPv6 DIP LSB max[7:0]	8b value for range maximal boundary
IPv6 COS[1:0]	Assigned COS

7.4.11 IP Next-known Protocol CoS Assignment

Assignment of CoS based on known IP next protocol value.

The IPv4 Header / IPv6 Next Header (Protocol[7:0]) is compared to list of known values, the first protocol value match assigns the configured CoS from the list.

The list includes 8 pre-configured values and 8 configurable values:

CONFIGURATION	***PROTOCOL***	***COS[1:0]***
0 1 2 3 4 5 6 7	1 – ICMP 2 – IGMP 7 – CBT 46 – RSVP 58 – ICMPv6 89 – OSPF 103 – PIM 112 – VRRP	Assigned per protocol[1:0]
8 .. 15	Configured[7:0]	Assigned per protocol[1:0]

There are four profiles IP Protocols COS that enable the COS assignment.

Each protocol defines:

- 16b IPv4 enable: where each enable bit enables the Protocol configuration line for IPv4.
- 16b IPv6 enable: where each enable bit enables the Protocol configuration line for IPv6.

7.4.12 Well-known TCP/UDP Ports CoS Assignment

The TCP/UDP port numbers in the range from 0 to 1023 are the *well-known ports*. They are used by system processes that provide widely-used types of network services.

There are 12 predefined TCP/UDP port numbers/ranges and four user configurable ranges.

INDEX	***PORT RANGE ICMPV6 TYPE***	***PROTOCOL USAGE***
0	130-132	MLD
1	133-136	ND
2	143	MLD
3	179	BGP
4	319-320	PTP
5	520	RIP
6	639	MSDP
7	646	LDP
8	1701	L2TP
9	3503	LSP ping
10	3784	BFD
11	4784	BFD
Configurable ranges:		
12	16b min	16b max
13	16b min	16b max
14	16b min	16b max
15	16b min	16b max

There are four profiles L4 Protocol COS that enable the COS assignment.

Each protocol defines 16 entries with a 5b enable bitmap:

- Enable for TCP port
- Enable for UDP port
- Enable for ICMPv6 Type
- Enable for source port
- Enable for destination port

7.4.13 VLAN Tagged Frame COS Classification Profiles

Priority is obtained from the User Priority (PRI) 3b.

There are four profiles to map the three-bit PRI to two bits of COS/Class and to use assigned two bits as COS or as Class.

The profile is assigned per Format Encapsulation profile (refer to the Format Encapsulation Options section).

7.4.14 MPLS Frame COS Classification Profiles

There are four unicast profiles and four multicast profiles to map MPLS EXP to COS. The profile is selected per Encapsulation profile.

Priority is generated from the 3b EXP/COS field. The three-bit EXP field is mapped to two bits of COS. There are separate map tables for multicast and unicast frames. The Multicast and Unicast tables are selected by the same profile.

7.4.15 IPv4/IPv6 COS Classification Profiles

There are four IPv4 profiles and four IPv6 profiles, per ICU, to control the IP classification.

The profile is assigned per Encapsulation profile.

The IPv4 profile translates IPv4 Differentiated Services Code Point and Explicit Congestion Notification (8 bits) to 2b of COS.

The IPv6 profile translates IPv6 TOS (8 bits) to 2b of COS.

7.4.16 Default Frame Classifications

Priority is defined by default configuration DEFAULT COS(1:0) (per Port ID). The default can be forced to override the classification.

Class is defined by default configuration DEFAULT Class(1:0) (per Port ID). The default can be forced to override the classification.

7.4.17 General Purpose Rules Classifier

The general purpose classifier is designed for unknown protocol types.

The last Ethertype is compared to configurable list of 4 values. The values are configurable per ICU.

The last Ethertype match results in a 2b Protocol match code:

No-match
00 – Match on first Ethertype
01 – Match on second Ethertype
10 – Match on third Ethertype
11 – Match on fourth Ethertype

If the protocol is matched to any value, the Protocol match code is combined with 4 bytes from the packet following the last Ethertype.

The [2b,4B] Key is used for a lookup in a 16 rule TCAM:

TCAM entry: 4B value / 4B mask, 2b Match value / 2b Match mask.

The first match results in 2b COS.

There is a configurable TCAM per ICU.

7.4.17.1 General Purpose Classifier Profile

There are four profiles per ICU to control the TCAM rules classification.

The profile is assigned per Port ID via General Purpose Classifier Profile profile [1:0].

Per Port ID there is a configurable 16b enable/disable per GP Classifier TCAM rule.

Table 7-6. General purpose classifier profile

FIELD	DESCRIPTION
TCAM rule enable[15:0]	16 bits. Bit per TCAM rule 0 – Rule disabled 1 – Rule enabled

7.4.18 COS Assignment Resolution

There are two global modes for COS assignment.

Default mode:

The precedence of COS assignment is:

1. My DA
2. My IP classification
3. Known MPLS classification
4. Known VLAN
5. L2 Control protocols
6. L3 Control protocols
7. L4 Control protocols
8. General classifier
9. Encapsulation COS assignment is maximal number of:
   - VLAN first tag COS mapping
   - VLAN second tag COS mapping
   - VLAN third tag COS mapping
   - MPLS first label COS mapping
   - MPLS second label COS mapping
   - MPLS third label COS mapping
   - IPv4/IPv6 TOS to COS mapping.

When none of the above are valid, the Port ID default COS is used.

Network level precedence mode:

1. L4 control sessions (TCP Source/Destination ports range)
2. L3 control protocols (over IP)
3. My IP
4. Known MPLS
5. Known VLAN
6. My DA
7. General L2 classifier (Ethertypes)
8. L2 control protocols (known Ethertypes)
9. Encapsulation COS assignment is maximal number of:
   - VLAN first tag COS mapping
   - VLAN second tag COS mapping
   - VLAN third tag COS mapping
   - MPLS first label COS mapping
   - MPLS second label COS mapping
   - MPLS third label COS mapping
   - IPv4/IPv6 TOS to COS mapping.

7.4.19 Class Assignment Resolution

The CLASS assignment uses the same precedence as the COS above, but only for the relevant logic of layer-2:

First option:

1. My DA
2. Known VLAN
3. L2 Control protocols
4. General classifier
5. Encapsulation

When none of the above is valid, the context's default CLASS is used.

And second option:

1. Known VLAN
2. My DA
3. General classifier
4. L2 Control protocols
5. Encapsulation

When none of the above is valid, the context's default CLASS is used.

7.4.20 Class Out Mapping

The Class can be mapped according to configuration by two options:

First option is used the mapping profile. The assigned COS 2b is mapped to output COS/CLASS.
The profile:

- 2 bits are the mapped COS
- 2 bits are the mapped CLASS.

The second way is to forward the CLASS as is, without mapping.

7.5 External Header Classification

When external header classification is enabled per port, the packet parsing and classification is not done and all information is extracted from the packet external header.

The information extracted is:

- PQ [3:0] load balancing hash, up to 4b.
- COS[1:0] assignment.

In external classification mode, a 16B header is attached to the packet after the Header offset (0B-64B).

There are 4 profiles that define the positions inside the header of the desired fields.
Profile is selected by External Header offset 2b profile assigned per Port ID.

The profile configures:

- COS field bit position: 0-127.
- COS field number of bits: 2 or 3. The 2 or 3 bits are mapped to 2b COS result.
- PQ LB Hash field bit position: 0-127.
- PQ LB Hash number of bits: 0-4.

7.6 PMU Physical Queue Assignment and Load Balancing

The ICU assigns the PMU-PQ (physical queue) to each packet.

There are 128 physical-queues in the PMU.

The PQ channel can be specified uniquely by PQ_base [6:0] per Port ID configuration (Rx NDMA configuration).

Alternatively the traffic of a Port ID can be load balanced using a hash function to spread over a range of channels:

- PQ = PQ_base[6:0] + PQ hash[3:0]

The PMU PQ load balancing hash is calculated by the ICU.

The PQ hash bit size is configured per Port ID: 0,1,2,3,4 bits

The valid values are:

- 0 – Single channel, PQ hash[6:0] = 0
- 1 – 2 channels, PQ hash[0]
- 2 – 4 channels, PQ hash[1:0]
- 3 – 8 channels, PQ hash[2:0]
- 4 – 16 channels, PQ hash[3:0]

The load balance hash value of 4b is calculated on a list of configuration and packet header fields. Each field can be enabled for hash calculation:

- Physical port / Interlaken channel
- MAC DA
- MAC SA
- Last Ethertype
- IPv4/IPv6 SIP
- IPv4/IPv6 DIP
- TCP/UDP Source port
- TCP/UDP Destination port
- VLAN ID 0
- VLAN ID 1
- MPLS labels 0/1/2
- User defined bytes for LB hash. The header offset and number of bytes is configurable. The header offset is taken from SOP (without the classification constant Header offset). The offset from SOP allows LB that is based on shim header. The offset can specify 0-127 bytes with configurable number of bytes for LB hash calculation: 1 - 6 bytes.

7.6.1 PQ Hash Profile

There are four hash profiles configurable per ICU.

The PQ load balance profile[1:0] is selected per Port ID.

The hash profile contains an enable bit per hash contributor field.

8. Processor Management Unit (PMU)

This section provides a high-level architectural overview of the Processor Management Unit (PMU) inside the NPS-400. The PMU is comprised of two units: PMU-0 (west) and PMU-1 (east).

The PMU serves as a task scheduler for jobs. Jobs are packet processing tasks or SW initiated tasks.

A job is a processing element tracked by the PMU, which has a unique Job Identifier (JID) that stays stable through the entire period that it is being processed.

A new job is created for a received packet, for a TM looped back packet (including timestamp response), for a job initiated by SW, or a job initiated by a timer. A job is terminated when it gets pulled out of PMU queue tracking. Either terminated internally by SW without forwarding, or forwarded to an external or loopback interface, which could go either through the TM or bypass the TM.

Each job is described by a Job Descriptor (JD) stored in IMEM and indexed by a JID. Up to 32K Job Descriptors are managed by the PMUs in shared management. Jobs that are sent to TM are replaced by TM packet descriptors which can be further extended to EMEM. The jobs are recycled in the PMU.

The PMU manages job queues (PQ-PMU queues). When required, the PMU schedules jobs according to a defined order relative to other queued jobs.

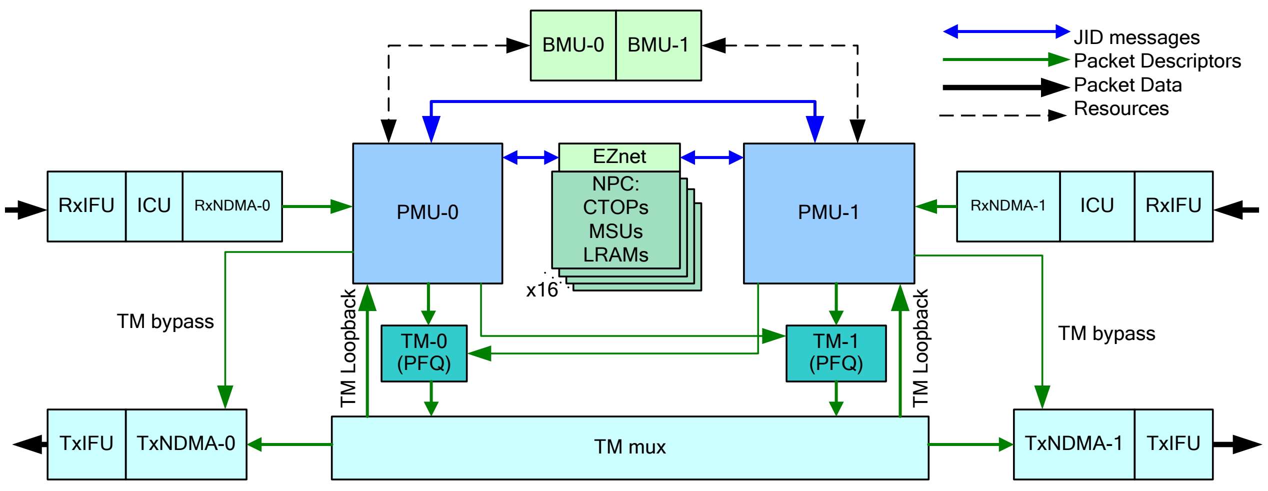
The PMU schedules jobs for SW processing (job dispatch) or forwards jobs to the Traffic Manager. The PMU can also send packets directly to the TxNDMA on its side for transmission, bypassing the TM path. Additionally this path can be used to send frames to flush service, recycling the frame buffers by the TxNDMA release machine.

The PMU may also schedule a job for re-queue to itself or to the other PMU unit.

The PMU job dispatch for processing, distributes the jobs to CTOPs to achieve load balancing. The PMU job dispatch mechanism is application aware, assigning a job to a designated pool of CTOP resources and accelerator resources.

The PMU scheduling is application SLA aware utilizing priority and WRR application arbitration scheduling. The PMU keeps track of job ordering during SW processing, scheduling to the TM and re-scheduling. The PMU supports job replication by SW or by the Traffic Manager loopback path via a Multicast Replicator (MR).

Figure 8-1. PMU interfaces



In the PMU interfaces figure above (Figure 8-1):

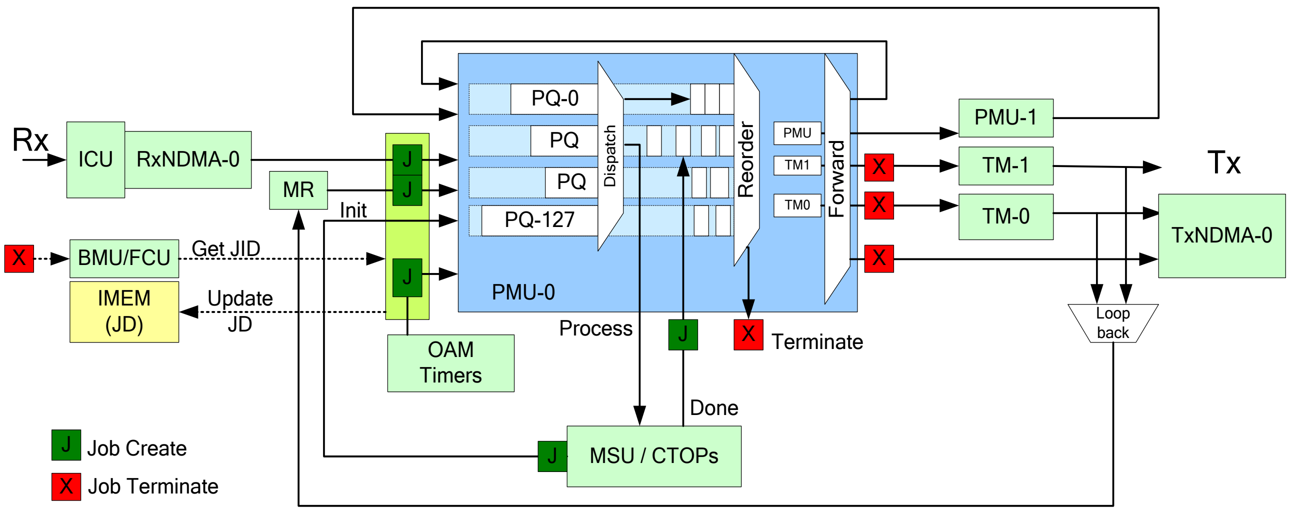
- RxNDMA – The PMU receives a packet descriptor and creates a job.
- TM loopback – The PMU receives a packet descriptor and creates a job with optional replication.
- EZnet messaging – The PMU dispatches jobs to CTOPs and receives completed jobs via EZnet messaging.
- EZnet transactions – The PMU reads/writes job descriptors in IMEM.
- PMU blocks – The jobs can be re-queued to the other PMU (on the other side of the chip).
- TM PFQ – The completed jobs are converted to packet descriptors and forwarded to the TM.
- TxNDMA – The PMU can bypass the TM scheduling, sending packets directly to the TxNDMA (also for resource recycling).
- BMU – The job resource management utilizes the BMU.

8.1 Internal Architecture Overview

The PMU is comprised of several functional modules arranged in a pipeline:

- Job initialization – The Job Descriptor is allocated via JID. The incoming packet descriptor (from its receive port, TM loopback, timer or SW) is converted to a Job Descriptor.
- Job queuing – Each PMU manages 128 queues of Job IDs.
- Job dispatch – Jobs are dispatched to CTOPs for processing with application-aware resource control and load balancing.
- Job reordering – The completed Jobs from the CTOPs are reordered before being forwarded.
- A job is forwarded to a TM or directly to TxNDMA and Job resources are freed, or the Job is re-queued back to the original PMU or to the other PMU.

Figure 8-2. PMU internal architecture overview



The PMU process flow is:

1. Job Descriptor is allocated in IMEM.
2. Job ID is queued to PMU queue awaiting dispatch.
3. Job is dispatched for SW execution by application-aware scheduling.
4. Out of order SW processed jobs are reordered before eviction from queue.
5. Jobs are forwarded to next destination.

Jobs are created by RxNDMA, TM Loopback (MR), PMU OAM timers and CTOPs SW.

Jobs are terminated and recycled when forwarded to the TM, TxNDMA and by a SW termination command.

The table below lists the possible basic flows.

Table 8-1. PMU basic flows

#	*MODE*	*PROCESSING FLOW*
1	Run to complete	Packet receive, ICU classification, PMU queue, SW processing, optional SW replication, either TM.
2	Pipeline processing	Packet receive, ICU classification, PMU queue, SW processing, optional SW replication, any PMU re-queue, SW processing, either TM.
4	SW packet	SW new job, SW processing, PMU, either TM.
5	SW Job	SW new job, SW processing, PMU queue, SW processing.
6	TM UC/MC	TM-Loopback (MR), PMU queue, SW processing, either TM.
7	Deferred pipeline	SW sends the job to TM towards the loopback port, and a new job creation is performed upon packet scheduling by the TM. TM-Loopback (MR), PMU queue, SW processing, PMU re-queue, SW.
8	Transmit mirroring	TM-Loopback (MR), PMU, any TM.
9	Timer trigger	Timer triggered new job, PMU queue, SW processing.
10	TM bypass	PMU forwards FD to TxNDMA (used also for resource recycling).

MR = multicast replicator

The following sections describe these topics:

- Job initialization
- Job budget accounting
- Job queuing
- Application-aware job scheduling
- Job dispatch distribution
- Job SW control
- Job re-ordering and evict
- Job replication
- Queue sequence numbering

8.2 Job Initialization

The PMU is responsible for creating and queuing jobs.

These sources instantiate or reschedule a PMU job:

- PMU: Create a new job from a received packet (from RxNDMA). See details.
- PMU: Create a new job from a TM loopback packet with optional replication. See details.
- SW job creation, either by "Init" or "Done" method.
- PMU timers: Each PMU has sixteen programmable timers to trigger new jobs.

An existing job can also be re-queued:

- Own-PMU existing job re-queue.
- Job received from the other PMU.

8.2.1 New Job Created from Received Packet

The received packet is parsed and initially classified by the ICU. The ICU assigns the target PMU-QUEUE and COS. The PMU queue is assigned by CoS and offset from a base queue that is configured per receive port. The packet data is buffered in an IMEM/EMEM chained buffer list by the RxNDMA.

The PMU allocates a Job Descriptor (JD) in IMEM for the received packet from a shared pool of free Job Descriptors managed by the BMU. The JD points to the packet's first data buffer. The JD is initialized by the PMU and stored in a pre-defined IMEM address pointed to by its JID.

The PMU queues the Job ID (JID) to a target PMU queue for SW processing.

The JD can be organized in containers of up to four jobs. The containers preserve messaging bandwidth, manage job dispatch and forwarding. The PMU is equipped with a configurable time interval.

If a container is filled before the timer expires, the container enters the PMU queue as a full container, else the container enters as a partially full container (fragmented).

8.2.2 TM Loopback Packet with Optional Replication

The PMU initializes a JD from TM loopback path (e.g. multicast flows or deferred queues).

Each TM can loopback to either PMU via the TM OQ multiplexer.

The next destination of loopback flows is the PMU-queue with dispatch for SW processing.

Packets looped back from TM are optionally replicated to produce a list of jobs. The TM Frame Descriptor (FD) controls the replication. All replicas share common packet data buffers.

The FD[MCC] field selects between unicast, multicast, broadcast or replication flows. For the case that replication flow FD[REPCNT] field determines the number of replicated copies (beyond the initial copy) and it gets converted to REPID counting from zero to REPCNT by the replicator. Each replica is allocated a new job ID.

See also Table 8-2. Job Descriptor (JD) structure format.

8.2.2.1 Unicast Loopback Target PQ Selection

For looped back unicast packets, the target PQ is selected by either:

- Per loopback (MR) unit configuration. There are 8 units per PMU.
- The PQ[6:0] index to selected PQ.
- The FD[COS[1:0]] can also be used to distribute flows to PQs according to TM Cell-OQ and COS.

8.2.2.2 Loopback Multicast Replicator Target PQ Selection

For the looped back packets that are multicast replicated, the target PQ is selected per MR unit configuration. The FD[COS[1:0]] can also be used to distribute MC flows to PQs according to TM Cell-OQ and COS.

See also section TM Loopback HW Job Replication.

8.2.3 SW Job Creation

SW threads can allocate new empty jobs by sending a special message to the PMU..The PMU returns the allocated JID. A single CTOP message can allocate multiple JIDs at once. For SW jobs, the PMU does not initialize the JD database and it is software's responsibility to initialize the JD.

SW handling of JD allows generic task creation using JD structures and job exchange between CTOPs.

The SW can insert a Job into a PMU via two methods:

- Job "Init" method:
  - New JD is allocated and queued to PQ (tail). This operation creates Jobs in order.
  - Jobs can be grouped in job containers. Job containers can be linked.
  - Job creation can request a Job sequence number update in JD.
- Job "Done" method:
  - An existing JD is converted into a job container and a new job inserted into the container. Each new job has its next target as either TM or PMU re-queue.
  - Per PQ, the forwarding order of new jobs instantiation is maintained by the PMU according to the relative order of the original job.

For each created Job, the SW can specify the next destination. Jobs in the container can have different destinations.

The destinations for Job "Init" method:

- Queue job in PMU-0 or PMU-1 PQ (tail) to dispatch for SW processing. This method is useful for pipeline processing. This method also useful to forward to another PMU queue with indication that the job should not be dispatched again. The current thread continues working of the job while it gets pushed to the tail of the target PQ. This flow enables ordering the job in PMU PQ without waiting for completing its processing.
- Forward to Traffic Manager (TM-0/TM-1).
- Forward to TxNDMA for transmission or for recycling the buffers, bypassing the TM.
- Terminate

The destinations for Job "Done" method for each new JID in the container:

- Re-queue job in PMU-0 or PMU-1 PQ (tail) to dispatch for SW processing. This method is useful for SW multicast processing pipeline.
- Forward to Traffic Manager (TM-0/TM-1) or TxNDMA.
- Terminate. The SW terminates the original job. The SW may initialize derivative jobs via the "Init" method to several PMU queues. The initialization can be one job at a time without containers or utilizing containers when a job batch targets the same PQ.

See also section SW Job Replication.

8.2.4 Timer Triggered Job Generation

Each PMU maintains 16 configurable timers to trigger job generation. The timer triggered jobs are useful to generate periodic Operation and Maintenance network traffic.

All timers are driven by a common time base. The time base reference clock is the core clock.

The common time base is configurable in range: [(reference clock/512) to (reference clock)].

The common time base configuration is (reference clock) / (Integer + Fraction).

Where Integer range is [1, 2, 4, 8 ... 512], and Fraction is configurable (0...(2^{32}-1))/ 2^{32}.

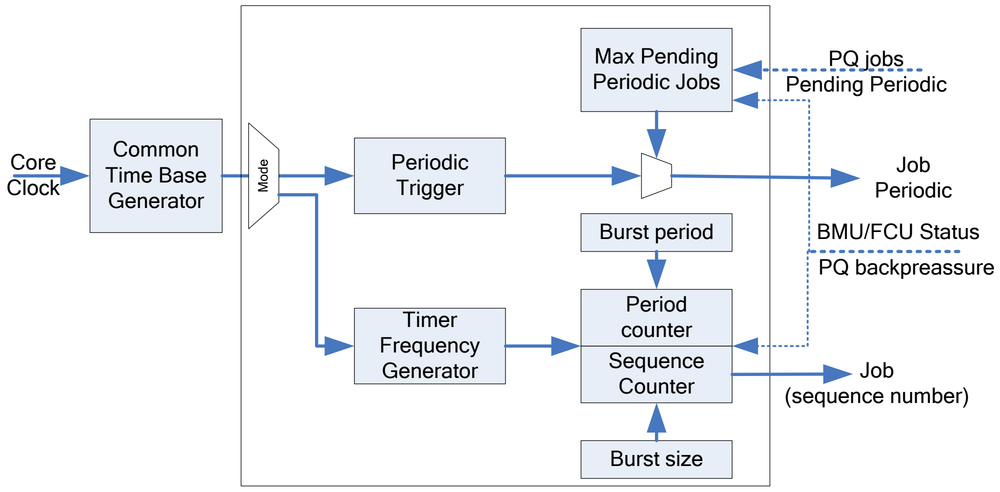
Each one of 16 timers has its own configurable frequency based on common time base:
Timer frequency = Common Time Base frequency / (Interval)
where Interval is configurable in range [1...2^{32}].

Each timer has a configurable Job budget ID that is assigned to manage the resource accounting in the FCU.

Each timer has a configurable Port ID field that is assigned to a Job Descriptor.

Each timer has a configurable target PMU queue for generated jobs.

Figure 8-3. PMU timers diagram



There are two modes for job generation supported per timer – single periodic and burst periodic generation.

8.2.4.1 Periodic Single Generation

Each period, the timer triggers single job generation with congestion backpressure from BMU/FCU. The configured period is the minimal time interval between two successive generations. The jobs are generated as long as BMU/FCU budget status is green. The status is based on Jobs and Buffers budget accounting per target PQ and target Port (LIF). When the number of pending jobs crosses the threshold of Max Pending Periodic, the Timer injects new jobs as long as BMU/FCU budget status is not red.

8.2.4.2 Periodic Burst Generation

The timer generates a configurable burst of jobs in a configurable time period. Each job is assigned with a sequence number. The sequence numbering is restarted from 0 each period. If job burst generation is not completed by the end of the period, the burst sequence continues on to next period and restarts only when the burst is completed.

8.2.5 PMU Existing Job Re-queue

When a Job's next destination is the PMU, the Job is forwarded in-order from the original PQ and re-queued into the destination PQ, keeping the same JID. The destination PQ can reside on either PMU-0 or PMU-1.

PMU-0 manages queues 0-127 while PMU-1 manages queues 128-255.

The target PMU-QUEUE is specified in the SW "Done" message to the PMU. The SW may also update the JD for its next stage.

After re-queue, the JD next destination is SW processing. The job can be queued awaiting SW processing either with or without dispatching the job via MSU messaging.

The SW may implement re-queue without an additional dispatch. This action shall re-queue the JID to PQ (same or other PMU) but shall not trigger dispatch messaging, and await a SW "Done" signaling for PQ order restoration. The subsequent "Done" command must forward or terminate the Job but cannot utilize "re-queue without dispatch" again. This subsequent "Done" command must be directed to re-queue to the queue that was targeted by the first forward without dispatch operation in the target PMU.

Job forwarding performance within the same PMU is significantly higher than job forwarding performance from a queue in one PMU to a queue in the other PMU. System level SW architecture should take this into account and minimize the number of times that jobs are forwarded between PMU queues in a pipeline processing environment. The re-queue to the other PMU can be useful when forwarding directly to TxNDMA (TM bypass) since each PMU can forward only to its associated TxNDMA.

8.2.6 Job Sequence Numbering

Each PMU queue maintains a job sequence counter (16b) that is incremented with each new job initiated by RxNDMA or TM loopback. This sequence number is saved in the Job Descriptor [SEQNUM] (see Table 8-2. Job Descriptor (JD) structure format). The sequence number can be used by SW for ordered processing of the traffic. The Job sequencing can be enabled per logical interface or for all jobs of a PMU queue.

8.3 Job Budget Accounting

The PMU uses the BMU services to manage jobs as index pools. The budgets are maintained by Job-ID resource counters (refer to the *Congestion Management and Flow Control* chapter) that can be associated with different functions. Whenever a JID is allocated, the budget usage counter is incremented. Whenever a JID is recycled, the budget usage counter is decremented.

The FCU/PMU configuration associates a receive port/channel or group of ports/channels with a job budget counter for new job allocation for a received packet.

When a receive interface exceeds its budget or group/global budget, the packet is dropped and counted.

The budget is managed by four hierarchies of counters. A port may initiate flow control when it is approaching the limits of its allocated job counts under each of the tracked hierarchies.

The FCU/PMU (shared by PMU-0/PMU-1) maintains:

- 128 job budget counters (channels). Each budget counter is hierarchically assigned to a LIF (logical IF) counter.
- 64 job LIF counters. A LIF counter aggregates channel budget counters. Each LIF counter is hierarchy assigned to a group counter.
- 8 Group counters. Group counters aggregate LIF counters.
- Total global (application) counter.

Each counter’s budget utilization is monitored by a set of seven configurable thresholds. The budget utilization is an 8-level value: from 0 - low utilization to 7 - fully utilized.

The job budget utilization is reported to SW and via OOB Status serial interface. The job budget utilization can be used for priority flow control generation with a combination of packet buffer utilization.

The job budget utilization is used for RxNDMA service rate criteria to slow packet rate for congested PMU queues utilizing the input FIFO buffering of packet descriptors.

The job budget utilization is used for ICU/RxNDMA packet drop decision criteria in order to drop packets before memory buffering when packet is destined to a congested resource.

The SW job allocations may use more precise budget level indications. SW that tries to allocate a job which exceeds its specified budget gets a rejection (no-job) response. Particularly SW can try to allocate multiple jobs, and either get all of them or none.

A PMU queue can be indirectly associated with a job budget (by tying the configuration of a port budget to a specific PMU queue (or few queues for the port). One PMU queue can be used for generation of priority based flow control, while separate PMU queues can achieve class based flow control.

As part of this configuration when SW pulls a job out of the queue it may credit its budget back by a message to the PMU.

This budget can control the fill level of different PMU queues. The PQs can be associated with traffic COS classification, and it is up to SW or external OOB flow-control hardware to associate the PQ with the proper flow control action (for example, send a priority based flow control packet through a specific port to its peer device based on PQ (COS) fullness).

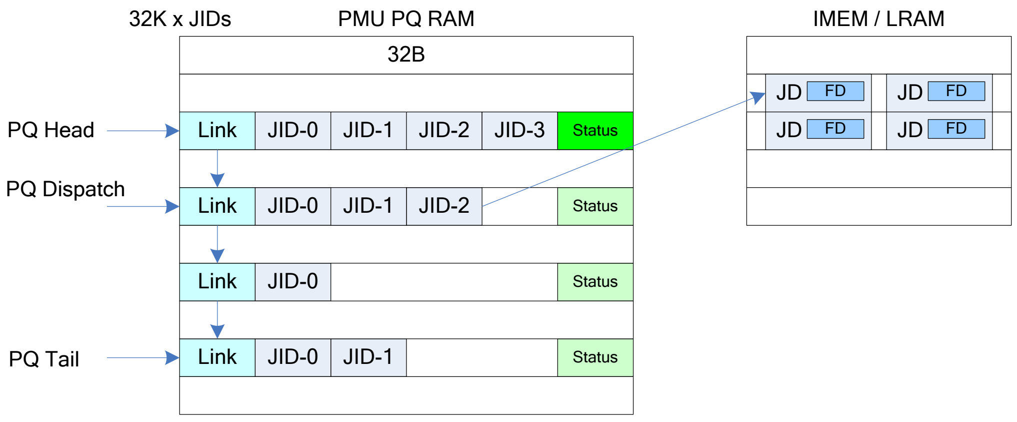
SW may associate the job with a different budget during processing and initiate budget usage updates as a consequence (i.e. increment/decrement the associated to budget counters).

8.4 Job Queuing

A total of 32K shared jobs can be managed in the PMU system, and each PMU unit can manage up to the entire amount of jobs.

- The Job Descriptor allocated in IMEM is assigned a Job ID (JID).
- The JIDs are queued in the PQs. Each PMU maintains 128 PMU queues (PQs).
- The PQ is maintained as linked list of Jobs.
- JID are allocated from free pool, when a Job is created.
- JID are recycled to a free pool when a Job is forwarded to the TM, TxNDMA or discarded.
- JID pool is global serving the two PMUs.
- The JID queued in memory by containers aggregating 1 to 4 JIDs.
- The PMU maintains counters of total jobs queued, number of containers, and number of jobs dispatched.

Figure 8-4. Job queuing in memory



8.4.1 Job Descriptor

Job Descriptors (JD) are stored in a continuous table in IMEM/LRAM under Global Data IO memory space (IDMEMGIO, MSID=14). The actual JD address in IDMEMGIO space in Job Table Base Address (JTBA) plus JID multiplied by 32.

The PMU initializes a JD after frame reception, either from the RxNDMA path or from the TM loopback path or OAM timer (e.g. multicast flows or deferred queues). SW initializes the JD in case of empty job allocation from PMU. SW also updates JD fields before forwarding a frame to the other PMU queue or sending the frame to the TM (either to Tx ports for transmission or to loopback ports). The JD structure includes the FD and additional fields (Table 8-2).

The PMU dispatches one to four JID values from the same queue to the selected MSU in a single job dispatch message. The MSU spreads each JID to a selected SW thread in the NPC for processing. The SW thread fetches a JD from the LRAM job table and processes the frame according to the FD/JD types and attributes. Typically the JD is first copied by the thread to its private scratchpad area, and later on the updated JD is copied back to the job table location.

The PMU accesses the JD:

- On packet receive, the JD/JID allocated and JD is written.
- On SW job init with service request, the Job is updated with PQ sequence number.
- On job forwarding to the TM, the JD is read.

Table 8-2. Job Descriptor (JD) structure format

The JD contains:

FIELD NAME	USAGE	FIELD DESCRIPTION
JT	JD Type	Job Descriptor type: Incoming, Outgoing, Forwarded, Loopback, Timer job, SW Job, Timestamp response. seven job entries, and an optional pointer to another container JD that further extends the list.
FD Bytes 1-15	Frame descriptor contents excluding ECC field.	For single frame JD the frame descriptor is stored in bytes 1-15, excluding its ECC most significant byte (which stores the JD type). FD bytes exist only for a JD that describes an incoming or outgoing frame. The FD contains CoS[1:0] that is used by TM path forwarding.
Incoming JD Fields		
INFO_IN[63:0]	JD Incoming Job Information	▪ Rx port frames hold Rx information, flags and potentially also Rx timestamp. ▪ Loopback port frames hold 16-bit INFO field from the loopback port (potentially holding replication count, PMU queue, or generic SW info). ▪ Tx confirmation port may hold 1588 timestamp response. JD flags further qualify the existence of various fields in INFO_IN.
RXINFO[15:0]	Rx port frame info	Receive frame information. ▪ IMEM buffer count. This is the number of leading buffers allocated by Rx NDMA from IEMEM for frame storage. The remaining buffers (if exist) are allocated in EMEM. ▪ Congestion State: This field provides the congestion grade calculated by ICU of the Rx port (at time of frame reception) - Green/Yellow/Orange /Red. ▪ IMEM budget threshold 2b reported by flow control at reception time. ▪ EMEM budget threshold 2b reported by flow control at reception time. ▪ PMU job budget threshold 2b reported by flow control at reception time. ▪ PMU Forward Pressure Threshold 2b, indicating the depth of PQ from which the job was sent, compared to queue thresholds. ▪ RX info thresholds represent measured fullness by color codes ranging from fully confirming Green to non-conforming Red.
REPID[15:0]	Replication Number	Replica Identification When port ID represents a loopback port and MCC (identifying that multicast replication preparation flow had been chosen by the original outgoing job), the REPID indicates the current job replication number. The PMU loopback port replication counter counts from 0 to REPCNT provided by SW and injects a new job to PMU generated with an incremented REPID.
LBINFO[15:0]	Reflected TM INFO	Reflected TM INFO ▪ When port ID represents a loopback port, the 16-bit information represents the reflection of the original outgoing packet SW info. This may be general purpose SW info, explicit PSID selection, or SW selection of the PQ number. ▪ When port ID represents Tx confirmation or timestamp confirmation, LBINFO holds the actual MAC which transmitted the frame.

FIELD NAME	***USAGE***	***FIELD DESCRIPTION***
RXFLAGS[7:0]	Input Classification Unit Flags	Input Classification Unit Flags ▪ On Rx port packets, the Rx flags set either by the ICU or Rx NDMA data path indicate CRC checked and valid, and error states. ▪ IMAC Flags Bits: CRC checked by the MAC / CRC OK. ▪ Commit Indication. Indicating that internal buffers are consumed from the port committed budget and not from excess. ▪ Truncation Error. Indicating the frame data is illegal (truncated on the Rx path). ▪ ICU Successful Parsing. If this flag is cleared, the L2-L3 frame header could not be parsed by ICU (either due to an illegal header or unsupported header type), and RXINFO fields dealing with packet flow control drop decisions are invalid.
TS_NS[31:0]	1588 timestamp nanosecond counter	▪ On Rx ports, the 1588 timestamp sampled by the Rx MAC when the frame is speculated to be within the PTP length range (based on frame length) to minimize overheads. The FD flags TS bit is set to indicate the validity of TS_NS, which holds the 32-bit nanosecond clock portion of the timestamp. ▪ On Tx timestamp confirmation ports, the Tx MAC samples the timestamp at the beginning of transmission time and embeds the measurement in the response PMU job. ▪ On loopback ports, the architecture allows the PMU to sample the 1588 timestamp at the moment the JD is injected back into a PMU queue, enabling latency measurements for the loopback paths.
TS_SEC[7:0]	1588 timestamp lower seconds counter lower byte	The timestamp seconds counter is sampled at one with the nanoseconds counter, providing together the standard 5B timestamp. It is also validated by asserted TS flag.
TIMERINFO[15:0]	Timer Information	Each PMU OAM generation timer provides relevant information based on its operation mode and status (for example when a timer is lagging behind due to job limit or arbitration delays in PMU).
TIMERID[7:0]	Job Originating Timer ID	Each PMU OAM generation timer has unique ID: 0x00-0x0F: west PMU timers 0x0-0xF respectively. 0x80-0x8F: east PMU timers 0x0-0xF respectively.
EVTNUM[31:0] / TIMERVAL[31:0]	Timer event number / value	The timer event number represents a sequentially advancing event generated by the timer. It may be the timer value itself, or a separate counting extracted from timer bits and internal conditions that affect its number.
SWINFO_IN[63:0]	Forwarded JD SW info	When forwarding a job from thread to thread through the PMU, software may reallocate the INFO_IN field and use it as general purpose SW info.
PQ[11:0]	PMU Queue	The PMU queue is initialized by PMU for incoming jobs. When SW selects the next PQ, it selects the destination by job forwarding control, and may optionally update this field as well to the next queue, to inform the next pipelined processing stage. On jobs sent to TM loopback ports, SW may use this field to select a Loopback PQ when the job returns (overriding the loopback PQ configuration of the loopback port). On other TM loopback ports (such as multicast replication), this field is not used and the loopback port configuration selects the PQ. Each PMU has 128 queues represented by PQ[6:0]. PQ[7] distinguishes west PMU (when cleared) from east PMU (when set). Rx FD that generates a job dispatched by the PMU is initialized and owned by that PMU instance and the PQ value reflects that it owns the job. SW must return the job to the PMU that owns it.

FIELD NAME	USAGE	FIELD DESCRIPTION
		When SW allocates an empty job from any PMU it is not yet assigned to that PMU and does not have associated PQ. SW then selects the next PQ. Defining the PMU instance that owns the target PQ and sends the JD to that instance. On forward without dispatch operation, the owner PMU of the next job-done is the one that holds the next PQ.
GC[15:0]	Gross Checksum	On an incoming job, RxNDMA may calculate gross checksum on frame data. When programmed to do so it sets the FD GCE flag to indicate that gross checksum exists. Gross checksum calculation done by RxNDMA path starts at Header Offset and continues to end of frame, excluding bytes stripped by the Rx port MAC (e.g. four CRC bytes at the end of the frame). It possibly includes leading and trailing shim data that has to be accounted by SW to finalize checksum calculation. To complete the frame checksum calculation the SW has to subtract the portions of the frame not participating in the checksum, with or without HW acceleration (e.g. CLDMA service). GC is not used on the outgoing Tx path.
SEQNUM[15:0]	Queue Sequential Number	This field is valid only when service but S bit is set. Entry holds physical queue sequence number. SEQNUM is written by PMU either following RxNDMA frame reception where LIF configuration requests sequential number service. When PMU forwards a job to another PMU queue, the job forward field may request PQ service (where PMUID is the target PQ number) . On service completion, the PMU writes to JD in IMEM, sets the S bit to marks the completion, and writes SEQNUM with the selected PQ counter value.
S	Service Flag	SW can set to request to update the SEQNUM.
V	SEQNUM existence flag	This flag is set by PMU in case it updated SEQNUM.
SR[1:0]	Service Request type	This field provides service request code. It is used in combination with job done messages that involve reading the JD by the PMU (such as forward to TM queue). The PMU reads the JD and determines the service: No Service Requested or Activate PMU per physical queue sequence numbering when forward to TM loopback port includes service requirement.
Outgoing JD Fields		
DP[1:0]	Drop Policy	This field controls the outgoing packet drop policy on TM scheduled flows: ▪ Standard TM drop policy. The packet can be dropped at TM queues by policer WRED, or by other TM scheduling levels. ▪ No Drop Packet. The packet is protected from getting dropped along TM data paths (either on policer WRED or other TM scheduling levels). The packet could be dropped in case the TM queue it belongs to is being flushed through configuration access. ▪ SYNC packet (never dropped). In addition to not being dropped by TM data paths, the SYNC packet is also not dropped on a flushed queue (the SYNC packet is used to mark the end of the queue flush). For loopback ports or transmit confirmation flows, selecting SYNC policy also requires to set TC=1, protecting the packet from drop by the data path following the TM (on TxNDMA).
B	TM PFQ Bypass	This field defines the packet scheduling traffic management mode. ▪ Full TM operation. Packet is queued in TM PFQ into the selected Flow ID (FID). Packet switching table mapping based on L0/L1 entity is available.

FIELD NAME	USAGE	FIELD DESCRIPTION
		▪ TM PFQ Bypass operation. The packet runs through TM, bypassing scheduling, and directly map into the packet switching table entry selection. In this mode, preconfigured or only explicit PSID Queue Mode is available. Other TM control fields (such as flow ID, policer WRED controls and WRED statistics) are ignored. Additionally the PMU holds dedicated queues for full TM bypass and packet drop/recycle. Selecting these queues ignores JD, and all attributes are provided by the job done message.
QM[2:0]	TM Output Queue Mapping	The QM selects the traffic manager output queue mapping mode. The TM port switching table selects an output channel, and the output channel further selects the side a Tx port it is assigned to. Each TM holds a base location per L1 (512 per side) and base location of L0 entity (128). Combinations of L1 entity bits and PSID Select (PSIDS) are used to map the base location. Each base location holds eight output channels that can be further selected by PSIDS 3 LS bits. Additionally, an explicit table location can be mapped directly by software without considering L1/L0 entity. According to QM selection the following mappings are possible: ▪ Standard Entry Selection. L0/L1 entity selects base table entry according to TM queue topology, PSIDS[2:0] selects offset, and the table entry provides an output channel. For x ports all eight output channels are destined to the same port located in either east or west side. The output channel may also be a TM loopback port back to PMU (by selected channels 128-143). Loopback ports are destined to the PMU on the same side of the TM. ▪ Preconfigured Entry. In this mode, the TM PSID table base is taken from TM configuration and PSIDS[2:0] selects an offset on that entry. This method bypasses TM topological L0/L1 queue selection when addressing the port switching table. Typically this flow is used by implicit TM loopback flows to one of eight PMU loopback channels (channel IDs 128-143). It can also be used for specific transmission path with eight selectable output channels. ▪ Explicit PSID selection. In this mode, SW puts an explicit PSID selection (EXPSID) in the 16-bit INFO fields carried in the TM packet header and EXPSID[12:0] explicitly selects an entry from the PSID table (from the portion typically used for L1 entities). This mode is used for PFQ bypass, and can also be used for explicit 1588 two-step channel selections and backward compatible modes which explicitly map the port. ▪ Variable size PSID selection and L1 entity combination. The port switching table entry is selected by combination of 3-6 L1 entity MS bits, concatenated to 6-9 PSIDS LS bits, building together the 12-bit entry address. Using 6 PSIDS bits provides for selecting any one of eight ports, and eight sub-ports per port. Using 9 PSIDS bits provides for selecting any one of 64 ports with eight sub-ports per port (there are 48 physical ports on each side). In this mode, the TM INFO field is not available (it is instead used for carrying the extended PSIDS field).

FIELD NAME	USAGE	FIELD DESCRIPTION
PSIDS[2:0] or PSIDS[8:0]	PSID Select	The PSIDS used is used as an offset to the PSID table beyond base address selected by L1 entity bits. ▪ PSIDS[2:0] is used on standard table mapping or on loopback mapping to provide sub-port offset (one of eight channels per port). In this case, the TM INFO field is available to carry SW information, TM loopback port information (such as replication counts), or TM loopback port selection of PMU queue. ▪ PSIDS[8:0] provides up to 9 bits which are concatenated to the remainder bits coming from L1 entity selection used as based address. This method provides mapping multiple ports and eight sub-ports per port where L1 entity MS bits define a range of ports and PSIDS reminder bits provide an offset in that group. TM static configuration determines how many bits are used from PSIDS, and the remaining bits are used from L1 entity as base. When PSIDS[8:0] field is used (QM=3), the TM INFO field is not available for the application (it is instead used in the TM packet descriptor to carry PSIDS).
PQN[7:0]	PMU Queue Number	On TM loopback paths where QM=0 or QM=1 (L1 entity or loopback base concatenated with PSIDS[2:0] selects port switching table entry) and loopback port is configured to enable override from SW, PQN directly selects the PMU queue for the incoming frame. If MCC (multicast preparation mode), the loopback port treats TM INFO as replication count and selects PMU queue from configuration.
REPCNT[15:0]	Repetition Count	On TM loopback paths where FD MCC field is set to 0b11, the TM INFO field carries REPCNT. The repetition count defines the number of repeated job descriptors generated by the PMU replication logic. The replication logic counts from zero to REPCNT (meaning that REPCNT determines the extra jobs created in addition to the first loopback job). Single job with no further repetition is a legal scenario (identified by REPCNT=0 and MCC=0b11).
EXPSID[12:0]	Explicit Packet Switch ID	When QM=2 on an outgoing job toward TM, the EXPSID field explicitly selects the port switching table address, where the TM output channel is configured. This mode is required when TM PFQ is bypassed, and may also be used to select TM specific entries servicing two-step 1588 timestamp queues.
SWINFO[15:0]	Software Info	When TM loopback port entry to PMU is not programmed to be overridden by PQN and MCC is not 0b11 (replication preparation), the INFO field in JD can carry general purpose software information that gets echoed back to the JD created by the loopback packet. Is it up to device configuration and software usage of loopback resources to determine if a loopback port carries SW info or explicitly selects PMU queue number. In either case, replication (MCC=0b11) overrides both and provides REPCNT.
TXINFO[15:0]	Tx port Information	The Tx port information may is used for controlling the INFO field in the TM packet descriptor. It may provide an explicit PSID entry selection, or up to 9 bits of variable sized PSIDS selection.
PC[2:0]	Policer Color	The 3-bit policer color provided in the JD (selected by SW using one or more token buckets and other application considerations). It is used at TM enqueue point for implementation of the WRED algorithm.
FID[19:0]	TM Flow ID	The FID selects one of the TM flows. FID[19] distinguishes flows managed on west TM (0) or east TM (1). The flow uniquely selects an L4 entity in the target TM, and TM configuration topology further maps it through TM scheduling levels down to the target port.

FIELD NAME	USAGE	FIELD DESCRIPTION
SCP1[2:0]	Statistics Code Profile 1	Statistics Code Profile 1 field is used in TM WRED statistics reporting, participates in generation of a 5-bit policer drop CODE in either topology based statistics or stream ID based statistics. An address comprising policer code pass/drop decision, an aggregated code depending on origins for pass drop decision, 3-bit color, Profile 1 bits and Profile 2 bits map together two tables, each generating a separate 5-bit reporting code. Each code selects an offset in a block of up to 32 counters. One code is used for topology based reporting while the other used for stream based reporting (and both can be used in parallel). Topology based reporting counter ID is: ID_BASE + CODE + (ID_OFFSET * Scale). ID_BASE points to the first reported counter. ID_OFFSET is set to select the start flow ID of a reported range, normalizing the report region to start at index zero. The scale is defined in power of two steps from 1 to 32, defining how many code values are available per reported event. Stream ID based reporting counter ID is: SID_BASE + CODE + (STREAM_ID * Scale). SID_BASE points to the first reported counter and STREAM_ID is provided by the JD.
SCP2[2:0]	Statistics Code Profile 2	Statistics Code Profile 2 participates together with Statistics Code Profile 1 in generation of a 5-bit policer drop CODE in either topology based statistics or stream ID based statistics. It provides 0-3 bits replacing bits from the aggregated policer result when mapping stream ID based reporting. Additionally it participates in mapping the topology based reporting together with profile 1 code and pass/drop decision.
STREAM_ID[23:0]	Statistics Stream Identifier	The 24-bit STREAM_ID points to the base location where a continuous block with statistic counters associated with the packet can be updated. On WRED reporting, together with policer codes and SW statistic code profile 1 and statistic code profile 2 the stream based reporting code is generated, mapping one counter in a block of up to 32 counters for the reported stream. The STREAM_ID field selects the block to be reported, where the reporting code selects the counter in the block.
PFTI[3:0]	Policer Flow Template ID	The PFTI selects one of sixteen RED behavior templates to be used by the WRED algorithm at the flow level (L4). Each template holds eight profiles (one per color), totaling 128 templates. A single entry in the template provides per priority the relative percentage of different ranges of the WRED graph for that priority.
PCTI[3:0]	Policer Class Template ID	The PCTI selects one of sixteen RED behavior templates to be used by the WRED algorithm at the class level (L3). Each template holds eight profiles (one per color), totaling 128 templates. A single entry in the template provides per priority the relative percentage of different ranges of the WRED graph for that priority.
PFAI[7:0]	Policer Flow Absolute ID	The PFAI selects an absolute scaling factor ID from a 256-entry table, used in combination with the selected PFTI template to define WRED algorithm behavior at the flow level (L4). Normalized regions provided per priority by the template are scaled to describe the final WRED graphs.
PCAI[7:0]	Policer Table Absolute ID	The PCAI selects an absolute scaling factor ID from a 256-entry table, used in combination with the selected PCTI template to define WRED algorithm behavior at the class level (L3). Normalized regions provided per priority by the template are scaled to describe the final WRED graphs.

FIELD NAME	USAGE	FIELD DESCRIPTION
IPGE[5:0]	Inter Packet Gap Emulation	The Inter Packet Gap emulation field provides per frame overhead for TM IPG emulation logic. It affects packet length taking into account parts that are added or excluded from shaping and fairness algorithms as well as from statistics reporting. The TM can independently use IPG emulation for altering frame length on shaper accounting and for statistics reporting. Additionally IPG emulation can affect TM WFQ scheduling at each level independently. Besides emulating physical media characteristics (such as Ethernet inter-packet gap bytes), IPGE may account for parts of a packet that are either ignored or added in TM algorithms (for example, it may consider only layer 3 IP traffic and use IPGE to exclude the layer 2 header as well as four CRC bytes from the packet length for shaping and scheduling). IPGE can also be used to account for proprietary header bytes. The IPGE field comprises a 5-bit mantissa (-16 to 15) and one bit for exponent selection mode (TM configuration has global exponent configuration to scale the mantissa by x1, x4, x16 or x128. The covered range is: (-16..15) x (1/4/16/128) bytes.
Generic Software JD Fields		
SW Payload	SW Payload bytes 1-31 (in user defined JD).	For a single frame JD bytes 1-31 are all user defined. SW job descriptors can carry any message between applications in a general purpose multiprocessing environment and implement different pipelined processing flows. If SW needs to use a PMU service, it should assign bytes 28-31 for the service definitions (S, V, SR, SEQNUM, PQ).
Container JD Fields		
JC[2:0]	Job Count (in Container JD)	In job container format, the job count provides the number of Job IDs and forward controls included in the container. At least one JID should be included (indicated by JC=0b000), and up to seven leaves can be included in the same container (JC=0b110). When JC=0b111, the container holds in addition to seven leaf jobs another extension job ID (EXTJID) which is a pointer to another job container. All jobs from JID0 to JID6 must be leaf jobs (no nesting allowed in the middle of the container). Additionally EXTJID, if exists, must point to another container and not to a leaf job. If any of JID0 to JID6 point to a job container, or EXTJID points to a leaf job, the PMU detects and reports an error and recycles all jobs with no forwarding. Note that there is no representation for an empty job container.
CJBID	Container Job Budget ID	The CJBID indicates to the PMU which job budget ID to credit when the job container is recycled (after being exploded to its pointed leaf jobs). The job container is recycled after (JC+1) leaf jobs are accessed and possibly EXTJID is accessed as well (if reading the extension container). In case of extension, the CJBID field of the next container indicates its budget without depending on the current container.
JID0[15:0] – JID6[15:0]	Container Job IDs	Each of JID0-JID6 valid in JC includes it in the size of the local container list. It is a pointer to a leaf job that needs to be forwarded by the container. It must never point to another job container.
FC0[3:0]-FC6[3:0] PQ0[11:0]-PQ6[11:0]	Container Forwarding Controls and PQ selections	The forwarding control selects the next PMU queue, TM ports, job recycling and other activations of PMU service features. FC0-FC6 must not select a container explosion queue (the pointed JID is never a job container).
EXTJID[15:0]	Extension Job ID	In an extended job container, the EXTJID holds the job number of an extension job container. Its validity is qualified by JC, meaning all seven leaf jobs in the current container are used, and additionally EXTJID points to the next container. The extension job must be of container type.

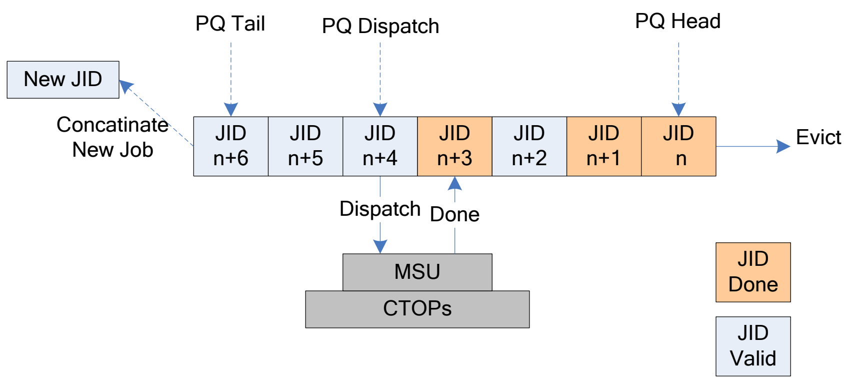
8.4.2 PMU Queues

The Job ID (JID) is dynamically allocated by the PMU from a shared pool and is used as a JD selection index. Each entry is 32B long.

The PQ is managed as a FIFO. The new jobs are inserted at the tail. The jobs are evicted from the head. The jobs are dispatched to CTOPs from the Dispatch pointer.

The PMU-0 manages queues 0-127 while PMU-1 manages queues 128-255.

Figure 8-5. PMU queues



The JID status can be:

- Not Valid – The JID is pointing to an invalid JD.
- Valid – The JID is pointing to a valid JD. The JID can be dispatched to SW processing.
- Done – The SW is done with Job processing. The JID can be evicted.

New JIDs concatenated to a PQ are Valid for SW processing or Done and ready to be evicted without processing. JIDs forwarded from PQ to PQ have their JID preserved. The Valid JIDs are dispatched to CTOPs via MSU messaging.

The SW processes the Job and returns the JID with a “Done” message. The SW message indicates the next JID destination after PQ eviction: TM-0/TM-1, PMU-0/PMU-1, or Terminate.

The SW processing completion can be out of order with the Dispatch. Thus early dispatched jobs that are not yet completed stall eviction of later dispatched jobs that were processed. The eviction order is managed as a FIFO with enqueue ordering. JIDs evicted from a PQ with next destination of TM or Terminate, have their JIDs recycled to the free pool. Evicted JIDs targeting PMU re-queue have their JIDs preserved. When a “Done” message’s next target is re-queue the JD to PMU-QUEUE, the destination after re-queue is SW processing.

The job can be queued awaiting SW processing either with or without dispatching the job via MSU messaging. The SW may implement re-queue without additional dispatch. This action shall re-queue the JID to PQ (same or other PMU) but shall not trigger dispatch messaging, and awaits SW “Done” signaling for PQ order restoration.

This operational mode is required when the current SW thread continues to process the job, but has to forward the job to another PMU queue in order to avoid head of line blocking by order restoration on the original PQ.

The subsequent “Done” command must evict the Job but cannot utilize “re-queue without dispatch” again. This subsequent “Done” command must be directed to re-queue target PMU. The application can utilize TM queues to extend the PMU job queues by using the loopback path, where the TM sends a frame back to the PMU for additional processing. In this flow, the frame gets a new job ID when it enters the PMU for the second time. Saved packet SW contexts or another means may be used to associate it with this flow and detect the next processing step that it requires.

8.5 Application Aware Job Scheduling

The PMU job scheduling is application aware. Each PQ is assigned to one of 8 application schedulers.

Each application scheduler is assigned to a pool of processing resources (CTOPs). This facilitates resource reservation and resource sharing among applications. Dispatching applications to designated processing resources utilizes resource locality and reuse, avoiding SW code memory, data memory and cache trashing by non-compatible applications. The resource locality achieves lower access latencies to data structures and lower overall data and message bandwidth.

Each PQ is configured with:

- Designated application scheduler
- Strict Priority 0-3
- WRR Weight 0-255

The application PQs arbitration is dual hierarchy:

- Select highest priority PQs (with starvation prevention timer).
- WRR service of PQs according to configured weight.

The number of CTOP active threads is monitored per application.

The MSU provides resource utilization metrics and the application scheduler selects which MSU to use. The metrics list utilization of the CTOP threads in the NPC and pending jobs for service. This enables balancing loads based on the most relevant criteria. Each MSU provides feedback and the application scheduler selects which of the four to use for load balancing.

The application is configured with an “optimal” and “maximal” number of active threads.

Each application holds a set of thresholds for dispatched job counts (job is execution by threads), and the priority of its scheduling towards the clusters depends on which range the counter is located. Within each range of the application thresholds WRR arbitration further selects which application gets served by the group schedules.

Each application scheduler maintains jobs counter (++ on dispatch, -- on done). This jobs counter is compared to a set of thresholds. The thresholds are initialized by the SW control plane according to two SW parameters Max and Commit:

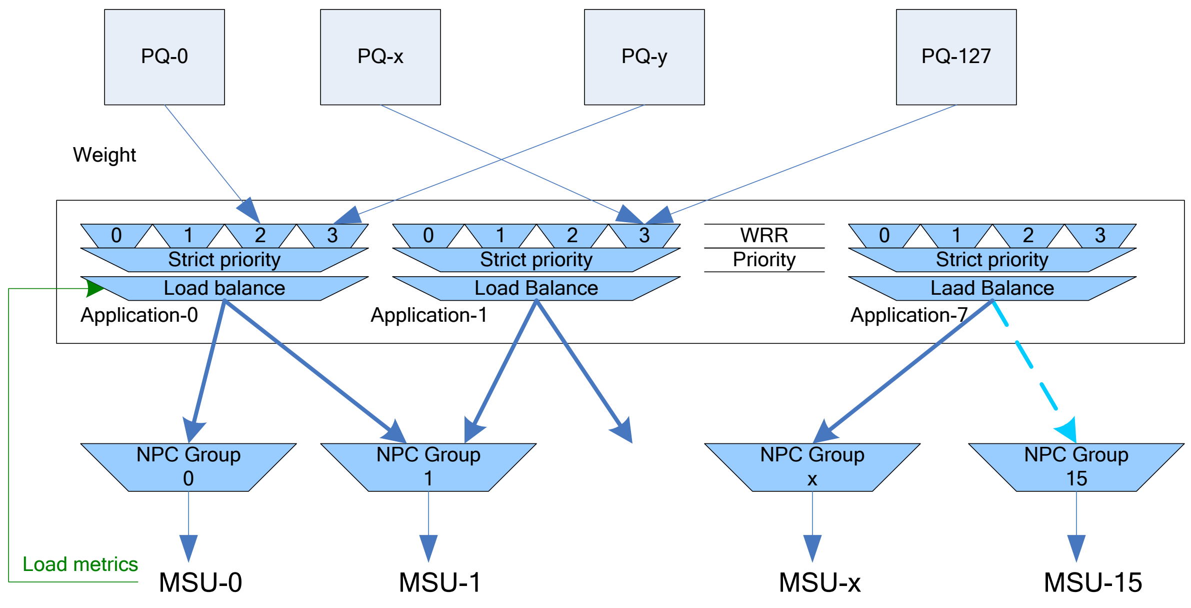
- Max = SW Max parameter
- Level 2 = (SW Max – SW Commit) * 2/3
- Level 1 = (SW Max – SW Commit) * 1/3
- Level 0 = SW Commit

The comparison yields priority colors for scheduling:

JOBS COUNTER LEVEL COMPARE	***ASSIGNED COLOR***	***SCHEDULING PRIORITY***
Max >= Jobs Counter > Level 2	Red	Lowest
Level 2 >= Jobs Counter > Level 1	Orange	
Level 2 >= Jobs Counter > Level 1	Yellow	
Level 1 >= Jobs Counter > Level 0	Green	Highest

When two application schedulers are mapped to same cluster, the one with higher priority color is served first.

Figure 8-6. Application-aware job scheduling



8.6 Job Dispatch Distribution

The PMU dispatches jobs to Messaging and Scheduling Units (MSU) in each of the NPCs, where jobs are allocated to the specific core and SW threads that processes it. Each NPC holds sixteen CPU cores for a total of 256 cores. Each CPU core can run up to 16 HW threads, totaling 4096 concurrent SW threads for packet processing.

8.6.1 Load Measurement / Load Balancing

Each MSU communicates with both PMUs and sends information about its thread occupancy and utilization load. In response it receives from the PMU a massage with a list of jobs, or alternatively a message indicating there are currently no jobs to be processed by cores/threads associated with this MSU.

The MSU provides resource utilization metrics and the application scheduler selects which MSU to use. The metrics list utilization of the HW accelerators in the NPC and provide thresholds relative to the percentage of threads that are available to MSU which are currently occupied with job processing (i.e. the remaining free threads). This is a quantitative metric that does not consider threads which are not available for MSU scheduling. This enables load balancing based on the most relevant criteria.

8.6.1.1 Job Request/Dispatch

Each MSU informs the PMU that it is willing to accept a small portion of jobs by the job request message.

The PMU increments the MSU requests in an MSU request counter.

The PMU considers the fact that there is a reserved slot in the MSU to dispatch four jobs per request. The MSU sends a new dispatch request to PMU only if it has a slot to absorb four jobs, and PMU may fill 1-4 jobs or even return an empty response.

When jobs are dispatched to MSU, the PMU decrements the MSU request counter. The PMU dispatches jobs in a container with one to four jobs.

The MSU stops requesting jobs when it has no pre-fetch slot to absorb them. MSU frees slots for jobs by scheduling jobs to threads. When the MSU gets a response with no jobs it can wait a configured time before requesting again.

8.6.1.2 Load Balancing

The MSU measures the CTOPs active thread load and quantifies the load into four levels.

The MSU reports to the PMU the thread processing load:

- Green – Low load
- Yellow – Some load
- Orange – High load
- Red – Overload (do not schedule more threads)

The PMU job scheduling arbitration is a two level hierarchy:

- Schedule to MSUs that are reporting Green, then to MSUs reporting Yellow, then to Orange MSUs.
- For each color group MSUs, scheduling is in a round robin fashion.

There is a round robin arbitration between applications that may use this MSU.

Per utilization load there is a round robin arbitration between applications. In this scheme more loaded applications get priority over less loaded applications through a mapping table. The input to the table per application base is its priority and utilization load and output of the table is its request priority for scheduling.

8.6.1.3 Messaging

Messaging is used for dispatching jobs from the PMU to the selected MSU to be serviced by threads in its NPC, and passing completed "Done" jobs indications from NPC threads back to the PMU for forwarding to the next pipeline stage or for evicting the job towards the traffic manager, transmit scheduling or loopback. The messages also include the load balancing information. Messaging between the NPC and PMU manages job budget accounting, as part of the system-level flow-control management.

8.7 Job Software Control

The NPS supports following list of SW actions for job management:

- Receive dispatched Job for processing.
- Complete processed Job and forward it (with/without order) to the TM or to TxNDMA.
- Complete processed Job and forward to re-queue in PQ (with/without order, with/without dispatch).
- Complete processing and terminate Job without forwarding.
- Create Job and queue in PQ for dispatch.
- Create Job and forward it to the TM or to TxNDMA.
- SW replicates dispatched Job to TM or re-queues it in the PQ (with/without order).
- Lightweight multicast utilizes a job container structure that points to an ordered list of leaf jobs and may be concatenated to another container job.

The SW uses the following services for job management:

Table 8-3. High level services for job management

SERVICE NAME	*ARGUMENTS*	*DESCRIPTION*
Receive job		Request a new job and copy its information to local CMEM storage. Gets in reply pointer to job handle in CMEM Gets in reply pointer to job descriptor in CMEM
Discard job	pointer to job handle in CMEM pointer to job descriptor in CMEM	Finalize a job, freeing it and all associated frame resources.
Dispatch job	pointer to job handle in CMEM pointer to job descriptor in CMEM with service keep order	Finalize a job, queuing to a PMU queue for further processing.
Transmit job	pointer to job handle in CMEM pointer to job descriptor in CMEM keep order	Finalize a job, sending it to the Traffic Manager to send it via indicated physical port.
Update job	pointer to job handle in CMEM pointer to job descriptor in CMEM with service keep order	Re-queue job to the next PMU queue and continue its processing.

Table 8-4. Low level services for job management

SERVICE NAME	*ARGUMENTS DESCRIPTION*	*DESCRIPTION*
Request job ID		Request a new job from the PMU. Reply: pointer to job handle in CMEM
Wait for job ID		Suspend execution until a job request is serviced and a new job is ready to process.
Return job ID Return job ID async	pointer to job handle in CMEM next dest info with service keep order	Finalize job, “Job Done”. Providing verdict on its next step, with possible parameter: Dispatch, Transmit. If special service is required, with service should be true. In order to maintain ordering, keep order should be true.
Free job ID Free job ID async	pointer to job handle in CMEM budget ID	Finalize a job, freeing it, keeping the associating frame resources.
Alloc job ID Alloc job ID async		Allocate Job ID from PMU.

SERVICE NAME	ARGUMENTS DESCRIPTION	DESCRIPTION
Update JD Update JD async	pointer to job handle in CMEM info with service keep order	Update the job in PMU. The job still under thread responsibility, job done should be called in order to finalize the job
Load job Load job async	pointer to job descriptor pointer to job handle in CMEM	Make a local copy of job data from IMEM to CMEM Reply: pointer to job descriptor
Store job Store job async	pointer to job handle in CMEM pointer to job descriptor	Sync local job data copy in CMEM to IMEM origin

8.7.1 Receive Dispatched Job for Processing

The PMU schedules up to four jobs to the MSU of the NPC via a message. The NPC allocates each job to the free thread for execution.

The SW thread is activated by the MSU when a dispatched Job is received and assigned for processing.

The activated thread invokes the “Receive-job” service to get the job.

8.7.2 Complete Processed Job and Forward (with/without order)

The SW process can forward the frame to TM for transmission, or forward the frame/job to a TM loopback port for deferred processing and/or multicast replication.

Additionally it can choose one of four PMU TM bypass queues which translate the job to a frame descriptor and send it directly to the TxNDMA on the same side of the PMU for immediate transmission. TxNDMA path does not involve TM scheduling and offers a high transmission rate for ordered queues, directly communicating with the TxNDMA COQ (Cell Output Queue) logic.

According to the usage, the SW thread can update the JD with required fields and write it back to its IMEM location before indicating a job done to PMU (and must ensure by data synchronization that it already resides on IMEM).

The JD contains Frame Descriptor with TM Info. The TM Info indicates the TM Flow queue, Packet Switch ID and other TM attributes. The JD format includes all the necessary TM control fields for outgoing jobs.

The SW uses the Transmit-job service to send the job to the TM.

The job “Done” message is sent through the MSU to the PMU which owns this job, and indicates the next destination is TM (distinguishing far TM from local TM) or one of the TM bypass queues that connect to the local TxNDMA direct transmission path.

When a frame is evicted from PMU to TM, the JD is converted to TM internal packet descriptor. The job ID and associated job descriptor are recycled. This enables holding a limited on-device amount of open jobs currently being processed by the NPC cores and holding very large amount of processed frames in TM queues. Similarly when the job is sent through TM bypass queue, the JD is translated to FD, and gets transmitted by TxNDMA though a selected logical port.

The TM scheduling hierarchy and Packet Switch ID schedule the packet to output interface or the loopback interface.

The TxNDMA can be utilized for traffic flush service. Sending to PMU flush queue which sends the FD to TxNDMA resource release machine and the frame buffers are released by HW instead of SW task.

8.7.3 Complete processed Job and Forward to Re-queue in PQ (with/without order, with/without dispatch)

The SW processing can forward the job to another PMU queue. This facilitates pipelined job processing, with guideline to the next pipelined processing stage.

A special forward operation can be used for processing a job in "run to completion model" without blocking other jobs in the PMU queue, thus removing head of line blocking in the PMU queue reordering logic. To activate this method, a thread sends a “Done” message the PMU to forward the job to another PMU queue with no additional dispatch operation (i.e. the job stays in the SW thread ownership).

The SW thread continues processing the job to completion while the PMU forwards the job to the required queue with/without order restoration as if the job has been completed in its current queue. To facilitate this operation, the PMU internally marks the job as done in its original queue and also marks it as already dispatched. This marking makes the job eligible to be removed from the current queue head of line (if order restoration is required) and pushed to the tail of the next PMU queue (when the time arrives, according to the original queue ordering). Later on when the SW thread sends a job done or job forward to another PMU queue (or to TM), the previously selected PMU queue is considered by SW the current queue and SW sends the message to the next PMU queue.

SW action to the job must always be sent to the PMU that currently owns the job. This is known from the MS bit of the PQID, where PMU-0 owns all PQID values with a cleared MS bit, and PMU-1 owns all PQID values with a set MS bit. If SW moves a job from one PQID to another PQID where the MS bit is toggled, the job is moved from one PMU to the other and changes ownership. In particular, in a run to completion model where SW sends the PMU a "forward without dispatch" message, it should acknowledge from the toggling of the MS bit that the job has changed PMU ownership. Forward without dispatch operations cannot be nested, and the second job done message must properly close the job, or forward it to the next PMU queue with normal dispatch.

8.7.4 Complete Processed Job and Terminate without Forwarding

On processing completion, the SW thread may decide to close the job (i.e. terminate the frame).

JD/JID is recycled back to the free pool.

SW uses the Discard-job service to terminate the job.

Alternatively, SW can send the JD to a PMU flush queue where the JD gets translated to FD and sent to TxNDMA recycle machine. Such a combined operation recycles both the PMU job and frame buffers.

8.7.5 Create Job and Queue in PQ for Dispatch

The application creates jobs for pipeline processing by other threads by allocating a unique JID from the PMU.

The JD is SW created in a thread’s scratchpad memory and copied (moved) to JD buffer memory in IMEM using the Store-job service to the address of the allocated job ID.

New jobs can be created by request messages from SW treads to the PMU.

SW uses the Allocate-job-id service to allocate a Job ID from the PMU.

SW uses the Dispatch-job service to dispatch the JID to the PMU.

8.7.6 Create Job and Forward to TM or to Direct Transmission

The application can create packets for transmission or create jobs for deferred processing by other threads. The TM loopback is used as a hierarchical job scheduler with replication capabilities.

The JD is typically created in a thread’s private scratchpad memory.

The JD is copied (moved) into JD memory using the Store-job service.

New jobs can be created by request messages from SW treads to the PMU.

SW use Allocate-job-ID service to allocate Job ID from PMU.

The SW uses the Transmit-job service to send the job to the TM or to TxNDMA through a TM bypass queue.

The Job-done message is sent through the MSU to the PMU which owns this job, and indicates the next destination is the TM.

When a frame is evicted from the PMU to the TM, the JD selected fields are converted to a TM internal packet descriptor. The job ID and associated job descriptor are recycled. This enables holding on-device a limited amount of open jobs currently being processed by the NPC cores and holding a very large number of processed frames in TM queues.

Additional frame attributes may be stored in the frame buffers for future processing. Typically such data is stored in the first frame data buffer preceding the frame header.

The TM scheduling hierarchy and Packet Switch ID schedule the packet to an output interface or the loopback interface.

8.7.7 SW Job Replication

A common case for job creation is light-weight multicast transmission. SW allocates a new job for every packet replica and allocates frame header buffers for the per-destination-different packet data part, while reusing shared buffers of the packet through multicast counters.

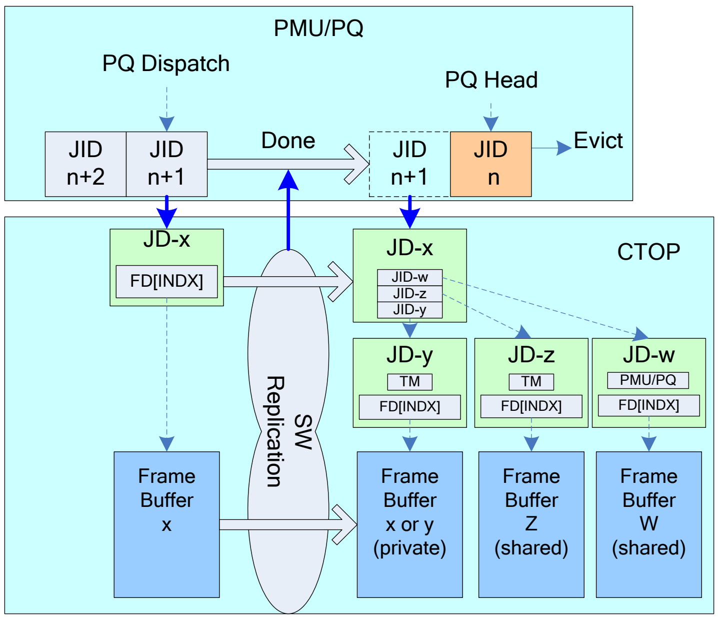
SW creates JD entries for each packet replica. The original JD entry is translated into a hierarchical job container that points to all the allocated JIDs. Finally SW sends the “Job done” message with the original JID to the PMU that owns it.

The PMU identifies the job as a hierarchical container and manages the request as an ordered list of jobs, according to the order they are pointed in the original JID entry.

Note that allocated empty jobs are not yet assigned to any PMU (they are not part of any PMU queue). Light weight multicast enables proper order restoration in the PMU. Only the root job is registered in the PMU and its order can be enforced relative to other jobs (possibly from the same flow).

The PMU atomically processes all the sub-jobs of the root JD and evicts them to the next processing stage according to the registered order of the root job. The next stage may be the TM (e.g. for multicast transmission/loopback) or other PMU queue pipeline processing stage (split of one job to multiple processing stages, similar to a "fork" operation).

Figure 8-7. Job creation for light-weight multicast transmission



In the figure above, PMU/PQ dispatches JID[n+1] to CTOP. The dispatched JID index Job Descriptor JD-x. The Job Descriptor contains Frame Descriptor FD with index [INDX] to Frame Buffer-x.

The SW processing allocates more frame buffers (y, z, w) and Job Descriptors via Job IDs.

The replicas may share frame buffers through multicast counters typically starting from the second buffer of each frame. The first header buffer as well as LBD buffer (if exists) are usually private to each frame replica.

The SW copies and modifies the packet data. Each replica is assigned a JD and an associated JID.

For each JD, the next destination attribute selects either TM (with target TM-info) or PMU re-queue.

The three JIDs are packaged in a container of the original JD-x.

The JID[n+1] that points to JD-x is messaged to PMU via the “SW Done” method.

8.8 Reordering and Evict

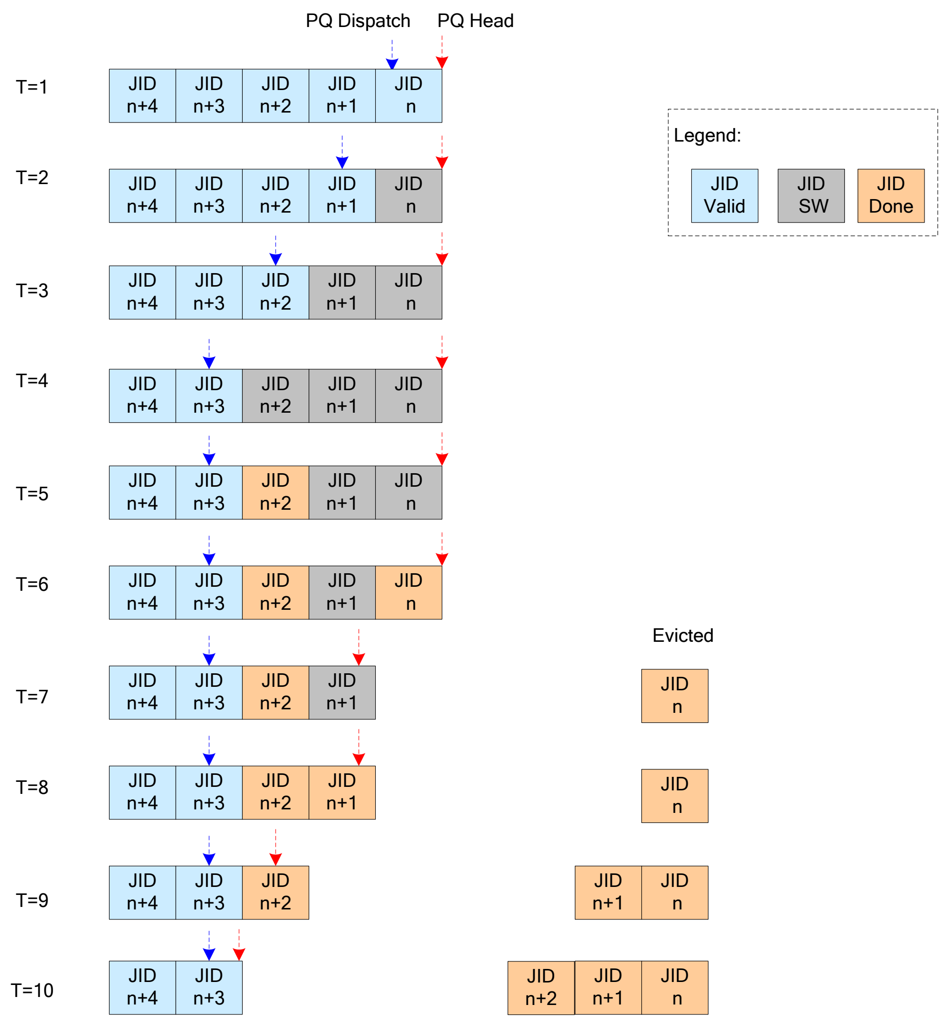
The PMU tracks the original order in the PMU queue from which the frame came from and on any forwarding operation it can enforce the same order before the frame is evicted to the next stage (either a PMU queue or TM).

The figure illustrates a PQ of five jobs [n..n+4] at time line T=1 to T=10.

At T=1 all jobs are valid for dispatch. At T=2 JID[n] is dispatched following in order by JID[n+1] and JID[n+2]. The JID[n+2] is completed first at time T=5. The JID[n] is completed second at T=6. The JID[n] being at queue head is evicted at T=7. The JID[n+1] is completed at T=8 and evicted at T=9, followed by JID[n+2] at T=10.

The PMU keeps the eviction order of jobs same in the same order as their enqueue.

Figure 8-8. Job reordering and evict

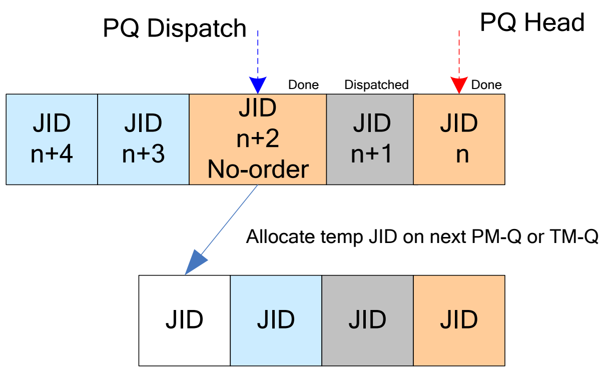


When the JID is forwarded to the TM, the associated JD information is converted to a TM frame descriptor and the JD/JID is recycled.

The SW may mark the job with “No-order”. The “No-order” marked job shall not be blocked by dispatched nor by preceding “Done” jobs. The “No-order” dispatched but not yet done job does block following jobs.

When a “No-order” job is Done, an additional job is allocated and pushed immediately to the target PQ or TM-Q, thus skipping over preceding jobs. The existing job on the original PQ is marked as both Done and already forwarded. When such a job gets to the head of queue, it gets recycled without a forwarding operation, therefore it does not block other jobs from advancing in the queue. The time during which both the original job ID and the newly created marked ID coexist in the system temporarily consumes an additional JID.

Figure 8-9. “No order” processing



8.9 TM Loopback HW Job Replication

Each TM Output Queue can be configured as a loopback port.

Each configured loopback port may be assigned to one of 8 Multicast Replicators (MR) in PMU-0 or PMU-1.

Each packet can be marked for multicast replication and number of replicas by TM Info on packet TM-enqueue. The TM Info is copied from the JD. JD MCC field in a loopback port has encoding 11b for replication.

When forwarded to loopback ports the JD.INFO[MCC] flag indicates:

- "1" –JD.INFO contains REPCNT[15:0] that is used for number of required multicast replicas.
- "0" –JD.INFO contains LBINFO[15:0] that identifies target PMU-QUEUE for a looped backed packet or other SW information while the PMU queue is selected by loopback port configuration.

The JD.INFO[REPCNT] or JD.IN|FO[LBINFO] is copied to TM FD[INFO] and preserved on looped back packets.

The FD[REPCNT] is used by the Multicast Replicator. The REPCNT field holds the initial value for replica generation. Setting REPCNT to zero means no replications are required (one replication received back with REPID=0). Setting REPID to one means one extra replication is required (e.g. one TM loopback frame creates two future jobs with REPID=0, 1).

The replication logic on the TM loopback path allocates and injects new jobs to the PMU from zero up to this REPID number.

The replica counter is inserted to JD.INFO[REPID] to identify each replica. On each injected job the FD[REPID] field indicates its replication number from zero to (REPCNT).

On a loopback frame, the echoed MCC flag indicates that the REPID field is valid and holds the current replication number.

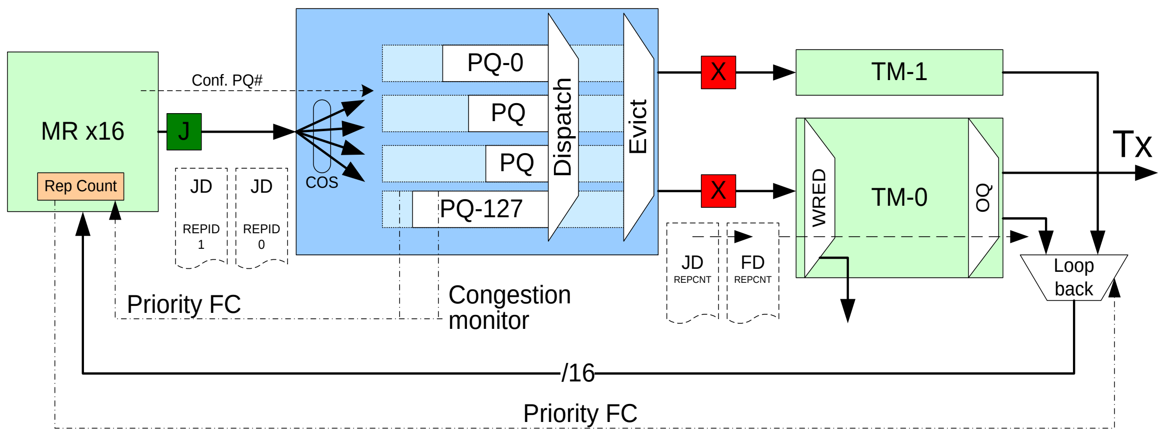
The target PMU-QUEUE for multicast replications is configured per MR unit.

The configuration allows allocation of range of up to 4 PQs per MR. The packets are queued to a PQ in range according to packet COS[1:0] assignment as it was remapped through TM scheduling hierarchies.

The PQ congestion per-priority flow controls the MR unit. The MR unit per-priority flow controls the TM loopback OQ. Resource deadlock is prevented by TM-WRED packet drop, since the TM never backpressures the PMU.

The TM loopback replicated jobs are pushed to the PMU according to the logical port budget configuration, and subjected to FCU flow control similar to incoming Rx packets. When the budget is full the injection stops and TM queues accumulate the future loopback packets.

Figure 8-10. TM loopback HW job replication



🟩 J Job Create

🟥 X Job Terminate

The TM supports 144 * (4 priorities) Cell output queues (COQ) at the output stage. The 16 COQs in range 128-144 are dedicated for loopback channels to the PMU. Each loopback COQ supports four priority queues.

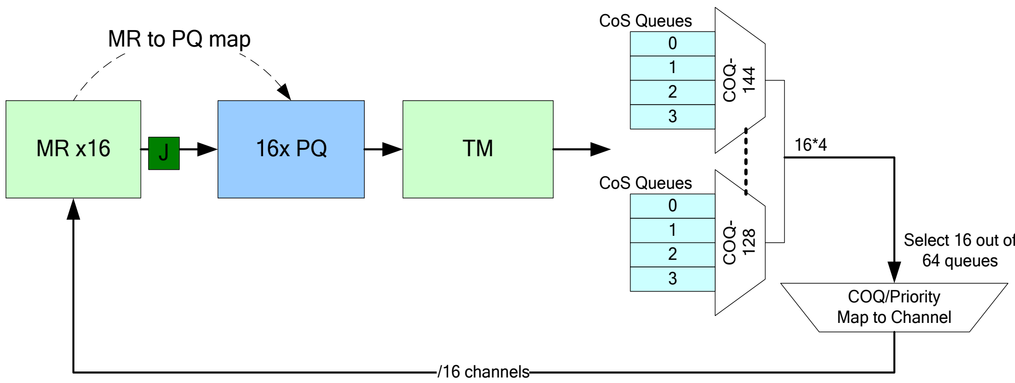
The CoS and COQ are selected by the Job Descriptor. The COQ is selected by various modes combining TM L1 hierarchy level and PSID.

The 16*4 queues can be assigned to 16 channels. The typical configurations are:

- The 16 COQs all same priority are assigned to 16 loopback channels.
- The 8 COQs x 2 priorities are assigned to 16 loopback channels.
- The 4 COQ x 4 priorities are assigned to 16 loopback channels.

Each loopback channel is served by a multicast replication (MR) unit.

Each loopback channel has a configured target PMU PQ.



8.9.1 Application Resource Remapping for Loopback Packets

The application SW usually performs job and frame buffer resources remapping for loopback packets.

Unicast Loopback

1. For a received frame both its frame buffers and job are allocated on budget ID-X (X being the budget ID assigned by the receive data-path configuration).
2. A CTOP receives the frame and processes it.
3. The CTOP passes the frame to the PMU to be sent to the TM loopback interface. At this point both the frame and job resources are still assigned to budget ID-X.
4. The PMU recycles the job resource from budget X and passes the frame to the TM. Frame buffer resources are still assigned to budget X.
5. The frame is passed to the TM. If it is discarded by WRED, the frame buffer resources are released from budget X.
6. When exiting the TM, the frame is passed to the PMU loopback interface which is associated with a dedicated loopback logical port.
7. The PMU allocates a new job resource on budget ID-L (L being the budget ID assigned to the loopback port in the PMU configuration).
8. A CTOP receives the looped back frame. The application may re-budget the frame buffer resources from budget X to budget L and updated the frame budget ID in the FD accordingly. In other scenarios, the application may assume the frame buffers are still associated with the input port (for example when Rx packet is getting deferred processing through TM loopback path).
9. The CTOPs processes the frame.
10. The CTOP passes the frame to the PMU/TM to be sent to the external interface. At this point both the frame and job resources are still assigned to budget ID-L. The job is associated with budget L, while the frame buffers are either associated with budget L or still associated with budget X, according to the SW verdict.
11. The PMU recycles the job resource from budget L and passes the frame to the TM (or directly to TxNDMA through TM bypass queues). Frame buffer resources are still assigned to budget L.
12. The frame is passed to the TM. If it is discarded by WRED, the frame resources are released from budget L.
13. The frame goes to the Tx DMA and frame buffer resources are recycled from budget L (or from budget X, in case that SW did not change the frame buffer budgets).

Multicast Loopback

1. For a received frame both its frame buffers and job are allocated on budget ID-X (X being the budget ID assigned to the port in the RX data-path configuration).
2. A CTOP receives the frame and processes it.
3. The CTOP prepares the frame for multicast transmission by marking the multicast counter of all buffers to N (N being the number of multicast copies).
4. The CTOP passes the frame to the PMU to be sent to the TM loopback interface, requesting to replicate the frame after the TM. At this point both the frame and job resources are still assigned to budget ID-X.
5. The PMU recycles the job resource from budget X and passes the frame to the TM. Frame buffer resources are still assigned to budget X.
6. The frame goes to the TM. If it is discarded by WRED, the frame resources are released from budget X. At this point the frame is still marked as a unicast frame (still has not undergone replication), so resources are recycled regardless of the multicast counter value.
7. When exiting the TM the frame is passed to the PMU loopback interface.
8. The PMU replicates the job. For each job it allocates a new job resource on budget ID-L (L being the budget ID assigned to the loopback port in the PMU configuration). It also marks each frame as a replicated broadcast frame.
9. A CTOP receive a frame replica. Each CTOP usually replaces the first buffer with a private copy. It allocates the new buffer on budget L and recycles the old buffer on budget X. The thread receiving the first replication (replication count=0) may (or may not) re-budgets the other frame resources (all buffers other than the first) from budget X to budget L and updates the frame budget ID in the FD accordingly. All other threads (receiving the other replicas) only update frame budget ID in the FD to budget L. It also updates the frame from replicated unicast to replicated multicast frame type. Broadcast flow is similar except that the header is replicated as well and all frames are identical. On multi-buffer frames with LBD, multicast frames also split the LBD buffers.
10. The CTOP processes the frame.
11. The CTOP passes the frame to the PMU/TM to be sent to the external interface. At this point both the frame and job resources are still assigned to budget ID- L. The frame data may stay on budget ID X; it is a SW decision on which budget to use.
12. The PMU recycles the job resource from budget L (or X) and passes the frame to the TM or to TM bypass queue. Frame buffer resources are still assigned to budget L.
13. The frame is passed to the TM. If it is discarded by WRED, the frame resources are recycled from budget L (or X) (the buffer is not released until all replicas are recycled).
14. The frame goes to the Tx DMA and frame resources are recycled from budget L (or X). (The buffer is not released until all replications are recycled.)
15. Since the frame is marked multicast, in all recycle cases, for the non-first buffer, only the last recycle actually frees the resources and budget. At that point it is guaranteed that the thread handling copy 0 has already re-budgeted the resources, thus there is no race condition. Recycle decrements the multicast counter and actually releases the buffer (affecting the budget) only if the MC counter reached a value of 0. Since copy 0 is still stalled, the multicast counter is guaranteed not to reach 0.

8.10 Queue Sequence Numbering

The PMU can assist in generating sequence numbers for a flow of packets. The packet sequencing allows in order flow processing. The ordered processing is important for stateful operations such as TCP/IP terminated flows.

Each packet of the flow is assigned with a sequence number at the time it is enqueued into a PQ according to its PQ receive order. The SW processing may utilize the packet sequence numbering to process packets according to assigned order. The distributed parallel SW thread processing may implement a counting semaphore per PQ to manage execution serialization.

The PMU maintains per PQ a sequence counter of 16b for job numbering. The sequence numbering can be enabled per PQ.

The packet job attributes to control sequencing are:

- Physical Queue Sequence Number (SEQNUM) – The 16b sequence number assigned to the job.
- V – The PMU sets the flag when a JD[SEQNUM] was updated with PQ sequence counter value.
- S – Service request. The SW created jobs passed to PMU may request setting the SEQNUM.

The CTOP sends JID via a message to the PMU. JID attributes may indicate “Service request” to update the SEQNUM by the PMU.

9. Buffer Management Unit (BMU)

The NPS architecture relies on the Buffer Management Unit (BMU) for the central management of memory buffer allocation and recycling. The BMU maintains lists of indexes that represent memory resources.

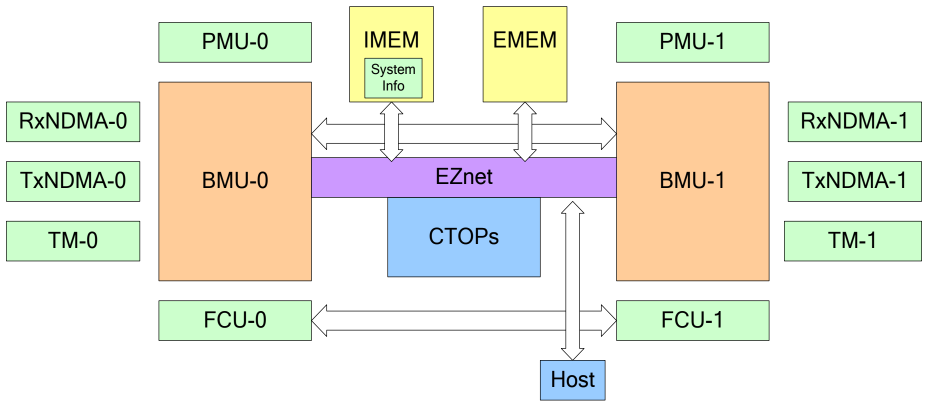
The BMU serves several architectural units:

- RxNDMA – The RxNDMA is served by allocating memory buffers for frame storage.
- TxNDMA – The TxNDMA is served by recycling the memory buffers after frame transmission.
- PMU – The PMU is served by allocating and recycling memory for Job descriptors.
- SW – The SW application uses the BMU for managing general purpose index lists for memory allocation and general index queue management.
- The BMU uses the Flow Control Unit (FCU) to manage the resource usage accounting of IMEM frame buffers, EMEM frame buffers and PMU jobs. The FCU is updated with allocation and recycle of resources.

There are two BMU units – one on each device side (west/east).

Each BMU serves the NPS units on its side. The two BMU units interconnect for resource sharing.

Figure 9-1. Internal blocks served by the BMU



The resources managed by BMU are:

- EMEM buffers for frames.
- IMEM buffers for frames and Jobs.
- General Index queues for SW usage.

The BMU manages lists (pools) of resources:

NAME	RESOURCE LOCATION	RESOURCE	UNIT SERVED	POOLS	# INDEXES / MANAGEMENT
EDP	EMEM	Frame buffer 256B	RxNDMA allocate TxNDMA recycle Allocate/recycle via dedicated messages.	24 sub-pools (one per DRAM controller) (12 sub-pools per BMU unit)	Up to 192M per pool. Managed in DRAM.
IDP	IMEM	Frame buffer 256B	RxNDMA allocate TxNDMA recycle Allocate/recycle via dedicated messages.	Shared	64K Managed in SRAM
IJP	IMEM	Job (IMEM 32B)	PMU allocate/ recycle	Shared	32K Managed in SRAM

NAME	RESOURCE LOCATION	RESOURCE	UNIT SERVED	POOLS	# INDEXES / MANAGEMENT
ETMP	EMEM	PD buffer 32B/64B/128B/ 512B	TM buffer for Packet Descriptors (PD)	2 sub-pools	Up to 192M per pool. Managed in DRAM.
ITMP	IMEM	PD buffer 32B/64B/128B/ 512B	TM buffer for Packet Descriptors (PD)	2 sub-pools	64K Managed in SRAM
SW pools		Index	CTOP/MTM	Up to 64 pools; 32 pools per BMU.	Up to 4 billion per pool. Managed in DRAM.

9.1 Feature List

- The number of indexes in each pool is configurable.
- EMEM pools: load balancing for RxNDMA between sub-pools (DRAM controllers) is achieved by recommending sub-pools to choose from.
  - Compound EMEM frame data pool comprising 24 sub-pools for DDR interfaces, each sub-pool is DRAM device bank aware for load balancing.
  - Both memory device fullness and bank load balancing are considered in the selection of best candidate index. The index explicitly selects the DDR device and the bank within that device.
- EMEM traffic optimization to release/get multiple management entries per access burst. Also minimizing the number of bursts by holding local storage.
- Transactions load balance EZnet routers, on both row and column, when managing an internal pool.
- Message optimized SW pool management by allocating 1-8 buffers in a single command.
- Graceful stop to enable warm restarts.
- Command triggered self initialization of index pool. Two working modes:
  - Self initialization – The pool is initialized at startup to be full of indexes.
  - The pool starts empty and filled up by the application (resource release).
- BMU serves as CTOP/MSU message channel for resource budget updates managed by FCU.
- BMU configuration direct access management by the control plane application including resource allocation and release.
- Four classes of budget counters and thresholds, mapped as colored 2-bit flow control that is mapped to an associated resource (e.g. RxNDMA ports/channels/global).
- Index error checking if index was recycled twice for IMEM.
- The BMUs are working in a distributed mode, east and west units. Each BMU serves its consumers locally but gets help from the other BMU unit. Nevertheless, each BMU unit can release buffers that were allocated by the other BMU unit.
- IMEM pools management:
  - Bitmap index management, located in BMU database (SRAM).
  - The indexes are split between the BMUs and can pass indexes to each other when one BMU unit runs out of indexes or, on the contrary, is overfilled with them.

9.2 Performance

BMU performance per side:

Table 9-1. BMU performance per side

POOL	ALLOCATIONS	RECYCLES
IDP	One per core clock per side, i.e. 1000M/s	One per core clock per side
IJP	One per core clock per side	One per core clock per side
ITMP	600M/s	600M/s
EDP	One per core clock per side	One per core clock per side
ETMP	330M/s	330M/s
Per SW pool	300M/s*	300M/s
All SW pools	600M/s	600M/s
Total	6B/s	

* A single SW pool can be configured for 600M/s actions. Typically multiple smaller pools are slower than 300M/s, according to their resource configuration.

9.3 BMU Index Pool Management

This section details management and capabilities of various index pools.

9.3.1 EDP – EMEM Frame Data Buffer Pool Controller

The EDP pool manages indexes to frame buffers in DRAM.

The buffer index allocation is executed per RxNDMA request while it is the TxNDMA that recycles the buffer index.

The SW application can also allocate/recycle the buffer index.

EDP 28b index construction:

- DRAM controller ID – 5 bits
- Buffer index – 23 bits: 2GB / 256B = 8M buffers of 256B each.

The EDP is managed as 24 sub-pools (12 sub-pools per BMU unit). For each pool one BMU unit acts as server while the other BMU is a client.

- A total 192M indexes can be managed: 24 sub-pools x 8M buffers per sub-pool.
- Per BMU there are 24 sub-pool managers (SPM):
  - Each SPM can be configured as: Server, Client, Independent (a subset of server where there happens to be no client), or Disabled.
  - Each BMU is typically configured to hold 12 pools acting as server and 12 pools acting as clients.
  - Master SPM fetches/sends indexes from/to the private BMU database in the EMEM.
  - Slave SPM accesses the EMEM through its master.
  - All SPMs are conjured to use the same MSID, and differentiate themselves by offsets in that space.

EDP pool features:

- The EDP has a configurable number of indexes in the pool.
- Configurable total number of indexes in all SPMs.
- The EDP pool supports graceful stop of indexes allocation.

- EMEM pool management utilizes index caching:
  - Hot cache of 768 indexes per sub-pool, SRAM implemented, is used for rapid response to some of the requests without incurring the long access latency database in DRAM. Per sub-pool Private SRAM database cache holds dynamic link lists for each DRAM bank.
  - Released indexes are stored internally in the BMU SRAM cache. When the BMU cache is overfilled or emptied, it fetches or sends additional indexes from/to the DRAM database. The cache re-fill management transactions use efficiently packed 128B DRAM transactions over EZnet.
  - There are 24 sub-pool manager units on each BMU; a unit for each DRAM device. The 12 units as masters can access the DRAM and the other 12 as slaves can give local service but have to access the remote cache corresponding master units on the other side in order to access the DRAM private database.
  - Cache uses a FIFO index caching policy – recently released indexes stored in the SRAM cache are the last to be allocated.
  - Configurable local cache overflow and underflow thresholds.

9.3.2 ETMP – EMEM TM PD Buffer Pool Controller

This pool serves the TM by managing a pool of Packet Descriptor EMEM buffers. Each BMU serves its adjacent TM.

- The pool is managed in EMEM.
- Built out of 2 SPMs: one master and one slave.
- The size of PD buffer is globally set: 32B/64B/128B/512B.
- Up to 192M PD buffers per pool.

9.3.3 IDP – IMEM Frame Data Pool Controller

The IDP pool manages indexes to frame buffers in IMEM.

The buffer index allocation is executed per RxNDMA request while it is the TxNDMA that recycles the buffer index. The SW application can also allocate/recycle the buffer index.

The IMEM buffer is 256B.

IMEM 16 [MB] divided to 256 [B/buffer] yields 64 [K buffers].

The 64K buffers are indexed by 16 bit IDP index.

The 64K index pool is managed as a bitmap in a dedicated SRAM structure. Shared pool or separated pools modes are supported.

The IDP has a configurable number of indexes in the pool. The IDP pool supports the graceful stop of indexes allocation.

The IDP supports two BMU collaboration mechanisms. When a pool is depleted below a configurable minimal size, the BMU attempts to receive indexes from the other BMU.

The IDP pool status elaborates the number of available indexes in each BMU and the total.

The IDP pool supports an index self-generation mechanism that can be utilized by system initialization. The self-generation mechanism has a configurable start index and amount of indexes to generate.

9.3.4 IJP – IMEM Job Pool Controller

The IJP manages the Job descriptor index pool.

The job descriptor allocation and recycling is executed per PMU request. The SW application can also allocate/recycle the Job via message to the PMU.

- Each Job descriptor is 32B in IMEM in addition to occupying per job internal PMU resources.
- IJP manages a pool of up to 32K jobs and uses a 16-bit job index. Shared pool or separated pools modes are supported.
- The 32K index pool is managed as a bitmap in a dedicated SRAM structure.
- The IJP has a configurable number of indexes in the pool. The IJP pool supports the graceful stop of indexes allocation.
- The IJP supports two BMU collaboration mechanisms. When the pool is depleted below a configurable minimal size, the BMU attempts to receive indexes from the other BMU.
- The IJP pool status elaborates the number of available indexes in each BMU and the total.
- The IJP pool supports an index self-generation mechanism that can be utilized by system initialization.
- The self-generation mechanism has a configurable start index and amount of indexes to generate.

9.3.5 ITMP – IMEM TM Pool Controller

This pool serves the TM by managing a pool of Packet Descriptor IMEM buffers. Each BMU serves adjacent TM.

- The pool is managed in IMEM. The pool is constructed of two sub-pools. Shared pool or separated pools modes are supported.
- The size of PD buffer is globally set: 32B/64B/128B/512B.
- Up to 64K PD buffers in pool, up to 32K per BMU sub-pool. The index is 16 bit.
- Manages the pool private DB in SRAM (bitmap).
- Communication to other BMU to fetch/send indexes.
- The ITMP has configurable number of indexes in the pool. The ITMP pool supports graceful stop of indexes allocation.
- The ITMP supports two BMU collaboration mechanisms. When pool depletes below configurable minimal size, the BMU attempts to receive indexes from other BMU.
- The ITMP pool status elaborates the number of available indexes in each BMU and total.
- The ITMP pool supports index self generation mechanism that can be utilized by system initialization.
- The self generation mechanism has configurable Start index and Amount of indexes to generate.

9.3.6 SW Pools

32 general purpose pools each accessible though each BMU in shared mode, or up to 64 independent pools (32 per BMU).

- Each pool is configurable with:
  - Hash search structure or SW pool use
  - Start index (for hash structure the start index is 0)
  - Number of indexes (for hash structure the number of indexes is set according to hash table size).
  - Number of cache RAM slices allocated (according to required performance)
  - Automatic initialization
- Cache RAM:
  - Shared 16 KB RAM per BMU is used to manage the pools
  - The cache RAM is managed as 32 slices of 0.5KB
  - The RAM used as pool allocation cache. Each pool can be allocated 0..10 slices (where sum of all slices is up to 32).
- Performance requirements from the SW pools:
  - Per pool: 320M/s x (allocation + release) in each side. For all 32 pools: 640M/s x (allocation + release) per side.
  - The number of cache RAM slices determine the pool performance: (number of slices 0..10) x 32M/s (allocation/release requests)
  - A single SW pool can be configured to provide full rate of 640M/s allocation + release. This can be achieved by reducing the service rate of other SW pools.

9.4 Application Interface

The CTOP application utilizes the BMU for index allocation and recycling. The CTOP accesses the BMU via MSU messages for FCU budget update. CTOPs can send message to either BMU for allocate / release/ budget management regardless which one manages the target pool.

The SW works with a subset of BMU pools through high performance messages (all pools are accessible though configuration accesses):

  - EDP, IDP, IJP – For frame buffer allocation recycle
  - SW – General use index pool
- The (host/CTOP) control plane application can allocate/recycle an index from any BMU pool. Configuration direct access management by control plane application including resource allocation and release.

9.5 System Info Update

The BMU updates system information structures in IMEM:

- FCU status:
  - Flow Control (FC) state and status per channel/port/group and global.
  - Priority Drop state per channel/port/group and global (from FCU and RxIF).
- Per BMU (HW/SW) pool the amount of available indexes for allocation.
- The BMU manages resource counters for TM IMEM packet buffers (allocated from IDP pool) that are waiting for TM transmission.
  - The application message increments the resource counter when a packet is passed to the TM.
  - The TM decrements the resource counter when a packet is transmitted and associated buffers are released.
  - The BMU periodically updates the System Info structure with counter value.

The BMU system info update is configuration controlled:

- The MSID for system info buffers.
- Base address for the Flow Control Report (FCR) with IMEM states.
- Base address for the Flow Control Report (FCR) with EMEM states.
- Base address for the Flow Control Report (FCR) with JD states.
- Base address for the Drop Precedence Report (PDR) for all memory spaces.
- Base address for the availability counters for all SW pools.

The system info update frequency is configurable.

9.5.1 Flow Control Report

The BMU collects the Flow Control (FC) state of budget counters from the FCU and periodically updates the System Info structure.

A Flow Control state buffer is written for each memory space (IMEM/EMEM/JD) to a different base address in the same MSID.

Each Budget ID (BID) has an FC state of 3 bits collected by BMU from FCU.

The FCU FC state is 3 bits per BID (channel, port and group). In addition, each BID has one FC status bit. Concatenation of the two yields 4 bits per BID:

FC = { FC_STATUS , FC_STATE[2:0] }.

The FC bit coding is:

- FC_STATE[2]: 0 - No congestion, 1- Some congestion (above lower threshold)
- FC_STATE[1:0] - Level of congestion: 00 (just above lowest threshold), 01, 10, 11 (above highest threshold).
- FC_STATUS: Bitwise OR of 8b Priority FC level. Set to 0 when there is no congestion.

The Budget ID identifies Channel, Port and Group. The System Info also contains the Global FC counter.

Budget ID = Channel 0..127 / Port 0..63 / Group 0..7 / Global

The IMEM TM Data Cache (DC) usage counter is also reported via System Info.

The BMU also updates the real fullness counters in System Info. These counters represent the amount of valid indexes left for allocation, 4B counter per HW pool (IMEM, EMEM, JD).

The FC buffer structure is as follows:

Table 9-2. Flow control unit report structure

LINE	*31:28*	*27:24*	*23:20*	*19:16*	*15:12*	*11:8*	*7:4*	*3:0*
0	Ch-7	Ch-6	Ch-5	Ch-4	Ch-3	Ch-2	Ch-1	Ch-0
1	Ch-15	Ch-14	Ch-13	Ch-12	Ch-11	Ch-10	Ch-9	Ch-8
...	...	...	...	...	...	...	...	...
15	Ch-127	Ch-126	Ch-125	Ch-124	Ch-123	Ch-122	Ch-121	Ch-120
16	Port-7	Port-6	Port-5	Port-4	Port-3	Port-2	Port-1	Port-0
17	Port-15	Port-14	Port-13	Port-12	Port-11	Port-10	Port-9	Port-8
...	...	...	...	...	...	...	...	...
23	Port-63	Port-62	Port-61	Port-60	Port-59	Port-58	Port-57	Port-56
24	Group-7	Group-6	Group-5	Group-4	Group-3	Group-2	Group-1	Group-0
25	Global Counter[27:0]							
26			DC Counter (IMEM only)		BMU Available Index Amount			

9.5.2 Drop Precedence

The Drop Precedence Report (PDR) contains 2 bits per BID (channel/port/group) per memory space (IMEM/EMEM/JD) – for a total of 6 bits from the FCU. An additional 2 bits of RX_PD status arrive from the RxIF for each physical port (0-47).

The 8 bit PD (priority drop) status:

PD_STATUS[7:0] = { IMEM_PD[1:0] , EMEM_PD[1:0] , JD_PD[1:0] , RX_PD[1:0] }

- *RX_PD is valid for P0-P47 only.*

The 2b Priority Drop coding is:

- 00 - Below low threshold
- 01 - Above low threshold and below middle threshold
- 10 - Above middle threshold and below high threshold
- 11- Above high threshold

PD_STATUS bytes will be arranged as follows:

LINE	***31:24***	***23:16***	***15:8***	***7:0***
0	Ch-3	Ch-2	Ch-1	Ch-0
1	Ch-7	Ch-6	Ch-5	Ch-4
...	...	...	...	...
31	Ch-127	Ch-126	Ch-125	Ch-124
32	Port-3	Port-2	Port-1	Port-0
33	Port-7	Port-6	Port-5	Port-4
...	...	...	...	...
47	Port-63	Port-62	Port-61	Port-60
48	Group-3	Group-2	Group-1	Group-0
49	Group-7	Group-6	Group-5	Group-4
50				Global

9.5.3 SW Pool Counters

The number of valid available indexes per SW pool per side.

Table 9-3. Number of valid available indexes per SW pool

LINE	***31:24***	***23:16***	***15:8***	***7:0***
0	SW_POOL0			
1	SW_POOL1			
...	...			
31	SW_POOL-31			

Note: System info of SW pool available indexes may indicate a temporal inaccuracy due to allocate/release update latency.

9.6 Flow Control Unit Interface

The FCU manages resource budget accounting. The managed resources are packet data buffers in IMEM and EMEM and Job descriptors in IMEM.

Each packet data buffer (allocated from IDP, EDP) is mapped to a Budget ID of 16b (BID).

The Job (allocated from IJP) is mapped to Job Budget ID (JBID).

The TxNDMA updates the FCU for the assigned budget by decrementing the budget counter.

The PMU manages the Job budgets in the FCU by incrementing and decrementing the budget counters.

The SW application can update the resource budget counter in the FCU by channeling the message through the BMU.

10. Traffic Management

The NPS-400 offers extensive traffic management capabilities on both the ingress and egress data paths through integrated traffic managers. Mechanisms are provided for advanced multi-service provisioning with programmable classification, metering and marking on the CTOPs, as well as congestion avoidance, traffic conditioning, traffic shaping and congestion management on the TM. This chapter describes the NPS-400 Traffic Management (TM) concepts, the queuing structures and traffic manager features. The traffic manager has autonomous operation, once the packet descriptor is submitted to TM the mechanisms perform all TM stages: WRED, queuing, scheduling, shaping, statistics and timeout.

The NPS-400 contains two TM engines that perform the traffic management functionality. The two TMs are identical, yet each one may be configured individually. Each TM engine can be used to transmit via any interface on its side. For line cards, interface sides can be assigned to ingress and egress traffic.

10.1 Key Features

- Two Traffic Manager engines with total sustained 480Gbps traffic for all packet sizes
  - Processing rate peaking up to 600M packets/sec. (300Mpps per TM)
- Both work conserving and non-work conserving scheduler
- Up to 64GB addressable memory space; up to 256M frames
  - Frame sizes from 1 byte to 12KB (the limitation is due to packet buffer structure LBD)
  - Per frame timestamp and timeout drop
- Internal Packet Switching decouples the scheduling entity from transmit port/channel
  - Per frame assignment of transmit port/channel
  - Aggregates scheduling levels
  - Fast reconfiguration of transmit ports for path resiliency
  - LAG shaping
- Per Flow Queuing (PFQ) with 5-level hierarchical scheduling of total TM capacity:
  - L0: 2x 128 interfaces (interface, groups of interfaces, loopback)
  - L1: 2x 0.5K Logical Ports aggregated to physical ports or interface channels
  - L2: 2x 4K Subports (or tunnels)
  - L3: 2x 32K Classes (subscribers, users)
  - L4: 2x 0.5M or 1x1M TM Flows (queues or services) – 16, 32 or 64 per Class
  - Dynamic mapping of L3-L0 hierarchies
- Congestion management policing by hierarchical WRED
  - 6-level hierarchical WRED with per color behavior profile (8 colors supported)
  - Policing/dropping schemes between the all WRED hierarchy levels
  - Configurable congestion accounting: TM-frame-buffers or frames/events
  - Buffer reservation for guaranteed memory resources
- Shaping:
  - Single and Dual token buckets (CIR, CBS, PIR, PBS) in L0-L3 hierarchies
  - Single token bucket (CIR, CBS) for L4 queues
  - IPG emulation
- Scheduling priority and fairness:
  - Programmable priority propagation in all hierarchies
  - Per queue single or dual WFQ
  - 5 strict priority WFQ schedulers in levels L0-L1. Highest priority (-1) for Fast queues.
  - 4 strict priority WFQ schedulers in levels L2-L3

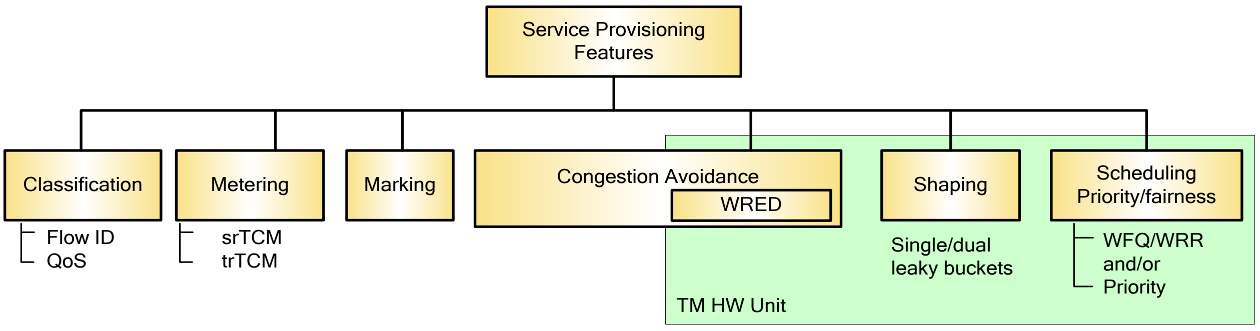
  - 8 strict priority WFQ schedulers in level L4
- Improved queue scheduling performance by internally managed queues in IMEM
- Up to 128 fast queues per TM for scheduling hierarchy bypass (direct attachment of L4 to L1)
- Advanced per packet control:
  - Per packet QOS IPG used for shaping, fairness and statistics
  - Per packet internal switching destination & COS
  - Per packet WRED profile for L3 and L4
  - Per packet pseudo-WRED
  - Selection of IMEM/EMEM memory per TM buffer
- Physical level connections:
  - Class-based hardware flow control per 512 MACs/channels by out-of-band (OOB) signaling, MAC link level flow control, TM congestion accounting, and SW activated flow control
  - Flexible flow control generation management scheme based on COS and per TM congestion accounting
- QCN support on queue aggregation levels
  - SW-based flow control for each entity in each TM hierarchical level L4-L0
  - Rapid shaper reconfiguration
  - TM hierarchical congestion monitoring
- Support for hardware and software based queue flushing for any TM hierarchical level entity

▸ *Note that in this chapter the term "flow" relates to the Level 4 of the TM hierarchical scheduler.*

10.2 Service Provisioning Overview

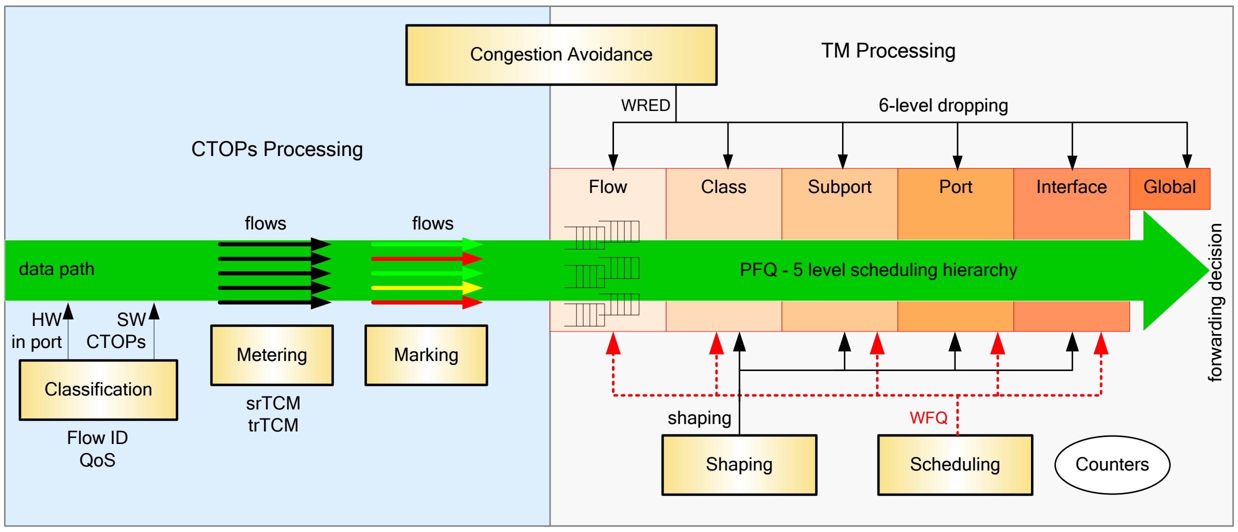
The NPS-400 offers various mechanisms to enable advanced service provisioning. The mechanisms are divided into the following categories as illustrated in Figure 9-1:

Figure 9-1. NPS-400 service provisioning features



The figure below illustrates where in the data flow the service provisioning features are implemented:

Figure 9-2. Implementation of service provisioning features



10.2.1 Classification

Classification is the process in which frames are assigned a flow ID (FID). Each flow ID is associated with the QoS (Quality of Service) and traffic contract parameters that apply to the TM flow. Classification in the NPS-400 is accomplished in two stages. The first stage is hardware-based (pre-classification) and is performed by the ICU at the input interface. The second stage is performed by the CTOPs and is fully programmable. Classification, for example, can be based on:

- 5-tuple search
- DiffServ PHB classifier
- VLAN
- MPLS
- Other content-dependent information

10.2.2 Metering

Metering refers to measuring a TM flow's traffic. A metering result determines whether a frame, which belongs to a TM flow, is within the expected traffic parameters for the flow or violates the expected traffic parameters. There are levels of traffic violation with most mechanisms working with three levels that are called colors. Traffic parameters that are metered are typically the traffic rate and traffic burst sizes. The NPS-400 uses the following Token Bucket mechanisms for traffic metering:

- srTCM (single rate three color marker) – RFC2697
- trTCM (two rate three color marker) – RFC2698
- trTCM (two rate three color marker) – Metro Ethernet Forum 5 specification
- Advanced Hybrid Token-Buckets (a.k.a. coupled policers)

The metering of frames in the network processor is performed by the NPS-400 token bucket counters in hardware, under control of the CTOP packet processing code.

The metering is implemented by Token Buckets in the Statistics Manager. For more details refer to the *Statistics Management* chapter.

10.2.3 Marking

Frames can be marked to reflect their metering results. Marking is usually done for the next hop in the network to prevent congestion. The marking of frames in the NPS-400 is performed by the CTOP software. Marking is also passed along to the TM for congestion avoidance. The 8-level color marking is used by TM WRED for profile selection.

10.2.4 Congestion Avoidance

In order to avoid congestion, early packet discards can be applied based on the congestion level and the frame's importance and metering results. The NPS-400 uses the WRED (Weighted Random Early Detection) mechanism for congestion avoidance.

WRED either drops the packet or passes it to the TM for queuing with per packet drop probability.

The metered frame color may be combined with other frame/flow properties to form an 8-level priority that is used for the congestion avoidance algorithm.

The WRED function consumes memory bandwidth:

- Queue Descriptors read.
- Frame Descriptor read.
- Packet data write.

The TM en-queue (WRED+ TM buffering) consumed bandwidth can be limited by configuration, freeing bandwidth for scheduling and transmission. Setting the TM bandwidth limitation results in backpressure to the CTOPs' application throughput.

- *For more information, see section 9.6.*

10.2.5 Traffic Shaping

Traffic Shaping enforces traffic rules on each TM hierarchical level. Traffic rules are based on bandwidth and burst parameters. Traffic eligibility for scheduling is determined by shapers in all levels. Non eligible traffic scheduling is deferred, or optionally scheduled as best effort priority.

- *More information, see section 9.7.*

10.2.6 Scheduling Priority and Fairness

During congestion, the allocation of the available bandwidth between the different TM entities must be determined. The following mechanisms are used in the NPS-400 for congestion management:

- **WFQ** – Weighted Fair Queuing enables each of the participating TM entities to receive a fair share of the available bandwidth, where fairness is defined by weights. WFQ profiles allow you to assign different weights to different entities to implement policy scheduling using the same WFQ mechanism. Dual WFQ may be imposed using each WFQ with different weights per priority shaper status (four colors in L0-3, and 8 colors in L4).
- **Priority Scheme** – TM flows can be assigned with up to 8 priority levels. Higher priority TM flows will be serviced before any other lower priority TM flows.
- Combination of both WFQ and priority.

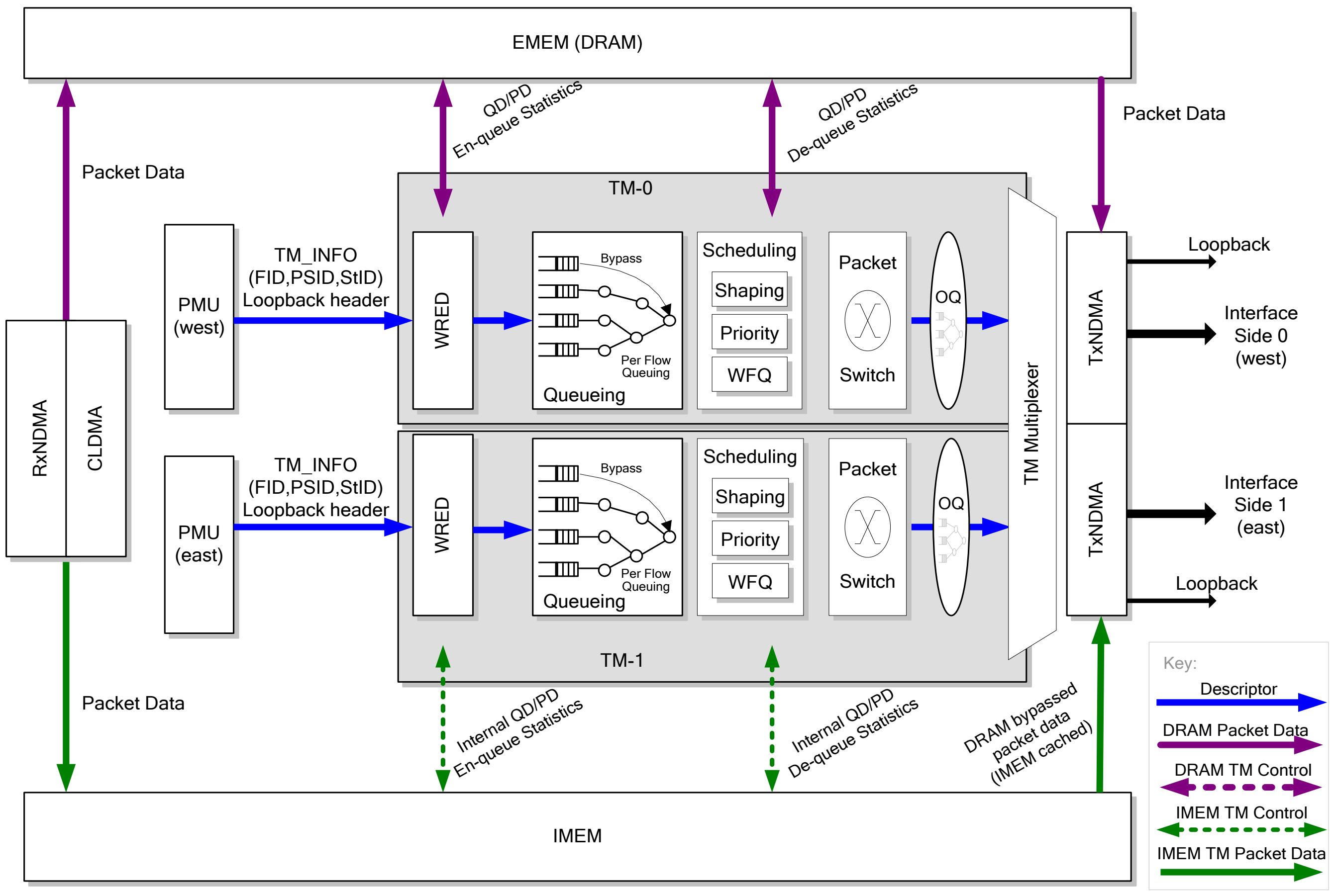
A propagated priority mechanism between the hierarchical levels in the TM ensures fastest service response under congestion for any high priority packet in any of the queues.

▸ *For more information, see section 9.8.*

10.3 TM Data Flow

Received packets are stored by the RxNDMA in the IMEM or EMEM (per buffer selection); see Figure 9-3. The packets are then processed by the CTOPs. The CTOPs can potentially move buffers from internal to external memory using the cluster DMA (CLDMA). The application can process the frame header in IMEM, and then move the modified header to a new buffer allocated in EMEM (in order to recycle the IMEM).

Figure 9-3. TM data flow



The RxNDMA generates a Job Descriptor (JD) that contains the SW frame descriptor and other fields. Software modifies the JD and when forwarding the job to the TM, the PMU converts the JD into a TM_INFO descriptor.

The TM_INFO is controlled by the CTOPs software by writing to a Job Descriptor (JD).

The main TM_INFO control fields are:

- Flow ID – the TM L4 flow queue - Normal/Fast/Internal-Queue-Descriptor queue.
- PSOFS[2:0] – Packet Switch ID offset for selecting up to 8 interfaces associated with the TM flow queue.
- COS[1:0] – packet's Class Of Service.
- WRED color, profiles and scale factor for L3 and L4.
- Scheduler IPG emulation.
- Loopback PMU channel 0-255.
- Number of multicast replications.
- TM statistics control.

The TM_INFO controls the packet's TM processing. The TM generates an internal TM packet descriptor which points to the frame location.

The TM descriptor contains the Flow ID (queue location in the TM topology), packet length, packet metering color, inter-packet gap (IPG) settings, and Packet Switch ID (PSID).

Packet descriptors go through the WRED block which makes a decision to either drop or queue the frame based on the hierarchical congestion level, the packet metering color and the TM hierarchical entity's WRED profiles. Packet descriptors are then queued into the PFQ (Per Flow Queuing) block according to the Flow ID. The WRED block updates the queue descriptor (QD) and the packet associated statistics with an enqueue event. The queue descriptors and statistics counters are stored either in the IMEM or EMEM.

The Hierarchical Scheduler schedules packets from the queues based on the QoS, bandwidth guarantee status at all the hierarchies and the scheduling weight. The Hierarchical Scheduler maintains traffic shaping, priority and WFQ fairness for executing the scheduling process.

Packet scheduling (de-queue) events update the QD and associated TM hierarchical statistics.

Scheduled packet descriptors are delivered to the Packet Switch and are placed into output queues based on the TM-L1/PSID destination and class of service associated with the frame.

Each TM engine also supports 128 queues for fast traffic scheduling. The 128 fast queues are directly mapped to 32 L1 ports (4 queues per L1 port). This direct mapping effectively bypasses the TM scheduling hierarchy and associated scheduling latency. The fast queues control structures are managed in dedicated internal memory.

The TM high bandwidth queues can be managed in IMEM for improved performance. These high bandwidth TM queues are called Internal-Queue-Descriptor queues.

Packets go to the TxNDMA for transmission via the TM Multiplexer. The TxNDMA sends the packet contents to the selected interface, or directs multicast traffic to the loopback queues for additional processing by the CTOPs. The TM Multiplexer allows packet switching from any TM to any interface on any side of the chip.

The TxNDMA reads the packet data from the IMEM/EMEM.

The Loopback packets can be replicated by the TxNDMA replication mechanism as instructed by the packet's Loopback control and get injected as new jobs for CTOP processing.

10.3.1 TM Packet Buffering and Control Resources

The application and TM configuration have full control over packet buffering and TM control memory bandwidth.

The RxNDMA typically places the first buffer of the packet in IMEM and subsequent buffers in EMEM, and under certain low threshold conditions (per port/channel) may also place other frame buffers in IMEM. Software may move selected buffers from IMEM into EMEM.

The TM accesses the buffer wherever they are located based on the buffer addresses that points either to the IMEM or EMEM memory spaces.

	BASED ON TM TOPOLOGY		*BASED ON CTOP SOFTWARE*	
	Queue Descriptor (4x 16B access per packet)	**Frame Descriptors** (2x16B access per packet) and Free List (CPTRQ)	**Packet Buffer** IMEM caching	**Packet Buffer** DRAM buffering
Regular queues	DRAM	DRAM	IMEM (for high priority and low latency)	DRAM (preferred)
IMEM QDs	IMEM	DRAM	IMEM (for high BW and low latency)	DRAM
Fast queues	SRAM	IMEM (overflow to DRAM)	IMEM (preferred)	DRAM

10.3.2 TM Packet Control

- The packet’s TM_INFO controls the TM per packet behavior
  - Frame type (selection of data structure)
  - Header offset: data start in the first buffer
  - Number of buffers per frame
  - Statistics accounting control:
    - Statistics ID (24b)
    - Code profile
  - Flow control
    - Source port for congestion management and flow control
    - Transmission confirmation
    - Multicast control (loopback duplication or selected release after transmission)
    - Keep buffers (no recycle)
- Queue type control
  - Flow queue ID (FID) / fast queue / IMEM QD
  - Loopback interface (PMU queue) replication count (16b)
- IEEE1588 timestamping
- WRED control
  - Do not drop packet / always drop
  - Metering/marking color 0-7
  - WRED profile and scale factor for L4
  - WRED profile and scale factor for L3
- Output-queue control
  - Packet Switch ID queue
    - TM L1 (1K) * 8 PSIDs
  - Queue CoS 0-3
- Traffic Shaping control
  - IPG emulation

10.4 Hierarchical Scheduler

The figures that follow illustrate the NPS-400's 5-level hierarchical scheduler. The scheduling decision is made from the bottom up in the figures. At each level, the scheduler picks the next (upper) level source, according to its priority, shaping and WFQ state, until the uppermost level is reached where a queue is selected, from which data will be transmitted.

As each frame enters the TM, the WRED mechanism uses a 6-level hierarchy dropping algorithm to decide whether to drop or pass the frame to the queue.

Details regarding WRED, shaping, WFQ and priority are provided in the sections that follow.

Although the queuing hierarchy is based on logical ports and interfaces at the lower levels, the packets are switched to the output interface based on the Packet Switch ID information combined with the TM L1 hierarchical entity. This enables a degree of flexibility in structuring the traffic management queuing and scheduling model to match physical requirements at the interface or system level. The TM L1 can service up to 8 physical interfaces/channels.

10.4.1 Hierarchical Scheduler Features

Hierarchical scheduling with 5-level mapping:

- Logical Interface level – Level 0 for up to 256 interfaces (128 per TM engine)
  - The TMs support transmission via any interface side
  - Flexibly associated to physical interface, or group of interfaces
  - Each TM engine can send to both sides via TM-Mux
  - The TM queue must serve one side
- Logical Port level – Level 1
  - Up to 1024 entities (512 per TM engine) which can be associated either to physical ports, or logical channels
  - Support for HW-based backpressure per physical-port / interface-channel
  - Support for SW-based flow control, OOB and/or TM congestion accounting
  - User-defined TM topology mapping of any Port to any Interface
  - Optional directly attached TM hierarchies:
    - Up to 4 fast queues per L1 entity
- Subport level – Level 2
  - Up to 8K Subports or tunnels (4K per TM engine)
  - Support for SW-based flow control by OOB and/or TM congestion accounting
  - User defined mapping of any 4 consecutive Subports to any logical Port (TM Level-1)
  - HW based hitless swapping between Subports
- Class level – Level 3
  - Up to 64K Classes, Users or subscribers (32K per TM engine)
  - OOB and SW-based backpressure per TM entity:
    - OOB flow control per entity
    - SW flow control per entity
  - User defined mapping of any 4 consecutive Classes to any Subport
  - Class bonding for 16 TM Flows per Class
  - Optionally 2 consecutive user bondings for 32 TM Flows per Class (dual Class bonding)
  - Optionally 4 consecutive user bondings for 64 TM Flows per Class (quad Class bonding)
- TM Flow level – Level 4
  - Up to 1M TM Flows, Services or Queues (512K per TM)
  - 16/32/64 TM Flows per Class dependent on Class bonding

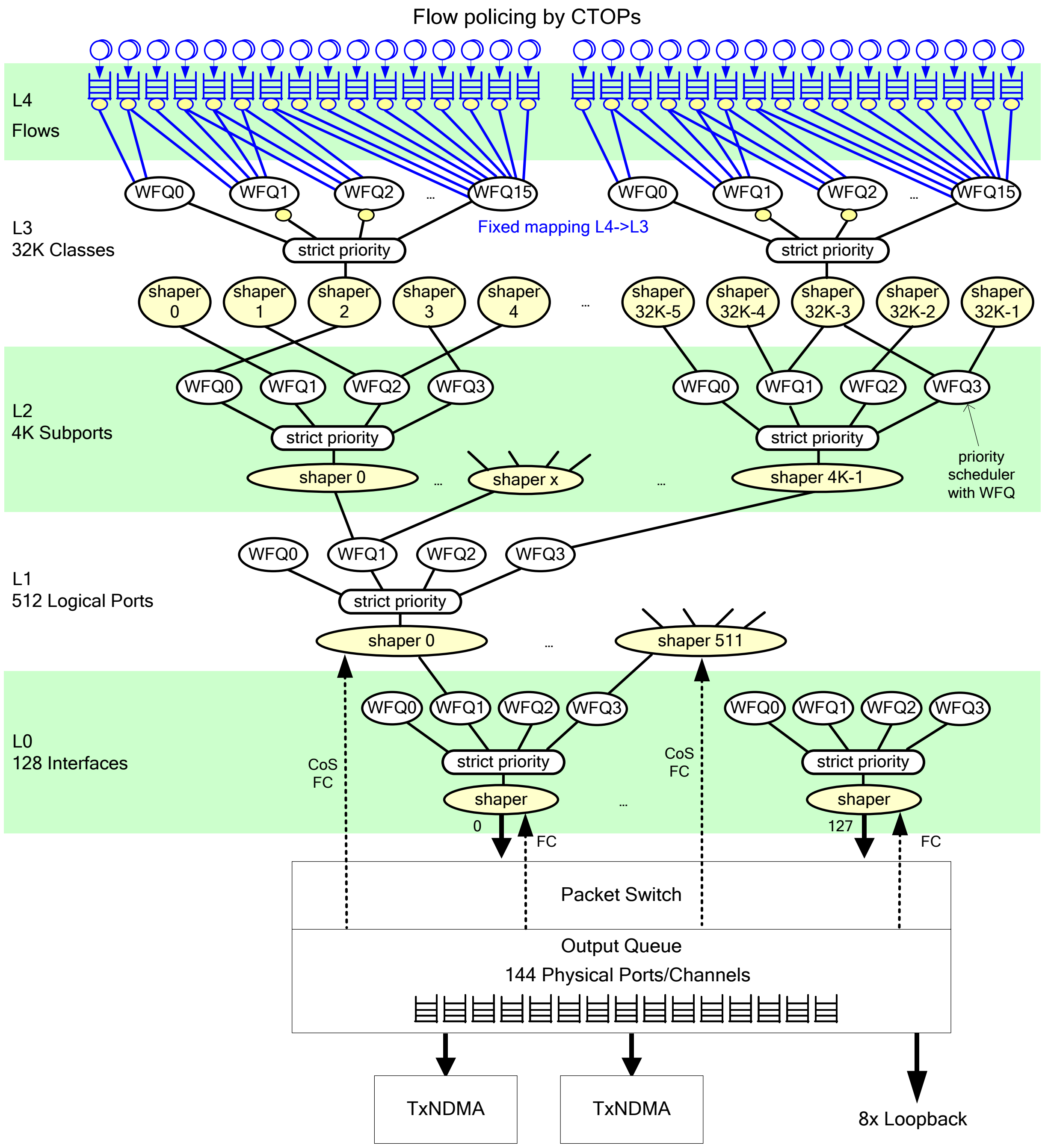
    - Up to 128 fast queues directly attached to L1 (four to one)
    - Internally managed queues (internal QDs) for high bandwidth queues
    - Support for SW based flow-control
- Queue flushing at any TM hierarchical entity

The following sections outline the hierarchical scheduler options for each of the two TM engines.

10.4.2 TM Engine 512K TM Flows Mode

Each TM engine normally operates in 512K TM Flows mode with 5 hierarchical levels, although not all queues or levels may be configured at all times and on all branches of the scheduler tree.

Figure 9-4. Hierarchical scheduler – with up to 512K TM Flows



10.4.2.1 Directly Attached Fast Queues

Up to 32 groups of four TM L4 dedicated fast-queue flows can be mapped to 32 TM L1 ports (x4 L4 to L1) in the scheduler tree. These queues are associated with queue descriptors management and frame descriptor management in internal memory. The mapping attaches the TM L4 flows to four existing WFQ priorities and to a dedicated highest priority (fifth scheduling priority).

The fast queues should be used for high packet rate traffic that does not require hierarchical TM QoS services.

The fast queues are unshaped per L4 entity.

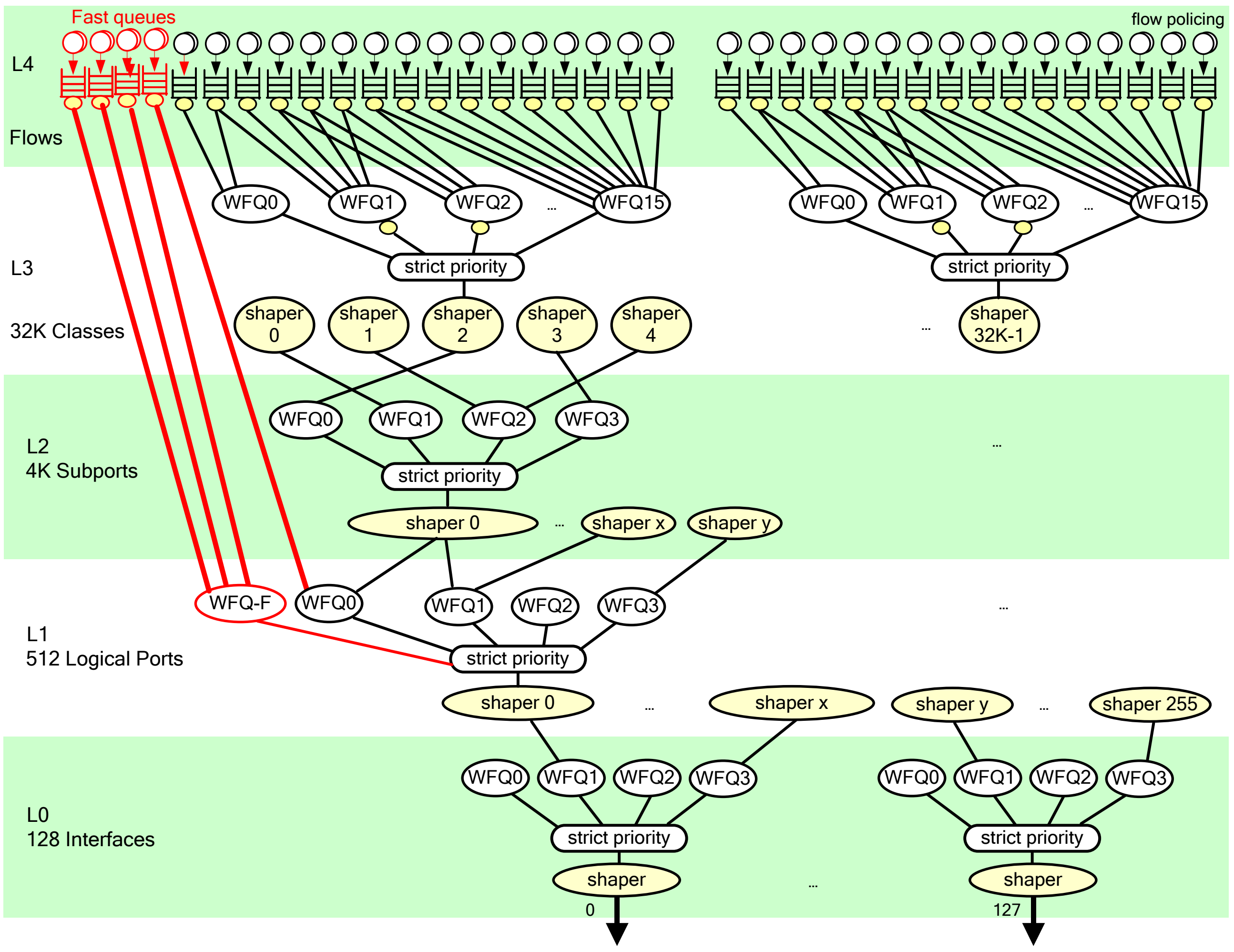
The fast queues have static scheduling priority assignment: Highest (-1), (0-3).

The fast queues have a single WFQ weight assigned.

The fast queues have a configurable size IMEM memory space for frame descriptor management and the free frame descriptors list. When IMEM FD space is full, the external DRAM is used for queues' FD storage until IMEM space is available.

The usage of fast queues saves DRAM bandwidth. Instead of six DRAM control transactions per packet, the IMEM and SRAM usage for fast queues performs only two transactions to IMEM.

Figure 9-5. Hierarchical scheduler – directly attached Fast queues



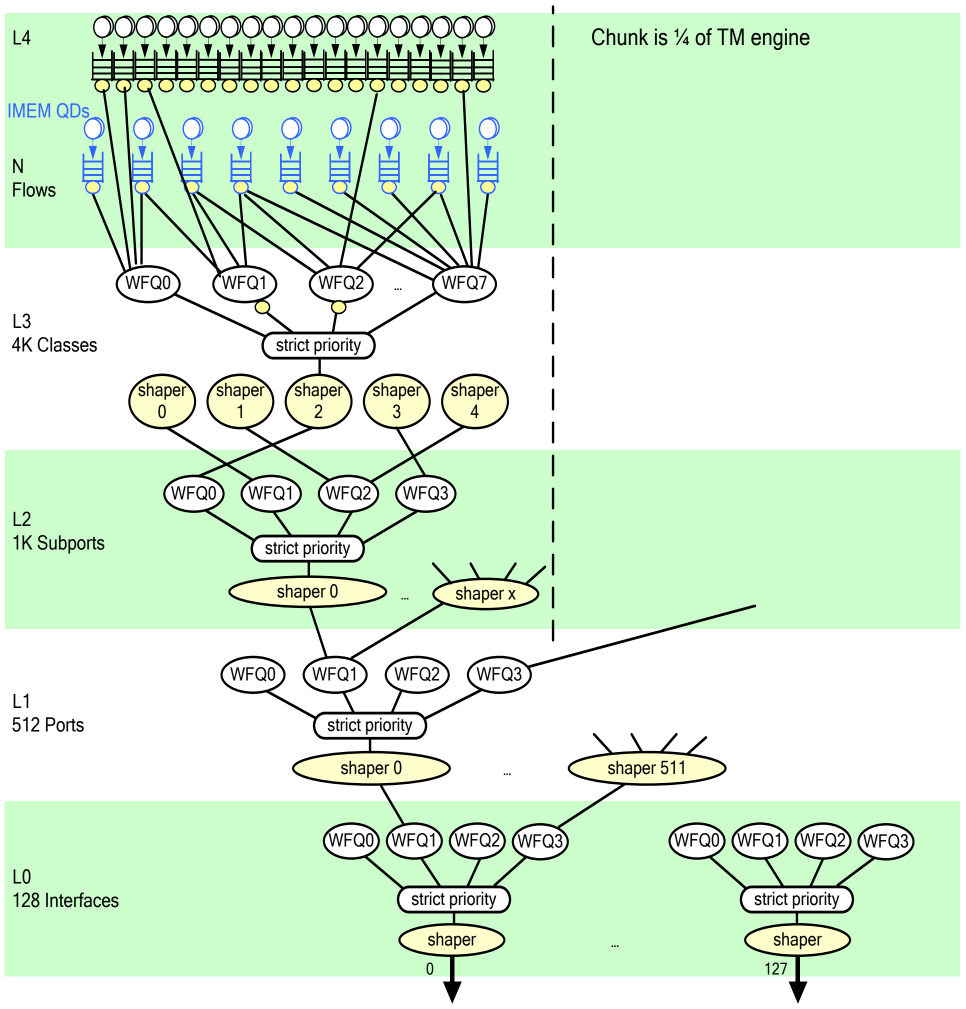
10.4.3 Improved Performance for TM Flow Entities (IMEM QDs)

The usage of fast queues saves DRAM bandwidth. Instead of six DRAM control transactions per packet, the IMEM and SRAM usage for fast queues performs only two transactions to IMEM.

The TM can operate with internally-managed TM Flows with the full 5 hierarchical levels. This mode enables improved performance operation especially for small sized packet traffic; by managing selected queue descriptors internally in IMEM.

The improved-performance TM Flows are superimposed on 512K TM Flows by flexibly selecting the internally managed Flows out of the 512K. The remaining flows are managed by external DRAM queue structures. Up to all 512K can be enabled for internal queue management. Each improved-performance TM flow consumes 16B of IMEM.

Figure 9-6. Hierarchical scheduler – with improved-performance flows



The 2^K TM flows are configured by selecting P bits (out of 19 bits) of a Flow index of a TM, and comparing the selected P pointer bits to a configurable constant. The remaining K=19-P bits are an index to $N=2^K$ internally managed flows list.

Examples:

- Selecting Flow index 5 LSB and comparing to 0, selects the Flow 0 of Class-0 each group of 2 Double Bonded Classes: 0, 2, 4 ... The internal flows are 0, 20H, 40H, 60H …
- Selecting Flow index bits [8:4] and comparing to 0, selects all 16 Flows of first Class of each group of 32 Double Bonded classes. The internal flows are 0-15, 200H-20FH, 400H-40FH, 600H-60FH …
- Selecting Flow index 5 MSB and comparing to 0, selects the 8K flows 0- (16K-1).

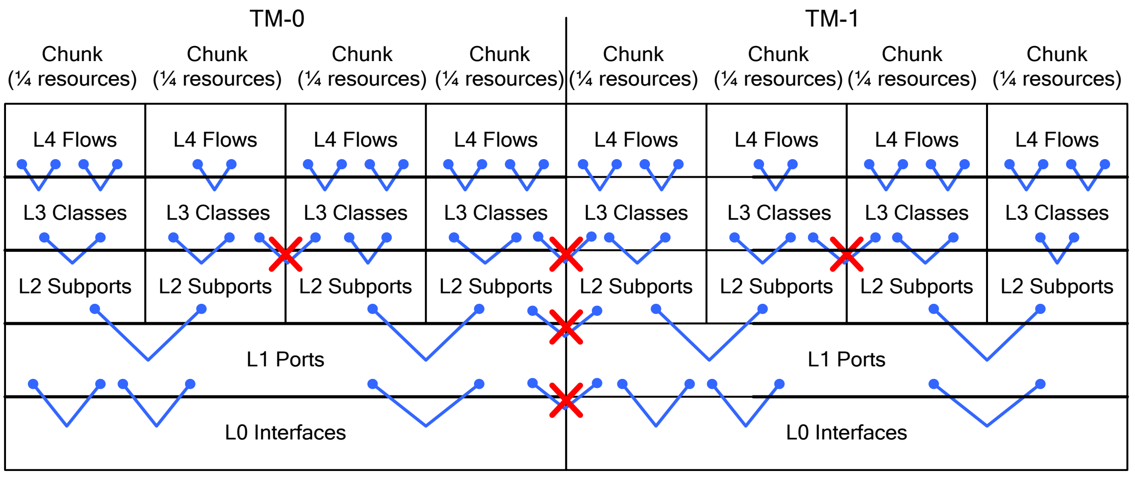
10.4.4 Mapping of Chunks per TM Engine

Although Port-to-Interface and Subport-to-Port mapping is fully flexible per TM engine, there is a restriction when it comes to the mapping of Classes to Subports per TM. The TM's mapping notation relates to a 'chunk' entity, which is basically a group of 1K Subports.

Each TM is evenly divided into four such chunks and each chunk holds one-quarter of the higher levels' resources (e.g., a chunk contains 1K Subports, 8K Classes and 128K TM Flows).

The mapping of Classes (and their TM Flows) to Subports must remain within the chunk's boundaries. For example, a Class from the range 8K-16K may only be mapped to a Subport from the range of 1K-2K.

Figure 9-7. Each of the four chucks contains one-quarter of the TM Subports, Classes and Flows



The chunks also extend the range of available configuration profiles for L2/L3/L4 Shaping and WFQ as each chunk has separate profile configurations.

10.4.5 TM Flow to Class Level Mapping – Class Bonding

The NPS-400 TM allows for the mapping of several TM Flows to a Class. Each Class entity contains two shapers and eight Flows. At the Flow level, each Flow contains an additional dual shaper. It is possible to bond together two or four consecutive Classes to form a scalable Class entity with more shapers and Flows.

The above modes are selectable for each group of 4 consecutive Classes.

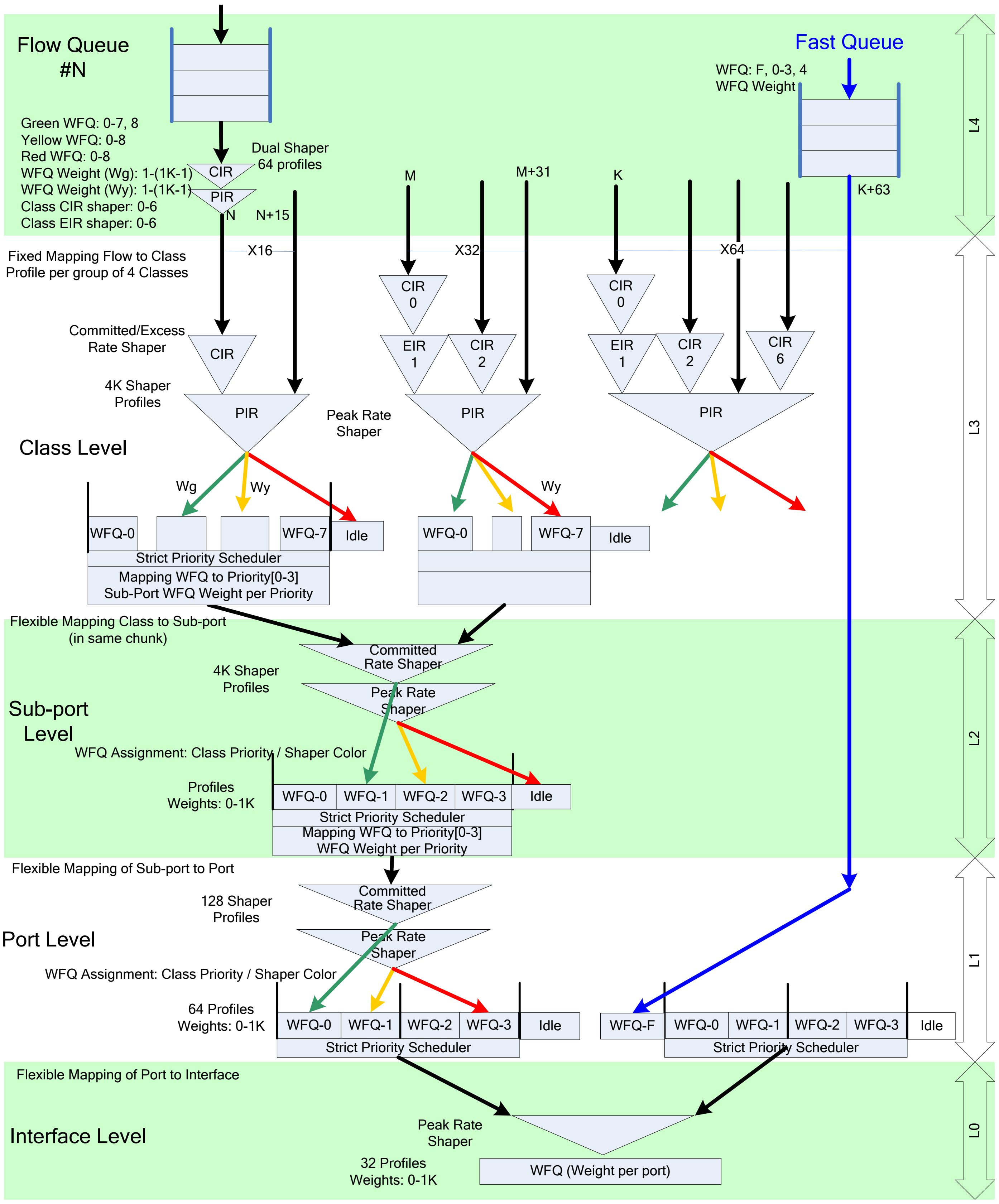
The three Class modes provide (see Figure 9-8):

MODE	*MAX CLASSES*	*TM FLOWS PER CLASS*	*SHAPERS PER CLASS*
Single Class	32K	16	Two: 1 peak rate 1 commit rate
Dual Class bonding	16K	32	Four: 1 common peak rate 3 excess/commit rate
Quad Class bonding	8K	64	Eight: 1 common peak rate 7 excess/commit rate

The assignment of the shapers to TM Flows and the Flow's behavior is described in the Service Profile section.

The fast queue configuration reclaims up to 128 Flow-IDs per TM to bypass the TM hierarchical scheduling. The fast queues are usually associated with the highest WFQ priority (-1).

Figure 9-8. Class bonding



10.4.5.1 Service Profile

The TM enables extended flexibility in the per Flow service mapping.

The configuration that defines the service characteristics of all queues in the Class is called a service profile. There are eight service profiles per TM engine. There are 4 service profiles for normal classes. An additional 4 profiles are available for double- and quad-bonded classes. Each 4 consecutive Classes are mapped to any of the service profiles.

Each TM Flow may be assigned to a different service level based on the shaper status of the commit and excess shapers. For example, a data Flow may get a higher priority when the dual shaper is green (both excess and commit shapers are green) and a lower priority when the dual shaper is yellow (excess shaper is green). Configuration also allows to permit scheduling when a dual shaper is red (excess shaper is red), however this feature should only be used for low bandwidth Flows where low delay and low jitter are crucial. When only a commit shaper is allocated, traffic is considered green as long as the commit shaper is green, otherwise it is considered excess and is only limited by the total peak rate of the Class as well as the per-Flow shapers.

The following table summarizes the service profile configuration:

Table 10-1. Per user service configuration

QUEUE NUMBER	*0*	*1*	*2*	*3*	*4*	*5*	*…*	*31*
Committed priority	0-8	0-8	0-8	0-8	0-8	0-8	0-8	0-8
Excess priority	0-8	0-8	0-8	0-8	0-8	0-8	0-8	0-8
Red user priority	0-8	0-8	0-8	0-8	0-8	0-8	0-8	0-8
Commit Shaper	NA, 0-6	NA, 0-6	NA, 0-6	NA, 0-6	NA, 0-6	NA, 0-6	NA, 0-6	NA, 0-6
Excess Shaper	NA, 0-6	NA, 0-6	NA, 0-6	NA, 0-6	NA, 0-6	NA, 0-6	NA, 0-6	NA, 0-6

*Priority 8 states 'no-transmission allowed' (same as red color).

The committed, excess and red priorities define one of eight priority levels (0-7) that each Flow is assigned under the committed/excess/red state of the Class shaper. Priority level 0 is the highest priority and Priority 8 refers to no transmission allowed. The shaper combination which is defined for each Flow determines which shapers to consume credits from. Up to three Class shapers are updated per scheduling operation. A common peak shaper is always updated per Class and an additional two shapers (commit, excess) can optionally be defined. In addition to this, a per-Flow dual-rate shaper can be defined at the flow level (L4).

In addition to the peak shaper, in single Class mode (with 16 Flows per Class) there is only a single committed shaper option; in Dual Class bonding three CIR/EIR shapers are available; and for Quad Class bonding seven CIR/EIR shapers are available.

The fast queue's service profile selects a single scheduling priority: Fast, 0-3, (4 is Pause, not supported).

The fast queue has a single configurable WFQ weight.

The fast queues bypass the TM hierarchical scheduling and are shaped by L1/L0 shapers only.

The fast queues are not shaped per L4-L2 levels.

The following are typical service profile examples for converged voice, video and data Flows per Class.

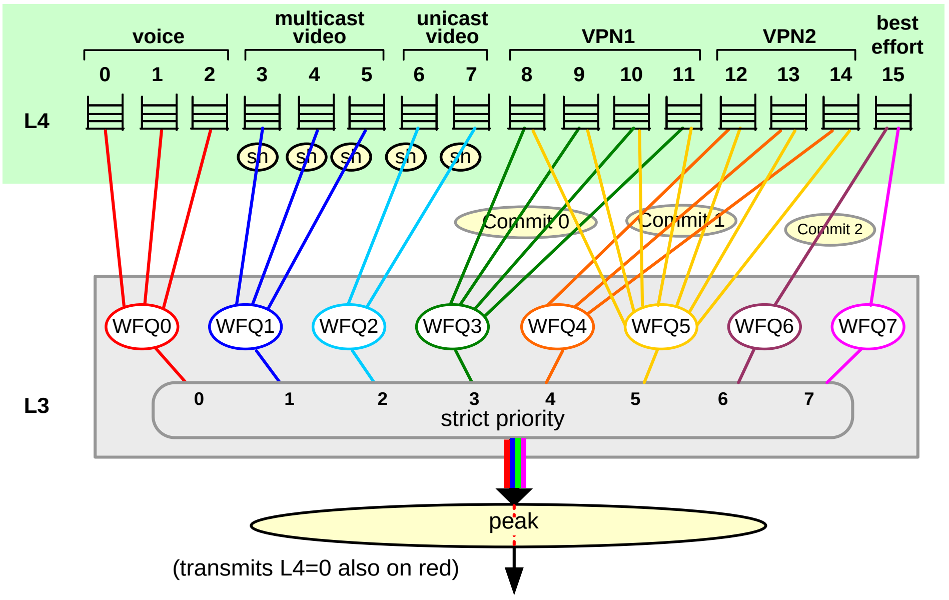
Example 1:

Table 10-2. Typical service profile example 1

Queue Number	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
	voice			**multicast video**			**unicast video**		**VPN1**				**VPN2**			**best effort**
Commit priority	0	0	0	1	1	1	2	2	3	3	3	3	4	4	4	6
Excess priority	0	0	0	1	1	1	2	2	5	5	5	5	5	5	5	7
Red user priority	0	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
Commit Shaper	NA	NA	NA	NA	NA	NA	NA	NA	0	0	0	0	1	1	1	2
Excess Shaper	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA

The following figure exemplifies the Class services and scheduling diagram:

Figure 9-9. Service profile example – 16 TM Flows



In the following example, in addition to the service profile, L4 shapers are configured to rate limit the video traffic streams by an L4-PIR shaper. Also, different WFQ weights may be configured for each of the Flows based on the L4-CIR shaper for further class differentiation within the same priority level, which is primarily of importance for data services.

The example places the guaranteed traffic of voice and video on strict priority levels 0-2 while using levels 3-4 for data committed services, and priority 5 for excess data traffic. The commit shaper meters only the data Flows allowing data traffic bandwidth guarantee among the rest of the Classes. Service levels 6-7 are used for best effort traffic and utilize the remaining bandwidth resources up to the peak shaper capacity.

Since priority propagates throughout the levels, bandwidth will first be allocated to any guaranteed or higher priority service before the excess or best effort traffic is scheduled.

Example 2:

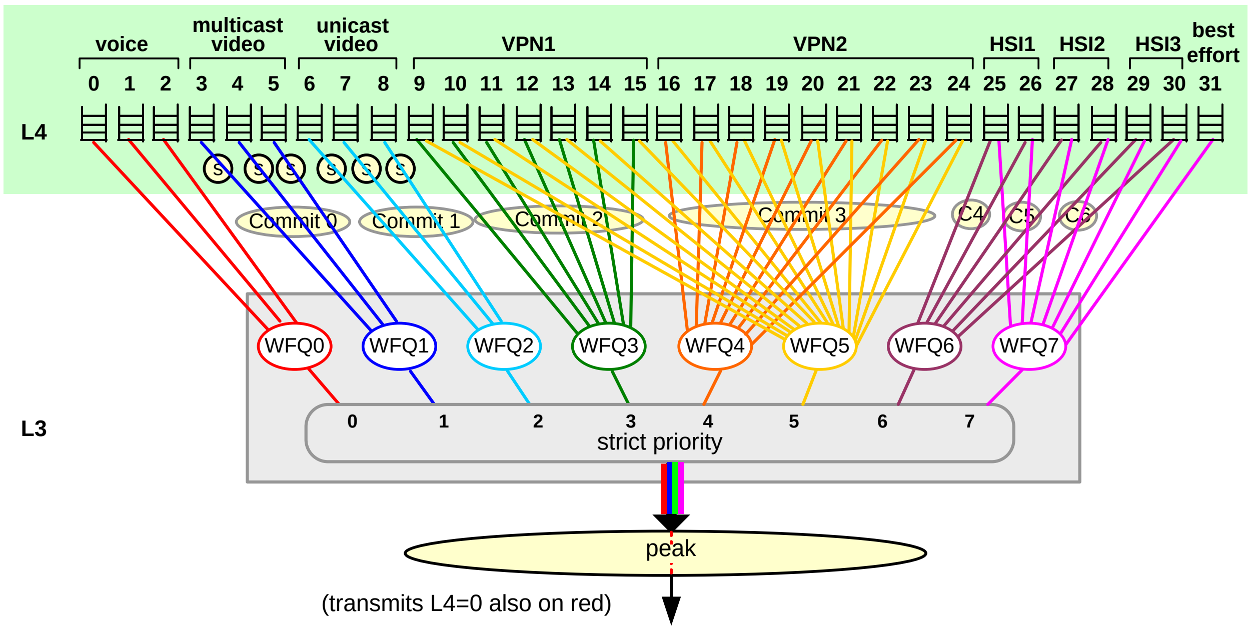
Table 10-3. Typical service profile example 2

Queue Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	voice			multicast video			unicast video			VPN1						
Commit priority	0	0	0	1	1	1	2	2	2	3	3	3	3	3	3	3
Excess priority	0	0	0	8	8	8	8	8	8	5	5	5	5	5	5	5
Red user priority	0	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
Commit Shaper	NA	NA	NA	0	0	0	1	1	1	2	2	2	2	2	2	2
Excess Shaper	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA

Queue Number	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	VPN2									HSI1		HSI2		HSI3		best effort
Commit priority	4	4	4	4	4	4	4	4	4	6	6	6	6	6	6	7
Excess priority	5	5	5	5	5	5	5	5	5	7	7	7	7	7	7	7
Red user priority	0	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
Commit Shaper	3	3	3	3	3	3	3	3	3	4	4	5	5	6	6	NA
Excess Shaper	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA

The following figure exemplifies the Class services and scheduling diagram:

Figure 9-10. Service profile example – 32 TM Flows



Example 3:

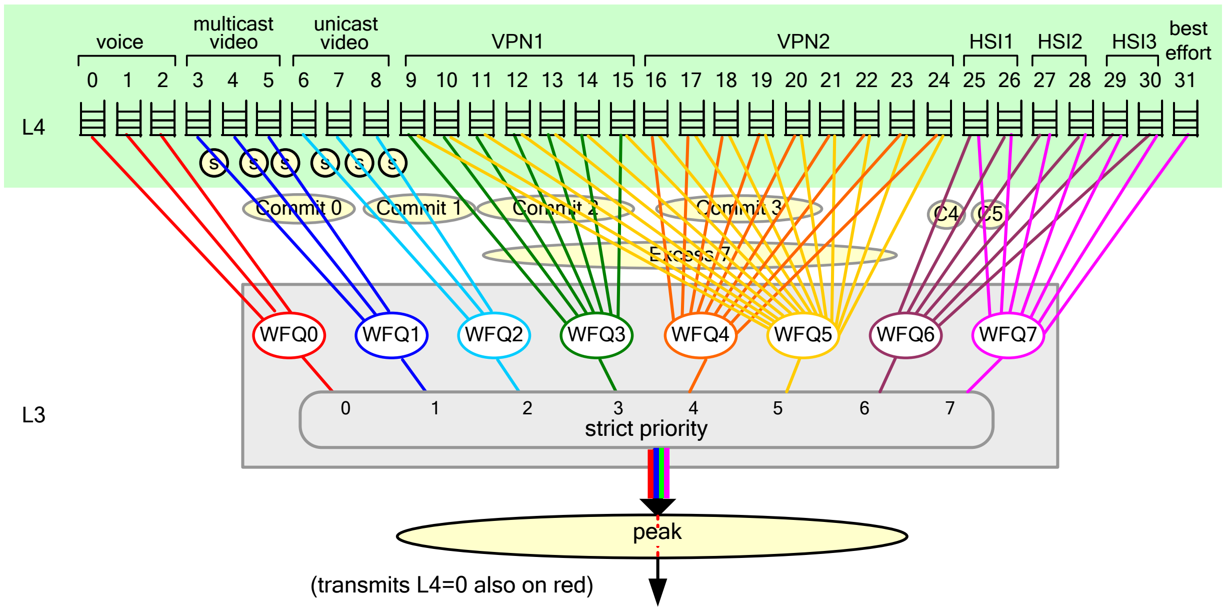
Table 10-4. Typical service profile example 3 uses both commit and excess shapers for services

Queue Number	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
	voice			**multicast video**			**unicast video**			**VPN1**						
Commit priority	0	0	0	1	1	1	2	2	2	3	3	3	3	3	3	3
Excess priority	0	0	0	8	8	8	8	8	8	5	5	5	5	5	5	5
Red user priority	0	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
Commit Shaper	NA	NA	NA	0	0	0	1	1	1	2	2	2	2	2	2	2
Excess Shaper	NA	NA	NA	NA	NA	NA	NA	NA	NA	6	6	6	6	6	6	6

Queue Number	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
	VPN2									**HSI1**		**HSI2**		**HSI3**		**best effort**
Commit priority	4	4	4	4	4	4	4	4	4	6	6	6	6	6	6	7
Excess priority	5	5	5	5	5	5	5	5	5	7	7	7	7	7	7	7
Red user priority	0	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
Commit Shaper	3	3	3	3	3	3	3	3	3	4	4	5	5	NA	NA	NA
Excess Shaper	6	6	6	6	6	6	6	6	6	NA	NA	NA	NA	NA	NA	NA

The following figure exemplifies the user service queues and scheduling diagram. The excess shaper is used in order to allow for oversubscribing the commit traffic of different services.

Figure 9-11. Service profile example – both commit and excess shapers for services



10.4.5.1.1 Shaper Status Effects

The Flow aggregation to the TM Class level (L3) is shaped by up to three shapers in serial: CIR, EIR and PIR.

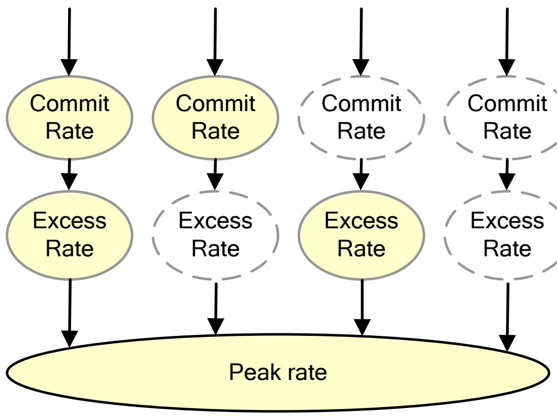
- With a single Class, one Commit shaper and one Peak shaper are available.
- With dual Class bonding, three Commit/Excess rate shapers and one Peak shaper are available.
- With quad Class bonding, seven Commit/Excess rate shapers and one Peak shaper are available.

The L3 service profile configuration determines the shapers that affect the flow scheduling priority (WFQ0 - WFQ7, Pause) and shapers that are updated by the flow scheduling event.

The figure below details all possible shaper concatenations affected by the shaper status:

- Only Peak rate shaper.
- Excess rate and Peak rate shapers.
- Commit rate and Peak rate shapers.
- Commit rate, Excess rate and Peak rate shapers. (Not supported for a single Class.)

Figure 9-12. Shaper status



Each shaper can be configured as:

- Always conformant with rate or shaper not present. Mark status as (+).
- Always non-conformant with rate. Mark status as (-).
- Shaper status per rate conformance. Mark status as (-)/(+).

The Commit/Excess/Peak rate shapers cascading yields combined shaper status as Green, Yellow, Red:

COMMIT RATE SHAPER STATUS	EXCESS RATE SHAPER STATUS	PEAK RATE SHAPER STATUS	COMBINED SHAPER STATUS
+	+	+	Green
-	+	+	Yellow
Ø (don’t care)	-	+	Red
	+	-	
	-	-	

The combined shaper status determines the flow scheduling priority per service profile:

- Green: Commit priority
- Yellow: Excess priority
- Red: Red Priority

10.4.5.1.2 WFQ Priority Remapping between Levels

Each group of 4 Classes selects one of 8 remapping tables that applies the priority remapping function to all 4 Classes. The flow scheduling priority L3-WFQ0 : L3-WFQ7 is remapped to 4 priorities supported by lower levels: WFQ0 - WFQ3. The remapping is done by a 64 entry table. There are 8 remapping tables selectable per group of 4 Classes.

L2 to L1 and L1 to L0 also support WFQ remapping. The remapping is indexed by upper level WFQ priority and lower level combined shaping status (green, yellow, red). See section L3/L2 and L2/L1 and L1/L0 Service Profiles below.

10.4.5.1.3 Shaper Update Effects

The scheduling event updates the associated shapers by deducting the packet byte size from the shaper's token bucket.

The shaper update effects associated with a service profile selects the mode of the shaper's update.

- Up to 4 shaping type tables can be defined to control shaper update

The Flow scheduling priority, which is remapped to four priorities, combined with the L3-Shaper status (green, yellow, red) indexes a 12-entry table that determines the Commit rate shaper and Excess/Peak rate shaper update. Thus enabling various shaper update schemes including (pre-color aware/blind) trTCM and MEF requirements.

(G,Y,R) x (WFQ0-WFQ3)	*COMMIT SHAPER UPDATE*	*EXCESS AND PEAK SHAPER UPDATE*
G, WFQ0	1	0
G, WFQ1	0	0
….	…	…
R, WFQ2		
R, WFQ3		

10.4.5.1.4 L3/L2 and L2/L1 and L1/L0 Service Profiles

In addition to the L4 service profiles, the NPS-400 also supports L3/L2 and L2/L1 and L1/L0 service profiles based on the priority propagation mechanism (described in section 9.8.2). In the Subport, Port and Interface levels there are both dual shapers and translation tables.

The priority of a higher level entity is remapped to a different priority at a lower-level entity per lower-level shaper color. The re-mapping table is a global configuration per level.

Example: Typical setting

Reduce priority for non-conforming CIR traffic and pause on PIR non-conforming except for High priority. Note that only priorities 0-3 are used, since 4 means "do not schedule".

Higher Level priority:	**0**	**1**	**2**	**3**
Lower level dual rate shaper status (color):	High WFQ Priority			Low WFQ Priority
Green (CIR and PIR conforming)	0	1	2	3
Yellow (Only PIR conforming)	0	1	3	3
Red (Non conforming)	0	4	4	4

Assigned priority:	**0**	**1**	**2**	**3**
Lower level dual rate shaper status (color):	High WFQ Priority			Low WFQ Priority
Green (CIR and PIR conforming)	Peak Commit	Peak Commit	Peak Commit	Peak
Yellow (Only PIR conforming)	Peak Commit	Peak Commit	Peak	Peak
Red (Non conforming)	Peak Commit	None	None	None

Example: Color aware trTCM setting:

Higher Level priority:	**0**	**1**	**2**	**3**
Lower level dual rate shaper status (color):	High WFQ Priority			Low WFQ Priority
Green (CIR and PIR conforming)	0	1	2	3
Yellow (Only PIR conforming)	1	1	3	3
Red (Non conforming)	2	4	4	4

The remapped priority is configured per shaper color to affect/not-affect the CIR and PIR shapers. This flexible mapping allows implementation of trTCM/srTCM color aware/blind per priority.

Color aware trTCM setting example:

Assigned priority:	**0**	**1**	**2**	**3**
Lower level dual rate shaper status (color):	High WFQ Priority			Low WFQ Priority
Green (CIR and PIR conforming)	Peak Commit	Peak Commit	Peak Commit	Peak Commit
Yellow (Only PIR conforming)	Peak	Peak	Peak	Peak
Red (Non conforming)	Peak	None	None	None

Example: Color blind srTCM setting example with both shapers token filled at same rate:

Higher Level priority:	**0**	**1**	**2**	**3**
Lower level dual rate shaper status (color):	High WFQ Priority			Low WFQ Priority
Green (CIR and EIR conforming)	0	0	0	0
Yellow (Only EIR conforming)	0	1	1	1
Red (Non conforming)	0	4	4	4

For srTCM, the CIR and EIR are both drained when CIR status is Conforming. When only EIR is conforming, only EIR is drained. When none of the shapers are conforming, then none are drained.

Color blind srTCM setting example:

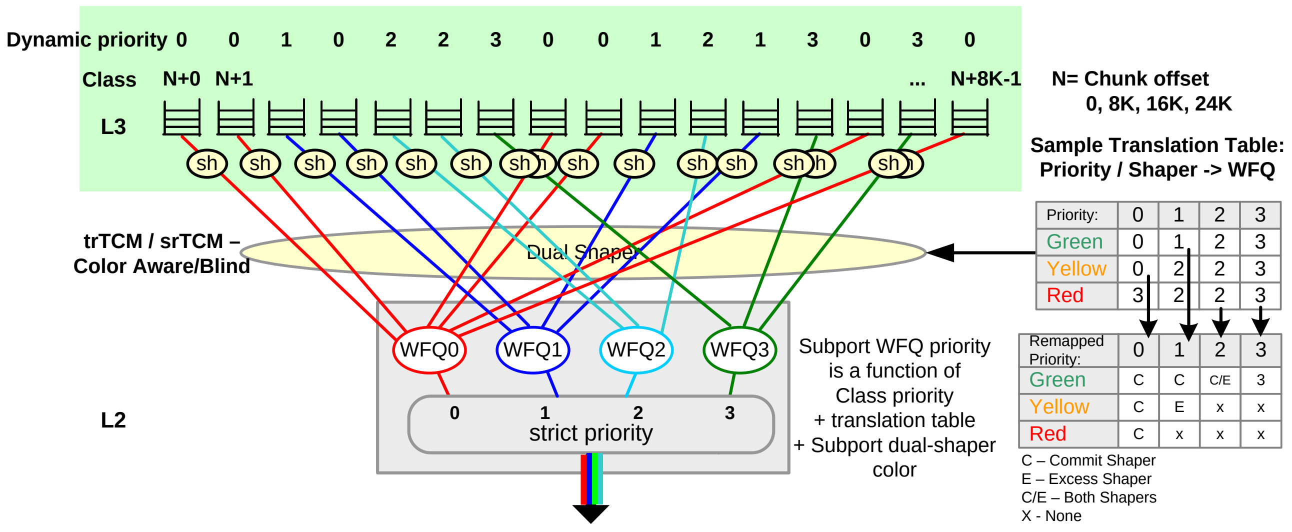
Assigned priority:	**0**	**1**	**2**	**3**
Lower level dual rate shaper status (color):	High WFQ Priority			Low WFQ Priority
Green (CIR and EIR conforming)	Peak Excess	Peak Excess	Peak Excess	Peak Excess
Yellow (Only EIR conforming)	Excess	Excess	Excess	Excess
Red (Non conforming)	Excess	None	None	None

Fast queue service profile TM-L1/TM-L0:

Higher Level priority:	**-1**	**0**	**1**	**2**	**3**
Lower level dual rate shaper status (color):	Fast Queue	High WFQ Priority			Low WFQ Priority
Green (CIR and PIR conforming)	0	0	1	2	3
Yellow (Only PIR conforming)	0	0	1	3	3
Red (Non conforming)	0	0	4	4	4

Example 7: The following figure exemplifies the subport/port queues and scheduling diagram. The dual shaper combined with the translation table is used to determine the priority.

Figure 9-13. L3/L2 service profiles – sample

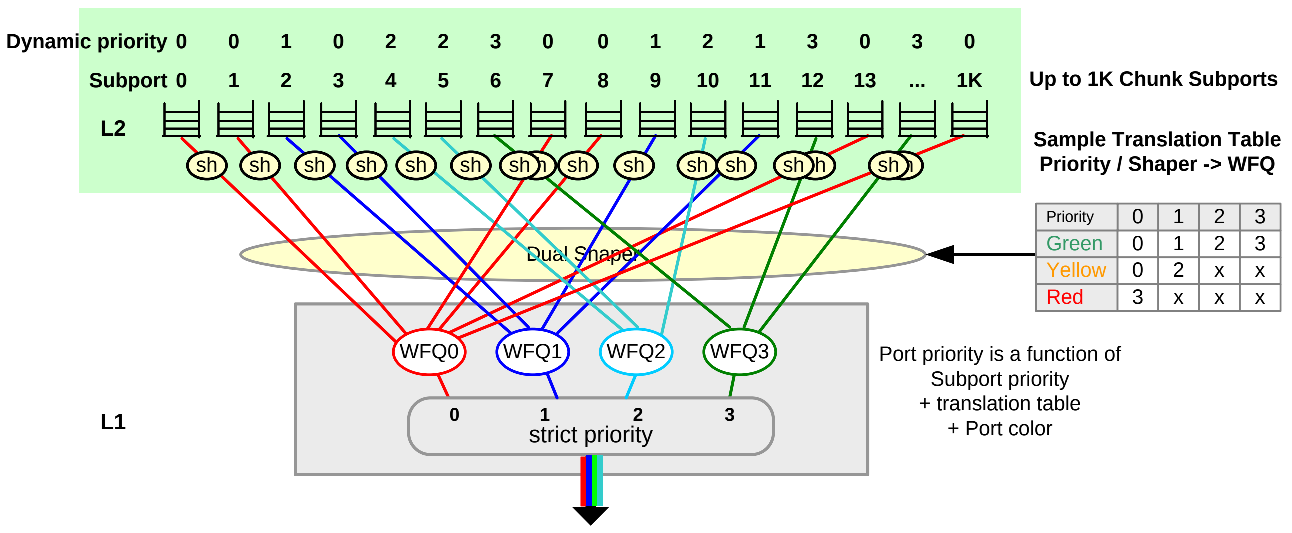

Dynamic priority 0 0 1 0 2 2 3 0 0 1 2 1 3 0 3 0
Class N+0 N+1 ... N+8K-1
L3
sh
trTCM / srTCM – Color Aware/Blind
Dual Shaper
WFQ0
WFQ1
WFQ2
WFQ3
0 1 2 3
strict priority
L2
Subport WFQ priority is a function of Class priority + translation table + Subport dual-shaper color
N= Chunk offset 0, 8K, 16K, 24K
Sample Translation Table: Priority / Shaper -> WFQ

Priority:	0	1	2	3
Green	0	1	2	3
Yellow	0	2	2	3
Red	3	2	2	3

Remapped Priority:	0	1	2	3
Green	C	C	C/E	3
Yellow	C	E	x	x
Red	C	x	x	x

C – Commit Shaper
E – Excess Shaper
C/E – Both Shapers
X - None

Figure 9-14. L1/L2 service profiles - sample


Dynamic priority 0 0 1 0 2 2 3 0 0 1 2 1 3 0 3 0
Subport 0 1 2 3 4 5 6 7 8 9 10 11 12 13 ... 1K
L2
sh
Dual Shaper
WFQ0
WFQ1
WFQ2
WFQ3
0 1 2 3
strict priority
L1
Port priority is a function of Subport priority + translation table + Port color
Up to 1K Chunk Subports
Sample Translation Table Priority / Shaper -> WFQ

Priority	0	1	2	3
Green	0	1	2	3
Yellow	0	2	x	x
Red	3	x	x	x

10.4.6 Queue Flushing

Queue flushing is supported by hardware at any level of the hierarchical scheduler. Queue flushing may be controlled by the SW.

The SW can configure the TM to flush one entity at time. This is mandatory for packets that are buffered by the TM.

In addition to the CTOPs’ software ability to discard packets, a WRED profile could be assigned that causes these packets as to be discarded by the TM. (This operation consumes DRAM bandwidth for writing the packet in the TM buffer.)

Queue flush process termination for individual TM flows may be indicated by a CTOP marked packet. This packet is the last queued in the flushed queue and is labeled as a non-droppable loopback packet, via Packet Switch. Since this packet is the last in the queue, once it is scheduled, it will be sent back to the CTOPs via the loopback interface indicating the completion of the flushing process.

10.5 TM Packet Switch

Each of two TM engines contains an internal Packet Switch which enables flexible and dynamic mapping of TM hierarchies into up to 8 physical underlying ports and interfaces.

Packets are scheduled from the TM queues based on the hierarchical scheduling algorithms and sent to the Packet Switch. Packets are switched to output interfaces based on the TM_INFO, which is set by SW per frame by the CTOPs, and the port remapping table.

The application must direct the packet to the appropriate TM engine. Each TM can send traffic via both interface sides.

The application sets per packet:

- The Packet Switch ID Offset – PSOFS[2:0]
- Flow ID – the TM topology remapped to TM-L1[8:0].
- CoS[1:0]

Packets are scheduled out of the Packet Switch on a per Class of Service basis. Class of Service is obtained from the internal switch header of the packet.

The internal 4K Packet Switch ID is indexed by {L1[8:0], PSOFS[2:0]}.

Each Packet Switch queues the packet descriptors in the internal Transmit Output Queue (part of the TxNDMA). The 4K PSIDs per TM are mapped to 128 regular + 16 special Output Queue IDs. Each output queue ID indexes 4 CoS queues. The CoS queue is selected per packet by the software set CoS[1:0] field.

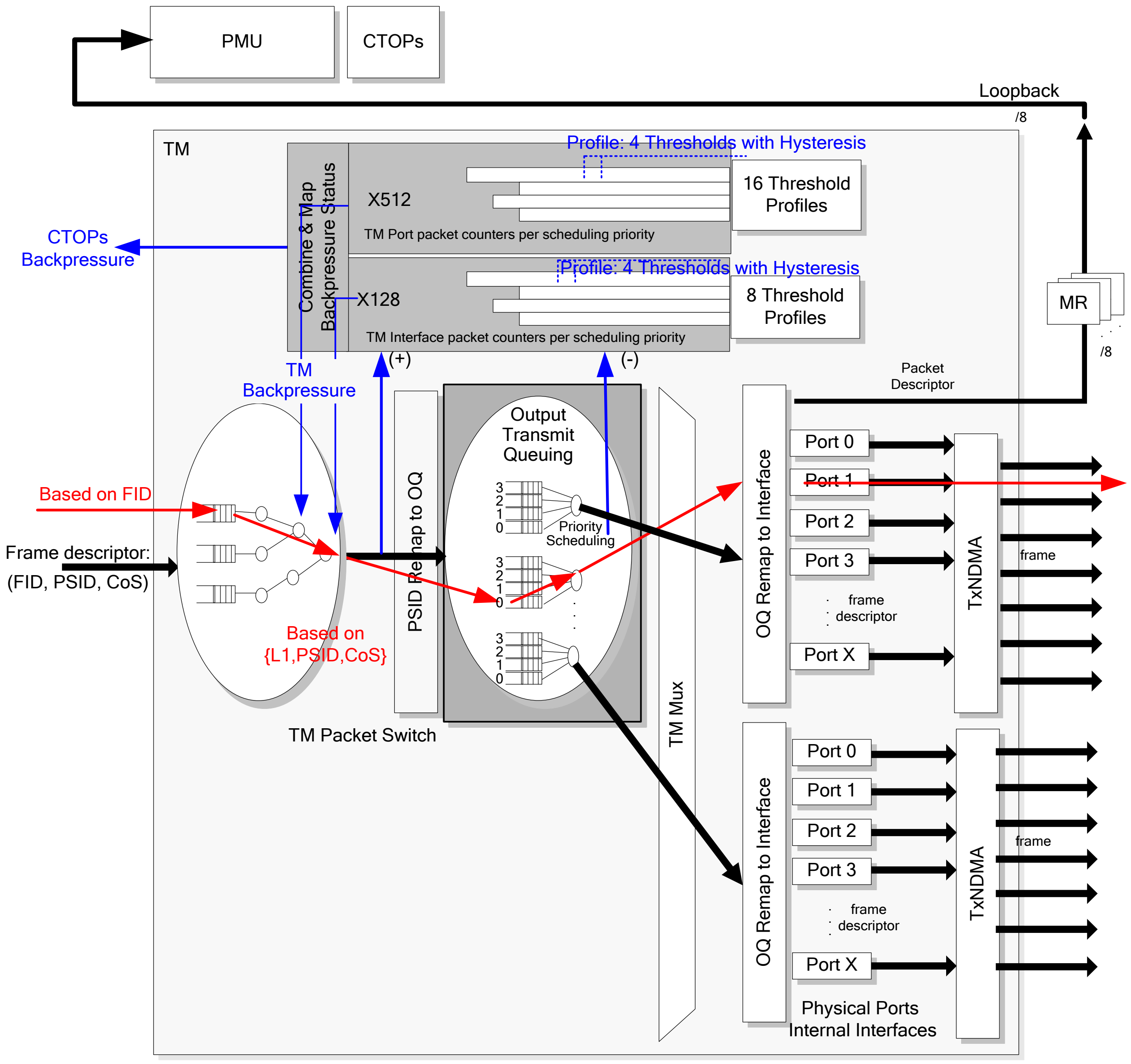
Each of the channels CoS-0 OQ can be configured as low-latency (high priority serviced) for the transmit DMA. The low-latency queues get expedited service by transmit DMA.

Each TM Output Queue can hold up to 8K packet descriptors for all queues.

Note that packet assigned CoS and derived OQ-CoS have no relation to TM scheduling priority.

For each Packet Switch, each of the 128+16 output queues is flexibly mapped to physical interfaces, interface channels and internal interfaces.

Figure 9-15. Packet Switch data flow



MR = Multicast Replicator

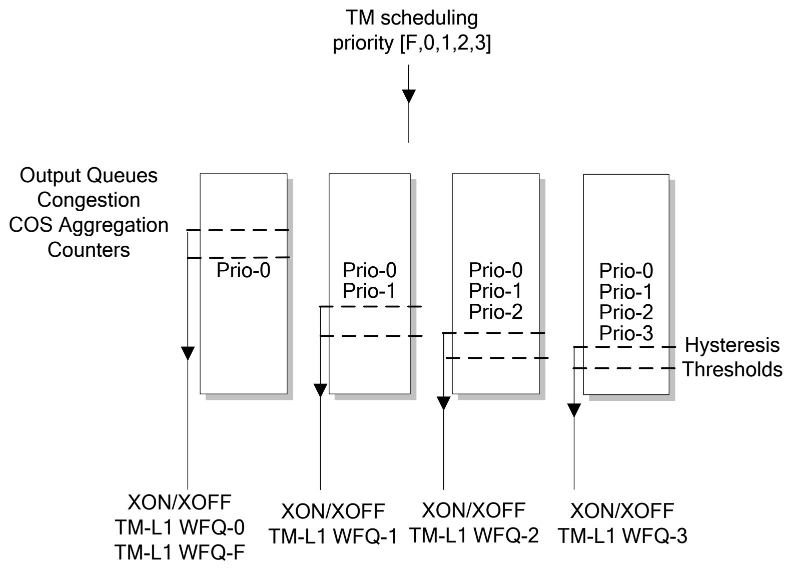
10.5.1 Transmit Output Queue Congestion State and TM Backpressure

The Output Transmit Queue's congestion is monitored by packet-based congestion accounting per TM Port/Interface hierarchy * CoS.

- There are 512x4 packet counters for each TM-0/1 Port entity (total 4K).
- There are 128x4 packet counters for each TM-0/1 Interface entity (total 1K).

The internal Packet Switch has lossless operation. Backpressure operates on a per TM Port/Interface entity per priority basis from the Packet Switch to the TM topology priority scheduler.

Backpressure exists per Interface and per Port on a priority basis. The backpressure is triggered by the Output Transmit Queue's congestion of queued traffic per specific TM L1 and L0 entities. The congestion is monitored via packet counting per scheduling WFQ priority.


TM scheduling priority [F,0,1,2,3]
Output Queues Congestion COS Aggregation Counters
Prio-0
Prio-0 Prio-1
Prio-0 Prio-1 Prio-2
Prio-0 Prio-1 Prio-2 Prio-3
Hysteresis Thresholds
XON/XOFF TM-L1 WFQ-0 TM-L1 WFQ-F
XON/XOFF TM-L1 WFQ-1
XON/XOFF TM-L1 WFQ-2
XON/XOFF TM-L1 WFQ-3

Per TM Port/Interface entity * Priority, an hysteresis threshold is configured.

1. The Prio-0 congestion counter monitors OQ packet congestion for WFQ-0 scheduled packets and backpressure TM-L1 (L0) scheduler priorities WFQ-0 and WFQ-Fast.
2. The Prio-1 congestion counter monitors OQ packet congestion for WFQ-0 and WFQ-1 scheduled packets and backpressure TM-L1 (L0) scheduler priority WFQ-1.
3. The Prio-2 congestion counter monitors OQ packet congestion for WFQ-0 1,2 scheduled packets and backpressure TM-L1 (L0) scheduler priority WFQ-2.
4. The Prio-3 congestion counter monitors OQ packet congestion for WFQ- 0,1,2,3 scheduled packets and backpressure TM-L1 (L0) scheduler priority WFQ-3.

The thresholds are configurable by 16 TM-Port (L1) level profiles and 8 TM-Interface (L0) level profiles.

The Packet Switch schedules packets from the Output Transmit Queue on a strict 4-level priority arbitration.

CoS and priority guidelines:

The OQ to TM scheduler backpressure may result in head of line blocking.

It is recommended to assign a unique PSID (OQ) per TM L0 or L1 entity.

The Packet COS assignment guideline:

Packet COS = TM scheduling priority when all shapers are conforming (green).

Else the L4 shallow mechanism and shallow indication per priority are sub-optimal.

10.5.2 Physical Interface Remapping

The Packet Switch of each TM supports remapping of the PSID to 128+16 OQs and flexibly mapped physical interfaces, loopback interface and interface channels. Each channel has four priority queues and can be assigned to either the closely attached interface or to the distant interface attached to the other TM. The PSID to OQ mapping table can be switched between active and backup versions. This feature can be used for fast switching between active and backup ports in case of failure. The switching is lossless, enabling TM backlogged traffic to be switched directly to the new active interface, even though at the time of processing and queuing the active port was different.

The OQ are assigned per PSID:

- TM hierarchy coupled mode:
  - {PSID[11:0]=TM-L1[8:0], PSOFS[2:0]}. PSID is assigned an OQ in the range [0-127, 128-143].
  - SW can switch usage from active remapping table to backup remapping table by single write action.

The TxNDMA OQ is mapped to a physical interface. The interface is either a transmit port (e.g. 10GbE, 40GbE, 100GbE), Interlaken interface, or internal service interface.

TM multiplexer capability assigns each OQ to either TxNDMA-0 or TxNDMA-1.

Each TxNDMA maintains an OQ mapped 1:1 to PSID-OQ.

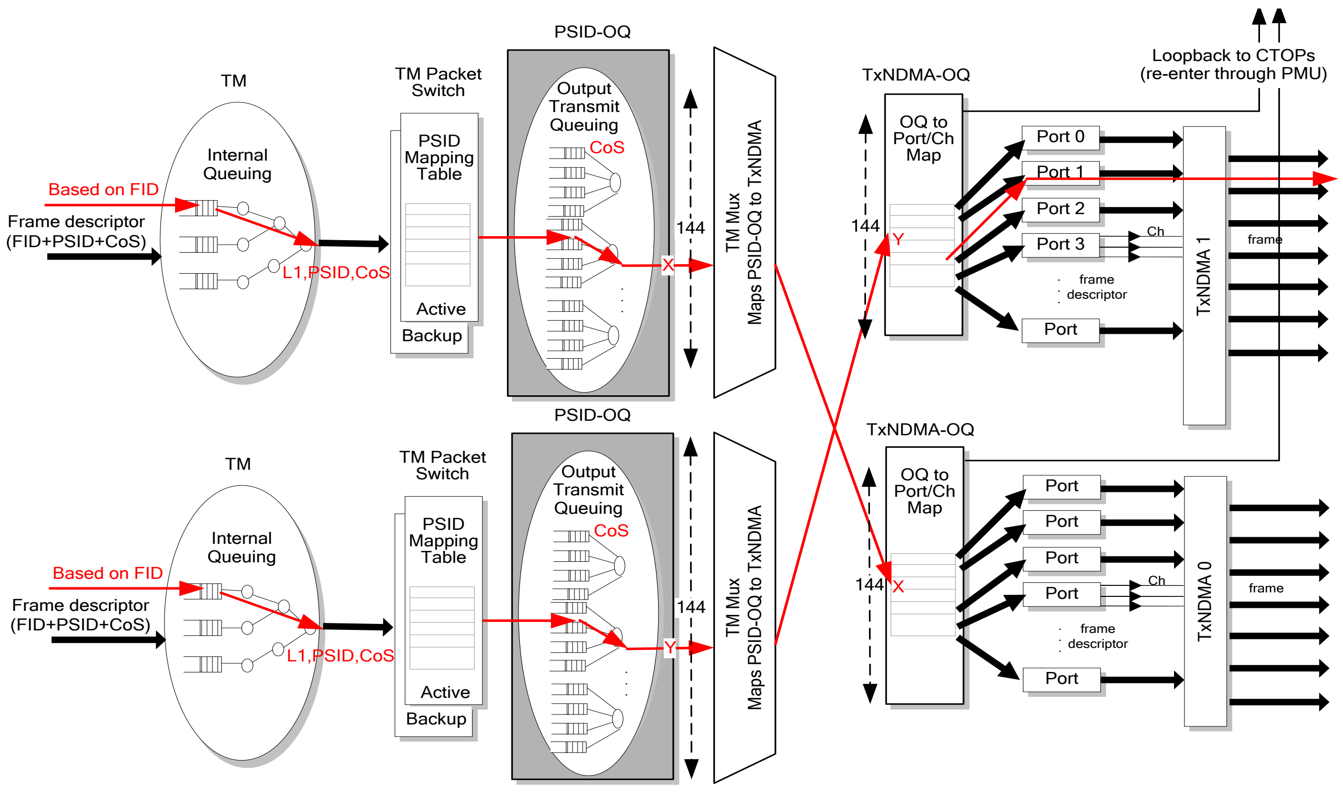
The TxNDMA OQ is mapped to a physical interface.

The TxNDMA OQ mapping to physical interface is controlled by 80 range configurations. The OQ ranges are mutually exclusive.

Several queues can service same physical interface.

When a range of TxNDMA OQs are assigned to an Interlaken interface, each OQ is assigned to a separate Interlaken channel that is identified by the Interlaken Control Word transmitted.

Figure 9-16. PSID remapping



10.5.2.1 Interface Mapping Table

The TM Packet Switch includes 128+16 output queues. Three levels of mapping are provided:

1. The first level maps between the PSID[11:0] to PSID-OQs, which is done by the PSID mapping table.
2. The second level, serving as TM mux, assigns PSID-OQs to TxNDMA that serves an interface side. The PSID-OQ is mapped 1:1 to TxNDMA-OQ.
3. The third level aggregates ranges of TxNDMA-OQs to physical interfaces, internal interfaces and channels.

For each side (west/east) the physical interface is identified by a 9-bit configuration ID:

- Type (3 bits), Interface 12 SerDes section (2 bits), Port number offset (4 bits)
- Physical port type (3 bits):
  - 010 - 10GbE ports, XFI, KR
  - 100 - 40GbE ports, XLUI, XLPPI
  - 110 - 100GbE ports, CAUI, CLPPI
  - 001 - Interlaken
- Interface 12 SerDes section (2 bits):
  - 00 - lanes 11:0
  - 01 - lanes 23:12
  - 10 - lanes 35:24
  - 11 - lanes 47:36
- Port number offset (4 bits):
  - Port number = Section + Port number offset

Table 10-5. Interface ID table

INTERFACE ID (9B)	*SECTION*	*SERDES LANE(S)*	*NPS-400 INTERFACE / TYPE*
80-8B	0	[11:0]	XFI-0...XFI-11
90-9B	1	[23:12]	XFI-12...XFI-23
A0-AB	2	[35:24]	XFI-24...XFI-35
B0-BB	3	[47:36]	XFI-36...XFI-47
100,103,106	0	[3:0],[7:4],[11:8]	XLAUI-0...XLAUI-2
110,113,116	1	[15:12],[19:16],[23:20]	XLAUI-3...XLAUI-5
120,123,126	2	[27:24],[31:28],[35:32]	XLAUI-6...XLAUI-8
130,133,136	3	[39:36],[43:40],[47:44]	XLAUI-7...XLAUI-11
180	0	[9:0]	CAUI-0
190	1	[19:10]	CAUI-1
1A0	2	[29:20]	CAUI-2
1B0	3	[39:30]	CAUI-3
50	0	[47:0]	Interlaken-0
40	-	49	External host CPU

10.5.3 TM Loopback to the PMU/CTOPs

The TM can loopback the scheduled packet descriptor to the PMU as unicast or replicate the looped back packet descriptor.

Each TM supports eight Packet Switch internal channels for the TM loopback interfaces to the PMU/ CTOPs. Each channel has a separate Multicast Replicator (MR). The replication is performed on the packet descriptor, while sharing the packet data buffers by all replicas.

The unicast looped back packet can specify any PMU queue (PMU-0 0-127, PMU-1 0-127) as the loopback target. The multicast looped back packet has a preconfigured target PMU queue according to the PSID mapped loopback channel.

Dedicated backpressure is sent from the PMU back to the loopback channels to eliminate blocking and to control the processing resources allocated for loopback and multicasting. The PMU backpressure is according to job budgets.

The Loopback channel is flow controlled by PMU queues and queue grouping:

- Each MR is flow controlled by the assigned PMU queue.
- The Loopback channel is flow controlled by the assigned group of PMU queues.

Since packets can switch, via the Packet Switch, to the loopback interface from any queue, there is no need for auxiliary loopback TM queues. Each TM queue can serve unicast traffic as well as multicast packets.

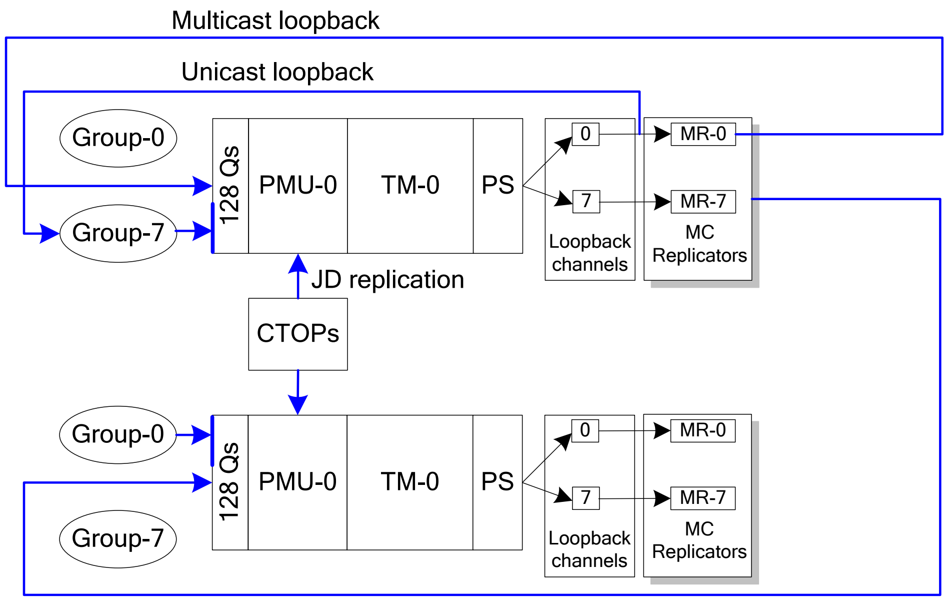
The CTOP software sets the required number of replicas (0-[64K-1]) in the packet TM_INFO via the PMU Job Descriptor.

To guard from head of the line blocking of unicast traffic by Multicast-Replicator-destined packets, it is advantageous to allocate a dedicated TM L1 or TM L0 entity for each Multicast Replicator. This ensures that backpressure from the Multicast Replicator will affect only the designated TM hierarchical entities.

The multicast replications can be executed by two separate mechanisms:

- TM scheduled packet descriptor multicast replication (by TxNDMA)
- Lightweight Multicast replication by CTOP translation of a single Job Descriptor (JD) into multiple Job Descriptors (JDs)

The mechanism is selected by the application per packet.



TM Multicast Replication

Software prepares a unicast frame destined to the multicast port, sets the replica number. The TM directs the multicast frame to the Multicast Replicator which pushes N+1 jobs to the PMU to the selected PMU queue.

The software in the CTOPS gets each frame, duplicates its header BD and LBD (in each of multi-buffer frame) and sends the frame to the Tx port through the selected TM flow.

The Multicast Control bit (MC) marks the frame as a multicast frame and controls the buffer release mechanism. Each frame buffer has an associated MC counter, which is read and decremented atomically by the buffer release mechanism. When the MC reaches zero, it is not decremented and the release logic recycles the buffer. When all LBD buffers are recycled then the Link Buffer Descriptor (LBD) is also recycled.

Lightweight Multicast Replication

In the lightweight multicast replication, the CTOP software translates a single frame JD to a container of multiple JDs (all the job IDs are allocated by the PMU) and can forward the container to transmission by the TM, where it gets broken into multiple frames. The lightweight multicast flow keeps the internal frame order of the job container relative to other ordered JDs in the PMU queues.

Replication Mechanism Selection

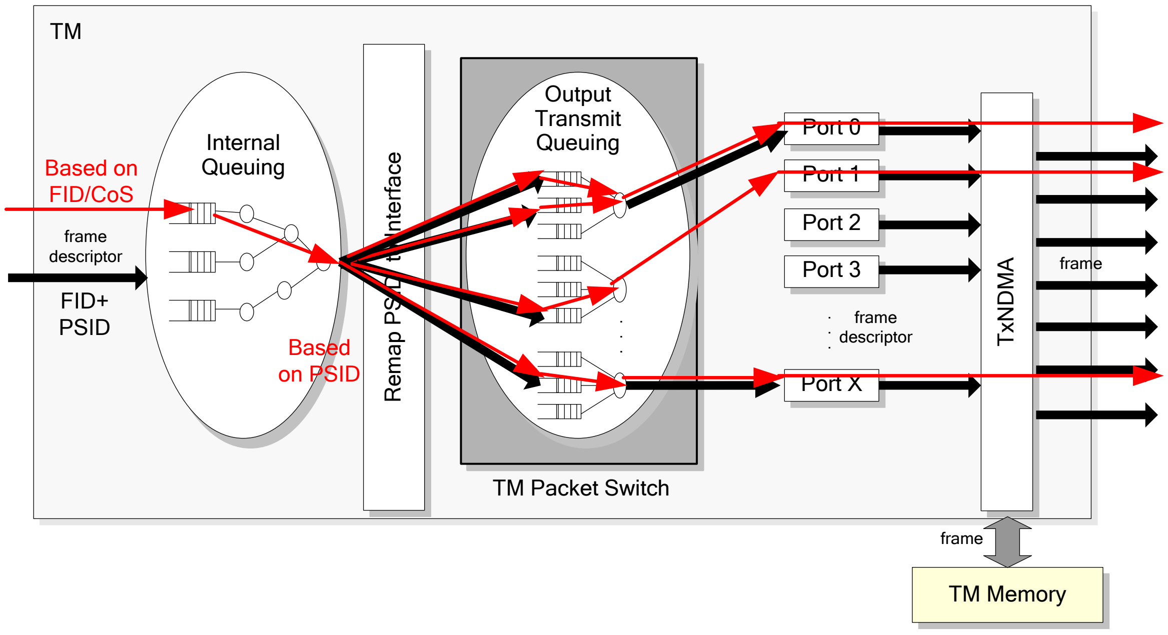
When a packet’s PSID is assigns it to the loopback special interface, the application can control the mechanism for multicast replication. For loopback packets, set the MC=1 and REP=1.

PSOFS[2:0], concatenated to the L1 entity, selects another packet switch table entry that maps it to a selected loopback interface. Up to eight loopback interfaces are selected per L1 entity.

10.5.4 LAG Shaping

The following figure exemplifies data flow for shaping a group of ports together for link aggregation purposes. All traffic belonging to a LAG group is placed in the same TM entity and a group of queues which are shaped together. By assigning different PSOFS[2:0] and thus PSID that is mapped to different ports when packets are scheduled from the TM, they are distributed to the group of ports belonging to the LAG. This allows shaping for up to 8 LAG members of the same device.

Figure 9-17. LAG shaping



10.6 WRED: Congestion Avoidance & Traffic Conditioning

The WRED (Weighted Random Early Detection) mechanism is used in the NPS-400 in order to perform traffic policing and congestion avoidance. WRED can make decisions whether to drop or forward a frame. By making random early packet discard decisions before a congestion level is reached, congestion may be avoided. There is a dedicated WRED for each of the TM engines.

10.6.1 WRED Features

- 6-level hierarchical dropping:
  - Global per TM level, Interface (TM level 0), Port (TM level 1), Subport (TM level 2), Class (TM level 3), Flow (TM level 4)
  - 32K/4K WRED functions derived from:
    - 8 colors per packet (packet TM_INFO COS[2:0])
    - 16 WRED profile behavior templates per level; each with 8 behavioral functions (selectable per color)
    - 256 absolute values per level in levels 2-4 or 32 absolute values in the lower levels
- Configurable policing/dropping schemes between the 6 WRED hierarchy levels
- Per TM level configurable congestion accounting: frame-buffers / 64B units / 16B units or frames/ events. Used for WRED and TM congestion monitoring for flow control.
- Each TM level supports separated configuration for congestion accounting. The TM Global congestion should be for TM-frame-buffers in order to guard from TM memory excess consumption.
- WRED profiles allow buffer reservation for guaranteed memory resources per entity
- The Fast queues have an approximated WRED decision to expedite the queuing process

10.6.2 Hierarchical Dropping

The WRED decisions are based on three elements:

- **Congestion Level** – The amount of resources occupied (e.g. queue fullness). The congestion level is measured in units of buffers (e.g. 256 bytes (frame buffer size) or 64B or 16B) or frames (e.g. events).
- **Color** – Color is the result of the classification and metering process passed from the CTOPs. The color indicates the conformance of the frame to the expected traffic parameters as well as the COS. The priority used in scheduling is separate and can be modified based on entity shaper color, not the color of the packet.
- **Profile Behavior Template** – A set of relative parameters associating each color with a drop function template, where each drop function describes the congestion levels in percentage according to which drop decisions should be taken. Drop points in the profile behavior template are in percentage to the maximum buffer size. Each profile behavior template represents a different COS enabling different drop patterns for each color and each COS.
- **Scaling Factor** – Absolute memory resource value representing queue occupancy.
- **WRED Function** – The WRED function results from a multiplication of the profile behavior template with the scaling factor.

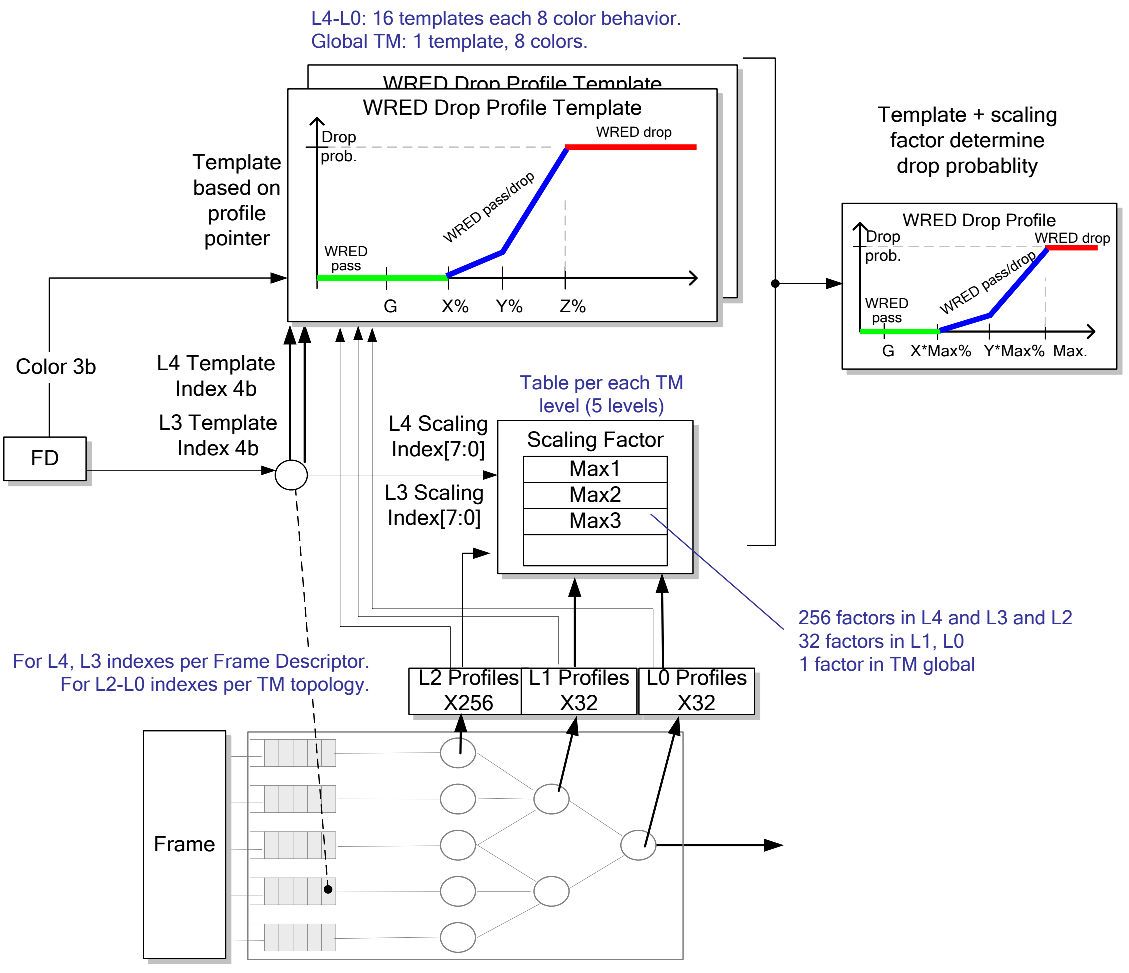
When a frame descriptor (FD) is forwarded to the TM, it is assigned with a Flow ID (FID), and a color (one of eight). The colors for WRED are generated using the most common metering methods (e.g. srTCM, trTCM) of CTOPs token bucket policers.

Each *FD* is associated with an 8-bit scaling index and 4-bit template index for the TM Flow level, and an additional 8-bit scaling index and 4-bit template index for the Class level. Each TM *entity* in the lower levels (Subport, Port and Interface) is configured with a pair of similar WRED indices (absolute table and template table) which points to profile tables. The per FD scaling and template allows CoS differentiation for flows sharing the same L4 or L3 entities.

According to the frame color and the index pairs, a WRED behavior function is chosen, combined with the scaling factor generates the WRED function. (See Figure 9-18. WRED with scaling parameters.)

Based on the WRED function and the congestion level in each entity, a WRED decision is made based on the accumulative results in all levels and assigned WRED mode (aggressive, relaxed, etc). The WRED decision selects either to drop the frame or forward the frame to the queue based on its FID.

Figure 9-18. WRED with scaling parameters


L4-L0: 16 templates each 8 color behavior.
Global TM: 1 template, 8 colors.
WRED Drop Profile Template
WRED Drop Profile Template
Drop prob.
WRED drop
WRED pass/drop
WRED pass
G
X%
Y%
Z%
Template based on profile pointer
Template + scaling factor determine drop probablity
WRED Drop Profile
Drop prob.
WRED drop
WRED pass/drop
WRED pass
G
X*Max%
Y*Max%
Max.
Color 3b
L4 Template Index 4b
L3 Template Index 4b
FD
Table per each TM level (5 levels)
L4 Scaling Index[7:0]
L3 Scaling Index[7:0]
Scaling Factor
Max1
Max2
Max3
256 factors in L4 and L3 and L2
32 factors in L1, L0
1 factor in TM global
For L4, L3 indexes per Frame Descriptor.
For L2-L0 indexes per TM topology.
L2 Profiles X256
L1 Profiles X32
L0 Profiles X32
Frame

10.6.3 WRED Drop Function

Figure 9-19 shows a WRED drop function with the five parameters of the WRED drop function marked. The parameters of the drop functions are:

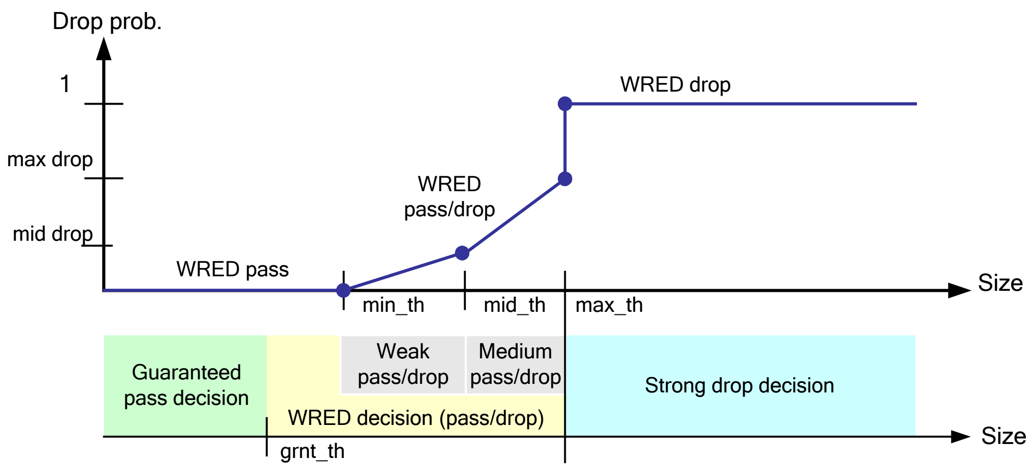
- grnt_th (guarantee threshold, common to all 8 color profiles),
- min_th (minimum congestion level),
- mid_th (medium congestion level),
- max_th (maximum congestion level) and
- max_drop (the drop probability at the maximum congestion level).

Usually, red colored frames will be dropped, while green colored frames have a better chance of being forwarded under congestion than yellow frames.

Each hierarchy calculates its local WRED decision according to following graph (Figure 9-19); since the final decision takes into account all local decisions at all levels, the local decisions are only *markers*, or recommendations:

- Under grnt_th, the local decision is marked as guarantee-pass (GP)
- Under min_th, the local decision is marked as strong-pass (SP)
- Between min_th and mid_th, the drop probability increases as the queue size increases; the local decision is marked as either weak-pass (WP) or weak-drop (WD)
- Between mid_th and max_th, the drop probability increases as the queue size increases; the local decision is marked as either medium-pass (MP) or medium-drop (MD)
- Above the max_th, the local decision is marked as strong-drop (SD).

Figure 9-19. WRED drop profile illustration

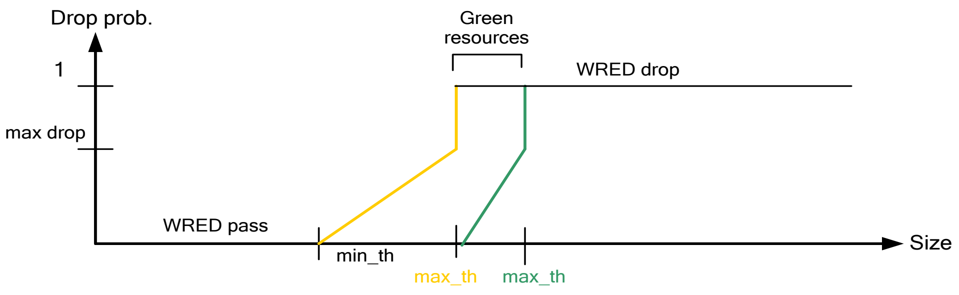


The graph is based on a combination of both the 16 WRED profile behavior templates as well as 256 scaling values which offers a wider range of WRED absolute profiles. The general pre-configured absolute values together with the relative values enable the graph to be shared by queues with different occupancy capabilities.

Since each color has its own configurable strong-drop threshold (max_th), resources may be better utilized for frame of different colors.

Each TM entity may be configured with a different grnt_th value representing its own reserved resources; assuming no oversubscription of guaranteed resources is made, the queue's resources are reserved regardless of the other queues occupancy in any hierarchical level.

Figure 9-20. Dropping example - green frames utilize extra resources not available to yellow frames



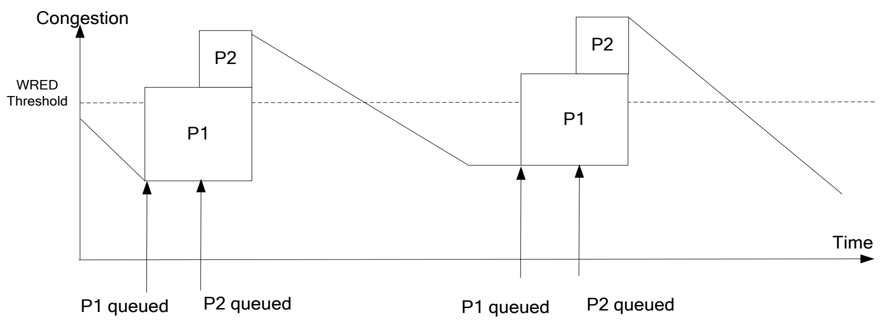
In addition, the same dropping behavior with a different strong-drop threshold (max_th) can be used by all colors. The figure above shows an example where frames that are green will always be able to utilize extra resources not available to yellow frames.

Each WRED mechanism operates on a different resource. In this way, congestion can be measured and accounted for in different levels. For example, one WRED configuration can measure the congestion of the specific flow that the frames belong to, whereas another WRED configuration simultaneously measures the global memory congestion level.

10.6.4 Randomized WRED Thresholds

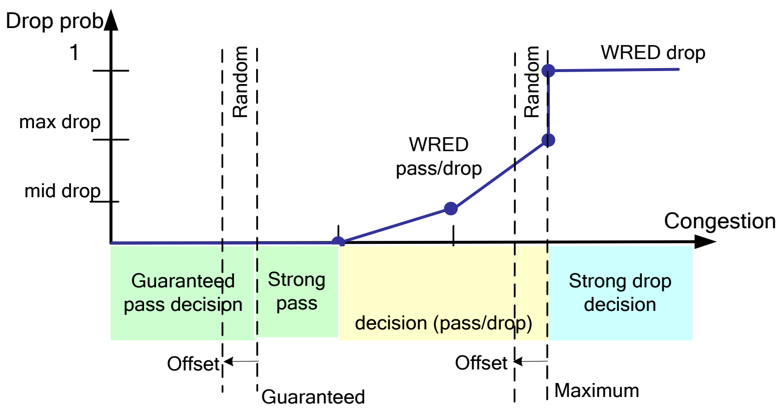
The guaranteed and maximal WRED thresholds can be randomized in a configured threshold range. The threshold randomization is helpful when the traffic flow has a specific synchronized pattern that causes repeated unfair decisions one some flow(s).

Example of two synchronized flows where packets repeatedly observe same WRED decision:



The packet of flow P1 always arrives when congestion is below threshold while the packet following P1 always arrives when congestion is above threshold.

The randomization configuration specifies a range offset from threshold that within this range the decision is taken 50% as if congestion is above threshold and 50% as if the congestion is below threshold:



10.6.5 Policing/Dropping Schemes between the 6 Levels

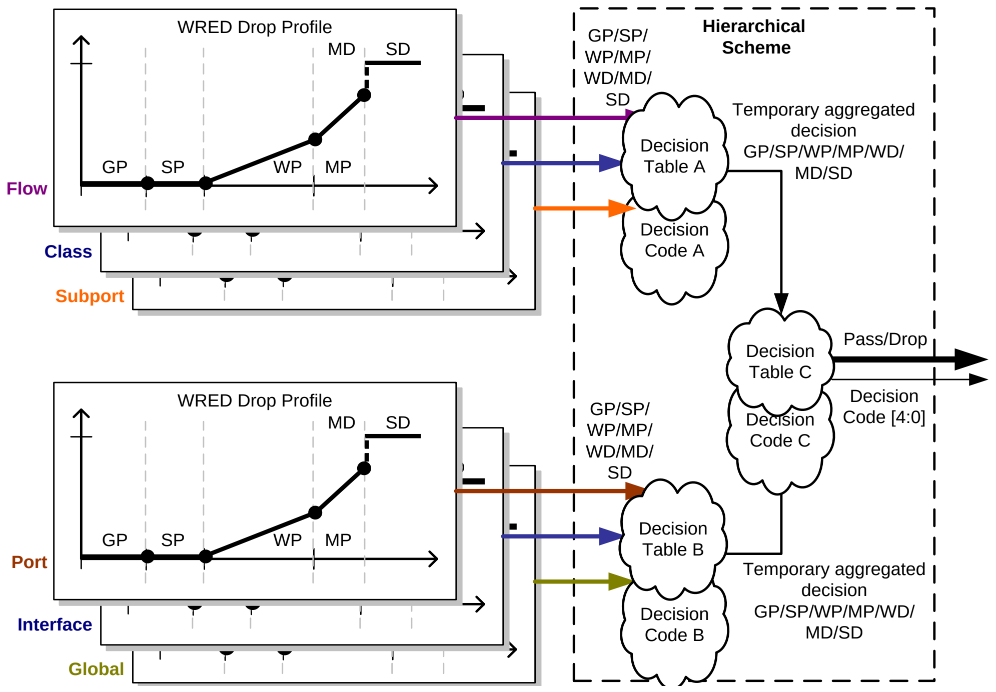
All six WRED functions are activated simultaneously on each frame. The final drop decision is based on all the following parameters:

- The local WRED results of each level (the decision labels: GP, SP, WP, WD MP, MD and SD).
- The configured scheme's settings (gives full flexibility for translating any combination of the 6 local decisions into one final pass/drop decision).

The pass/drop decision also assigns a 5-bit code that is used for pass/drop event statistics accounting. (Refer to the *Statistics Management* chapter.)

The following figure shows the decision-making process; the three tables are all part of a single hierarchical scheme (one of four available):

Figure 9-21. Policing/dropping schemes between the 6 levels



The TM hierarchy WRED results are flexibly combined for a unified pass/drop resolution.

The TM L4/L3/L2 WRED results Guarantee-Pass, Strong-Pass/Drop, Medium-Pass/Drop, Weak-Pass/Drop are used to index a configurable decision table (Table A in figure above). The 3 level * 3b results form 9 bits that index the table to retrieve a 4b L4-L2 decision.

The TM L1/L0/Global WRED results Guarantee-Pass, Strong-Pass/Drop, Medium-Pass/Drop, Weak-Pass/Drop are used to index a configurable decision table (Table B in figure above). The 3 level * 3b results form 9 bits that index the table to retrieve a 4b L1-L0-Gloabl decision.

The two 4b decisions index decision table C to resolve the hierarchical combined pass/drop decision.

The flexible configuration allows elaborated schemes of hierarchical WRED.

The following schemes exemplify common hierarchical configuration modes:

- Aggressive mode to Guarantee pass and Strong drop:
  - If any level decides on guaranteed pass then pass.
  - Else if any level decided on drop, the packet is dropped.
  - Else pass.

- Conservative mode of priority resolution:
  - Guarantee pass > Strong Drop > Strong Pass> Medium drop> Weak pass> Weak drop.
- Relaxed mode of priority resolution:
  - Guarantee pass > Strong Drop > Medium/Weak pass > Medium/Weak drop
- Pseudo guarantee mode with TM level precedence L4 > L3 > L2 > L1 > L0:
  - Guarantee or strong pass of higher level results in pass.
  - Else drop by higher level results as drop.
  - Else drop results in drop.

The following table shows the WRED resolution for various typical modes.

Table 10-6. WRED resolution for various typical modes

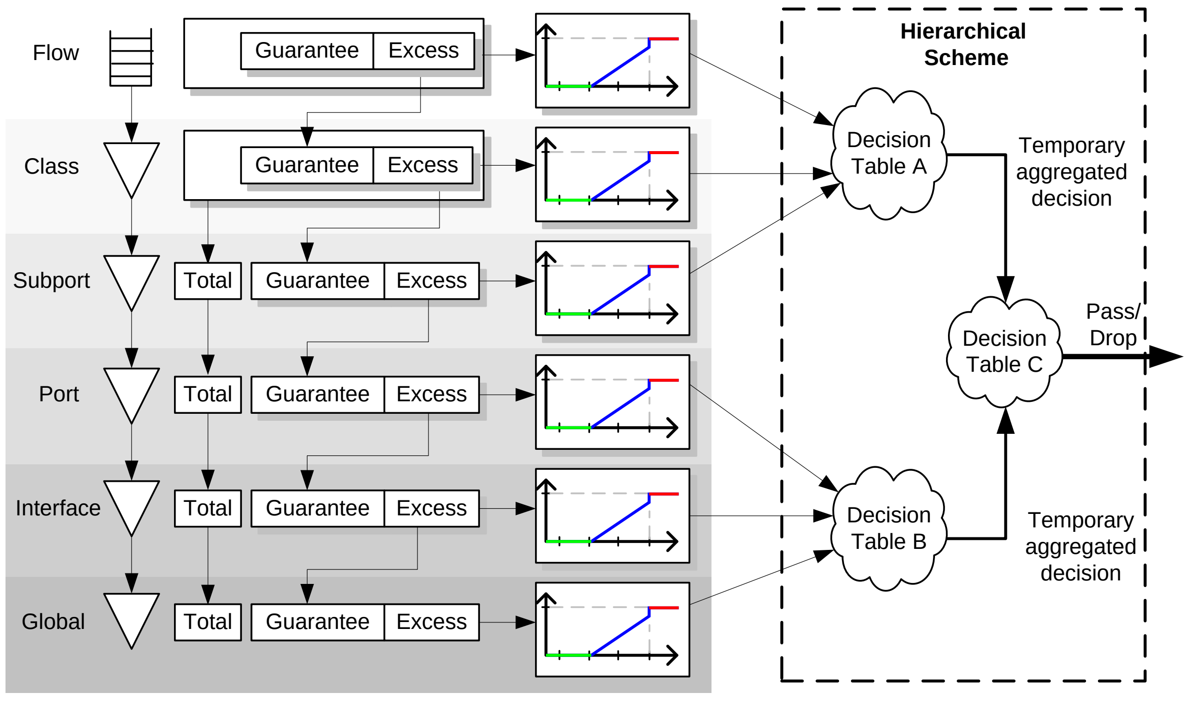
	Gur-Th	Min Th		Mid Th		Max Th		
MODE	***GUR***	***SP***	***WP***	***WD***	***MP***	***MD***	***SD***	***RESOLVED***
Aggressive	Any	X	X	X	X	X	X	Pass
	None	X	X	X	X	X	Any	Drop
	None	X	X	X	X	Any	None	Drop
	None	X	X	Any	X	None	None	Drop
	None	Any	Any	None	Any	None	None	Pass
Conservative	Any	X	X	X	X	X	X	Pass
	None	X	X	X	X	X	Any	Drop
	None	Any	X	X	X	X	None	Pass
	None	None	X	X	X	Any	None	Drop
	None	None	Any	X	X	None	None	Pass
	None	None	None	Any	X	None	None	Drop
	None	None	None	None	All/None	None	None	Pass
Relaxed	Any	X	X	X	X	X	X	Pass
	None	X	X	X	X	X	Any	Drop
	None	Any	X	X	X	X	None	Pass
	None	None	Any	X	X	X	None	Pass
	None	None	None	X	Any	X	None	Pass
	None	None	None	Any	None	Any	None	Drop
PS-GUR	L4		X					Pass
	No L4							
	L3		L4	No L4	L4	No L4		Pass
	None L3/L4							
	L2		X	None L3/L4	X	None L3/L4		Pass
	None L2/L3/L4							
	L1		X	None L2/L3/L4	X	None L2/L3/L4		Pass
	None L1/L2/L3/L4							
	L0		X	None L1/L2/L3/L4	X	None L1/L2/L3/L4		Pass
	X			Any	X	Any		Drop
	X			None	X	None		Pass

10.6.6 Memory Resource Guarantee

Resources can be guaranteed at any level explicitly by accounting guaranteed and excess buffers on separate dedicated counters. A guaranteed threshold point accompanies each template enabling incoming traffic to utilize either guaranteed resources or excess resources.

The local WRED decision is based on the Excess resource accounting. Software, however, may read both counters. Note that at the highest level (Flow), only one counter (total resources) is required. The **Class level has only one counter.**

Figure 9-22. Memory resource guarantee



Memory resource reservation can be made at any level or entity such that for each entity the guaranteed memory may be used only if the higher level entities have exhausted their own guaranteed resources. For example, if a TM Flow is allocated with 10 guaranteed buffers and a Class is allocated with 100 guaranteed buffers, when a frame arrives to the Flow as long as the Flow consumes fewer than 10 buffers the Flow will consume its guaranteed resources and the Class will not consume any resources. If the Flow utilizes more than 10 buffers, it will account the frame as excess memory and will pass the decision to the Class. At this point, if the total memory consumed by the Class is fewer than 100 buffers, it will guarantee the queuing of the frame. If it is greater than 100 buffers, it will be subject to the excess memory allocation based on the WRED function.

In order to prevent improper usage of the total resources, a dedicated threshold is also supplied so a local guarantee decision can be reduced to a weaker pass decision. This way the system is never over-flooded.

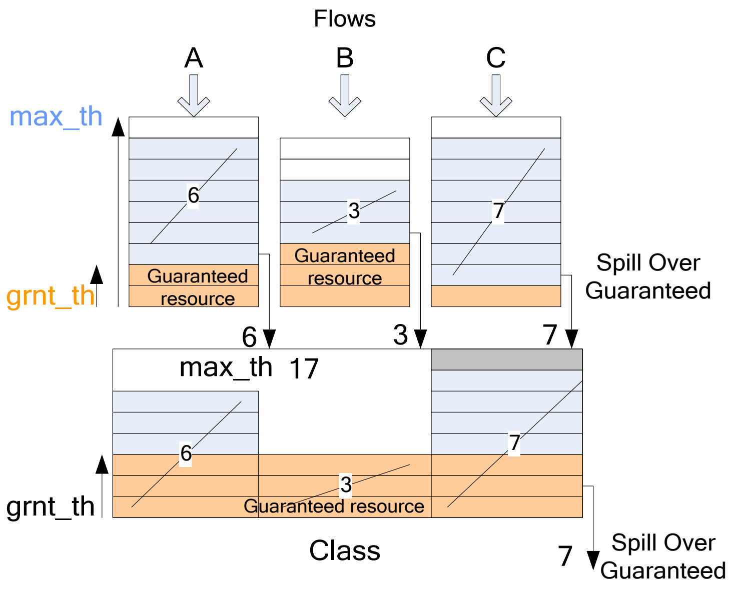
Resource guarantee and limit example:

TM Flow A is set with 2 guaranteed buffers and a maximum of 8 buffers.
TM Flow B is set with 3 guaranteed buffers and a maximum of 6 buffers.
TM Flow C is set with 1 guaranteed buffer and a maximum of 8 buffers.

Flow buffer consumption above the guaranteed threshold spills over to the Class. For example in Flow A, 6 buffers over the guaranteed threshold of 2 are counted also for the Class level.

Although the sum of all maximal Flow resources is 22 buffers, the Class' maximum setting of 17 buffers will limit the buffer consumption. The Class is set with a guaranteed resource of 9 buffers.

Figure 9-23. Resource guarantee and limit example



10.6.7 WRED of Fast Queues

Up to 128 TM L4 Fast queues can be directly mapped to 32 L1 logical ports (4 queues per logical port).

These queues are associated with queue descriptor management and frame descriptor management in internal memory.

The Fast queue's WRED decision is expedited by an approximated WRED.

There are two WRED decision modes supported. The mode is per Fast queue selected by assigning one of eight profiles. The profile configuration selects the mode of operation per WRED color (TM_INFO COS[2:0]). Thus the combination of Fast queues and color can be controlled.

1. WRED approximated decision is based on last available accurate WRED pass/drop decision.
2. WRED approximated decision is based on queue fullness.

Fast flows with a high rate can utilize last available decision mode without risking the decision being stale. The last decision mode should be applied to TM flows that utilize a single color, otherwise different colors follow the same decision.

For low rate flows and flows utilizing more than a single color, the queue fullness approximated WRED is more suitable.

The WRED pass decision may consume Flow guaranteed or non-guaranteed resources. As a Fast flow directly attached to an L1 entity, the spill over-accounting of non-guaranteed (excess) resources affects only TM hierarchies L1/L0 (skipping the L3, L2).

In both modes, the Guarantee or overflow decision is based on the last available (previous) status.

The WRED statistic code (5 bits):
{3bits = color, Stat_ID[0] , 1 bit decision (pass/drop)}

There is a configuration option to select statistic code according to fast flows configuration or according to the last WRED statistics decision after remapping (see TM Statistics).

10.6.7.1 WRED Approximation based on Queue Fullness

Each of the 128 fast queues have a dedicated WRED fullness counter of up to 4GB similar to a regular queue. The fast queue fullness is maintained in dedicated cache entries to bypass the DRAM latency.

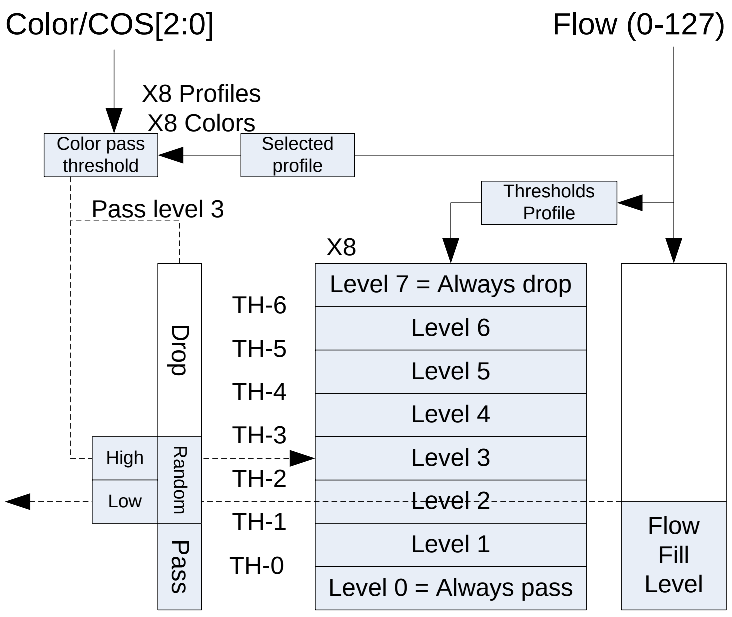
Like a regular TM flow queue, the fullness can be measured in resolutions of 16B, 64B, 256B, or packet.

The flow fullness is compared to 7 configurable thresholds. The thresholds are configured by 8 profiles.

Each fast queue is assigned a profile.

The flow is also assigned a profile to select per color Pass Level. There are 8 profiles each for 8 COS colors.

Figure 9-24. WRED approximation



- If fullness level is below TH-0, the decision is always pass.
- If fullness level is above TH-6, the decision is always drop.
- If fullness level is above the Pass Level, the decision is drop.
- If fullness level is equal to the Pass Level, the decision is random according to PROB_HIGH probability.
- If fullness level is equal to the level below the Pass Level, the decision is random according to PROB_LOW probability.

The random 6b number is compared to PROB_HIGH/LOW, and if the random value is greater than PROB_HIGH/LOW then the decision is pass.

10.6.8 Combined WRED of Dual TM Pass

The system architecture is often split between ingress and egress paths. In such a typical system, all ingress traffic is forwarded through the switch fabric towards the egress device. For systems that utilize the device's local switching (without a switch fabric) but still maintain the ingress/egress functional split, the traffic is locally switched from ingress to egress via TM loopback. The egress target TM hierarchy and target interface/port is determined by the ingress application.

Typical ingress path:

1. Ingress application and decision of target device/port.
2. Ingress packet modification to traverse switch fabric
3. Ingress WRED
4. Ingress TM queuing before switch fabric or loopback.

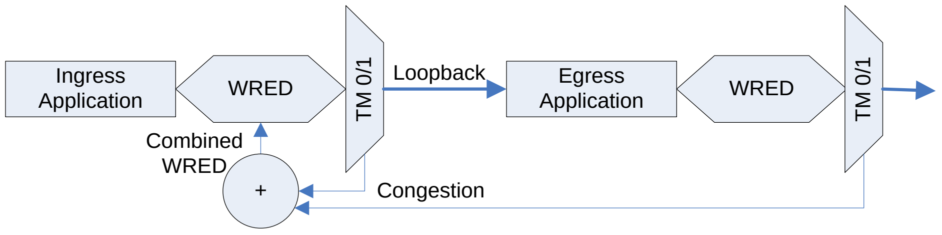
Typical egress path:

1. Egress packet modification
2. Egress WRED
3. Egress TM queuing

For split ingress/egress applications that utilize local switching via loopback, the ingress application can be aware of the egress TM congestion since ingress and egress TM functions are on the same device (could be in implemented in different TM units).

The ingress application is aware of the egress TM queuing congestion and thus it is beneficial to exercise egress aware WRED before ingress queuing. The benefit is evident when a WRED decision is to drop an ingress packet before the loopback since it will be eventually dropped by the egress WRED.

The TM supports combining ingress + egress WRED decisions to facilitate early drop and loopback bandwidth preservation.



10.6.8.1 Combining WRED Accounting

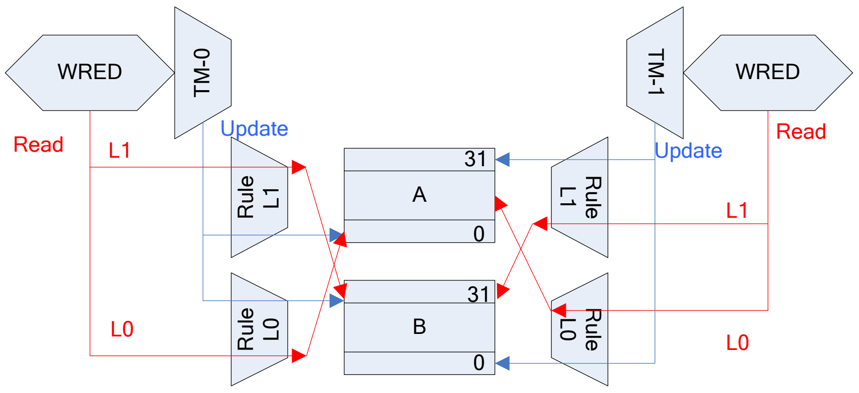
Each TM allocates 2x32 counters for both TM's congestion co-accounting. The counters are flexibly assigned to range of L0 or L1 TM hierarchal entities. These shared congestion counters are updated by both TMs on enqueue/dequeue events. The TM that performs the ingress WRED, combines the shared counter with its local congestion counter to yield a combined WRED decision based on ingress + egress congestion.

The shared counters can be asymmetrically assigned by both TMs to hierarchal levels. The ingress TM-0 can assign a shared counter to L1 entity while the egress TM-1 can assign the same shared counter to an L0 entity.

There are two (A/B) shared counter tables; each 32 counters long.

Each TM assigns each table to hierarchy L0 or L1. Each TM has entity assignment rules to shared counters for both tables A/B. The same counter index in A and B is used by one TM to update the other WRED decision.

Egress TM-1 updates counter N in table A (assigned to L0) while ingress TM-0 performs WRED on counter N in table A (assigned to L1).



Typical configurations examples:

USAGE CASE	***TM-1***		***TM-0***	
Egress TM-1 Ingress TM-0	Rule for update to 64 L0 entities (32 in A, 32 in B)	A: L0 B: L0	Rule for WRED 32 L1 entities in A. Match on A rule reads B index. Rule for WRED 32 L1 entities in B. Match on B rule reads A index.	A:L1 B:L1
Egress TM-0 Ingress TM-1	Rule for WRED 32 L1 entities in A. Match on A rule reads B index. Rule WRED for 32 L1 entities in B. Match on B rule reads A index.	A: L1 B: L1	Rule for update to 64 L0 entities (32 in A, 32 in B)	A: L0 B: L0
Ingress/Egress TM-0			Rule in B for update to 32 L0 entities. Rule WRED for 32 L1 entities in A. Match on A rule reads B index.	A: L1 B: L0
Ingress/Egress TM-1	Rule in B for update to 32 L0 entities. Rule WRED for 32 L1 entities in A. Match on A rule reads B index.	A: L1 B: L0		

When an egress TM L0 entity needs to update ingress TM L1 entities, the 32 entry table can be partitioned:

- Entries updated by TM-0 L0 and WRED by TM-0 L1.
- Entries updated by TM-0 L0 and WRED by TM-1 L1.
- Entries updated by TM-1 L0 and WRED by TM-1 L1.
- Entries updated by TM-1 L0 and WRED by TM-0 L1.

10.6.9 High Priority Shallow Queues Affect on Scheduling Performance

Each TM scheduler’s maximal decision rate is TMCoreClock/2.

A TM hierarchical entity with high priority traffic is given precedence for scheduling decisions. Such a high priority entity gets many scheduling decisions at the expense of other entities.

Several scheduling decisions are issued before entity queue status is updated, due to long DRAM latency to fetch the Queue Descriptor.

When such a high priority entity has just a few queued packets (shallow queue condition), only a few scheduling decisions are actually used while subsequent scheduling decisions are wasted. This may potentially degrade overall scheduling performance.

The TM is equipped with tunable mechanism to detect shallow queues in order to limit the number of scheduling decisions issued to shallow queues.

The mechanism is implemented for all TM hierarchies. The mechanism utilizes the WRED measured queue congestion monitoring by several thresholds. The queue shallowness is measured per COS priority.

The shallow level is used to control the pace of entity scheduling decisions.

The packet enqueue COS must be set to scheduling conforming (green) priority.

Table 10-7. Shallow level controlling the pace of entity scheduling decisions

TM HIERARCHY	SHALLOW BUFFER COUNTERS PER ENTITY	THRESHOLDS	SHALLOW INDICATIONS	MAX. DECISION RATE
L0	4 dedicated shallow counters: COS-0 COS-0,1 COS-0,1,2 COS-0,1,2,3	3 common to all counters	Per priority 4 shallow levels	4 decision rates one per shallow level
L1	4 dedicated shallow counters: COS-0 COS-0,1 COS-0,1,2 COS-0,1,2,3	3 common to all counters	Per priority 4 shallow levels	4 decision rates one per shallow level
L2	2 shallow counters: Shallow (excess traffic) counter, Shallow (total) counter	3 thresholds	4 shallow levels	4 decision rates one per shallow level
L3	Single counter: Common WRED and Shallow Configurable for total or excess.	3 thresholds	4 shallow levels	4 decision rates one per shallow level
L4	Single total counter WRED and Shallow.	3 thresholds	4 shallow levels. Per TM chunk, 64 shallow indications are cached.	4 decision rates one per shallow level

Each TM entity is limited by clocks per scheduling decision based on its Shallow level and configured decision rate per TM hierarchy level / Shallow level.

10.7 Traffic Conditioning - Shaping

The NPS-400 features shaping for allocating maximum bandwidth guarantees without packet drop, while smoothing bursty traffic flows. Shapers can be configured to a variety of ranges starting from 1 Kbit per second and up to 480 Gbits per second. The shaper mechanism uses a real time clock (RTC) and credit system to determine both the rate and the maximum burst sizes. Depending upon the RTC and credit size configuration, a few dynamic ranges of rate shaping may be achieved.

Dual rate shaping is supported on TM hierarchical Levels 0-4

The shaper engine in Level 0 to Level 3 is designed to provide shaping of 4Kbps to 480 Gbps at less than 0.1% inaccuracy.

The TM L4 shaper has two modes:

- Low latency (bursty) shaper
- Low jitter shaper

Low latency shaping at Flow level (TM L4) supports dual shaping (rate limiting) from 1Kbps to 480 Gbps (10Gbps accurate to 0.1%). This may result in bursts of traffic and then stall until the next transmission; adequate for data traffic.

The low jitter shaper supports single rate shaping from 1Kbps to 500Mbps (accurate to 0.1%). Low jitter constant interval transmission by limiting traffic burstyness; adequate for real time streaming traffic.

The low jitter shaper mode is supported for TM flows with all scheduling modes: normal, dual, quad, octal scheduling (i.e. scheduling of 1, 2, 4 or 8 packets at a time) and Internal Queue Descriptor modes.

Low-jitter shaper mode should be used where jitter needs to be minimized, and the queue is provisioned in a priority such that congestion is not likely to occur for that queue. Low jitter shapers time the scheduling events in such a way that achieves a more predictable delay and thus minimal jitter.

▸ *Due to the nature of scheduling of low jitter queues, congestion management, i.e. WFQ cannot achieve high accuracy and may deviate from expected results under congestion.*

It is recommended to use the low jitter shaper mode as well for traffic which is high priority that should not experience congestion.

Note that high-priority TM flows are regarded as uncongested low bandwidth, and should be assigned to low-jitter shapers. Since they are uncongested, their fairness attributes (WFQ weights) should be discarded, or merely used for jitter-limitation purposes only (i.e. high-weight flows will suffer higher-jitter transmission as opposed to the same-priority flow with a lower weight).

Low latency queues do not time their scheduling, and thus under low congestion can supply low delay, however during congested periods the delay in low latency queues in not deterministic as in low jitter and is subject to WFQ scheduling. When high priority traffic is configured as low latency, some performance hit may be observed since the scheduler may spend many speculative scheduling cycles per packet, in order to prefer latency on top of performance. In order to eliminate a performance hit, it is recommended to use low jitter queues for high priority traffic, where WFQ accuracy is less important.

10.7.1 Shaping Features

Optional traffic shaping in hierarchical levels:

- Level 0:
  - Dual shaper yielding combined three color status: green/yellow/red.
  - 16 rate shaping profiles that can be assigned to commit rate or/and peak rate of the dual shaper.
  - Any shaper can be associated independently to commit and peak generating up to 16x16 effective profiles based on 16 shaping rates.
- Level 1:
  - Dual shaper yielding combined three color status: green/yellow/red.
  - 128 rate shaping profiles that can be assigned to commit rate or/and peak rate of the dual shaper.
  - Any shaper can be associated independently to commit and peak generating up to 128x128 effective profiles.
  - 2 profiles reserved for always-positive and always-negative shaper status, enabling dedicated shaping modes (unlimited excess, unshaped, excess-only & no-excess, halted).
  - The Fast queues are shaped by Level 1 shapers.
- Level 2:
  - Dual shaper yielding combined three color status: green/yellow/red.
  - 4K shaping profiles that can be assigned to commit rate and/or peak rate of dual shaper rates.
  - For each chunk of 1K Subports,
    256 general purpose rates of any value
    + 768 rates based on up to 15 base rates scaled by 1, 2,...256.
      - Alternatively 3K rates based on 3 base rates scaled by 1, 2,...1K.
  - Grid shapers span pre-defined rate ranges with each entity assigned to any rate within the range
  - 2 profiles reserved for always-positive and always-negative shaper status, enabling dedicated shaping modes (unlimited excess, unshaped, excess-only & no-excess, halted).
  - Any shaper can be associated independently to commit and peak generating unique combinations of commit/peak rates for each of the 4K subports.
- Level 3:
  - Dual shaper yielding combined three color status: green/yellow/red.
  - Possibility to shape with tree shapers, commit, excess and peak combined three color status: green/yellow/red. The ‘red’ status results when either excess or peak is non conforming.
  - 4K shaping profiles that can be assigned to commit rate and/or peak rate of dual shaper rates.
  - For each chunk of 8K Classes,
    256 general purpose rates of any value
    + 768 rates based on up to 15 base rates scaled by 1, 2,...256.
      - Alternatively3K rates based on 3 base rates scaled by 1, 2,...1K.
  - Grid shapers span pre-defined rate ranges with each entity assigned to any rate within the range
  - 2 profiles reserved for always-positive and always-negative shaper status, enabling dedicated shaping modes (unlimited excess, unshaped, excess-only & no-excess, halted).
  - Any shaper can be associated independently to commit and peak generating unique combinations of commit/peak rates for each of the 32K classes.
- Level 4:
  - Dual shaper (green/yellow /red) that controls scheduling priority and WFQ weight assignment.
  - 256 shaping profiles. Each Flow service profile configures the shaper to low latency or low jitter mode.

- 2 profiles reserved for always-positive and always-negative shaper status, enabling dedicated shaping modes (unshaped, halted)

Configurable shaping parameters:

- Profile parameters:
  - CIR – Committed Information Rate
  - CBS – Committed Burst Size
  - PIR – Peak Information Rate (for dual shapers only)
  - PBS – Peak Burst Size (for dual shapers only)
  - Global Shaper parameters:
    - Real time clock (RTC)
    - Maximum burst size
  - Each queue has two profile pointers: one for commit (CIR+CBS) and one for excess (PIR+ PBS). Hence the number of profiles is actually multiplied; CIR and PIR can be chosen independently.
- Operation mode of the dual token bucket can be also configured to achieve specific behaviors:
  - Green + Yellow + Red (EXPLICIT 3-color dual token bucket profile)
  - Green + Yellow (UNLIMITED_EXCESS)
  - Yellow + Red (NO_COMMIT)
  - Green + Red (NO_EXCESS)
  - Green only (UNSHAPED)
  - Yellow only (BEST_EFFORT)
- Shaper parameters:
  - L4 shaper 18b credit token counter (17b value + sign bit). Counter can be in token deficit.
  - L3-L0 shaper 27b credit token counter. Counter can be in token deficit.
  - L4 shaper credit token update value of 17b.
  - L3-L0 shaper credit token update value of 18b signed.
  - 15 Credit token byte size values: 1/128, 1/64 …1/2, 1, 2…64, 128.
  - 4 RTC for token update base rate plus RTC for low jitter shapers token update rate.
  - Typical L0-L2 shaper configurations ranges:

RTC 128,000 tokens/second

CREDIT TOKEN SIZE (BYTE SIZE VALUE)	*MIN RATE (BYTES/SEC)*	*MAX RATE (GB/S)*	*MAX BURST (MB)*
1/128 B	1000	2	1
1/8 B	16,000	33	16
½ B	64,000	134	64
8 B	1,024,000	Unlimited	1024

  - Typical L3 shaper configurations ranges:

RTC 20480 tokens/second

CREDIT TOKEN SIZE (BYTE SIZE VALUE)	*MIN RATE (BYTES/SEC)*	*MAX RATE (MB/S)*	*MAX BURST (MB)*
1/128 B	160	335	1
1/8 B	2,560	5368	16
½ B	10,240	21474	64
8 B	163,840	Unlimited	1024

10.7.2 Token Bucket Mechanism

A single or dual token bucket mechanism is used for shaping traffic out of the TM, enabling the maximum bandwidth per entity to be limited. When a single token bucket is applied, the flow's conformance is marked with one of two colors. Green implies that the flow is within its bandwidth limitation and is eligible for transmission. Red implies that the flow has exceeded its maximum bandwidth limitation, and should be delayed until a new budget is given to it.

When a dual token bucket is applied, a flow is marked with one of three colors. Green implies that the flow is within its committed bandwidth limitation and is eligible for transmission. Yellow means that the flow has exceeded its committed bandwidth limitation, but is within its excess bandwidth limitation, and can be transmitted only if there are no green flows pending. Red implies that a flow has exceeded both its committed and its excess bandwidth limitations, and further transmission decisions should be delayed until a new budget is given to it.

The dual shaper can also be configured for Single Rate Three Color Marking (srTCM) with configured burst size for Green traffic, and burst size for Excess traffic.

Each token bucket mechanism gets an additional budget once every refresh interval, via Real Time Clock (RTC), that may be configured. The number of credits and RTC refresh interval determine the CIR (Committed Information Rate) and PIR (Peak Information Rate) flow shaping parameters when a dual token bucket applies. The size of the buckets determines the CBS (Committed Burst Size) and PBS (Peak Burst Size).

Once a frame is scheduled for transmission, shaping buckets are updated with the frame's byte count. Per dual shaper color status each bucket can be configured to be updated. The shaper profile determines which token buckets are updated based on shaper status to allow conformance to IETF or MEF standards.

Each scheduling hierarchy has its own configurable sets of parameters and its own shaping mechanism, to determine and measure the specific context shaping values.

10.7.3 L2/L3 Shaper Rate Configuration

The traffic rate shaper of TM hierarchical levels L2 and L3 is configured through 4K shaping profiles that specify commit rate and/or peak rate of dual shaper rates.

Two profiles are reserved for always-positive and always-negative shaper status, enabling dedicated shaping modes (unlimited excess, unshaped, excess-only & no-excess, halted).

Any shaper rate can be associated independently to commit and to peak generating unique combinations of commit/excess/peak rates.

Each TM chunk (one-quarter TM) has dedicated configuration profiles.

There are two profile types:

- Explicit rate profile – Each profile is configured with explicit rate and burst size.
- Rates grid profile – The rate is set by multiplying a selected base rate.

The profile type/index is coded by a 12b profile select value:

PROFILE SELECT	[11:10]	[9:8]	[7]	[6:0]	COMMENT
Explicit rate	00	00	Explicit rate index		256 explicit rates
Rate grid	[11:8] not equal 0 Grid profile index				7b, 8b, 10b multiplier per grid profile type
		Base rate multiplier			

Grid profile

There are up to 30 grid profiles. The grid profile type may cause 2, 4 or 8 profiles collapsing into a single profile.

PROFILE TYPE	PROFILE-SELECTOR MULTIPLIER BITS	MAX. GRID PROFILES	RATE	RATE RANGE
7b rate fraction multiplier	[6:0]	30	Base + Base*Multiplier/128	Base to 1.992*Base
8b rate integer multiplier	[7:0]	15	Base * Multiplier	Base to 256*Base
8b rate fraction multiplier	[7:0]	15	Base + Base*Multiplier/256	Base to 1.996*Base
10b rate integer multiplier	[9:0]	3	Base * Multiplier	Base to 1K*Base

Grid profiles dynamic range

The 30 profiles of 7b rate fraction multiplier, allow +/- 0.4% accuracy over 2^{30} rate dynamic range. This dynamic range can cover 186Hz to 200Gbps.

10.7.4 IPG Emulation for Shaping, WFQ and Statistics

When shaping traffic into different kinds of underlying physical interfaces, rate mismatch can occur due to the differing overheads applied on the physical level. Overhead mainly results from encapsulation or decapsulation of packets with a constant overhead per frame, or sometimes from packet segmentation with per-segment overhead, or both. When it is important to account for this overhead (for example, to eliminate downstream congestion or packet drop when there is no explicit backpressure mechanism), Inter Packet Gap (IPG) emulation mode can be enabled to adjust the shaping rates accordingly.

The IPG emulation mechanism operates by altering the length of the frame upon scheduling for shaper accounting. IPG can be also used to change the frame length when updating the statistics counter. IPG for QoS and IPG for Statistics can be enabled or disabled independently on a per level basis. Furthermore, IPG can also affect the TM scheduling WFQ (fairness) mechanism at each level independently.

The IPG field in the frame descriptor is 6 bits long:

[a] 5-bit mantissa (-16 to +15)

[b] 1-bit exponent-selector (mode dependent).

The exponent(s) is a global value, and can multiply the mantissa by x1, x4, x16 or x128.

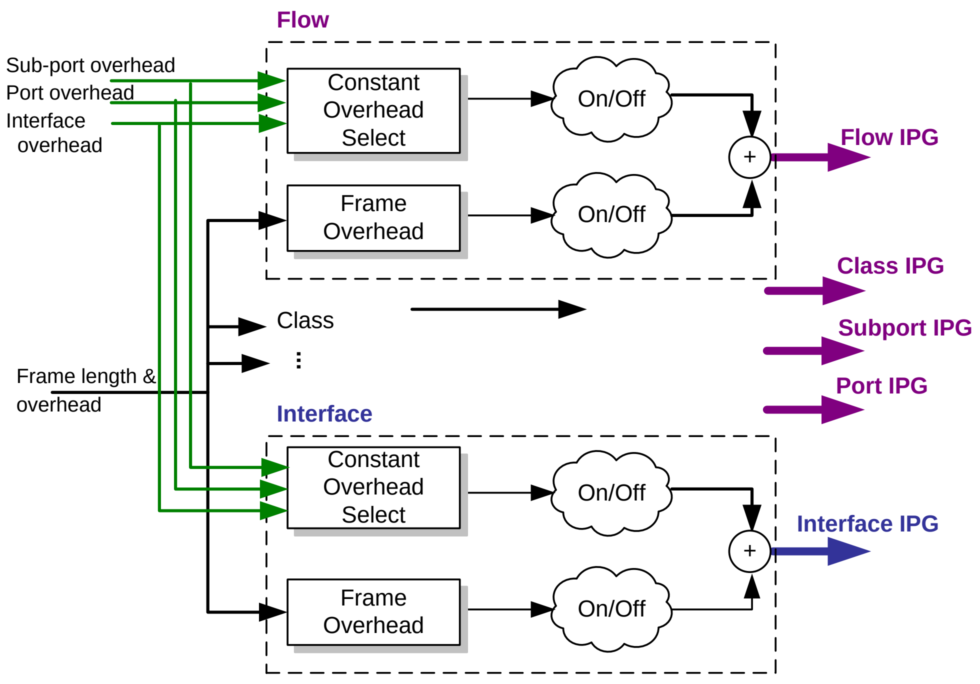
Thus, the IPG range is (-16..+15) x (1/4/16/128) bytes.

In addition, there is a per-level constant overhead. This overhead is only generated at the lower levels (Interface/Port/Subport), but can be used in each of the participating 5 levels. The Constant Overhead and the per Packet Overhead affects can be disabled independently in order to produce the required IPG value.

There are three Constant Overhead IPG settings: Subport IPG, Port IPG, and Interface IPG. Each one represented by IPG byte value between (-)16*128 to (+)15*128.

Each TM level selects which Constant Overhead IPG to use.

Figure 9-25. Inter Packet Gap emulation in the 5 levels



The IPG settings vary between the levels; while in the higher levels there's a single setting representing the entire level, the aggregated levels (Subport, Port and Interface) which usually represent separate (and different) physical entities have higher degree of differentiation:

- Flow: a single setting
- Class: a single setting
- Subport: per-nibble setting
- Port: per-queue setting
- Interface: per-queue setting

Each of the higher levels has a cell overhead in addition to the global overheads.

The cell IPG overhead accounts for frame overhead, original frame size, cell header, EOP cell fragmentation padding to minimal size.

Note that a negative-IPG value cannot exceed the minimum packet size in L4. For example, if the smallest packet-size is 64B, then the overall IPG overhead (for frame + cell + global) that affects L4 shaping cannot be smaller than -63B.

10.8 Congestion Management – WFQ and Priority

The NPS-400 TM uses the following mechanisms to determine the allocation of available bandwidth during congestion:

- WFQ – Weighted Fair Queuing: enables each of the participating queues to receive a fair share of the available bandwidth, where fairness is defined by weights.
- Priority Scheme – Queues can be assigned with up to 8 strict priority levels. Higher priority queues are always scheduled before lower priority queues.
- Combination of both WFQ and Priority.

The NPS-400 uses WFQ to maintain byte-wise fairness between all queues in any scheduling level, as well as to guarantee minimum bandwidth allocation per hierarchical element, under a congested system.

WFQ in NPS-400 is implemented using a Modified Deficit Round Robin algorithm, which allocates bandwidth according to weights, while minimizing the latency and jitter that is caused in various simple round robin algorithms.

WFQ can be configured to act as WRR by counting packets instead of bytes.

A multiple-WFQ implementation allows the configuration of different weights, thus assigning WFQ behavior per priority.

10.8.1 WFQ Features

Congestion management: WFQ (based on MDRR) / WRR in all hierarchical levels:

- Level 0:
  - Relative weights of 1...8K
  - 16 different profiles, each holds per-priority weight for multiple WFQ capabilities; enables separate fairness for commit/excess traffic scheduling.
  - Five priority levels. (WFQ is performed among all high priority queues, and all low priority queues separately)
- Level 1:
  - Relative weights of 1...8K
  - 64 different profiles, each holds per-priority weight for multiple WFQ capabilities; enables separate fairness for commit/excess traffic scheduling.
  - Five priority levels. (WFQ is performed among all high priority queues, and all low priority queues separately)
- Level 2:
  - Relative weights of 1...8K
  - 128*(4 priorities) / 256*(2 priorities) - Different profiles per chunk, each holds per-priority weight for multiple WFQ capabilities; enables separate fairness for commit/excess traffic scheduling
    128 profiles that can mange four priorities. This is achieved by sharing the 128 available profiles by the L2 and L3 hierarchies.
  - Each hierarchical level (L2/L3) can be configured to operate with two WFQ priorities. With two WFQ priorities, the number of WFQ profiles is 256. With two WFQ priorities, each TM L2/L3 entity has a selector of which two priorities to use.
  - Four priority levels. (WFQ is performed among all high priority queues, and all low priority queues separately).
- Level 3:
  - Relative weights of 1...8K
  - 128*(4 priorities) / 256*(2 priorities) - Different profiles per chunk, each holds per-priority weight for multiple WFQ capabilities; enables separate fairness for commit/excess traffic scheduling

    128 profiles that can mange four priorities. This is achieved by sharing the 128 available profiles by the L2 and L3 hierarchies.

  - Each hierarchical level (L2/L3) can be configured to operate with two WFQ priorities. With two WFQ priorities, the number of WFQ profiles is 256. With two WFQ priorities, each TM L2/L3 entity has a selector of which two priorities to use.

- Level 4:
  - Relative weights of 1...8K
  - TM L4 number of WFQ profiles is 256 per chunk.
  - 256 different profiles per chunk, each holds per-priority weight for multiple WFQ capabilities; enables separate fairness for commit/excess traffic scheduling.
  - Eight priority levels. (WFQ is performed among all high priority queues, and all low priority queues separately).

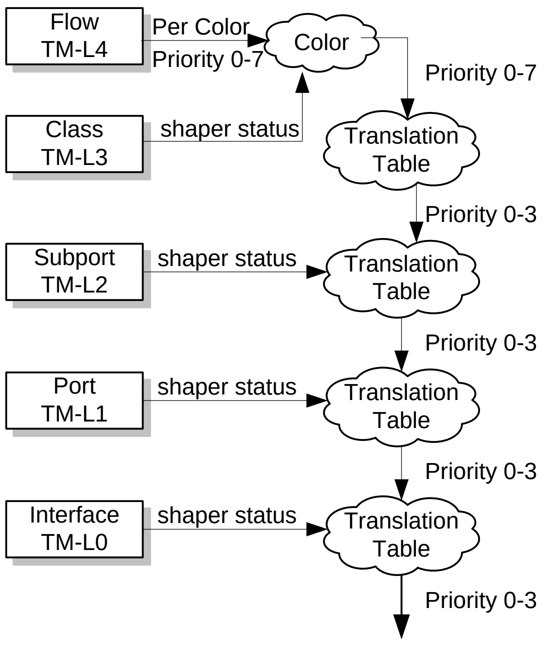
10.8.2 Priority Scheduling

The TM scheduler is based on a combination of a work conserving and a non work conserving schedulers. The non work conserving scheduler is based on the shaping mechanism and may stop, resume or change the priority dynamically for a specific queue or hierarchy. The work conserving scheduler is based on a priority scheduler and WFQ scheduler, where both of these mechanisms never stop scheduling packets as long as there are eligible queues to serve.

The work conserving scheduler is built on eight strict priority level scheduling where packets from a higher priority level will always be scheduled before packets from a lower priority level. In case of competing packets from the same priority level, the WFQ scheduler will schedule them based on the weight assigned to each of their queues.

Priority can propagate from the TM Flow level down to the Interface level such that the hierarchical scheduling process will take into consideration a global view of the priority for all of the Flows simultaneously. For example, a Class that has high priority Flows will always be scheduled before a Class that has lower priority Flows.

Figure 9-26. Priority scheduling for the 5 levels



Priority propagation may be enabled, disable or partially enabled based on configuration.

Users can define a translation table to map between the 8-priority levels at TM hierarchy level 4 (Flow) to four priority levels at the lower TM levels.

At each other level (Class to Subport, Subport to Port, Port to Interface), a translation table exists to map between four priority levels of upper queues and local shaper status (commit, yellow, red) to an output priority level.

For L4 to L3 priority translation, each TM supports 4 translation tables (one per chunk).
For L3 to L2 priority translation, each TM supports 4 translation tables (one per chunk).

Usually all per chunk tables are configured the same.

For L2 to L1 priority translation, there is a single table.
For L1 to L0 priority translation, there is a single table.

10.8.3 L3 Inter-chunk Fairness

The fairness among up to 8K L3 entities is supported by a single chunk by mapping all L3 entities to a single L2 entity.

To allow fairness among more than 8K L3 entities, among 1, 2 or 3 chunks:

- Up to 16 L2 entities per chunk can be fairness disabled (degenerated entities).
- L2 entities of 1, 2 or 3 chunks can be mapped to the same L1 entity.
- It is recommended to disable the L2 shapers (of the degenerated entries) for L3 inter-chunk fairness.

10.9 TM Statistics

The TM activity can be monitored by statistics accounting.

The TM packet en-queue, WRED drop and schedule events can be monitored.

The statistics collection includes event based accumulations and periodic reports of status.

The statistics can be packet (event) count or packet byte size accumulation.

The statistics is differentiated per TM hierarchy, COS and scheduled priority as well as by SW control.

Each TM counter type can be configured as on-demand or posted. It is recommended to associate TM statistics with posted counters.

Table 10-8. TM statistics

TYPE OF STATISTICS	*STATISTICS GROUPS, MEMORY ADDRESS*	*TARGET MEMORY*
WRED en-queue event	Single group set per globally selected TM hierarchy level. Per packet SW set StatID for counter indexing. Per packet code based statistics.	EMEM posted or IMEM/EMEM on-demand
WRED packet drop event		
Schedule packet event	Group per scheduler color (green, yellow, red) on globally selected TM hierarchy level.	
Time-out packet drop event	Single group per globally selected TM hierarchy level.	
Hierarchy queue occupancy report, en-queue/de-queue event based*	Group per hierarchy level L4-L0 and global.	IMEM/EMEM on-demand
Hierarchy scheduler color status periodic report	Group per hierarchy level L4-L0 and global.	IMEM/EMEM on-demand
Hierarchy queue occupancy periodic report	Group per hierarchy level L4-L0	

*For more information see the TM Congestion Monitoring and Messaging for E2E Flow Control section.

TM processing of a single packet may trigger up to 14 statistics events: packet WRED en-queue event, en-queue occupancy reports per 5 TM levels and global, packet schedule event, packet schedule queue occupancy reports per 5 TM levels and global.

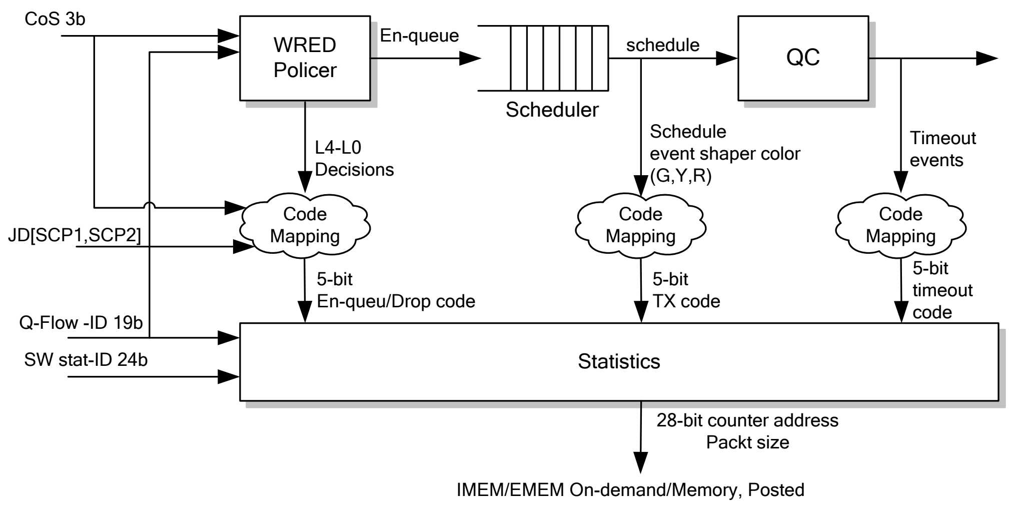
The TM statistics can index up to 16M counters.

The statistics per group can be randomly spread to load balance target memory. The statistics can be replicated to target memories.

The TM counters have configurable 28-bit offset.

The Figure 9-27 illustrates TM statistics events.

Figure 9-27. TM Statistics diagram per TM engine



10.9.1 TM Statistics Features

- Byte count and/or frame count statistics gathering dedicated base-address for each event type.
  - Byte-based accounting on a single statistics counter (any counter type)
  - Frame-based accounting on a single statistics counter (any counter type)
  - Frame+byte accounting on a single dual/double counter.
- Per packet counting is done according to one of the three types of TM events:
  - En-queued /dropped by WRED
  - Dropped by timeout
  - Scheduled frames
- The WRED statistics can be differentiated based on:
  - WRED Statistics can be reported based on the queue's topology or based on a per-frame dedicated ID; when the latter is used, only WRED-based statistics may be gathered.
  - TM Queue Flow- ID (per frame TM topology attribute) / Stream StatID (per-frame SW attribute)
  - TM WRED statistics are counted concurrently with Flow-ID and Stat-ID; in each case a dedicated CODE factor (taken from a dedicated table) is used to select a statistics counter. The end-user is able to affect the code selection via two dedicated 'profile' fields in JD(SCP1, SCP2).
  - Packet’s initial CoS
  - WRED pass/drop decision
  - WRED profile
- With/without reference to IPG (Scheduler statistics only)
- The Time-Out and Forwarding events are counted for all queues for one of the five hierarchical levels (i.e. per-Flow/Class/Subport/Port or Interface). The TM level for statistics reporting is globally selected.
  - The TM scheduled traffic event statistics are gathered per a selected TM hierarchy entity ID.
- Scheduler (transmitted-only) statistics have a Statistics code based on the selected TM hierarchy shaper color. For example, when a green queue transmits a packet of priority 2 one counter can be incremented, and when yellow queue transmits frame of any priority, a second counter is incremented.
- Statistics counters are managed in internal or external memory as either on-demand or posted counters.
- Additional cumulative statistics are kept internally within the TM block and may be accessed by the SW. These are aggregated statistics which may have different attributes than the regular, external statistics. Cumulative statistics are also pushed to the external memory, but at a lower rate.
- Periodic queue status updates (PSR) are also directed to the Statistics block. PSR counters are on-demand type counters.
- Probe statistics are also available; these statistics are gathered for a specific-queue, regardless of the reported external statistics. Probe statistics are also pushed to the IMEM/EMEM memory, but at a lower rate. (Mostly used for debugging.)

10.9.2 Statistics Accumulation

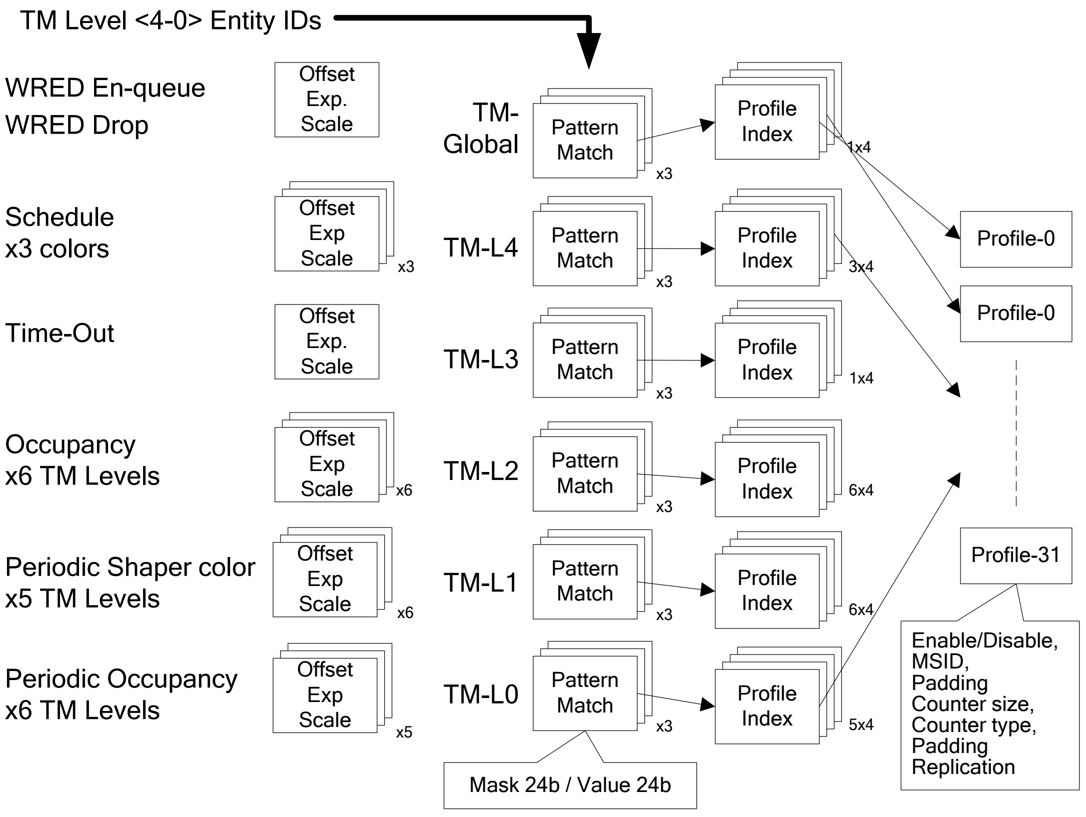
Each statistics type group has a configurable method to assign per counter address:

(1+Offset[9:0]/1024)*(2^Exp[5:0]) + Q-ID*(2^Scale[2:0]) + Mapped Code[4:0].

The Offset, Exp and Scale are per statistics type.

The Mapped Code is per WRED decision for WRED statistics and per scheduler shaper color per scheduler statistics.

The packet-event associated TM-hierarchy levels are compared to three Pattern Match entries. Each Patter Match entry is comprised of 24b mask/value. The Pattern Match result selects one of four Profile Indexes (3 indexes associated with 3 PMs and one default match index). The Profile Index points to one of 32 profiles that define the TM statistics transaction.



10.9.2.1 Statistics Profile

The profile configures the parameters for statistics accumulation and memory addressing:

- Enable/disable the pattern-match/profile.
- MSID select to target memory region.
- Counter-type/size – single/double counter allocated in 32b/64b/128b.
- Pad address with zeroes.
- Shuffle/scramble configurable address bits for memory banks load balance.
- Replication mode/number. Replicate or load balance counter to 1/2/4/8 addresses.

The statistics replication allows:

- Load balancing the high rate statistics counter update to several memory banks and posted statistics units.
- Parallel update several counters at a low rate while allowing high rate load balance polling of the counters.

10.9.2.2 TM Entity Pattern Match

The statistics monitoring allows selecting a portion of TM hierarchy level entities. The selection is done by comparing the packet event entity ID with mask/value patterns:

Entity ID & Mask =?= Value

There are three configurable patterns per TM statistics group type. The three patterns are compared by fixed priority. First matched pattern selects the associated statistics profile. When none of three patterns is matched, the default profile is selected.

The entity ID is used for statistics counter address calculation. Entity ID bits that are masked out are removed from calculation to compress the address space. Example, if two LSBs are masked out the consecutive addresses are assigned to entities 4*N.

10.9.2.3 WRED Policer Statistics

The WRED en-queue (and drop) event is monitored by statistics on a globally selected TM hierarchy level. The TM level entity of the en-queued packet is compared to three mask/value patterns. Parallel to TM hierarchy topology based statistics, the WRED statistics can be enabled per packet SW set Stat-ID of 24b.

The WRED statistics can count events, packet bytes or both in different counters or the same double counter. The packet bytes statistics can be scaled down by 64B resolution by rounding (floor, ceiling, deterministic round and statistical round).

The WRED statistics can be based on per packet set Code Profiles that conveyed by the Job Descriptor. The Statistics Code Profile 1 (SCP1) field participates in generation of a 5-bit WRED policer CODE in either topology based statistics or Stat-ID based statistics.

There are two code tables and each table generates a 5 bit code. Tables are indexed by an address comprising:

- WRED code pass/drop decision, an aggregated code depending on origins for pass drop decision,
- Packet's initial CoS 3b,
- SCP1 and SCP2.

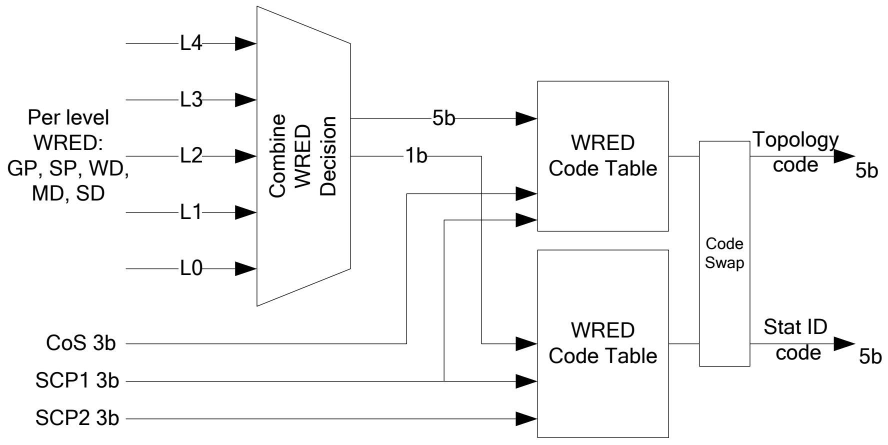
Each 5b code selects an offset in a block of up to 32 counters. One code is used for topology based reporting while the other is used for Stat-ID reporting (and both can be used in parallel).

Topology based reporting counter ID is:

(1+Topology Offset[9:0]/1024)*(2^Exp[5:0]) + Q-ID*(2^Scale[2:0]) + WRED Topology Code[4:0].

Stream ID based reporting counter ID is:

(1+StatID Offset[9:0]/1024)*(2^Exp[5:0]) + Q-ID*(2^Scale[2:0]) + WRED StatID Code[4:0].



The output of the two code tables can be swapped to allow more detailed WRED decisions for one of the WRED statistics schemes.

10.9.2.4 TM Scheduler Statistics

The packet schedule event is monitored by statistics on a globally selected TM hierarchy level. The TM level entity of the scheduled packet is compared by three mask/value patterns. The scheduling events are discriminated per scheduling color, with per color configuration.

The packet schedule event statistics can count events, packet bytes or both in different counters or the same double counter. The packet bytes statistics can be scaled down by 64B resolution by rounding (floor, ceiling, deterministic round and statistical round).

The schedule shaper color (Green, Yellow, Red) is remapped to 5b statistics code for counter indexing:

(1+Offset[9:0]/1024)*(2^Exp[5:0]) + Q-ID*(2^Scale[2:0]) + Mapped Code[4:0].

10.9.2.5 TM Scheduler Time-out Statistics

The packet schedule time-out event is monitored by statistics on a globally selected TM hierarchy level. The TM level entity of the schedule time-out packet is compared to three mask/value patterns. The packet schedule event statistics can count events, packet bytes or both in different counters or the same double counter. The packet bytes statistics can be scaled down by 64B resolution by rounding (floor, ceiling, deterministic round and statistical round).

The drop action (Time-out / Purge) is remapped to 5b statistics code for counter indexing:

(1+Offset[9:0]/1024)*(2^Exp[5:0]) + Q-ID*(2^Scale[2:0]) + Mapped Code[4:0].

10.9.2.6 TM Occupancy Reports Statistics

The TM hierarchy entities occupancy is monitored by statistics on all TM levels. The packet en-queue and de-queue (scheduling) results in occupancy change and thus may trigger occupancy reports. (See the TM Congestion Monitoring and Messaging for E2E Flow Control section.) Each TM level has a dedicated configuration, entity pattern matching and profile assignment.

The periodic statistics monitoring is supported for shaper color and TM hierarchy entities occupancy. The shaper color reports are supported for all TM levels. The occupancy reports are supported for all TM levels and global TM occupancy. Each report/level has configurable parameters, entity pattern matching and configuration profile.

10.10 TM Congestion Monitoring and Messaging for E2E Flow Control

The TM entity queue occupancy (congestion) is periodically reported via messages and logged as status to statistics counters (bit wise statistics).

TM congestion can be monitored at each hierarchical level and on all levels of the TM (per TM engine).

There are two methods to monitor congestion:

- Probabilistic method – suitable for TM-L3 and TM-L4: The packet en-queue and de-queue events trigger random decisions to generate congestion update reports.
- Congestion segmentation method – suitable for TM-L0/L1/L2: The TM entity queue occupancy range is segmented into regions and congestion status region crossing triggers update report generation.

Each TM level can be configured to either method.

The method determines the update messaging rate and update accuracy tradeoffs. As congestion updates are logged by statistics, a high update rate may cause high statistics bandwidth consumption.

Each TM hierarchical level entity (and TM global) can be selected for congestion monitoring:

- Each level Entity ID * Mask bit map is compared to Value bit map to be selected.
- Each level is configured with three bit map filters for classifying to four groups.
- Each group is assigned to a profile (out of 32 available).
- Each profile configures: Enable count, Statistics memory target (MSID- Memory space ID), single or double counter and counter size 32b /64b. The profiles can also filter out events.

10.10.1 Congestion Change based Probabilistic Report Generation Method

The update event generation is probabilistic. The probability is controlled via configuration per TM level.

The probability is determined by P = PacketSizeProbability / CongestionFactor.

Packet size can be measured in units of 1B, 2B, 4B ...16KB as configured per TM level.

The CongestionFactor is selected based on TM queue hierarchical entity change level. Thus congested queues that undergo significant changes can have a higher probability of generating updates.

The TM queue hierarchical entity congestion change is measured by a combination of two values:

- ShangedMSB – The bit index of the congestion counter MSB that was changed after an en-queue/ de-queue event.
- SetMSB – The bit index of the congestion counter MSB that is set to '1'.

The index difference: ChangeMeasure = SetMSB - ShangedMSB

The ChangeMeasure difference gets a high value when the congestion change due to an event is not significant. The ChangeMeasure difference gets a low value when the congestion change due to an event is significant.

For example, if the queue size had changed from 0x00200135 to 0x00200147, then:

- ShangedMSB = 6 and SetMSB = 21
- ChangeMeasure = 21- 6=15 (larger value indicating a small change).

The SetMSB value (0-28) is logarithmic proportional to queue congestion. When SetMSB is greater than a pre-configured, per-level, maximal normal congestion (MAX_Y) the probability of generating update reports is set to a pre-configured value PROB_LARGE.

When SetMSB value is not larger than the maximal normal congestion, the probability to generate updates is determined by a combination with the ChangeMeasure value.

The ChangeMeasure is used as an index to a configuration table to retrieve the CongestionFactor of 4b. The CongestionFactor configuration table is 18 rows by 8 columns.

- The rows are indexed by (SetMSB - 10). This translates congestion to 1KB units: 1KB, 2KB, 4KB ..256MB.
- The columns are indexed by ChangeMeasure when the change measure value is below 7, else it is set to 7. Because change of interest is when greater than 1/127th of the current congestion while lower changes (i.e. 1/256) are considered as normal (Standard) congestion updates indicating gradual change. These Standard updates (when ChangeMeasure is below 7) use a pre-configured probability for generating update message events.

The congestion updates are carried in messages of a configurable size. The message size determines the usage of NPS EZnet message busses.

10.10.1.1 Update Rate Limiting of Probabilistic Reports

The update rates are monitored by dual rate shapers. Each TM level: TM-L0 to TM-L4 and TM-global has a dedicated shaper. Each shaper limits the message generation rate in message units 0-255 per configurable period of 16b core-clocks.

The dual rate shaper is configured with: Commit rate, Excess rate, Commit burst size, Excess burst size. The burst sizes are configured in x16 messages units 0-255.

The dual-shaper status results as:

- Red - Excess rate not conforming
- Yellow - Excess rate conforming and Commit rate non-conforming
- Green - Commit rate conforming.

The dual shaper status determines the probability (rate) to generate update messages:

- When Red, configured PROB_RED is used for update generation
- When Yellow, configured PROB_YELLOW is used for update generation.
- When Green, congestion change based probabilistic method is used.

10.10.2 Congestion Segmentation Method

The congestion can be monitored for packet or TM buffers occupancy. The TM entity congestion is compared to a scale of 16 constant size segments, where is 16*segment_size is the scale's absolute maximum for congestion comparison.

It is advised to set the scale maximum value below WRED min_th. This ensures accurate monitoring congestion in a congestion region that is not affected by WRED drops.

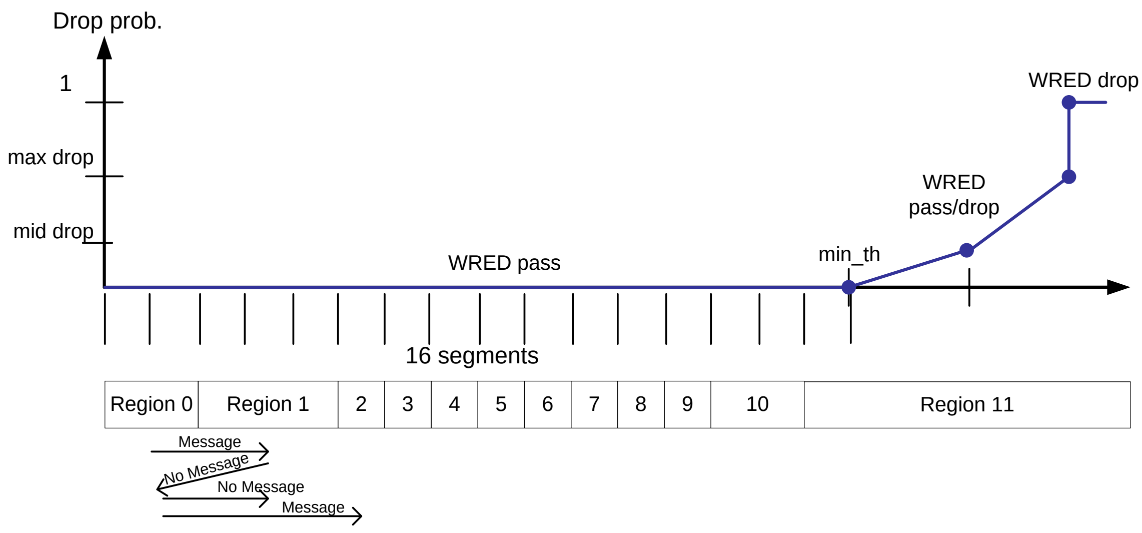
Each TM hierarchical level has a number of profiles to set the segment size per level entities.

The segment size is represented as 10b value and exponent of the multiplier - $2^{EXP[4:0]}$.

The profile is selected in conjunction with WRED parameters:

- TM L4 profile configuration is shared.
- TM L3 profile configuration is shared.
- TM L2 profile, out of 256, is indexed by entity WRED L3 profile index.
- TM L1 profile, out of 32, is indexed by entity WRED L1 profile index.
- TM L0 profile, out of 32, is indexed by entity WRED L0 profile index.
- The TM global congestion is monitored by global segment profile.

Figure 9-28. Segmented TM congestion monitoring



The segmented congestion monitoring can be renumbered to Regions 0-15 by ascending numbering. This renumbering aligns reports of different entities to the same scale. For example, regions of {0,0,0,0,0,0,0, 15,15,15,15,15,15,15,15} would be useful for small queues with crude segment distinction.

The region renumbering is set by 4 profiles per TM L2/L1/L0 hierarchy level. The table is selected per segment size profile. The TM L4/L3 and Global congestion regions have a single profile.

Congestion monitoring results that cross a region boundary in either direction (up or down) may generate a message indicating the queue occupancy has exceeded its threshold or dropped below its threshold value. The message generation is subject to low pass filtering, which suppresses message generation when the same region boundary is crossed subsequent times. Messages generated may also be assigned one of two priority levels associated with the direction of the boundary crossing. These priorities are used for internal message dropping when message bandwidth is limited. For example, for the first cross from Region 0 to Region 1 a message will be generated. After dropping to Region 0 and re-crossing to Region 1, a message will not be generated to save message bandwidth. A message will be generated when crossing to Region 2.

Messages can also be directed to the statistics block in the PSR feature.

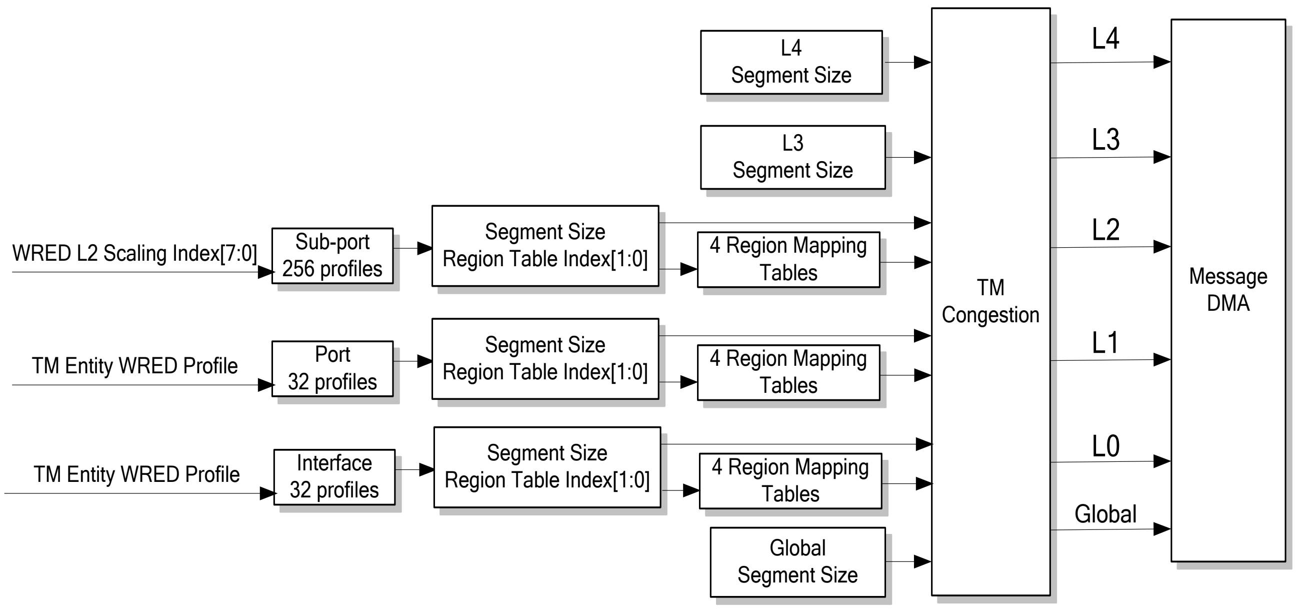
In order to prevent fluctuating occupancy levels from generating an excessive number of messages and overloading the system, an occupancy message will not be issued when crossing the same region boundary unless another boundary has been crossed, and/or unless another boundary has been previously crossed in the same direction.

Single TM event of packet scheduling or WRED drop may potentially generate up to 12 TM congestion report messages (6 on entrance, 6 on exit), one for each of the TM levels plus global congestion.

Per level TM events can be enabled/disabled for message reports: WRED drops, TM scheduling.

In general, the TM Congestion Report messaging feature is enabled per hierarchical level, and may be changed during run time. In addition, messaging may be disabled per TM sub-tree in both the Port and Interface levels.

Figure 9-29. TM congestion report messaging



Even if a status update message was not generated, the queue occupancy is stored and may be used to derive the 4-bit congestion status.

In addition, a self-generated periodic update mechanism (non-event generated) runs through each entity in each level to collect and forward queue status periodically.

10.11 TM Loopback and Multicast Handling

The NPS-400 has a dedicated loopback path that allows delivering packets from each of the TM engines back towards the CTOPs. Due to the fact the external frame memory handled by the TM is much larger than the internal frame memory reserved for the CTOPs, the most important usage of the TM loopback is mitigating bursts that consume significant frame memory, allowing the CTOPs to spread packet handling over time. The most common examples are multicast and long term deferred processing.

Any packet delivered from the CTOPs to the TM is associated with a destination tag (PSID). The PSID typically defines the output interface (or output channel, for channelized interfaces) that the packet will be transmitted through when scheduled by the TM. The PSID can also define some auxiliary destinations inside the chip. Eight loopback ports are available per PMU (total of 16 for both PMUs) and each flow queue can use the PSID[2:0] as modifiers to select any of them.

For the purpose of multicast, each loopback port can be configured independently to operate in one of these modes:

- Unicast – In this mode the loopback port does not support multicasting (and does not support differentiating between replicas of the same packet transmitted by different replay queues).
- Multicast_TM – TxNDMA performs multicast, generating multiple copies of the packet descriptor to the PMU/CTOP SW. The SW prepares a unicast frame destined to the multicast port, sets the replica number. The TM directs the multicast frame to the Multicast replicator which pushes N+1 jobs to the PMU to the selected PMU queue. The SW in the CTOPs gets each frame, duplicates its header BD and LBD (in each of multi-buffer frame) and sends the frame to the Tx port through the selected TM flow. The Multicast Control bit (MC) marks the frame as a multicast frame and controls the buffer release mechanism. Each frame buffer has an associated MC counter, which is read and decremented atomically by the buffer release mechanism. When the MC gets to zero, it is not decremented and the release logic recycles the buffer. When all LBD buffers are recycled then the Link Buffer Descriptor (LBD) is also recycled.
- Lightweight multicast through job replication and CTOP SW processing.

11. Statistics Management

11.1 Statistics Overview

The NPS-400 implements statistics management using either internal SRAM or EMEM with DRAM as storage in order to minimize the cost of the memory subsystem.

The NPS-400 offers three modes of operation for the statistics based on the expected usage of the counters: on-demand, posted and atomic. The **on-demand** mode is intended for counters that are expected to be written to and also read by the data plane packet-processing code and need to be coherent. The **posted** mode provides a “fire and forget” optimized mode of operation for updating counters and is intended for counters that are expected to be updated by the data plane application and read (collect statistics) by the control plane applications. It is expected that most of the counters that are written to by the data plane SW and read by the control plane SW, such as standard MIB counters, can make use of the posted mode of operation. Additionally, the CTOPs support native **atomic** operations to any address in IMEM or EMEM for implementing SW defined counters, semaphores and bitwise manipulations. The selection of mode to use for updating a counter is done by the code when it specifies the type of statistics operation to be executed.

The statistics management is implemented by dedicated statistics units that perform statistics entity initialization, update, periodic scans for updating and reporting, and message queues for statistics collection.

- Eight on-demand (OD) statistics units: four on the north side and four on the south side.
  The OD statistics is managed in EMEM and also in a dedicated statistics SRAM of 8*128Kb.
- Four posted statistics units: two on the north side and two on the south side.

The application can implement dynamic allocation, recycle and auto association between counters and network flows.

The statistic units serve several clients:

- CTOPs use statistics for counting (including multicast counters), token buckets and watchdog.
- PCIe uses statistics by emulation of CTOP statistics commands via configuration registers.
- TM uses statistics to count TM WRED, scheduling events and periodic updates.
- TxNDMA uses statistics to update the packet buffer associated multicast counter upon transmit events.

In addition to statistics management by OD statistics units, the CTOP may SW manage statistics in IMEM and EMEM via dedicated EZ-ISA. The statistics command is addressed to an OD unit, posted unit, or directly to IMEM or EMEM via MSID mapping.

Table 11-1. Memory for statistics

STATISTICS OPERATION	*MEMORY*		
	ON-DEMAND INTERNAL SRAM	*IMEM*	*EMEM*
On-demand (OD) unit command HW managed	All commands		All commands
Posted unit command HW managed			All commands
SW direct command (only inc./dec. counters)		Write, Read, Add	
Packet buffer multicast counter		For IMEM buffer. SW managed	For EMEM buffer. OD unit managed.

The statistics units support all available commands while direct commands to memory support only: write, read and atomic add signed value.

While statistics units monitor the counters and generate maintenance messages to message queues, the SW managed statistics are not scanned by the HW mechanism.

The SW managed counters do not support token bucket nor bitwise counters.

The statistics architecture (Figure 11-1) comprised of

- North: Posted-0, Posted-1, OD-0, OD-1, OD-2, OD-3.
- South: Posted-0, Posted-1, OD-0, OD-1, OD-2, OD-3.

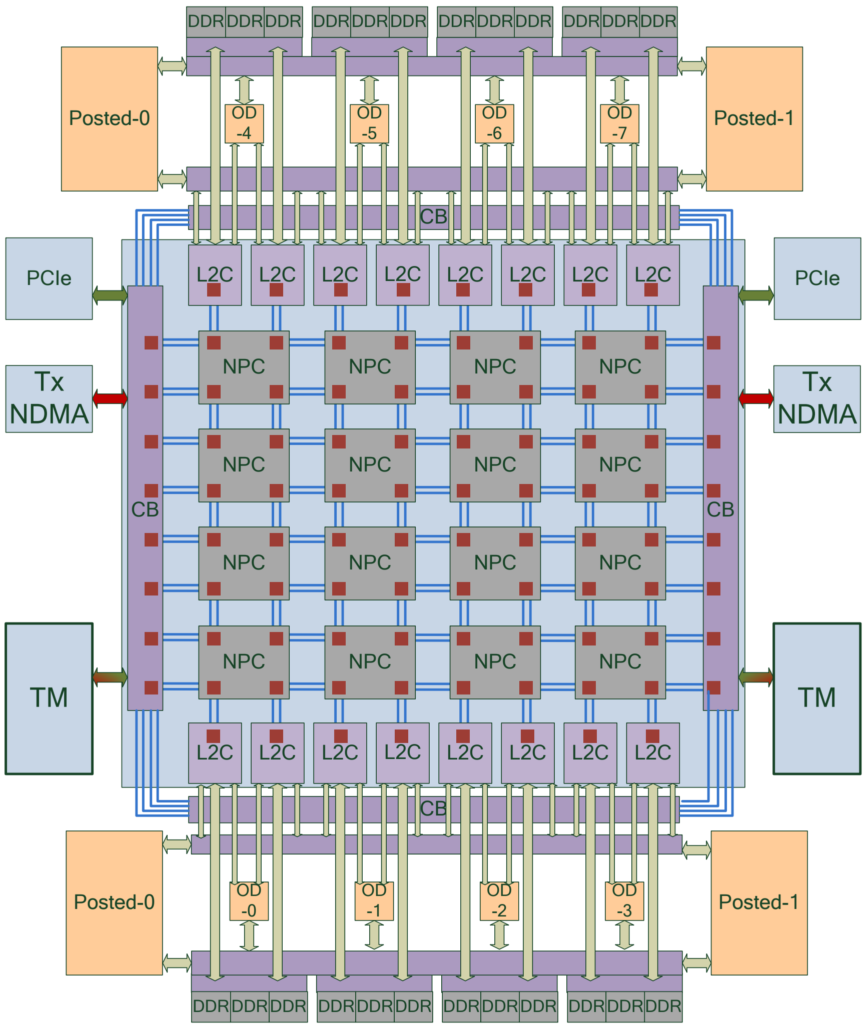
The L2C units serve as terminal points for EZnet messages to the statistics units. Two L2C units serve a single OD statistics unit. All L2C units serve both posted units on same side.

The OD and posted units can access all DRAM devices on same side. The system configuration guarantees statistics bandwidth load balancing to statistics units and DRAM devices and banks.

The OD units operate at 0.5*core clock rate.

The posted units operate at TM clock rate (which is lower than the core clock).

Figure 11-1. NPS-400 statistics architecture



11.1.1 On-demand Counter Overview

On-demand statistics counters are updated and collected by the data plane application. All supported counter types can be configured as “on-demand”. Each counter update requires a read-modify-write transaction and the update operation is atomic. The on-demand counters support all applicable commands per counter type.

The OD-unit counters support embedded ECC soft error protection.

Each one of eight OD units embeds 128Kb SRAM that can be partitioned for two purposes:

- Implement any type of OD counter. This is useful for high update/access rate counters.
- Hold optimized token bucket approximated status (the color). This is useful to achieve fast response to TB color inquiries.

The OD units provide a statistics cache memory. Both Statistics SRAM and EMEM statistics requests are directed through the cache. An application using a small set of flows will benefit from the flow associated counters migrating to the cache, thereby reducing the latency to access them. The cache is transparent to the programmer and does not require any management.

In addition, the prefetch statistics operation helps hide the latency of accessing the statistics counters by the CTOP data plane application. Counters can be prefetched at any point, which causes the counters to be loaded into the cache, and then accessed later with reduced latency. Prefetching can be performed on any counter type.

11.1.1.1 On-demand Statistics Features

- Statistics are updated and collected by the data plane application
- Eight on-demand units; each unit supports up to 64M counters.
  - Counters can reside in the unit’s dedicated 16Kbyte internal SRAM memory or in external DRAM.
  - Total of up to 512M on-demand counters (8 units * 64M counters); 256M of these are allocated for EMEM buffer multicast counters.
  - Counters have load balanced allocation to DRAM devices and banks.
- OD unit peak performance of commands is per two core clocks. Sustained performance of commands is per three core clocks.
- The OD statistics serve per-flow statistics, traffic metering and policing shaper counters.
  - Long and Double counters
  - OAM watchdog counters
  - Bitwise counters – enhanced command set, including semaphore commands of bitwise counters
  - Token Bucket counters:
    - Variations of Token Bucket (TB) counters include: TB, Optimized TB, Hierarchical TB, Optimized Hierarchical TB.
    - Auto implementation of per-flow token bucket and dual token bucket policers (srTCM, trTCM or MEF5)
    - Sophisticated Hierarchical Token Bucket (hierarchically coupled policers) scheme
    - Reduced latency response by Optimized Token Bucket color status caching.
- On-demand counter application pre-fetch and caching
- Programmable threshold settings and threshold exceeded notification.
- Per OD unit message queue for monitor notifications.
- Shadow counters with synchronized swapping

The statistics units support the following features:

- ECC protected statistics embedded in data structure.

11.1.2 Posted Counter Overview

Most of the statistics counter updates are simple MIB counters where the application is keeping count of number of bytes, or packets for a given entity. This type of counter does not require a strong consistency, as required by the on-demand counters, enabling a “fire and forget” mode of operation.

Posted statistics counters are updated by the data plane application while collected by the control plane application at lower rates. The posted statistics are maintained in DRAM. The statistics memory access pattern is random access while locations accessed with read-write. Since DRAM demonstrates poor performance when operated with random access of short transaction, the Posted counter mechanism overcomes this DRAM performance shortage.

With posted units, the statistics commands are accumulated in buffer structures in the DRAM. The accumulated commands are then fetched from the DRAM memory in large chunks to better utilize the memory bandwidth, and updated by performing a long burst of a 64-byte line.

The goal of the posted statistics is to minimize the DRAM bandwidth per update. This is accomplished by the following features:

- Accumulation of four commands per counter on average. This enables a set of four commands to perform a single initial read of the counter and then four successive write operations.
- Minimization of the storage required for common commands (3 bytes for an 8-bit value) inside the command buffer.
- Using four 16-bit chunks per counter, where usually only the LSB chunk is read and written.

11.1.2.1 Posted Statistics Features

- Statistics are updated by data plane application and collected by control plane application
- Four posted units (two north, two south).
- Posted unit supports two memory spaces:
  - Standard space - Used for single-aggregation-hierarchy posted counters
  - Optimized space - Used for dual-aggregation-hierarchy posted counters
- Each posted unit supports
  - Up to 64M indirect counters per unit. Total for all units up to approximately 256M “indirect" posted counters.
  - Up to 2M direct counters per unit. Total for all units up to 8M “direct" posted counters.
- Each posted unit supports peak performance of update command per posted unit clock (equal to TM core clock).
- Supported counters types: Long counter (64b) and Double counter (two counters sharing a 128b structure, i.e. frame and byte counters).
- Programmable threshold settings and threshold exceeded notification
- Per Posted unit message queue for monitor notifications.

The statistics units support the following features:

- ECC protected statistics embedded in data structure.

11.2 On-demand Counters

On-demand statistics counters are updated and collected by the application.

Each counter update requires a read-modify-write transaction. The update is an atomic operation.

The on-demand counters support all available commands.

The on-demand counters support ECC soft error protection.

Statistics groups are defined and associated with the statistics counters and profiles (where relevant). Possible group types (Table 11-2):

- Standard group – Each counter assigned to a standard group can be of a separate counter type:
  - Long counters – 59 bit counter value.
  - Double counters – Support byte and frame accounting in a single counter. Each double counter supports 96 data bits, with byte and frame portions.
  - Bitwise counters – Support bitwise operations on a subset of bits in the counters. This can be used to implement semaphores, state machines or lookup structures with small results.
  - Watchdog counters – Monitor the number of event in a period of time.
  - Token Bucket/DiffServ counters – Used for measuring traffic rates and marking packets using one of several protocols.
- Optimized Token Bucket group – Functionally equivalent to Token Bucket/DiffServ counters but with the color bits (2 or 4 bits) stored within the dedicated internal statistics memory for improved response time, and the actual buckets are stored separately in EMEM.
  - Only Token Bucket type counters may be assigned to this group which will in effect make them Optimized Token Buckets.
- Hierarchical Token Bucket group – A multi token bucket structure where several counters may be updated by each token bucket command.
  - Only Hierarchical Token Bucket type counters may be assigned to this group.
- Optimized Hierarchical Token Bucket group – Functionally equivalent to a Hierarchical Token Buckets but with the color bits stored within the dedicated internal statistics memory for improved response time.
  - Only Hierarchical Token Bucket type counters may be assigned to this group which will in effect make them Optimized Hierarchical Token Buckets.

The statistics counter space is divided into groups by a programmable range register. There are eight on-demand Statistics units with each supporting up to eight groups. Each group is configured to support counter type per line of 16B.

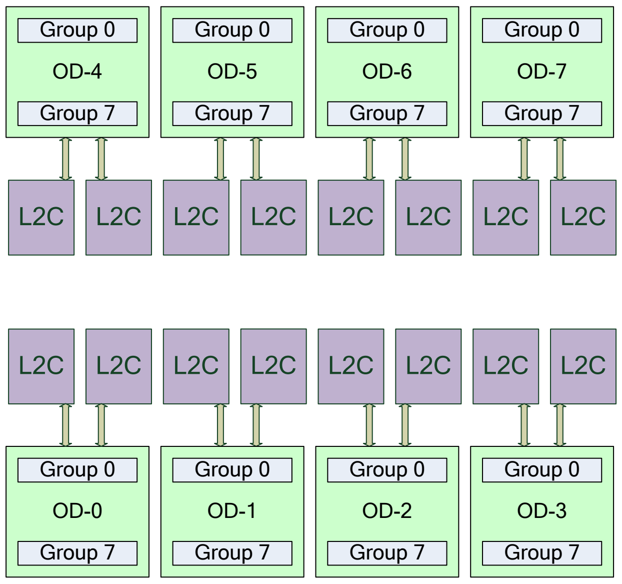
The following internal mechanisms access the on-demand counters:

- Long and Double counters scan mechanism (garbage collection mechanism) for monitor message generation.
- Internal refresh mechanism for policers token rate update

11.2.1 On-demand Counter Architecture

11.2.1.1 On-demand Counter Addressing

The on-demand statistic functionality is provided by eight OD statistics units. Each unit supports up to eight OD groups.



Each group needs to be assigned a start counter and number of counters within the partition's logical address range. Counter groups do not need to be consecutive.

The counters are arranged in 16B memory lines. Each line contains a single counter.

Each group needs to be assigned one of the group types (Table 11-2).

Table 11-2. Matrix of statistics on-demand counter groups and types supported

	ON-DEMAND STATISTICS GROUP TYPES			
COUNTER TYPES	***STANDARD***	***OPTIMIZED TOKEN BUCKET***	***HIERARCHICAL TOKEN BUCKET***	***OPTIMIZED HIERARCHICAL TOKEN BUCKET***
Long counter	Supported	--	--	--
Double counter	Supported	--	--	--
Token Bucket	Supported	Supported	--	--
Hierarchical Token Bucket	--	--	Supported	Supported
Watchdog counter	Supported	--	--	--
Bitwise counter	Supported	--	--	--
General purpose counter	Supported	--	--	--

Configuration dictates the memory interface in which the group will reside:

- EMEM - The counters are ECC/Parity protected by embedded ECC/parity bit in data entry.
- INTERNAL_SRAM – dedicated SRAM. Each on-demand unit has a dedicated INTERNAL_SRAM of 2K lines x 128 bits + ECC bits, unlike EMEM that are shared by all the units. Each line in the dedicated INTERNAL_SRAM can be a counter, or Optimized Token Bucket colors.

For each counter group, the following parameters are defined:

- Group type – The type of the group (Standard, Optimized Token Bucket, Hierarchical Token Bucket, Optimized Hierarchical Token Bucket).
- Memory type – The type of memory in which these counters are stored (dedicated internal SRAM, EMEM).
- Start counter – The first counter in the group.
- Num counters – The number of counters in the group.
- Enable shadow counters – supported only for standard counter groups that contain only long and/or double counters.

The SW structure for OD statistics configuration aggregates counters to partitions. The OD unit is allocated to partition. The OD counter groups of all OD units allocated to one partition are configured the same. The partition is addressed by MSID configuration that load balance OD units of the partition.

The partition can be balanced on 1, 2, 4 or 8 OD units. The balancing is done over L2C units, where the two L2C units share an OD unit.

The MSID configuration for OD statistics load balancing is:

- Side balance:
  - North only (OD-4 : OD-7),
  - South only (OD-0 : OD-3),
  - Both North and South sides (OD-0 : OD-7).
- Number of L2C units on each side used for load balance: 1, 2, 4, 8
- Offset of L2C unit: 0-7.

The load balancing is according to counter address 21:7:

- LB hash [3:0] = Hash (Address[21:11])
- L2C select [3:0]= LB hash[3:0] xor Address[10:7]

The transaction is directed to Side and L2C per side:

- Side = L2C select[0]
- L2C per side[2:0]:

Number of L2C units on each side:	2	4	8
Balance on single side	Offset[3:0] + L2C select[0]	Offset[3:0] + L2C select[1:0]	Offset[3:0] + L2C select[2:0]
Balance on both sides	Offset[3:0] + L2C select[1]	Offset[3:0] + L2C select[2:1]	Offset[3:0] + L2C select[3:1]

- Counter index:

Number of L2C units on each side:	2	4	8
Balance on single side	Address[25:0]	Address[25:8][6:0]	Address[25:9][6:0]
Balance on both sides	Address[25:8][6:0]	Address[25:9][6:0]	Address[25:10][6:0]

11.2.1.2 Counter Pre-fetch and Caching

The counters support a speculative, non-blocking, pre-fetch command for any on-demand counter in EMEM. The pre-fetch issued by the application to shorten the latency of counter read. The pre-fetch can be triggered at any point in the CTOP SW execution prior to the actual counter usage. The pre-fetch of a counter value is to OD-unit cache. The caching is performed for application pre-fetched and regular access. The cache serves internal and DRAM counters.

11.2.1.3 On-demand Counter Commands

The table below summarizes the relevance of each statistics operation for each counter type.

Table 11-3. NPS statistics on-demand operations summary

Operation	Operand	SW Statistics in IMEM	Long	Double	Bitwise	All Token Bucket types	Watchdog	Reply	Response	Message on Long/Double
Init counter LSB Init counter MSB	64b	Yes	Yes	Yes	Yes	Yes	Yes	No	--	--
Read + Init counter LSB Read + Init counter MSB	64b	Yes				--		Yes	64b	--
Read counter	--	Yes	Yes	Yes	Yes	--	Yes	Yes	64b	--
Reset counter Reset counter MSB	--	--	Yes	Yes	Yes	--	--	No	--	--
Read + Reset counter Read + Reset counter MSB	--	--						Yes	64b	--
Add signed value to counter	16b	Yes	Yes	Yes	Yes	--	--	No	--	Yes
Read + Add signed	16b	Yes						Yes	64b	Yes
Increment counter by 1	32b	--	Yes	Yes	Yes	--	Yes	No	--	Yes
Read + Increment by 1	32b	Yes						Yes	64b	Yes
Decrement counter by 1	32b	--	Yes	Yes	Yes	--	--	No	--	--
Read + decrement by 1	32b	--						Yes	64b	--
Multiply two values and accumulate	32b (16b,16b)	--	Yes	--	--	--	--	No	--	--
Read + Multiply two values and accumulate	32b (16b,16b)	--						Yes	64b	--
Conditional decrement	32b (16b value)	--	Yes	--	--	--	--	No	--	--
Conditional decrement + read	16b	Yes						Yes	64b	--
Increment two values	2 x 16b	Yes	--	Yes	--	--	--	No	--	Yes
Read + Increment two values	32b	--						Yes	64b	Yes
Get color and update TB Read + Get color + Update	16b	--	--	--	--	Yes	--	Yes	32b	--
Check color Read + Check color	16b	--	--	--	--	Yes	--	Yes	32b	--
Force Add/ subtract tokens (with/without read)	32b	--				Yes		Yes	32b	--
Bitwise set	32b	--	--	--	Yes	--	--	No	--	--
Read + Bitwise set	32b							Yes	16b	--
Bitwise clear	32b	--	--	--	Yes	--	--	No	--	--
Read + Bitwise clear	32b							Yes	16b	--
Bitwise write	32b	--	--	--	Yes	--	--	No	--	--
Read + Bitwise write	32b							Yes	16b	--
Bitwise read	--	--	--	--	Yes	--	--	Yes	16b	--
Bitwise conditional set	32b	--	--	--	Yes	--	--	No	--	--
Read + Bitwise conditional set	32b							Yes	16b	--
Pre-fetch	--	--	Yes	Yes	Yes	Yes	Yes	No	--	--
Write entry LSB Write entry MSB	64b	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	--
Read entry LSB Read entry MSB	--	Yes	Yes	Yes	Yes	Yes	Yes	Yes	64b	--

11.2.1.4 On-demand Counter Periodic Scan

The OD-statistics has 16 general purpose machines per OD unit that scan counters for refreshing and garbage collection, which may result in message generation.

The refresh feature periodically updates token buckets that are not accessed by the CTOPs.

11.2.1.4.1 Periodic Scan (Garbage Collection) Mechanism

The machines periodically scan long and double counters.

When the machines scan the counter, they reset the counter value, and generate a message to a message queue indicating the counter value before the reset. This can be used to periodically export the statistic counter data to the CTOPs (via the message FIFO queues).

The counters periodic scan compares the counter value to configurable threshold. When the counter is above the threshold the message is generated and counter is zeroed. The configured threshold is used for long counter value and double counter event-counter and byte-value threshold.

There are two thresholds for byte-value and event-value:

- Low value thresholds– Result in a potentially high message rate.
- High value thresholds – Result in a potentially lower message rate.

A configurable number of scanned counters are compared to high-value set of thresholds and then next counter is compared to low value set of thresholds. This method allows controlling the message generation rate while not missing counter exceeding high value threshold. After scanning all counters in OD group range the scan machine assigns the low value threshold to next counter compare, thus each counter has chance to be compared with low value threshold.

The number of high thresholds to scan before using low threshold is configurable in range 0-255.

Example: 4 High + 1 Low configuration

SCAN ORDER:	*COUNTERS IN RANGE*	*SCAN CYCLE-0*	*SCAN CYCLE-1*	*SCAN CYCLE-3*
		Thresholds:	**Thresholds:**	**Thresholds:**
1	Long	Low	High	High
-	OAM Watchdog			
2	Double	High/High	Low	High
3	Long	High	High/High	Low
4	Long	High	High	High
5	Double	Low/Low	High	High
6	Long	High	Low	High

The thresholds are configured via 6b value that selects the counter bit index.

HIGH 6b	High threshold value. For message generation.	6 bit indication of the counter bit (LOG): counter value >= 2^ HIGH HIGH: 0-63.
LOW 6b	High threshold value. For message generation.	6 bit indication of the counter bit (LOG): counter value >= 2^ LOW LOW: 0-63

11.2.1.5 On-demand Counter Messaging

On-demand messages are sent to message queues according to user configuration.

Messaging for long/double counters:

- Counter Update Message - periodically generated by scan (garbage collection) machines.
- Counter Exceeded Message - when changing of a counter exceeds specified threshold. This serves as protection to prevent counter overflow.

The periodic scan (garbage collection) mechanism may reset the counter and send a message with the counter value via the message queue.

11.2.1.6 On-demand Message FIFOs

Each OD engine is equipped with a FIFO message queue in EMEM, totaling 8 FIFOs. The message FIFOs act as a messaging channel from OD units to SW control thread. Each message consumes 16B in memory.

The message queue is maintained by a Read pointer and a Write pointer advancing cyclically in EMEM memory region. The queue size is configurable.

The OD unit writes the massage to EMEM and increments the Write pointer.

The SW reads messages and increment cyclically the Read pointer.

The Read and Write pointers are located in EMEM configured location.

The OD unit has a configurable time interval for reading the Read pointer from EMEM.

The OD unit has a configurable time interval for writing the Write pointer to EMEM.

The CTOP has dedicated MSID to access message pointers and data in EMEM.

The designated SW thread is responsible to read the messages and advance the Read pointer.

The SW thread should be activated periodically. The SW thread reads the queue Read/Write pointers from EMEM to identify number of queued messages. The SW thread process the messages and updates the Read pointer in EMEM.

When message queue is full the OD unit stalls message generation. While messages cannot be generated the counters are not zeroed even if non-compliant per monitoring thresholds.

11.2.2 On-demand Counter Types

11.2.2.1 Long Counter Type

Long counters count events and hold a value of up to 59 bits + an overflow bit.

The Long counter type can be specified per 16B line when counter group configured for Standard counter type. The counter value and overflow bit is reset when OAM message is generated on counter threshold exceed or counter scan.

The 16B counter entry contains:

- Counter type = Long counter
- Counter value 59 bits. This value is updated by commands and replied in read response.
- Overflow bit. Set when update rollover maximal/minimal value.
- ECC protection of 8b.
- Configurations:
  - Enable message generation on increment with threshold exceed.
  - Counter exceed threshold of 6 bits (value range 0 - 58). When counter exceeds threshold, an OAM message is generated and queued to OD units message FIFO.
    - The value 0 - Any counter update generate a message.
    - The value 1 - Any counter update to value 2 and above.
      ..
    - The value 10 - Any counter update to value 1K and above.
      ..
    - The value 58 - Any counter update that sets the MSB bit 59.

The supported commands are:

OPERATION	***DESCRIPTION***
Init counter / Read + init counter	Initialize a counter value, with or without read. The Overflow bit is reset.
Read counter	Read a counter.
Reset counter / Read + reset counter	Reset a counter value, with or without read.
Add signed to counter Read + Add signed to counter	Each counter can be incremented or decremented, by value up to 16 bits, with or without read. Increment may trigger message generation due to threshold exceed.
Increment counter by 1 / Read + increment by 1 Decrement counter by 1 / Read + decrement by 1	Each counter can be incremented or decremented by 1, with or without read. Increment by 1 may trigger message generation due to threshold exceed.
Conditional decrement / Conditional decrement + read	If value after decrement is negative do not decrement, and optionally returns the success or failure of the operation.
Multiply and accumulate / Multiply and accumulate + read	This command may be used as a calculator to obtain the product of 2 operands (each up to 16 bits). Alternatively it can also increment the counter with this product.
Write counter LSB Write counter MSB	Counter initial configuration of parameters.

11.2.2.2 Double Counter Type

Double counters pack two counters in single 128b entry that can be updated simultaneously by same command. Double counter is useful for managing both byte count and frame count with the same counter.

The double counter allocates 96 bits to hold both values of the embedded counters. An additional bit allocated for the Overflow. The 96 bits are flexibly partitioned to two counters (byte counter, event counter): (63, 33), (62, 34) ...(48,48).

The double counter type can be specified per 16B line when the counter group configured for Standard counter type. The both the counter's value and its overflow bit are reset when an OAM message is generated on counter threshold exceed or counter scan.

The 16B counter entry contains:

- Counter type = Double counter
- Counter value 96 bits. This value is updated by commands and replied in read response.
- Overflow bit. Set when update rollover maximal/minimal value on either embedded counter.
- ECC protection of 8b.
- Configurations:
  - Double Counter value field partition. Sixteen possible partitions supported:
    0 - (63, 33), 1- (62, 34) ...15- (48,48).
  - Enable message generation on increment with threshold exceed.
  - When counter exceeds threshold, an OAM message is generated and queued to OD units message FIFO. Each embedded counter has a counter exceed threshold of 6 bits.
  - Byte counter threshold 6b (value range 0 - 62). Event counter threshold 6b (value range 0 - 47).
    The value 0 - Any counter update generate a message.
    The value 1 - Any counter update to value 2 and above.
    ...
    The value 10 - Any counter update to value 1K and above.
    ...
    The value 62 - Any counter update that sets the bit 63.

The supported commands are:

OPERATION	*DESCRIPTION*
Init counter / Read + init counter	Initialize a counter value, with or without read. The Overflow bit is reset.
Read counter	Read a counter.
Reset counter / Read + reset counter	Reset a counter value, with or without read.
Add signed to counter Read + Add signed to counter	Each counter can be incremented or decremented, by value up to 16 bits, with or without read. Increment may trigger message generation due to threshold exceed.
Increment counter by 1 / Read + increment by 1 Decrement counter by 1 / Read + decrement by 1	Each counter can be incremented or decremented by 1, with or without read. Increment by 1 may trigger message generation due to threshold exceed.
Increment two values / Read + increment two values	Increment both the byte count and event count by 16 bit values. May trigger message generation due to threshold exceed.
Write counter LSB Write counter MSB	Counter initial configuration of parameters.

Note that OD double counters and posted double counters should not be interchanged nor confused.

11.2.2.3 Bitwise Counter Type

The bitwise counter occupies 64 bits. The counter may be accessed in a resolution of 1, 2, 4, 8 or 16 bits from an offset specified by command operand. These counters can be used as semaphores.

The bitwise counter type can be specified per 16B line when the counter group configured for Standard counter type.

The bitwise counters are not monitored by scanning and do not generate messages.

The 16B counter entry contains:

- Counter type = Bitwise counter
- Counter value 64 bits. This value is updated by commands and replied in read response.

The command operands are:

- Size (3 bits) – Selects the number of bits for the operation: 1, 2, 4, 8 or 16 bits.
- Offset (6 bits) – Determines the bit offset in the counter for the operation. Must be a multiple of the Size.
- Value (16 bits max.) – Value for Bitwise Write command, or mask for the Bitwise Set and Bitwise Clear commands. The Value holds Compare Value (4 bits) and Set Value (4 bits) for Conditional Set commands.

The supported commands are:

OPERATION	DESCRIPTION
Init counter / Read + init counter	Initialize a counter (64 bits), with or without read.
Read counter	Read a counter (64 bits).
Reset counter / Read + reset counter	Reset a counter (64 bits), with or without read.
Bitwise Set / Read + bitwise set	Perform OR with operand Value to the selected bits, based on the Size. Optionally returns the bits before the operation.
Bitwise Clear / Read + bitwise clear	Perform AND with (NOT(Value)) to the selected bits, based on the Size. Optionally returns the bits before the operation.
Bitwise Write/ Read + bitwise write	Write the operand Value to the Counter selected bits, based on the Size. Optionally returns the bits before the operation.
Bitwise Read	Read a counter, based on the Size.
Bitwise Conditional Set / Read + bitwise Conditional Set	Checks if the Size/Offset selected bits of the counter equal to the operand Compare Value. If so, it replaces the bits in the counter with Set Value, and optionally reads the old value. This command may be used for a state machine implementation.
Bitwise Increment counter / Read + increment counter Bitwise Decrement counter / Read + decrement counter	The selected bits can be incremented or decremented, by value up to 16 bits, with or without read.
Bitwise Increment counter by 1 / Bitwise Read + increment by 1 Decrement counter by 1 / Read + decrement by 1	The selected bits can be incremented or decremented by 1, with or without read.

11.2.2.4 Token Bucket Type

Dual Token Bucket calculations and coloring (i.e. green, yellow, red) is hardware implemented by the OD units. The Token Bucket type can be specified per 16B line when the counter group is configured for Standard counter type. Each dual TB occupies a 16B entry.

Each Token Bucket counter is assigned with two associated profiles that define the parameters for the Token Bucket algorithms used. One profile defines commit settings (CIR, CBS) and the other profile configures the excess settings (PIR, PBS). Each OD unit supports 1024 profiles for commit settings and 1024 profiles for excess settings.

The metering color encoding defines the encoding used for specifying the three possible metering colors (green, yellow, red) and a user-defined color. The TBs are scanned to accumulate tokens.

The Token Bucket counters are not monitored by scanning and do not generate messages.

The 16B counter entry contains:

- Counter type = Token Bucket
- TB type:
  - Inactive
  - Single Bucket – single bucket counter.
  - srTCM – single rate three color marker
  - trTCM – two rate three color marker.
  - trTCM MEF – two rate three color marker using MEF standard.
  - Color Awareness – either color aware or color blind mode.
  - Coupling flag – determines the coupling flag mode for trTCM MEF algorithms.
- Commit TB bucket 42 bit signed value. The bucket can get negative.
- Excess TB bucket 42 bit signed value. The bucket can get negative.
- Commit TB profile index. Profile is used to set the rates.
- Excess TB profile index. Profile is used to set the rates.

The supported commands are:

OPERATION	DESCRIPTION
Init counter	Initialize a counter. The operands are: Commit and excess TB profile indexes.
Update TB and Get color Read + Update TB and Get color	Get the resulting color after updating a Token Bucket.
Check TB color Read + Check TB color	Check the current color of a Token Bucket, without updating it.
Force Add tokens (with/without read) Force subtract tokens (with/without read)	Force add token or force decrement tokens regardless of TB status color (rate conformance).

The Update, Check and Force command operands are:

- Frame byte size to deduct from TB (or add with Force Add command).
- Frame pre-color for color aware TB.
- The force command targets bucket commit/excess.

11.2.2.4.1 Token Bucket Profiles

Token Bucket profiles are used to define the rate and bucket size parameters for the dual Token Bucket.

- Up to 1024 Commit TB profiles are supported for each OD unit.
  For each profile, the following parameters are defined:
  - Committed Information Rate (CIR) – in bytes per second.
  - Committed Burst Size (CBS) – in bytes.
  - Random threshold on empty bucket profile index to one of four profiles.
- Up to 1024 Excess TB profiles are supported for each OD unit.
  For each profile, the following parameters are defined:
  - Peak Information Rate (PIR) – in bytes per second.
  - Peak Burst Size (PBS) or Excess Burst Size (EBS) – in bytes.
  - Random threshold on empty bucket profile index to one of four profiles.

The rates are defined by a 10 bit mantissa and 6 bit exponent to span the dynamic range.

The Random threshold allows reporting of rate conformance even when the bucket is negative with 50% probability. The bucket allowed negatives is configurable by four profiles.

Each profile can be associated to one or more Token Buckets. Profiles may be associated to Token Bucket counters of all the various types, and counters of varying types may share profiles.

11.2.2.4.2 Token Bucket Refresh

The refresh machines periodically update Token Bucket and Optimized Token Bucket counters that are not frequently accessed by the CTOPs.

The following configurations exist for each refresh machine:

- Enable/disable – determines if the machine is activated.
- Start counter/number of counters – determines the counters scanned by this machine.
- Min/max scan interval – configures the rate at which the machine scans the counters. Both a minimum and maximum time period between each two counters can be configured.
- Intermediate cycles – configures the number of intermediate cycles between main cycles. A value of 0 indicates that there are no intermediate cycles (every scan is a main cycle).
- Reporting thresholds – configures the thresholds for reporting events for the main and intermediate cycles.
- Enable Token Bucket – enables the scan of Token Bucket counters; affects the scan interval.

Token Bucket enabled refresh machines should be configured to scan all Token Bucket counters of all the various types. Using Token Bucket counters that are not refreshed by any machine may cause inaccuracies when Token Bucket counters are not accessed from the application for a long time period.

11.2.2.4.3 Optimized Token Bucket Color Front Ends in SRAM

A standard Token Bucket counter can be allocated to SRAM or the DRAM as configured per group configuration. The Optimized Token Bucket (OPT_TB) enables 2- or 4-bit color status storage in SRAM while the TB is allocated to DRAM. The OPT_TB reduces the latency of retrieving the TB color for incoming packets by simply accessing the stored color status in the on-chip SRAM. The hardware takes care of updating the actual buckets which are stored in the DRAM.

Token Bucket operations are used to implement policers that monitor and limit a flow's rate (bandwidth) and burst size. Rate can be statistically policed, meaning that the short term accuracy of the rate limit can be somewhat low as long as the long term accuracy is high. Burst errors may also be acceptable depending on the actual characteristics of the flow. A burst error of up to 512 bytes should be acceptable for most flows.

This means that the OPT_TB implementation uses a small number of front end bits that hold the approximate token bucket color in low access latency SRAM. These bits issue fast response to color inquiries while accumulating the update and assuring it does not cross the error threshold. While accumulating the updates, the device fetches the back end (the full TB counter) from DRAM memory. When it finally updates the back end, the front end color gets updated with the current color as well. If a command raises the error above the threshold, the fast response path is disabled and the command incurs the entire memory latency. Up to 512K dual token buckets are supported for this optimization. Each one of eight OD units can allocate up to 128Kb SRAM for 64K token bucket frontend color status (2 bits).

The dedicated SRAM memory is partitioned to hold both internal counters and OPT_TB front end colors. Two bits are consumed for each TB color front end.

11.2.2.4.4 Hierarchical Token Bucket

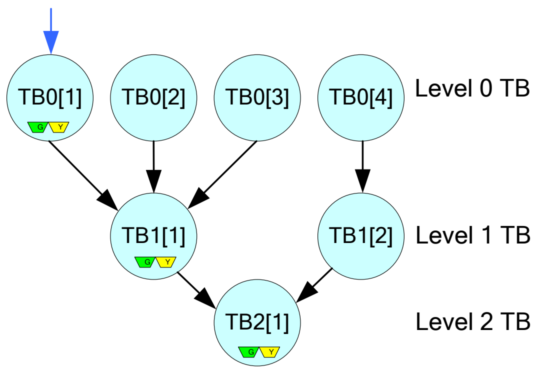
Hierarchical Token Bucket (HIER_TB) is a HW-SW mechanism that allows implementation of advanced metering of traffic using multiple token buckets with a set of pre-configured relationships between them.

HIER_TB can accommodate complex relational structures of TBs and dual TBs. Structures can be associated with the following configurations:

- **Hierarchical** – This refers to a structure where each dual token bucket belongs to a hierarchy. It can either be a parent (higher or aggregating level), a child (lower, linked to parent level) or both. Usually when a packet arrives it is checked against the lowest (child) hierarchy first and only if there are enough tokens in the low hierarchy, the higher hierarchies are checked. This way rates can be assigned to a complex structure (flows per port for example) where both the single (flow) rate is metered and the overall (interface) rate is metered.
- **Class of Service** – This refers to a flat structure where each dual bucket is associated with a class of service. The incoming packet is pre-classified into a specific class and usually when it arrives, only the corresponding class bucket is checked. However, a complex filling procedure can be supported so that excess tokens can be shared among different classes.
- **Mixed** – This refers to a combination of the hierarchical and COS methods, for example the lowest hierarchy can be a COS structure with 3 classes and a parent.

The figure below exemplifies the hierarchical structure of Token Buckets.

Figure 11-2. Token Bucket hierarchical structure



The Hierarchical Token Bucket algorithm consists of two main functions:

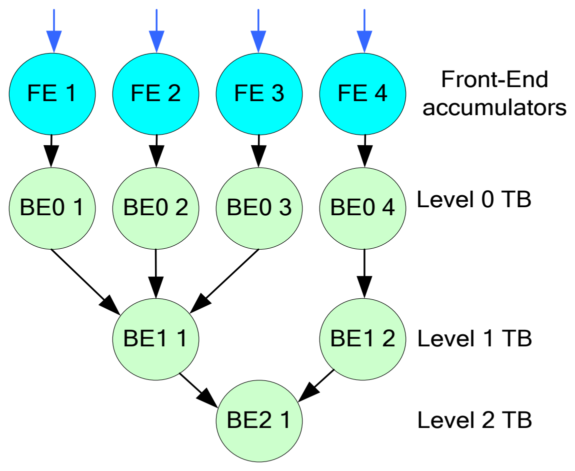
- **Metering function** – Deciding which value/priority to mark the frame with for handling/discarding. Usually this will be done by assigning either a red/green indication or a red/yellow/green indication to the frame. The metering function inputs are the states/colors of the dual TBs in the structure. In the example above: To execute metering for hierarchy structure path starting at TB0[1] the algorithm needs to retrieve dual TB status color of TB0[1], TB1[1], TB2[1].
- **Token update** – Deciding if and how tokens are consumed from TB hierarchy path. The tokens may be shared/coupled between different buckets. Usually tokens can be shared between the overflow tokens of the commit bucket and the excess bucket in the same dual TB, or can be coupled between excess and commit/excess of other dual TBs. In the example above: To execute tokens update for hierarchy structure path starting at TB0[1] the algorithm needs to update dual Token Buckets TB0[1], TB1[1], TB2[1].

A naive implementation approach would require implementing SW semaphore lock on the path starting at TB0[1] while Metering and Update are being executed. This naive implementation would impact application performance when accessing hierarchically structured policers.

The NPS400 assists in implementing hierarchical policers by supporting the Hierarchical Token Bucket counter type that implements a Front-End Accumulators statistics entity.

An HIER_TB contain multiple dual TBs counters to implement the structure (referred to here as Back Ends), and multiple Front End Accumulators, each of which can be SW associated with hierarchy path of BEs Token Buckets. For example in the figure below, each FE is associated with four BEs in higher hierarchy: (FE 1) is associated with (BE0 1), (BE1 1), (BE2 1).

Figure 11-3. Hierarchical Token Bucket hierarchical structure



Key: FE = front end; BE = back end

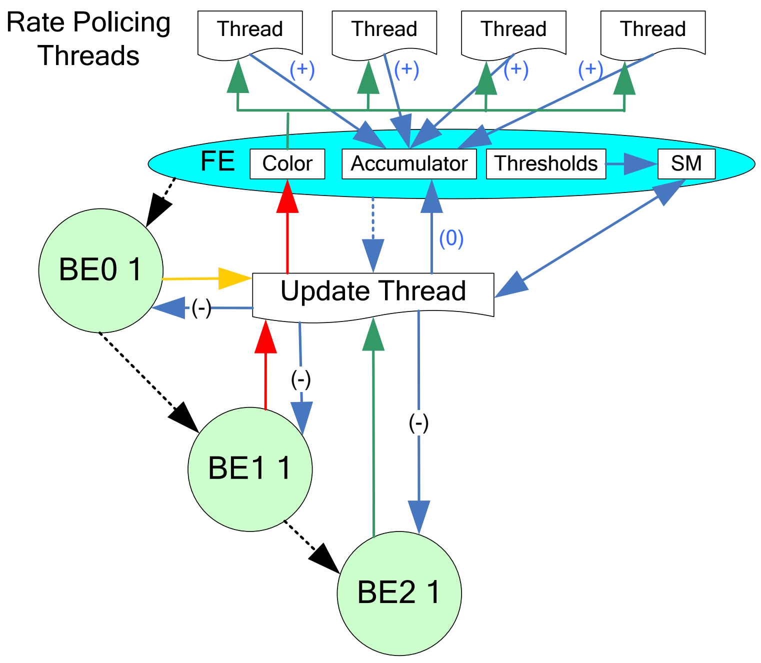
- Front-End Accumulator:
  - The accumulator holds the TB status color that is combined of TB colors along the hierarchy path as was calculated by SW. The rate-policer TB metering function is executed by application examining the status color saved in the accumulator.
  - The accumulator holds the number of tokens (bytes) that need to be subtracted from TBs on the hierarchy path.
  - The accumulator holds a semaphore structure that implements a state machine for path locking in order to allow implementation of atomic update of the TBs on the hierarchy path.
- Back-End counter implements regular dual Token Buckets.
  - The SW application maintains the hierarchy structure and uses TB commands to retrieve status color and update token consumption.
  - The HW periodic scan adds tokens to buckets at a configured rate set by the operation profile.

The Front-End allows a multitude of application threads to perform concurrent TB color check, as required for metering, at a very high rate and with low response latency. The hierarchy path TB update is allowed only to single thread at a time.

Every time a thread performs a metering color check on an FE entity, the FE accumulates the bytes colored by the meter. The metering inaccuracy increases as the accumulator byte counter gets higher because the accumulated bytes did not affect the color.

The FE accumulator byte counter is monitored by a threshold. When an accumulator counter is higher than the threshold, the next thread accessing the FE is designated for TB hierarchy update. The SW update process consumes tokens for the accumulated bytes from the TBs along the path. The new combined color status is calculated based on updated TBs along the path. The color status is registered in the FE accumulator.

Figure 11-4. Front-end accumulator



Key: SM = state machine

An FE accumulator entry is 16B.

Since a token bucket can be pre-color aware, the FE implements accumulator for green and yellow pre-colored bytes.

The thresholds are also duplicated for green and yellow.

Each FE accumulator is monitored by two thresholds:

- Update threshold - Used to trigger update of TBs.
- Fail threshold - Indicates to monitoring threads that the update task's latency is too long and status color is stale. The operation should be re-attempted until colors are updated.

PRE-COLOR	***FE ACCUMULATOR FIELD***	***HW***	***MONITORING THREAD***	***UPDATING THREAD***
	FE state: Active, Update P1, Update P2 (See Figure 11-5. State machine below.)	Switch to 'Update P1' to designate thread for Update task.	Examine FE state to assume Updating thread task.	Signal Update phase end and set state to Active.
Green	Green Pre-colored byte counter 18 bits.	If any accumulator over its threshold, the thread is designated for Updating and new state is Update P1.	Thread assumes Updating state when state is Update P1.	At the end of the Update P1 state the next state is Update P2 if accumulators are not zero.
	Green accumulator update threshold 5 bits (0-17). Select counter LSB to check.			
	Green accumulator fail threshold 5 bits (0-17). Select counter LSB to check			
Yellow	Yellow pre-colored byte counter 18 bits.			
	Yellow accumulator update threshold 5 bits (0-17). Select counter LSB to check.			
	Yellow accumulator fail threshold 5 bits (0-17). Select counter LSB to check			
Sum	Sum accumulator update threshold 5 bits (0-18). Select counter LSB to check.			
	Sum accumulator fail threshold 5 bits (0-18). Select counter LSB to check.			

The threshold is specified by a 5 bit value (0-18),where the value indicates the accumulator value bit shift right positions before bitwise OR of higher bits.

There is a dedicated accumulator for color aware pre-colored Green, Yellow and sum of Green+Yellow. The FE state machine is common for all accumulators and a threshold is considered crossed if any of the thresholds are crossed.

Front-End state machine:

The state machine remains in Active state until accumulators are below the Update threshold. This allows aggregate TB updates until specified accumulated color inaccuracy.

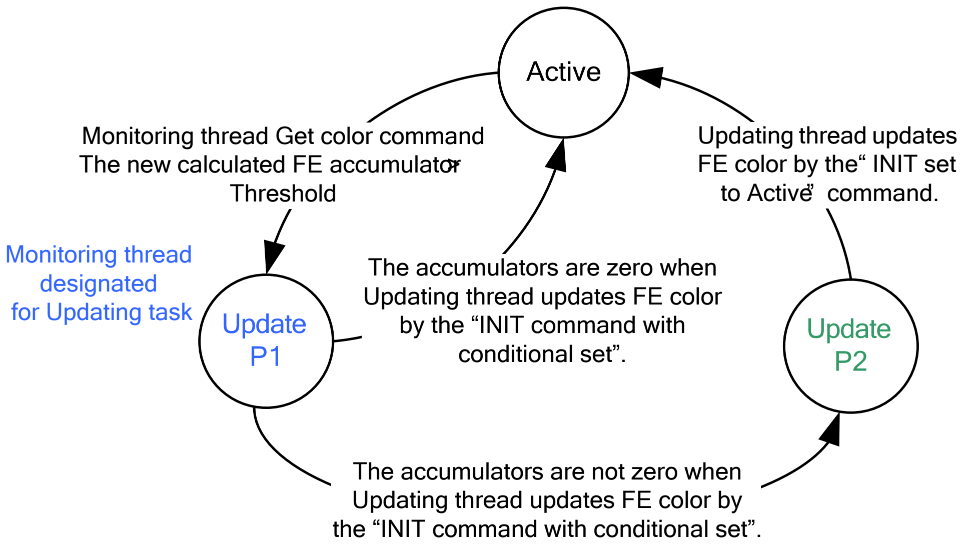
The state machine moves to Update P1 state locking specific thread as responsible for BE TBs updates.

If BE TBs updates are accomplished without accumulating additional bytes, the S.M returns to Active state with updated status color.

If while updating BE TBs the accumulator accumulates more old-color bytes, the updating thread reiterates the BE TBs update and the color is updated per calculation of Update P1.

After second iteration of BE TBs update the Update P2 state is terminated and S.M resumes Active state.

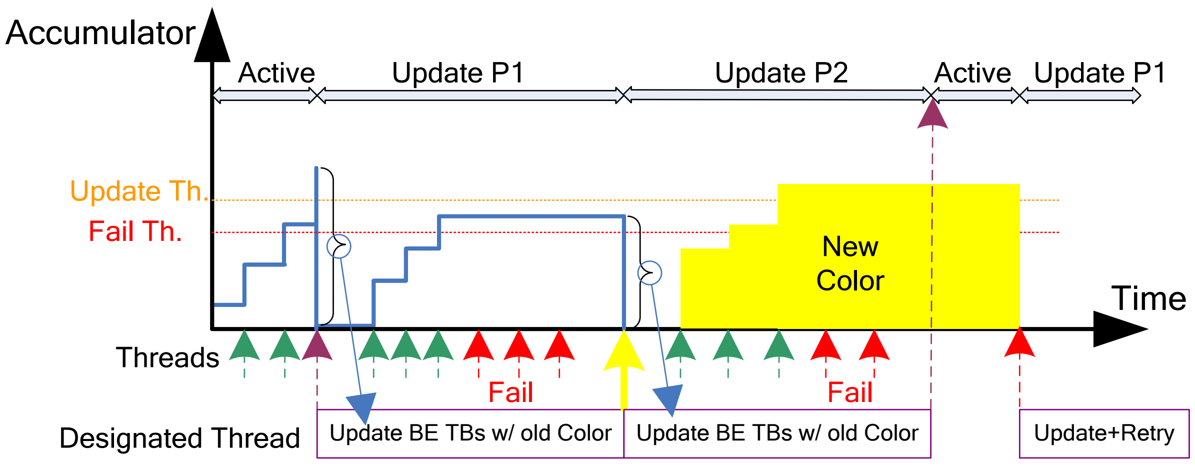
Figure 11-5. State machine


Active
Monitoring thread Get color command
The new calculated FE accumulator > Threshold
Updating thread updates FE color by the" INIT set to Active" command.
Monitoring thread designated for Updating task
Update P1
The accumulators are zero when Updating thread updates FE color by the "INIT command with conditional set".
Update P2
The accumulators are not zero when Updating thread updates FE color by the "INIT command with conditional set".

Thresholds usage:

The accumulator/time diagram below exemplifies FE accumulator transitions.

Figure 11-6. Front-end accumulator transitions



While in the Active state and the accumulator is below Update threshold, the accumulator accumulates TB monitoring events. An attempt to increment above the threshold results in designating the thread as Updater and its transition to Update P1 state. The designated thread uses the accumulated value to update the TBs on the hierarchical path and calculates a new status color. While in Update P1 state, the accumulator allows to increment until passing the Fail threshold. The monitoring threads requesting FE access while the accumulator is above the Fail threshold are replied with Fail and need to re-attempt the access. The Update P1 state is terminated by a new color update that is based on accumulated bytes during Active state. The updating thread gets the old-colored accumulated value during Update P1 state and re-iterates the BE TB update. The Update P1 accumulator is limited by Fail threshold + single packet byte size. The Update P2 state is dedicated for the second phase of update. While in Update P2 state, the accumulator allows to accumulate until Update threshold + packet size. Any Failed access needs to be re-iterated by the monitoring thread. After updating BE TBs, the updating thread transitions the state to Active.

When monitoring thread access FE when the accumulator is already above Update threshold, the thread is designated as Updater but should retry its own TB increment only after the update is done.

Hierarchical Token Bucket Front-End SW commands:

COMMAND	*USAGE*	*OPERANDS*	*RETURN*
Get Color	Metering	Packet byte size Pre-color Green/Yellow	FE status / Application bits BE update assignment Green/Yellow accumulator values Fail / Ok
Init MSB	Initialize HIER_TB FE entry.	Set/clear per field. Status, Application bits FE accumulators Thresholds.	FE status / Application bits BE update assignment Green/Yellow accumulator values
Init	Update FE specific fields. Conditional State transition.	Set/clear per field. Status, Application bits	FE status / Application bits BE update assignment Green/Yellow accumulator values

11.2.2.4.5 Optimized Hierarchical Token Bucket

When Hierarchical Token Bucket type counters are assigned to an Optimized Hierarchical Token Bucket type group, they in affect become Optimized Hierarchical Token Bucket counters. The Optimized Hierarchical Token Bucket storage is split between the internal statistics memory and the EMEM. The color bits are stored within the internal statistics memory for an improved response time, and the actual buckets are stored separately in EMEM.

11.2.2.5 Watchdog Counter Type

Watchdog counters count the number of events in a time period. These watchdog type timers are designed to count KeepAlive events per session. An OAM scan mechanism monitors watchdog timers based on configurable intervals and verifies that each timer is updated according to a specified rate.

The Statistics block performs these commands by CTOPs on the on the Watchdog (WD) counters:

OPERATION	***DESCRIPTION***
Init counter / Read + Init counter (LSB/MSB)	The developer can initialize a counter, with or without read.
Read counter	Read a counter.
Increment counter by 1 Read + increment by 1	Each counter can be incremented by 1, with or without read.
Check watchdog	Check specific WD counter status. Performed by periodic OAM SW action.

In parallel to the application's Watchdog increments, the OAM SW scan mechanism monitors the counter. The OAM SW scan mechanism events are periodically triggered.

The Watchdog counter calculates the events deficit/surplus as compared to profile expected value. If the number of events during the period exceeds min. or max. thresholds, the OAM SW triggers an OAM management event.

The SW query (Check watchdog command) of a Watchdog counter is immediately replied to allowing for low latency identification of OAM protocol disruptions.

The OAM Watchdog counters can operate in two modes:

- Accumulative Window Mode – Constantly average deficit/surplus of events as compared to expected.
- Sliding Window Mode – Monitor deficit of events as compared to expected for few periods of scan.

11.2.2.5.1 Accumulative Window Mode

The Watchdog (WD) entry structure is shown in the following table:

Table 11-3. Watchdog counter entry – Accumulative window mode

NAME	***BITS #***	***DESCRIPTION***
LAST	12	The previous CURRENT event counter to calculate the new events. Each scan, the CURRENT is loaded to the LAST field.
CURRENT	12	The current number of events in the corresponding session. Updated by the application.
INIT BIT	1	Raise after the counter is initialized. Reset after the first check.
VALID	1	Indicates if the counters value is valid. Only valid WD counter is incremented.
PROFILE	4	A pointer to one of 16 profiles, where each profile holds one set of minimum and maximum event thresholds.
ACCUMULATIVE EVENTS COUNTER	5	Counts the number of lost events deficit as compared to per profile expected events in a given scan period.

The OAM SW periodically scans the WD entry (using the Check watchdog command). If the valid bit of the monitored counter is set, the watchdog mechanism will check the number of events that occurred from the previous monitor time until the current monitor time. If the rate of new events underperformed the minimum threshold or exceeded the maximum threshold, the new events deficit or surplus as compared to profile expected value accumulated in Accumulative Events Counter. When the Accumulative Events Counter reaches a configured threshold, the OAM SW is notified by command response and the valid bit is reset until the next initialization.

If the rate of new events was as expected, the entry valid bit remains high.

Accumulative window algorithm:

```
NEW = WD[CURRENT] - WD[LAST] // New events in scan period
If (NEW > PROFILE[MAX_TH])
{ //  Exceed maximal expected threshold. Stop updating WD counters.
            Generate error on non conformant accumulated surplus.
            Clear WD[VALID] bit. // Disable counts till re-initialize
}
Elseif (NEW < PROFILE[MIN_TH])
{  // Events under expected minimal threshold
            // Increment deficit counter. The maximal deficit increment is
            // configurable 0 - 31.
            Deficit = min (|PROFILE[MIN_TH] – NEW|, PROFILE[MAX_ADD_CREDITS])
            WD[ACCUMULATIVE EVENTS COUNTER] = + + Deficit // And no more than 31
            If (WD[ACCUMULATIVE EVENTS COUNTER] < PROFILE[COUNTER THRESHOLD]
                        { // Conformant accumulated deficit
                        WD[ LAST] = WD[CURRENT]; // Save current to last.
                        }
            Else { // Non conformant accumulated deficit. Stop updating WD counters.
                        Generate error non conformant accumulated deficit.
                        Clear WD[VALID] bit. // Disable counts till re-initialized.
            }
}
Else { // Good window. Events in expected range
                  If (PROFILE[RST_CNT]) // Mode of operation is non accumulative for
                                        // good window.
                       WD[ACCUMULATIVE EVENTS COUNTER] = 0
                  Else // Mode of operation is accumulative for good window
                     Surplus = min (|NEW - PROFILE[MIN_TH]|, PROFILE[MAX_SUB_CREDITS])
                         WD[ACCUMULATIVE EVENTS COUNTER] =
                         WD[ACCUMULATIVE EVENTS COUNTER] – Surplus – PROFILE[Drop rate
                         tolerance]       // And no less than 0
      WD[ LAST] = WD[CURRENT]; // Save current to last.
}
```

11.2.2.5.2 Sliding Window Mode

The WD entry maintains an event counter for each of several recent periodic scan windows.

Each window is the time between two adjacent scans of the WD entry.

The number of recent windows is profile configurable to 3, 4, 6, 8, 12 or 24.

The application increments are always to the most recent (current) window. After a scan, the windows slide, losing the oldest one and resetting the new current one.

The OAM SW triggers a request to monitor the watchdog counter (using the Check watchdog command). If the valid bit of the monitored counter is set, the scan WD will check one of two sub-modes:

- Event Accumulation – The monitored criterion is an accumulated number of events that occurred in the last N windows. If the number of events is below the expected minimum or above the expected maximum, the OAM SW is notified via command response and the valid bit is reset until next initialization. When the calculated sum is less than the configured threshold, the OAM SW is notified via command response and the valid bit is reset until the next initialization.
- Good Window Count – The windows are marked as either “good” or “bad”. The monitored criterion is the number of bad windows in the last N checks. A window is considered “bad” if the number of events in it is less than the minimum threshold. When the calculated sum reaches a configured threshold, the OAM SW is notified via command response and the valid bit is reset until the next initialization.

Table 11-4. Watchdog counter entry – Sliding window mode

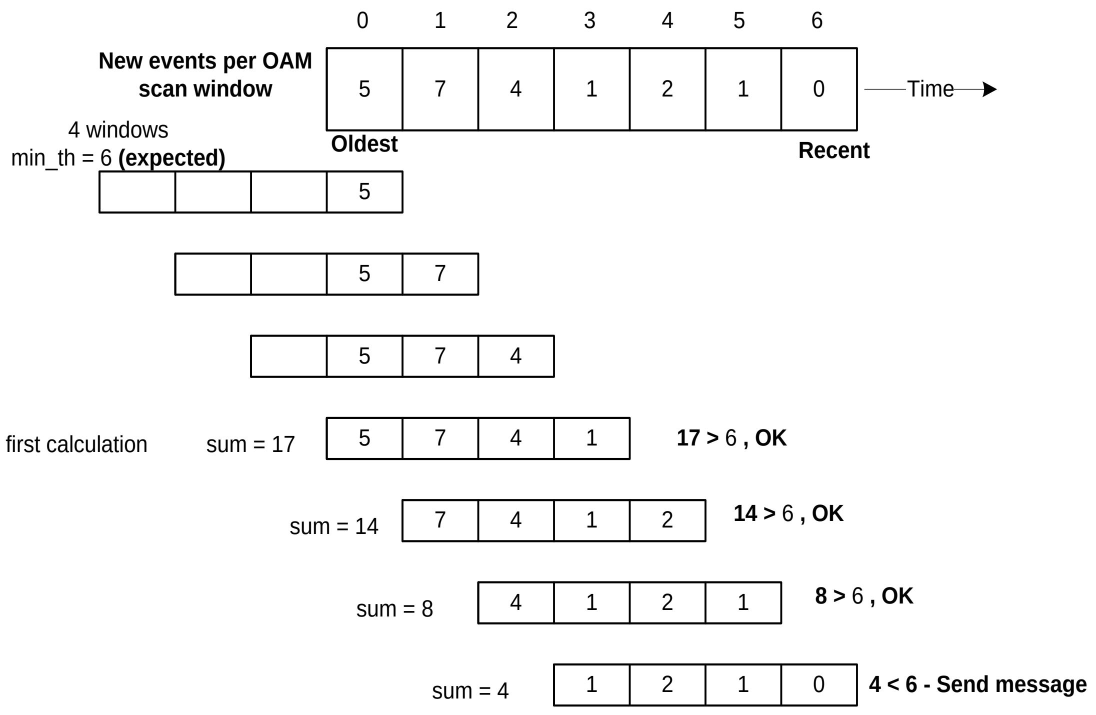
NAME	***BITS #***	***DESCRIPTION***
COUNTERS	24	The field is partitioned to number of windows and event counter bits per window: 3 windows – 8 bits each. 4 windows – 6 bits each. 6 windows – 4 bits each. 8 windows – 3 bits each. 12 windows – 2 bits each. 24 windows – 1 bit each.
VALID	1	Indicates if the counters value is valid. Only valid WD counter is incremented.
PROFILE	4	A pointer to one of 16 profiles, where each profile holds one set of minimum and maximum event thresholds.
VALID WINDOWS	5	Counts the number of valid sliding windows (mainly for reset and recovery from init).

11.2.2.5.2.1 Examples of Sliding Window Sub-modes

Provided below are examples for the two sliding window sub-modes:

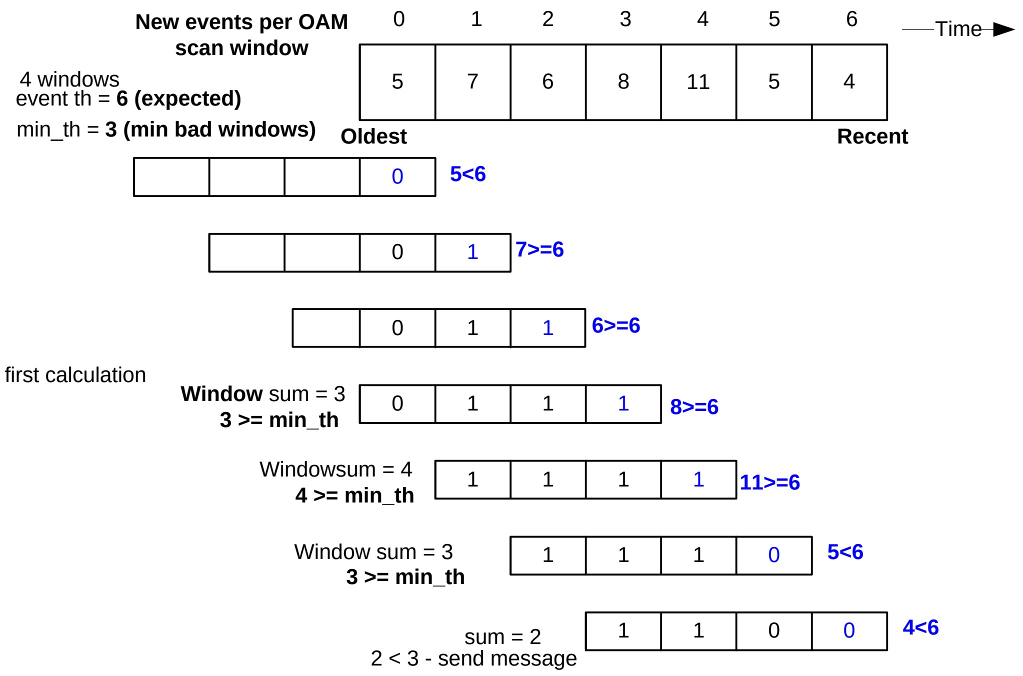
Sliding window example (events accumulation mode):

Figure 11-5. Sliding window (event accumulation mode) example



Sliding window example (good window count mode):

Figure 11-6. Sliding window example (good window count mode) example



11.2.2.5.3 OAM Watchdog Profile Scheme

The watchdog counters within the group can be configured separately. Watchdog counters are configured with an initial value and an associated watchdog profile that defines the thresholds for OAM events.

Watchdog profiles are used to define the behavior and thresholds for the watchdog counters.

Eight watchdog counter profiles are supported for each on-demand group.

Each profile can be associated to several counters.

Table 11-5. OAM Profile Scheme - accumulative window mode

NAME	DESCRIPTION
Minimum threshold	12b value
Maximum threshold	12b value
Check mode	In case of accumulative window's counter mode
Reset counter on good window	0 – Don't reset counter on good window; 1 – Reset
Maximum added credits	Maximum credits to add to the counter on Accumulative windows: 5b value
Maximum subtracted credits	Maximum credits to subtract from the counter on good windows: 5b value
Drop rate tolerance	5b value
Counter threshold	5b threshold of the counter – when counter >= threshold

Table 11-6. OAM Profile Scheme - sliding window mode

NAME	DESCRIPTION
Minimum threshold	12b value
Maximum threshold	12b value
Check mode	In case of Accumulative window's counter mode
Num of sliding windows	1-24 depend on number of sliding windows
Sliding window history threshold	3 windows – each 8 bits. 4 windows – each 6 bits. 6 windows – each 4 bits. 8 windows – each 3 bits. 12 windows – each 2 bits. 24 windows – each a single bit.
Num of events threshold	Must be compatible to window's width. For event, its maximum for window its minimum.
Window/Event mode	0 – event; 1 – window

11.3 Posted Counters

11.3.1 Posted Counter Architecture

Most of the statistics counter updates are simple MIB counters where the application is keeping count of number of bytes, or packets for a given entity. This type of counter does not require a strong consistency, like the on-demand counters, enabling a “fire and forget” mode of operation.

Posted statistics counters are updated by the data plane application while collected by control plane application at lower rates. The posted counters are also preferred target for TM statistics.

The posted statistics management accumulates update commands in commands buffer structures in DRAM. The accumulated commands are then fetched and aggregated in large chunks. The aggregated result is used to update the counter value.

- Four Posted statistics units; two on the north side and two on the south side.
- Each posted unit supports peak performance of update command per posted unit clock (equal to TM core clock).
- Supported counters types: Long counter (64b), Double operation counter (two counters sharing 128b structure, i.e. frame and byte counters).
- Soft error protection is implemented per bundle of 31 counters (64b each) that are protected by 40b ECC.

The update aggregation efficiently utilizes DRAM bandwidth because many updates result in just single command write access to commands buffer while read-modify-writes occur at a lower frequency.

Some commands buffer structures have a single-aggregation-hierarchy (standard), others have dual-aggregation-hierarchy levels (optimized).

Average counter bandwidth per counter update is (64b/4 commands) / 4 (LSB of a counter) x 2(read and write) = 8 bits.

The counter commands supported for the application are: increment, increment with offset, decrement, set, clear, report, report and clear, recycle, recycle and clear.

The posted counters exercise eventual consistency and unordered execution.

The application’s recycle commands exercise a synchronization event that splits commands before and after. Typically the recycle command may be used when you want to stop using a specific counter for one purpose and start using it for another purpose. The Recycle command should be used after the last counter update before the counter is retired. You must ensure that the counter is not used for the new purpose until the recycle reply is received.

The control plane application can issue all the commands, the data plane application can issue and it gets four types of messages from posted counters.

11.3.1.1 Posted Partition

The posted partition is a logical entity which represents a posted counter space. There can be up to 4 posted partitions. Each partition can be spread across 1, 2 or 4 posted units.

The following parameters are defined:

- Application Mask – NPS can run multiple applications where each cluster may run a different application. The application mask indicates to which applications the partition is relevant.
- Unit Mask – Mask of on demand hardware units
  - 4 bits (bit per unit) – A posted unit can only be part of a single partition.
  - Number of units which can share a partition can be 1, 2, or 4.
  - Valid units combinations per partition: (0),(1),(2),(3),(0,1),(0,2),(1,3),(2,3),(0,1,2,3)

- MSID – The memory space ID used by the CTOPs to access the counter range represented by the partition.

11.3.1.2 Posted Groups

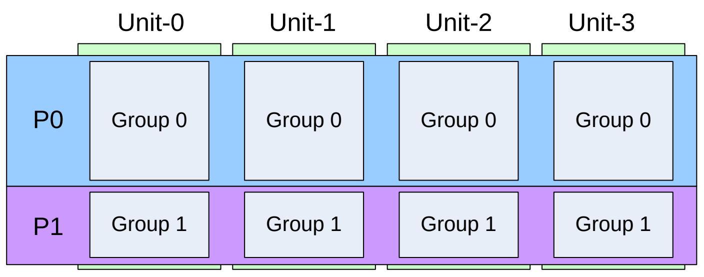
Up to two posted counter groups are supported. All groups of same index per partition are configured the same.

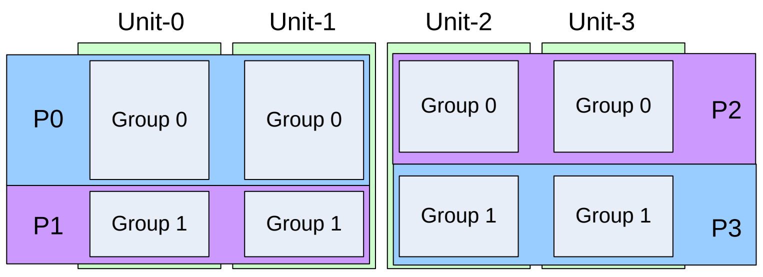
For each of posted counter group, the following parameters are defined:

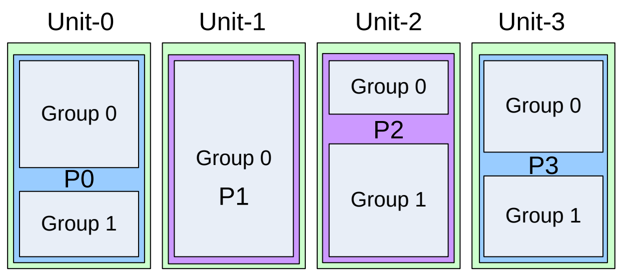
- Partition – Indicates which partition the shadow range is in.
- Group Type – Indicates whether the group represents a standard or optimized posted range.
- Start counter – The first counter in the group.
- Num counters – The number of counters in the group.

Three partition examples below:

- First example: Two partitions (P0, P1) balanced on four units each.
- Second example: Four partitions (P0,P1,P2,P3) with each balanced on two units.
- Third example: Four partitions (P0,P1,P2,P3) with each on single unit with one or two groups per unit.







11.3.1.3 Shadow Counters with Synchronized Swapping

For each group of counters, the control plane application can define an additional set of group boundaries and select the active set for the Statistics block. During run-time, the control plane application can perform a synchronized swap of the group boundaries active set.

This feature enables the control plane application to lock the current set of counters, and then to read all the counters while the Statistics block continues working with the second set of counters.

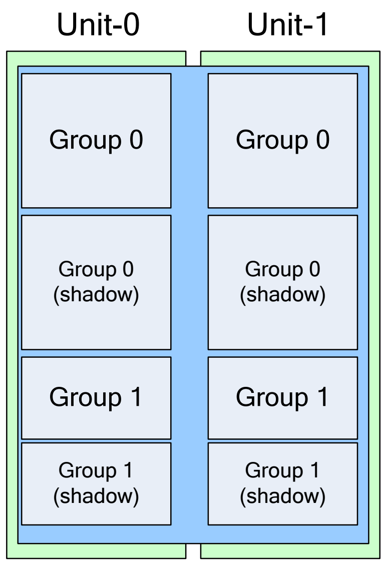
11.3.1.3.1 Posted Shadow Group Parameters

The posted shadow group allows user to remap a range of posted addresses within a partition to another range in the partition. Remapping is bidirectional. Each posted partition contains up to two shadow remapping ranges, one for the standard and one for the optimized posted counters.

The following parameters are defined:

- Partition – Indicates which partition the shadow range is in.
- Group Type – Indicates whether shadow range is within the standard or optimized posted-group/counter-range.
- Enable – Enabling shadow remapping.
- Start Counter – First destination counter of shadow remapping.
- Num Counters – Number of remapped counters
- Remap Counter – First source counter of shadow remapping.

Example of partition balanced on two units with two groups per unit. Each group has shadow group for synchronization swapping.



11.3.1.4 Posted Overflow Profile

Up to 7 posted overflow profiles are supported. Profiles dictate the thresholds and other overflow related configurations for the counter range that they cover. Counters not specifically covered by an enabled posted overflow profile will not generate overflow messages. Counters within range of more than one profile, will be affected by the profile with the higher index (Profile).

All posted counters which are not specifically covered by a posted overflow profile are covered by posted overflow profile 0.

For each of the posted overflow profiles, the following parameters are defined:

- Enable – Dictates if counters covered by this profile will generate messages.
- Start counter – The first counter affected by profile.
- Num counters – The number of counters in the affected by profile.
- Signed – Whether the counters covered by the profile may have negative values.
- Range Type – Profile will affect only even indexed counters, odd indexed counters, or all counters within the range.
- Overflow low threshold – This threshold is used to trigger an overflow message when message queue is not congested.
- Overflow High threshold – This threshold is used to trigger an overflow message when message queue is congested.

11.3.1.5 Posted Garbage Collection Scan Profile

The posted counters are monitored by periodic scan of Garbage Collection (GC) machine profiles. There is configurable GC machine per unit. Profiles dictate the thresholds and other GC related configurations for the counter range that they cover. When counter is above profile threshold, the counter is zeroed and message with counter value is queued to message queue.

Counters not specifically covered by an enabled posted GC profile will not generate garbage collection messages. Counters within range of more than one profile, will be affected by the profile with the higher index (Profile).

Up to 7 posted garbage collection profiles are supported per unit. All posted counters which are not specifically covered by a posted GC profile are covered by posted GC profile 0.

For each of the posted GC profiles, the following parameters are defined:

- Enable – Dictates if counters covered by this profile will generate messages.
- Start counter – The first counter affected by profile.
- Num counters – The number of counters in the affected by profile.
- Signed – Whether the counters covered by the profile may have negative values.
- Range Type – Profile will affect only even indexed counters, odd indexed counters, or all counters within the range.
- Overflow low threshold – This threshold is used to trigger a GC message when message queue is not congested.
- Overflow high threshold – This threshold is used to trigger a GC message when message queue is congested.

11.3.1.6 Posted Message FIFOs

Two posted units on each north/south side share a FIFO message queue in EMEM. The message FIFOs act as a messaging channel from posted units to SW control thread. Each message consumes 16B in memory.

The message queue is maintained by a Read pointer and Write pointer advancing cyclically in the EMEM memory region. The queue size is configurable.

The posted unit writes the message to EMEM and increments the Write pointer.

The SW reads messages and increment cyclically the Read pointer.

The Read and Write pointers are located in EMEM configured location.

The posted unit has a configurable time interval for reading the Read pointer from EMEM.

The posted unit has a configurable time interval for writing the Write pointer to EMEM.

The CTOP has dedicated MSID to access message pointers and data in EMEM.

The designated SW thread is responsible to read the messages and advance the Read pointer.

The SW thread should be activated periodically. The SW thread reads the queue Read/Write pointers from EMEM to identify number of queued messages. The SW thread process the messages and updates the Read pointer in EMEM.

If message generation stalls due to the message queue fullness, counters exceeding their respected thresholds will cause the posted block to halt until SW has evacuated some messages.

The message queue fullness should be avoided by setting an appropriate size to the message queue and allowing the SW thread the priority needed to assure the queue never gets full.

11.3.2 Posted Counter Types

The statistics provides commands to set/clear posted counters, increment/decrement posted counters, as well as to report/recycle posted counters to issue a message with the current value of posted counters.

Report and recycle commands return the counter values through the message queue with optional subsequent zero of the counter.

Two types of posted counters supported:

- Long counter 64b value incremented by a 32 bit value.
- Double counter 64b + 64b (separate command per counter, incremented by two 16b values)

Only the Recycle command is performed in order with regards to other commands in the buffer structures. The Report command issue order does not guarantee execution order.

11.3.2.1 Posted Long Counters

Long counter 64b value incremented by a 32 bit value.

The Statistics block performs these tasks on the posted long counters:

OPERATION	***DESCRIPTION***
Set counter	The user can initialize a counter with 64b value.
Increment Decrement	The counter can be incremented or decremented, by value up to 32 bits. The increment compares the counter to overflow thresholds for message report generation.
Increment with offset	This command increments the counter by the 32b value supplied after shifting left the value to the specified shift left of (0, 16, 32, 48) bits. The increment compares the counter to overflow thresholds for message report generation.
Increment by one Decrement by one	The counter value can be incremented or decremented by one. The increment compares the counter to overflow thresholds for message report generation.
Clear	This command clears the counter value.
Report Report and clear	This command creates a message for the control plane application with the value of the counter. Command issue order does not guarantee execution order. Option to also clear the counter value in DRAM.
Recycle Recycle and clear	This command creates a message for the control plane application with the value of the counter. Recycle has ordered execution with regards to existing commands in the posted buffer. Option to also clear the counter value in DRAM.
GC compare to profile thresholds	The periodic scan compares the counter to thresholds for message report generation.

11.3.2.2 Posted Double Operation Counters

Double counter 64b + 64b (separate command per counter, incremented by two 16b values)

The Statistics block performs these tasks on the posted double counters:

OPERATION	DESCRIPTION
Set counter	The user can initialize a counter with 64b+64b value.
Increment	This command increments both the byte count portion and event count portion of a double counter. The increment values may be up to 16 bits.
Clear	This command clears the counter value.
Report Report and clear	This command creates a message for the control plane application with the value of the counter. Command issue order does not guarantee execution order. Option to also clear the counter value in DRAM.
Recycle Recycle and clear	This command creates a message for the control plane application with the value of the counter. Recycle has ordered execution with regards to existing commands in the posted buffer. Option to also clear the counter value in DRAM.
GC Compare to profile thresholds	The periodic scan compares the counter to thresholds for message report generation.

12. Look-up Acceleration

This section provides a high-level description of the packet processing application data structures look-up acceleration provided by the NPS-400.

Data processing programs make intensive use of various types of data structures for look-ups:

- Direct tables where the key size is fixed in length and small (typically less than 4 bytes). Examples of such tables are input port tables, VLAN tables, service access point attributes, CoS remapping, etc.
- Hash structures where the key size is fixed in length but much longer (typically 4 to 64 bytes). Examples of such structures are MAC forwarding information base, flow tables using an IPv4 or IPv6 5-tuple as the key, MPLS labels, etc.
- M-trie type structures for performing the routing look-ups of IPv4/IPv6 longest prefix match.
- TCAM based look-ups for use in Access Control Lists type look-ups requiring data wildcard masking.

Typically the result size of the various data structures is limited to 256 bytes, except IMEM results which are limited to 32 bytes maximum.

NPS provides a unified framework for efficient querying in user-defined databases, supporting multiple databases formats, including longest prefix match (LPM), multi-tuple Access-Lists (ACL), direct tables, hashed structures, tries and more. Two key ingredients build this framework: a linear-scale database compiler and fast data plane lookup libraries.

The database compiler employs EZchip’s proprietary algorithms to compactly represent user-defined structures in on/off chip memory tables. With linear growth in footprint, database size is effectively unlimited, depending only on the amount of external memory connected. In-service incremental updates are supported in all query formats.

Lookup libraries, optimized for the NPS data plane, allow complex line-rate packet processing with multiple queries per packet, requiring a minimal number of memory accesses.

NPS search framework allows for tuning different runtime parameters per customer-specific requirements, available system resources and performance profiles. In particular, the system allows to trade off memory footprint for lookup performance, so that increased memory budget is translated to increased lookup performance, with fewer cycles and memory accesses per lookup. This is particularly beneficial for per-packet lookup queries. On the other hand, in per-flow processing, both the database compiler and lookup libraries can be tuned to favor reduced memory structures, with sub-linear scale in footprint, at the cost of additional accesses or latency. NPS lookup algorithms also support fine-grained acceleration to a subset of the database records, where the exact subset to be accelerated can be specified by the user, or alternatively can be learned in an adaptive fashion during run time.

12.1 Longest Prefix Match

Internet (IP) address lookup is a major bottleneck in high-performance routers. In principle, an Internet address lookup would be simple if we could look-up an IP destination address in a table that lists the output link for each assigned Internet address. In this case, the lookup could be done by hashing, but a router would have to keep an entry per each destination address, which is clearly infeasible. To reduce database size and routing update traffic, a router database consists of a smaller set of prefixes. This reduces the database size, but at the cost of requiring a more complex lookup called *longest prefix match*. It also requires a more complex update procedure when prefixes are added and deleted.

Classless Inter-Domain Routing (CIDR) is principally a bitwise, prefix-based standard for the representation of IP addresses and their routing properties. It facilitates routing by allowing blocks of addresses to be grouped into single routing table entries. These groups, commonly called CIDR blocks, share an initial sequence of bits in the binary representation of their IP addresses. The CIDR address lookup is challenging because it requires a *longest prefix match (LPM)* lookup.

The LPM lookup is complex because of a nested network hierarchy (each entry in a LPM forwarding table may specify a sub-network) so that, one destination address may match more than one forwarding table entry. The most specific prefix among the matching table entries – the one with the longest subnet mask – is called the longest prefix match. It is also the entry with the largest number of leading address bits of the destination address matching those in the table entry.

The LPM database can be implemented by several techniques:

- Multi-bit Trie – digital radix (prefix) tree. This is an ordered tree data structure that is used to store a dynamic set or associative array. Lookup performance is O(|IP address|/s) time (where 's' is the average tree stride). There are no collisions of different keys in a trie. Trie retrieval is efficient when the tree is balanced. Worst case performance can be slow O(|IP address|).
- TCAM – Ternary Content Addressable Memory. Fully associative memories that allow a “don’t care” state to be stored in each memory cell in addition to 0s and 1s. Lookup performance is O(1). TCAM data structures are expansive in area and power, and typically too small in size to host a whole IP database.
- Hash tables – Use hash tables for prefixes of each distinct length or to a certain prefix length (e.g., /16, /24 and /32 for IPv4) to search in a longest prefix table. Lookup performance is O(prefix length). The number of distinct prefixes can be reduced by expanding the prefixes. Bloom filters are sometimes used to query the existence of the prefix before finding the next-hop information of the prefix. The combination of Hash and Trie algorithms is able to achieve high throughput for long IPv6 addresses by hash function which is used to jump over the sparse part of the IP prefix tree.

The LPM data structure and lookup requirements are:

- Lookup performance – The performance is achieved by a minimal number of memory accesses, especially to the device’s external memory (i.e. DRAM), for average case and worst case. The number of accesses can be minimized with larger stride traverse of the trie data structure (by consuming more IP address bits per stride/lookup). The number of accesses can be minimized by the lookup procedure directly skipping of trie levels.
- Database footprint minimization – The data structures should be compact and minimize wasted (null info.) memory entries. The footprint minimization can be achieved by a correct trade-off of trie node data representation.
- High update rate – The data structure must support incremental real-time updates at high rate. The high update rate can be achieved by minimizing number of accesses per update. The large stride of trie traverse may increase update accesses. The prefix-expansion also increases the number of accesses per update.

- High availability – The management control plane SW should be able reconstruct the data structure image based on the router database in case of management SW re-initialization. This requires that the data structure image can be reconstructed with order agnostic incremental updates replay and with minimal query access to data plane forwarding structures.

12.1.1 UltraIP LPM Lookup Acceleration

EZchip's UltraIP supports mixed IPv4 and IPv6 records with VRF (Virtual Router Function) in a single unified database. UltraIP offers advanced methods and capabilities that are efficient in both footprint and runtime aspects. EZchip's earlier NP devices offered a FastIP package which effectively consumed the search key in a byte-by-byte order.

The UltraIP solution comprised of:

- EZchip's LPM database compiler – The UltraIP is a dedicated lookup structure specialized for performing IPv4/IPv6 routing Longest Prefix Match look-ups. The UltraIP structure implements variants of radix trees (or tries) to store associative arrays containing large ranges of values with few exceptions. UltraIP structures support insertion, deletion and search operations in O(maximal element length). Each UltraIP node is a decision point that takes a combination of several proprietary decision structures. All UltraIP node structures have embedded ECC protection.
- Runtime libraries – UltraIP hardware has built in methods and processing capabilities that reduce memory accesses by consuming the input key at a typical rate of several bytes per search node. UltraIP accommodates multi-byte progression the entire search path, namely even when the number of split branches at every byte position is significantly larger than one. Compared with FastIP which requires 5 memory accesses for IPv4 and up to 17 accesses for IPv6, the UltraIP data plane completes such queries within as low as 2 accesses for IPv4 and typically between 3 to 5 accesses for IPv6.
- UltraIP lookup acceleration – The data structure traversal and memory access are handled by HW accelerated dedicated CTOP instructions. The SW processing is minimized to trigger UltraIP instruction per trie node decode. The UltraIP execution and memory access latency is hidden from CTOP execution by thread scheduling.

12.1.2 UltraIP LPM Database

12.1.2.1 Virtual Router

The LPM database is constructed per Virtual Router ID and typically the VR-ID is 16b. The LPM lookup starts with VR-ID index to VR table to retrieve the LPM trie root.

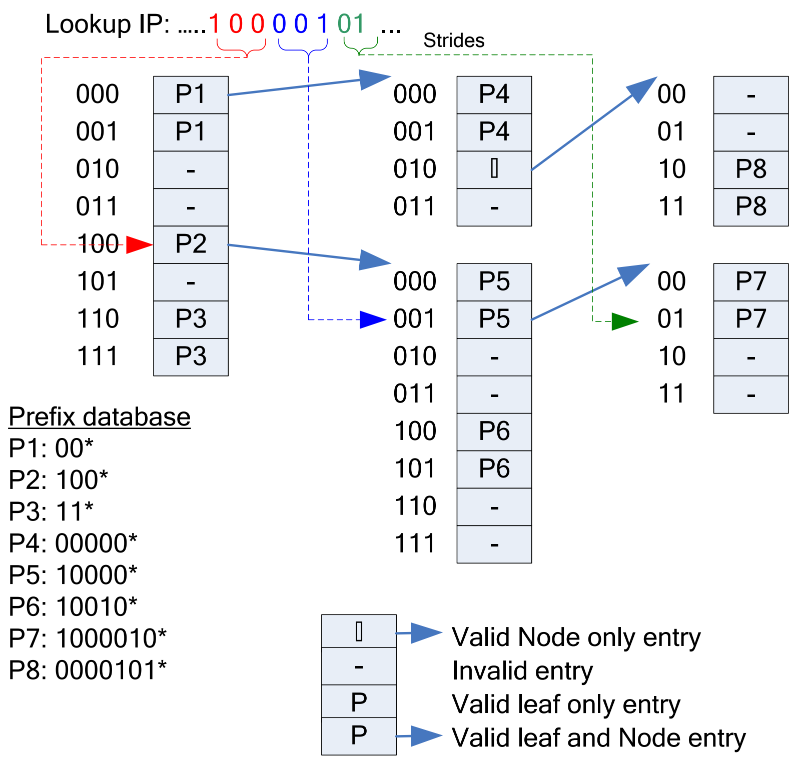
12.1.2.2 LPM Trie Data Structure

The compiler compiles the database into a sequential, multi-bit, Patricia trie. A Patricia trie is a search tree in which every edge represents a 'sub-string' and every node represents a 'string' consisting of the 'sub-strings' of the unique path from the root to that node. Thus, the depth of a trie for a set of strings is equal to the maximum length of a string in the set of strings, broken into 'sub-strings' and, therefore, can be much larger than the depth of a balanced search tree.

Every node has up to the number of children of the radix *r*, where *r* is 2^x and *x* is the trie stride ($1 \leq x \leq 12$). Small *x* minimizes sparseness at the expense of increasing trie depth. When x is large it lessens the depth of the radix trie at the expense of potential sparseness.

To maintain prefixes with arbitrary lengths, the Patricia trie also allows inner nodes with just one child, but only if such a node represents a prefix in the database (that is, node lists in the trie cannot be compressed beyond these nodes).

The lookup algorithm traverses the trie from the root reading the node data by single memory access (typically of 16B). Each node can index an array of 2^x sub-trie entries. Each node may hold a valid LPM result (route next hop) or hold a null result. Each child node has a single father node indexing it (tree structure).



Each trie node codes the following information:

- Validity. Prefix leaf and trie node can co-exist:
  - Valid prefix leaf holding LPM next-hop result.
  - Valid trie-node.
  - Invalid entry. Terminates trie traverse.
  - Prefix leaf information (or index to prefix leaf).
- Trie-node information:
  - Pointer to child structure.
  - Prefix bit selected (the trie stride) for child index.
  - Validity bitmap table for child structure. The (K:1) bitmap encodes K child entries to one validity bit. Child validity bit is set as non-valid when all K child entries are invalid. The validity bits save redundant access to non-valid (NV) child entries.

12.1.2.3 Prefix Expansion to Increase Trie Stride

The prefix expansion is useful to increase the stride of the trie traverse.

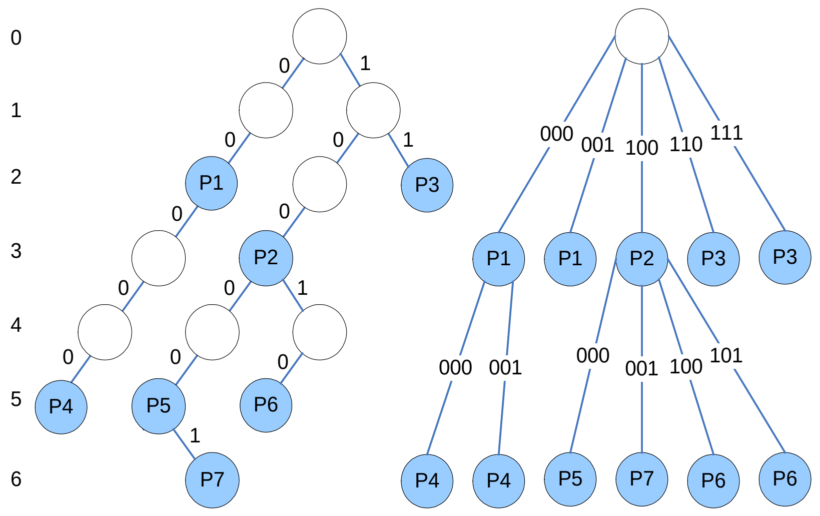
Prefix expansion converts a set of prefixes with *I* (initial number of) distinct lengths to an equivalent set of prefixes with *F* (final number of) distinct lengths where *F* < *I*. Expansion can considerably increase storage.

When a lower length prefix is expanded in length and one of its expansions "collides" with an existing prefix, then we say that the existing prefix *captures* the expansion prefix. When this happens, we simply get rid of the expansion prefix.

If the number of distinct prefix length is *F*, the trie's worst-case performance bound is O(F).

Figure 12-1. Prefix expand

PREFIX	***I = 4 (2,3,5,6)***	***F = 2 (3,6)***
P1	00*	000* 001*
P2	100*	100*
P3	11*	110* 111*
P4	00000*	000000* 000001*
P5	10000*	100000* 100001*
P6	10010*	100100* 100101*
P7	100001*	100001*



- *Note: When P5 is expanded it collides with P7 and gets "captured".*

12.1.2.4 Sub-trie Skips

In order to achieve LPM lookup performance requirements the LPM trie traversal must minimize the average number of memory access. Reducing the number of accesses can be achieved by larger traverse strides. The large strides introduce an associated cost of large memory requirements with probability of low memory space utilization due to trie structure sparseness. A trie structured with large traversal strides but still resulting in high memory utilization is an adequate solution.

When a trie data structure is sparse, the trie node traversal can be augmented by skipping sparse parts of the trie.

The inner prefix nodes of skipped sub-trie are pushed (prefix expended) to the nodes of the sub-trie's last level.

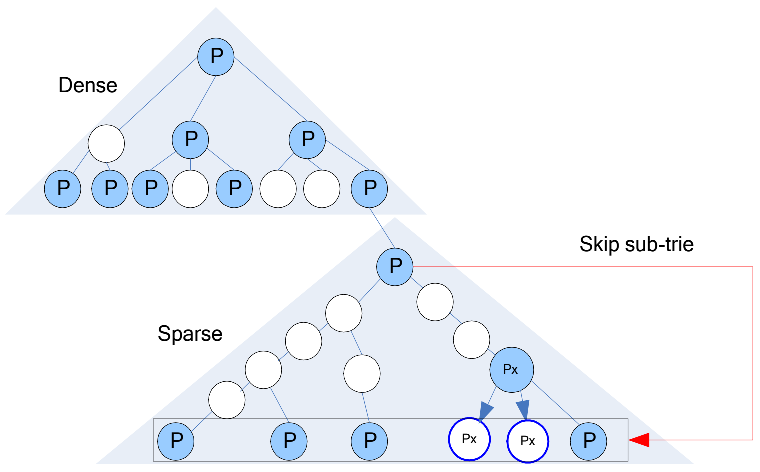
The root of the skipped sub-trie must be a prefix node or prefix expanded node from the father.

The memory space efficient structure to hold the skipped sub-trie prefixes can be implemented by:

- Hash table – well suited for a large list of equal length sub-prefixes.
- Pseudo-TCAM – can support a short list of sub-prefixes that are of different lengths.

Figure 12-2. Sparse sub-trie skip

Inner node Px is expanded to sub-trie last level.



12.1.2.5 Sub-trie Skip by Hashing

Examination of router forwarding database reveals that IP prefix length distribution is far from being uniform; the distribution is heavily concentrated at specific prefix lengths:

PREFIX LENGTH	***IPV4***	***IPV6***
/20	~2%	
/21	~3.5%	
/22	~12%	
/23	~20%	
/24	~57%	
/29		~2.5%
/32		~30%
/40		~5%
/44		~4%
/48		~40%

The sparse sub-trie, after expansion of inner nodes, constituting a large number of equal length sub-prefixes is a good candidate for sub-trie skip by a hash-table method.

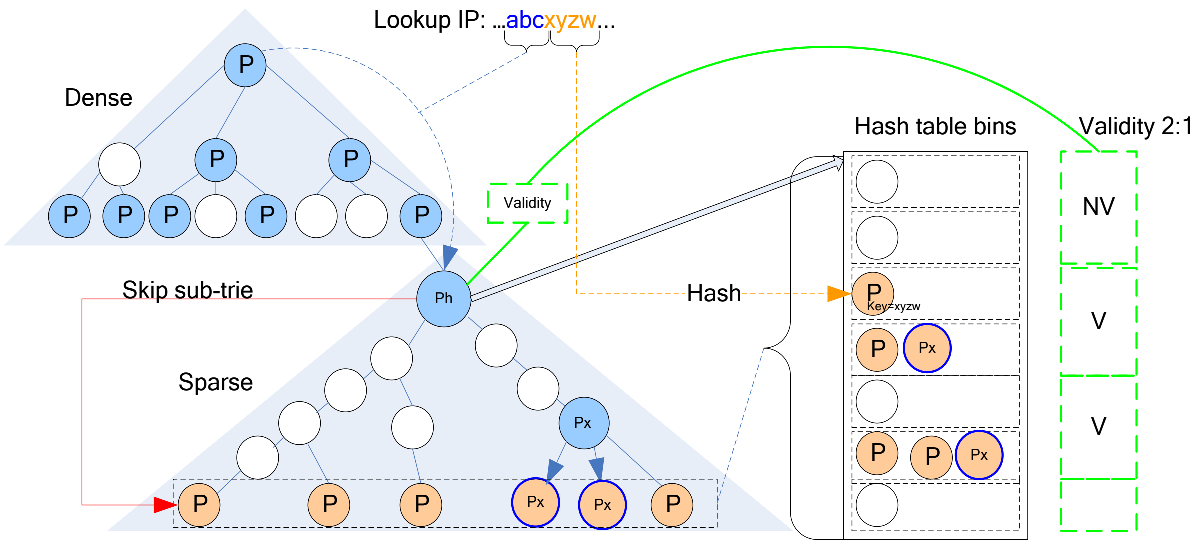
The sub-trie root prefix (Ph) node encodes the sub-trie prefix bits and holds a pointer to the hash table.

The sub-trie root prefix (Ph) node holds Validity bit map table for hash bins. The (K:1) bit-map encodes K hash bins to one validity bit. The validity bits save redundant access to non-valid (NV) hash bins.

The hash table entry holds the partial prefix as a key.

The hash table entry insert/delete update is executed efficiently with O(1). Usually the hash table space is x2 to x3 the number of entries. The hash table must handle collisions on entry insertions to prevent hash bin overflow.

Figure 12-3. Sub-trie skip by hash table



12.1.2.6 Sub-trie Skip by Pseudo-TCAM

The sparse sub-trie that constitutes a small number of various length sub-prefixes, where all lengths are of approximately the same order, is a good candidate for sub-trie skip by the pseudo-TCAM method.

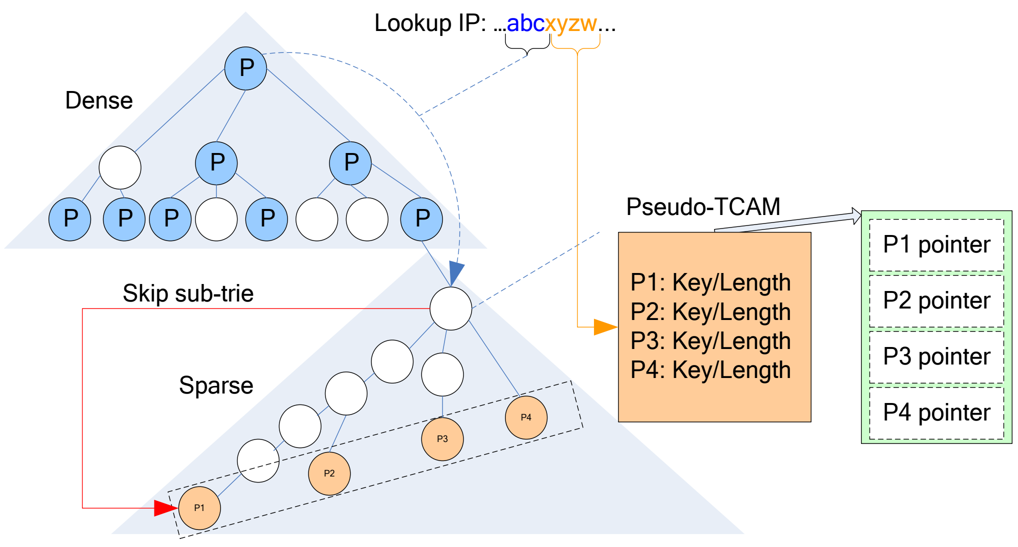
The sub-trie root prefix node encodes a small list of pseudo TCAM entries holding a list of Key=Prefix and prefix length with a pointer to next hop table.

The 16B allocated for pseudo-TCAM prefix node can hold short list of keys:

- x8 entries of up to 8b
- x5 entries of up to 16b
- x3 entries of up to 24b
- x2 entries of up to 32b
- x1 entry of up to 64b

The pseudo-TCAM table entry insert/delete update is executed efficiently with O(1) until an entry does not overflow. When a pseudo-TCAM entry overflows, the trie is rebalanced.

Figure 12-4. Sub-trie skip by pseudo-TCAM



12.1.3 UltraIP LPM Compiler and Performance Estimation

The compiler is used to compile an input set of prefixes to an LPM database. The user supplies the IMEM space restrictions for the compilation.

The compiler reports the estimated database image footprint in IMEM and DRAM.

The compiler can estimate database performance by simulating IP lookups. The compiler generates a uniformly distributed set of IP addresses to match each prefix rule.

The IP lookup simulation reports results statistics:

- Average number of accesses to IMEM. Maximal number of accesses to IMEM.
- Average number of accesses to DRAM. Maximal number of accesses to DRAM.
- Average number of accesses. Maximal number of accesses.

12.1.4 UltraIP SW Driver Services

The SW driver serves the control plane management at run time. The SW is expected to update the database at a rate of 100K updates/second.

The services are:

- Update database: Add prefix
- Update database: Delete prefix
- Update database: Modify prefix
- Optimize database
- Defragment memory

The database update services perform "add/delete" of entries without an attempt to optimize the resulting structure.

The optimize database service needs to be evoked periodically to rebalance the trie structure and optimize the trie skip methods. The data plane IP lookup procedure can run-time monitor the data structure entries usage statistics and request optimization activation when statistics deviate from expected performance.

The database add/delete updates eventually lead to memory fragmentation. The fragmentation may impose constraints on memory allocation of contiguous space required for an efficient database. The management activates the memory defragmentation service to heal the situation.

12.1.5 UltraIP Data Plane Usage

The UltraIP lookup execution is accelerated by dedicated HW shared by CTOPs in the same cluster.

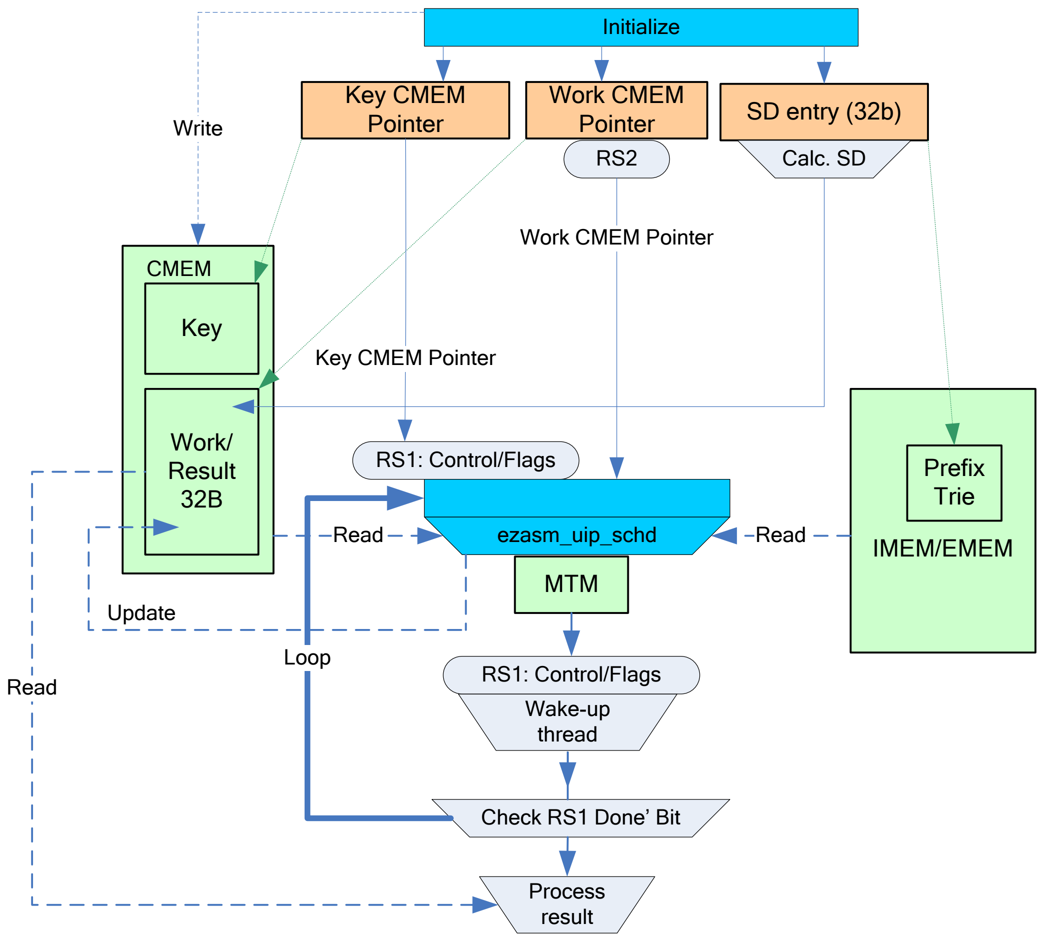
The usage steps are:

1. Initialize (zero working flags, set modes) CMEM Working/Result area of 32B.
2. Initialize CMEM lookup Key area (per size of the Key).
3. Calculate Search Descriptor of Prefix database, based on SD entries from control plane.
4. Write the calculated SD and address-type to CMEM Working/Result area.
5. RS1 register <= {Done = False, Key CMEM ADDR = Key pointer + 1byte, zero other fields}.
6. RS2 register <= {SRC/DST CMEM ADDR = pointer to CMEM Working/Result area}.
7. Loop:
   a. UltraIP instruction (Destination register, RS1, RS2)
   b. Schedule thread (wait until UIP execution terminates)
   c. Check instruction-destination register for Loop-Done, jump to loop.
8. Get result in CMEM Working/Result area, either 4B or 15B.

Each loop iteration, the UIP instruction performs a memory read access and decodes the read data format, then the CMEM working area and destination register are updated.

Each loop iteration costs three CTOP execution cycles. The acceleration HW memory access latency and decode latency are hidden from CTOP core by thread scheduling.

Figure 12-5. UltraIP instruction flow



12.2 ACL, Algorithmic ACL, Multi-Tuple Match

NPS provides the algorithmic TCAM as an alternative to using the conventional TCAM approach. External TCAMs are expensive, high-power devices that while necessary in some designs, can be problematic in designs that do not require the full performance of the external TCAM and have stricter power or real estate design constraints. EZchip's NP devices provide search data structures (i.e. tables, hash) in DRAM as alternative to using external TCAMs. The DRAM-based data structures provide a high search rate for direct table and hash table accesses but do not provide satisfactory performance for tree-only based searches that are used to emulate TCAM look-ups.

See the *Algorithmic TCAM Extension* chapter.

12.3 Advanced Hash

NPS contains several types of hash functions, including both linear and nonlinear hash operators. A single memory access typically resolves between a large number of collisions. Built-in single-cycle instructions compute a selected hash function on input sizes between 1B to 8B. Hash over input sizes 9B to 64B requires multi-cycle computation with thread context switching to hide latency. Wider hash computations are done in software using these hardware building blocks.

12.4 Look-up Acceleration Approach

The NPS-400 architecture utilizes the DMAs and HW accelerated look-ups for data access.

The data lookup approach relies on:

- Structure definition entries – The structure definition entries contain pre-configured attributes.
  - Dedicated instructions and HW accelerators for lookup key preparation – hashing and SD combination. Dedicated instructions for hash entry compare.
  - HW accelerations and DMA operations to access EMEM/IMEM data.
  - CMEM for lookup results destination. The lookup results are usually >4B and thus must utilize CMEM for the result's destination. IMEM results are limited to 32 bytes maximum.

The lookup acceleration instructions are of two types:

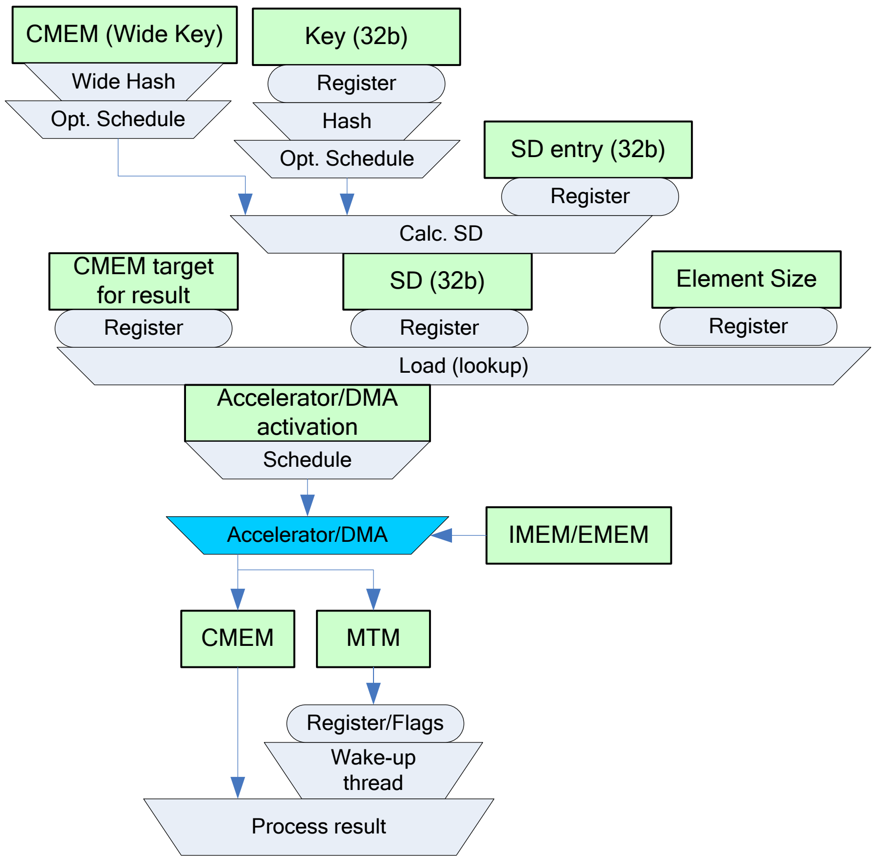
- CTOP level implemented instructions that can be completed in a single CTOP cycle. Usually these instructions utilize parameters/results that can be passed via CTOP registers. These instructions calculate hash and index offsets, decode and compare Lookup results.
- Multi-cycle instructions that activate cluster level shared accelerations or access IMEM/EMEM. Usually these instructions utilize parameters/results that are passed via CMEM. The HW implementation of these instructions is shared and thus results in a variable length execution delay that is better served by thread switching (scheduling).

12.4.1 The Look-up Flow

The basic look-up flow is as follows (Figure 12-6):

1. The lookup key is loaded to a register or CMEM. Short keys are loaded to a register while wide keys are loaded to CMEM.
2. For a hash lookup, the key is hashed. For wide keys, the hashing requires shared accelerator so the thread is scheduled out. Short keys are loaded to register while wide keys are loaded to CMEM.
3. The SD entry is loaded to a register. The SD entry is in CMEM and does not require thread switch (scheduling out). The SD entry specifies the structure address and size.
4. The Search Descriptor (SD) is calculated based on the SD entry and the key. The SD results are placed in a register.
5. The look-up target address in CMEM is loaded to a register.
6. The lookup element size is specified in the instruction as an immediate value.
7. The lookup is triggered (load 16B / load 32B). The lookup activates the accelerator or DMA for >32B EMEM loads.
8. The thread is scheduled out while waiting for the result.
9. The HW accelerator, or DMA, reads the IMEM/EMEM to the target CMEM address.
10. The MTM monitors the transaction and wakens the thread when the transaction is completed.
11. The result's flags indicate the look-up result status.
12. The result can be read from CMEM and processed.

Figure 12-6. Basic look-up flow



12.5 Look-up Structure Definition Entries

A Structure Definition entry is used to describe each of the search structures. The entry is configurable and is initialized by EZcp. There is no architectural limitation for the structure's size since it can be implemented either in the CMEM or in the local IMEM. Practically, 128 structures (i.e. entries) are more than sufficient.

The Structure Definition entry is 32-bits and allows efficient reading and writing of the SD_ENTRY by the CTOP core to/from registers.

The SD_ENTRY, combined with the lookup key, is used to calculate the Search Descriptor (SD) for lookup access by dedicated CTOP instructions.

The lookup key can be of two types:

- Table index: Directly index the specific entry of the structure.
- Hash key: Index hash structure entry (leaf) or hash signature page.

The following is a detailed description of an SD_ENTRY:

Table 12-1. Structure Definition entry format

NAME	BITS #	BYTES #	DESCRIPTION
INDEX_BASE_ADDR(15:0)	b15:0	1-0	Base address of the structure. This is the logic offset within a specific memory space identifier (MSID), index resolution: IMEM: 1KB EMEM: 1MB
MSID_SELECT(3:0)	b19:16	2	Memory space identifier (MSID) selector of the structure. Up to 16 MSIDs per cluster in IMEM/EMEM.
KEY_BITS(4:0)	b24:20	2-3	Structure number of elements specified as number of key valid bits. Table structure key size in bits is 1b to 32b. Hash1 size in bits is up to 32b. [0] – Number of elements is 2^{0+1}=2 .. [9] - Number of elements is 2^{9+1}=1024 .. [31] - Number of elements is 2^{31+1}= 4G Used for masking the input key: [0] – 1b LSB key…. [31] – 32b LSB key
RESERVED	b26:25		
IMEM/EMEM	b27	3	Structure location: internal or external memory
KEY_SHUFF_BITS(2:0)	b30:28	3	Bitmap for (table index) key shuffling
KEY_SHUFF_EN	b31	3	Enable/disable key (table index) shuffling

12.5.1 Search Descriptor (SD) Usage

The Search Descriptor is a handler to a specific search structure data elements and it serves as a summarized physical address of this structure. The SD is 32-bits and can be calculated in a CTOP register. The SD can index the entire range of the physical addresses by utilizing MSID_SELECT and the use of index as the structure's base address within a specific MSID.

The SD is calculated by a dedicated instruction named *calcsd*. The instruction uses an SD_ENTRY, key and result element size to calculate the SD to the element.

```
R1 <- ld SD_ENTRY
R0 <- ld KEY
R4 <- calcsd_i R1,R0,ELEMENT_SIZE
```

The SD generation process has two steps:

- Element index generation
- MSID incorporation

12.5.1.1 Element Index Generation

Generally, an index is calculated by adding an element offset (based on key/index) to the structure's base address. The structure's based address is provided within the structure's SD_ENTRY or explicitly loaded by software as an immediate value.

The base address is specified relatively to a specific MSID and it has a granularity of 1K when the structure is in the IMEM or 1M when the structure is in the EMEM. Therefore, the base-address index needs to be padded with 10-bits for IMEM and 20-bits for EMEM in order to convert the base address to a physical byte address.

The structure is composed of fixed size elements, thus the base index should be trimmed (by bit right shift) to align with the element size.

Once the index is normalized, the required element-index (key offset) can be added to retrieve a specific element indexed within the structure. However, in order to add the offset to the normalized base index the offset needs to be aligned with the structure size (number of elements).

For table structures, the index can be shuffled to avoid sequential elements being mapped to the same physical device.

Figure 12-7. Index element calculation

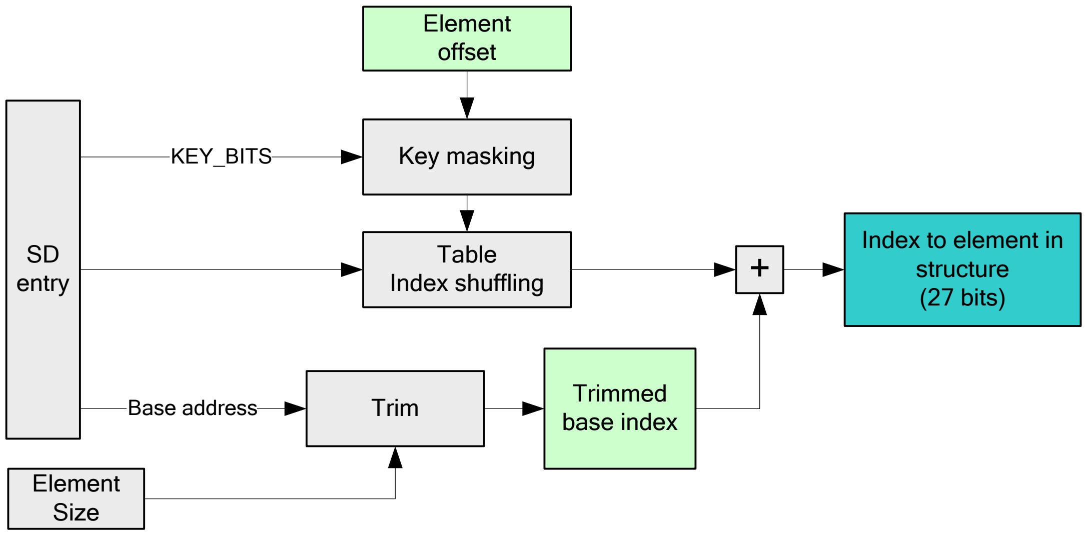
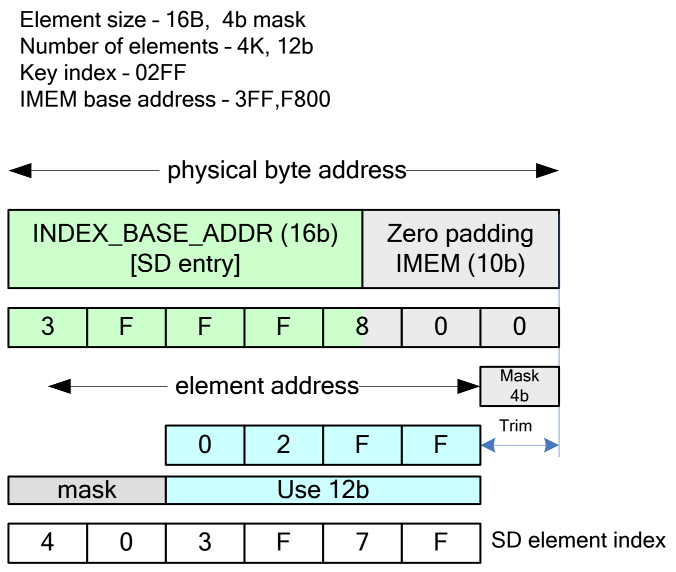


Figure 12-8. Base address to index conversion example



12.5.1.2 MSID Incorporation

The SD requires that the MSID be specified to perform a memory access.

Per cluster there is a configurable range of 16 MSIDs to use out of the 256 available.

There are two SD formats:

- Regular SD for IMEM/EMEM access.
- Extended EMEM access SD.

Regular SD format:

The SD is an element index within a specific MSID either in IMEM or EMEM. The SD contains the MSID selector. The regular SD can manage data structures with 2^{27} = 128MB of elements.

NAME	*BITS #*	*DESCRIPTION*
ELEMENT_INDEX(26:0)	b26:0	Index of the element within a specific structure
MSID_SELECT(3:0)	b30:27	Select MSID of the structure from cluster configurable range of 16 MSIDs (from 256 available).
IMEM/EMEM	b31	Structure location: internal or external memory

Extended SD format:

The extended SD supports data structures with huge number of elements located in EMEM. The element index size can grow up to 30 bits allowing structure sizes of 1GB elements.

XSD (eXtended Search Descriptor)

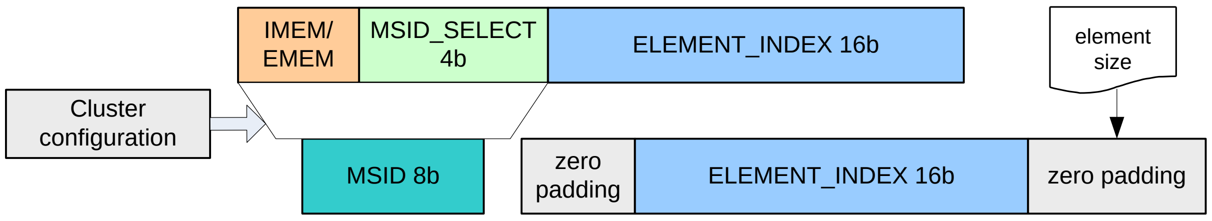
NAME	*BITS #*	*DESCRIPTION*
ELEMENT_INDEX(29:0)	b29:0	Index of the element within a specific structure.
MSID_SELECT(1:0)	b31:30	Select MSID of the structure from cluster configurable range of 2 MSIDs (from of 256 available).

12.5.1.3 SD to Physical Address Transformation

Transformation of an SD to a physical address includes the following steps:

1. The SD [IMEM/EMEM, MSID_SELECT[3:0]] selects the 8-bit MSID from configured values of the sub-cluster.
2. Zero padding of the element index according to the structure element size. IMEM results are limited to 32 bytes maximum. E.g. A structure with a base element of 16B will be padded with 4 zeros.

Figure 12-9. Search Descriptor to physical address transformation



The MSID assignment function:

- When SD specifies IMEM, the MSID[7:0] = 0000,MSID_SELECT[3:0];
  This range of MSIDs is also mapped by the SW Fixed Mapping Table (FMT).
- When SD specifies EMEM, the MSID[7:0] is assigned by sub-cluster MSID table entry selected by MSID_SELECT[3:0].

The per sub-cluster MSID attributes specify:

- Address range sharing.
- Access privilege rights.
- Target EMEM L2-cache coherency groups.

12.6 Lookup Acceleration Instructions

The CTOP is equipped with dedicated instructions to accelerate the look-up in data structures managed via Search Descriptors and an SD entry.

There are several types of instructions based on the functional element that executes the command:

- CTOP core level instructions – These instructions are executed by the CTOP. The CTOP thread waits for execution to complete in a few core clock cycles. All parameters and results are up to 32b in registers or CMEM.
- MTM level instructions – These instructions executed by the MTM (or shared by two MTMs). The CTOP thread is scheduled to allow execution of other threads.
- DMA commands – These instructions trigger the CLDMA. There are two DMAs per cluster to execute the command. The CTOP thread is scheduled to allow execution of other threads.
- Cluster level instructions – These instructions executed by cluster shared resource.

Table 12-2. CTOP ISA instructions for look-up acceleration

INSTRUCTION NAME	INSTRUCTION TYPE	DESCRIPTION	RESULT
hash	Core Level (ISAE)	Hashing of 1B to 8B, supports 16 different hashing flavors	Up to 32-bit hash result, with masking capabilities
calcsd_i calcsd_i.xd calcbsd_i calcbsd_i.xd		Calculate SD: Gets SD entry, key and element size and calculates the corresponding search descriptor (SD) Calculate base SD: Gets SD entry and element size and calculates the corresponding search descriptor (SD) to start of structure.	32-bit register
calckey		Calculate key based on SD entry (performs key mask and shuffling according to SD entry). Used together with calcbsd when entry size is not known at compilation time.	
qcmp_i.ar qcmp_i.al		Quad bytes compare. Compares a single byte against four bytes. Used for hash signature page processing.	In register: Index to the first matched byte
ld16_i ld32_i	Per MTM	Reads 16B/32B structure entry based on SD from IMEM/EMEM and stores in the CMEM. Tracks up to 4 load transactions per thread (1024 per cluster).	First 16B/32B word of the structure entry
mcmp_i	Per 2 MTMs	Memory compare of up to 256 bytes of data stored within the CMEM. Supports offset addition and skip bytes.	Success/fail indication
uip		UltraIP: IPv4/IPv6 Longest Prefix Match via M-trie trees.	
whash	Cluster Level	Wide hash of 1B to 64B key in CMEM.	32-bit hash result
ldsd	CLDMA	DMA load data from IMEM/EMEM based on SD to CMEM.	1-1K bytes

12.7 Direct Table Look-up

A direct table is one of the most common yet simple structures supported by NPS-400.

A direct table structure is defined by a base index within an MSID and a table size. Each entry in the table may hold a value (result) which is copied to the CMEM by the 'lookup' instruction.

The lookup is executed by several 'ld16_i'/'ld32_i' instructions or DMA 'ldsd'.

The table entry size may vary from table to table and is not limited to a specific size.

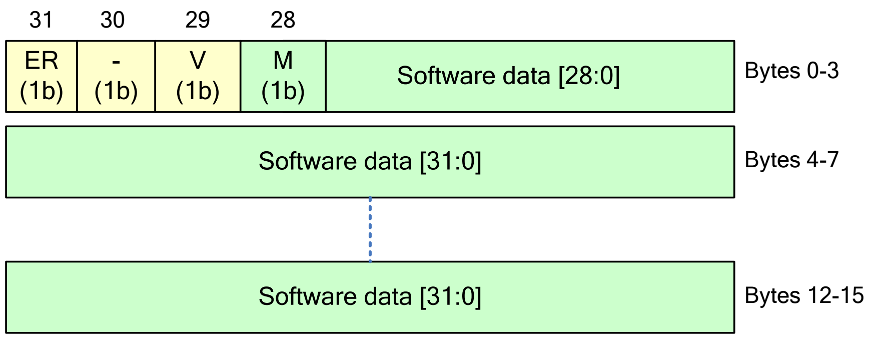
The table key is limited to 27 b with a regular SD.

The table key is limited to 30b with an extended SD.

The table result is limited to 256 bytes.

The table can be located in IMEM or EMEM depending on its size.

Figure 12-10. Direct table look-up format



The memory access HW uses 3 bits out of 16B or 32B for ECC embedded protection when writing the 16B/32B structure to memory.

The SW can optionally use 1b for entry Match (M) indication.

When a table entry is read, the first 4B can be optionally loaded to a CTOP register (in parallel to CMEM).

The HW sets the first 3b of the first 4B:

- ER – Error indication. The ECC uncorrectable error is indicated by setting this bit.
- V – Valid indication. Indicates the entry was initialized by SW with M=1. V= not(ER) & M.

This allows SW to check only bit 29 for Match and no Error.

12.7.1 Lookup Table Entry up to 32B

Each thread can issue up to 4 lookups/load instructions before being stalled on waiting for result(s).

The MTM tracks load competition to wake up the waiting thread.

The results are written to CMEM.

The lookup steps are:

1. Load SD entry and key to register.
2. Calculate SD based on SD entry, key and result size.
3. Load the result pointer to CMEM to register.
4. Issue ld16_i or ld32_i command to lookup 16B or 32B.
5. Examine result in the CMEM.

For table results of up to 4x32B, the steps can be repeated with incremental key up to 4 times before the thread is stalled.

12.7.2 Lookup Table Entry up to 256B

The Cluster DMA (CLDMA) is utilized for the lookup of long entries. IMEM results are limited to 32 bytes maximum.

There are two CLDMAs per cluster. The CLDMA provides data transaction services to all NPC threads (each CLDMA serves 8 CTOPs and 128 threads). Each CLDMA can accept up to 256 DMA commands from 128 HW threads.

After activating the DMA transaction the CTOP thread can be suspended until the transaction terminates and is awakened by the MTM. The CTOP schedules other threads while the suspended thread waits for CLDMA termination.

The results are written to CMEM.

The lookup steps are:

1. Load an SD entry and key to register.
2. Calculate the SD based on the SD entry, key and result size.
3. Load the result pointer to CMEM to register.
4. Issue an ‘ldsd’ command for DMA lookup.
5. Schedule the thread awaiting result.
6. Examine the result in the CMEM.

12.8 Hash Look-up

The hash data structure implements an associative array that maps keys to results (values).

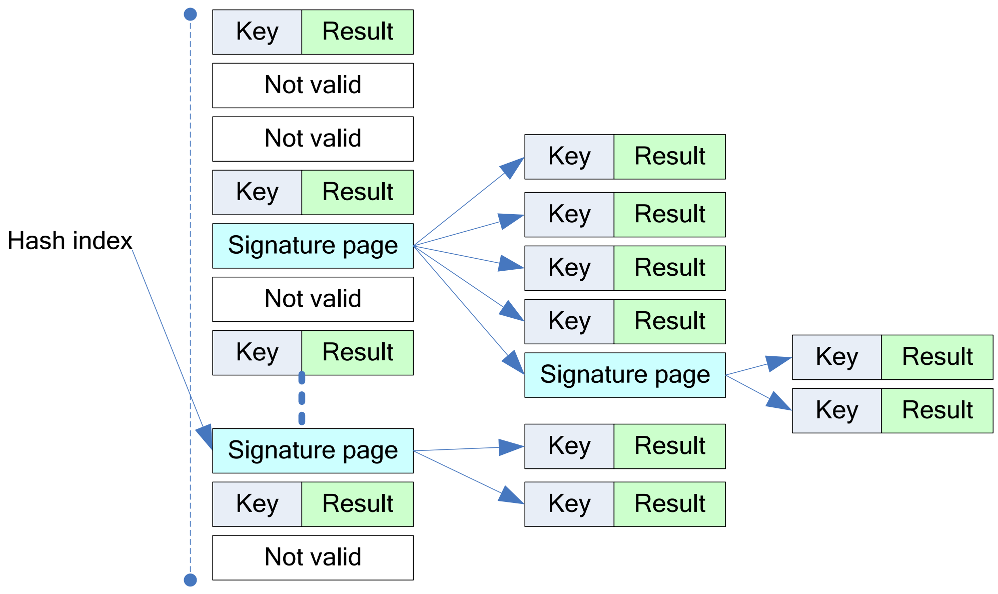
The hash function is used to compute index to array of elements (bucket).

The hash structure processing is performed by SW with assistance of dedicated instructions.

The hash structure defines three types of entries:

- Key/Result entry – This entry is hash leaf containing SW structure of key and result. The raw entry size can be 32B to 256B in 16B multiples. IMEM results are limited to 32 bytes maximum. The HW uses only embedded ECC bits.
- Signature Page entry – This entry is used for hash collision resolution. The structure is SW defined to contain up to four links to {Key, Result} and link to concatenated (next) signature page. The raw entry size is 32B. The HW uses only embedded ECC bits.
- Invalid entry – Same size as max(Key/Result, Signature page).

All raw entry sizes in memory must have the same size maximum ({Key, Result}, Signature page). All entries must have same memory alignment.



- Single cycle hash – When the hash index points to {Key, Result} or {Not Valid} entry, the hash lookup is completed in a single lookup cycle.
- Multi-cycle hash – When the hash index points to a signature page entry, the hash lookup is repeated until the {Key, Result} leaf is read.

The hash structure is usually dynamic regarding the number of entries and entry data.

When the portion of the hash's Not-valid entries, indexed by Hash Index is relatively low, there is a higher probability that insertion of {Key, Result} entry will collide with an existing entry. Such collisions are resolved by signature page entries. The signature page holds **up to four** potential matched elements. For each potential element, the signature page stores a pointer (Search Descriptor) to its {Key, Result} entry and a hash2 result on the same key. The hash2 value is then used to filter out some of the potential elements. Signature pages can be concatenated to form a linked list in cases of more than four potential matches.

12.8.1 The {Key, Result} Hash Entry

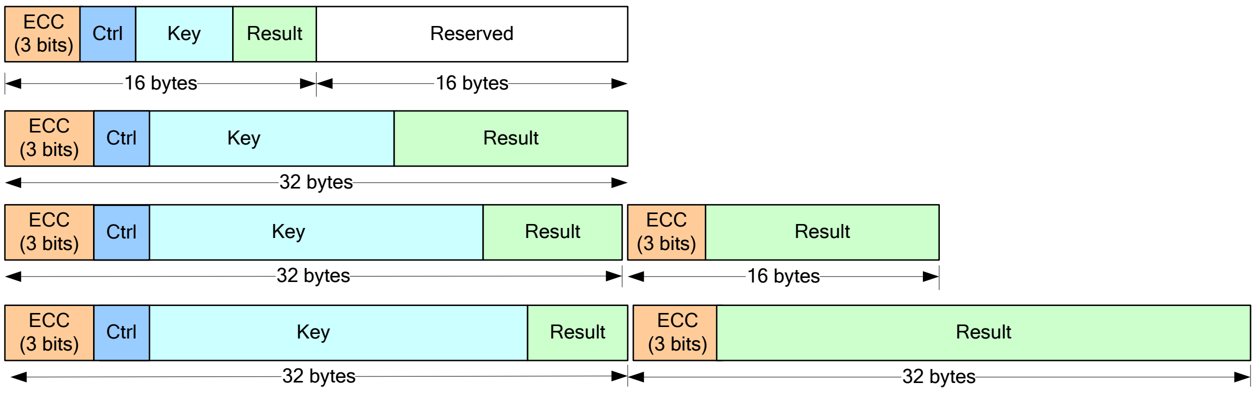
This entry is a hash leaf containing SW defined structure of key and result.

The raw entry size can be 32B to 256B in 16B multiples. IMEM results are limited to 32 bytes maximum.

The HW uses only embedded ECC bits. The ECC consumes 3b from 32B or 16B.

The hash entry's usual control fields (refresh, age, valid, etc.) are SW allocated and managed. It is advantageous to allocate control fields in the first 16B of the entry structure.

Figure 12-11. {Key, Result} hash entry



12.8.2 Signature Page Hash Entry

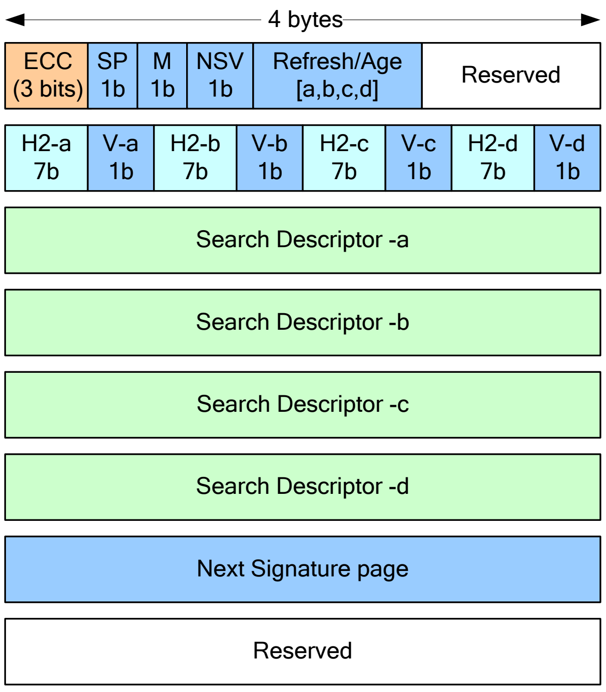
The signature page is SW defined and managed. The raw entry size is 32B. The HW uses only embedded ECC bits. The ECC consumes 3b from 32B or 16B.

The signature page implements secondary hash structure of 4 entries and implements a bucket chaining method. The hash entry's usual control fields (refresh, age, valid, etc.) are SW allocated and managed. It is advantageous to allocate control fields in the first 16B of the entry structure.

Signature page fields:

- SP – Indicates this is signature page entry
- M – Match
- NSV – Next Signature page pointer is valid
- Refresh[a,b,c,d] – Entries refresh bits
- Age[a,b,c,d] – Entries age bit
- H2[a,b,c,d] – Secondary hash values
- V[a,b,c,d] – Entry valid bits
- Search Descriptor [a,b,c,d] – The pointer to {Key,Result} entry
- Next signature page pointer

Figure 12-12. Signature page hash entry



12.8.3 Hash Lookup Acceleration Instructions

The CTOP supports dedicated instructions to accelerate the hash index calculation:

- Hash – Core level implemented instruction of hash of up to 8B.
- Whash – Wide hash of up to 64B. Cluster level implemented instruction.
- Mcmp_i – Data compare in CMEM, used for hash entry key compare.

12.8.3.1 Hash

Core level hash is used for keys of up to 8B. The instruction execution is one core clock.

The 8B key is passed via the CTOP registers.

The hash index result is passed via registers.

An immediate value to the hash instruction selects the hash mode and supplies additional arguments to the hash computations. It includes, key bit size and output bit size, both values between 1b-32b, which defines pre- and post-hash masking to determine the input/output bit widths. In addition, the immediate specifies a selection of hash functions from a wide range of linear and non linear combinations.

12.8.3.2 Wide Hash

Cluster level hash accelerator (whash) is for larger keys of up to 64B. The CTOP thread is scheduled while waiting for instruction to complete.

The key is taken from the CMEM location. The key size of 1b-6b is an instruction immediate value.

The hash index result is 32b.

12.8.3.3 Hash Key Compare Instructions

The CTOP supports dedicated instructions for hash key compare.

The *mcmp_i* instruction performs data compare in CMEM from 1B to 255B.

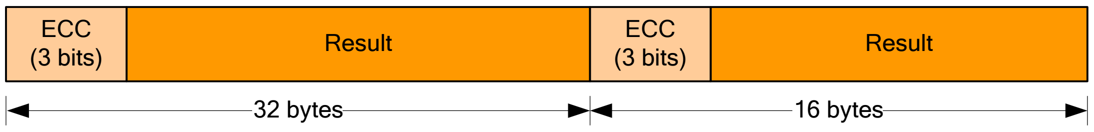
12.9 TCAM Look-up

See the *TCAM Interfaces* chapter.

12.10 ECC Protection

The ECC methodology that is used for the search structures is 3 parity bits per 32B.

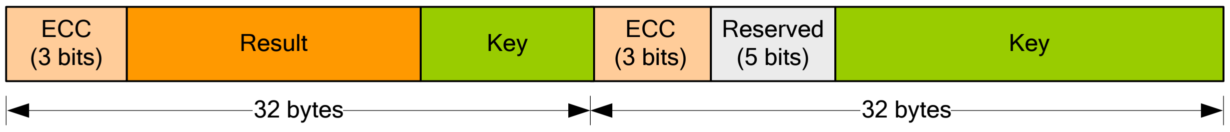
This means that for large result sizes (>32B) some of the ECC bits are embedded within the result data.



In such cases where there are multiple ECC fields, additional ECC fields of the **key data** are considered to be 1B long, to ease the parsing of the result later on. Practically, the same 3 parity bits are used to protect the key but padded with an additional 5 bits, to start the key with 1B alignment. (This is relevant for hash structures only when the result includes the key.)

There is a special support within the **mcmp_i** instruction to skip these additional ECC fields.

The following figure shows a single cycle hash memory format with a result size of 64 bytes:



13. Deep Packet Inspection (DPI)

This section provides a high-level description of the Deep Packet Inspection (DPI) capabilities of the NPS-400.

Deep Packet Inspection is commonly implemented in appliances, service line cards and integrated service routers as an infrastructure for Layer 7 (L7) features such as firewalls and Intrusion Pretension Systems (IPS). In comparison to traditional L2-4 functionalities DPI is extremely stateful (i.e. memory intensive), requires access to large parts of the flow (i.e. L3/L4 headers and L4 payload) and carries a non-uniform cost making the scaling up of a DPI solution an extremely challenging feat. EZ-DPI is an NPS-400 optimized DPI engine that is designed to address the inherent DPI challenges utilizing the NPS' massive parallel architecture and hardware accelerations.

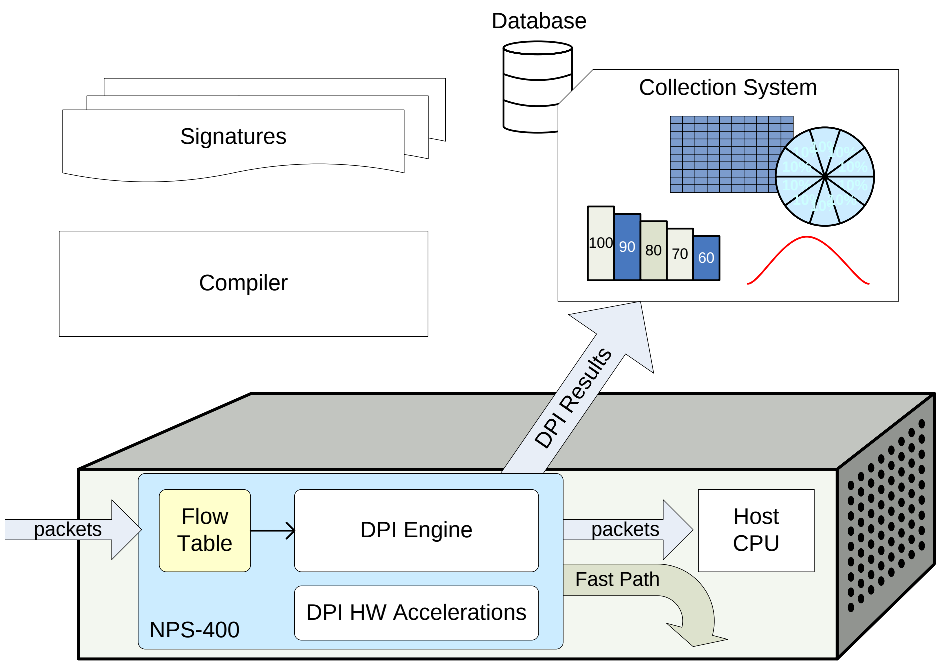
Initially EZ-DPI is centered on application recognition (AR). Hardware accelerated content filtering is also provided and described below.

13.1 What is DPI?

DPI is a broad name commonly referred to Layer 7 traffic classification and inspection. DPI classification is commonly used as an infrastructure for L7 services such as application recognition, security threat identification, L7 load balancing, L7 based tiered services and more.

In general DPI capabilities vary between solutions. In some cases the DPI is required to map flows to one of a set of known applications or threats, whereas in other cases it is also required to provide field extraction (e.g. URL required for URL filtering) while in other more demanding cases it is also required to identify important protocol events (e.g. identify each HTTP request separately within a flow). The DPI's L7 classification must be cross packet and involve numerous classification technologies.

Figure 13-1. DPI architecture



The architecture of a DPI technology can be separated into two: The data path which sets the performance and the other non-runtime path which includes signatures and a compiler which dictate the eventual fidelity of classification.

DPI signatures which are not limited to RegExp signatures are data files describing application and security threats. DPI signatures are typically implemented as data files which are compiled and loaded into the network element. This is done since the rate of change of these signatures is much larger than the ability of vendors to release new software version of the network elements. Signatures are often compiled off-box so that a combined efficient classification tree is created. The signature compiler typically does not create a run code but rather another intermediate data representation of the signatures. That intermediate representation is made of building blocks that are tuned to the data path for efficiency. The intermediate representation also assumes an underling computational model. At the data path, the compiled signatures are loaded and run in a DPI engine. The DPI engine classifies flows based on the signatures and is a client of the flow table.

EZchip is *not* a signature company and will rely on partners to deliver signatures. On the other hand, EZchip is an expert in creating state of art, wire-speed, data-path technologies. The NPS-400 architecture and hardware accelerators support DPI at rates up to 800Gbs. EZchip initially provides an optimized DPI engine and compiler providing Application Recognition on the NPS-400. Future software versions should add additional HTTP and SIP parsing capabilities.

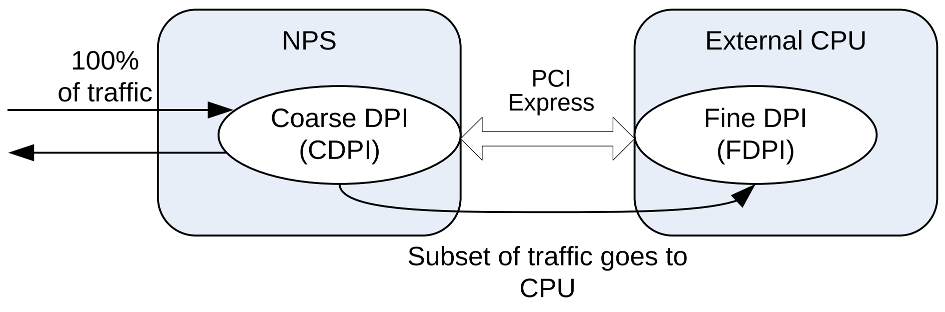
13.1.1 DPI Performance

DPI is an infrastructure capability that deals with L7 classification. Various DPI based features vary in the type of signatures they detect, in the set of applications they are interested in, and in the depth of classification required. For example, an application recognition solution may be required to classify all flows but typically requires around three payload packets to reach the conclusion; all the other packets may be bypassed. Such a solution may scale up to 800Gbps running on NPS-400.

At the other extreme, solutions such as IPS start with application recognition and follows it with possibly thousands of specific signatures addressing vulnerabilities of that application. This means that some flows may require a large amount of further processing. Other solutions, such as WOC, may do plenty of L7 computation on a limited set of applications.

An architecture that allows the NPS-400 to take a major role in DPI even when the amount of computation is large involves separation of the DPI into two parts. The first part runs on the NPS-400 while the second higher resolution classification runs on an external CPU. The two parts communicate through the NPS' PCI Express bus as shown in the figure below.

Figure 13-2. EZchip DPI architecture with external CPU



Depending on the nature of the DPI classification, the NPS may be able to handle all the classification. If this is not the case then the NPS' coarse DPI (CDPI) module would provide the initial coarse classification for some flows which would be later sent to the fine DPI (FDPI) module for further processing, while the rest of the flows would be processed only by the NPS. This architecture would allow an NPS based solution to scale even for highly demanding DPI classification services.

Readers should note that the term *coarse* as used here does not indicate that the results produced are incorrect, instead it is used to indicate that a finer-grained deeper classification is available and may be required.

13.2 Application Recognition

Application recognition (AR) is a DPI based functionality which aims at classifying flows to applications and protocols. State of the art AR solutions can typically correctly classify around 1500 different applications and protocols.

Note: The terms application *and* protocol *are used interchangeably in this section.*

To achieve proper fidelity, AR solutions deploy multiple classification technologies including:

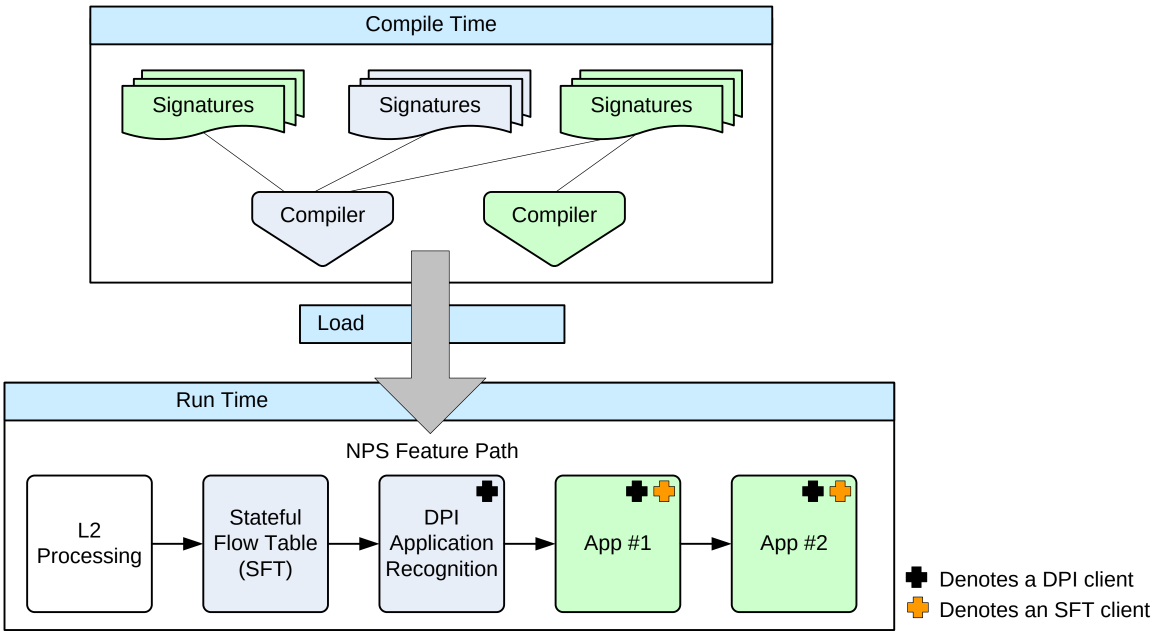
- Cross packet RegExp classification
- Heuristical classification
- Behavioral classification
- Stateful parsing and multi flow session classification
- L3\L4 packet based classification
- Other encoded non RegExp based signatures (e.g. ASN1)
- And more

In some products AR provides the full DPI needs. This is true for services such as L7 network visibility, tiered services, application firewall, and L7 router policies. In others products, AR is the first step for further L7 classification. For example SNORT applies IPS signatures based on the nature of the L7 application. EZchip has chosen to initially focus the NPS-400 DPI solution on AR since it is the common ground for most L7 classification. Future versions will gradually add further metadata capabilities to some of the parsers.

The AR output is a uint16 AppId that uniquely identifies the specific flow's application or protocol. The AR solution also provides a file that maps the result (i.e. AppId) into categories allowing vendors to deploy business logic and policies on groups of applications rather than on each separately. For example, a policy can be applied to shape down all the flows identified as a Peer to Peer (P2P) application instead of applying a similar policies separately to each of the 100+ such applications that are mapped to this category.

The EZchip AR solution runs completely on the NPS and does not require an external CPU.

Figure 13-3. Application recognition solution



EZchip expertise is in the hardware accelerated NPS data path which applies to all features including EZ-DPI. Signatures may be provided by EZchip's partners or by the NPS customers with DPI technology of their own. The partner signatures would enjoy an optimized DPI engine developed by the partner in conjunction with EZchip. The term *EZ-DPI* refers to the DPI engine that EZchip developed which would be accompanied by an EZ-DPI AR compiler. Vendors would have the ability to use the partner optimized AR solution, to use the EZ-DPI as is, or to use EZ-DPI as a reference implementation and optimize their code based on it.

DPI is a Layer 7 computation. To compute at Layer 7 one must be able to properly process the lower layers and abstract as much as possible the packet-ness providing a stream based flow computation model to the DPI engine dealing with issues such as duplicated packets, out of order, fragmentation, tunneling and more. Failing to do so ahead of the DPI engine would complicate that engine and obstruct efficiency of code and implementation. In a massively parallel processor such as the NPS-400, memory efficiency and the ability to not use software flow locks is critical for line rate performance of AR. The NPS-400's run to completion model coupled with the efficiency of the stateful flow table (SFT) provide the computational model required by the DPI engine.

The EZ-DPI engine is a client of the SFT and located right after it in the feature path. As a client of the SFT, the DPI engine is guaranteed that only a single thread is processing the flow at any given moment. When a packet is received by the DPI engine it is already mapped to a flow even if it is tunneled or NATed. The current thread DPI engine gets the current flow status and will store its AR result (i.e. DPI AppID) in the SFT at which point it would use the SFT Bypass_Flow() API so that no further DPI processing is performed. The DPI engine would typically also set the SFT's flow aging time based on the AppId.

The DPI engine is highly stateful; it uses the SFT's client object to store its memory such as the traffic pointer. The client object is retrieved automatically for it by SFT prior to it processing the packet. For multi flow applications such as FTP, BitTorrent and SIP, the session SFT super context is used and the DPI introduces the negotiated flows to the table prior to the actual arrival of these packets. All flows belonging to such a session would be mapped based on the session simplifying the way policies are applied. For example, the RTP and SDP flows belonging to a SIP session will be marked as SIP so that policies can be applied intelligently.

The DPI engine primitives are described in a the DPI engine section of the Application specification. Among these are multi-string search API routines. The non-anchored variation of this primitive would use the hardware accelerated Bloom filter capability described in the next section. In particular, the HTTP parser would use that capability to scan deep into the request in order to identify protocols using HTTP as a transport layer and identify them through unique headers and\or specific values in the URL and User_agent fields.

13.3 Bloom Filter based Content Filtering

Bloom filter is a space-efficient probabilistic data structure that is used to test whether an element (e.g. a string or a RegExp Pattern) is a member of a set (i.e. a packet). False positive matches are possible, but false negatives are not. In other words, a query returns either "possibly in set" or "definitely not in set". The more elements that are added to the set, the larger the probability of false positives.

The NPS-400 has dedicated hardware acceleration for such Bloom filter based content filtering.

This capability may be used in two separate ways:

- As part of a larger DPI solution: For example it he NPS-400 AR solution this is used internally within the HTTP parser to look for TTP request headers, URLs, and user agent based application recognition.
- For some markets the requirement to look for rarely seen non-anchored (i.e. anywhere in the packet) strings with no false negatives is all that is required. An example of such a market is a service to look for strings such as “Bin Laden” or “Bomb” somewhere in the traffic. The structure, features and performance of EZchip’s Bloom filter based content filtering solution are introduced in the following sub sections.

13.3.1 Architecture

The NPS content filtering solution consists of four parts which together scale up to 200Gbps of CPDI content filtering.

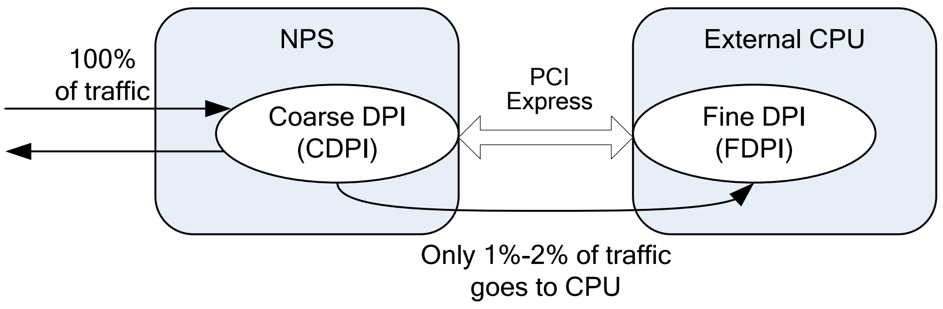
The first part is a set of Perl-Compatible Regular Expressions (PCRE) signatures. The expectation is that these will be provided by the vendor. The second part of the solution is a PCRE compiler which extracts a unique relatively rare sub-string for each PCRE signature. The compiler runs on a server and uses a dictionary to pick the best sub-string for each such PCRE signature. The compilation results in two databases which are loaded into the NPS. The first database includes the sub-strings indicating the possible existence of a signature while the second database is used for further filtering of the identified suspicious sub-string.

The first database is used by the NPS data-plane hardware accelerators to identify suspicious string matches at a high rate. The second database is used by the NPS data-plane SW running on the NPS to further filter the suspicious packets identified by the hardware. Together the HW and SW data-plane functions comprise the CDPI part of the content filtering solution. If the PCRE patterns are simple enough, all of this can be performed by the CDPI, otherwise the packets that were partially identified by the CDPI will be passed to the FDPI for further processing. The FDPI will typically be running a vendor supplied pre-compiled set of Deterministic/Nondeterministic Finite Automaton (DFA/NFA) state machines. These automata are invoked upon triggers from CDPI.

Using this approach, the NPS-400 is capable of up to 200Gbps of CDPI filtering.

The idea is to perform an initial scan of the payload data in order to filter the possible positive matches of data against the rules and significantly reduce the amount of data that needs to be processed by the content filtering solution.

Figure 13-4. Content filtering architecture with CDPI pre-filter



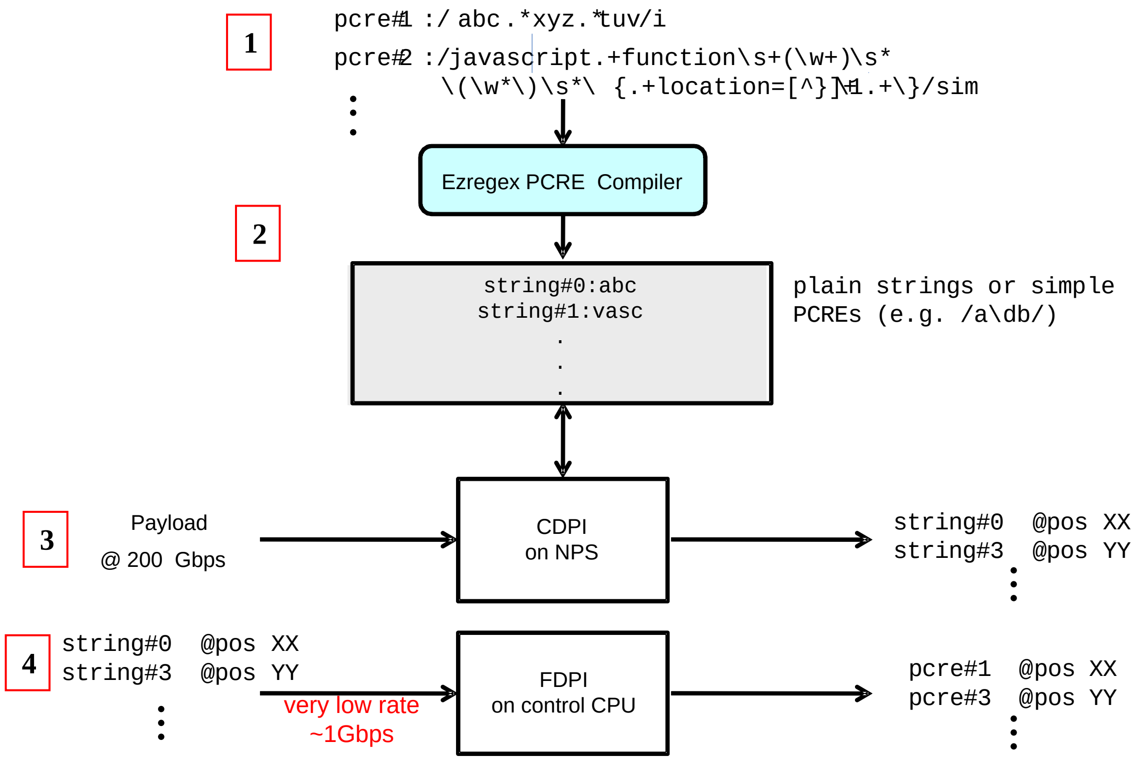
- *For more information see the CDPI Detailed Implementation section.*

13.3.2 Detailed Content Filtering Computation Sequence

The content filtering sequence is listed below. The numbers relate to the Figure 13-5 below.

1. For example, for a PCRE database with the regular expressions pcre#1, pcre#2, etc.
2. The EZregex PCRE compiler builds a database for the NPS consisting of one sub-string for each PCRE expression, such as string#0: abc, string#1: vasc, etc., which are going to be used for the CDPI running on the NPS. The EZregex also builds a complete PCRE database for the control CPU that is augmented with suitable DFA/NFA automatons for each PCRE string.
3. The CDPI on the NPS inspects the packet payload for up to 1Gbps traffic per CTOP, and aggregated performance of up to 200Gbps overall, and reports every sub-string match and its position in the packet payload.
4. The suspicious traffic, which is typically only a few Gbps, is sent to the optional external control CPU for FDPI that completes the PCRE matching based on the information from the CDPI. The FDPI reports the PCRE match and its position in the packet payload to the DPI user application.

Figure 13-5. Sample of coarse and fine DPI overview



13.3.3 Content Filtering Features

EZchip's NPS content filtering technology offers:

- Up to 200Gbps content filtering, with at least 1Gbps single-thread DPI filtering per CTOP.
- Configurable assignment of processing cycles to content filtering on a per CTOP HW thread level. A single CTOP can simultaneously process content filtering and run other packet processing or control plane code on different HW threads.
- Parallel matching of hundreds of thousands Perl-Compatible Regular Expression (PCRE) rules.
- Stateful cross-packet inspection, i.e. start and end points for PCRE matches can be in different packets.
- Easily-customized, SW-based DPI implementation, augmented with ISA extensions and HW accelerators.
- EZregex, EZchip PCRE compiler, creates the CDPI string matching database for the NPS and the corresponding full PCRE database for the control CPU.
  - NPS character-set and sub-string matching using Bloom filter, saving memory space and execution time.
  - Control CPU selective PCRE matching triggered by NPS sub-string matches, using pre-compiled suitable Deterministic/Nondeterministic Finite Automaton (DFA/NFA).
- NPS offloads the control CPU by filtering incoming wire-speed traffic, so that only a few percent of the traffic needs inspection by the control CPU
- On-chip zero false-negative and controlled false-positive scanning of strings and fixed length patterns anywhere in packet payload, including fragmented packets.
  - Searches for a sub-string can be anywhere within the PCRE (optionally case sensitive).
  - On-the-fly detection of UTF8 (non-ASCII) encoding.
  - Supports in-service software update (ISSU) including the PCRE database.
- Packet flow awareness
- The content filtering SW package is supplied by EZchip for both NPS and its external/internal control CPU.

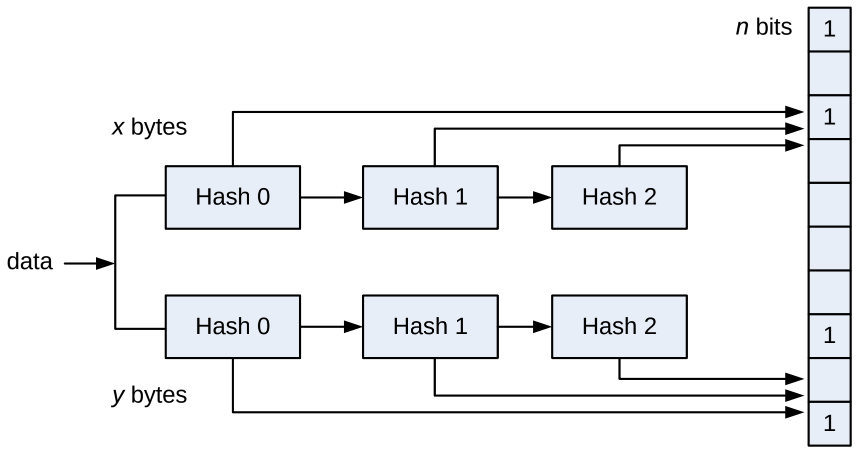
13.3.4 CDPI Detailed Implementation

The CDPI implementation is SW-based, augmented with ISA extensions and HW accelerators, and can easily be optimized for various applications. The content filtering is using a Bloom filter for the CDPI running on the NPS, saving memory space and execution time.

The Bloom filter algorithm works as follows:

- Divide strings to length x (short), y (normal), e.g. x=3, y=6
- On each path:
  - Compute Hash0, access Bloom filter
  - If match, compute Hash1, access Bloom filter
  - If match, compute Hash2, access Bloom filter
  - If match, compare data to exact string -> report FDPI
- Advance +1 byte

Figure 13-6. CDPI via Bloom filter



CDPI running on the NPS may be sufficient for full inspection of simple PCRE expressions and there is no need for FDPI running on an external control CPU.

13.3.5 EZ-ISA Instructions Relating to Content Filtering

Listed below are the NPS EZ-ISA instructions related to DPI content filtering:

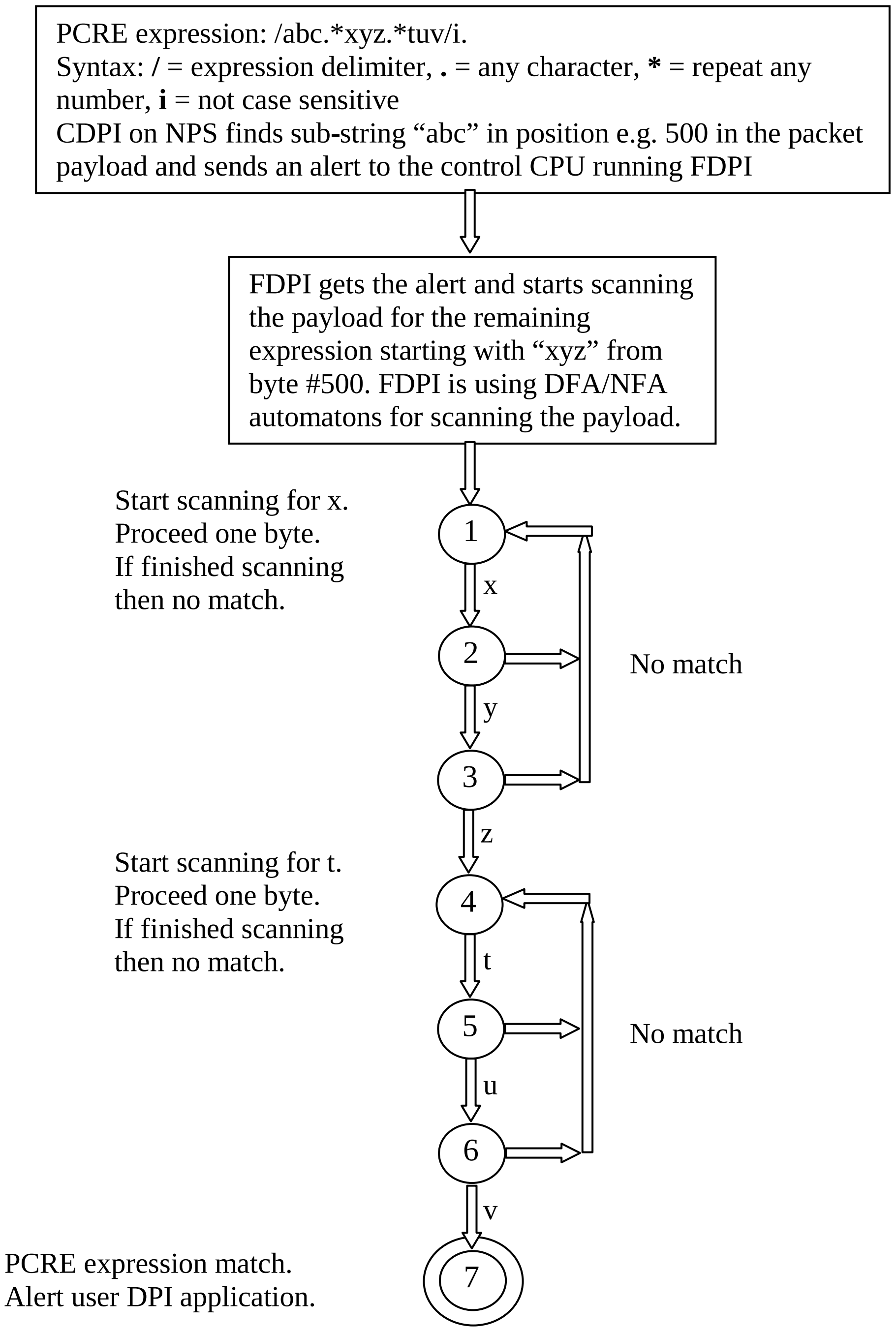
- Hash – 16 general-purpose hash functions
- Hash.p0 to hash.p3 – DPI-specific hash functions
- TR = Translate, perform lower-upper case conversion and simple PCRE matching
- UTF8 – acceleration for UTF8 codeword detection Addf – count CTOP flags. Used in conjunction with UTF8 to improve UTF8 detection.
- ldBit load bit – bit-resolution load operation for Bloom filter matching
- ldbit.x2,ldbit.x4 load 2 or 4 bits – double/quad bit load for advanced Bloom filter matching

All the instructions will be described in detail in the *NPS CTOP Instruction Reference Manual*.

13.3.6 Fine DPI (FDPI)

The FDPI user application runs on the control CPU uses the EZchip content filtering APIs to get the alerts from the NPS CDPI matches and complete the PCRE matching using any user-proprietary regular expression library's full matching functions. The content filtering reference FDPI application uses the open source RE2 regular expression library (http://code.google.com/p/re2/).

Figure 13-7. FDPI DFA/NFA automaton


PCRE expression: /abc.*xyz.*tuv/i.
Syntax: / = expression delimiter, . = any character, * = repeat any number, i = not case sensitive
CDPI on NPS finds sub-string "abc" in position e.g. 500 in the packet payload and sends an alert to the control CPU running FDPI
FDPI gets the alert and starts scanning the payload for the remaining expression starting with "xyz" from byte #500. FDPI is using DFA/NFA automatons for scanning the payload.
Start scanning for x. Proceed one byte. If finished scanning then no match.
1
x
2
y
3
No match
z
Start scanning for t. Proceed one byte. If finished scanning then no match.
4
t
5
u
6
No match
v
PCRE expression match. Alert user DPI application.
7

13.3.7 Performance

The NPS-400 DPI performance is up to 1Gbps per CTOP with a graceful degradation when the rate of string matches increases, due to accessing external SDRAM. A configurable number of the 256 CTOPs can be allocated for performing content filtering. Free CTOP HW threads can also be used for L2-L3 packet processing.

The CDPI high filtering rate is shown in the example below:

- Snort DB v.2904 (www.snort.org)
  - 100KB of plain text: 9 alerts to FDPI
  - 100KB of binary data such as an mp3 file: 4 alerts to FDPI
  - Filtering rate is better than 0.01%

14. Cryptographic Features

This section describes the cryptographic service accelerations provided by the NPS-400. The NPS provides HW implemented accelerations for common security cryptographic algorithms.

Important: ***The features outlined in this chapter are relevant only for NPS-400 models that support cryptographic features (order no. 207850xx).***

The security accelerators address the network application requirement to support reliable communications that guarantee integrity, confidentiality, authentication and anti-replay protection. The acceleration provides for a high bandwidth implementation of IPSec/SSL/TLS applications that require heavy encryption/ decryption processing.

The NPS integrates several acceleration engines per each CTOP cluster. The security processing can be designed as either run-to-complete or pipelined. The run-to-complete model executes whole flow processing on the same CTOP thread utilizing 4K threads. The pipeline mode breaks the flow processing to stages and assigns stages to CTOP threads.

The table below lists HW accelerator instances for supported ciphers and hash functions.

The key processing relies on external management CPU.

STANDARD	*MODES*	*NUMBER OF INSTANCES PER CLUSTER*	*TOTAL INSTANCES*
Ciphers			
AES 128/192/256 bit	ECB, CBC, CTR, CCM-CMAC, OFB, CFB, GCM-GMAC	3	48
3DES (DES)	ECB, CBC, CTR, OFB, CFB	3	48
Hash functions			
MD5	Raw, HMAC	3	48
SHA-1		3	48
SHA-2 224-256, 318-512			
Key processing			
RSA	External management CPU		
ECC			
TRNG			

The crypto HW accelerator engines operate at NPS core-clock/2.

14.1 Ciphers

This section details the accelerator engines for ciphers:

- AES – Advanced Encryption Standard
- 3DES – Triple Data Encryption Standard

14.1.1 Advanced Encryption Standard (AES)

14.1.1.1 Standards Support Summary

NPS supports the following industry standards:

- FIPS 197 – Advanced Encryption Standard, National Institute of Standards and Technology, 2001.
- SP800-30A – Recommendation for Block Cipher Modes of Operation, National Institute of Standards and Technology, 2001. Compliant modes (ECB, CBC, CFB, OFB, CTR).
- SP800-38B – Recommendation for Block Cipher Modes of Operation: The Compliant CMAC Mode for Authentication, National Institute of Standards and Technology, 2005.
- SP800-38C – Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality, National Institute of Standards and Technology, 2004.
- SP800-30E – Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, National Institute of Standards and Technology, 2007.
- RFC 3566 – The AES-XCBC-MAC-96 Algorithm and Its Use With IPSec, Network Working Group, 1996
- Context switching for all modes including (new) GCM and CCM
- Compatible with standards including IEEE 802.1ae, IEEE 802.11i and IEEE 802.16e

The AES is a 128-bit block cipher defined by FIPS 197. AES performs encryption of a plain text message by repeatedly applying groups of exclusive-OR operations with key material, substitutions, and mixing operations. Each group of these operations is called a **round**, and the number of rounds executed will depend upon the amount of key material supplied for the operation, and are summarized in key-size/rounds table below. For decryption, the inverses of each operation performed in the encryption are performed, and these operations require the same computational effort as in the encryption operation, so AES is a symmetric cipher.

Table 14-1. AES key-size/rounds table

KEY SIZE [B]	*ROUNDS*
128	10
192	12
256	14

14.1.1.2 Key Schedule Generation

The AES uses a key size of 128, 192, or 256-bits. The key is used to generate a sequence of data that is used in each round of the encryption. This sequence of data is called a **key schedule**, and the process of generating the key schedule from the key is called key **expansion**.

The key expansion opcode activates the AES to expand the key. The key is expanded by encrypting 16B constant $0x01...01_{128}$ with a given key.

14.1.1.3 Operating Modes

The AES may be used with several different operating modes as defined in NIST SP800-38A, SP800-38B, SP800-38C, SP800-38D, and SP800-38E. The attributes of these modes are summarized in the table below. The ECB mode is implemented by directly using the AES algorithm defined in FIPS 197. Significantly, the ECB/CTR modes are stateless in that each encryption and decryption operation is independent. The other modes of operation are stateful in that they require an initial state vector (value) ("IV - initialization vector/value"), and the results from the prior operation are used to produce the results for the next operation. Implementation of the ECB and CBC modes for both encryption and decryption requires that the AES cipher implementation be capable of performing both encryption and decryption, whereas the other modes only require AES cipher encryption, both for encryption and decryption. While the ECB, CBC, CFB, OFB and CTR modes provide confidentiality (encryption), the CCM and GCM modes provide both confidentiality and integrity by adding a message authentication code (MAC). CMAC, XCBC, and GHASH only provide integrity.

Table 14-2. Operating mode attributes mode chaining enc/dec req. initialization vector standard

MODE	*CHAINING*	*ENCODE/ DECODE*	*INITIALIZATION VECTOR*	*STANDARD*
ECB- Electronic Codebook	N	Enc/Dec	N	FIPS 197
CBC - Cipher Block Chaining	Y	Enc/Dec	Y	SP800-38A
CFB - Cipher Block Feedback	Y	Enc only	Y	SP800-38A
OFB - Output Feedback	Y	Enc only	Y	SP800-38A
CTR - Counter	N	Enc only	Y	SP800-38A
CMAC-Cipher Based MAC	Y	Enc only	Y	SP800-38B
XCBC- Extended CBC-MAC	Y	Enc Only	Y	RFC 3566
CCM - CTR/CBC-MAC	Y	Enc only	Y	SP800-38C
GCM - Galois/Counter Mode	Y	Enc only	Y	SP800-38D
GHASH - Galois Hash	Y	N/A	Y	SP800-38D

14.1.1.4 AES Engine Performance

Each of the 48 engines support the following listed performance per AES key size:

ENGINE	*DATA (BITS)*	*KEY (BITS)*	*LATENCY CYCLES PER DATA*	*BIT/CYCLE*	*GBIT/SEC @ 500MHZ: BIT/CYCLE*CLOCK RATE*
AES	128	128	10 + 1	11.6	5.8
		192	12 + 1	9.8	4.9
		256	14 + 1	8.5	4.25

500MHz refers to engine clock when NPS core clock is 1000MHz.

14.1.1.5 AES Block Size Constraints

MODE	*NIST REQUIREMENT*	*NPS CIPHER SUPPORT*	*COMMENTS*
ECB, CBC	The size of the plaintext must be a multiple of the fixed size "Cipher block size".	"Cipher block size" = 16B. Buffer size K*16B.	
CCM			SW formats number of blocks of Plaintext, Nonce, and Additional Data to "B blocks" for cipher.
OFB, CTR, GCM	The Plaintext can be of any byte size. (Plaintext blocks -1)*(block size) + (last block). 1B≤ (last block) ≤ (block size)	"Cipher block size" =16B. Last buffer size= 1B ≤ u ≤ 16B.	
CFB	Plaintext size = K*s. 1≤ s ≤ block size.	s = 16B	SW adjustment when s ≠ 16B,

14.1.2 Triple Data Encryption Standard (3DES)

The 3DES is a 64-bit block cipher defined by FIPS 46-3. 3DES performs encryption of a plain text message by repeatedly applying groups of exclusive-OR operations, substitutions, and mixing operations combined with key material. Each group of these operations is called a **round**, and 3DES encryption and decryption requires sixteen rounds.

Features:

- FIPS 46-3 compliant DES/3DES engine. Supports FIPS 81 defined operating modes.
- Supports key sizes of 56, 112, and 168-bits
- FIPS 46-3 TDEA Keying Options 1, 2, and 3

Triple Data Encryption Algorithm (TDEA):

The FIPS 46-3 defined TDEA, also referred to as Triple DES or 3DES, makes DES usable for the near future. Encryption of a 64-bit message *m* with TDEA is performed according to:

$c = E_{K3}(D_{K2}(E_{K1}(m)))$
where $_K1$, $_K2$, $_a$nd $_K3$ are 56-bit keys, E() denotes DES encryption, D() denotes DES decryption, and c is the resulting cipher text.

Similarly, TDEA decryption of a 64-bit block of cipher text is performed according to:

$c = D_{K1}(E_{K2}(D_{K3}(c)))$

The TDEA defines three keying options, which are summarized in the table below. To perform a single DES execution, the 3DES core needs to be used with keying option 3 as seen in the table below:

KEY OPTION	KEY BIT SIZE	DESCRIPTION
1	168	$_K1$, $_K2$, and $_K3$ are independent keys
2	112	$_K1$ and $_K2$ are independent keys; $_K1=_K3$
3	56	DES: $_K1=_K2=_K3$

14.1.2.1 Operating Modes

The DES/3DES may be applied with several different operating modes as defined in FIPS 81. The attributes of these modes are summarized in the table below. The ECB mode is implemented by directly using the DES/3DES algorithm defined in FIPS 46-3. Significantly, the ECB/CTR modes are stateless in that each encryption and decryption operation is independent. The other modes of operation are stateful in that they require an initial state vector (“initialization vector”) and use the results from the prior operation to produce the results for the next operation. Implementation of the ECB and CBC modes for both encryption and decryption requires that the DES/3DES implementation be capable of performing both encryption and decryption, whereas the CFB and OFB modes only require DES/3DES encryption, both for CFB and OFB mode encryption and decryption.

MODE	CHAINING	ENCODE/DECODE REQUIREMENT	INITIALIZATION VECTOR
ECB - Electronic Codebook	N	Enc/Dec	N
CBC - Cipher Block Chaining	Y	Enc/Dec	Y
CFB - Cipher Block Feedback	Y	Enc only	Y
OFB - Output Feedback	Y	Enc only	Y
CTR- Counter	N	Enc/Dec	N

14.1.2.2 3DES Engine Performance

For 3DES engine operations the latency is 32 NPS core clock cycles. This means that each of the 48 3DES engines can operate with a throughput of 1.66Gbps at 1000MHz NPS core-clock, maximum total of 80Gbps.

The reset operation does not reset the 3DES data path. For applications that require neutralization of any residual data in the data path, it is necessary to perform an operation using non-critical data and key material.

14.1.2.3 DES/3DES Block Size Constraints

MODE	NIST REQUIREMENT	NPS CIPHER SUPPORT	COMMENTS
ECB, CBC	The size of the plaintext must be a multiple of the fixed size "Cipher block size".	"Cipher block size" = 8B. Buffer size K*8B.	
OFB, CTR	The Plaintext can be of any byte size. (Plaintext blocks -1)*(block size) + (last block). 1B≤ (last block) ≤ (block size)	"Cipher block size" =8B. Last buffer size= 1B ≤ u ≤ 8B.	
CFB	Plaintext size = K*s. 1≤ s ≤ block size.	s = 8B	SW adjustment when s ≠ 16B, Byte resolution.

14.2 Hashing Algorithms

A hash function is an algorithm that transforms (hashes) an arbitrary set of data elements, such as a text file, into a single fixed length value (the hash). The computed hash value may then be used to verify the integrity of copies of the original data without providing any means to derive said original data.

14.2.1 Message Digest 5

NPS features:

- RFC 1321 compliant MD5
- Used by hashed message authentication code (HMAC) defined by FIPS 180-3
- Automatic message padding

MD5 processes a message of up to 2^{64}-1 bits to produce a 128-bit message digest. The digest enables the determination of the integrity of the message; any change in the message will, with a very high probability, result in a different message digest. This property is useful in the generation and verification of digital signatures, in message authentication codes, and in the generation of random numbers.

Note: MD5 is considered to be insufficiently secured.

Prior to MD-5 engine job activation the context must be loaded with Message bit size.

The data is fed to hash engine in blocks of 64B or 128B. The last data block 1B-128B is auto padded as indicated in Message length.

The SW should arrange the message as 64B/128B blocks and activate MD-5 job per block.

The first message segment should be marked with Init opcode option flag that causes the Hash to initialize the security context state.

The MD-5 engine updates the context Message bit size and MAC values, that SW can save for its next activation.

Performance

Performance is determined by the length of the message and the operation type. For a message of *N*-bits, the minimum latency of the MD5-engine is given by:

Roundup[(N+65)/512]68 + 1

The maximum processing performance of the MD5 is greater than 7.5-bits/engine-clock-cycle.

HMAC Key Load

Prior to starting a new HMAC operation, a key must be loaded. The key is loaded similarly to a block of data.

The MD5 supports HMAC key sizes up the block size of one message packet (i.e., 512 bits).

The MD5 supports HMAC keys up to the block size of the associated MD5; for keys larger than the specified block size, the key must be reduced to a single block key by the user using the procedure described in the standard (FIPS 198-1).

There are dedicated CTOP opcodes for HMAC first segment and last segment.

14.2.2 Secure Hash Algorithm/HMAC

NPS features:

- FIPS 180-3 compliant SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512
- FIPS 198-1 compliant HMAC-SHA

The SHA engine is a high-performance implementation of the secure hash algorithms (SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512) defined by FIPS 180-3 and the hashed message authentication code.

(HMAC) defined by FIPS 198-1. The secure hash algorithm processes messages of up to 2^{64} or 2^{128} bits to produce a message digest ranging from 160-bits to 512-bits. The digest enables the determination of the integrity of the message: any change in the message will, with a very high probability, result in a different message digest. This property is useful in the generation and verification of digital signatures, in message authentication codes, and in the generation of random numbers. The specific parameters of the SHAs are summarized in the table below.

Table 14-3. SHA parameters

ALGORITHM	MESSAGE SIZE (BITS)	BLOCK SIZE (BITS)	WORD SIZE (BITS)	MESSAGE DIGEST SIZE (BITS)
SHA-256	$<2^{64}$	512	32	256
SHA-512	$<2^{128}$	1024	64	512
SHA-1	$<2^{64}$	512	64	160
SHA-224	$<2^{64}$	512	64	224
SHA-384	$<2^{64}$	1024	64	256

HMAC-SHA augments the SHA algorithm with a secret key that enables both the authenticity and integrity of a message to be verified.

- **SHA-256 should be chosen in most cases** where a high speed hash function is desired. It is considered secure with no known theoretical vulnerabilities and it has a reasonable digest size of 32 bytes. SHA-256 uses 32 bit numbers and 64 rounds.
- **SHA-224** uses the same algorithm as SHA-256 (except for the initial seed values) simply truncating its output. It was created because its digest size has the same length as two-key Triple DES keys (http://en.wikipedia.org/wiki/Triple_DES).
- **SHA-512** is different, using 64 bit numbers and having 80 rounds. Its digest size is 64 bytes.
- **SHA-384** is the same as SHA-512, except for the initial seed values, but truncated to reduce its digest size.

14.2.2.1 Performance

Performance is determined by the length of the message and the operation type.

For a message of *N*-bits, the minimum latency for a **SHA** engine is given by:

Roundup[(N+P)/B] * k + 1

For a message of *N*-bits, the minimum latency for a **HMAC**-SHA MAC is given by:

(Roundup[(N+P)/B] +2)* k + 1

N - message bit size
B - block bit size (512b/1024b)
P, k - parameters are listed in the table below

The maximum processing performance for each SHA and HMAC-SHA algorithm is summarized below:

Table 14-4. SHA maximum processing performance

ALGORITHM	*P*	*BLOCK (BITS)*	*K*	*MAX THROUGHPUT (BITS/ENGINE CLOCK CYCLE)*
SHA-384	129	1024	89	11.5
SHA-512	129	1024	89	11.5
SHA-1	65	512	41	12.5
SHA-224	65	512	33	15.5
SHA-256	65	512	33	15.5

The SHA engine is designed to operate on a block or blocks of data. Prior to processing the data block(s), the engine must be configured for the operation that will be performed. Configuration comprises initiating the engine to process a new message or loading a prior context. Once the engine is configured, the engine can accept message data input. The SHA is designed to allow each block of data to be burst into the core in as few as sixteen engine-clock cycles.

Prior to SHA engine job activation, the context must be loaded with Message bit size.

The data is fed to the hash engine in blocks of 64B or 128B. The last data block 1B-128B is auto padded as indicated in Message length.

The SW should arrange the message as 64B/128B blocks and activate SHA job per block.

The first message segment should be marked with Init opcode option flag that causes the Hash to initialize the security context state.

The SHA engine updates the context Message bit size and MAC values, that SW can save for next activation.

HMAC Key Load

Prior to starting a new HMAC-SHA operation, a key must be loaded. The key is loaded similarly to the block of data.

The SHA supports HMAC keys up to the block size of the associated SHA; for keys larger than the specified block size, the key must be reduced to a single block key by the user using the procedure described in the standard (FIPS 198-1).

There are dedicated CTOP opcodes for HMAC first segment and last segment.

14.3 Security Accelerator Architecture

The crypto accelerator engines are integrated in the Network Processor Cluster (NPC) units.

The engines served by 16 CTOPs (256 threads).

The security unit supports 128 active security contexts.

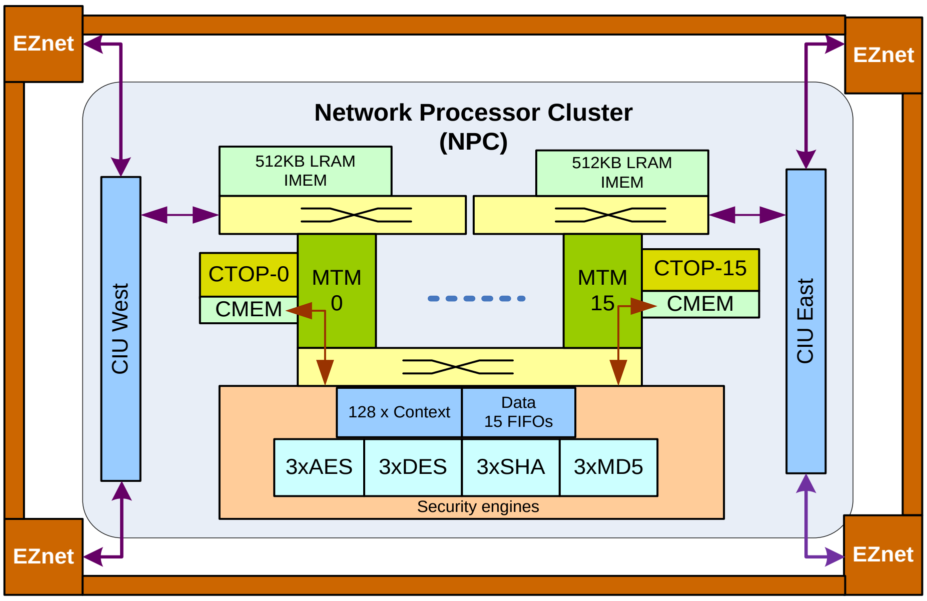
A per security engine FIFO is used for buffering incoming security data blocks.

The data block is processed per security task round.

The MTM is responsible for executing context/data copy from security unit to CTOP CMEM.

Each MTM supports 8B/clock BW towards the accelerator-shared crossbar interconnect.

Figure 14-1. Security accelerator architecture



The MTM is activated by security acceleration CTOP opcodes.

The SW manages the security context allocation for job execution. The thread that activated the security opcode is suspended until the job is executed.

14.3.1 Programming Model

The accelerators are activated by a list of dedicated commands:

OPCODE	COMMAND	EXECUTED
encr	Encrypt	MTM, Crypto engine
decr	Decrypt	
gcm.fin	AES finalize GCM	
shash	Secured HASH	
hmac	HMAC	
expskey	Expand key	
cpskey	Copy secured key to/from CMEM	MTM
cpssta	Copy secured state to/from CMEM	
cpsctx	Copy secured context to/from CMEM	
cpsiv	Copy secured IV	
cpsmac	Copy secured MAC	

The CTOP registers indicate to MTM the CMEM addresses for Key, input data block in CMEM, output data block in CMEM:

CTOP REGISTER	BYTE-0 [31:24]	BYTE-1 [23:16]	BYTE-2 [15:8]	BYTE-3 [7:0]
R1	Algorithm type	Data Size 1B-128B		Security Handle
R2/3			CMEM pointer to Key	
R4	CMEM pointer to output data block (destination)		CMEM pointer to input data block (source)	

*Data Size, Hash Init and Hash padding can be specified as opcode immediate parameters.

All security commands (except write commands) are multi-cycle commands that require the thread execution to be suspended (by next thread schedule) until command is executed.

The opcode specifies the cipher/hash algorithm, the number of keys and key size and the desired chaining mode.

Security encrypt code flow example:

STEP	OPCODE	DESCRIPTION
1		Prepare security context in CMEM.
2	cpskey	Copy security key from CMEM to accelerator context handle.
3	cpsiv	Copy security IV from CMEM to accelerator context handle.
4	#loop#	Prepare (128B) data block in CMEM. Pad last data block.
5	encr	Encrypt block of data.
6	sched	Wait until done (optional).
7		Move ciphertext from CMEM. Go to #loop#

The thread schedule-out to wait for block end of processing is optional, since the security engine can queue several jobs to the same handle.

At the end of hash operation, the 'digest' result needs to be copied from the security context to CMEM by the 'cpsmac' opcode.

14.3.2 Security Acceleration Unit Architecture

The SW is responsible for allocating and initializing the context. The context is identified by the handle.

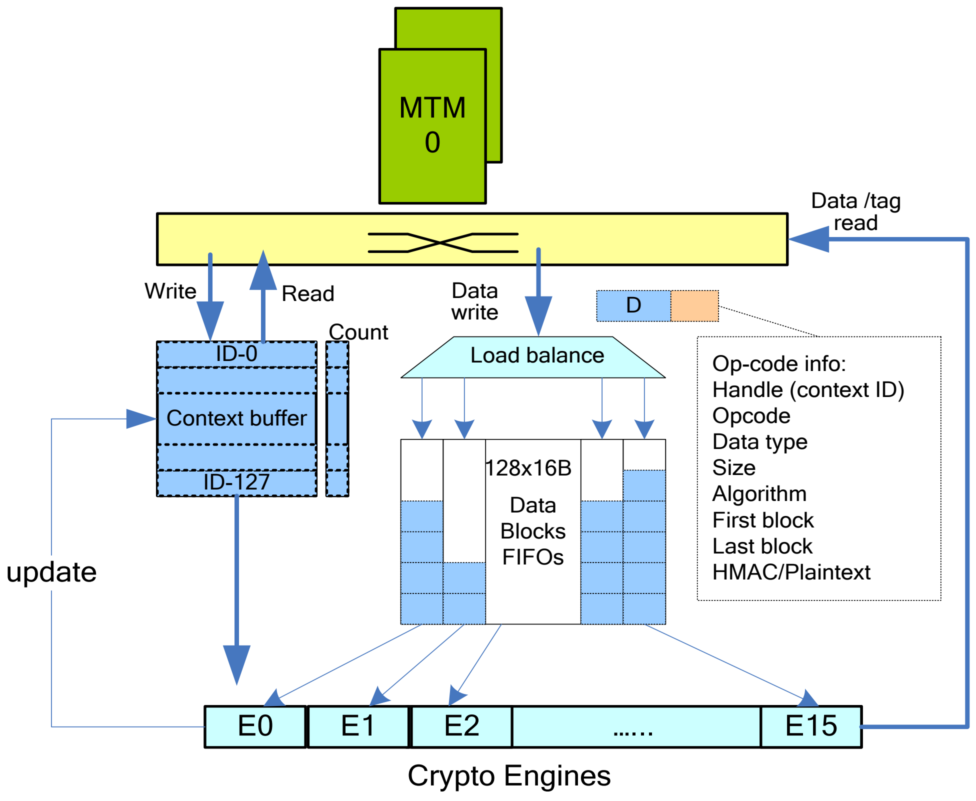
The data is copied by MTM from CTOP CMEM and passed in blocks to the appropriate engine’s FIFO.

The data is tagged with Opcode Info specifying all required information for a security engine block processing round.

At the end of processing, the context is updated for the next round and the processed data/tag is copied by MTM to CTOP CMEM.

Each context ID is managed by a job counter. When no jobs are pending in the FIFO for that context, the new incoming job can be allocated to a required engine type based on load balancing considerations.

Figure 14-2. Security acceleration unit architecture



14.3.3 Security Context

The security context holds all information required from block processing round to round:

KEY	*INITIALIZED BY THE SW*
Message Authentication Code	Initialized by command flag for first message segment. Updated by engine each round. Read by SW at the end. Written by SW to continue execution.
Mode of operation	Initialized by the SW
Initial Vector/Value (IV) for symmetric ciphers	Initialized by the SW and updated by engine each round. Read by SW to save security context
Message size (bits)	Initialized by the SW Updated by hash execution. Read by SW to save security context.

The context buffer is segmented into up to 128 contexts, 80B each.

The context ID 0-127 is used as a SW handle for crypto acceleration commands.

The context is initialized by CTOP in CMEM scratchpad and copied to the context buffer by MTM.

The context handle allocation is SW managed.

Code example for security context load:

```
cpskey [sm: R1],       Write KEY to context. R1 holds the security handle ID and
       [cm: R2]        R2 is a pointer to the Key in CMEM.

cpsiv  [sm: R1],       Write IV to context.
       [cm: R3]
```

Context 80-byte entry format per crypto function:

ALG./BYTE	*79:72*	*71:64*	*63:56*	*55:48*	*47:40*	*39:32*	*31:24*	*23:16*	*15:8*	*7:0*
DES					IV		Key			
3DES					IV		3 x Keys			
AES	H/CTR0/CCM		MAC		IV		Key			
MD5	Message Size [bits]		MAC							
SHA1			MAC (20 Byte : 63:44)							
SHA-224/256			MAC							
SHA-384/512			MAC							

The execution of a crypto function for a block of data updates the context to allow resuming the operation of next block as defined by selected block chaining mode.

14.3.4 Security Engine Data Block

Each data block (up to 128B) queued for security engine processing is tagged with following control information:

- Handle - Context ID
- Algorithm
    - AES (Chaining options / key options)
    - DES (Chaining options / key options)
    - SHA (sub modes)
    - MD5
- Opcode
    - Encrypt
    - Decrypt
    - Secure Hash
    - Expand Key
- First block marking - Used for HMAC initialization
- Last block marking - Used for HMAC padding
- HMAC or Plaintext indication

The job data is read by MTM from CTOP CMEM and pushed to specified engine's FIFO. The 15 engine FIFOs share 128 x 16B entries.

The job data block size can be 1B - 128B. The data can be fed to each cluster's security engines at a rate of 8B per core-clock/2.

The input/output data block alignment in CMEM can be on a byte boundary.

14.3.5 Encryption Chaining Modes

The table below lists security engine encryption/decryption operations for various chaining modes.

The context IV is used for feeding forward the chaining data.

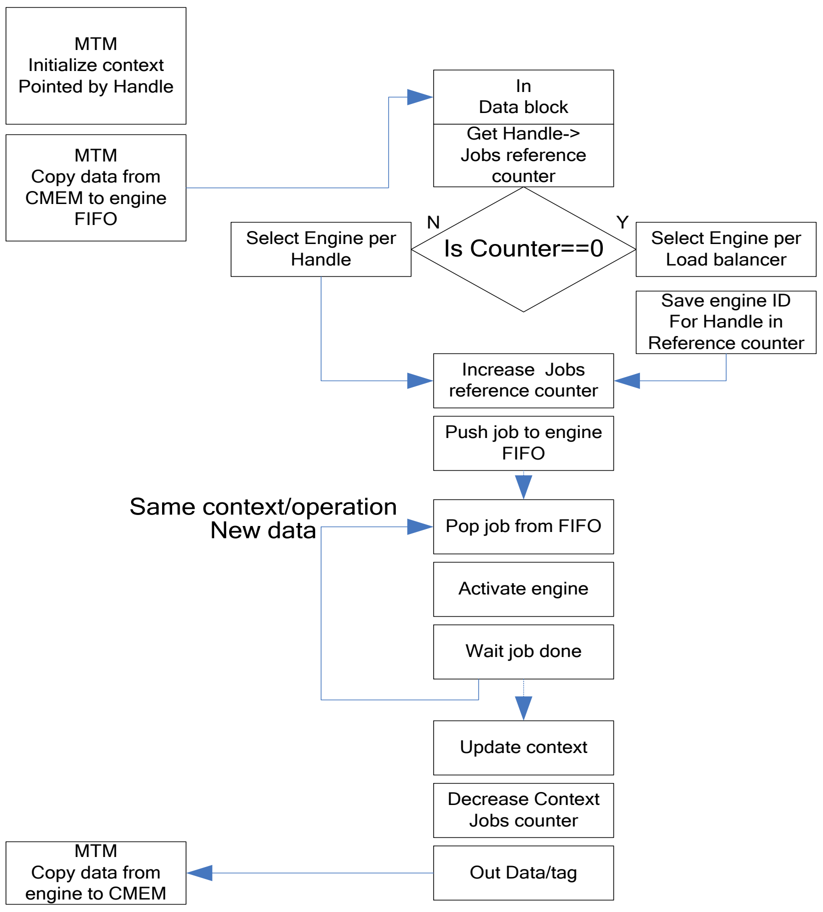
CHAINING MODE	***ENCRYPTION***		***DESCRIPTION***	
	CIPHERTEXT OUT	***IV OUT***	***PLAINTEXT OUT***	***IV OUT***
ECB	$E_k(P_i)$		$D_k(C_i)$	
CBC	$E_k(P_i \oplus IV)$	$E_k(P_i \oplus IV)$	$D_k(C_i) \oplus IV$	C_i
CFB	$E_k(IV) \oplus P_i$	$E_k(IV) \oplus P_i$	$D_k(IV) \oplus C_i$	C_i
OFB	$E_k(IV) \oplus P_i$	$E_k(IV)$	$D_k(IV) \oplus C_i$	$D_k(IV)$
CNTR	$E_k(IV) \oplus P_i$	IV.nonce || IV.cntr+1	$D_k(IV) \oplus C_i$	IV.nonce || IV.cntr+1

14.3.6 Security Accelerator Unit Process Flow

The flow diagram describes the security accelerator unit processing flow.

The engine for execution is selected either the previous used engine for the job or a newly assigned engine for a new job.

Figure 14-3. Security accelerator unit process flow



14.3.7 HMAC - Keyed Hashed Message Authentication Code Acceleration

The HMAC, per RFC 2104, dictates specific construction of MAC:

hash = H (Key $\oplus$ o_pad, H((Key $\oplus$ i_pad) || Message)

The security engines support acceleration of the HMAC stages. The stage is indicated by Data block configuration information.

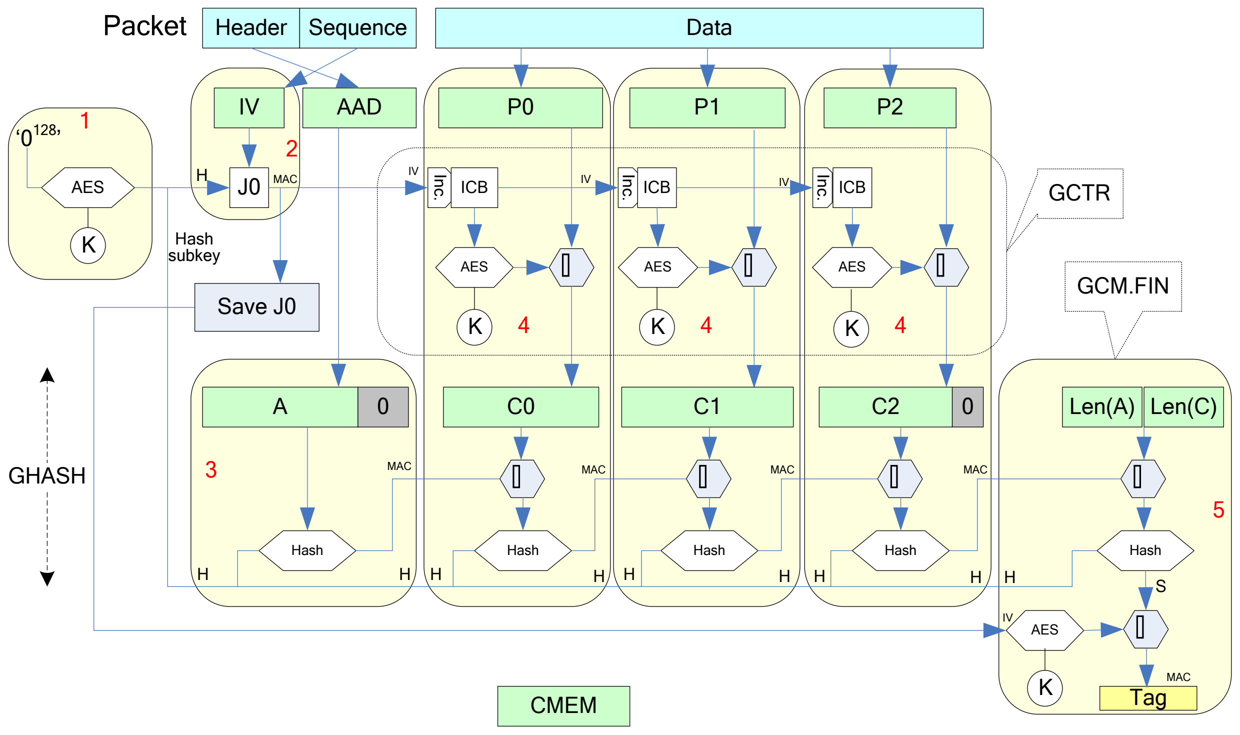
The HMAC supports HW key and data padding.

STAGE	***ACTION***	***CODE***	***DESCRIPTION***
HMAC first	state = H(Key $\oplus$ i_pad)	hmac.ipad	Use key from CMEM. Xor with i_pad and update context state to MAC and Message length.
HMAC	state = H(state, m)	shash	Use data from CMEM. Hash and update context state to MAC and message length.
HMAC last	hash = H(Key $\oplus$ o_pad , state, m)	hmac.opad	Use key from CMEM. Xor with o_pad and update context state to MAC.

14.3.8 AES GCM Encryption-Authentication Flow

The diagram below describes the functional stages of AES Galois Counter Mode encryption-authentication flow.

Figure 14-4. Functional stages of AES Galois Counter Mode encryption-authentication flow

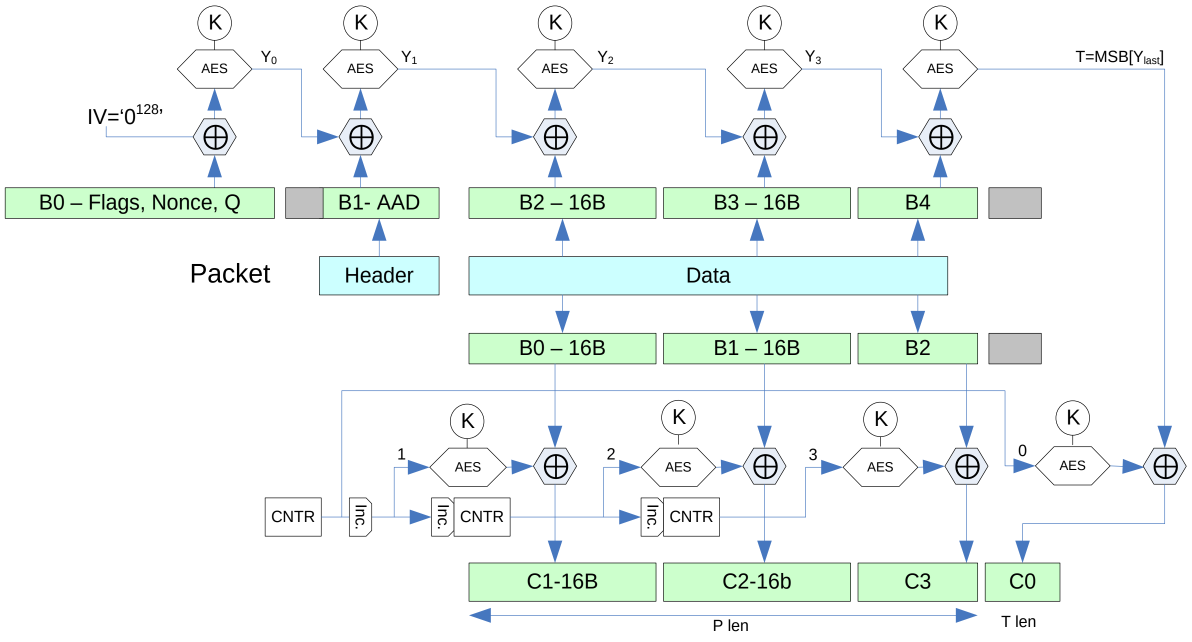


STAGE	OPCODE	DESCRIPTION
1	encr	Generate H=AES$_K$(0^{128}) the Hash subkey.
2	shash	Generate J0 and save for final stage. If len(IV)=96, then J_0 = IV || 0^{31} ||1. If len(IV) ≠ 96, then let s = 128 $\lceil$len(IV)/128$\rceil$-len(IV), and J_0=GHASH$_H$(IV||0^{s+64}||[len(IV)]$_{64}$). The hash data (IV) is taken from CMEM. The hash digest result is in context MAC.
3	shash	GHASH Additional Authentication Data (AAD) using H as hash subkey. The AAD is auto padded. The hash digest is saved in context[MAC].
4	encr	Repeat for each data block: AES encrypt plaintext and GHASH. The AES uses the context[Key, IV] The GHASH uses the context[H, MAC] The plaintext is auto padded.
5	gcm.fin	Generate last block [len(A)]$_{64}$ || [len(C)]$_{64}$ in CMEM GHASH last block with saved J0 as IV.
6	cpsmac	Copy the secured MAC from context to CMEM.

14.3.9 AES CCM Encryption-Authentication Flow

The diagram below illustrates the functional stages of AES Counter with CBC-MAC mode encryption-authentication flow. The mode is defined in NIST's SP800-38C specification. RFC 3610 defines CCM to use AES. The CCM provides authentication assurances over the Additional Authenticated Data (AAD) and provides both confidentiality and authentication over the plaintext data. The CCM combines counter mode encryption with CBC-MAC mode authentication with the same key. The CBC-MAC uses initialization vector (IV) as a nonce.

Figure 14-5. Functional stages of AES Counter with CBC-MAC mode encryption-authentication flow



15. Atomic Operations

This section is intended to provide a description of the different atomic operations and semaphores available in the NPS. The NPS includes a set of instructions added to provide efficient operations as the following examples will demonstrate.

15.1 Atomic Operations

This section describes the operations and provides examples demonstrating the improved performance achieved by using these operations.

In addition to legacy LL/SC sequences, the L2 supports a wide variety of atomic ReadModifyWrite (RMW) operations that are activated through special commands. These commands are implemented by instruction set extensions to the base architecture, where a destination register is updated with the read data (in atomic read-modify) or with the success or failure of an atomic operation (on atomic write-modify). Any later program reference to the destination register results in pipeline interlock due to data dependency and ensures completion of the atomic access operation. A thread may stall until the destination register gets updated, while other threads continue to execute normally. L2 atomic operations are performed on the physical address space.

15.1.1 Read Operations

15.1.1.1 Atomic Test and Set

Single read with atomic test-and-set of 1B, 2B or 4B aligned data. This non-cached transaction implements a Boolean semaphore lock operation by a conditional test-and-set operation. The transaction response returns the original memory contents to the reader and a fail/success flag in its destination register.

Access is naturally aligned to a 2B, 4B or 8B boundary.

A Boolean semaphore lock operation is achieved by the single reader that receives an all zeros response. Any subsequent readers receive all ones which indicate lock failure. Semaphore release operation is implemented by a simple write that clears the bytes. Due to WTI coherency scheme such a write would invalidate all stale L1 copies in other cores. Therefore instead of attempting a new lock, another core can loop reading the address and wait for the semaphore to be cleared. The data may be locally stored in their L1 data caches, preventing redundant accesses. Their copies would finally be invalidated by write of the releasing core, or by another core's successful atomic test-and-set operation.

Example sequence:

1. Cores A and B try to lock the same 32-bit semaphore by an atomic test-and-set operation. Semaphore data is missed in their data caches.
2. On L2, accesses from A and B are serialized. Core A sets the semaphore first. The directory shows no other sharers exist, therefore no invalidation request and no state update are required.
3. Core A gets all zeros (0x00000000) which indicate success.
4. Core B access to the semaphore sees it as all ones and does not alter memory. There is no need to check the directory because no memory update takes place. Core B gets all ones (0xFFFFFFFF) which indicate failure.
5. Core B may optionally spin reading on the semaphore address through regular load commands. After the first cache miss, the semaphore is stored locally in its L1 data cache and it no longer sends transactions to L2. The L2 directory updates core data cache B as sharer.

6. Core A completes its critical code section and releases the semaphore by a simple store of zeros to the semaphore address. The write misses its data cache and goes directly to L2.
7. L2 cache clears the contents and the directory sends an invalidation message to core B data cache.
8. Core B gets a cache miss, and on the next read it gets an updated zero value from L2. It can now try again to lock the semaphore by a new test-and-set command.

15.1.1.2 Atomic Read-Increment

Single atomic read-increment of 1B, 2B, 4B or 8B aligned data. This transaction implements read-increment command. The transaction always succeeds and returns the original memory contents to the reader. Selected data size is incremented in L2 by one. On a coherent address space all stale copies in L1 data caches are invalidated, including the reader's own data cache.

Counting semaphores achieve multiple ordered lock operations. Each reader receives a different data response due to increments of the previous readers. Semaphore release is complemented by atomic read-decrement-conditional command.

An ordered "ticket lock" mechanism can be implemented by combining two counting semaphores. The first counter provides allocated ticket numbers and second counter provides the next ticket owner. Initially both counters are zeros.

Any core can send a read-increment command to the first counter and get a unique ticket ID. It can then read the second counter (by a simple load) to check if its own arbitration turn has arrived (indicated by the second counter being equal to its allocated ticket ID). Only one core will win (the one that got the first ticket ID). Other cores can spin reading the second counter and have it stored in their local L1 data caches.

Once the winning core completes its critical code section, it performs atomic read-increment to the second counter (or simply stores its current ID plus one), effectively advancing the winner ID to the next core. As a result, all L1 caches that share this data get invalidated and the other cores read the updated ticket value through a cache miss. The next core in turn wins the semaphore which now matches its allocated ID.

Using the ticket lock semaphore on a coherent address space ensures fairness by assigning order (ID), and saves many cache miss transactions by locally storing the data in L1. Another variation of the protocol uses array lock to update each running ticket ID on a separate cache line. This method assigns a full cache line per each ID value and prevents redundant cache misses. Each core spins reading its unique cache line and gets a cache miss due to semaphore release only when its turn has arrived.

15.1.1.3 Atomic Read-Decrement-Conditional

Single atomic read-decrement-conditional of 1B, 2B, 4B or 8B aligned. The transaction returns the original memory contents to the reader and success/fail indication in the destination register. Selected data size in L2 memory gets decremented by one (1B, 2B, 4B or 8B, naturally aligned) if the original value was non-zero.

If the original selected data value was zero, the memory contents are not modified, and an update failure is recognized. On a coherent address space, when memory contents are successfully modified all L1 stale copies are invalidated before the transaction is acknowledged (including the readers own L1 copy). When memory contents are not modified no invalidations are required.

Atomic read-decrement operation enables a few useful implementations. On a counting semaphore implementation it enables each reader to atomically release its own lock without coordinating its access with other reader that shares the counter (note that in ticket-lock and array-lock implementation only one client releases the semaphore at a given time, therefore simple store can be used).

Another usage of conditional read-decrement in NPS-400 is automated frame buffer releases of Block Descriptors (BD) and Link Buffer Descriptors (LBD) on multicast frames. The frame data structure holds a block descriptor that points to a 32-bit multicast counter in memory which is initialized to the number of additional copies to transmit, up to and including the last shared block descriptor. Whenever a non-zero value is read from the multicast counter by read-decrement-conditional the buffers shared by multiple transmissions are not released. When a zero value is read, there are no more copies to transmit and the buffer can be released.

15.1.1.4 Mutually Exclusive Read-Increment-Conditional Pair

Mutually exclusive read-increment-conditional pair commands (combined MUTEX and counting semaphores). This non-cached transaction implements dual four byte conditional counting semaphores with mutual exclusive updates. The original accessed counter word is returned to the reader and the response field indicates update failure or success.

The mutually exclusive read-increment-conditional pair can work in two modes, reserved and non-reserved. In non-reserved mode the accessed counter word is conditionally incremented by one if the other counter word is zero, otherwise it is not modified. The read-increment-conditional command does not reserve its intent to lock-increment the counter, therefore the other counter is allowed to be incremented.

The dual mutually exclusive counter semaphore enables a coherency mechanism between mutually exclusive groups, where each group may have multiple locks but both groups can never lock together. One such case is table update where a single writer has to update a database shared by multiple readers. One counter is used by all readers while the other counter is used by the single writer (and therefore never incremented beyond 1). When a reader accesses the critical database it has to lock the semaphore by read-increment-conditional of the first counter. The same reader releases its allocation by write-decrement-conditional operation. The writer must succeed in a read-increment-conditional operation of the second counter before it can safely access the shared database. In case it has locked its semaphore, it is ensured that no reader can access the database while it is being updated. After the last update (followed by data SYNC command), the writer can decrement the second counter (or it may simply write it with a zero value as it is the only owner of that counter). Subsequent reader attempts to increment the first counter would now succeed, and the shared database can be accessed.

The same mechanism may use reserving mode for the writer semaphore lock. In case the writer did not succeed in incrementing the second counter (which is zero), it sets its lock-intent bit. Any subsequent read-increment-conditional on the first counter (initiated by a reader) would fail, while previous readers would succeed in decrementing the first counter. Eventually the first counter would reach zero after all past readers released their lock, and the writer would successfully increment the second counter. This reservation mechanism ensures bounded time for the writer to gain access to the shared database, which is the time it takes for all past readers to complete their critical operation and release their allocation by decrementing the first counter.

15.1.1.5 Read-And-Clear

Return original memory contents and clear it (half word to double word). This command can be used for reading and accumulating the contents of statistic counters by SW. The accumulated count is received by the reader and added to a wider counter stored in another memory location.

15.1.2 Write Operations

15.1.2.1 Atomic Add

Atomic add of one 8-bit, 16-bit or 32-bit value to selected naturally aligned physical address. BE combinations are limited power of two aligned accesses. Matching WDATA bytes are added to the selected bytes. This mode can be used to update statistic counters from time to time with any accumulated values, or for summing calculation results from multiple threads in a coordinated operation.

15.1.2.2 Atomic Double Add

Atomic dual add two 16-bit values to 2 words (32x2-bit target) or two double-words (54x2-bit target). The accessed structure address is 8-byte aligned for two words, or 16-byte aligned for two double words. This mode is used for atomic statistic counter pair update such as combining packet count and byte count in a networking application. In another application this operation can be used to update any sum of two values such as real and imaginary portions of a complex number.

15.1.2.3 Atomic AND

Atomic AND of 8 bit up to32 bit value to selected bytes (selective bit clear) naturally aligned physical address. The atomic response indicates if at least one bit has changed state. This operation can implement bitwise atomic semaphore clear. When selecting a single bit, the atomic response is similar to a test and clear operation indicating success in the initial bit value was one, otherwise indicating failure.

15.1.2.4 Atomic OR

Atomic OR of 8 bit up to 32 bit value to selected bytes in (selective bit set) address is naturally aligned physical address. The atomic response indicates if at least one bit has changed state. This operation can implement bitwise atomic semaphore set. When selecting a single bit, the atomic response is similar to a test and set operation indicating success in the initial bit value was zero, otherwise indicating failure.

15.1.2.5 Atomic XOR

Atomic XOR of up to 32 bit value to selected bytes in according to BE pattern (selective bit flip) address is naturally aligned.

16. TCAM Interfaces

The NPS-400 provides:

- Two internal ternary content addressable memory (TCAM) units; 1.25Mb each.
- Two SRAM banks of 640Kb for TCAM associated data and algorithmic lookups.
- Two Interlaken-LookAside interfaces compatible to an external NetLogic NL12K knowledge-based processor as well as an EZ-ILKN-LA proprietary interface.
- Two TCAM controllers that serve the internal and external TCAMs. Each controller serves the 1.25Mb internal TCAM and external TCAM on its side.

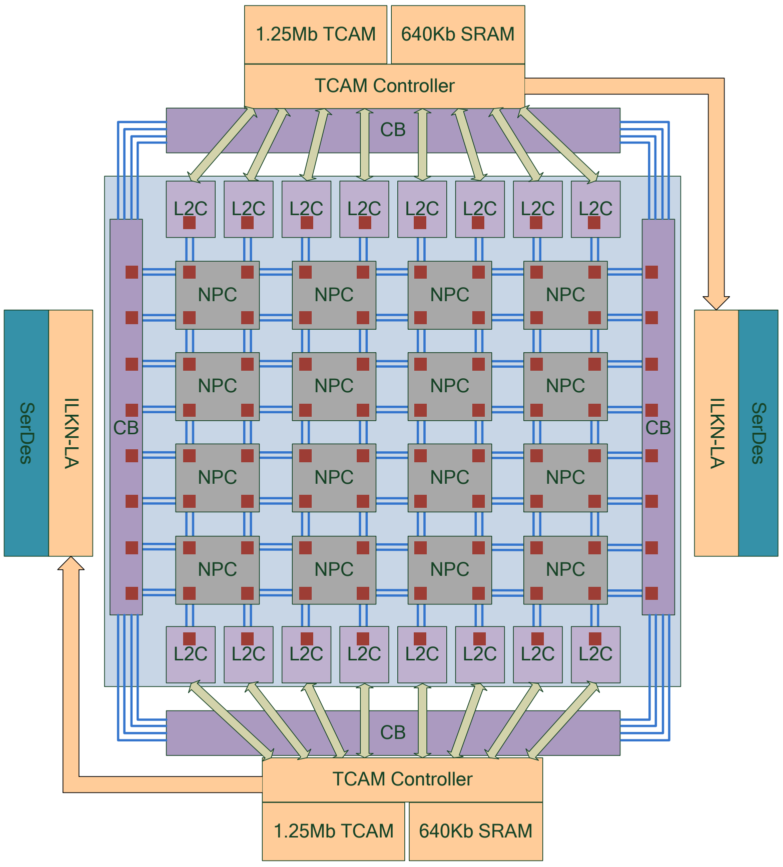
Either TCAM controller executes the internal and external TCAM lookups. Either TCAM controller can perform up to a single lookup per core clock. The controller exercises configurable WRR arbitration between lookup commands to/from the internal vs. external TCAMs.

Any CTOP can access any TCAM controller.

The internal TCAM can perform up to four lookups per command and return match index and associated data.

The external TCAM can also perform several lookups per command and return several results and associated data.

Figure 16-1. NPS-400 TCAM architecture



16.1 TCAM Lookup Trigger Command

The CTOP SW is responsible for triggering the TCAM lookup via a lookup command.

The command parameters are provided in the table below.

Table 16-1. TCAM command parameters

PARAMETER	SIZE	DESCRIPTION
Target		Internal TCAM (two sides) or External TCAM (two sides)
Key	640b	The lookup key (max 80B)
Key size	4b	Base key size 8B, 10B, 16B, 20B, 24B, 30B, 32B, 40B, 50B, 60B, 70B, or 80B. External TCAM use the Key size parameter of the command. Internal TCAM derives up to four keys with different sizes per lookup profile.
Key Mask	0-639	Key prefix mask for internal TCAM lookup. Masks out specified number of LSBs. The Key Mask allows using the same key with different masks as required by Longest Prefix Match (LPM) lookups.
LTSEL	6b	Lookup Table Structure Selector. Used as indirect selector of lookup profile. Up to 64 configured structures are available for lookup profiles combinations. For external TCAM, the profile also selects the TCAM command.
Result Size	4 values	The TCAM lookup result buffer size allocated by the CTOP.

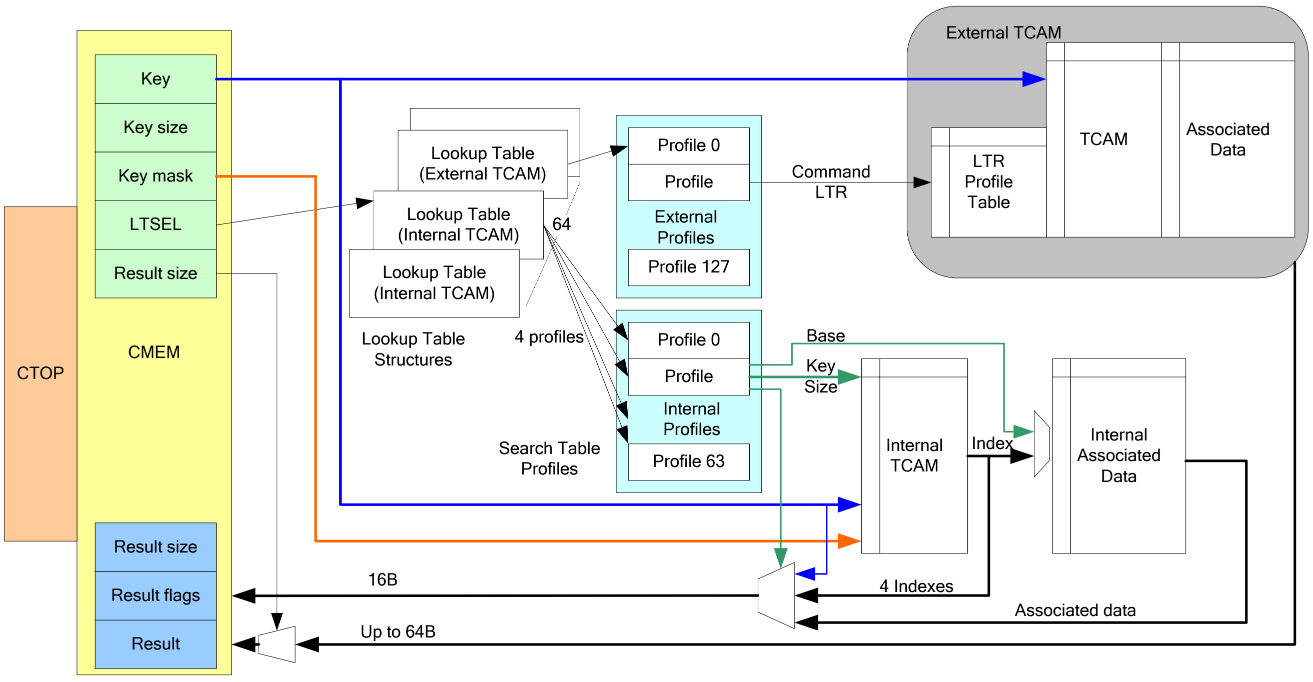
Table 16-2. TCAM lookup result response

PARAMETER	SIZE	DESCRIPTION
Result Flags	8b	The TCAM lookup reply flags.
Result Size	4 values	The result includes the resulting list of Indexes and associated data. The internal TCAM result is 16B. The external TCAM result can be 16B, 32B, 48B, or 64B.
Result	up to 64B	TCAM lookup result response.

The CTOP commands and TCAM result are passed via CMEM (see Figure 16-2 below).

The TCAM lookup command is messaged via EZnet from CTOP/CMEM by MSU to TCAM controller and result is messaged back. The EZnet message identifiers match request to result allowing CTOPs to issue several concurrent lookups.

Figure 16-2. TCAM trigger diagram



The command structure specifies an LTSEL that indexes a Lookup Table of profile combinations. There are 64 lookup tables, each assigned for internal or external TCAM.

The internal TCAM lookup table points to four internal TCAM profiles (out of 64 available) for up to four lookups.

The external TCAM lookup table points to one external TCAM profile that is configured with TCAM command and LTR pointer to an external TCAM on-chip Logical Table Registry that controls the key construction and lookup parameters. The Key Size for the external TCAM is set by the TCAM LTR profile table. The external TCAM result is loaded transparently to a CTOP CMEM allocated structure. The result could be truncated when its actual result size is larger than the allocated CMEM size.

The external TCAM result is described by result flags: Any error, Truncated response, Multi match, Any Match, Time out, TCAM error, MAC error, and No match on context.

The profile for internal TCAM lookup specifies the Key Size and search table definition. The internal TCAM key can be masked on LSB prefix. The first of four lookups can have associated data retrieved as a result or have algorithmic TCAM functionality enabled to support key reconstruction.

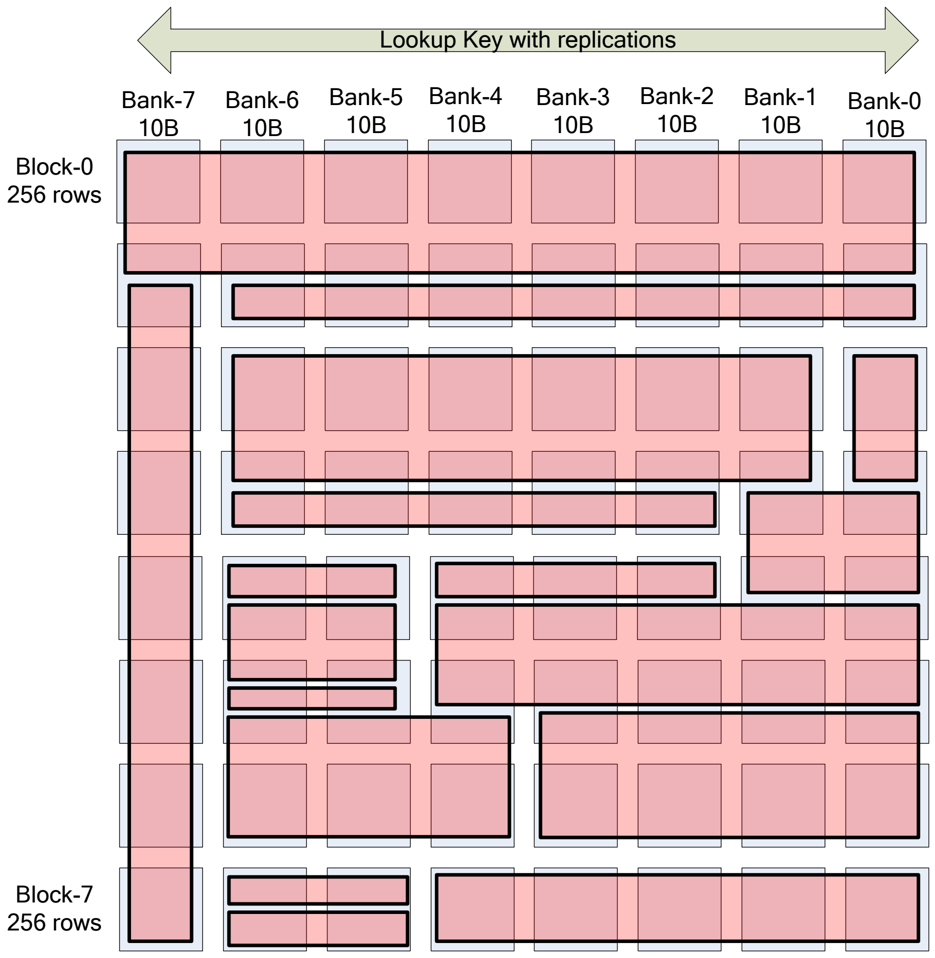
16.2 Internal TCAM

The NPS-400 contains two units of integrated 1.25Mbit Ternary CAM. The TCAM unit is constructed of 64 blocks 10B x 256 rows, tiled 8 banks x 8 blocks. The TCAM structure is parity protected and supports parity scan mechanism. The TCAM structure can be logically subdivided into 64 search tables.

The TCAM supports:

- Compare/Lookup command by data path application and Write, Read, Flush management commands.
- Various size lookup keys: 10B, 20B, 30B, 40B, 50B, 60B, 70B and 80B.
- The key can be masked with a global mask.
- Supports up to 4 results per lookup command for different search tables and up to 4 lookups in the same table. Each search table is allocated a block tile area BxN, where B=1-8 banks and N=1-8 blocks.
- The table tiles definition can overlap but a single search table per block is selected per lookup.

Figure 16-3. Internal TCAM architecture



The TCAM controller maximal performance is one key lookup per core clock cycle.

The lookup can be performed concurrently (parallel) in up to 4 search tables that do not occupy the same block (as defined by Lookup Table Definition structure). With parallel lookups, for each block, only a single search table can be selected for compare per lookup (meaning the single lookup cannot select two (or more) search tables overlapping same block).

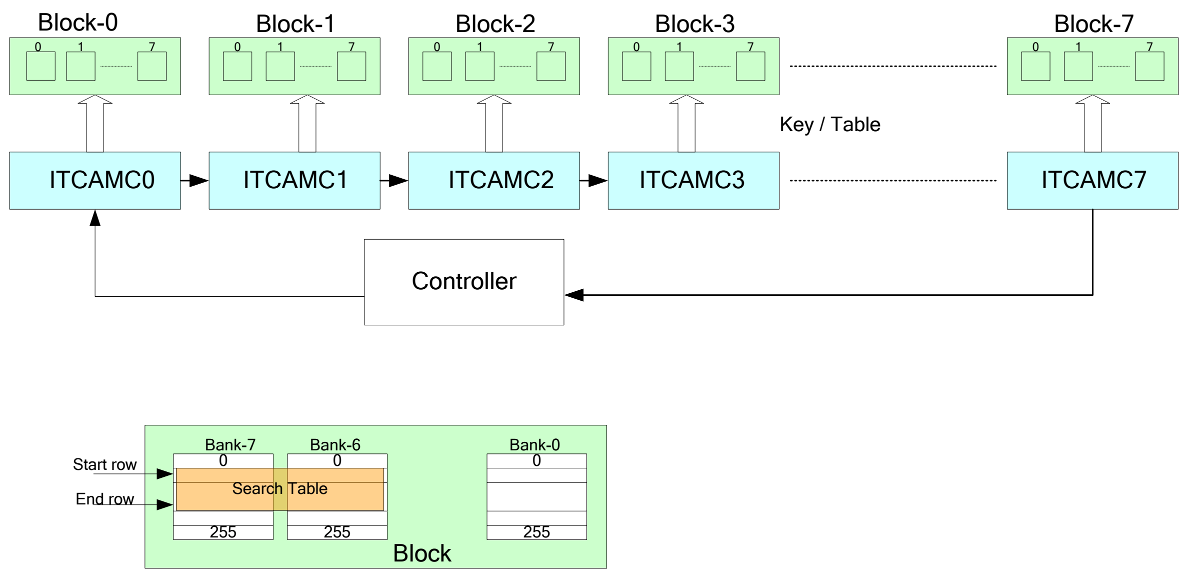
The lookups can be performed in sequence (as defined by Lookup Table Definition structure). The sequential lookups are executed individually, each lookup consuming a clock cycle. For the sequential lookups, the tables can overlap in the blocks definition.

Each search table defines its key construction and result format.

A search table is defined by (Start block / Start row) and (End block / End row).

Each block contains 256 rows. The compare is a cascade process starting from Block-0 row down to last block row of the selected table.

Figure 16-4. Internal TCAM compare process



16.2.1 Lookup Table Definition

The lookup command indicates LTSEL that selects one of 64 Lookup Table Structures (pointer table). The selected Lookup Table Structure index combination of up to four Search Table profiles each defining search table in the internal TCAM block/banks.

The tables are defined via 64 Search Table definition profiles that detail the table size, key usage and replication and result format.

Each search table profile is configured with key size and key construction attributes.

Each search table profile is configured with result format attributes.

The lookup command can target different combinations of tables via the LTSEL. The lookup table structure is indexed by the CTOP command's LTSEL parameter. The table combination can specify up to 4 search table profiles. The Lookup Table structure specifies parallel or sequential lookups.

The lookup table definition structure (a.k.a. Pointer Table) supports 64 combinations, each combination with up to 4 search table definition profiles (see 16.2.2 below).

There are 64 search table profiles. Each Search Table Profile defines the table's blocks, block rows and banks structure, key construction and result format.

The TCAM search table structure is defined by selecting:

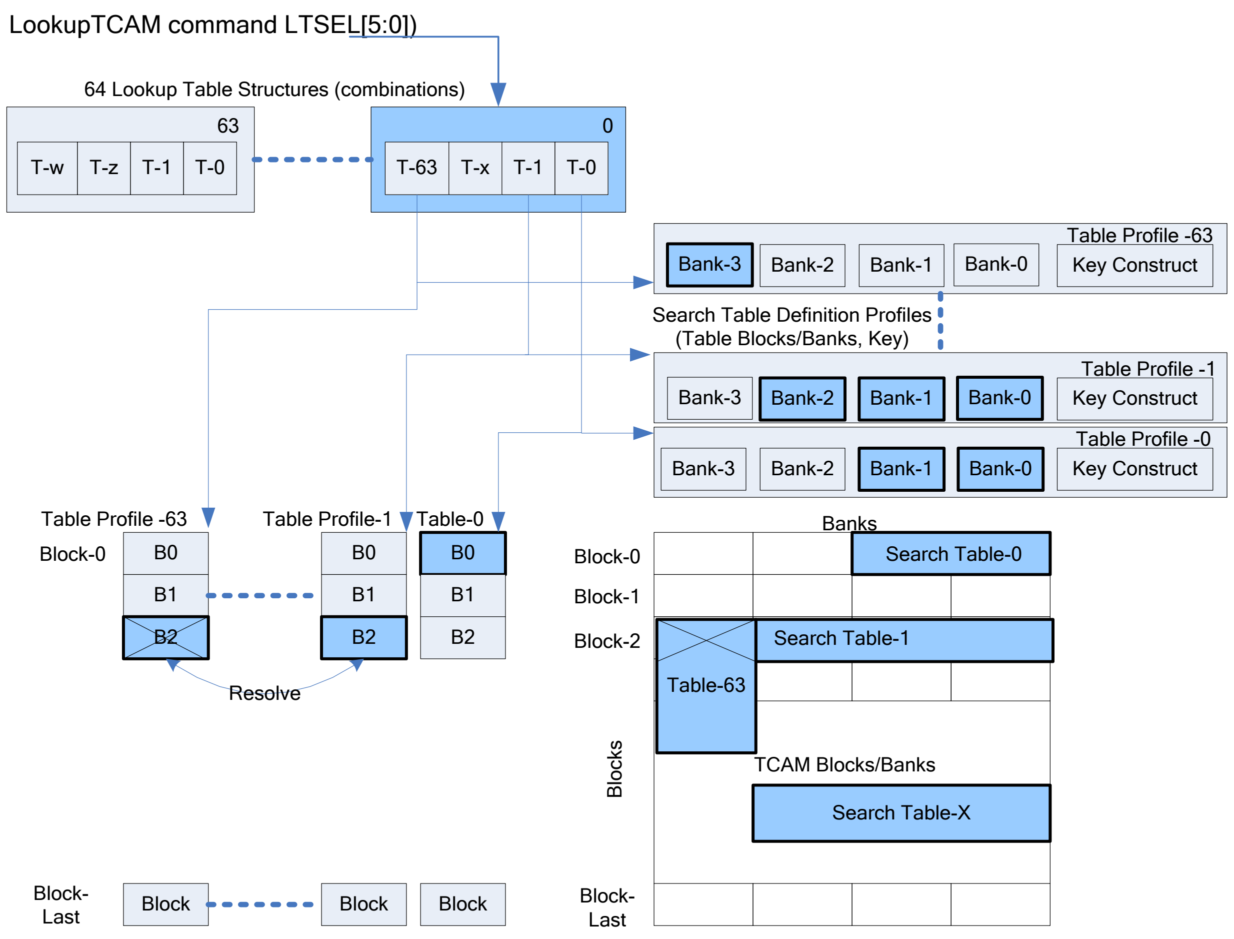
- Blocks – 1 to 8 adjacent blocks can be specified (via Start block and End block)
- Banks – 1 to 8 adjacent blocks can be specified (via Start bank and End bank)
- First block first row – a table can start on any row (of any block) that is a multiple of 16

- End block end row– a table can end on any row (of any block) that is a multiple of 16

The search tables can be defined overlapping the same blocks. The search table combination should avoid combinations that overlap and thus collide on same lookup command.

If a lookup command selects a search table combination that results in search table overlap collisions with more than a single search table per block, the mechanism resolves only a single search table for compare.

Figure 16-5. Internal TCAM - lookup structures and search table profiles – showing overlap



The figure above exemplifies table definition and overlap collision.

Three search tables are defined:

- Search Table-0 : Bank-0..Bank-1 / Block-0
- Search Table-1 : Bank-0..Bank-2 / Block-2
- Search Table-63 : Bank-3/ Block-2

The lookup command specifies Lookup Table structure combination of four search table profiles (63, x, 1, 0).

This combination causes an overlap collision on Block-2 by search tables 1 and 63. The collision marked in the figure above as Block/Bank box with an "X".

For parallel lookups, the collision is resolved and only one search table is accessed. The sequential lookups can be executed with overlapped tables.

16.2.2 Search Table Definition Profiles

Each profile specifies:

- Key size: 10B, 20B, 30B, 40B, 50B, 60B, 70B or 80B.
- Key construction bitmap. The key is constructed from groups of 10B taken in order from the 80B key of the Lookup command. Each bit enables 10B.
  - Example: 11000011 selects 40 bytes: 79:70, 69:60, 19:10, 9:0 for a lookup in table that spans 4 or 8 banks (as defined by Key replication).
- Start bank: 0-7.
- Start block/row: Block 0-7, Row (0-15) x 16
- End block/row: Block 0-7, Row (0-15) x 16
- Start bank: 0-7. The table can start on any bank and span continuous banks until bank-7.
- Key replications: 0-7 to span banks. When 10B, 20B, 30B key table spans multiple banks, the key needs to be replicated. The key replication specifies the number of times the key need to be replicated.
  - The 10B key can be replicated up to 4 banks: 10B, 10B, 10B, 10B.
  - The 20B key can be replicated to all banks: 20B, 20B, 20B, 20B.
  - The 30B key can be replicated to banks: (20B padding), 30B, 30B.
  - The 40B key can be replicated to all banks: 40B, 40B.
- Result Index shift:
  - No shift – The key match result is {block, row index, bank[2:0]}. Useful for tables that span 7 and 8 banks.
  - Shift right 1 bit – The key match result is {block, row index, bank[2:1]}.
  - Shift right 2 bits– The key match result is {block, row index, bank[2]}.
  - Shift right 3 bits– The key match result is {block, row index}.
  - Shift left bank 1 bit– The key match result is {block, row index, bank[1:0]}.
  - Shift left bank 2 bits– The key match result is {block, row index, bank[1]}.

- Result Index increment/decrement by value.
  - Result Index Decrement value: 0-32K. The value is decremented from result {block, row index, bank[x:y]} to null the constant table offset. The decrement value allows application software to get all lookup results relative to 0.
  - Result Index Increment value: 0-32K. The value is incremented from result {block, row index, bank[x:y]}. The incremented value allows application software to get all lookup results relative to an offset.
- Associated data (for first table only) – Indicates the result of the first search table is associated data and not the index (see 16.2.6.1 below).

16.2.3 Triggering TCAM Compare Lookup

The CTOP triggers the TCAM lookup key of up to 80B.

The command specifies:

- Key
- LTSEL - Index to lookup table structure entry (0-63) that contains 4 search table profiles.
- Key prefix masked bits 0b-640b

The key is used to construct up to 4 lookup keys; one per search table profile selected.

The TCAM lookup is messaged via EZnet from CTOP/MSU to TCAM controller and result is messaged back. The EZnet message identifiers match request to result allowing CTOPs to issue several concurrent lookups.

16.2.4 Key Compare

The search table profile defines a key construction bitmap.

The lookup table structure combination allows a construction of 4 keys for concurrent lookups in 4 search tables.

The key construction selects N*10B from an 80B lookup key (where N=1..8).

To lookup the key in a table that spans multiple keys per row, the key is replicated (…{Key}, {Key}).

The key can be replicated to all banks from 'Start bank'.

- Key of 10B is replicated: {Start-bank+ Replication}, …{Start-bank +1}, {Start-bank}
- Key of 20B is replicated:
  {Start-bank+ 2*Replication+1,Start-bank+ 2*Replication} …{Start-bank +1, Start-bank}
- Key of 30B is replicated:
  {Start-bank+ 3*Replication+2 : Start-bank+ 3*Replication} …{Start-bank +2: Start-bank}
- Key of 40B is replicated:
  {Start-bank+ 4*Replication+3:Start-bank+ 4*Replication} …{Start-bank +3: Start-bank}
- Key of 50B occupy: {Start-bank +4: Start-bank}
- Key of 60B occupy: { Bank 5: Bank 0} or { Bank 6: Bank 1} or { Bank 7: Bank 2}
- Key of 60B occupy: { Bank 6: Bank 0} or { Bank 7: Bank 1}
- Key of 80B is compared in banks: {Bank 7: Bank 0}

Key replication example:

BANK-7	BANK-6	BANK-5	BANK-4	BANK-3	BANK-2	BANK-1	BANK-0
				Key 10B	Key 10B	Key 10B	Key 10B
			Key 10B	Key 10B	Key 10B	Key 10B	
		Key 10B	Key 10B	Key 10B	Key 10B		
	Key 10B	Key 10B	Key 10B	Key 10B			
Key 10B	Key 10B	Key 10B	Key 10B				
Key 20B		Key 20B		Key 20B		Key 20B	
	Key 20B		Key 20B		Key 20B		
		Key 30B			Key 30B		
	Key 30B			Key 30B			
Key 30B			Key 30B				
Key 40B				Key 40B			
			Key 50B				
		Key 50B					
	Key 50B						
Key 50B							
		Key 60B					
	Key 60B						
Key 60B							
	Key 70B						
Key 70B							
Key 80B							

16.2.5 Key Match Index

The TCAM lookup result is an index to the row/bank that first matched the key.

The compare order is: Key-0, Key-1, Key-2, Key-3….

TABLE	*BANK-7*	*BANK-6*	*BANK-5*	*BANK-4*	*BANK-3*	*BANK-2*	*BANK-1*	*BANK-0*	*RECOMMENDED RESULT INDEX SHIFT*
x4-x1 10B Keys			Key-3 Key-7	Key-2 Key-6	Key-1 Key-5	Key-0 Key-4			{Block, row index, bank[2:0] }
x4 10B Keys					Key-3 Key-7	Key-2 Key-6	Key-1 Key-5	Key-0 Key-4	{Block, row index, bank[1:0] }
x2 10B Keys							Key-1	Key-0	{ Block, row index, bank[0] }
							Key-3	Key-2	
x1 10B Keys	T1-Key-0				T0-Key-0				{ Block, row index }
	T1-Key-1				T0-Key-1				
x4-x2 20B Keys	Key-3		Key-2		Key-1		Key-0		{ Block, row index, bank[2:1] }
	Key-7		Key-6		Key-5		Key-4		
2 Banks 40B Keys	Key-1				Key-0				{ Block, row index, bank[2] }
	Key-3				Key-2				

16.2.6 Result Format

The internal TCAM result is 16B.

The result has ECC protection. The Error bit indicates whether the result is valid or invalid.

The result is comprised of up to 4 key lookup results; one 4B result per search table.

Each one of four search table result (of 4B) includes:

- M – Key Match bit field indication. This bit field is set when key is found in the table.
- Index – 14b value of the first match of the key in the search table: {Block[2:0], row index[7:0], bank[2:0] }. The index is shifted and incremented/decremented per search table profile configuration.

Table 16-3. Internal TCAM - 16B result format containing four indexes

127	126-111	110-96	95	94 - 79	78-64	63	62 - 47	46-32	31	30-15	14-0
M-0 (1 bit)	Reserved (16 bits)	Index-0 (15 bits)	M-1 (1 bit)	Reserved (16 bits)	Index-1 (15 bits)	M-2 (1 bit)	Reserved (16 bits)	Index-2 (15 bits)	M-3 (1 bit)	Reserved (16 bits)	Index-3 (15 bits)

16.2.6.1 Indexed Associated Data

The internal TCAM supports 4K*20B SRAM that can be indexed by the first search table Index result.

The lookup command selected search table profile enables associated data result.

Per lookup the first search table (profile 0) result can have associated data returned instead of the Index.

To retrieve the associated data the TCAM lookup result index is used to address the SRAM.

The SRAM is indexed by Search Table Profile [SRAM Base address[11:0]] + TCAM result index. The SRAM index is scaled according to the result's byte size.

The Table profile selects the number of bytes that should be read from the SRAM and returned in the result and the result format.

- For keys > 30B the result index is up to 4K
  - The SRAM is indexed by Search Table Profile [SRAM Base address[11:0]] + TCAM result index [11:0]
- For keys = 20B the result index is up to 8K
  - The SRAM is divided to two portions of 10B.
  - The TCAM result Index [0] selects the SRAM portion.
  - The SRAM is indexed by Search Table Profile [SRAM Base address[11:0]] + TCAM result index [12:1]
- For keys = 10B the result index is up to 16K
  - The SRAM is divided to four portions of 5B.
  - The TCAM result Index [1:0] selects the SRAM portion.
  - The SRAM is indexed by Search Table Profile [SRAM Base address[11:0]] + TCAM result index [13:2]

Associated data reply format

The lookup command selected search table profile specifies the format of the result associated data.

The result format modes are:

- No data should be read. Up to four index results are returned.
  - See table above.
- 4 bytes of associated data (31b data + Match bit). Up to three additional indexes can be returned.

127	126-96	95	94 - 79	78-64	63	62 - 47	46-32	31	30-15	14-0
M-0 (1 bit)	Data [30:0]	M-1 (1 bit)	Reserved (16 bits)	Index-1 (15 bits)	M-2 (1 bit)	Reserved (16 bits)	Index-2 (15 bits)	M-3 (1 bit)	Reserved (16 bits)	Index-3 (15 bits)

- 8 bytes of associated data (63b data + Match bit). Up to two additional indexes can be returned.

127	126-64	63	62 - 47	46-32	31	30-15	14-0
M-0 (1 bit)	Data [62:0]	M-2 (1 bit)	Reserved (16 bits)	Index-2 (15 bits)	M-3 (1 bit)	Reserved (16 bits)	Index-3 (15 bits)

- 12 bytes of associated data (95b data + Match bit). Up to one additional index can be returned.

127	126-32	31	30-15	14-0
M-0 (1 bit)	Data [94:0]	M-3 (1 bit)	Reserved (16 bits)	Index-3 (15 bits)

- 16 bytes of associated data (127b data + Match bit).

127	126-0
M-0	Data [126:0]

16.2.6.2 Algorithmic TCAM Usage of Associated Data

The associated data is used by an algorithmic TCAM application that supports large virtual TCAM by combining internal TCAM and DRAM resident search trees.

The lookup command selected search table profile enables algorithmic TCAM result.

Per lookup the first search table (profile 0) result can have associated data returned instead of the Index to be formatted per algorithmic TCAM.

The algorithmic TCAM use the associated data to format result that is constructed from selected lookup-Key bits. The Key bits selected are indexed by the associated data.

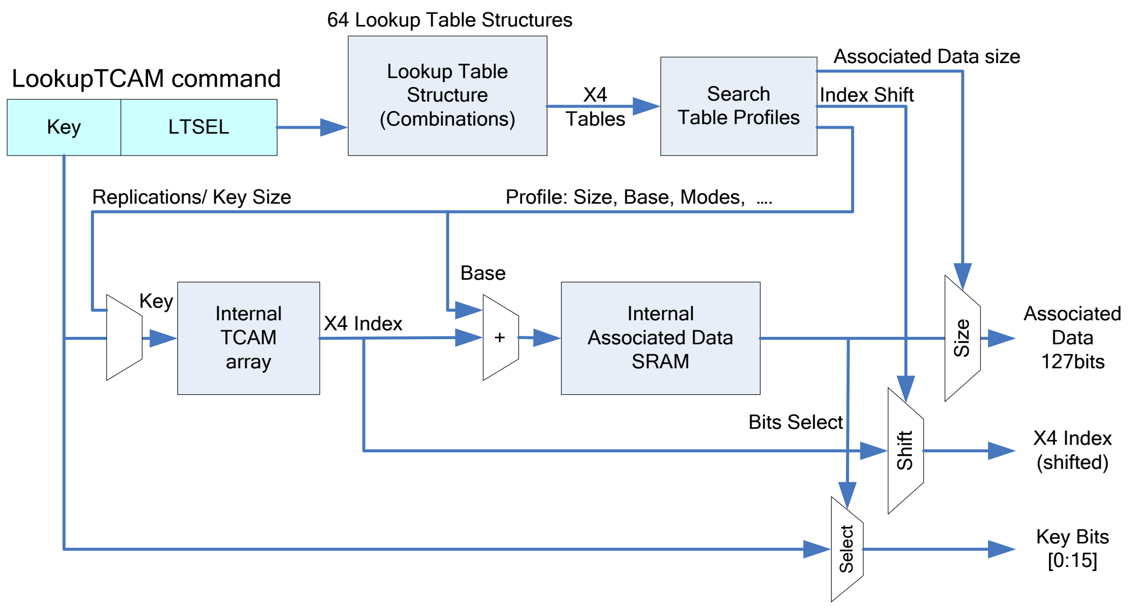
The algorithmic TCAM result is supported with 4B associated data format:

127	126-111	110:96	95	94 - 79	78-64	63	62 - 47	46-32	31	30-15	14-0
M-0 (1 bit)	Key Bits [0:15]	Index-0 (15 bits)	M-1 (1 bit)	Reserved (16 bits)	Index-1 (15 bits)	M-2 (1 bit)	Reserved (16 bits)	Index-2 (15 bits)	M-3 (1 bit)	Reserved (16 bits)	Index-3 (15 bits)

- For keys > 30B the result index is up to 4K
  - The SRAM is indexed by Search Table Profile [SRAM Base address[11:0]] + TCAM result index [11:0]
  - The selected Key Bits [0:15] returned are assigned by selecting bits from input lookup key. Index-0 is used to retrieve the 160b from associated SRAM.
  - The associated data of 160b is segmented to 16 entries of 10b. Each 10b entry selects a single bit from lookup key. If 10b selector is set 3FF hex, the bit is set to 0 in the Key Bits result.
    - Key Bits [i] is set from Key[Associated Data [10*i+9 : 10*i]], for i 0 to 15

- For keys = 20B the result index is up to 8K
  - The SRAM is divided to two portions of 10B.
  - The TCAM result Index [0] selects the SRAM portion.
  - The SRAM is indexed by Search Table Profile [SRAM Base address[11:0]] + TCAM result index [12:1]
  - The selected Key Bits [0:15] returned are assigned by selecting bits from input lookup Key. Index-0 is used to retrieve the 80b from associated SRAM.
  - The associated data of 80b is segmented to 10 entries of 8b. Each 8b entry selects a single bit from lookup Key[255:0].
    - The Key Bits [i] is set from Key[Associated Data [8*i+7 : 8*i]], for i 0 to 9
    - The Key Bits [10] is set from Key[Associated Data[39:32] +1]
    - The Key Bits [11] is set from Key[Associated Data[47:40] +1]
    - The Key Bits [12] is set from Key[Associated Data[55:48] +1]
    - The Key Bits [13] is set from Key[Associated Data[63:56] +1]
    - The Key Bits [14] is set from Key[Associated Data[71:64] +1]
    - The Key Bits [15] is set from Key[Associated Data[79:72] +1]

Figure 16-6. TCAM result Associated and Algorithmic TCAM reply block diagram



16.3 External TCAM / Knowledge-based Processor

The NPS-400 has an Interlaken-Look-Aside (ILKN-LA) interface compatible to:

- NetLogic NL12K knowledge-based processor
- EZ-ILKN-LA specification.

16.3.1 Interlaken-LA Interface

16.3.1.1 Interlaken-LA Features

The NPS Interlaken-LA interfaces key features include:

- Compatible with Interlaken Look-aside Protocol Definition Revision 1.1
- Number of channels: 1 (channel 0)
- Receiver’s BurstShort = 16 bytes and 8 bytes
- Transmitter’s user-selectable operational mode: BurstShort = 16 bytes or BurstShort = 8 bytes.
- Only NBO (Network Byte Order) data format is supported.
- ILKN-LA is up to 24 lanes and supported on SerDes lanes 24-47
- Configurable number of receive lanes and transmit lanes.
  Supported number of Tx lanes: 4, 8, 12, 16, or 24.
  Supported number of Rx lanes: 1-24
- The receive lanes and transmit lanes must operate at same bit rate: 6.25Gbps, 10.3125Gbps, 12.5Gbps.
- Receive lane swap and transmit lane swap to ease board design.
- Test patterns: PRBS7, PRBS23, PRBS31, and two programmable patterns.
- It is recommended to use an identical reference clock for the NPS SerDes and Look-Aside co-processor.

The number of lanes is configurable separately for transmit and receive. The number of transmit lanes should correspond to required key transactions bandwidth while number of receive lanes should correspond to required result and associated information bandwidth.

16.3.1.2 Data Formats

Data and Control Words are striped across the lanes sequentially, beginning with lane K, ending at lane K+M, and repeating for the next block of data. 64B/67B encoding occurs on each lane individually.

All bursts of Data Words are transmitted in the order of LSB to MSB, and the number of Data Words is variable per the Interlaken Look-Aside specifications with a minimum of one Data Word per packet.

EZ-ILKN-LA is supported with Network Byte Order (NBO) mode only.

Both data and control information is transmitted in bit order (MSB to LSB within each byte), from bit[66] through bit[0]. All Interlaken-LA transactions are SOP and EOP with valid 8 last bytes.

LANE	*66 64*	*63 0*							*7 0*
K	**X10**	**Control Word (SOP + EOP last 8B valid)**							
K+1	X01	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
K+2	X01	Byte 8	Byte 9	...					Byte 15
K+3	X01	Byte 16	...						Byte 23
	X01								
K+M	X01	M*8	...						M*8+7

The NL12K is supported with Network Byte Order (NBO) mode only.

Below is example of 640b Data Word NBO mapping.

DATA WORD / BITS:	*63:56*	*55:48*	*47:40*	*39:32*	*31:24*	*23:16*	*15:8*	*7:0*
Word 0	639:632	631:624	623:616	615:608	607:600	599:592	591:584	583:576
Word 1	575:568	567:560	559:552	551:544	543:536	535:528	527:520	519:512
Word 2	511:504	503:496	495:488	487:480	479:472	471:464	463:456	455:448
Word 3	447:440	439:432	431:424	423:416	415:408	407:400	399:392	391:384
Word 4	383:376	375:368	367:360	359:352	351:344	343:336	335:328	327:320
Word 5	319:312	311:304	303:296	295:288	287:280	279:272	271:264	263:256
Word 6	255:248	247:240	239:232	231:224	223:216	215:208	207:200	199:192
Word 7	191:184	183:176	175:168	167:160	159:152	151:144	143:136	135:128
Word 8	127:120	119:112	111:104	103:96	95:88	87:80	79:72	71:64
Word 9	63:56	55:48	47:40	39:32	31:24	23:16	15:8	7:0

16.3.1.3 Interlaken-LA Control Word

Interlaken-LA Control Words are transmitted first across a more significant lane, followed by an optional burst of Data Words.

The Control Word is per Interlaken-LA definition with proprietary definition of application specific fields. Only channel 0 is supported and only LA Protocol type.

BITS	ILKN-LA	NL12K	EZ-ILKN-LA INTERFACE USAGE
66	Inversion	ILKN-LA	
65:64	Framing		
63	Control		
62	Type		
61	SOP		
60:57	EOP_Format		
56:42	Application Specific-0	56:54 User Data (NL12K) 53:42 ContextAddr (NL12K) 56:42 ContextAddr (NL12K)	Opcode[14:0] / Status[14:0]
41	Ch-1 FC XON/XOFF	Reserved	
40	Ch-0 FC XON/XOFF	Ch-0 FC XON/XOFF	
39	Protocol	1- ILKN LA	
38:33	Application Specific-1	38 Reserved (NL12K) 37:33 Opcode[9:5] (NL12K) 38:33 Opcode[5:0] (NL12K)	LA Thread ID[13:8]
32	ILKN-LA Ch number	0 – Channel 0 used	
31:24	Application Specific-2	31:29 Status (NL12K) 28:24 Opcode[4:0] (NL12K) 31:30 ErrorStatus (NL12K) 29:28 Reserved (NL12K) 27 LTR[6]/BankSel (NL12K) 26:24 Opcode[8:6] (NL12K)	LA Thread ID[7:0]
23:0	CRC24	ILKN-LA	

16.3.2 Triggering TCAM Operation

The CTOP TCAM lookup command supplies the lookup key of up to 80B.

The command also specifies the following parameters to the TCAM controller:

- LTSEL – 6b Lookup Profile Index (0-63). The Lookup Profile Index points to Profile Pointer Table. The Table indicates Internal or External TCAM lookup.
- Key SIZE – TCAM Key size for lookup to set the key transaction size.
    - The key sizes supported: 8, 10, 16, 20, 24, 30, 32, 40 and 80 bytes.
- Result Size allocated in CMEM. When actual result is larger, the result is truncated.

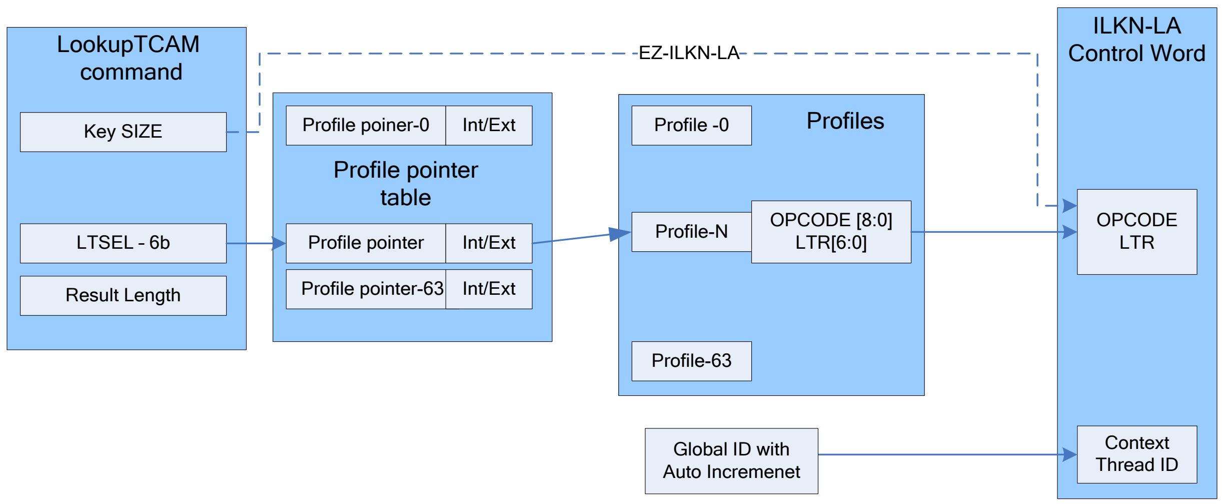
Each Lookup Table structure (Profile Pointer Table) has an associated External TCAM search table Profile.

The External TCAM search table profile defines the:

- TCAM Opcode: Compare1, Compare2, Compare3
- LTR[6:0]: Logical Table Registry index

The TCAM result is reformatted or passed unchanged except for 3 bits that can be optionally overwritten. The global configuration can specify to Modify Result of response Word-0 overwriting bits 58:56 with internally generated status {Any Data Truncation, Any Error, Any Match}. The response truncation may occur when TCAM response length is more than the expected result.

The TCAM operation trigger can indicate any TCAM Opcode including Lookup and Write.



The NL12K definition uses Opcode also for Logical Table Register specification.

16.3.3 TCAM Operation Context-Buffer/TCAM-Thread-ID

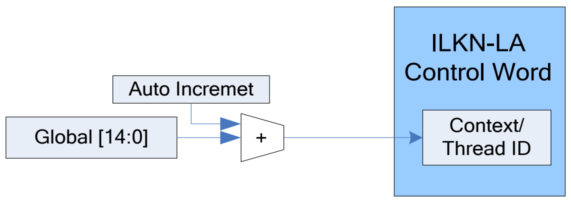
The TCAM device may implement a buffer for loading search keys. The buffer can be used for large key construction and re-use. Such a buffer, when used, is indexed by the command. The ILKN-LA Control Word is used to convey the Context/TCAM-Thread ID.

The TCAM controller assigns the Context / TCAM-Thread ID by:

- Global base and auto increment

The management methods allow uniqueness of Context/TCAM-Thread ID value for TCAM lookups in progress.

The Context/TCAM-Thread-ID is conveyed in the ILKN-LA Control Word.



The global pre-configured Context/TCAM-Thread ID value of 15b is allocation and use is under SW control.

Each time that a TCAM command is issued, the Context/Thread ID is incremented by a configurable step and wraps to 0 after a configurable value:

GLOBAL_CONTEXT + step_counter * STEP ;

where STEP is 1,2,4 or 8.

16.3.4 TCAM Response Matching

The request issue rate can be shaped to match the TCAM processing speed and reply bandwidth.

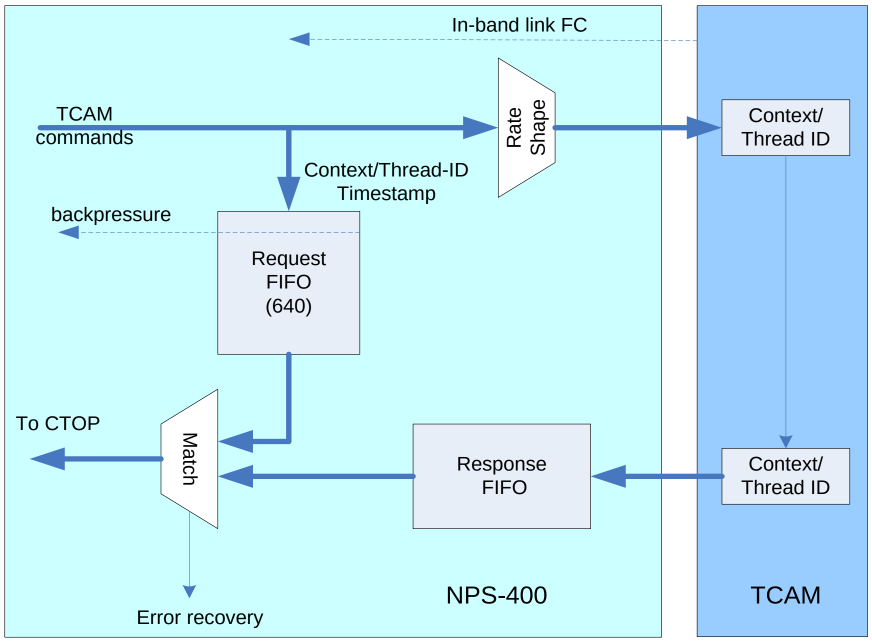
The requests sent to the TCAM are logged into a Request FIFO. The FIFO entry contains the Context/Thread ID to match the TCAM response and timestamp. The Request FIFO has 640 entries.

The TCAM is required to respond in order to all command requests.

The responses are matched to outstanding requests in the Request FIFO. The match is according to Context/Thread ID.

The TCAM is required to respond with at least 11b MSB of the Context/Thread ID.

In case of a response to request mismatch, request receive error, or timeout, the TCAM controller performs an error recovery procedure.



16.3.5 Error Recovery

The request/response transaction may encounter errors of several types:

- Interlaken-LA receive MAC errors
- TCAM device reported errors
- Response mismatch
- Request time-out
- Response FIFO overflow

16.3.5.1 Interlaken-LA Receive MAC Errors

The receive MAC can detect and report several errors:

- Link synchronization loss – The recovery procedure requires a Request FIFO flush and SW intervention.
- Packet CRC error – The recovery is graceful. The system tries to resynchronize request to response matching.
- Packet Format error - The recovery is graceful. The system tries to resynchronize request to response matching.

16.3.5.2 TCAM Device Reported Errors

The TCAM may report several types of errors:

- Interface errors
- Packet errors – affects only single packet.
- Device errors – may be persistent until SW intervention.

The TCAM error is logged for SW and a maskable interrupt is generated.

If masked the recovery is not executed.

The error is reported to a Search engine for SW processing.

16.3.5.3 Response Mismatch

The error is due to mismatch of response Context/TCAM-Thread-ID to request first in FIFO.

The TCAM error is logged for SW and a maskable interrupt is generated.

The recovery is graceful. The system tries to resynchronize request to response matching.

16.3.5.4 Request Time-out

The error is due to loss of transmitted request or response.

The TCAM error is logged for SW and a maskable interrupt is generated.

The recovery is graceful. The system tries to resynchronize request to response matching.

16.3.5.5 Response FIFO Overflow

This error is due to incorrect TCAM behavior on link level flow control.

The TCAM error is logged for SW and a maskable interrupt is generated.

The error cause response FIFO flush and SW intervention to restart.

16.4 Management Commands

The TCAM controller supports management generated transactions.

The SW can prepare an 80B transaction to internal/external TCAM and receive the response.

The SW configures the transaction data size: 8, 10, 16, 20, 24, 30, 32, 40, 50, 56, 60, 64, 70, 72, 80 bytes.

The SW configures the Interlaken-LA control word parameters:

BITS	ILKN-LA
56:42	Application Specific-0
40	Ch-0 FC XON/XOFF
39	Protocol
38:33	Application Specific-1
32	ILKN-LA Ch number
31:24	Application Specific-2

The TCAM controller extracts the Context/TCAM-Thread-ID[14:2]/[13:2] from the Interlaken-LA Control Word (TCAM format pending) for the response matching.

16.4.1 TCAM Response to Management

After triggering the command, the management response buffer is cleared.

The internal/external TCAM response is loaded to a management response buffer.

Internal TCAM response is up to four table results (block, row, bank).

For Interlaken-LA TCAM the response is up to 80B (not reformatted).

The external TCAM response with only indexes can be formatted as up to four results.

Software is responsible for reading the response and transaction error status. If the response is not read, it is cleared by the next command trigger.

17. Algorithmic TCAM Extension

17.1.1 Purpose

NPS provides the algorithmic TCAM as an alternative to using the conventional TCAM approach. External TCAMs are expensive, high-power devices that while necessary in some designs, can be problematic in designs that do not require the full performance of the external TCAM and have stricter power or real estate design constraints. EZchip's NP devices provide search data structures (i.e. trees, hash) in DRAM as alternative to using external TCAMs. The DRAM-based data structures provide a high search rate for direct table and hash table accesses but do not provide satisfactory performance for tree-only based searches that are used to emulate TCAM look-ups.

17.1.2 Solution

NPS provides a large internal TCAM which can be used for standard TCAM look-ups. In addition, the NPS supports an innovative way to combine the use of the on-chip TCAM with internal IMEM and external DRAM to provide an extension to the internal TCAM that enables larger database sizes to be stored while still providing TCAM-like look-up performance. High compression factors for typical ACL records is obtained thereby providing effective TCAM size on the order of tens of Mbs. The algorithmic TCAM extension provides an extension of the on-chip TCAMs stored in a combination of internal TCAM and memory while providing the look-up result in a few memory accesses, enabling a look-up rate of 600M lookups/s per side of the NPS. The internal TCAM is not used as a cache but stores internal data structures used to walk thru the TCAM database, therefore providing similar performance across the entire database and not a subset of it.

Some applications require a single TCAM lookup to process the first packets of network session establishment, requiring lookups for a fraction (~10%) of the forwarded packets. Such applications can benefit greatly from the algorithmic TCAM that presents improved cost/power solution for a large TCAM database while sustaining NPS wire speed performance.

EZchip's algorithmic TCAM extension also supports ranges in the TCAM rules for additional flexibility.

The solution consists of:

- A compiler tool that translates the TCAM-like database and produces an equivalent algorithmic-TCAM data structures allocated in internal TCAM, IMEM and DRAM.
- Internal TCAM with associated data SRAM. The associated data stores opcodes that are automatically executed, to assist with the data-structure walk thru.
  - The internal TCAM is composed of two parts; 1.25Mb each.
  - For internal TCAM and associated data SRAM details refer to the *Internal TCAM* section of the *TCAM Interfaces* chapter.
- Dedicated ISA extension in the CTOP/MTM to parse/decode the algorithmic-TCAM data structure opcodes that are used to store the TCAM database. The algorithmic TCAM look-up in the data plane SW is achieved thru a library call provided by EZchip that encapsulates the details of the internal data structures used to store the database and the use of the specific CTOP instructions used to decode them.

17.1.3 Compiling TCAM Rules

The compiler converts general TCAM rules to algorithmic TCAM structures.

The general TCAM rules are composed of:

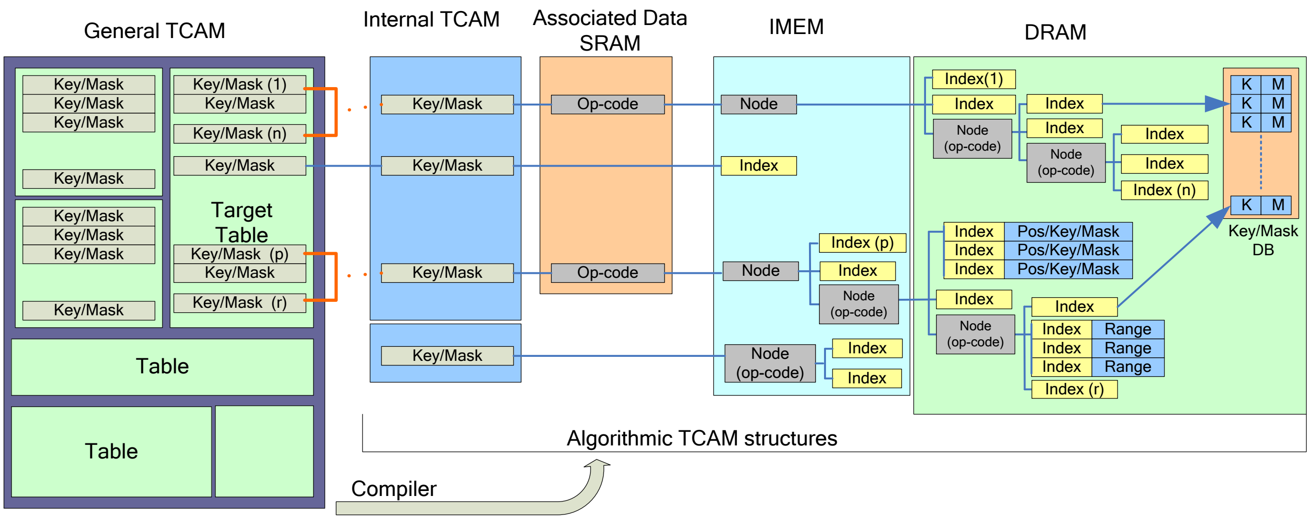
- Associated Table – The rules are stored in TCAM device tables. Usually one or several tables can be searched to match the search key. The index-result is returned for each table.
- Key – The binary key data (10B, 20B, 40B or 80B) stored in the TCAM.
- Key Mask – The binary bit masking the key. Mask is same bit size as the key.
- Priority – The keys are sequentially placed in the TCAM in descending implicit priority.

The algorithmic TCAM simulates a key lookup in a single (default) associated table. The TCAM rules are compiled to (up to 32K) internal TCAM entries and (up to 32K) tree structures resident in IMEM and DRAM. The algorithmic TCAM look-up result is an index to the first matched entry.

The compiler produces a set of data structures:

- Internal TCAM Table/Key/Mask – This is the starting point of the algorithmic search.
- Internal TCAM Associated Data SRAM – The associated data contains opcodes for structure walk-thru.
- IMEM search tree structure nodes and leaves. The IMEM entry and atomic access is 16B.
- DRAM search tree structure nodes and leaves. The DRAM entry and atomic access is 32B.
- DRAM full database Key/Mask.

Figure 17-1. Mapping the TCAM database to algorithmic TCAM resources



The compiler groups TCAM database entries into batches. Typically, but not necessarily, adjacent entries are grouped as a batch. The group is allocated an entry in the internal TCAM. Some TCAM database entries are not grouped and allocated as dedicated entries in the internal TCAM.

Each internal TCAM entry is implicitly associated with a table entry in IMEM. The IMEM entry contains a tree-node or a leaf-index result.

For some internal TCAM entries there is an associated opcode in the Associated Data SRAM. The opcode is used to expand the first search tree node. The opcode is HW processed to expand the first search tree node located in IMEM.

The internal TCAM entry index may point directly to a lookup result index leaf.

Internal TCAM entries that do not have opcode in associated data SRAM, point to tree nodes that contain the opcode for SW expanding the node. The node opcode expansion is executed by a CTOP command. The performance advantage of associated data opcodes results from the HW accelerated expansion without usage of CTOP commands.

The IMEM and DRAM contain a collection of search tree structures. Each tree is composed of linked nodes and nodes indexing leaf entries containing the lookup result index. Eventually the tree walk-thru ends up with a leaf entry containing the result which is one of several types:

- Leaf with Index result or empty leaf.
- Leaf with Index result with pointer pointing to full key/mask.
- Leaf with Index result with partial bits of the key: bit position, key data bits, key mask bits.
- Short list of (2-3) leaves with Index or with partial key bits (as above). The short list is serially scanned to compare each partial key.
- Leaf with specified range of values for key compare.

17.1.3.1 Rule Grouping

While processing the TCAM database the compiler groups sets of rules to compress the database into the internal TCAM. The grouping is done by identification of common unmasked rule bits. The common bits act as a 'Head' of the group.

Example of rules grouping:

101010*****10001001*****0**000**1**
011011*****10001001*****1**000**0**
100100*****10001001*****0**000**1**
001010*****10001001*****1**000**0**

The common bits ‘Head’:

***********10001001********000*****

Where * indicates Don't Care and the 'Head' is placed in the internal TCAM.

The rule grouping is common in network ACL rules due to inherent subnet rules. This grouping approach leads to high compression ratios of the database to the algorithmic TCAM structures.

The rule grouping compression is typically capable of compressing TCAM database rules to internal TCAM by an average factor of x20 for extensive benchmarks. The NPS 2.5Mb internal TCAM can support a compressed database of 50Mb (or more) TCAM rules.

Table 17-1. Example of 50Mb rule database compression

RULE DATABASE 50MB		INTERNAL TCAM 2.5MB		TARGET COMPRESSION RATIO
RULE WIDTH [BITS]	NO. OF RULES	KEY WIDTH [BITS]	NO. OF RULES	
80	655K	80	32K	1:20
160	327K	160	16K	1:20
240	218K	320	8K	1:27
320	163K	320	8K	1:20
400	131K	640	4K	1:32
480	109K	640	4K	1:27
560	93K	640	4K	1:23
640	81K	640	4K	1:20

17.1.3.2 Searching in Parallel Rule Groups

The rule database may contain rules that can be grouped, but with interleaved groups. The groups interleaving cannot be efficiently compressed into a single table of internal TCAM.

Example of rule groups (G1 and G2) interleaving:

G1) 101010***10001001***10*00**
G2) 011011***01000110***10000**
G1) 100100***10001001***10010**
G2) *01010***10**10*1***10000**

The common bits Heads for two groups:

G1) ***********10001001************
G2) **********************10000****

The key lookup 1010100001000100100010000011 results in a match to 1st and 4th rules and a match for both groups.

However because the 1st rule is highest priority, the result should return only the first index.

Both groups' Heads need to be matched in parallel by the internal TCAM.

To resolve a match on both groups, the algorithm needs to walk thru both search trees and compare the final result index priority.

In order to support parallel rule groups, the internal TCAM supports up to four tables. The key is used for a look up in parallel in up to four tables. The SW algorithm performs a search tree walk thru on all internal TCAM matched results.

The internal TCAM supports associated data opcode only for single table. Thus, only a single table can auto expand first tree node. For the expansion of other tables tree nodes, the SW uses dedicated CTOP instructions.

When parallel rule groups require more than four tables, the compiler generates an additional tree structure with Head rule of all key bits masked as Head = *.

17.1.3.3 Search Tree

The compiler compiles each rule group into a search tree structure. The compiled trees are balanced to yield shortest tree depth. Each tree node can be split into a number of branches. The node split branching factor can vary from 2 to 64K.

Given the first tree node, the search algorithm traverses the tree to its leaves.

The leaves support two types:

- Single leaf, holding the Index result of the lookup.
- Short list of results where the list contains a few results. Each result holds a partial key/mask and the index. The SW compares the partial key of each result on the list to the lookup key.

The search tree nodes are allocated to IMEM and DRAM.

The compiler ensures to use the IMEM area that fits to IMEM region allocated for algorithmic TCAM.

The compiler will fit the entire rule database to allocated DRAM or generate a warning notification.

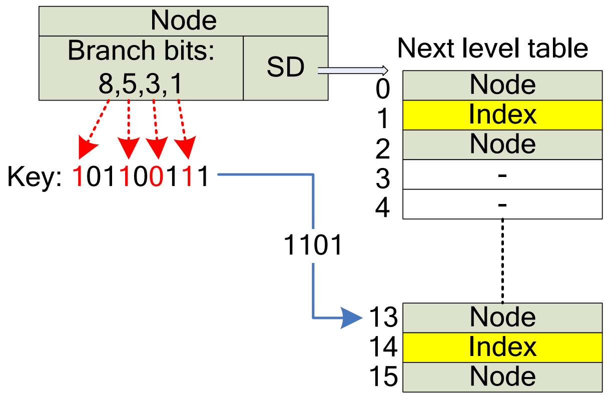
17.1.3.4 Tree Node

Each search tree node is coded with information to expand the node branch. The coding information lists 1 to 16 bits selected from the key that are used as a branching index to next level. The node holds a pointer to the next level table in the Structure Descriptor.

For an IMEM node, the next table can be in IMEM or DRAM.

For a DRAM node, the next level can be in IMEM or DRAM.

Figure 17-2. Search tree node split



17.1.3.5 Leaf Types

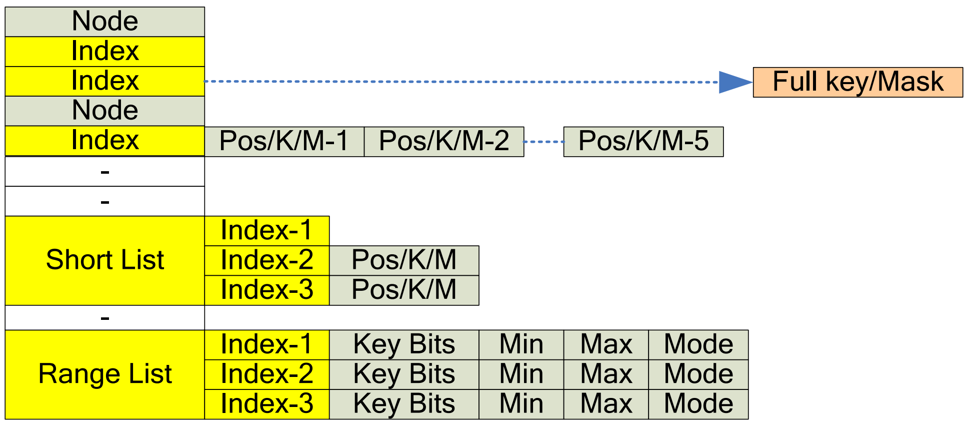
Eventually the tree walk-thru ends up with a leaf entry.

The leaf entry supports several types:

- Empty – The lookup key was not found in this node path.
- Single Index result – The lookup key finds a match in the structure. The retrieved Index result indicates the TCAM database location (priority).
- Single Index result with pointer to full Key/Mask – If Key/Mask matches, lookup key found in the structure. The retrieved Index result indicates the TCAM database location (priority).
- Single Index result with short (1-5) list of partial bits of the key: bit position, key data bits, key mask bits – If any partial key matches, the retrieved Index result indicates the TCAM database location (priority).
- Short List – The lookup key should be compared serially to short list of bit-partial key values. If a match is found, the Index is retrieved. The short list contains 2 - 3 entries, each entry:
  - Key bits to select for compare; any number of key bits can be selected.
  - Key bits values for compare.
  - Result Index in case of match.
- Range List – The lookup key should be compared serially to short list of range values. If a match is found, the Index is retrieved. The short list contains 1 - 3 entries, each entry:
  - Result Index in case of match.
  - Key bits to select for range compare - up to 24b.
  - Range min value 24b.
  - Range max value 24b.
  - Range compare modes:
    - Less than: Key selected bits value ≤ Range min value.
    - Greater than: Key selected bits value ≥ Range min value.

- Between: Range min value ≤ Key selected bits value ≤ Range max value.
- Outside: (Key selected bits value ≤ Range min value) or (Range max value ≤ Key selected bits value)

Figure 17-3. Search tree leaf types



17.1.4 Compiler Target Constraints

The compiler receives a set of user constraints:

- Internal TCAM maximal footprint.
- IMEM maximal footprint.
- Target budget of accesses to IMEM that are according to target application performance.
- DRAM maximal footprint.
- Target budget of accesses to DRAM that are according to target application performance.
- Target budget for CTOP instruction cycles for algorithmic decode that are according to target application performance. The instruction budget effects tree depth and type of nodes.

A given database can be compiled to a spectrum of valid results with various weights between footprint/accesses and decoding cycles.

The compiler targets a solution that is as close as possible to weights balance supplied by the user constraints.

If the constraints are too tight and therefore not all rules can be compiled, then a report will be generated indicating options to relax some constraints.

The algorithmic structure supports incremental updates: insertions, deletion and modification. The control plane SW driver monitors the algorithmic TCAM performance and footprint for deviations from target constraints.

The data plane application monitors the actual algorithmic performance for CTOP cycles and memory access.

When the structure violates the constraints due to incremental updates, the structure can be re-compiled and re-balanced.

17.1.5 Algorithmic Performance and Accelerations

The algorithm uses accelerations for node split processing and leaf processing.

The accelerations comprised of:

- HW elements in the internal TCAM
- Per CTOP cluster DMA and accelerators
- CTOP ISA special accelerators

17.1.5.1 Node Split Processing

With strict DRAM access constraints (1 to 2 accesses), the compiler targets first node split as much as possible by selecting more split bits in associated data SRAM. The first node split is HW decoded.

The high initial split lowers the tree depth and associated SW processing. Based on the structure of the database, it might be necessary to open additional search paths in which automatic HW decoding is not provided.

Additional opened tree paths shall be SW processed to decode the node split. The SW processing is accelerated by CTOP core opcode 'LOAD-BIT' ISA. LOAD-BIT ISA receives byte-address (=key address) + bit-offset (selected bit). It performs a load of 1B + shift right + masking. When the address is in CMEM (since key is in CMEM), this whole operation takes 1 core execution cycle.
The processing of a “SW split node” which is split on ~12 arbitrary bits of the key takes 90 cycles: 50 cycles is overhead for node processing and about 40 cycles are the cost of selecting the (12) bits and aggregating their values into a branch-index register.

17.1.5.2 Leaf Processing - Partial Key/Mask (70%)

Majority of leaves are of single rule (Index result) without full key/mask compare:

- Index without key - 32B.
- Index with short (1-5) list of partial key/mask - total 32B. Each entry holds 5x (position/key/mask).

Compiler benchmarks result in ~70% of leaves having up to 1 rule to resolve (and in many cases have no rules to resolve).

Such leaf processing is accelerated by SW processing by a special “fast-path” for these cases in SW. The CTOP SW consumes ~73 cycles per rule with up to 5 partial key/mask. This fast path includes checking a match against that single rule index, based on up to 5 partial checks. Each partial check uses a CTOP ISA accelerator.

17.1.5.3 Leaf Processing - Full Key/Mask (20%)

Approximately 20% of leaves point to full key/mask rules for final resolution. The full key/mask may consume up to 160B and is fetched in 128B reads.

Full key/mask comparison uses the ternary-memcmp cluster-level accelerator instruction which takes 100 execution cycles. Execution cycles consumed to decode the rule index + fetch key/mask for comparison, depending on the total rule length. The memory loads and accelerator execution are done by a DMA in the background, while the CTOP thread is suspended.

17.1.5.4 Leaf Processing - List of Rules (10%)

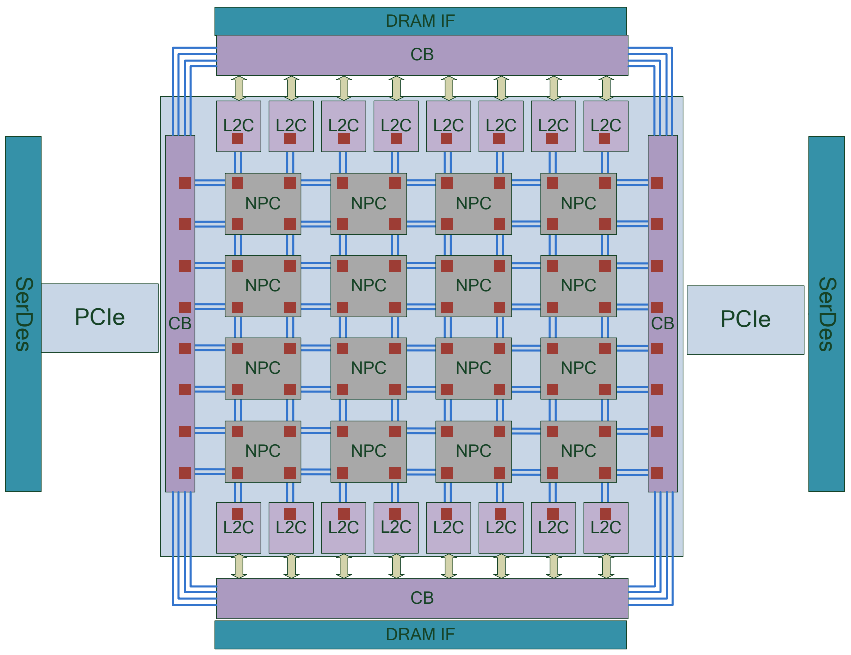
When the DRAM access constraints are loose (user allows >3 DRAM accesses), a leaf is likely to contain more than a single rule to resolve. Typically, the compiler prefers to resolve the rules using core-level ISA accelerator by specifying sets of partial key/mask for comparison. Partial comparison takes 8 cycles per key/mask. Out of these 8 cycles, the ISA itself runs in a single clock. The other 7 cycles are used to decode the start-end position of each key/mask pair.
Leaf processing takes 50 cycles (constant overhead) + (24 cycles * #rules) + (8 * total #partial-checks).

18. PCI Express Controller

The NPS-400 integrates two PCI Express end-point controllers; each supporting up to 8 lanes.

- West side
  - PCIe – x1, x2, x4, x8
  - x1 PCIe on SerDes lane 48
  - Only PCIe on the west side can access the NPS device SoC configuration and management address space. The control plane CPU must be connected on the west side.
- East side
  - PCIe – x1, x2, x4, x8
  - x1 PCIe on SerDes lane 48

Figure 18-1. PCIe interfaces



Key: CB = crossbar

Table 18-1. SerDes assignment options for PCIe on west and east sides

SerDes	2x8	2x4	2x2	1x1
40	PCIe [0]	--	--	--
41	PCIe [1]	--	--	--
42	PCIe [2]	--	--	--
43	PCIe [3]	--	--	--
44	PCIe [4]	PCIe [0]	--	--
45	PCIe [5]	PCIe [1]	--	--
46	PCIe [6]	PCIe [2]	PCIe [0]	--
47	PCIe [7]	PCIe [3]	PCIe [1]	--
48	--	--	--	x1 PCIe

See also the “SerDes Allocation Scheme” in Chapter 2.

18.1 Features

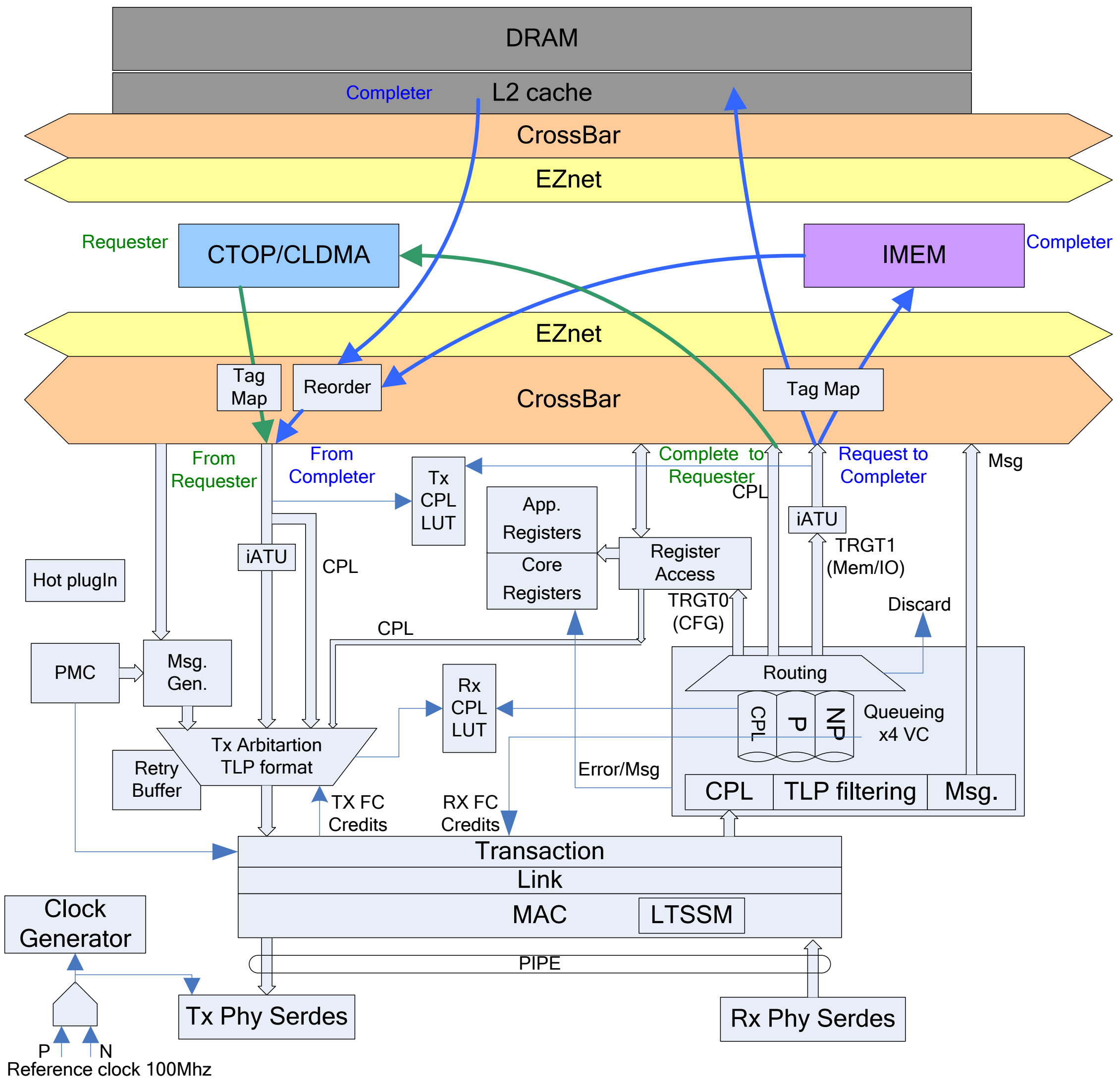
The NPS PCI Express interface supports:

- All non-optional features of the PCI Express Base 3.0 Specification, revision 1.0
- Gen1 lanes @ 2.5 Gbps (x1, x2, x4 or x8)
- Gen2 lanes @ 5.0 Gbps (x1, x2, x4 or x8)
- Gen3 lanes @ 8.0 Gbps (x1, x2, x4or x8)
- The following optional features of the PCI Express Base specification:
  - Single Root I/O Virtualization (SR-IOV)
  - Alternative Routing-ID Interpretation (ARI)
  - Latency Tolerance Reporting (LTR)
  - Address Translation Services (ATS)
  - Function Level Reset
  - Resizable BAR
  - Optimized Buffer Fill and Flush (OBFF)
- Up to 256 outstanding requests
- Max_Payload_Size size of 128 bytes; 128B maximum Request size
- Internal Address Translation Unit (iATU) - 16 inbound regions, 16 outbound regions
- Automatic Lane reversal.
- Polarity inversion on receive
- 2 Virtual Channels (VCs)
- Multiple Traffic Classes (TCs)
- 4 Physical Functions (PFs) with 128 Virtual Functions (VFs) per each of the two PCIe controllers
- Store-and-Forward queue modes for received Transaction Level Packets (TLPs)
- PCI Express beacon and wake-up mechanism
- PCI Power Management: Active State Power Management (ASPM)
- PCI Express Advanced Error Reporting
- PCIe messages for both transmit and receive
- Configurable filtering rules for Posted, Non-Posted, and Completion traffic
- Configurable BAR filtering, I/O filtering, configuration filtering and Completion lookup/timeout
- Master and slave operations with flexible SW interface
  - Concurrent slave and master mode operation
  - Host direct access to internal configuration and IMEM/EMEM Memory Spaces
  - Multi Threaded SW controlled DMA Master operations (via CTOP CLDMA)
  - Support for No Snoop (NS) per transaction
  - Support SW defined Traffic Classes (TC) per transaction
  - SW defined for strict and relaxed ordering modes (RO) per transaction
- Controller implementation supports either parity checking or error checking and correction (ECC) for SRAM memories
- Type 0 configuration space
- MSI-X

18.2 PCIe Controller Architecture

The PCIe controller implements the PCI Express Transaction layer, of the Data Link Layer, and the MAC portion of the Physical Layer including the link training and status state machine (LTSSM).

Figure 18-2. PCIe controller architecture



18.2.1 Transmit Services

The transmit path supports the functionality of the PCI Express transaction layer for packet transmission. Its functions include arbitration, TLP formation, and flow control (FC) credit checking. The transmit path uses a cut-through architecture. It does not implement transmit buffering/queues (other than the retry buffer).

The transmit path service:

- TLP reply from completer. The master can be IMEM, EMEM (DRAM), or GCI memory.
- TLP from requester. The requester can be a CTOP transaction or CLDMA transaction.

Completion lookup table (LUT)

The controller stores and tracks the information in an internal target completion lookup table (LUT). The target completion LUT is watching for received completions corresponding to previously received non-posted requests from the PCIe wire. The receive completion LUT is watching for received PCIe completions on the *wire*, corresponding to previously transmitted non-posted requests.

Completion reply reorder

The inbound requests may be received with same PCIe Tag due to request decomposition, and the requests maybe completed out of order by IMEM banks and EMEM L2 cache units. The transmit path restores the original order of the completions. Reordering is also required when read access that is not aligned to 128B must be split into two transactions over EZnet.

Credit checking

The controller checks that enough flow control (FC) credits are available in the remote device for the specific type of transaction (posted, non-posted, completion) before allowing a transmission of a TLP. TLPs that passed the credit check are arbitrated according to the supported arbitration method. Internally generated completions and messages are also gated by the arbitration logic, though at highest priority, and must also pass the FC credit test before they are accepted for transmission.

The EZnet crossbar is using shared transmit client interface for more than one TLP type (for example, posted and non-posted), and the current request (for example, a posted request) is being blocked due to lack of available FC credits, then that client interface is effectively blocked from sending other requests (for example, non-posted) even though credits might be available for that type.

18.2.2 Receive Services

The receive path implements the mode-specific functionality of the PCI Express transaction layer for TLP packet reception:

- Sort/filter the received TLPs
- Receive completion lookup table (LUT) – used for completion tracking and completion timeout monitoring of transmitted non-posted requests.
- Buffering and queuing of the received TLPs
- Routing of received TLPs

The filtering rules and routing are configurable for all TLP types.

The receive path filter passes the posted and non-posted requests and completion transactions to the receive queues and then via crossbar and EZnet to transaction requester or completer.

The filter separates posted and non-posted TLPs into “valid supported” and “valid unsupported” requests and forwards them to the queue. The filter processes each request and determines each TLP’s destination. The received request TLPs are classified per target:

- TRGT0 – Receive Target 0. This is an internal receive interface used to access Controller registers and application registers. The controller automatically executes any required registers access before automatically generating the completion.
- TRGT1 – Receive Target 1. This is the internal receive interface used for request TLPs. The EZnet target processes the requests and generates the completion.

Receive Completion TLP Processing

Received completions are compared against the receive completion LUT content before presenting the completion to the queue. The controller also implements a completion time-out mechanism (using the receive completion LUT) and notifies the application when a completion, corresponding to a previously transmitted non-posted TLP, does not arrive within a specified time.

The completion queue configured for store-and-forward mode. When a completion lookup has failed or some other completion filtering has failed, then the PCIe controller asserts an abort signal at the end of the transaction. The completion is discarded by the controller and flow control credits are updated as necessary when an abort signal is detected.

PCI Express Configuration Space

The receive controller implements the standard PCI Express configuration space and the application-specific register space.

A CFG TLP has a 6-bit *Register Number Field* and a 4-bit *Extended Register Number* field allowing 1024 DWORDs (4096 bytes) to be accessed.

The controller has 4096 bytes of register PCIe configuration and port logic space per function. This address space is fully accessible from the EZnet. It can be accessed from the PCIe wire using CFG requests. The port logic and application registers parts of this configuration space can also be accessed from the PCIe wire with BAR matched MEM and I/O requests.

Table 18-2. PCIe configuration space

BYE ADDRESS	*REGISTER SPACE*	
0xFFF CONFIG_LIMIT default =0xFFF	**Application Registers**	**PCI Express Extended Configuration Space** (3840 bytes 960 DWORDs)
0x700	**Port Logic Registers**	
0x100	**PCIe Extended Capability Structures** AER, VC, SN, PB, ARI, SPCIE, IOV, TPH, ATS, LTR, L1SS, RBAR	
		PCI Configuration Space (256 bytes 64 DWORDs)
	PCI Standard Capability Structures PM, MSI, PCIE, MSI-X, VPD	
	CapPtr ↑	
0x03F	**PCI Configuration Header Space** (64 bytes / 16 DWORDs)	

- *Details on the NPS PCIe configuration registers are provided in the "NPS-400 Configuration Registers Reference Manual".*

PCI Configuration Header and Capability Registers

The PCI configuration header and capability registers are PCIe controller configuration registers specified by the *PCI Express 3.0 Specification.* Access from the PCIe wire is possible with CFG requests. These registers are also fully accessible from the EZnet without any restrictions.

Port Logic Registers

The port logic (PL) registers are configuration registers which are not specified by the *PCI Express 3.0 Specification* but which are controller specific. Access from the PCIe wire is with CFG requests. These registers are also fully accessible from the EZnet without any restrictions.

Application Register

Access from the PCIe wire is with CFG, MEM, or I/O requests. These registers are fully accessible from the EZnet without any restrictions.

Wire Access

- Any CFG <= CONFIG_LIMIT goes to Controller Registers.
- Any CFG > CONFIG_LIMIT goes to Application Registers.
- Any MEM/IO captured by a BAR whose interface target is TRGT0 goes to Application Registers.
- Possible to memory-map the port logic (PL) register space.
- Not possible to memory-map the PCIe configuration register spaces. They must always be accessed with a CFG request.

18.3 PCIe Controller Operation

This section describes the operations of the PCI Express controller.

18.3.1 Reset Requirements

The types of reset:

- Cold Reset.
  This is the initial reset of the controller after power up.
- Warm Reset (Link Down Reset).
  During normal operation, the Link may go down and the controller requests a Link Reset Request.
- Hot Reset.
  A Downstream port (RC, Downstream Port of SW) can "Hot Reset" an Upstream port (EP, Upstream Port of SW) by sending two consecutive TS1 ordered sets with the Hot Reset bit asserted. This is also referred to as a "training reset".
- Function Level Reset (FLR).
  This is a mechanism to allow specific functions to be reset without affecting other functions or the Link as a whole. FLR applies to both Physical Functions (PFs) and Virtual Functions (VFs).

18.3.2 Initialization

The initialization causes the internal configuration registers assume their default reset values. The CTOP application may hold LTSSM disabled after reset until it is ready to establish a link and to start receiving/transmitting TLPs. When an application needs to update configuration registers as part of the initialization process, it holds LTSSM disabled until it has programmed all the necessary configuration registers. After initializing the necessary configuration registers, an application can enable LTSSM to begin link establishment.

Table 18-3. Initialization sequence

1	Reset Asserted/De-asserted	LTSSM disabled
2	CTOP application can update the configuration registers before link training	
3	PCIe Link Training / Establishment. Start link training	LTSSM enabled
4	Downstream Device Enumeration by Root Complex: (1) Confirm the link is up. (2) Read the configuration space of the downstream devices (3) Program their capabilities (4) Program the base and limit registers of switch ports to reflect the BAR range of the devices enumerated downstream (5) Program the BARs of endpoints	
5	Host Software writes to Bus Master Enable (BME), Memory Space Enable (MSE), and I/O Space Enable (ISE) bits. The EP application logic must not generate any MEM or I/O requests until the host software has enabled the BME. The application logic must monitor the status of the BME.	
6	Start Application Traffic Generation	

18.3.3 Link Establishment

The controller implements the LTSSM function according to the *PCI Express 3.0 specification*. In general, the process for establishing a link is as follows:

1. Upon power-up (or directly out of reset), it is assumed that the power supply becomes stable, and the link partner PHYs reach frequency lock before they attempt to establish a valid Link. When in a valid state, the PHY either communicates a ready status to the controller, or simply begins transmitting and receiving valid data.
2. After bit and symbol synchronizations are complete, the controller initiates the following sequence to establish a link (assuming a valid and properly functioning Link partner):
   2.1. Receiver detection on available lanes for the port.

   2.2. Exchange of training sequences to determine link configuration (for example, link speed, number of lanes, and order).

   - *The controller supports link widths of x1, x2, x4 or x8.*

   2.3. After both partners reach a valid negotiated state, the link state is set up and the LTSSM is in L0.
3. After Link up is achieved, the Data Link Layer (DLL) takes over to manage the link and initialize Flow Control.
4. After Flow Control initialization is complete, the DLL signals the Transaction Layer (TL) that the link is ready to allow transmission/reception of TLP traffic.
5. During normal operation, the LTSSM and DLL continue to manage the underlying link integrity while data traffic is communicated across the PCI Express link.

Runtime Link Width Adjustment Through Detect (Link Goes Down)

When the remote link partner initiates an action to change the link width, then the controller does *not* respond unless you direct it to as follows using the *Link Mode Enable* field (LINK_CAPABLE) of the Port Link Control Register (PORT_LINK_CTRL_OFF), and the *Predetermined Number of Lanes* field (NUM_OF_LANES) of the Link Width and Speed Change Control Register (GEN2_CTRL_OFF).

1. Ensure the link is in the L0 LTSSM state.
2. Program the LINK_CAPABLE field of PORT_LINK_CTRL_OFF. LTSSM uses this in Detect.
3. Program the NUM_OF_LANES field of GEN2_CTRL. This indicates to the LTSSM, the number of lanes to check for exiting from L2.Idle or Polling.Active.
4. Trigger loopback by setting the *Loopback Enable* (LOOPBACK_ENABLE) bit of the Port Link Control register (PORT_LINK_CTRL_OFF) to “1”.
5. Wait for 24 msec. The controller 's LTSSM transitions from L0 -> Recovery -> Loopback.
6. Clear the LOOPBACK_ENABLE field of PORT_LINK_CTRL_OFF. The controller’s LTSSM transitions from Loopback -> Detect.

- *You must not use this process to increase the link width if the link partner did not advertise the capability to upconfigure the link during the configuration state.*
- *This procedure is not the same up/downsizing procedure defined in the PCIe Specification.*
- *The controller cannot autonomously initiate an action to change the link width without using a loopback event.*

Runtime Link Width Adjustment through Recovery (Link Remains Up)

When the remote link partner initiates an action to reduce the link width, then the controller passively responds as follows.

1. Remote partner initiates the link width change by directing LTSSM transition from L0 to Recovery.
2. Controller receives TS1 ordered sets and moves to Recovery.
3. Remote partner moves into Recovery.Idle state, and then directs LTSSM into Configuration.Linkwidth.Start state where the remote partner sends TS1s with lane number =PAD.
4. Controller receives TS1 ordered sets with PAD lane number, and then moves into the Configuration.Linkwidth.Start state.
5. Remote partner asserts Electrical Idle (EI) on the lanes which it does not want to be active.
6. Controller cannot receive correct TS1 ordered sets on these lanes (with EI) and then downsizes the link width at the end of Configuration.Complete state.
7. Both sides link up with new reduced link width and move to L0 state.

- *Upsizing the link is not possible using this process.*

Table 18-4. Runtime link resizing options

Link Resizing Direction	Entering LTSSM Detect State Link Goes Down (Recovery ->Loopback->Detect)	Entering LTSSM Recovery State Link Remains Up (Recovery -> Configuration)
When lanes are not reversed:		
Upsizing	V	X
Downsizing	V	V
When lanes are reversed:		
Upsizing	X Supported when resizing back up to use maximum number of lanes.	X
Downsizing	X	X

18.3.4 Transmit TLP Processing

The following section describe the flow of transmit TLPs through the PCIe controller.

The transmit transactions are originated by either:

- CTOP Cluster DMA read/write transaction of up to 128B.
- Memory/IO read reply completion from IMEM/EMEM/GCI and NPS Registers.

The completion transactions are reordered as they may be completed out of order by EZnet. The reorder buffer supports up to 32 pending split read requests, each split into up to 3 segments of up to 128B.

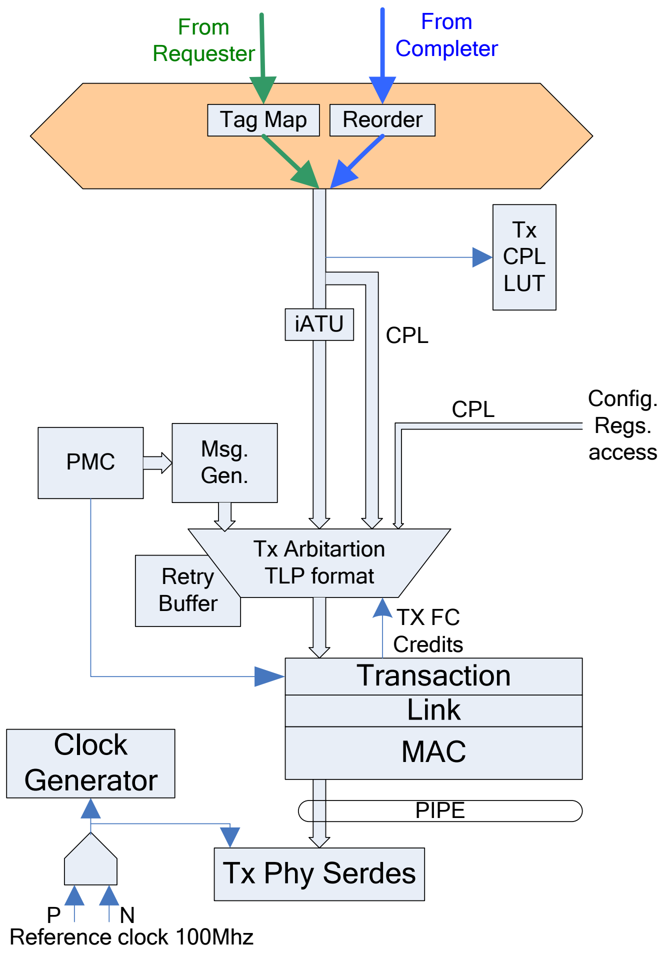
Completion Lookup Table (LUT)

The controller stores and tracks the information in an internal target completion lookup table (LUT).

The transmit completion LUT is watching for received completions corresponding to previously received non-posted requests from the PCIe wire. The LUT converts the Completer ID to the saved PCIe Tag.

Request transaction EZnet ID is converted to PCIe Tag.

Figure 18-3. Internal target completion lookup table (LUT)



18.3.4.1 Transmit Overview

All types of transmit TLPs (P, NP, and CPL) generated by the application travel through the controller in the following flow:

1. The CTOP/CLDMA generates transaction requests and IMEM/EMEM/GCI memory and device Registers generate transaction replies (competition) that are formatted as a TLP transaction transmission request with header information and payload (if applicable).
2. The PCIe controller forms the transaction into a TLP and checks the TLP against the current Flow Control credit availability. If the TLP passes the Flow Control checks and wins the arbitration with TLPs from the internal sources (message generator or PCIe controller internal register access), then the TLP goes to the PCIe Transaction Layer.
3. The controller snoops/stores the necessary TLP information for CPL lookup (for NP requests only).
4. The controller inserts the Sequence Number and LCRC into the TLP and the Retry Buffer stores the TLP.
5. The controller inserts start and end delimiters and performs data scrambling.
6. The controller presents the packet to the Serdes PHY through the PIPE interface.
7. The PHY receives the packet, performs 8b10b encoding and serialization, then sends the packet for transmission on the link.

18.3.4.2 Transmit TLP Arbitration

The PCIe Unit arbitrates between the following for TLP transmission:

- EZnet transactions
- Messages from the Message Generator, triggered by the Power Management Controller (PMC), errors, or SW controlled logic.
- Internally generated completions:
  Internally generated CPLs are responses for type 0 Configuration Read and Write Requests from Upstream Components, memory or I/O-mapped application register space Read and Write Requests, or responses to error conditions (Unsupported Requests).

The priority order for all transmitted TLPs is:

1. Messages (internally-generated).
2. Internally-generated CPLs.
3. Transmit TLPs from EZnet according to one of the following schemes, depending on the number of Virtual Channels (VC).

The controller transmits TLPs and DLLPs according to the following priority, highest priority first:

- Completion of any transmission currently in progress
- High-priority DLLPs:
  - Nak DLLP
  - Ack DLLP
  - Flow control DLLPs required to satisfy Section 2.6 of the *PCI Express Base 3.0 Specification, revision 1.0*
- Retry buffer retransmissions
- TLPs from the transaction layer (in the following order of priority):
  - Internally generated messages from the Message Generator triggered by any of the following:
    - PME

        - Errors
    - Internally generated completions. These are responses for type 0 configuration read and write requests from upstream components, memory or I/O-mapped application register space read and write requests, or responses to error conditions (unsupported requests).
    - TLPs presented by EZnet.
- Flow control DLLP transmissions excluding those required to satisfy Section 2.6 of the *PCI Express Base 3.0 Specification, revision 1.0*
- All other DLLPs
    - Messages (excluding power management)
    - Power management
    - Power management AC

Multi-VC Transmit Arbitration

The VC arbitration is based on programmable weighted round robin arbitration (WRR) using two different arbitration methods for the two groups of VCs:

- Strict Priority for the High-Priority VC (HPVC) group
- RR or WRR for the Low-Priority VC (LPVC) group

Between the two groups of VCs, arbitration is as follows:

- The HPVC group is always the highest priority. Within the HPVC group, priority order is by VC ID. The highest VC ID has the highest priority.
- The LPVC group is of lower priority than the HPVC group. Within the LPVC group, priority is determined by RR or WRR arbitration as described later. Ties within the LPVC group are resolved by client-based RR arbitration.

18.3.4.3 Transmit Retry

There is a retry module in the controller that stores a copy of each transmitted TLP in a retry buffer. The controller resends the TLP unless it receives a positive acknowledgement of receipt from the link partner (an Ack). Therefore a retry can be initiated by the expiration of replay timer or by the receipt of a Nak.

Retry Buffers

The Retry module consists of two buffers: Retry Buffer and Start-of-TLP (SOT) Buffer. The SOT Buffer stores the starting address of each unacknowledged TLP stored in the Retry Buffer. The SOT Buffer is indexed by the Sequence Number of the TLP whose starting address is being stored or retrieved. The selected SOT buffer size allows the Retry buffer to store the maximum number of shortest TLPs (3 DWORDs). The Retry Buffer does not function as a transmit queue. The controller transmits TLPs immediately after they pass arbitration. The copy in the Retry Buffer is only sent in the event that the TLP must be re-transmitted.

Replay Timer

The controller stores TLPs in the retry buffer and resends them unless it receives a positive acknowledgement of receipt from the link partner. Therefore a retry can be initiated by the expiration of replay timer, or by the receipt of a Nak. This is a Replay Timer Timeout error (Correctable) and the controller reports it. The timer limit is set by ACK_LATENCY_TIMER.replay_time_limit. The timer starts when the last symbol of any TLP is transmitted.

18.3.5 Ack Scheduling

The transmission of Ack DLLPs is controlled by the Ack frequency and latency timers. By default, the controller accumulates 255 Ack events before issuing an Ack.

Ack Frequency Timer

The Ack frequency (i.e. number of accumulated Ack events before an Ack is sent) is set by the *Ack Frequency* field in the Ack Frequency and L0-L1 ASPM Control Register (ACK_F_ASPM_CTRL_OFF). By changing the value of this field (or its default DEFAULT_ACK_FREQUENCY) from "0", you can instruct the controller to not accumulate "Ack" events (for received TLPs) before scheduling an Ack for transmission. The controller will then schedule an Ack for every received TLP. The controller normally sends a pending Ack DLLP if there is no competing TLP traffic. When the Ack frequency timer expires then the controller will send the pending Ack and bypass any competing traffic.

Ack Latency Timer

The Ack latency timer monitors the elapsed time since the last Ack DLLP was scheduled for transmission. When it expires, the controller returns an Ack DLLP thereby overriding the Ack frequency timer. The latency timer guarantees that the controller schedules an Ack before the replay timer expires (which would cause a TLP retry event). The latency timer limit is set by the *Ack Latency Timer Limit* field in the Ack Latency Timer and Replay Timer Register (ACK_LATENCY_TIMER).

18.3.6 Receive TLP Processing

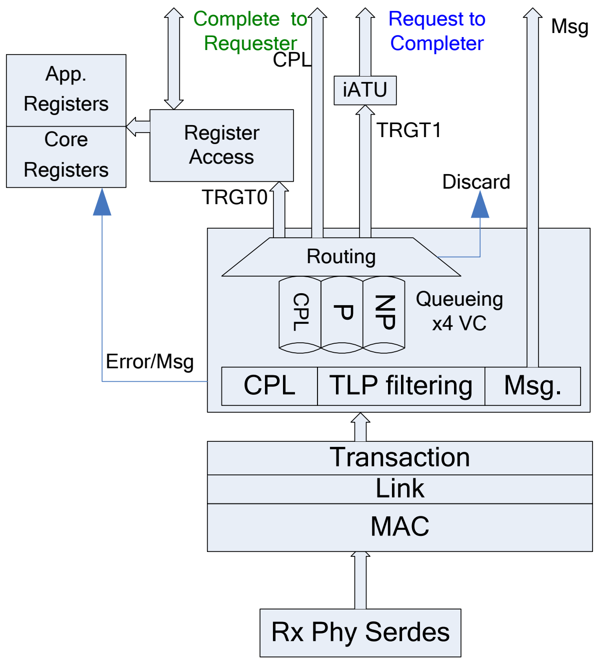
The following section describes the flow of received TLPs through the PCIe controller.

18.3.6.1 Receive Overview

Received transactions travel through the controller in the following flow:

1. The PHY receives a stream of bits and aligns/forms them into 10-bit symbols (Gen1/2) or 130-bit blocks (Gen3).
2. The PHY decodes the 10b stream into an 8b stream (Gen1/2) or a 128b stream with a sync header (Gen3).
3. The PHY retimes the data to the local TX clock and presents it to the PIPE.
4. The controller descrambles and deskews the incoming data, checks for receiver (Gen1/2) or framing (Gen3) errors, then extracts packets.
5. The controller strips off the LCRC and Sequence Number.
6. The controller checks for a malformed TLP.
7. The Rx path filters the transaction based on the transaction type (P, NP, or CPL) and the rules.
8. Filtered transactions are sent to Receive queues.
9. Transactions residing in the Receive queues are presented to the EZnet or locally handled by the Configuration/Application register access module, depending upon the filtration result.

Figure 18-4. Receive TLP processing



18.3.6.2 Receive Filtering

The controller contains a filter module that is responsible for the following tasks:

- Determine the status of a received TLP using filtering rules.
- Determine the destination of a received TLP based on the filtering status.
- Indicate the status of the received TLP.
- Report Errors to AER registers based on filter results, when more than one type of error is detected.

The controller filters and routes received TLPs according to a set of rules determined by the TLP type based on the *PCI Express Base 3.0 Specification* and user-configurable filtering options.

The following general rules apply to all incoming TLPs that are not malformed. By default:

- For a function in device power states D1, D2, and D3hot, the controller only accepts CFG and MSG Requests TLPs for that function. All other incoming request types for that function are treated as Unsupported Requests (UR).
- When the controller detects an error in a received TLP:
  - Discards the TLP.
  - Generates a completion (for non-posted requests) with the completion status set to CA or UR.
  - Sets the status in the PCI-compatible Status register.
  - Sets the status in the AER registers (when you enable AER).
  - Generates an error message.

Notation of filter results:

▪ UR	= Unsupported Request “CPL Status”
▪ CA	= Completer Abort “CPL Status”
▪ CRS	= Configuration Request Retry Status “CPL Status”
▪ SU	= Successful Completion “CPL Status”
▪ UC	= Unexpected CPL
▪ MLF	= Malformed
▪ MA	= (Received) Master Abort set in PCI-compatible Status Register
▪ TA	= (Received) Target Abort set in PCI-compatible Status Register
▪ <blank>	= Filtering rule does not apply to TLP type

Table 18-5 describes the filtering rules (based on the *PCIe Specification*) and the results of the controller’s filter. When a received TLP passes all of the filter rules, then it is considered to have no errors, and is routed to the destination that is configured.

For Non-Posted TLPs, this filter result also determines the status of the Completion that the controller sends back to the requestor.

Table 18-5. Result of Filtering Rules Applied to all TLPs

FILTERING RULE	TLP TYPE					
	MRd IORd	MWr IOWr	CFG	MSG	CPL with UR/CA/CRS status	CPL with SU status
PowerState is not in D0	UR	UR	SU	SU	UC	UC
Address is not within any configured Memory BAR or I/O BAR if it is an I/O Request	UR	UR				
TLP header poison bit is set and the filter mask CX_FLT_MASK_UR_POIS bit is not set	UR	UR	UR	UR	SU	SU
Address within a BAR that is configured to TRGT0 and TLP DW length > 1	CA	CA				
MRd with lock and filter mask CX_FLT_MASK_LOCKED_RD_AS_UR bit is not set	UR					
The Function number of a completer ID within a CFG Request does not match an implemented function within the receiver device and the filter mask CX_FLT_MASK_UR_FUNC_MISMATCH bit is not set			UR			
Configuration type1 TLP request and the filter Mask CX_FLT_MASK_CFG_TYPE1_REQ_AS_UR is not set			UR			
Not Valid Message				UR/MLF		
Illegal payload length of a message				UR		
Vendor MSG Type0 with filter mask CX_FLT_MASK_VENMSG0_DROP bit not set				UR		
Vendor MSG Type1 with r[2:0] to 3'b010 and {Bus#, Dev#, Func#} mis-match				UR		
TLP with ECRC error detected	CA	CA	CA			
Requester ID mis-match					MA/TA	MLF
Requester TAG mis-match					MA/TA	MLF
TAG error (non-pad zero for reserved TAG bits					MA/TA	MLF
Byte Count Mismatch (PCIe Gen2)					MA/TA	UC/MLF
CPL received with status of UR					MA	
CPL received with status of CA					TA	
CPL received with status of CRS					CRS	
CPL received with CRS status and CPL is not a pending Configuration Request					MLF	

Filtering Rules Not Defined in PCIe Specification

When a zero-byte request TLP is received, also called "flush" command, the controller can drop the zero-byte request. Program a bit in the filter mask CX_FLT_MASK_HANDLE_FLUSH to turn on/off this rule.

18.3.7 Receive Routing

This section discusses how the PCIe receive controller routes different TLP types depending on the TLP type and filter result. TRGT0 refers to the internal interface used to access the Configuration registers or Application registers.

EP Mode Routing Overview

The possible destinations of a Posted or Non-Posted Request TLP are TRGT1 interface, TRGT0 interface and "Controller Discard". By default:

- CFG Requests are routed to TRGT0 and then to Configuration registers.
- BAR-matched MEM and I/O Requests are routed to TRGT1.
- MSG requests are decoded internally.

18.3.7.1 Request TLP Routing Rules

The next table shows the applicability of routing rules for Request TLPs, and indicates whether the destination is as stated by the rule when the conditions of the rule are met.

In many cases, the standard routing rules may be masked or ignored by setting the corresponding bit in the Symbol Timer Register and Filter Mask Register 1.

Notation of routing results:

UR	Unsupported Request
CA	Completer Abort
CRS	Configuration Request Retry Status
SU	Successful
No	Destination is not as specified in rule, even when conditions of rule are met
Yes	Destination is as specified in rule, when conditions of rule are met
<blank>	Routing rule does not apply to TLP type

Table 18-6. Routing Rules for Request TLPs

FILTERING RULE	*TLP TYPE*						
	MRd	**MWr**	**CFG**	**I/O**	**Vendor MSG Type 0**	**Vendor MSG Type 1**	**Other MSG**
When a request is filtered with SU status, and is in BAR range.	Yes	Yes	No	Yes			
When a request is filtered with UR/CA/CRS status, and the DEFAULT_TARGET parameter is 0, the TLP is dropped. For NP requests, a CPL is also generated.	Yes	Yes	Yes	Yes	Yes	Yes	Yes
When a request is filtered with UR/CA/CRS status, and the DEFAULT_TARGET parameter is 1, the TLP is dropped.	No	No	No	No	Yes	Yes	Yes
When a request is filtered with UR/CA/CRS status, and the DEFAULT_TARGET parameter is 1, the destination is TRGT1 interface.	Yes	Yes	Yes	Yes	No	No	No
When a *CFG Request* is filtered with SU status and the CFG register address is > CONFIG_LIMIT, TARGET_ABOVE_CONFIG determines the destination.			Yes				

FILTERING RULE	*TLP TYPE*						
	MRd	**MWr**	**CFG**	**I/O**	**Vendor MSG Type 0**	**Vendor MSG Type 1**	**Other MSG**
When a *CFG Request* is filtered with SU status and the CFG register address is < CONFIG_LIMIT, TRGT0 interface is the destination.			Yes				
The TLP is dropped, when the filter mask CX_FLT_MASK_MSG_DROP bit is 0 and the non-Vendor MSG is filtered with SU status							Yes
The TLP is dropped, when the filter mask CX_FLT_MASK_VENMSG0_DROP bit is 0 and the VEN0 MSG is filtered with SU status					Yes		
The TLP is dropped, when the filter mask CX_FLT_MASK_VENMSG1_DROP bit is 0 and the VEN1 MSG is filtered with SU status.						Yes	

Receive Checking

The controller responds with UR, to CfgRd0 requests with an incorrect Function number, it never checks the Bus.Device numbers. The controller responds with UR, to all CFG1 requests.

Bus.Device Number Assignment

The controller snoops every [completed] CfgWr0 request and updates the Bus.Device[1] numbers in its Completer ID.

[1] For SR-IOV (or ARI) enabled devices, there is no captured Device number, as it is simply defined to be 0.

18.3.8 Error Handling

Errors are classified into two levels:

- Correctable Error (CORR).
  This means that the PCIe controller has a way of automatically handling the error. There is no loss of information. For example, a Link CRC (LCRC) error that is fixed by replaying the DLL.
- Uncorrectable Error (UNCORR).
  The PCIe controller cannot fix these and they are classified as:
  - Fatal Error (FATAL). The link is not functioning correctly and may require a link reset.
  - Non-Fatal Error (NONFATAL). The problem is not related to link operation.

The controller implements the following types of error handling.

- PCIe Baseline Capability.
  These reporting capabilities are a minimum set, and are required of all PCI Express devices. Errors are reported through Messages, CPL Status, and the Device Status Register.
- PCIe Advanced Error Reporting (AER) Capability.
  Allows more sophisticated error reporting, control, masking and logging using the PCIe Extended AER Capability Register Structure.

18.3.8.1 PCIe Baseline Capability

The Baseline reporting capabilities are a minimum set, and are required of all PCI Express devices. Errors are reported in three ways:

- Messages Sent to Root Complex (RC)
- CPL Status Errors
- Reporting Through the Device Status Register

The PCI Express Capability Register Structure provides the following support for Baseline Error Reporting.

- Enable/disable error reporting (Device Control Register).
- Provide error status (Device Status Register) for:
  - UR
  - Correctable Error (CORR).
  - Fatal Error (FATAL).
  - Non-Fatal Error (NONFATAL).
- A method for software to force Link Retraining (Device Control Register).

Messages Sent to Root Complex (RC)

Reporting of errors is achieved by sending a notification to the RC (a CPL with UR/CA/CRS status for NP requests, and optionally an error message). The decision to [not] send an error MSG is controlled by a complex set of associated control and status bits. For more details, see the flow diagram in section *6.2.6, Error Message Controls* of the *PCI Express Specification.* Messages sent to the RC are of the ERR_CORR, ERR_NONFATAL, and ERR_FATAL types.

Completion (CPL) Status Errors

Completion (CPL) Status errors for NP requests may be any of the following:

- Unsupported Request (UR)
- Configuration Request Retry Status (CRS)
- Completer Abort (CA)

Reporting Through the Device Status Register

The status is also logged in the Device Status Register for the following errors: UR, FATAL, NONFATAL and CORR. The flow diagram in section *6.2.5, Sequence of Device Error Signaling and Logging Operations* of the *PCI Express Specification* shows the sequence of operations related to signaling and logging of errors detected by a PCIe device. This also covers mapping of PCIe errors to legacy PCI generic error handling such as PERR# and SERR#.

18.3.8.2 Advanced Error Reporting (AER)

AER allows more sophisticated error reporting, control, masking and logging using the optional Extended AER Capability Register Structure.

AER Registers

The AER registers are described in the *PCI Express Base 3.0 Specification, revision 1.0*. You can enable and mask all possible error sources; and Uncorrectable Error errors are assigned a severity. The Correctable set of registers handles (for example) errors arising from bad DLLPs or TLPs. The Uncorrectable set of registers handles (for example) errors arising from UR, ECRC, Malformed TLPs, Buffer Overflow, UC, CA, CPL Timeout and Poisoned TLP. There is a set each (for Correctable and Uncorrectable errors) of the following registers:

- Error Enable Register
- Error Severity Register (Uncorrectable errors only)
- Error Mask Register.

Severity Programming

The Uncorrectable Error Severity register allows each uncorrectable error to be programmed to Fatal or Non-Fatal. The transmission of these error Messages by class (correctable, non-fatal, fatal) is enabled using the Reporting Enable fields of the Device Control Register or the SERR# Enable bit in the PCI Command Register.

The Uncorrectable Error Mask register and Correctable Error Mask register allow each error condition to be masked independently. If Messages for a particular class of error are not enabled by the combined settings in the Device Control register and the PCI Command register, then no Messages of that class will be sent regardless of the values for the corresponding mask register. If an individual error is masked when it is detected, its error status bit is still affected, but no error reporting Message is sent to the Root Complex, and the Header Log and First Error Pointer registers are unmodified. For more details, see Table 18-7. TLP Header Logging for Uncorrectable Errors.

18.3.8.3 Advisory Non-Fatal Error Messages

The PCIe controller supports Advisory reporting for both the Baseline and AER capabilities, which is the configurable withholding of reporting for Non-Fatal errors.

- With Baseline Error Reporting, the controller produces no error message.
- With AER, the controller can instead, signal a non-fatal error with ERR_COR, which serves as an advisory notification to software.

The controller always signals a fatal error with ERR_FATAL. Refer to section *6.2 Error Signaling and Logging* of the *PCI Express Specification*, for details of when an error should be handled as advisory.

The following are two examples of advisory error handling:

UR/CA Advisory

The PCIe controller generally sends a CPL with UR/CA status to signal a uncorrectable error for an NP Request. If the severity of the UR/CA error is non-fatal, the PCIe controller will handle this case as an

Advisory Non-Fatal Error. By default, the PCIe controller will signal the non-fatal error (if enabled) by sending an ERR_COR Message.

If AER is disabled, the PCIe controller sends no error Message for this case. Even though there was an uncorrectable error for this specific transaction, the PCIe controller will handle this case as an Advisory Non-Fatal Error, since the Requester upon receiving the CPL with UR/CA Status is responsible for reporting the error (if necessary) using a Requester-specific mechanism.

UC Advisory

When the PCIe controller receives a UC and the severity of the UC error is non-fatal, the PCIe controller will handle this case as an Advisory Non-Fatal Error. By default, the PCIe controller will signal the error (if enabled) by sending an ERR_COR Message. If AER is disabled, the PCIe controller sends no error Message for this case.

Table 18-7. TLP Header Logging for Uncorrectable Errors

Uncorrectable Error Severity Register	Advisory Non-Fatal Error?	Uncorrectable Error Mask Register	Advisory Non-Fatal Error Mask in Correctable Error Mask Register	TLP Header Logged
x	x	1	x	No
Fatal	x	0	x	Yes
Non-fatal	No	0	x	Yes
Non-fatal	Yes	0	0	Yes

18.3.8.4 Error Detection for Received TLPs

The controller performs all mandatory error detections, the error report mechanism based on the *PCI Express Specification*, and some optional error detections.

When the controller detects an error in a received TLP, it normally performs the following:

1. Discards the TLP.
2. Generates a completion (for non-posted requests) with the completion status set to CA or UR.
3. Sets the status in the PCI-compatible status register.
4. Sets the status in the AER registers (when you enable AER).
5. Generates an error MSG.

The following table indicates how some of the more common low level errors are classified.

Table 18-8. Error Detection for Received TLPs

ERROR TYPE	*TLP TYPE*	*POSSIBLE CAUSES (INCOMPLETE)*
UR (Unsupported Request)	Request	▪ Poisoned TLP (EP bit is 1) ▪ No BAR match ▪ MRd length > Max_Read_Request_Size
CA (Completer Abort)	Request	CFG Request TLP length > 1 DW
CPL TimeOut	Request	CPL of a NP request not received.
Unexpected CPL	CPL	▪ TAG mismatch. ▪ Requester ID (RID) mismatch.
ECRC error	Any	Corrupted TLP
Malformed TLP	Any	Bad TLP header. For example, an invalid type or format field.
Bad DLLP	Any	▪ LCRC ▪ Other Data Link Layer (DLL) errors ▪ PHY error.

18.4 Messages

The following section describes the processing of messages through the PCI Express controller.

18.4.1 Message Generation

Messages that are transmitted by the PCI Express controller are derived from the following sources:

- The controller (automatically)
- The controller (triggered by application)
- SW

ID Based Ordering (IDO) is not supported for internally generated messages.

Table 18-9. Messages that the Controller Transmits

MESSAGE SOURCE (TYPE)	*NOTES*
Power Management (Msg)	Internally triggered.
Error Signaling inside the controller (Msg)	Internally triggered COR_ERR / ERR_NONFATAL / ERR_FATAL. See Error Handling for more details.
LTR Clear (Msg) LTR Request (Msg)	Internally triggered. See “Latency Tolerance Reporting (LTR) Message Generation” below.
Direct Supply of any class of Message (Msg/MsgD)	Application generated.
Indirect Supply of any class of Message (Msg/MsgD)	Internally triggered. See “iATU Outbound Msg Handling” in the Outbound iATU Operation section for more details on generating Msg/MsgD from MWr/IOWr using an internal address translation unit (ATU).
Error Signaling from the application (Msg)	Internally triggered. The controller generates Error Signaling Messages in response to application requests.

Latency Tolerance Reporting (LTR) Message Generation

The LTR feature is not supported for Virtual Functions (VF).

The controller generates LTR messages in response to requests made by an application.

The actual values in the LTR Message that was transmitted by this Upstream Port are captured in the LTR Latency Register.

The controller automatically generates a new LTR message with the *Requirement* bits set to 0, for both Snoop and No-Snoop latencies when:

- A function is directed to a non-D0 state through a write to the Power State field of the Power Management Control and Status Register, and the LTR mechanism is active since the last DL_Down to DL_Up transition,
  or
- The LTR mechanism is disabled by clearing the LTR Mechanism Enable field of the Device Control 2 Register. This field is cleared through a CFG write or a function level reset (FLR).

You must not send an LTR message when the *LTR Mechanism Enable* bit of the Link Control 2 Register is 0. When this bit is 0, the controller does not block transmission of LTR Messages that you generate.

18.4.2 Message Reception

The PCI Express controller can receive the following types of messages.

Table 18-10. Types of Received Messages

MESSAGE SOURCE (TYPE)	***EP MODE***
Power Management (Msg)	PME_Turn_Off.
Slot Power Limit (MsgD)	Set_Slot_Power_Limit Support Message.
Error Signaling from Downstream Component (Msg)	NA
Locked Transaction (Msg)	Unlock Message.
Optimized Buffer Flush Fill (OBFF) (Msg)	OBFF Messages received.
ATS Invalidate Request (Msg)	See PCI-SIG Address Translation Services (ATS).

Routing of Received Messages

All error-free MSG requests are decoded internally. When a MSG request is filtered with UR/CA/CRS status, the TLP is always dropped.

- *By default received messages (Msg/MsgD) are delivered (without the payload for MsgD).*

OBFF Message Reception

When you have not enabled OBFF in the Device Control 2 register, or when the Traffic Class is non-zero; then the controller generates an UR CPL when it receives an OBFF message.

18.5 Flow Control

PCI Express defines a differentiated, credit-based Flow Control system to prevent overflows at the receiver. In contrast to the simple XON/XOFF type flow control of the Ethernet protocol, the PCI Express Flow Control system requires that the consumer of data advertise the available buffer space for each type and priority of traffic. The Flow Control mechanism is divided into two phases: the initialization phase and the update phase. The controller automatically performs both of these phases.

The Flow Control for VC0 must be initialized following Link initialization, but prior to sending normal traffic. This initialization is performed by the Flow Control Initialization state machine. The initialization process involves exchanging information with the Link partner about the size of the receiver's buffers for each type of packet data: P, NP, and CPL. Header and payload buffers for each of these types are tracked and reported independently. Flow Control must be initialized for Virtual Channel 0 (VC0) before the Data Link Layer's Link state moves into the DL_ACTIVE state, and normal traffic can begin flowing. Additional VCs (if any) are initialized following this, intermingled with the regular traffic already flowing on VC0. Initialization of other VCs begins when the VCs are enabled. The VC0 traffic has priority over non-VC0 flow control initialization.

The controller provides a configurable solution to choose the number of credits to advertise per type and per VC. It can be configured to support multiple VCs as well as infinite credits. The controller performs all required Flow Control protocol handshakes. The controller does not return Flow Control credits for packets that have Data Link Layer errors.

Initial Credits and Receive Buffer Sizes

The default buffer sizes and credits for each TLP type are automatically calculated from the number of lanes, the maximum PCIe payload (128B), flow control update latencies, internal delays, and the PHY latency.

You can determine the transmit posted, non-posted, and completion credits that are advertised by the receiver at the other end of the link by reading the following port logic registers:

- Transmit Posted FC Credit Status (TX_P_FC_CREDIT_STATUS)
- Transmit Non-Posted FC Credit Status (TX_NP_FC_CREDIT_STATUS)
- Transmit Completion FC Credit Status (TX_CPL_FC_CREDIT_STATUS)

User can override the default flow control update latency timer value using the *FC Latency Timer Override Value* field (TIMER_MOD_FLOW_CONTROL) in the Queue Status register (Q_STATUS).

18.6 Internal Address Translation Unit (iATU)

This is a local address translation implementation and is not related to the PCI-SIG ATS specification support. When address translation support is enabled for the controller, the controller uses the address translation unit (ATU) to replace the TLP address and TLP header fields in the current TLP request header.

Address translation is used for mapping different address ranges to different memory spaces supported by NPS-400.

TYPE translation is also supported. Without address translation, your application address is passed to/from the PCIe TLPs directly. Users can configure (by software) the iATU to implement a custom address (and TYPE/FORMAT) translation scheme.

Inbound Features:

- Match mode operation for MEM, I/O, CFG, and MSG TLPs. No address translation for CPL.
- Supports TYPE translation through TLP TYPE header field replacement for MEM or I/O types to MSG/CFG types.
    - This includes translation from P to NP (for example, MWr to CfgWr0).
    - No TYPE translation from CPL TLPs.
- Programmable TLP header per region for the following fields for TLP field *replacement*.
    - TYPE / TD / TC / AT / ATTR / MSG Code
    - Function Number (Physical and Virtual).
- Up to 16 Address Regions based on programmable registers for location and size.
- x16 programmable enable/disable per region.
- Automatic format (FMT) field translation between 3 DW and 4 DW for 64-bit addresses.
- Invert Address Matching Mode to translate accesses outside of a successful address match.
- ECAM Configuration Shift Mode to allow a 256 MB CFG1 space to be located anywhere in the 64-bit address space.
- Supports regions from 64 KB to 4 GB in size.

Outbound Features:

- Address Match Mode operation for MEM and I/O, CFG, and MSG TLPs. No address translation for CPLs. Selectable BAR Match Mode operation for I/O and MEM TLPs.
    - TLPs destined for TRGT0 (internal registers) will **not** be translated.
- Programmable TLP header per region for the following fields for *matching*.
    - TYPE / TD / TC / AT / ATTR / MSG Code
    - Function Number (Physical and Virtual).
- Up to 16 Address Regions based on programmable registers for location and size.
- x16 programmable enable/disable per region.
- Automatic format (FMT) field translation between 3 DW and 4 DW for 64-bit addresses.
- Invert Address Matching Mode to translate accesses outside of a successful address match.
- Configuration Shift Mode. Optimizes the memory footprint of CFG accesses destined for the Response Code defines the CPL status to return for accesses matching a region.
- Supports regions from 64 KB to 4 GB in size.

18.6.1 Programming

The iATU registers are in the PCIe controllers' Port logic register space.

Table 18-11. iATU Registers

iATU Viewport Register
iATU Region Control 1 Register
iATU Region Control 2 Register
iATU Region Lower Base Address Register
iATU Region Upper Base Address Register
iATU Region Limit Address Register
iATU Region Lower Target Address Register
iATU Region Upper Target Address Register
iATU Region Control 3 Register

18.6.2 Outbound iATU Operation

Address Match Mode

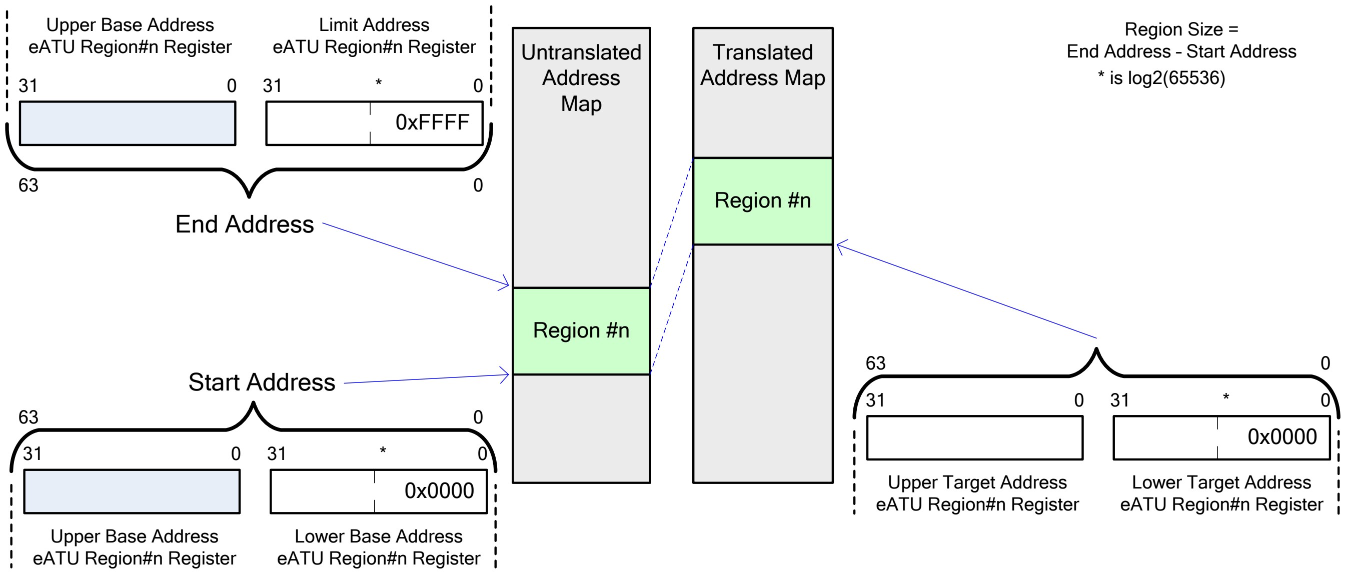
The address field of each request MEM and I/O TLP is checked to see if it falls into any of the enabled[2] address regions defined by the 'Start' and 'End' addresses. If an address match is found, then the TLP address field is modified as follows:

```
Address = Address - Base Address + Target Address
```

and the TYPE, TD, TC, AT, "Function Number", and ATTR TLP header fields are replaced with the corresponding fields in the iATU Control 1 Register. If your application address field matches more than one of the 16 address regions, then the first (lowest of the numbers from 0 to 15) enabled region to be matched is used.

If there is no address match then the address is untranslated. In the Outbound direction (only) the TLP header information (for fields that are programmable) will come from the relevant fields.

Figure 18-5. iATU Address Region Mapping: Outbound and Inbound (Address Match mode)


Upper Base Address eATU Region#n Register
Limit Address eATU Region#n Register
31
0
31
*
0
0xFFFF
63
0
End Address
Untranslated Address Map
Translated Address Map
Region #n
Region #n
Region Size = End Address - Start Address
* is log2(65536)
Start Address
63
0
31
0
31
*
0
0x0000
Upper Base Address eATU Region#n Register
Lower Base Address eATU Region#n Register
63
0
31
0
31
*
0
0x0000
Upper Target Address eATU Region#n Register
Lower Target Address eATU Region#n Register

[2] If the 'Region Enable' bit of the 'Region Control 2 Register' is '0', then that region is not used for address matching.

The upper 32 bits of the Target Address Register will always form the upper 32 bits of the translated address because:

- The maximum region size is 4 GB.
- A region may not cross a 4 GB boundary.

The 64 KB specifies the minimum size of an address translation region. The lower 16 bits of the Base, Limit and Target registers are zero and all address regions are aligned on 64 KB boundaries.

RID BDF Number Replacement

When there is a successful address match on an Outbound TLP, then the Function number used in generating the 'Function' part of the Requester ID (RID) field of the TLP is taken from the 3-bit *Function Number* field of the iATU Control 1 Register. The value in this field must be 0x0 unless MultiFunction operation in the controller is enabled.

If SR-IOV is enabled, and the *Virtual Function Active* field of the iATU Control 3 Register is set, then the 'Function' part of the Requester ID (RID) field of the TLP is formed from by looking at the *Function Number* of the iATU Control 1 Register, VF Offset, VF Stride, Primary Bus number, and the 8-bit *Virtual Function Number* field of the iATU Control 3 Register.

iATU Outbound Msg Handling

The iATU supports TYPE translation/conversion of MEM TLP's to Msg/MsgD TLPs. This supports applications that are unable to generate Msg/MsgD type TLPs natively. When there is a successful address match on an Outbound MEM TLP, and the translated TLP TYPE field is 'Message' (that is, the *TYPE* field of the iATU Control 1 Register is 10xxx); then the Message Code field of the TLP is set to the value in the *Message Code* field of the iATU Control 2 Register.

Note: The iATU does not translate outbound messages.

FMT Translation

The iATU automatically sets the TLP format (FMT) field for 3DW when it detects all zeroes in the upper 32-bits of the *translated* address. Otherwise, it sets it to 4DW when it detects a 64-bit address (that is, when there is a '1' in the upper 32-bits of the translated address). If the original address and the translated address are of different format, the iATU ensures that the TLP header size matches the translated address format.

Invert Feature

Normally, an address match on an Outbound TLP occurs, occurs when the untranslated address is in the region bounded by the Base Address and Limit Address. When the Invert feature is activated, an address match occurs when the untranslated address is NOT in the region bounded by the Base Address and Limit Address. This feature is activated by setting the *Invert* field of the iATU Control 2 Register.

Function Number Translation Bypass Feature

In this mode, the function number of the translated TLP is taken from CTOP MSID and not from the *Function Number* field of the iATU Control 1 Register or the *Virtual Function* field of the iATU Control 3 Register. You can activate the function number bypass mode by setting the *Function Number Translation Bypass Enable* field in the iATU Control 2 Register to "1".

This is useful for SR-IOV applications where the CTOP MSID configuration is used to identify the source (PF or VF) of a request. There is then no need for the iATU to translate the PF and VF information.

No Address Match Result

When there is no address match then the address is untranslated but the TLP header information (for fields that are programmable) comes from the CTOP MSID configuration.

Writing to a MRdLk Region

When there is a successful address match for an Outbound WRITE, and the TYPE header field — as replaced with the TYPE field in iATU Control 1 Register – is MRdLk, then the TYPE header field will be set to MEM (that is 00000b).

18.6.3 Inbound iATU Operation

The main difference between inbound and outbound iATU operation is that the TLP type is never changed in the inbound direction. Instead, the type field is used for more precise matching. Other fields can also be optionally used to further refine the matching process.

Another difference is that for MEM and I/O TLPs, use BAR matching (vs. address matching).

If there is no match then the address is untranslated.
In addition,

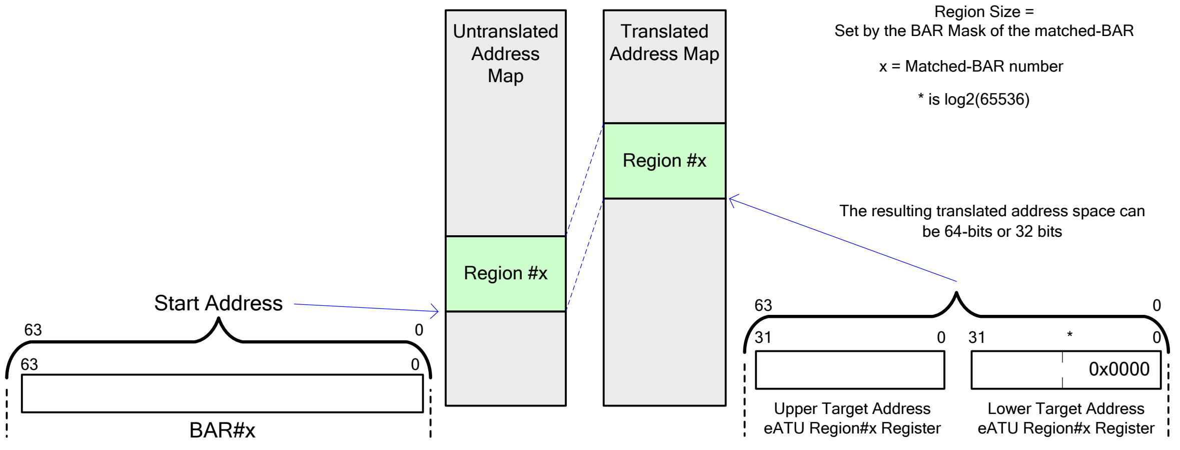
- TLPs destined for TRGT0 (internal configuration/application registers) will not be translated.
- TLPs that are not error-free will not be translated.
- Address translation of I/O and MEM is supported.

MEM or I/O	BAR Match Mode
CFG0	Accept Mode

BAR Match Mode: Looking for an address match is a two-step process.

1. The address field of MEM and I/O *(only)* request TLPs is checked by the standard internal *PCI Express BAR Matching Mechanism* to see if it falls into any address region defined by the enabled BAR Addresses and Masks.
2. If a matched BAR was found, then that matched BAR ID is compared by the iATU to the *BAR Number* field in the iATU Control 2 Register for all enabled regions.

Figure 18-6. iATU Address Region Mapping: Inbound (BAR Match mode): 64-bit BAR


Untranslated Address Map
Translated Address Map
Region Size = Set by the BAR Mask of the matched-BAR
x = Matched-BAR number
* is log2(65536)
Region #x
Region #x
The resulting translated address space can be 64-bits or 32 bits
Start Address
63
0
63
0
BAR#x
63
0
31
0
31
*
0
0x0000
Upper Target Address eATU Region#x Register
Lower Target Address eATU Region#x Register

CFG Handling

CFG TLPs are normally routed to TRGT0. These will **not** be translated. Only CFG0 TLPs routed to TRGT1 will be translated. Inbound address translation for CFG0 TLPs will operate in one of two matching modes as determined by the 'Inbound CFG0 Match Mode' field in the iATU Control 2 Register.

Routing ID Match Mode: The operation is similar to Outbound iATU Operation. The Routing ID of the Inbound CFG0 TLP must fall within the Base and Limit of the defined iATU region for matching to proceed. The iATU interprets the Routing ID (Bytes 8 to 11 of TLP header) as an address. This corresponds to the upper 16 bits of the address in MEM and I/O transactions.

Accept Mode: CFG0 TLPs should always be accepted and processed even if the bus number does not match the current Bus number of the device. This mode follows that behavior. The Routing ID of received CFG0 TLPs will be ignored when determining a match.

CFG1 Transactions: For CfgRd1/CfgWr1 transactions the Base and Limit Address could enclose the entire 32-bit 4G memory space with Routing ID forming the upper 16 bits. The Target Address maps these CFG transactions to anywhere in application address space.

CFG Shift Feature: Inbound CFG transactions (routed for TRGT1) can exist anywhere in address space, because the Routing ID or BDF, is processed by the PCIe receive controller filter. This BDF changes according on the PCIe bus topology. A compressor feature (CFG Shift Feature) can be enabled by setting the *CFG Shift* bit of the iATU Control 2 Register. Bits [15:12] of the 3rd DWORD of CFG TLPs are reserved. The compressor feature uses this to reduce the memory requirement. This shifts/maps the BDF (bits [31:16] of 3rd header DWORD, which would be matched against the Base and Limit Addresses) of the incoming CfgRd0/CfgWr0 down to bits [27:12] of the translated address.

Optional Matching Fields

A successful BAR match can be optionally gated by successful matching of the following programmable TLP header fields (per region):

- TYPE / TD / TC / AT / ATTR
- MSG Code (MSG TLP's only)
- Function Number (MEM, I/O or CFG TLPs only)
- Virtual Function Number (MEM or I/O TLPs only)

For each of the above fields in the iATU Control 1 Register (iATU Control 2 Register for MSG) there is an associated *Match Enable* bit in the iATU Control 2 Register. Address translation will only proceed if all enabled field-matches are successful.

Response Code Feature

When the 'Response Code' field of the Inbound iATU Control 2 Register is set to a value other than 00b, it will determine the CPL Status of the CPL TLP sent in response to a successfully matched NP TLP. This can be set to Unsupported Request (UR) or Completer Abort (CA). When the error response field is set to 00b, then the normal receive filter response for this TLP is used.

Fuzzy Type Match Mode

When enabled, the iATU relaxes the matching of the TLP TYPE field against the expected TYPE field so that

- CfgRd0 and CfgRd1 TLPs are seen as identical. Similarly with CfgWr0 and CfgWr1.
- MRd and MRdLk TLPs are seen as identical
- The Routing field of MsgD TLPs is ignored

For example, CFG0 in the TYPE field in the iATU Control 1 Register will match against an Inbound CfgRd0, CfgRd1, CfgWr0 or CfgWr1 TLP. To enable this feature, then set the *Fuzzy Type Match Mode* bit of the iATU Control 2 Register.

FMT Translation

The iATU automatically sets the TLP format (FMT) field for 3DW when it detects all zeroes in the upper 32-bits of the *translated* address. Otherwise, it sets it to 4DW when it detects a 64-bit address (that is, when there is a '1' in the upper 32-bits of the translated address). If the original address and the translated address are of different format, the iATU ensures that the TLP header size matches the translated address format.

Invert Feature

Normally, an address match on an Outbound TLP occurs, occurs when the untranslated address is in the region bounded by the Base Address and Limit Address. When the Invert feature is activated, an address match occurs when the untranslated address is **not** in the region bounded by the Base Address and Limit Address. This feature is activated by setting the *Invert* field of the iATU Control 2 Register.

18.7 PCI-SIG Address Translation Services (ATS)

ATS is a complete specification that describes an end-to-end protocol for offloading address translation resources in the platform. An EP issues ATS Translation Requests (**MRd** request with AT header field set to 01b) to a Translation Agent (TA) in the RC, to populate a locate Address Translation Cache (ATC) with translated addresses. The No Write (NW) header bit, when set in a translation request, indicates that the function is requesting read-only access for this translation. After the local ATC is populated, the EP may use translated addresses directly from its local ATC, when generating Memory Requests. The EP sets the AT field of the Memory Request to 10b to indicate to the RC, that the address is already translated.

When a TA determines that a function should no longer maintain a translation within its local ATC, the TA initiates the ATS Invalidation Protocol. This consists of a the RC sending a single Invalidation Request Message (**MsgD** with two DWORDs of payload) on the transmit application interface, with the EP returning one or more Invalidation Completion Messages (**Msg**). The PCIe controller does not implement the ATS cache. For a fuller description of ATS, see the *PCI Express Address Translation Services 1.1 Specification, January 26 2009*.

The ATS Capability structure is instantiated in each Physical and Virtual Function.

- *The controller does not support Page Request Services.*
- *The application SW is responsible for generating the ATS Request or Invalidation Completion.*
- *All ATS Invalidate Requests pass through to TRGT1 regardless of the filtering rule settings.*
- *Translation requests (AT =“01”) are only supported for a single address (length =two DWORDs).*
- *Requests for multiple translations are not supported.*

18.8 Gen2 5.0 GT/s and Gen3 8.0 GT/s Operation

The PCIe controller supports all of the non-optional Gen2 5.0 GT/s features defined in the *PCI Express Base Specification*.

The PCIe controller operates at 62.5 MHz for Gen1, 125 MHz for Gen2, or 250 MHz for Gen3.

If bit 17 *Directed Speed Change* of the Gen2 Control Register is set to ‘1’, then the LTSSM will initiate a speed change after the Link is initialized. The default value of this register is the '0'.

A change to Gen3 speed will occur if both sides of the Link advertised support for Gen3 rate. A change to Gen2 speed will occur if both sides of the Link advertised support for Gen2 rate and did not advertise support for Gen3 rate.

18.8.1 Gen3 8.0 GT/s Transmitter Equalization (EQ)

The controller performs Link Equalization during Link Training to improve signal quality by adjusting the Transmitter and Receiver equalization parameters for each Lane on each side of the Link.

Defining EQ Master and Slave

An EQ Master is a port that is currently adjusting the remote transmitter's settings. This can be an Upstream port (USP) in EQ Phase 2 or a Downstream port (DSP) in EQ Phase 3. An EQ Slave is a port that is currently receiving and applying coefficients (requested by the remote EQ Master) to its local transmitter. This can be a USP in EQ Phase 3 or a DSP in EQ Phase 2.

The EQ Slave mode for the mapping of presets to coefficients is

- Dynamic PHY: The coefficients are dynamically mapped in the PHY.

EQ Master Feedback Modes

There are two EQ Master Feedback modes for determining the optimal equalization settings, programmable using the Gen3 EQ Control Register.

- Figure of Merit (FOM)
- Direction Change with optional Convergence Support.

18.9 Alternative Routing-ID Interpretation (ARI)

ARI allows Endpoints to support more than eight PFs (Physical Functions). It is an optional feature for non-SR-IOV Endpoints, and it is a required feature for SR-IOV Endpoints.

Features and limitations:

- PCIe controller supports a maximum total of 4 Physical Functions.
- The PF RIDs are sequential and not sparse/non-sequential.
- When a device supports ARI (even if ARI is not enabled in the system), then the following are determined solely by settings in Function 0 (PF0):
    - Max Payload Size
    - Clock Power Management
    - ASPM Control
    - Common Clock Configuration

18.10 Function Level Reset (FLR)

Function Level Reset is a mechanism to allow specific functions to be reset without affecting other functions, or the Link as a whole. FLR applies to both Physical Functions (PFs) and Virtual Functions (VFs). FLR is an optional feature for non-SR-IOV Endpoints, but mandatory for SR-IOV Endpoints.

Features and limitations:

- FLR must be enabled for SR-IOV operation.
- FLR implements per-function resets for PFs and VFs.
- The driver should put the function into a quiescent state before initiating an FLR.
- The controller will return Unsupported Request (UR) CPLs for requests to functions in FLR.

The application must ensure that all outstanding requests to the function have completed before initiating an FLR.

18.10.1 FLR Operational Notes

The controller concurrently executes the following actions during FLR of a function:

- Delivers from the receive queues to your application, all requests and completions for the function that arrived *before* FLR started.
- Silently discards (following update of flow control credits) all new requests and completions that it receives for the function. For non-posted requests, the controller also transmits a completion with UR status.
- Clears all entries in the completion LUT for that function. This causes the *Transaction Pending* bit in the Device Status register for that function to be cleared.
- While a function is required to complete the FLR operation within 100 ms.

When a PF is being reset, it has to reset all of the corresponding VFs of that PF.

The software driver should put the function into a quiescent state before initiating an FLR.

Stale Completions

An FLR causes the function to lose track of its outstanding non-posted requests. Any corresponding completions that arrive after the controller has completed the FLR or that were present in the receive queue at the time your software initiated FLR are referred to as being stale.

During FLR, the controller delivers from the receive queues to your application, all requests and completions for the function that arrived *before* FLR started.

The following scenario can cause data corruption:

- There are outstanding non-posted requests from a function.
- The software issues an FLR for that function.
- FLR for the function completes.
- There are stale completions in the receive buffer or on the wire.
- The software re-enables the function for operation without processing the stale completions.
- Stale completions arrive and cause data corruption by being mistaken by the function as belonging to requests issued after the FLR.

There is a software algorithm for avoiding such data corruption from stale completions in the implementation note in Chapter 6.6.2, *Function-Level Reset (FLR)* of the *PCI Express Base 3.0 Specification, revision 1.0*.

18.11 Single Root I/O Virtualization (SR-IOV)

This section describes the SR-IOV features implemented.

The PCI Express controller implements revision 1.1 of the *Single Root I/O Virtualization and Sharing Specification (SR-IOV)*, which defines extensions to enable multiple System Images (SI) running on a single Root to share PCI Express hardware resources.

General features and limitations:

- Every VF supports:
  - Error Handling
  - MSI-X Capability
  - PCI Express Capability
  - ARI Capability
  - AER Capability
- PCIe controller supports:
  - Maximum total of 4 Physical Functions (PFs)
  - Maximum total of 128 Virtual Functions (VFs) over all PFs.
- Programmable allocation of VFs over PFs (this is not VF migration, which is not supported).
- Supports two sets of VF Stride, First VF Offset, InitialVFs, and TotalVFs registers per PF; one each for ARI and non-ARI Hierarchies. Selection is performed by host software through the 'ARI Capable Hierarchy' bit of the Control register in the PF0 SR-IOV Capability Structure.
- VF Offset must be a power of two (that is, 1, 2, 4, 8, ..) greater than or equal to the number of PFs.
- VF Stride per PF must be a power of two (that is, 1, 2, 4, 8, ..) greater than or equal to the number of PFs.
- The controller processes a CFG Request targeting Device Number !=0, as an Unsupported Request (UR).

SRIOV and ARI Capable Hierarchy Bit

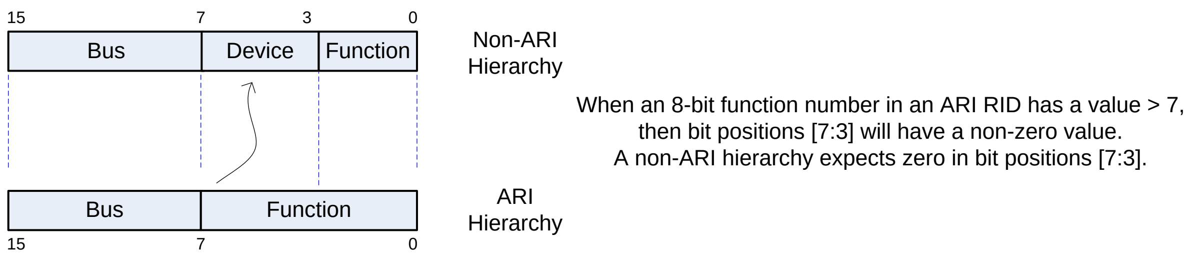
The *ARI Capable Hierarchy* bit in the SR-IOV Capability for PF0 is used by the system, to inform the SRIOV device whether the hierarchy above the device supports ARI or not. When the hierarchy supports ARI, the Device is permitted to locate VFs in Function numbers 8 to 255 (which corresponds to setting the Device Number bits of the RID to a non zero value). When the hierarchy doesn’t support ARI, the Device is required to locate VFs in such a way that the Device Number bits of the RID are all zero. The PCIe controller supports these different requirements by providing two different sets of VF Stride and First VF Offset, one to be used when the ARI Capable Hierarchy bit is set, the other when the ARI Capable Hierarchy bit is not set. The default values provided by the controller for both sets of values should suit most applications. It is possible to reprogram these values by accessing the Stride and Offset registers. When you reprogram the values, care must be taken to ensure that the new values are supported by the controller.

The determination of the VF RID depends on offset and stride settings now advertised by the device. In the general case these are per-PF settings (that is, different PFs may have different settings). The stride and offset values may also depend on how the system configures NumVFs. All VFs (within a single PF) have RIDs that are automatically calculated by the controller, by adding the VF Stride to the previous VF RID. The first VF RID is calculated by adding the First VF Offset to the PF RID.

You must correctly program the values of VF Stride, First VF Offset, InitialVFs, and TotalVFs registers for non-ARI hierarchies, such that the controller generates RIDs with “Device Number == 0”. The generation of RIDs from PF and VF numbers, is not dependent on the 'ARI Capable Hierarchy' bit. Therefore, when you configure the controller for SR-IOV, and set the 'ARI Capable Hierarchy' bit of the

Control register in the PF0 SR-IOV Capability Structure to zero, it will still generate RIDs based on VF Offset and VF Stride. If you do not program these values correctly, then the controller will generate RIDs with non-zero Device Numbers.

Figure 18-7. Interpretation of Device Number by Non-ARI Hierarchy


15
7
3
0
Bus
Device
Function
Non-ARI Hierarchy
When an 8-bit function number in an ARI RID has a value > 7, then bit positions [7:3] will have a non-zero value. A non-ARI hierarchy expects zero in bit positions [7:3].
Bus
Function
ARI Hierarchy
15
7
0

When the 'ARI Capable Hierarchy' bit of the Control register in the PF0 SR-IOV Capability Structure is zero, indicating a non-ARI hierarchy, then the controller processes a CFG Request targeting Device Number !=0 as an Unsupported Request (UR).

Dynamic Virtual Function Allocation

You can change the allocation of VFs to PFs at reset (this is not VF Migration, which the controller does not support). For example, host software can allocate all VFs to any particular PF, or distribute the VFs across a different subset of PFs. Your firmware reprograms the InitialVFs field in the SR-IOV capabilities of each PF. You must disable all VFs before remapping.

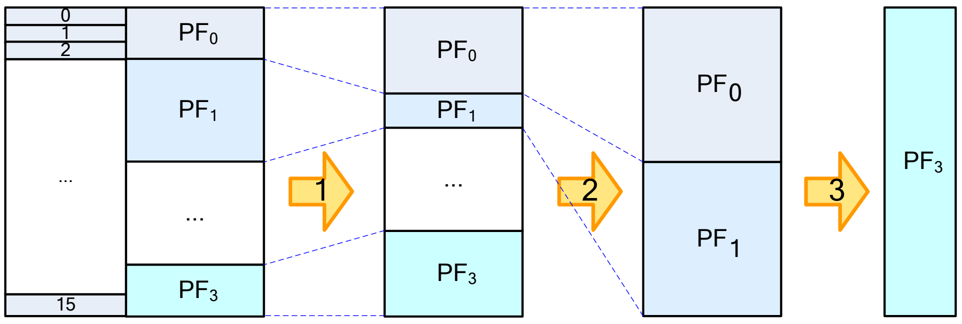
The figure below illustrates an example in which the VF-to-PF allocation is changed three times.

1. The number of VFs assigned to PF_0 and PF_3 is increased at the expense of PF_1.

2. All VFs are equally distributed across PF_0 and PF_1, with no VFs allocated to other PFs.

3. All VFs are assigned to PF_3.

Figure 18-8. Example of Dynamic VF Allocation



18.12 CPL Timeout Ranges

Timeout ranges are supported as defined in the *PCI Express 3.0 Specification*. The Device Capabilities 2 Register (Offset 24h) shows support for all ranges. The Device Control 2 Register (Offset 28h) will have a reset value equal to the default value in the spec: "0000b Default range: 50 us to 50 ms". If the default value is used then the timeout will be in "Range B: 0101b: 16 ms to 55 ms." This range was chosen for the default because the *PCI Express 3.0 Specification* states, "It is strongly recommended that the CPL Timeout mechanism not expire in less than 10 ms." Table 18-12 illustrates the specification values versus the PCI Express controller values for the ranges.

Attention: ***As the PCIe specification states, "This mechanism is intended to be activated only when there is no reasonable expectation that the CPL will be returned, and should never occur under normal operating conditions."***

Table 18-12. Comparison of PCIe Specification and PCIe Controller CPL Timeout Ranges

RANGE	*ENCODING*	*SPEC MINIMUM*	*SPEC MAXIMUM*	*PCIE CONTROLLER MINIMUM*	*PCIE CONTROLLER*
Default	0000b	50 µs	50 ms	28 ms	44 ms
A	0001b	50 µs	100 µs	65 µs	99 µs
A	0010b	1 ms	10 ms	4.1 ms	6.2 ms
B	0101b	16 ms	55 ms	28 ms	44 ms
B	0110b	65 ms	210 ms	86 ms	131 ms
C	1001b	260 ms	900 ms	260 ms	390 ms
C	1010b	1 s	3.5 s	1.8 s	2.8 s
D	1101b	4 s	13 s	5.4 s	8.2 s
D	1110b	17 s	64 s	38 s	58 s

18.13 Calculating Gen1 PCI Express Throughput

This section defines the throughput calculation (primarily) with respect to the PCI Express core in Gen1 2.5 GT/s mode.

PCI Express bandwidth is 2.5 Gb/s (gigabits per second), per lane, when operating at the Gen 1 data rate.

Because data is encoded using 8b10b encoding, the effective maximum throughput is 250 MB/s (megabytes per second), per lane, calculated as follows:

- 2.5 Gb * 8b/10b =2 Gb * 1B/8b =250 MB/sec per lane

The effective throughput on the link is the payload throughput after all the PCIe protocol's overheads have been factored out. The key protocol features are:

- 8b10b encoding at the physical layer. This takes away 20% of the raw bandwidth.
- Acknowledge and flow control update packets at the Data Link Layer (DLLPs).
  This takes away 1% to 5% of the remaining bandwidth.
- Packet overhead at the transaction layer. Your design choices have a great effect here:
  - Small packets can take away 75% of the bandwidth.
  - Large packets can take away as little as 1%.

STP 1B	SEQ 2B	TLP Header 12/16 B	Data Payload	LCRC 4B	END 1B
DLLP overhead		TLP		DLLP overhead	

Table 18-13. Effect of Packet Overhead on Link Throughput

Data Payload Bytes	Throughput after 8b10b encoding
128	67%
256	73%

18.13.1 Effect of Link Layer Flow Control and ACK/NAK DLLPs

DLLPs should be sent as often as required to avoid a negative impact on the TLP throughput. The controller sends a pending DLLP if there is no competing TLP traffic. Sending ACK/NAK and Update FC is controlled by timers. The controller provides flexibility to fine-tune these as required. The default setup is minimal flow control (one per time-out) and minimal ACK/NAK device latencies. The link partners retry buffer (payload) size might require a change in the ACK/NAK/Update FC time-out to obtain optimal system latencies. The controller provides a feature to accumulate up to 255 ACKs before issuing an ACK. This feature offers-fine tuning of ACK frequency impact. The core transmits ACKs at fixed intervals, by default.

DLLP package:

STP 1B	Type 1B	Data 4B	CRC 2B	END 1B

Effective throughput:

PACKET	*BYTES*	*CALCULATION*	*THROUGHPUT*
Worst Packet	One ACK plus one FC per 128 (8 packets of 16) bytes of data => 2 DLLPs per 8 data packets	(8*44) / (8*44 + 2*8)	95%
Typical Packet	One ACK plus one FC per 1.4 (256 byte payload) bytes of data => 2 DLLPs per 1.4 packets	1.4*280 / (1.4*280 + 2*8)	96%
Best Packet	One ACK plus one FC per 4096 bytes of data => 2 DLLP per data packet	4116 / (4116 + 2*8)	99%

18.14 Lane Reversal and Broken Lanes

Lane reversal refers to the situation when the lanes are reversed between the two chips at the ends of a PCI Express link. This can occur because of chip packaging, pinning, or board placement. The effects of lane reversal can be undone if at least one of the link partners has implemented the *optional* PCI Express lane reversal feature.

18.14.1 Link Width Negotiation and Lane Numbering

The process followed by the LTSSM in the downstream port (DSP), when negotiating with an x4 upstream port (USP) with no broken lanes, and no reversed lanes (A in Figure 18-9) is:

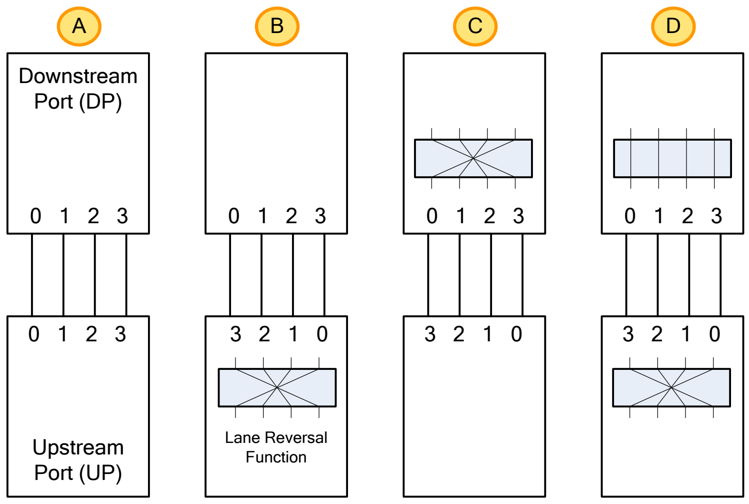
1. Link Width: (The controller supports link widths of x1, x2, x4 and x8.)
    - 1.1. DSP sends identical TS1 ordered sets on all detected lanes. Link number field is *n*, and lane number field is PAD.
    - 1.2. USP returns TS1 ordered sets with link number field for all lanes set to *n*.
    - 1.3. DSP deduces link width. In this example it is four.
2. Lane Numbering:
    - 2.1. DSP sends TS1 ordered sets on all lanes. Lane number fields are set from 0 to 3.
    - 2.2. USP returns TS1 ordered sets on all lanes, with the lane number fields derived from the physical lane number. In this case, that is from 0 to 3.
    - 2.3. DSP confirms lane numbering with TS2 ordered sets.

18.14.2 Reversed Lanes

The process followed by the LTSSM in the DSP, when negotiating with an x4 USP with no broken lanes, but with reversed lanes (B, C, and D in Figure 18-9) is:

1. Link Width: (same as above)
2. Lane Numbering:
    - 2.1. DSP sends TS1 ordered sets on all lanes. Lane number fields are set from 0 to 3.
    - 2.2. If USP supports line reversal (B and D in Figure 18-9):
        - 2.2.1. USP detects line reversal — by comparing for lane0, the lane number received in the TS1 against its physical lane number (that is, 0) — and activates its Lane Reversal Function.
        - 2.2.2. USP returns TS1 ordered sets on all lanes. Lane number fields are set from 0 to 3.
        - 2.2.3. DSP confirms lane numbering with TS2 ordered sets.
    - 2.3. Else, if USP does *not* support line reversal (C in Figure 18-9):
        - 2.3.1. USP returns TS1 ordered sets on all lanes. Lane number fields are bases on the physical lane number (as determined by its location in the USP data path). In this case, that is from 3 to 0.
        - 2.3.2. DSP detects line reversal — by comparing for lane0, the lane number received in the TS1 against the lane number it previously transmitted — and activates its Lane Reversal Function.
        - 2.3.3. DSP reattempts the standard lane numbering negotiation process, and succeeds this time.

Figure 18-9. Examples of lane reversal

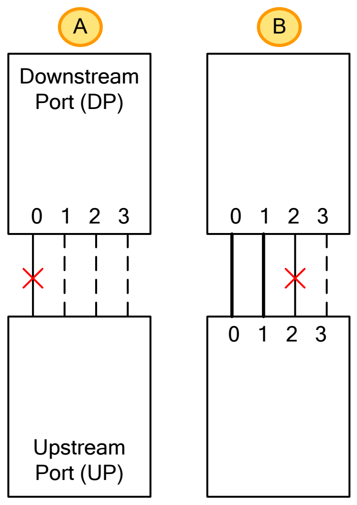


18.14.3 Broken Lanes

The process followed by the LTSSM in the DSP, when negotiating with an x4 USP with broken lanes is:

1. Link Width:
   1.1. DSP sends identical TS1 ordered sets on all detected lanes. Link number field is *n*, and lane number field is PAD.
   1.2. USP returns TS1 ordered sets with link number field for all lanes (except broken lanes) set to *n*.
   1.3. If for example, lane2 is broken (B in Figure 18-10), the DSP deduces that lane2 and upwards are unusable, and assumes a reduced link width of x2, using lane0 and lane1.
2. Lane Numbering:
   2.1. DSP sends TS1 ordered sets on lanes 0 and 1. Lane number fields are set from 0 to 1.
   2.2. If the lanes are not reversed, lane numbering will proceed as described in the previous section.

Figure 18-10. Examples of broken lanes



- *If a lane (not lane0) is broken*[3]*, and the lanes are* ***not*** *reversed, the controller can form a link.*
- *If lane0 is broken, and the lanes are* ***not*** *reversed, the controller* ***cannot*** *form a link. The PCIe LTSSM always assumes that lane0 is operational, and without lane0, it cannot determine the link width. In the PCIe controller, lane reversal always occurs after link width determination.*
- *If a lane (not the most significant lane) is broken, and the lanes are reversed, then the controller* ***cannot*** *automatically form a link. However, it is possible to form a link by manually un-reversing the lanes.*

Table 18-14. Link Formation Success Using Example of x4 Link

BROKEN LANE	LANES REVERSED ON BOARD	MANUAL LANE REVERSAL ACTIVATED	LINK FORMATION SUCCESSFUL?
None	No		Yes
None	Yes		Yes
Lane0	No	No	No
Lane1 or Lane2	No		Yes
Lane3	No		Yes
Lane0	Yes	No	No
Lane1 or Lane2			
Lane3			
Lane0	Yes	Yes	Yes
Lane1 or Lane2			Yes
Lane3			No

[3] Caused by any of (i) break on PCB (ii) electrical fault in transmitter (iii) electrical fault in receiver.

18.15 Loopback

The controller supports Local and Remote Loopback.

The controller supports Local PIPE Loopback without the need for a remote Link partner (Figure 18-11). It also supports Remote Loopback with a Remote Link Partner acting as the Loopback Slave, as shown in Figure 18-12. You can use this mode for board debug, BER testing, system debug, or in any other scenario where you have a remote Link partner.

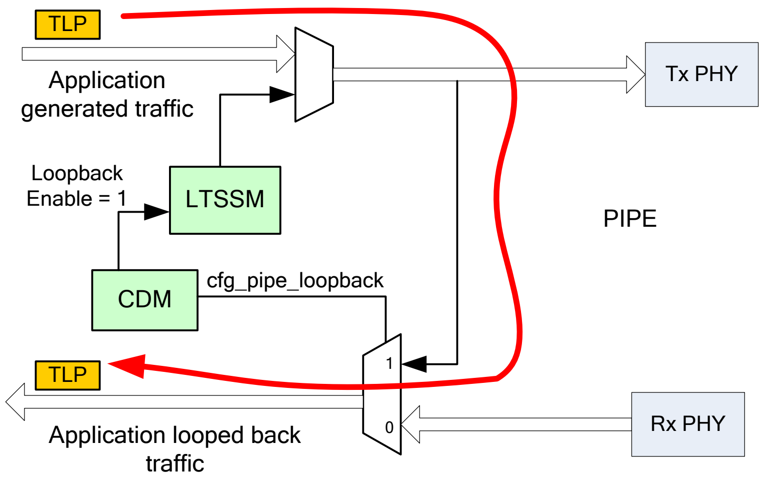
Loopback in the controller operates on a per Link basis only. You cannot operate it on a per Lane basis.

The Loopback implementation overrides LinkUp to one (it should be zero according to the specification), and allows the Link to initialize as if it were in L0.

18.15.1 Local PIPE Loopback

This section discusses the operation of the controller when entering, and exiting local PIPE Loopback. It also discusses the behavior of the controller when in local PIPE Loopback.

Figure 18-11. Components Involved In Local (PIPE) Loopback without a Link Partner



18.15.1.1 Entering Local PIPE Loopback

The PIPE loopback connects the PIPE RX signals to the PIPE TX signals, and only operates in the Loopback LTSSM state, as the link is not be able to train against itself to L0.

The procedure for entry into local loopback is:

- Set the Gen3 Equalization Disable bit in the Gen3 Control Register. You must do this, as the PHY has no role in PIPE loopback.
- Set the PIPE Loopback Enable bit in the PIPE Loopback Control Register.
- Set the Loopback Enable bit in the Port Link Control Register.

18.15.1.2 In Local PIPE Loopback

The Loopback mode allows the Link to initialize, as if it were in L0. After entering Loopback mode, the controller initializes flow control for VC0. When this completes, application can then send traffic (TLPs). As the traffic is looped back, the EZnet address the IMEM/EMEM and ‘completes’ its own requests.

When in Loopback mode, the Port initializes flow control against itself as every TLP and DLLP sent is also received. TLPs which are not received correctly as a result of Link errors are subject to replay as if the Link were in L0 state. Transmitted TLPs are subject to credit checks as if the Link were fully operational. Received TLPs are subject to error checking and filter checks as it the Link were operating in L0 state.

Message Considerations

When the Port is an Upstream Port (USP), you must not enable Error message generation. When an error occurs on the PCIe Link, and you have enabled Error message generation, the USP generates an Error message. Traffic generated by the Port is looped back to itself, and as an USP does not expect to receive messages, it generates an additional message, and so on.

A DSP must automatically send a Set_Slot_Power_Limit message when it enters L0, so the message is sent and is received by the Port itself. A DSP does not expect to receive a Set_Slot_Power_Limit message. Therefore, it is detected as an invalid message, and the Unsupported Request Detected bit in Device Status Register is be set.

When the Port is a Downstream Port (DSP), then internally generated messages are mixed together with traffic generation by application.

Enumeration and BAR Setup

You must configure the BARs (USP), memory/IO ranges (DSP), set the memory space enable and bus master enable, so that the receive filter accepts TLPs. Alternatively, you can turn off the filter rules —to allow TLPs that would normally be dropped— to be accepted.

In Loopback mode, you can initialize the BARs by writing from the local CPU through the DBI.

Gen3 Operation

You must set the Gen3 Equalization Disable bit in the Gen3 Control Register, as the PHY has no role in PIPE loopback.

18.15.1.3 Exiting Local PIPE Loopback

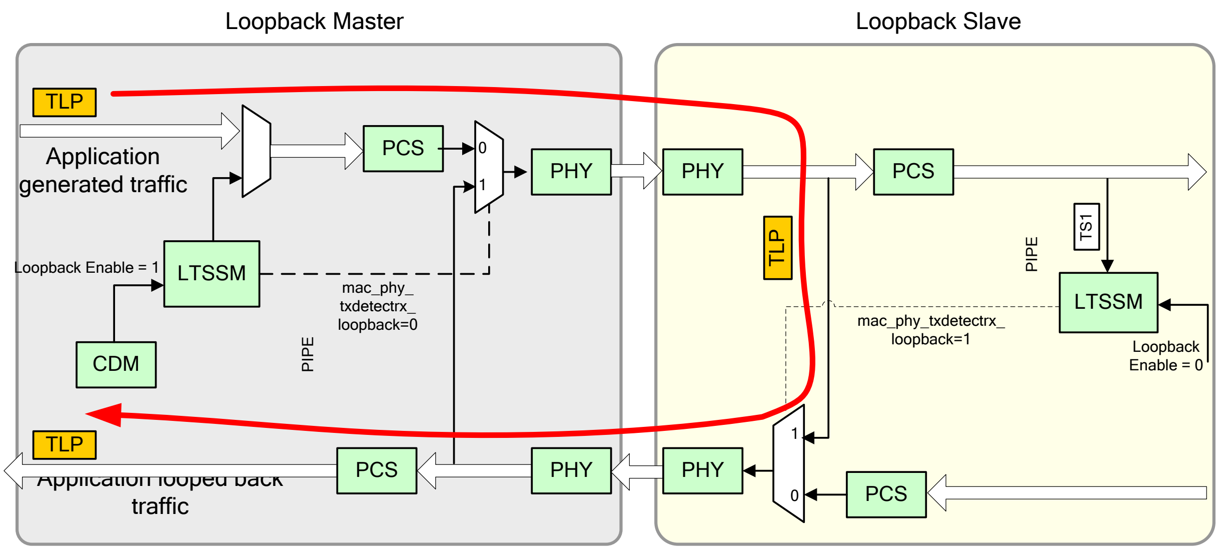
To exit Loopback mode:

- Clear the PIPE Loopback Enable bit in the PIPE Loopback Control Register.
- Clear the Loopback Enable bit in the Port Link Control Register

18.15.2 Remote Loopback

This section discusses the operation of the controller when entering, and exiting remote Loopback. It also discusses the behavior of the controller when in remote Loopback.

Figure 18-12. Components Involved in Remote Loopback with a Link Partner



18.15.2.1 Entering Remote Loopback

A Loopback Master is the component requesting Loopback and transmitting the data, for looping back by the Slave, as in Figure 18-12. A Loopback Slave is the component looping back the data. The behavior of a [Loopback] Master is not defined by the *PCI Express Base Specificatio*n. Only the protocol for negotiating Loopback in a Slave device is defined.

To cause the Master to enter Loopback mode, you must set bit 2 (Loopback Enable) in the Port Link Control Register. You do not have to do anything to enable Loopback mode in a Slave. It enters Loopback mode in response to the Master initiating Loopback mode.

Loopback uses bit 2 (Loopback) in the Training Control field which is sent within the TS1 and TS2 Ordered Sets (OSs). The Slave device enters Loopback whenever two consecutive TS1 OSs are received with the Loopback bit set. The Master device enters Loopback when it receives back from the Slave, the TS1 OSs that it sent. For more details see sections 4.2.5.10. *Loopback* and 4.2.6.10. *Loopback* of the *PCI Express Specification*.

18.15.2.2 In Remote Loopback

The Loopback Master mode allows the Link to initialize, as if it were in L0. After entering Loopback mode, the controller initializes flow control for VC0. When this completes, the application can then send traffic (TLPs). As the traffic is looped back by the Link partner, EZnet addresses the IMEM/EMEM and ‘completes’ its own requests. When in Loopback mode, the controller initializes flow control against itself as every TLP and DLLP sent is also received. TLPs which are not received correctly as a result of Link errors are subject to replay as if the Link were in L0 state. Transmitted TLPs are subject to credit checks as if the Link were fully operational. Received TLPs are subject to error checking and filter checks as it the Link were operating in L0 state.

Message Considerations

When the Loopback Master is an Upstream Port (USP), you must not enable Error message generation. When an error occurs on the PCIe Link, and you have enabled Error message generation, the USP generates an Error message. Traffic generated by the Master is looped back to itself by the Slave, and as an USP does not expect to receive messages, it generates an additional message, and so on. A DSP must automatically send a Set_Slot_Power_Limit message when it enters L0, so the message is sent and is received by the Master itself.

A DSP does not expect to receive a Set_Slot_Power_Limit message. Therefore, it is detected as an invalid message, and the Unsupported Request Detected bit in Device Status Register is be set. When the Loopback Master is a Downstream Port (DSP), then internally generated messages are mixed together with traffic generation by your application.

Enumeration and BAR Setup

SW must configure the BARs (USP), memory/IO ranges (DSP), set the memory space enable and bus master enable, so that the receive filter accepts TLPs. Alternatively, you can turn off the filter rules —to allow TLPs that would normally be dropped— to be accepted .

The RC Link Partner can perform Enumeration and BAR setup in L0, before you enable Loopback mode. Alternatively, in Loopback mode, you can initialize the BARs by writing from the CTOPs to configuration space. Any traffic generated by a Slave never reaches the PCIe Link. Therefore, if the RC is a Slave, it cannot do enumeration or BAR setup in Loopback mode.

Gen3 Consideration

According to section *4.2.3. Link Equalization Procedure for 8.0 GT/s Data Rate* of the *PCI Express Base Specification, Rev. 3.0, V. 0.9*, the Loopback Master is responsible for communicating the Transmitter and Receiver settings it wants the Slave to use. It does this through the:

- EQ TS1 Ordered Sets it transmits in the 2.5 GT/s or 5.0 GT/s data rate
- Preset or Coefficient TS1 Ordered Sets it transmits in the 8.0 GT/s data rate.

The implementation of Loopback does not support the changing of Preset and Coefficient values in the LOOPBACK.ENTRY LTSSM state.

18.15.2.3 Exiting Remote Loopback

To cause the [Loopback] Master to exit Loopback mode, you must clear bit 2 (Loopback Enable) in the Port Link Control Register. You do not have to explicitly direct the Slave to exit Loopback mode. It exits Loopback when it detects four consecutive EIOSs, which the Master transmits after exiting Loopback mode. If the current Link speed is 2.5 GT/s, and an EIOS is received, or Electrical Idle is detected or inferred on any Lane, then the Slave also exits Loopback mode.

18.16 Lane Deskew

Controller configurations with more than one Lane, remove Lane-to-Lane skew using de-skew buffers. The *PCI Express Specification* Lane-to-Lane de-skew requirements are given in Table 18-15. The *PCI Express Base Specification* de-skew requirement is comprised of a dynamic component (variation in the length of the SKP Ordered Sets received across all Lanes on a Port) expressed in (SKP) symbols, and a static component (differences in the interconnect delay) expressed in ns.

The total de-skew requirement for the PCIe controller is obtained adding, if necessary, one or more extra symbols to the dynamic component of the *PCI Express Base Specification* de-skew requirement. These extra symbols are needed to take into account variations across the Lanes in the SKP symbols added/removed by the PHY, if these variations exceed what is allowed by the spec.

Table 18-15. PCIe Specification Requirements for Lane-to-Lane De-Skew ($L_{RX-SKEW}$)

SKEW	*GEN1 (2.5 GT/S)*	*GEN2 (5.0 GT/S)*	*GEN3 (8.0 GT/S)*
PCI Express Base Specification (ns)	20 ns	8 ns	6 ns
PCI Express Base Specification (symbols) [Static + Dynamic]	5 [1+4] symbols	4 [2+2] symbols	6 [4+2] symbols
Maximum skew (symbols) the controller can de-skew	16 symbols	16 symbols	24 symbols

18.17 PCIe Ordering Rules

The PCI Express ordering rules are designed to satisfy the following three requirements:

1. Requirement: Producer-Consumer Model

   In this model, the Consumer must not start reading a data buffer before a Producer is finished writing it. To prevent violation of the model, it is necessary to ensure that no transaction passes a previously issued Posted (P) transaction. This is covered by rules 2, 5, and 9 in Table 18-16. Summarized PCIe Ordering Rules. The following performance optimizations are added:

   - Producer-Consumer Relaxed Ordering (RO) Optimization
     Completions and Writes that are not involved in Producer-Consumer sequences are permitted to pass a previously issued Posted (P) transaction. This is achieved by setting their RO bit to 1.
   - Producer-Consumer ID-Based Ordering (IDO) Optimization
     A CPL or request that originates from a different completer or requestor, and that has its IDO bit set to 1, is permitted to pass a blocked P. The controller supports, but does not implement IDO.

   Strong Ordering (routing all TLPs in-order without reordering) satisfies the Producer-Consumer model. The next two requirements seek to modify Strong Ordering.

2. Requirement: Deadlock Avoidance

   Avoid blocking of CPL and P TLPs, by NP requests that are stalled. This is covered by rules 1 and 4 in Table 18-16. Summarized PCIe Ordering Rules.

3. Requirement: Performance Optimizations

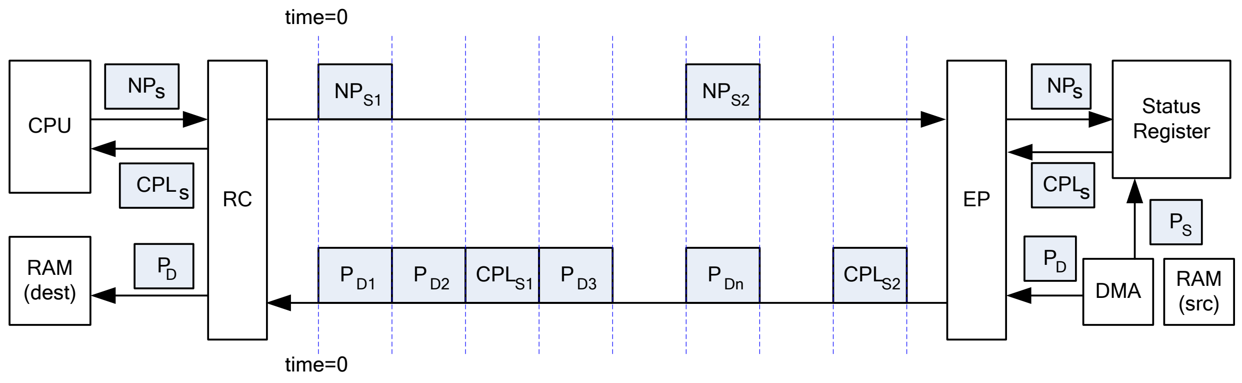
   TLPs that are not bound by any of the previous requirements can pass or be blocked by each other. This is covered by rules 3, 6, 7, and 8 in Table 18-16. Summarized PCIe Ordering Rules.

18.17.1 Producer-Consumer Example

Producer: The DMA in the EP is taking data from the local source RAM and writing it into the destination RAM in the RC. This posted data is indicated as P_D.

Consumer: The CPU in the RC is polling the Status register in the EP, checking if the DMA is finished producing the data. These polling Read requests are indicated as NP_S. The corresponding completion is indicated as CPL_S.

Producer-Consumer Model: The CPL_S must not pass any P_D, or else the CPU starts to consume the data before it is fully produced. Therefore the EP, any intermediate SW, and RC queues must obey this rule. The rule does not need to be applied to P TLPs and CPLs that are not part of this Producer-Consumer sequence, for example, RC CPU CFG reads to unrelated functions in the EP.



18.17.2 PCIe Ordering Rules Table

Ordering rules only apply for traffic within a single Traffic Class (TC). For a proper understanding of Ordering you should be familiar with *Table 2-34* in section 2.4.1, *Transaction Ordering Rules* of the *PCI Express Base Specification, Revision 3.0*. This list summarizes the PCI Express ordering rules.

Table 18-16. Summarized PCIe Ordering Rules

NO.	*DESCRIPTION*	*SYSTEM REQUIREMENT UNDERLYING THIS RULE*
1	P must be allowed to pass a previously-issued NP.	Deadlock Avoidance
2	**P** must not pass a previously-issued P, unless: ▪ the later P's RO bit is 1 or ▪ the later P's IDO bit is 1, and the Requestor IDs (RIDs) are different.	Producer-Consumer Model
3	**P** can optionally pass* a previously-issued CPL.	Performance Optimization
4	**CPL** must be allowed to pass a previously-issued NP.	Deadlock Avoidance
5	**CPL** must not pass a previously-issued P, unless: ▪ CPL's RO bit is 1 or ▪ CPL's IDO bit is 1, and the RIDs are different.	Producer-Consumer Model
6	**CPL** can optionally pass* a previously-issued CPL, when the RIDs are different.	Performance Optimization
7	**NP** can optionally pass* a previously-issued NP.	Performance Optimization
8	**NP** can optionally pass* a previously-issued CPL.	Performance Optimization
9	**NP** must not pass a previously-issued P, unless: ▪ NP's IDO bit is 1, and the RIDs are different, or (*in the case of NP Writes only)* ▪ NP's RO bit is 1.	Producer-Consumer Model

* or be blocked by

18.17.3 Inbound (Receive) Order Enforcement

The Inbound order scheme fully obeys the producer-consumer model and deadlock is avoided. All of the PCI Express ordering rules in *Table 2-34* in section 2.4.1, *Transaction Ordering Rules* of the *PCI Express Base Specification, Revision 3.0* are obeyed.

TLPs are filtered into three separate FIFO queues: posted, non-posted, and completion. The ordering controller determines which of the three queues have a TLP that is allowed to proceed. When more than one queue has a TLP that is allowed to proceed, it uses a priority arbitration algorithm to select a transaction. This prevents starvation, which occurs when delivery rate for a particular TLP type to application is unfairly reduced.

A completion or a posted transaction with its RO header bit set to "1", can pass a previously issued posted transaction that is blocked. RO is not implemented for non-posted writes.

The controller does not support ordering between channels. The controller routes all posted and non-posted transactions into the request FIFO, and all completions into the response buffer. Posted and non-posted TLPs do not pass each other inside the request FIFO.

To prevent violation of rule (5) – **Completion Must Not Pass Posted Write** – the dequeue controller checks (per VC) that there are no outstanding posted transactions of request FIFO before it takes a completion from the receive queue. If necessary it will halt the completion queue and starts a watchdog timer. If the timer expires (because there are still outstanding posted transactions) then the order manager does the following:

- Alert software that it is now violating rule (5) to avoid any potential deadlocks caused by the non-progress of completions.
- Disables rule (5) temporarily by unhalting all completion queues until they are empty.

Figure 18-13. Inbound PCIe Ordering

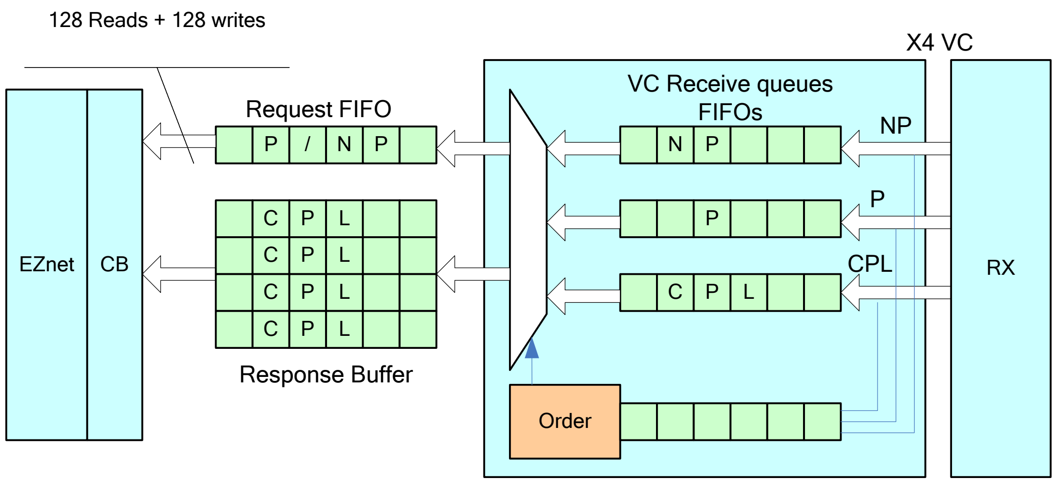


Table 18-17. Inbound Ordering Adherence to PCIe Ordering Rules

The controller does not implement ID-based ordering.

NO.	DESCRIPTION	SYSTEM REQUIREMENT UNDERLYING THIS RULE	OBEYED?	REASON
1	P must be allowed to pass a previously-issued NP.	Deadlock Avoidance	No	Although the bridge has a common P/NP FIFO, the NP will not block the P
2	**P** must not pass a previously-issued P, unless: ▪ the later P's RO bit is 1 or ▪ the later P's IDO bit is 1, and the Requestor IDs (RIDs) are different.	Producer-Consumer Model	Yes	Common P/NP FIFO. Controller does not consider the RO and IDO bits
3	**P** can optionally pass or be blocked by a previously-issued CPL.	Performance Optimization	Yes	Optional passing is supported
4	**CPL** must be allowed to pass a previously-issued NP.	Deadlock Avoidance	Yes	No ordering enforced between CPLs and NP. Separate queues used.
5	**CPL** must not pass a previously-issued P, unless: ▪ CPL's RO bit is 1 or ▪ CPL's IDO bit is 1, and the RIDs are different.	Producer-Consumer Model	Yes	Ordering enforced between CPLs and P.
6	**CPL** can optionally pass a previously-issued CPL, when the RIDs are different.	Performance Optimization	No	Single CPL queue. Optional passing not supported.
7	**NP** can optionally pass a previously-issued NP.	Performance Optimization	No	Single NP queue.
8	**NP** can optionally pass a previously-issued CPL.	Performance Optimization	Yes	Optional passing is supported.
9	**NP** must not pass a previously-issued P, unless: ▪ NP's IDO bit is 1, and the RIDs are different, or (*in the case of NP Writes only)* ▪ NP's RO bit is 1.	Producer-Consumer Model	Yes	Common P/NP FIFO in bridge. Configuration can modify this rule.

18.17.4 Outbound (Transmit) Order Enforcement

The request arbitration priority (highest first) for transmission of TLPs at the internal transmit interface between the AHB/AXI Bridge and the Native controller is:

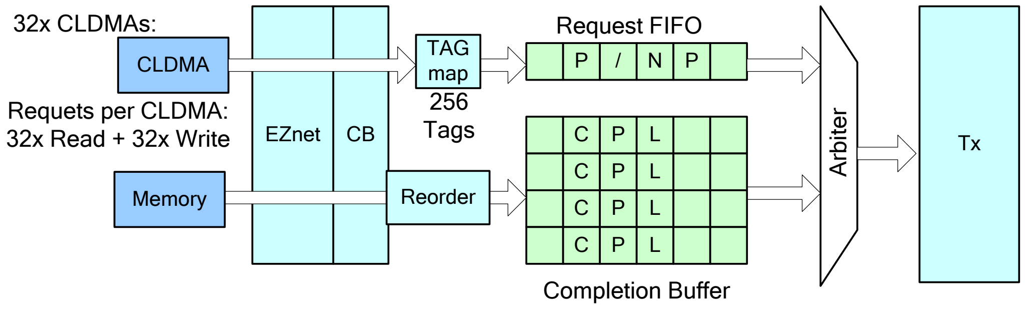
1. Messages
2. Internally-generated CPLs for received CFG requests
3. Transmit TLPs from EZnet (Requests and Completions).

Any NP or CPL TLPs on EZnet (behind the blocked P) are blocked by the halted posted queue.

EZnet routes all outbound posted and non-posted requests into the request FIFO, and completions into the completion buffer. It is possible for posted to be stuck behind a non-posted.

The controller enables producer-consumer ordering rules by delaying the generation of the posted write response until the posted write has been passed to the transmit layer.

Figure 18-14. Outbound PCIe Ordering



Each CLDMA request is uniquely mapped to one of 256 PCIe Tags.

A CLDMA outbound Read request could be sometimes decomposed into multiple PCIe TLP requests (these are called sub-TLPs), each with a different PCIe Tag. The PCIe protocol allows CPLs associated with the different TLPs to pass each other and be returned out-of-order. The PCIe controller Response Composer does not re-order the CPLs, resulting in data corruption. To avoid data corruption, a workaround is required:

- Prevent Outbound decomposition of Read requests, by ensuring that CLDMA does not generate bursts of size greater than Max_Read_Request_Size (<=128B MTU).

A remote device which completes a single request using multiple CPLs (CplD) always returns them in-order. The PCIe controller can accept an interleaved stream of multiple CPLs (in response to multiple requests it originally transmitted), provided that all the CPLs for any one original request are in-order.

The Following Rules are Violated

The P/NPs from the EZnet are queued in a common FIFO which causes a violation of "P must be allowed to pass NP".

The CPLs are in an independent stream within the Bridge, and are sent to the native controller without any ordering considerations to the P/NP stream, which causes a violation of "CPL must not pass P unless RO bit of CPL is 1".

- "P must be allowed to pass NP" -- This is not normally a serious violation because the FIFO will not be indefinitely backpressured, so blocking should not occur.
- "CPL must not pass P unless RO bit of CPL is 1" -- Supports relaxed ordering (effectively having RO=1 for all CPLs) by desensitizing your system to this Producer-Consumer violation.

Table 18-18. Outbound Ordering Adherence to PCIe Ordering Rules

NO.	*DESCRIPTION*	*SYSTEM REQUIREMENT UNDERLYING THIS RULE*	*OBEYED?*	*REASON*
1	**P** must be allowed to pass a previously-issued NP.	Deadlock Avoidance	No	Common P/NP FIFO causing the blocking of P if core does not have enough NP credits
2	**P** must not pass a previously-issued P, unless: ▪ the later P's RO bit is 1 or ▪ the later P's IDO bit is 1, and the Requestor IDs (RIDs) are different.	Producer-Consumer Model	Yes	Common P/NP FIFO. Controller does not consider the RO and IDO bits.
3	**P** can optionally pass (or be blocked by) a previously-issued CPL.	Performance Optimization	Yes	Optional passing supported
4	**CPL** must be allowed to pass a previously-issued NP.	Deadlock Avoidance	Yes	No ordering enforced between CPLs and P/NP. Separate queues used.
5	**CPL** must not pass a previously-issued P, unless: ▪ CPL's RO bit is 1 or ▪ CPL's IDO bit is 1, and the RIDs are different.	Producer-Consumer Model	No	No ordering enforced between CPLs and P. Not to limit performance.
6	**CPL** can optionally pass a previously-issued CPL, when the RIDs are different.	Performance Optimization	No	Optional passing not supported. Single CPL queue.
7	**NP** can optionally pass a previously-issued NP.	Performance Optimization	No	Optional passing not supported. Single NP queue.
8	**NP** can optionally pass a previously-issued CPL.	Performance Optimization	Yes	Optional passing supported
9	**NP** must not pass a previously-issued P, unless: ▪ NP's IDO bit is 1, and the RIDs are different, or (*in the case of NP Writes only)* ▪ NP's RO bit is 1.	Producer-Consumer Model	Yes	Common bridge P/NP FIFO.

18.18 Interrupts

The controller supports the MSI-X.

An MSI-X interrupt is identical to an MSI, except that an Endpoint may use one of up to 2048 address and data pairs in the MSI-X P Write packet. Endpoints with MSI-X capability also include logic to mask and hold pending interrupts, as well as a memory table for the address and data pairs. The large number of address values available to each Endpoint allows MSI-X Messages to be routed to different interrupt consumers in a system, as compared to the single address available to MSI packets. Upstream Switch Ports can send MSI-X packets; Root Ports and downstream Switch Ports cannot. In complex systems, MSI-X packets could be routed to devices other than the RC, including other Endpoints, based on the multiple address/data pairs available.

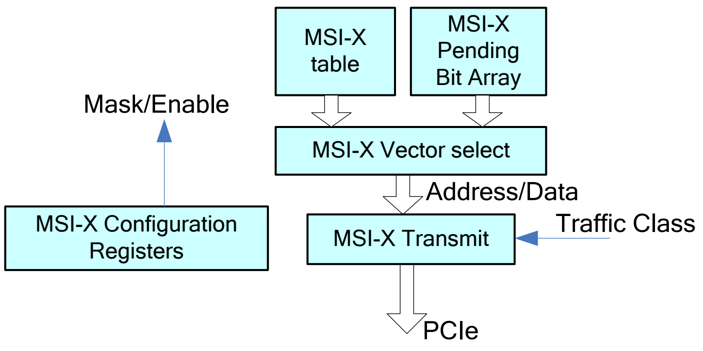
Message Signaled Interrupts (MSI-X) are Memory Write (MWr) TLPs. They are indistinguishable from normal MWr's apart from the target address used in conjunction with the MSI-X Capability Structure.

The address and data fields for MSI-X packets are defined in an MSI-X table located in the Endpoint's application logic, as shown in Figure 18-15. Each entry in the table corresponds to an MSI-X packet that your application may send. Each MSI-X table entry also includes a mask bit for that interrupt. The pending bit array (PBA), includes a pending bit for each MSI-X table entry.

Your application must configure the values for MSI-X Table and PBA Offset and BIR registers. System software loads the MSI-X address and data values into the MSI-X Table and PBA structures in system memory using MWr Requests.

Host software sets and clears mask bits. When an interrupt is masked, the controller does not request to send the interrupt. The controller re-issues the interrupt when the mask is cleared by the host software.

Figure 18-15. MSI-X Message Passing



18.18.1 MSI-X Overview in SR-IOV

The MSI-X Capability structures are optional. You can only include them on a per-device basis. You cannot include them on a per-PF basis. All VFs in a particular PF must have identical MSI-X settings, either enabled or disabled. MSI-X is not enabled on a per-VF basis.

Unique Per-VF Values

All VF MSI-X registers, with the exception of the VF MSI-X Control register, have identical values, and are Read-Only. The *MSI-X Table Size* field of the VF MSI-X Control register must have the same value for all VFs within a PF. To write to the *MSI-X Table Size* field of the common VF MSI-X Control register.

19. Congestion Management and Flow Control

This chapter describes NPS device and system resources congestion management and traffic flow control mechanisms.

The NPS integrates functionality which enables the implementation of system-wide QoS and distributed traffic management. NPS contains hardware mechanisms to enable rapid, flexible and accurate adaptation of the system to dynamically changing traffic conditions. The NPS allows flow control congestion management on physical links (Link-Level FC), link level per Priority Flow Control (PFC), and end-to-end system-level flow control. The congestion reporting and flow control are supported via out-of-band signaling as well as in-band and SW.

The QoS aware congestion management key elements are:

- QoS aware resources congestion monitoring.
- In-band and OOB congestion status messaging.
- Traffic QoS aware flow control: throttling or rate limiting.

19.1 Overview

NPS maintains comprehensive hardware and software support for congestion management and flow control (FC):

- Various internal congestion monitoring mechanisms:
  - Received traffic packet data memory buffer congestion accounting for IMEM and EMEM.
  - Job descriptors congestion accounting.
  - Traffic Manager frame data memory buffering congestion accounting for IMEM and EMEM. The congestion can be measured per TM hierarchy level or flexible SW-assigned criteria.
- Various congestion status signaling mechanisms:
  - In-band congestion signaling:
    - Link Level FC generation
    - Link Level per Priority flow control (PFC) generation
    - Class and Channel based flow control (CBFC) generation
    - Per channel per priority flow control generation
  - Out-of-band congestion signaling:
    - Link Level FC generation / Link Level PFC generation
    - Channel Level FC generation
    - Per channel per priority flow control generation
    - Interlaken Status Interface signaling
- Various traffic flow control mechanisms:
  - In-band and out-of-band Link Level FC obey
  - In-band and out-of-band Link Level per Priority FC obey
  - In-band and out-of-band Channel Level FC obey
  - Out-of-band Flow Control per TM hierarchical level
  - Out-of-band Flow Control per Channel per CoS
  - Out-of-band Flow Control per Port per CoS
  - SW rate control of TM’s hierarchical levels

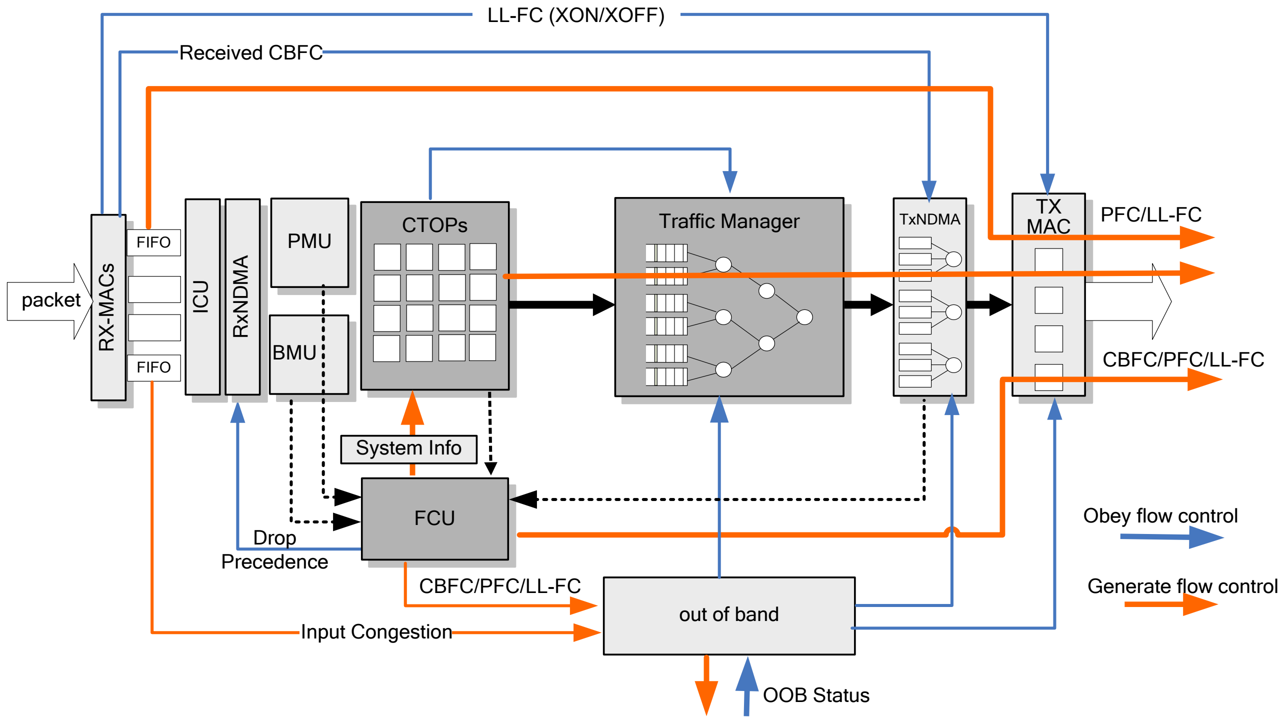
19.2 Flow Control Generation and Obey

The NPS flow control message handling is split into two main categories:

- **Obey flow control messages/events –** in-band FC messages, or OOB signaling from external devices, allow a congested external device to signal the NPS to stall sending packets of a certain type to a certain destination. SW control allows transmit traffic throttling and rate adjustment.
- **Generate flow control signaling and messages –** initiated by the NPS and delivered to an external device. These allow the NPS to request that external devices stall sending packets of a specific type in order to handle, or prevent congestion, of the NPS's resources.

The hierarchical flow control design spans from the fast responding MAC hardware support to the precise per TM queue software control for both incoming and outgoing traffic. The NPS supports in-band, out-of-band and SW-based flow control signaling, where all its features are configurable or programmable, yielding high flexibility and meeting with user's system requirements. Grouping mechanisms and class of service differentiation allow for elaborate oversubscription management schemes.

Figure 19-1. Main flow control stages and paths



The flow control architecture is supported by functionality in several NPS architectural modules:

- Receive MAC – Intercepts Link-Level and Class-Based flow control packets for Tx MAC FC.
- Receive FIFO – Based on FIFO congestion, the PFC and CBFC can be generated.
- ICU – Packet classification to Class and Priority.
- Receive Network DMA (RxNDMA) – Packet drop per Drop Precedence status from FCU.
- PMU – Jobs queue and per class job accounting.
- BMU – Job descriptor memory buffer and packet memory buffers resource management.
- FCU – Job descriptors, internal memory and external memory resource pools management.
- TM – Flow control and rate adjustment per hierarchical entity.
- OOB Interface – Serial interface for periodic congestion status and flow control exchange.
- Transmit Network DMA (TxNDMA) – Per class and per priority transmit throttling.
- Transmit MAC – Class Based and Priority flow control generation and LL traffic pause.

- CTOPs **–** System wide congestion monitoring and resource budget reassignments and traffic engineering. The local congestion is gathered via System Info structure updates by FCU.

19.2.1 Methods of Flow Control Messaging

There are several methods supported by NPS for the purpose of delivering flow control messages to report congestion status and allow external devices to flow control the NPS transmit scheduling.

- **In band** – Flow control can be delivered in-band employing standard Ethernet packets and an Interlaken control header. This signaling supports:
  - Ethernet Link Level flow control Pause packet
  - Ethernet per priority flow control or class-based flow control
  - Per channel FC for Interlaken interfaces. Each Interlaken interface supports per channel incoming flow control for up to 128 channels and flow control generation for up to 64 channels.
  - Interlaken link flow control.
- **Out of band** – In this method, flow control status is sent and received through the Interlaken Status Interface or OIF SPI 4.2 Status interface. The out-of-band controller maintains a constant incoming and outgoing bitstream and a calendar scrolling over all of the configured status types. The out-of-band controller continually updates all the Obey Flow Control Status bit fields mapped to it. It also polls all the Congestion Status Registers and transmits their status.
- **SW control** – The management CPU and CTOPs application SW can monitor and control the flow control status of the stages in NPS through software. The NPS internal congestion is available to SW through the IMEM based System Info structure. The SW can adjust TM scheduling rate and flow control TM hierarchical entities.

19.2.2 Transmit Traffic Scheduling Hierarchy Flow Control

Flow control is accomplished through multiple hierarchies in NPS. Low level hierarchies achieve a faster response time, whereas higher levels attain better resolution and precision.

- **Tx MAC** – The MAC is the lowest level hierarchy in the flow control scheme and includes hardware support for generation and acquisition of link level flow control. The transmit MAC can be per priority flow controlled via in-band and out-of-band signaling.
- **Interlaken** – The TxNDMA can be flow-controlled per 128 Interlaken channels on each side. The XON/XOFF is on an Interlaken burst boundary. Per channel flow control is on packet or burst boundary.
- **TM Output Queue** – The output DMA schedules packets from multiple channel queues; each channel maintains a queue per class of service. Several channels can lead to a single MAC interface through a joint class of service scheduler. This topology allows flow controlling on the class of service level, on the channel level on a packet boundary. The flow control is per channel per priority.
- **TM** – The TM’s queues are aggregated into five hierarchical levels. Flow control signals can be generated or acquired in each hierarchy of the Traffic Manager.

19.2.3 Per Priority Flow Control

The NPS supports the generation and acquisition of per class of service flow control packets on Ethernet interfaces according to the 802.1Qbb standard. The per class of service flow control is also supported on Interlaken interfaces.

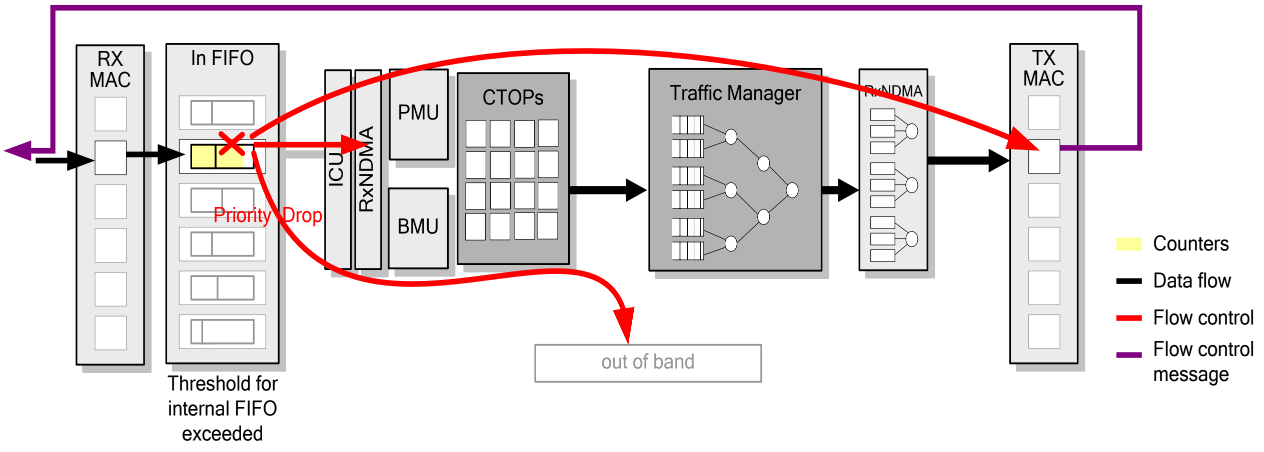
The Ethernet Priority Flow Control (PFC) pause packet pauses all traffic of the appropriate level and inferior levels. The scheduling FC is supported for up to four priorities. The incoming PFC packets throttle the packet scheduling of the transmit MAC and of the TxNDMA output queue.

The receive interface input FIFO and BMU/PMU resource usage are monitored by the FCU for PFC generation. For the purpose of congestion monitoring, each internal threshold is split into four configurable class of service related thresholds. When exceeded, a flow control message of a configurable level (out of eight possible classes of service) is generated. This mechanism is another method for gradually decreasing the incoming traffic, where lower priority classes of service are throttled first.

19.2.4 MAC Link Level Input Buffer (FIFO) Congestion

Each input interface of the NPS has a user configured threshold for link-level FC message generation. When its MAC layer input FIFO's occupancy exceeds this threshold, a link level message is initiated and sent through its coupled output interface (figure below).

Figure 19-2. Link level flow control

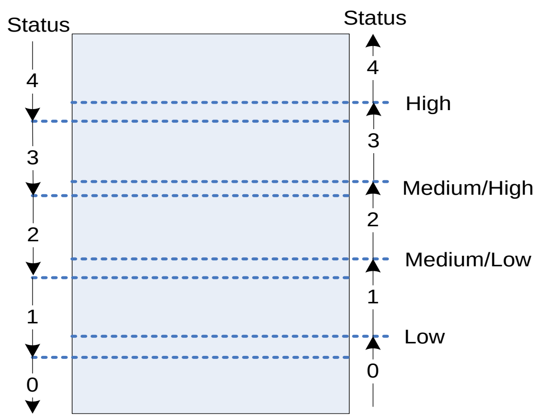


The MAC layer input FIFOs buffering space available per side is 512KB. This memory is allocated to interfaces. There are 48 input FIFOs per side. Each 10GbE interface has dedicated input FIFO.

- 40GbE interfaces are mapped to FIFO number N*4.
- 100GbE interfaces are mapped to FIFO number N*12.
- Interlaken interfaces are mapped to FIFO number N*24.

Each input FIFO is monitored for congestion by set of configurable thresholds (N*16B).

The set of congestion thresholds is comprised of four dual thresholds. The dual thresholds allow hysteresis on status change. The FIFO fullness is compared to a set of thresholds which results in congestion status 0-4. Each congestion status is assigned an 8b PFC indication.



When an input FIFO overflows, the received packet is dropped. The packet can be dropped by indicating a packet Drop Precedence to the RxNDMA, or dropped when there is no place in the FIFO to write the SOP. The input FIFO has a configurable discard threshold.

When an Interlaken input FIFO overflows, packets of all channels being re-assembled are marked with a receive error.

The interface input FIFO congestion is also signaled via the OOB Status interface.

19.2.4.1 Input FIFO based Priority Flow Control

The congestion indication is used to generate a PFC Ethernet frame on Ethernet interfaces.

The 8b PFC indication is combined with FCU congestion to generate a PFC frame on each status change as well as periodic flow control when 8b PFC is non zero.

Each Ethernet port can be enabled to generate PFC or Link-Level FC. Alternatively, these may be disabled.

19.2.4.2 Input FIFO based Link Level FC Generation

The 8b PFC congestion indication is also used for LL flow control generation.

- 0x00 – XON
- 0xFF – XOFF

The congestion status may trigger Link Level FC generation on either an Ethernet or Interlaken interface. The LL FC generation can be disabled per interface port. The Ethernet Pause packet generated sets the FC Time to maximal value (XOFF) when the congestion counter crosses the hysteresis high threshold, and sets FC Time value to 0 (XON) when the congestion counter crosses the low threshold. While in the XOFF state, the LL FC is generated periodically per configurable period.

The typical latency from high threshold crossing until XOFF generation is 250ns + transmitted packet link time.

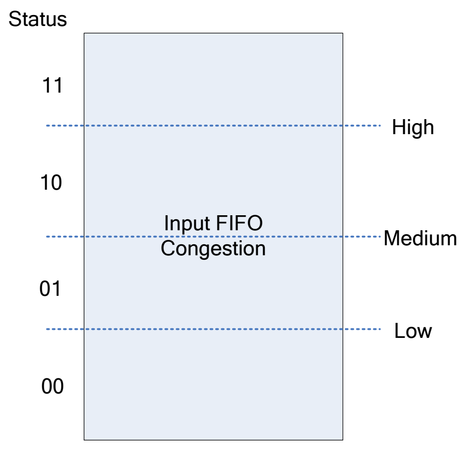
The typical latency from low threshold crossing until XON generation is 184ns.

19.2.4.3 Input FIFO based Drop Precedence

The interface group input FIFO congestion is monitored for priority drop generation. The congestion counter is compared to a set of 3 thresholds (N*16B): High, Medium and Low.

The comparison yields a 4-level congestion status. This congestion status is combined with FCU congestion status for drop precedence indication.

Figure 19-3. Four-level congestion status for drop precedence



19.3 Resources Usage Accounting (FCU)

The Flow Control Unit (FCU) is responsible for managing the resource usage counters for three budgets:

- Frame data memory buffers in IMEM.
- Frame data memory buffers in EMEM.
- Job descriptor memory buffers in IMEM.

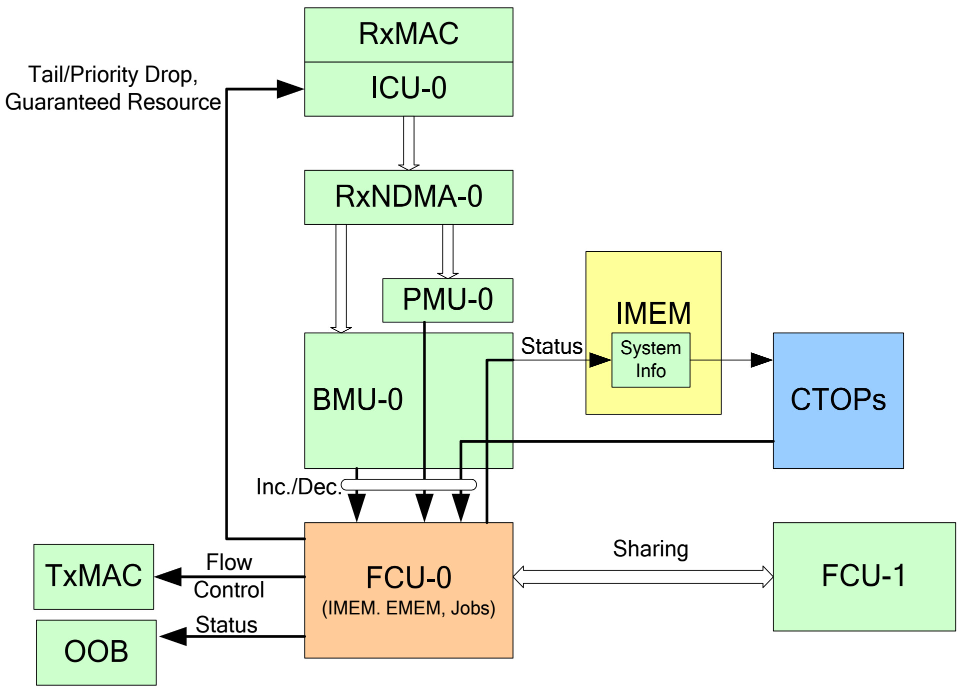
The FCU receives resource usage updates (increments/decrements) from BMU for frame memory buffers and from PMU (through BMU) for Job descriptors. The application SW also performs resource budget updates.

The resource management is organized in a hierarchical manner that allows various schemes of resource sharing. The hierarchy is composed of Budget ID, Logical Interface, Group and Global levels.

The FCU reports the resource usage status to several targets:

- ICU is informed to perform Tail packet drop and Drop Precedence based packet drop. The ICU is also informed when resources are consumed from the Guaranteed budget.
- Transmit MAC is signaled to transmit Link Level Flow Control, Priority Flow Control and Class Based Flow Control and signal per channel congestion on channelized interfaces.
- The resource congestion status is periodically signaled via the Out-Of-Band interface that is controlled by a configurable calendar.
- The resource congestion status is periodically updated in the System Info memory structure.

Each FCU works with units on its own NPS side. The FCU units are interconnected for updates of common resource sharing.

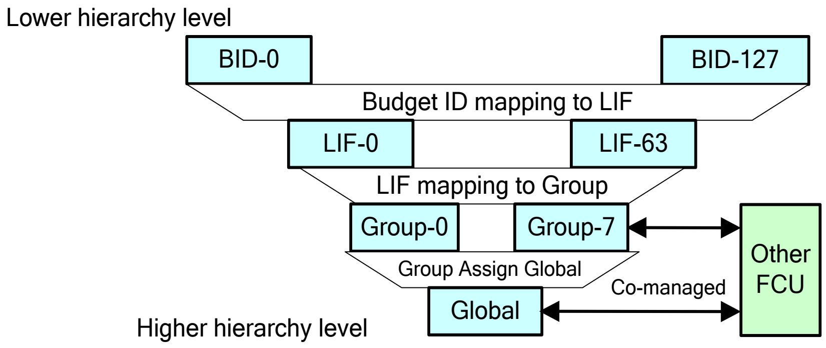


The two FCUs interact for co-managing shared pools resources. The inter FCU sharing can be enabled for each Group entity, each Global per Class entity, or Global entity. The sharing allows unified management of resources.

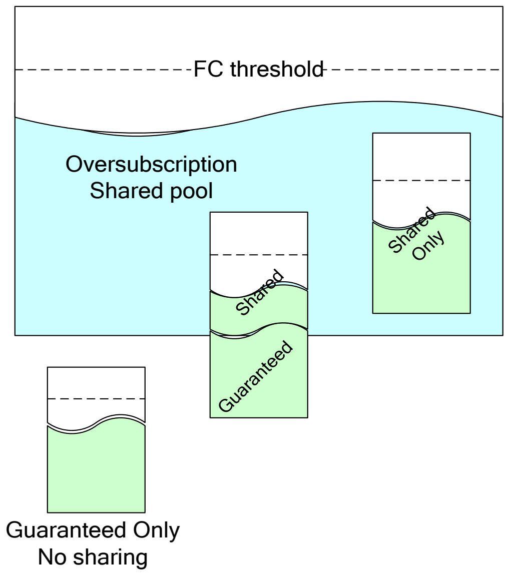
19.3.1 Resource Accounting Hierarchy

The resource management supports hierarchical accounting for IMEM, EMEM and Jobs. The hierarchy is comprised of four levels.

1. Budget ID (channel) – The initial handle for resource budget identification. Budget ID (BID) is mapped to Logical Interface. The Budget ID supports 128 entities per FCU unit.
2. Logical Interface – The aggregation of BIDs to Logical Interfaces for resources sharing. The Logical Interface supports 64 entities per FCU unit.
3. Group – The aggregation of Logical Interfaces to Groups for resource sharing. There are 8 available groups per FCU unit. Each group resources can be co-managed between FCUs.
4. Global – The aggregation of Groups. The Global can be co-managed between FCUs.



The hierarchical mapping allows various schemes of resource sharing. Some resources can be guaranteed (dedicated) to a member while some can be shared amongst members. The resource sharing, allows signaling of flow control, that is based on guaranteed and shared resources congestion status. The resource sharing can be defined between (low to high) hierarchy levels: BID to LIF, LIF to Group, Group to Global.



The BID (initial handle) is assigned per receive physical interface and Class of Service (by the ICU/RxNDMA) and per internal interface (Loopback and timers). The initial BID assignment is shared by IMEM, EMEM and Job budget pools. The application SW can later reassigned the budgets per packet processed.

19.3.2 Hierarchy Usage for IMEM/EMEM/Job Pools

The hierarchical entities for IMEM/EMEM packet data buffers are mapped to network physical interfaces.

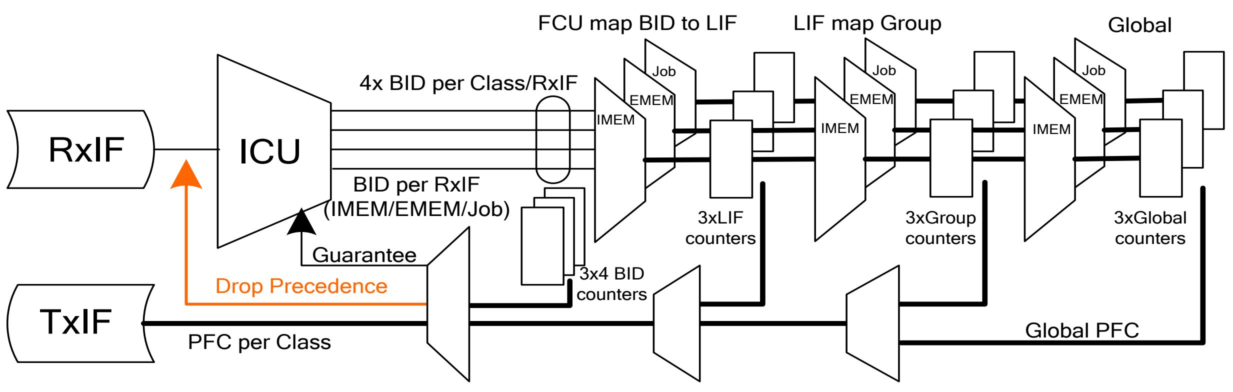
The mapping is different for packets' receive path and transmit path. The receive path maps network physical port and channel per COS to a Budget ID entity. The transmit port maps Budget ID hierarchy entity status to channel/class flow control and Logical Interface entity status to physical port flow control.

- Receive port/channel per class is mapped to Budget ID entity in IMEM/EMEM/Job pools.
  - Interlaken channel is mapped to BID
  - Ethernet 40GbE Rx port supports up to 4 Classes (ICU assigned per application)
  - Ethernet 10GbE Rx port supports up to 2 Classes (ICU assigned per application)
- Budget ID status is mapped to in-band Priority flow control generation on transmit channel/class.
- Budget ID entity is mapped to Logical IF (same LIF# for IMEM/EMEM/Job pools).
- Logical IF status is mapped to in-band flow control on transmit port.
- LIF is mapped to Group. Group status is used to generate PFC for group members.
- Group is mapped to Global status.

The channels are used for channelized links (i.e. Interlaken). The classes are used when the interface supports Class Based flow control, up to four classes per Ethernet port.

The hierarchical congestion monitoring is used also for Tail drop and Drop Precedence signaling to the Rx path.

The Logical Interface is mapped directly to transmit port: LIF 0-47 mapped to Ports 0-47. The LIF 48-63 are user-defined interfaces.



19.3.2.1 ICU Class of Service (Priority) vs. Class Assignment

The ICU assigns both CoS and Class. The assigned CoS is a strict priority that is assigned per packet encapsulation parsing for Priority/EXP/DSCP and various L2/L3/L4 control protocol identifications.

The Class is based on the source port and L2 packet classification to distinguish L2 channels, i.e. VLANs, tunnels, etc. The Class is used for FCU resource assignment and management. The Class assignment can be derived from CoS (priority).

The CoS priority is used by oversubscription management for drop precedence and various priority based scheduling decisions.

19.3.3 Resource Usage Counting

Each entity, on each hierarchical level, has a set of resource usage counters.

- For IMEM and EMEM resources, the counter counts usage of packet memory buffers.
- For Job resource the counter counts usage of Job descriptors.
- IMEM counters are 16b.
- EMEM counters are 28b.
- Job counters are 15b.

HIERARCHY LEVEL	*TOTAL RESOURCE COUNTER*	*GUARANTEED RESOURCE COUNTER*	*COUNTER OF GUARANTEED RESOURCE FOR FC BUFFER*
Budget ID	V	V	V
Logical IF	V	V	V
Group	V	-	-
Global	V	-	-

When an entity on a higher level hierarchy is mapped to a lower level entity, the lower level entity counter is updated in parallel. The update is determined per counting decision that is based on guarantee resources usage case.

19.3.4 Resource Usage Thresholds

There are several sets of resource congestion monitoring thresholds:

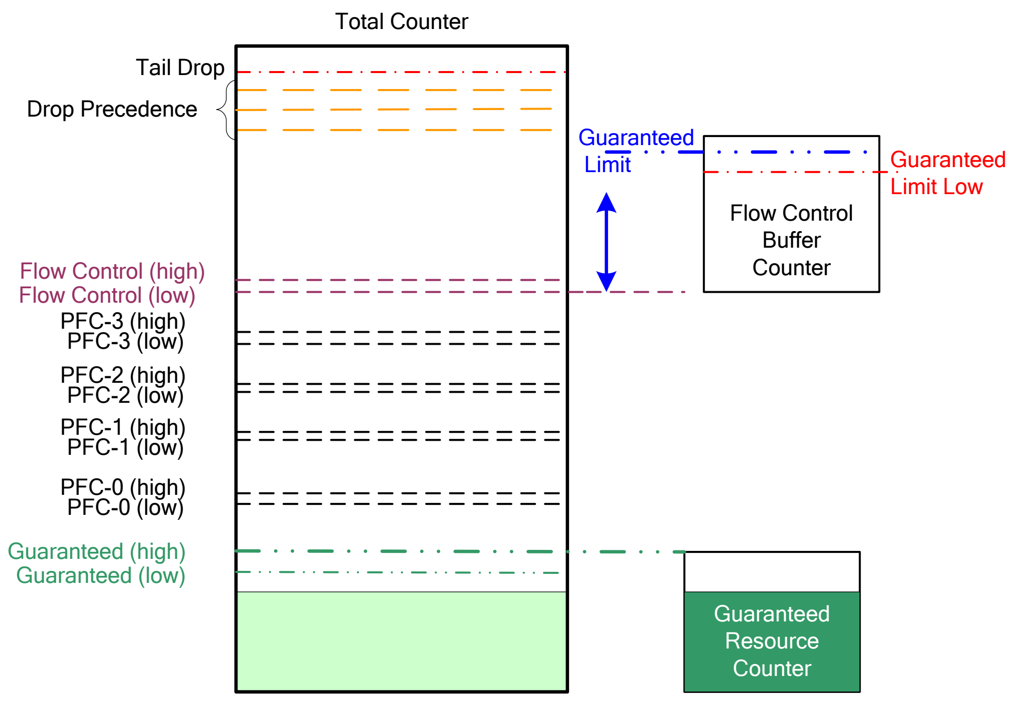
- **Guaranteed resources** are comprised of a set of hysteresis threshold that issued to determine the resource budget that is guaranteed (none shared) to entity. The resource that is allocated as guaranteed is marked as such till it is recycled. The packet drop decision takes in consideration the fact the resource was allocated from guaranteed budget.
- **Priority Flow Control (PFC)** is comprised of four sets of hysteresis thresholds. The PFC segment the budget to five congestion regions that trigger PFC generation to link partner.
- **Link Level Flow Control** is a set of hysteresis thresholds. The FC thresholds are used to trigger LL FC generation to link partner. When LL FC is generated (also because of percolated FC from lower level), the allocated resources are consumed also counted from Flow Control buffer counter and marked as Guaranteed.
- **Drop Precedence** is comprised of three thresholds that segment the budget to four congestion regions. Each congestion region allows packet drop per designated set of priorities.
- **Tail Drop** is a threshold that guards the entity from using un-assigned resources. The congestion above Tail drop cause all traffic mapped to entity to be dropped.

The table below describes threshold types supported per hierarchical level:

THRESHOLD	*BUDGET ID*	*LOGICAL IF*	*GROUP*	*GLOBAL*
Guaranteed	x2	x2	x2	-
Guarantee Limit (FC buffer Counter)	x1	x1	-	-
PFC	- (*)	4x2	4x2	4x2
LL FC	x2	-	-	-
Drop Precedence	x3	x3	x3	x3
Tail Drop	x1	x1	x1	x1
Guaranteed Resource Counter	x1	x1	-	-
Flow control buffer counter	V	V		

(*) The Budget ID can use Guaranteed and LL-FC thresholds as a set of PFC thresholds.

The threshold profiles configure all thresholds per entity. Each resource pool has its own set of profiles. There are 32 Threshold profiles for Budget ID. There are 32 Threshold profiles for Logical IF. There is an individual profile per Group entity and Global.



19.3.4.1 Resource Guarantee

In order to differentiate between guaranteed resource consumption and shared resources, the FCU maintains a per entity dedicated Guaranteed Resources counter and a set of hysteresis thresholds on Total counter.

The guaranteed budget can be configured per Budget ID (channel) and per Logical Interface.

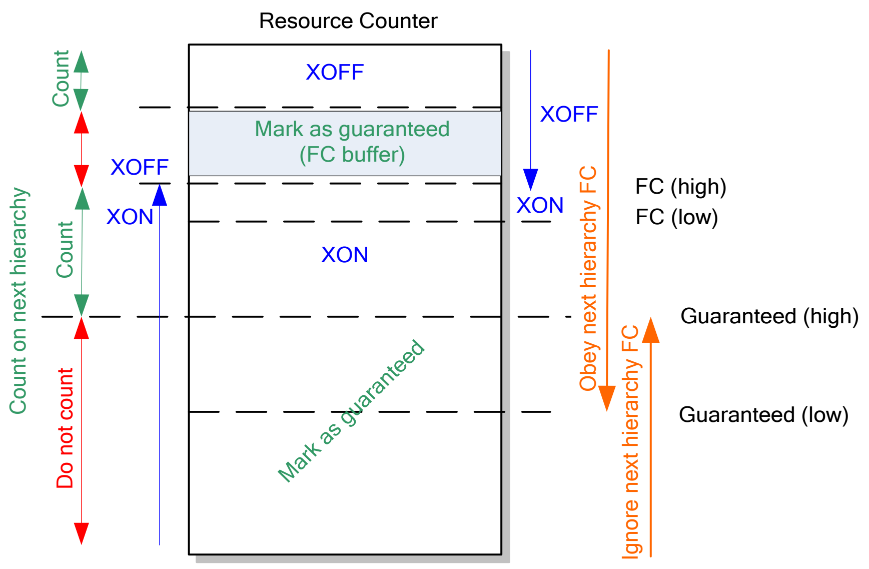
The entity consumes Guaranteed resources before it consumes shared resources. This allows elaborated schemes of resource sharing.

Consumption of Budget ID guaranteed resources before consuming from Logical Interface budget. And the Logical Interface resources consumed from its guaranteed budget before Group shared pool is used.

Each resource is marked as either a Guaranteed or Non-guaranteed consumed resource per BID and LIF level.

- *Note: Resources guarantee per packet cannot be enabled on LIF and BID simultaneously.*

The entity may consume guaranteed marked resources in addition to guaranteed budget from Flow Control buffer when the link partner is signaled XOFF due to congestion.



Until an entity consumes guaranteed resources (up to Guaranteed high threshold), it is not affected by depletion of shared resources and thus ignores the Flow Control status due to higher hierarchy.

After consumption of all guaranteed resources (above Guaranteed upper threshold), the entity consumes shared resources and thus becomes subject of higher hierarchy Flow Control status.

The table summarizes combination of BID and LIF guaranteed and FC buffer usage:

BID/LIF CONGESTION STATUS:	***LIF IN FC BUFFER ZONE***	***LIF IN SHARED ZONE***	***LIF IN GUARANTEED RESOURCE ZONE***
BID in FC Buffer Full	Count on BID total counter. Count on LIF.		
BID in FC Buffer Zone	Count on BID total counter. Count on BID FC counter. Do not count on LIF. BID and Channel/Class ignore LIF FC status. Mark packets as BID guaranteed.		
BID in Shared Zone	Count on BID total counter XOFF BID associated channel/class. Count on LIF total counter. Count on LIF FC buffer counter. Mark packets as LIF guaranteed.	Count on LIF total counter.	Count on LIF total counter. Count on LIF guaranteed resources counter. Mark packets as LIF guaranteed.
BID in Guaranteed Resources Zone	Count on BID total counter. Count on BID guaranteed resource counter. Do not count on LIF. BID and Channel/Class ignore LIF FC status. Mark packets as BID guaranteed.		

The Guarantee Limit threshold sets the effective size of the FC buffer. The FC buffer allows consuming additional guaranteed resources (beyond the Guaranteed counter quota) as a graceful reaction to XOFF flow control request per channel/class or link.

19.3.5 Per Priority Flow Control

The IMEM/EMEM packet data memory buffer resource congestion indication is used to generate a PFC Ethernet frame on Ethernet interfaces.

The 8b PFC indication is combined with Rx FIFO congestion to generate a PFC frame as well as periodic flow control when 8b PFC is non zero.

Each Ethernet port can be enabled to generate PFC or Link-Level FC. Alternatively, these may be disabled.

The Priority Flow Control (PFC) generation can be enabled on Ethernet and Interlaken interfaces.

The PFC is combination of:

- Receive port input FIFO fullness per PFC thresholds
- Job/IMEM/EMEM FCU Logical Interface resource counter and PFC thresholds
- Job/IMEM/EMEM FCU Group resource counter and PFC thresholds
- Job/IMEM/EMEM FCU Global resource counter and PFC thresholds

Jobs resource counter hierarchy (where Job ID = IMEM BID = EMEM BID).

The resource counters are compared to set of 4 hysteresis thresholds. The comparison yields a 5 level result. Each level is configured with an 8-bit PFC value, where level 0 is explicitly set to 00000000b.

Each contributor generates an 8-bit value. The generated PFC is an OR function of all contributed values. PFC indications use a bit mask of 8b, no inherent priority “levels”, i.e. priority X is not higher than any of the others.

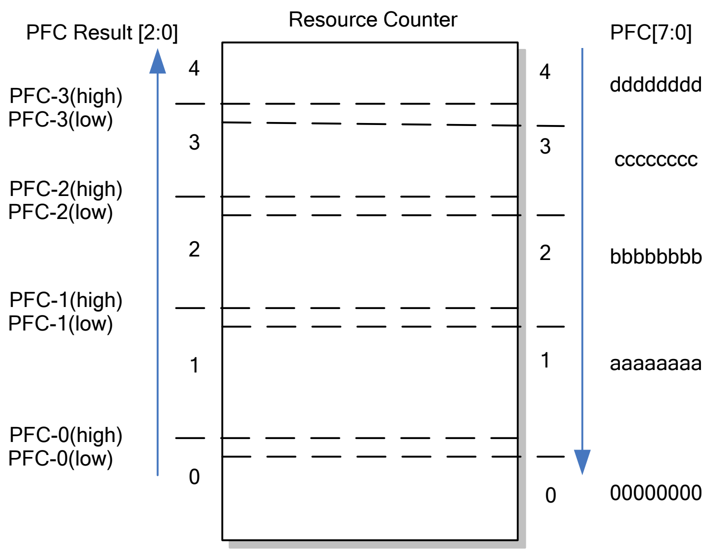
Budget ID based PFC:

The Budget ID congestion can be used for PFC generation for Interlaken channel or PFC per Class.

The Budget ID can use LL FC thresholds and Guaranteed thresholds for PFC (without hysteresis):

- LL FC high = PFC-3(high/low)
- LL FC low = PFC-2(high/low)
- Guaranteed High threshold = PFC-1(high/low)
- Guaranteed Low threshold = PFC-0(high/low)

Figure 19-4. Five-level Priority Flow Control



Example:

PFC[7:0] - bit value '0'=XON / '1'=XOFF

RESOURCE FILL REGION	PFC[7:0]
4	11111111
3	00000111
2	00000011
1	00000001
0	00000000

19.3.6 Link Level Flow Control

The hierarchical management entities of FCU IMEM/EMEM/Job resource pools support sets of hysteresis thresholds to generate Link Level Flow control (XOFF/XON) via in-band and out-of-band signaling.

- The Budget ID resource usage total counter supports set of thresholds: LL FC high, LL FC low.
- The Logical IF, Group and Global resource usage total counters support a set of Priority FC hysteresis thresholds PFC 0-3 (high/low); total of 8 thresholds per counter.

The LL FC status is signaled to the transmit MAC. The transmit MAC generates XON/XOFF on status changes and also periodically.

The status is combined of all hierarchy levels. The father hierarchy XOFF is ignored when:

- Channelized Interface: Per channel LL FC is combined of Budget ID, Logical IF, Group and Global.
- Un-channelized Interface: LL FC is combined of Logical IF, Group and Global.

The vectors logical OR of 8b PFC status of Logical IF, Group and Global is bitwise ANDed to get 1b LL FC status: 0-XON / 1-XOFF.

19.3.7 Class Based Flow Control (CBFC)

The FCU supports flow control per application Class category. The Class is per packet assigned by the ICU packet header parsing. The CBFC is per FCU resource (IMEM/EMEM/Job) global mode.

The BID total counters are assigned as per Class resource usage counters. The BID is fixed assigned to LIF. The LIF represents dedicated physical port.

Two modes are supported:

- Fixed 8 Classes per port. 16 LIFs. Up to 12 physical ports per side. Port BW >= 40Gbps.
- Fixed 2 Classes per port. 64 LIFs. Up to 48 ports per side. Port BW <= 10Gbps port.

There are 8 Global counters. Each Global counter is fixed mapped to a Class.

- For 8 Classes per port, all 8 Global Classes (Global Class 0 to Global Class 7) are used.
- For 2 Classes per port, two Global Classes (Global Class 0 and Global Class 1) are used.

8 CLASSES PER PORT. 16 LIFS (4*N). 12 PHYSICAL PORT PER SIDE			***2 CLASSES PER PORT. 63 LIFS. 48 PHYSICAL PORT PER SIDE***		
BID-0	LIF-0	Global Class-0	**BID-0**	LIF-0	Global Class-0
BID-1		Global Class-1	**BID-1**		Global Class-1
...		..	...	..	..
BID-7		Global Class-7	**BID-7**	LIF-3	Global Class-1
BID-8	LIF-4	Global Class-0	**BID-8**	LIF-4	Global Class-0
BID-9		Global Class-1	**BID-9**		Global Class-1
...			...	..	..
BID-15		Global Class-7	**BID-15**	LIF-7	Global Class-1
...	..	..	...		
BID-120	LIF-60	Global Class-0	**BID-120**	LIF-56	Global Class-0
BID-121		Global Class-1	**BID-121**		Global Class-1
...		..	...	..	..
BID-127		Global Class-7	**BID-127**	LIF-63	Global Class-1

The resource counting on BID total counter and corresponding Global class counter are subject to guaranteed resource counting scheme (resource sharing methods).

The BID Total counter per Class and Global total counter per Class are both monitored by Link Level Flow Control hysteresis thresholds. Both BID and Global LL FC indication are combined to generate per Class Flow Control.

CLASS BELOW GUARANTEED (HIGH)	***IGNORE GLOBAL FC***	***COUNT GLOBAL***	***CLASS FC-BUFFER***	***CLASS LL-FC***	***GLOBAL LL-FC***	***COMBINED LL-FC***
No	No	Yes	No	**XOFF**	-	**XOFF**
No	No	No	**Yes**	**XOFF**	-	**XOFF**
No	No	Yes	No	-	**XOFF**	**XOFF**
No	No	Yes	No	**XON**	**XON**	**XON**
Guaranteed	No	No	-	**XOFF**	-	**XOFF**
Guaranteed	No	No	-	-	**XOFF**	**XOFF**
Guaranteed	No	No	-	**XON**	**XON**	**XON**
Guaranteed	Ignore	No	-	**XOFF**	-	**XOFF**
Guaranteed	Ignore	No	-	**XON**	-	**XON**

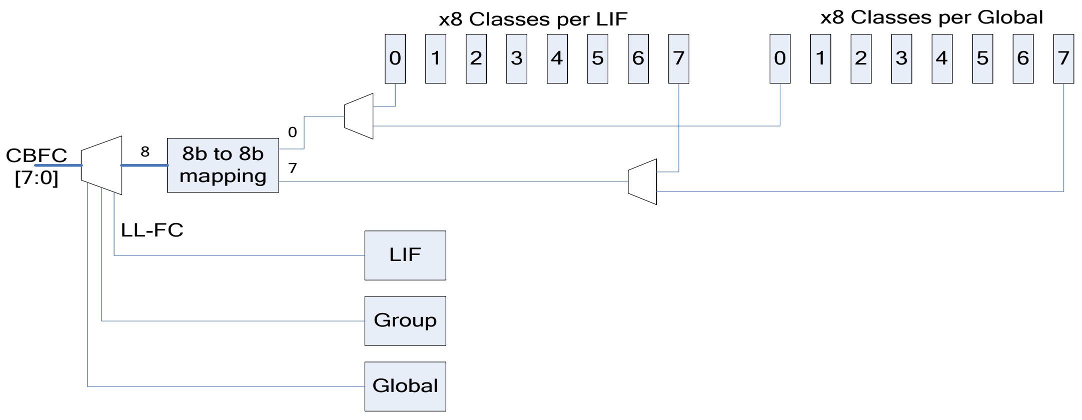
Count Global means to count on global Class counter due to this BID update.

The CBFC generates 8b or 2b per LIF. The LIF is mapped to a physical transmit IF to signal in-band CBFC.

The BID (0-XON/1-XOFF) status and corresponding Global Class status are combined by the logical Or function. When the BID counter is in Guaranteed zone (also FC buffer zone), the Global Class status is ignored.

The resulting 8b per LIF CBFC is flexibly remapped to a signaled 8b CBFC value by an 8 entry x8b table.

The CBFC of 8b is set to XOFF on all Classes when ether LIF, Group or Global are in LL FC XOFF state.



19.3.8 PFC and CBFC Status Interlaken In-band Reporting

The Interlaken RxFIFO PFC congestion combined with the FCU PFC/CBFC congestion is reported as in-band on the Control header. Several options are supported:

- Interlaken header 8 multi-use bits
- Any 8 consecutive bits in the Interlaken channel status calendar (8 PFC bits overwrite 8 channel slots). PFC is sent only once per calendar period.
- Interlaken channel status Calendar 72b bits. The 4 low bits of PFC are used to enable/disable four configurable vectors of 72b that are OR-ed.

19.3.9 Drop Precedence

All entities of hierarchical levels support three Drop Precedence (DP) thresholds. The thresholds are set with a per entity configurable threshold profile. The Drop Precedence of LIF affects the ICU/RxNDMA decisions on which packet priority is eligible to be dropped. For the BIDs (127-96) that are assigned to LIFs which are used by internal special interfaces (i.e. loopback IFs), the Drop Precedence affects the PMU decision on which Job priority is eligible to be dropped.

The Total counter value compared with DP thresholds yields a result that spans four values (0-3).

- The DP value 0 indicates not to drop any priority.
- The DP value 1 indicates eligibility to drop lowest priority (3).
- The DP value 2 indicates eligibility to drop lowest (3) and medium priority (2).
- The DP value 3 indicates eligibility to drop priorities (3,2,1) but not the highest (0).

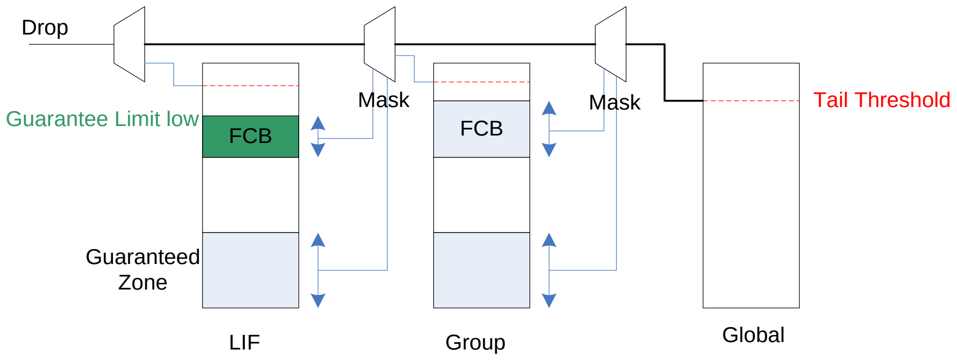
The DP of all hierarchy levels is combined per worst (3>2>1>0) case of any level. This combination is done as percolation from higher to lower hierarchies. The higher hierarchy DP affects can be disabled per entity.

19.3.10 Tail Drop Thresholds

All entities of hierarchical levels support Tail (absolute) Drop threshold. The threshold is set with a per entity configurable threshold profile. The Tail Drop signals ICU/RxNDMA to drop all packets per Logical IF.

The LIF Tail Drop is the worst case combination of Group and Global resources. The percolated Group TD status is ignored (masked) when LIF is in Guaranteed zone (below Guaranteed Threshold) or in FC buffer zone (below Guaranteed Limit low threshold). The percolated Global TD status is ignored when Group status is in Guaranteed zone (below Guaranteed high threshold) or in FC buffer zone (below Guaranteed Limit high threshold).

Note that LIF FC buffer used for TD is smaller than LIF FC buffer used for PFC and for Group counter update masking.



19.3.11 FCU Status Out-of-Band Reporting

The FCU resource management entity status can be reported via OOB serial bitstream signaling utilizing repeated calendar of bits.

The reported status conveys the entity Priority Flow Control 5 value result (0, 1, 2, 3, 4) coded in three bits.

The report also conveys a single bit for the XON/XOFF state for the managed entity.

The PFC Result reported:

- The Global PFC Result[2:0]
- The Group local PFC Result[2:0] or Group PFC Result combined with Global PFC Result
- The LIF local PFC Result[2:0] or LIF PFC Result combined with PFC Result percolated from Group
- The BID local PFC Result[2:0] or BID PFC Result combined with PFC Result percolated from LIF

The XON/XOFF state reported:

- The Global LL FC
- The Group local LL FC or Group LL FC combined with Global LL FC
- The LIF LL FC or LIF LL FC combined with LL FC percolated from Group
- The BID local LL FC or BID LL FC combined with LL FC percolated from LIF

The LL FC is derived from PFC[7:0] vector by PFC[7:0] AND MASK[7:0] and bitwise OR of 8 vector bits.

19.4 Transmit Throttling by Received Flow Control

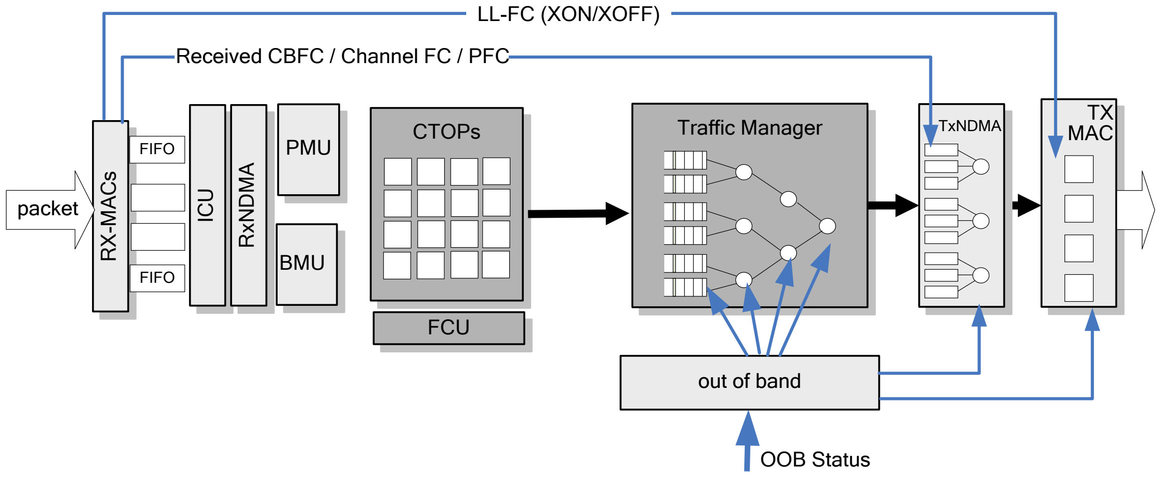
Received flow control messages allow a congested stage external to the NPS to notify it to stop scheduling/sending a specific type of traffic. NPS supports any level of detail in flow control, from the fast response link level flow control to the per queue specified proprietary messages:

- Link Level Flow Control per transmit interface
- Priority Flow Control per transmit interface
- Interlaken channelized FC on burst boundary per 128 channels
- Per TM: TxNDMA (Cell) Output Queue Channel Based Flow Control per 144 queues
- Per TM: TxNDMA (Cell) Output Queue Channel per Priority Flow Control per 144x4 priority queues
- TM L0-L4 Hierarchy Flow Control
- TM scheduler shaper rate adjustment by management software.

The origin of flow control is:

- In-band messaging: Ethernet pause, Ethernet PFC packets, Interlaken frame status fields.
- Out-of-band signaling: Interlaken status interface, OIF SPI 4.2 status interface
- SW control

Figure 19-5. Transmit throttling by received flow control



19.4.1 Link Level Flow Control

Incoming link level flow control is detected directly by the NPS receive MAC layer. Each input port is coupled with an appropriate output port allowing it to pause transmitting data.

Upon reception of an IEEE 802.3x Ethernet Pause packet, the transmit MAC will pause transmission for the time period specified in the packet timer. Alternatively, the MAC can be configured to ignore the received Pause frame.

The Interlaken interface supports Link Level FC either in the In-Band-Flow-Control field or the Multiple-Use field. In both cases, a single field bit is selected for Link Level FC.

The NPS supports LL FC re-mapping to Priority Flow Control. This feature allows high priority traffic not to be paused by LL FC. The globally configurable 4-bit PFC vector defines which classes are paused per LL FC.

Link level flow control can also be received through the out-of-band interface.

The typical response time from receive of LL XOFF FC to pause of transmit is transmit link time of 128B + current transmit packet.

The typical response time from receive of LL XON FC to start of transmit is 134ns.

19.4.2 Priority Link Level Flow Control

The NPS supports in-band per Priority flow control signal reception and traffic scheduling of permitted priorities. The in-band PFC is supported on Ethernet ports and Interlaken. The PFC is also supported for OOB received signaling.

The Ethernet PFC format supported is IEEE 8021.Qbb. An Ethernet port enabled for PFC does not support LL FC in parallel.

Note: The XOFF latency of PFC is not immediate, and the MAC probably keeps transmitting until the TxNDMA transmit FIFO is empty. The FIFO size is configurable by 8 profiles in 16B resolution. The FIFO size rather be set to accommodate the MTU packet size.

The Interlaken format Multi-Use 8b field can be enabled for PFC interpretation or the in-band calendar bits.

In-band XOFF causes the TxNDMA to stop transmitting data of the appropriate priorities.

The 10G/40G/100G Ethernet port supports per priority Pause Timer setting according to received PFC frame.

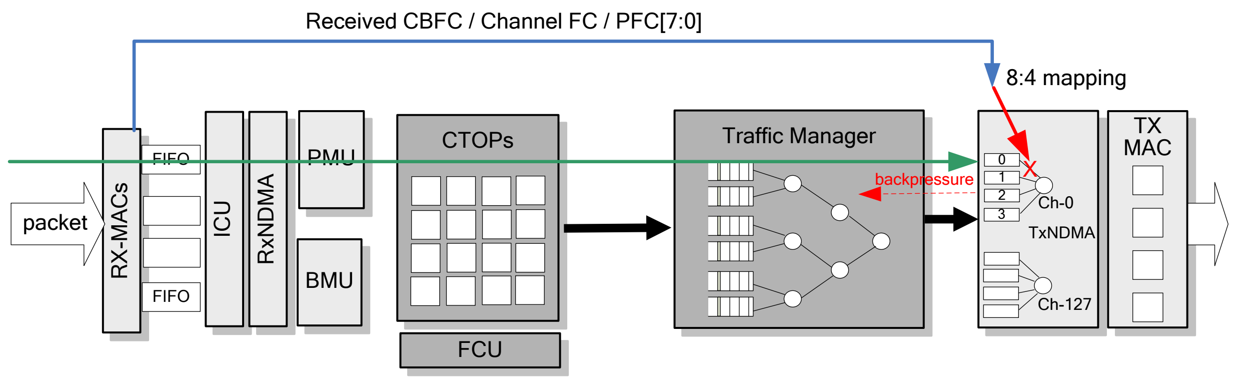
The received PFC per 8 priorities is mapped to 4 scheduler priorities supported by the TxNDMA Output Queue channel. There are 128+16 channels x 4 priorities per side. The 128 channels are used for network interfaces while 16 are for internal interfaces. The mapping of 8b to 4b is per 8 configurable 4b values, each AND-ed with its FC and then OR-ed to produce a 4b value.

The Output Queue congestion due to paused priority is translated to backpressure towards the TM scheduler. This channel congestion is monitored by a single monitor counter with 4 hysteresis thresholds. The resulting congestion status is translated into five states:

1. No backpressure, scheduling is done for all TM priority levels.
2. Stop scheduling from priority 3.
3. Stop scheduling from priorities 2-3.
4. Stop scheduling from priorities 1-3.
5. Stop scheduling in all priorities.

The thresholds are configurable by 16 profiles for packet-count thresholds (see the "TM Packet Switch" section in the *Traffic Management* chapter).

Figure 19-6. Per class of service flow control



19.4.3 Interlaken Transmit LL FC by per Channel FC

The Interlaken header Multi-use bits and channel Calendar bits can be used for Link Level FC on the Interlaken transmit interface. The transmit Xoff is on burst boundary.

- When 8 Multi-use bits are used for LL FC, the selected bit is regarded as LL FC per configurable polarity.
- When channel Calendar bits are used, a bitmap of up to 256 bits reconstructed from the Interlaken control word in-band calendar over several headers, 16b per header. When all channel bits assigned to an Interlaken ports are Xoff, the Interlaken transmission is paused. Alternatively, a single selected bit can be used to LL-FC the Interlaken port.

When Interlaken port transmission is paused by FC, the indication is also signaled to SW until read by the SW.

19.4.4 Interlaken Channelized Flow Control

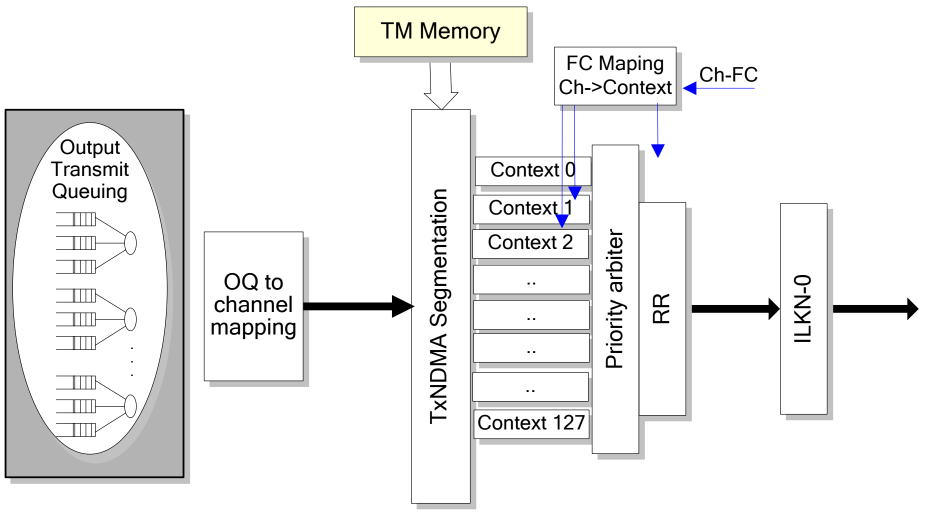
The Interlaken interfaces support packet mode only.

For each TxNDMA, up to 128 contexts can be supported. Each segmenting context can be mapped to an Interlaken channel.

The channels are scheduled by WRR arbitration with optional, strict, high-priority assignment.

The Interlaken received channelized flow control is remapped back to the context in order to flow control transmission of the assigned context.

Figure 19-7. Interlaken channelized flow control



19.4.4.1 Per Channel Flow Control

The NPS supports flow control per output transmit channel (selected by TM PSID).

The TxNDMA responds and stops a specific channel on packet boundary.

The per channel FC signal is received either in-band via Interlaken or via out-of-band signaling.

The Interlaken In-Band-Flow-Control field is assigned to a 256b calendar (may also exceed 256b when LL FC utilizes one of 16 bits in a calendar word). The calendar is mapped to TxNDMA channel FC. The configurable FC Start Channel and FC Number of Channels define the TxNDMA channel range to be flow controlled.

For a specific channel, the per Channel FC is mutually exclusive with Channel per Priority FC.

19.4.4.2 Channel per Priority Flow Control

The NPS supports flow control for each of four priorities of output transmit channels (selected by TM PSID).

The TxNDMA responds and stops specific channels/priority.

The TxNDMA supports 128 channel * 4 priorities.

The per channel FC signal is received via out-of-band signaling.

For a specific channel, the per Channel FC is mutually exclusive with Channel per Priority FC.

The Xoff is packet boundary only.

19.4.5 TM Traffic Transmit Throttling via OOB Signaling

To offer extended support for system wide QoS, the NPS architecture offers a distributable managed Virtual Weighted Fair Queuing to implement congestion avoidance via E2E backward congestion notification and VOQ throttling.

The relatively low buffering resources offered by the switch fabric results in very low latencies for congestion conditions affecting traffic flow and require a very fast management response to reduce the traffic flow. This results in the need for fast out-of-band congestion signaling and fast traffic scheduling control.

The Backward Congestion Notification flow control results in the NPS sending only the eligible share of its bandwidth for each target output interface, managing the congestion and freeing fabric bandwidth and resources for other flows. This provides the overall system with improved throughput, better performance and better control. The fabric is transformed from an unmanaged resource to a managed resource with user control over traffic flows under periods of congestion.

Each level of the TM hierarchy can be individually throttled by OOB signaling.

The TM hierarchical entities are grouped for OOB control. Each entity in a group can be individually flow controlled.

For each TM engine, the grouping per TM level is listed below:

- TM Flow level (L4) FC
  - Up to 8K flows can be controlled 2K first flows of each TM chunk (1/4 TM).
    The flows can be enabled for OOB FC per group.
    There are 128 groups of 4 consecutive Classes (a nibble) in each TM-chunk.
- TM Class level (L3) FC
  - The entire range of 32K Classes can be controlled.
    Each group of 4 consecutive Classes can be selected for flow control.
- TM Subport level (L2) FC
  - The entire range of 4K Subports can be controlled.
    Each group of 4 consecutive Subports can be selected for flow control.
- TM Port level (L1) FC
  - The entire range of 512 TM-Ports can be controlled.
    Each group of 16 consecutive TM-Ports can be selected for flow control.
- TM Interface level (L0) FC
  - The entire range of 128 TM-L0 entities can be controlled.
    Each TM-Interface can be selected for flow control.

19.5 Out-of-Band Signaling

The NPS supports two bi-directional OOB interfaces for congestion reporting and traffic transmit throttling. Each interface can be configured to operate as an Interlaken status interface or a DDR OIF SPI-4.2 status interface.

Table 19-1. NPS ILKN OOB FC interfaces

NPS PINS	*IN/OUT DDR*	*INTERLAKEN MODE*	*OIF SPI 4.2 MODE (DDR)*
FC<s>_RSCLK	In	FC_CLK	TXSCLK
FC<s>_RDATA	In	FC_DATA	TXSTAT[0]
FC<s>_RSYNC	In	FC_SYNCH	TXSTAT[1]
FC<s>_TSCLK	Out	FC_CLK	RXSCLK
FC<s>_TDATA	Out	FC_DATA	RXSTAT[0]
FC<s>_TSYNC	Out	FC_SYNCH	RXSTAT[1]

<s> indicates the side (East/West).

Each OOB interface is controlled by a configurable calendar.

The number of calendar slots of outgoing OOB is configurable from 1 to 2K.

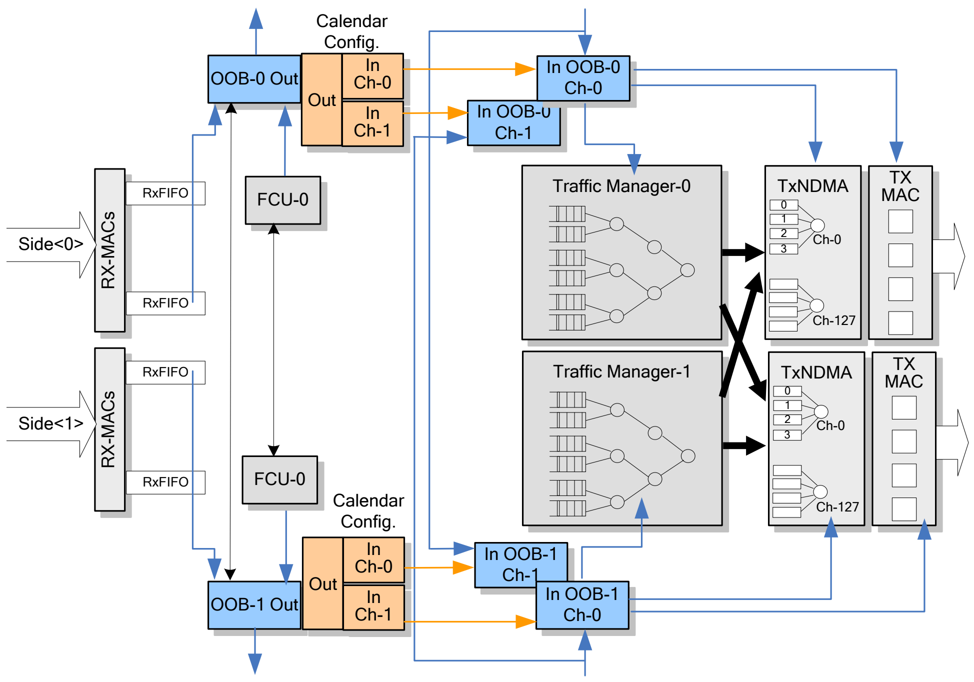
The number of calendar slots of incoming OOB is configurable from 1 to 64K.

For the OIF SPI-4.2 type interface there is also calendar multiplicity (repetition) configuration of 1-64.

For Interlaken OOB interface configuration, the OOB bitstream is CRC protected. For OIF SPI OOB interface configuration, the bitstream is DIP2 protected. The CRC/DIP2 is generated on transmission and verified on reception. When a CRC/DIP2 error is encountered upon reception, the mechanism employs hysteresis to return to ‘in-synch’ state. The number of consecutive errors to switch to ‘out-of-synch’ and the number of consecutive CRC/DIP2 correct calendars to switch to ‘in-synch’ state is configurable.

Each OOB input interface is connected to both In-OOB engines (0 and 1). Each In-OOB engine combines two channels with a dedicated calendar each. Both channels control TM engine and adjacent TxNDMA/ MACs on their same side.

Each Out-OOB engine is capable of reporting its own side’s congestion status and also status of the other side’s Out-OOB engine.

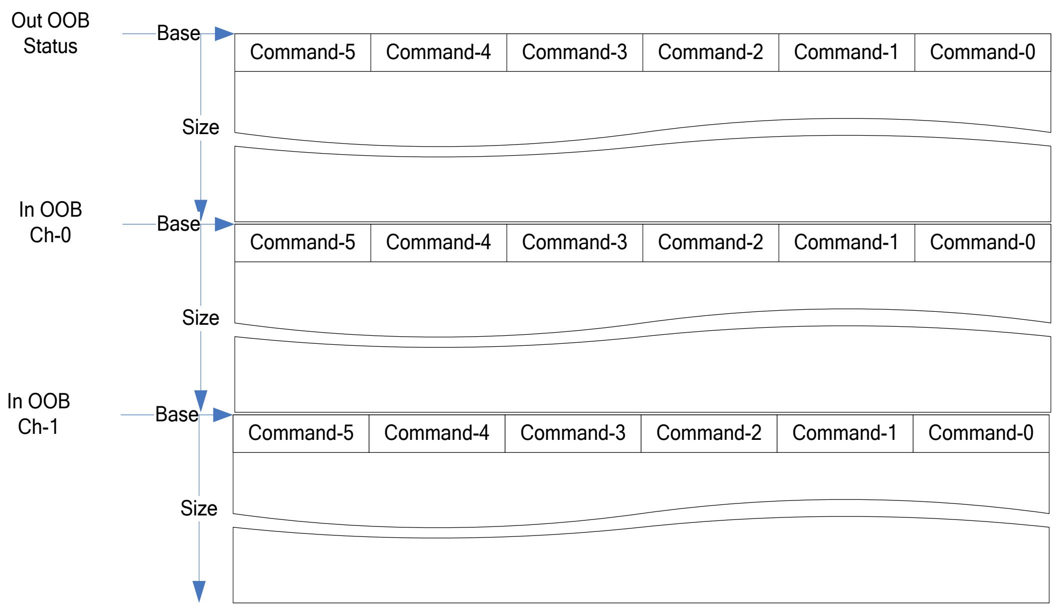


For each OOB engine (In and Out), the calendar slots assignment is controlled by configuration memory of 512x120b lines. Each line contains the configuration of 6 commands. Each command can be used to configure 1, 4, or 16 calendar slots. The configuration memory is flexibly partitioned to control slots for:

- Transmitted (Out OOB) congestion reporting. Single bit per entity XON/XOFF congestion status. Three bits for PFC congestion status.
- Received (In OOB) flow control for traffic transmission throttling. Single bit per entity XON/XOFF.
  - Based on Interlaken convention, a ‘1’ is chosen to identify the XON state, indicating permission for the transmitter to send data on that channel. A ‘0’ identifies the XOFF state, indicating that the transmitter should cease sending data on that channel.

For each partition there is a configurable base and size in line multiples:

Figure 19-8. OOB configuration memory mapping



19.5.1 Input OOB Flow Control

Up to 64K calendar slots can be allocated for 512 lines * 6 commands (20b command) * [1, 4, 16, 32] slot bits.

The incoming OOB FC calendar controls the:

- TM scheduling hierarchies
- TxNDMA (cell output queue) per channel and per priority
- Transmit MAC Link level flow control

The OOB configuration memory command controls 1/4/16/32 consecutive calendar slots. Each command is configured with Command code and controlled entity ID.

The 20b command is:

- ID 14b – The index of the controlled entity
- Command 2b – '00' Valid command; '01' -'11' Ignore command.
- Destination code 4b

DESTINATION CODE	***DESTINATION***	***CONSECUTIVE STATUS BITS IN CALENDAR***
0000	TM - class (type = 0)	4
0001	TM - sub-port (type = 1)	4
0010	TM - port (type = 2)	16
0011	TM - interface (type = 3)	1
0100	TM - flow (type = 4)	32
0101	TM - fast q (type = 5)	16
1000	Reserved	16
1001	COQ - IP event	16
1010	COQ - CH event	4
1011	COQ - CP event	16
1100	MAC - link level	1
1101	COQ-IP event single bit	1
1111	ILKN Retransmit	1

19.5.1.1 Ethernet/Interlaken Link Level Flow Control via OOB

The per port LL FC can be enabled for incoming OOB calendar slots. The slots selected for LL FC are at equal calendar slot intervals. Each OOB interface can be configured to allocate a slot, each 2^N slots for LL FC, where N is in the range 0-15.

The allocated LL FC slots are used to FC a configurable vector of Ethernet and Interlaken ports. The port vector is scanned in a round-robin fashion.

Example of slot assignment with N=2 and Ethernet port vector is:

XFI-0, XFI-1, XFI-2, XFI-3, XLAUI-2, XLAUI-3, XLAUI-4:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
XFI-0				XFI-1				XFI-2				XFI-3				XLAUI-2				XLAUI-3				XLAUI-4	

When using a calendar-multiple that is larger than one, the next sequence is not identical to the previous one, and the port's LL indication continues from the previous sequence.

19.5.2 Congestion Reporting via OOB Signaling

The OOB transmits congestion status for command selected entity. The congestion status is a single bit per entity and three bits for PFC congestion reporting.

The OOB calendar configuration memory command determines the reported entity and number of calendar slots consumed by the report.

FCU IMEM/EMEM/Job descriptor pool congestion.
Each pool congestion reports:

- Budget ID range of 128
- Logical IF range of 64
- Group range of 8
- Global
- For each reported entity the report can be:
  - LL FC XON/XOFF status 1b
  - PFC state 3b

  The maximal report length is 3 *(128 +64+8+1) * 4b = 2412 bits

The Receive input FIFO congestion report is:

- 48 (10GbE/40GbE IFs) + 4 (100GbE /ILKN IFs) * 3b = 156 bits

The Out OOB SRAM configuration command 20b syntax is:

- Type[2:0] - Local or remote FCU/FIFO and FCU hierarchy level.
  The Type[2:0] is:

TYPE[2:0]	REPORT SOURCE		POOL[2B]	ENTITY ID RANGE
000	Local FCU	Budget ID	0 - IMEM 1 - EMEM 2 - Job	128
001		Logical IF		64
010		Group Global		8 1
011	Local Rx FIFO		9b	48 physical IF, ILKN IF
100	Other FCU	Budget ID	0 - IMEM 1 - EMEM 2 - Job	128
101		Logical IF		64
110		Group Global		8 1
111	Other Rx FIFO		9b	48 physical IF, ILKN IF

- Reported Entity ID[12:0] - Index to table with pre-configured entities
- Slot Command[3:0] - Number of bits output via OOB.
- Worst case of FCU and RXFIFO congestion status

The Reported Entity ID [12:0] is:

For FCU: Report Entity table Line[6:0] and field Offset[3:0]

For RxFIFO: 9b selector of physical interface per side.

Report Entity table:

		LINE#	FIELDS INDEXED BY OFFSET[3:0]
XON/XOFF Flow control Status 1b	Budget ID	1	FC status CHNL0-CHNL11
		2	FC status CHNL12-CHNL23
		…	
		10	FC status CHNL109-CHNL120
		11	FC status CHNL120-CHNL127
	Logical IF	12	FC status PHYINT0-PHYINT11
		13	FC status PHYINT12-PHYINT23
		…	
		17	FC status PHYINT60-PHYINT64
	Group	18	FC status SRCGRP0-SRCGRP8
	Global	19	FC status Global
Priority Flow Control State 3b	Budget ID	20	FC state CHNL0-CHNL3
		21	FC state CHNL4-CHNL7
		…	
		51	FC state CHNL124-CHNL127
		52	FC statePHYINT0-PHYINT3
		52	FC statePHYINT4-PHYINT8
		…	
		67	FC statePHYINT60-PHYINT63
	Group	68	FC state SRCGRP0-SRCGRP3
		69	FC state SRCGRP4-SRCGRP7
	Group	70	FC state Global

The Slot Command allows specifying a single command to report to 1, 4 or 16 consecutive entities. The Commands are:

- 0000 – Drive 0 for single slot (unused slot)
- 0001 – Drive 1 for single slot (unused slot)
- **0010 – Drive congestion status single bit for Reported Entity ID**
- 0100 – Drive 4 consecutive 0 (unused 4 slots)
- 0101 – Drive 4 consecutive 1 (unused 4 slots)
- **0110 – Drive 4 consecutive congestion status bits for consecutive Reported Entity IDs**
- 1100 – Drive 16 consecutive 0 (unused 16 slots)
- 1101 – Drive 16 consecutive 1 (unused 16 slots)
- **1110 - Drive 16 consecutive congestion status bits for consecutive Reported Entity IDs**

The Reported Entity ID selects 1, 4 or 16 consecutive entities to report. Each congestion report is a single bit/slot and the PFC status is 3 slots.

20. Precision Clock Synchronization Support

20.1 Overview

The NPS supports the Precision Clock Synchronization Protocol for IP networks intended to synchronize between the different network nodes' internal time domain. This chapter describes how the Precision Clock Synchronization application is supported using the IEEE 1588 standard. The protocol can be used in data communications systems where TDM circuit emulation service is provided to support synchronized clocks between the different systems.

The protocol supports systems synchronization accuracy in the sub-microsecond range using minimal network computing resources and local clock hardware support.

The protocol defines the Best Clock source node in the network and distributes the node's internal time to the other nodes on the network. The distribution is done by periodic control packets (independent of the distributed network technology) that trigger internal clock timestamp sampling which is communicated to the other nodes of the network.

This method of using timestamps allows each slave clock in the network to analyze the master's timestamp and the network propagation delay, and then determine its clock delta from the master in the slave clock's synchronization algorithm.

When used in Metro Ethernet networks, NPS participates in the IEEE 1588 protocol as a master distributing its clock to other endpoints, as a transparent switch, as an endpoint, or as combination of the above functions.

As a complimentary mechanism to IEEE 1588, NPS supports ITU-T G.8261 synchronous Ethernet as a method of delivering carrier clocks between nodes. The NPS supports selectable receive recovery per side and a selectable transmit reference clock for SerDes interfaces.

NPS may support thousands of IEEE 1588 simultaneous connections in order to support an intermediate switch function carrying multiple end-to-end IEEE 1588 sessions for multiple time domains.

The IEEE 1588 protocol is often referred to as Precise Time Protocol or *PTP*; this acronym is used throughout the document.

The NPS supports both 1-step and 2-step timestamp modes of operation, required for support of IEEE 1588v2. The receive (ingress) support is common to both 1-step and 2-step operation. The transmit interface support is based on the port configuration (for a specific mode) or can be dynamic (per packet) as controlled by the CTOPs application software. The transmit 1-step time stamping requires that the transmit MAC modify the transmitted packet, while transmit 2-step time stamping requires the generation or processing of a follow-up packet.

All NPS interface types support the 2-step mode of operation.

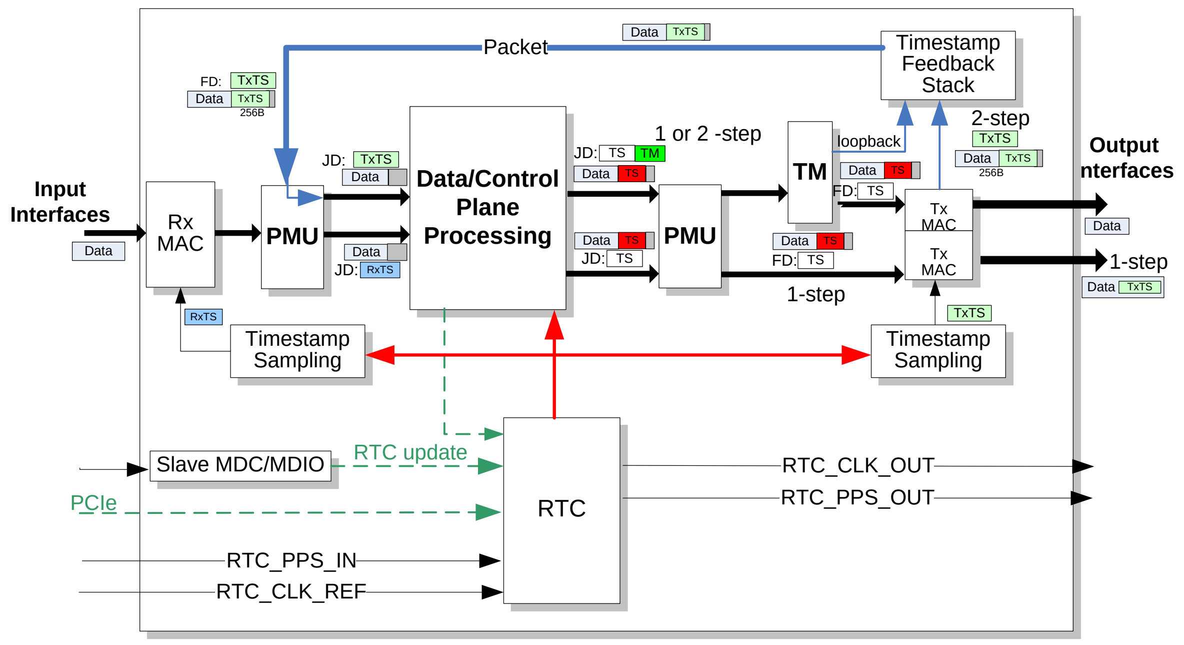
The NPS device is designed to support 1-step timestamp mode for all 10G/40G/100G interfaces by the on-chip MAC hardware.

- *Note that in this chapter the term "flow" relates to 1588 flows.*

20.2 Main IEEE 1588 Hardware Additions to Data Flow

The figure below illustrates the general NPS modules in the IEEE 1588 packet data flow:

Figure 20-1. Basic timestamp hardware support



The additional hardware that supports the IEEE 1588 protocol implementation can be divided to three sections: Ingress stage, Egress stage and the RTC clock.

The architecture includes:

- Accurate RTC clock with update mechanism controlled by the CTOP application, external CPU via PCIe, the Slave MDC, or by the 1PPS input.
- RTC sampling at the ingress Rx MAC.
- Frame modification per 1-step IEEE1588 protocol by the transmit TxMAC.
- Timestamp Feedback Stack that handles egress MAC timestamp for 2-step IEEE1588 implementation.
- TM special queue management for protecting 2-step timestamp feedback stack.
- PMU designated queues for feedback timestamp packets.

20.3 Ingress Stage - Receive Timestamping

The packet begins its path through the NPS at the input stage which includes the RX MAC. The ingress timestamping option support is common for both 1-step and 2-step, and can be configured on a per receive port basis.

The RX MAC sampled RTC timestamp is 5 bytes, including the seconds field (1 byte) and the 4-byte nanoseconds field.

The RTC is sampled when the packet's SFD symbol is detected at the input port.

Each ingress MAC includes a frame length filter for time stamping. A timestamp is sampled from the RTC by the RX MAC for all packets with a user configurable length range. The range includes minimum length and maximum length attributes which allow the user to control which size packets marked with timestamps, supporting all possible frame sizes. The length range is a global configuration attribute; for Interlaken, the maximum length must not be greater than the BurstMax (256B).

The sampled timestamp is written to Frame Descriptor part of the Job Descriptor.

The incoming Job Descriptor contains the

- TS_NS[31:0] - time stamp nano-seconds field.
- TS_SEC[7:0] - 1588 timestamp lower seconds counter lower byte.

The Frame Descriptor TS bit flag is set to indicate the validity of TS_NS and TS_SEC.

The timestamp is used when a packet is parsed and classified as part of an IEEE1588 flow.

20.4 Egress Stage - Transmit Timestamping

The CTOPs data path is responsible for identifying the encapsulated PTP packets, processing the frame based on the required handling (1-step or 2-step) and preparing headers that are attached to the frame and passed to the TX MAC interface of the NPS.

The TM is responsible for specific support requirements for 2-step processing. In applications where external PHYs are used on some system ports that implement the PTP timestamp functions, the CTOPs may also be utilized for translating formats for interoperability.

The egress packet is timestamped when the transmit MAC sends the SFD symbol to the output port of a frame that requires an egress timestamp. The CTOPs identify PTP frames that require the egress MAC timestamp. The JD/FD[TS] bit field indicates to the egress MAC that there is timestamp 16B header attached.

The TX interface supports the following modes of operation in relation to timestamp operation of the MAC:

- Disabled – The interface will not perform any timestamp operation.
- 1-step – The 1-step header is attached and packet is modified on transmit. The 1-step packet can be transmitted via the TM or bypass the TM.
- 2-step – The 2-step header is attached and packet is transmitted without modification, the timestamp feedback message is generated. The 2-step packet must be transmitted via the TM.
- Dynamic – The 1-step/2-step mode is indicated per packet.

The 1-step and 2-step mode settings will place the entire port in the indicated timestamp mode; for example, if in 1-step mode, all packets requiring timestamping for the port will have 1-step operation performed. In these modes, it is required that the application (i.e. CTOPs) guarantee the correct packet PTP format for the configured port mode of operation (i.e. the packets must match the port mode).

Dynamic mode for the port indicates that the CTOP application can utilize the defined TS header flag to indicate to the TX MAC whether to perform either 1-step or 2-step for a packet requiring egress timestamp handling. This is a per packet decision by SW to the MAC, allowing 1-step and 2-step operation concurrently on a "dynamically" configured TX port.

TX MAC CONFIGURATION	*TIMESTAMP*	
1-step	1-step format	
2-step	2-step format	
Dynamic (per TS header)	1-step format	2-step format

The 2-step packets must utilize designated TM queues. The 1-step packets requires no special handling by the TM and can utilize any TM queue.

The transmitted frame descriptor controls the egress packet processing:

- TS – Time Stamp request. When this bit is set, the frame requests PTP timestamp.
- TC – Transmit Confirmation. Instructs Tx NDMA to return confirmation when transmit is done. The confirmation triggers Timestamp Feedback generation.
- TKB – Transmit Keep Buffers. In case of confirmed transmission, instructs the NDMA Tx logic to keep the frame buffers (when set) or recycle them to the BMU (when cleared).

When both TS and TC bits are set this is a 2-step 1588 confirmation packet.

The 2-step IEEE1588 requires Timestamp Feedback generation that is triggered by (TC=1). The generated feedback JD holds the timestamp timer value captured on the Tx MAC.

On Tx timestamp confirmation, the Tx MAC samples the timestamp at the beginning of transmission time and embeds the measurement in the responding PMU job.

On loopback port, the architecture allows the PMU to sample a 1588 timestamp at the moment the JD is injected back into the PMU queue, enabling latency measurements for the loopback paths.

On 1-step IEEE1588 (TS=1 and TC=0), no confirmation is required and the Tx MAC is required to modify the packet header to add the captured timestamp.

For unicast packet transmit with confirmation (TC=1 and TKB=0), the Tx NDMA releases all frame buffers except for the frame header and confirms the transmission by generating JD back to the PMU.

If TC=1 and TS=1, the packet belongs to 2-step 1588 timestamp confirmation flow which has a limited packet rate. Dedicated TM queues are used with end-to-end credit management with their associated Tx MACs (the credit gets cleared after 1588 captured timestamp is pushed back to the PMU 1588 port). The confirmation job descriptor includes the timestamp measured at the Tx MAC.

If TC=1 and TS=0 (standard packets), the frame belongs to a high rate Tx confirmation flow, indicating to SW when a frame has been transmitted to the MAC. One usage example of such a flow is a TCP retransmit application, where SW chooses to implement the transmit window in memory FIFO and expects the NDMA to accelerate recycling of the frame buffers. The TCP timeout counter can be triggered by the confirmation to measure time from actual transmission to the line, excluding queuing by the TM (and also be able to detect drop at the TM if timeout was also set at launch time). In case of timeout (or TM drop), the frame is built again by SW using new buffers. The Tx NDMA does not release the header buffer and points to it in the confirmation FD response.

On a loopback flow TC=1 indicates sync operation, indicating to the data path following the TM that this packet must not be dropped even if the TM queue is being flushed.

On confirmed flows SW may use context data on the first buffer, preceding the frame header data, to keep information, such as mapping back to the TCP session, or setting an action to take when the confirmation is received (for example disable an expiry timer, count passed vs. dropped for a certain TM flow, etc.).

Unicast transmission with FD (TC=1, TKB=1 and TS=1), the Tx NDMA keeps buffers and confirms the transmission. This flow can be used, for example, on TCP retransmit buffer where the transmit window uses dynamically allocated frame buffers. The frame will be released by SW when the other end acknowledges it or when the transmission window times out. TCP transmission timeout can be triggered from actual transmission to the line. Another usage is multiple transmission of the same frame without protecting it by multicast counters, and after successful completion recycling the frame only once. When TS=1 this flow results in 2-step 1588 flow where the frame buffers are kept.

For non-confirmed transmission with FD (TKB=1 and TC=0) there is no trigger telling SW when the buffers are no longer used (i.e. the frame has been transmitted). The buffers should eventually be released to prevent memory leakage by a higher layer protocol, such as TCP SW handshake. Using TKB enables keeping the frame for future retransmissions and releasing it after the peer acknowledged successful reception. The TKB bit gets echoed on confirmation responses.

20.4.1 Timestamping 1-step IEEE1588

For 1-step processing, the CTOP application is responsible for performing a partial calculation of the:

- *CorrectionField* $_{\text{IEEE1588}}$ based on the ingress timestamp information
- Optional 16b checksum calculation (based on RFC 768)

These fields are passed to the TX MAC hardware in a prepended header for final 1-step *CorrectionField* and protocol checksum updates. The application is also responsible for any other PTP fields as required by the PTP frame type, such as the *OriginTimestamp* field.

The CTOPs are used to generate a 16B TS header which will include the resulting calculations for the intermediate *CorrectionField* and the intermediate checksum field as required by the protocol.

Overall, the final correction field required for PTP is:

- Final Correction Field = Original Correction Field + residence time,
  where residence time = TX TS – RX TS
- Or, Final Correction Field = Original Correction Field + TX Timestamp – RX Timestamp.

The partial or intermediate correction field is calculated as: Original Correction Field - RX Timestamp.

- *The RX timestamp format used in the intermediate calculation by the CTOPs must be converted to TimeInterval* $_{\textit{IEEE1588}}$ *format prior to subtraction from original CorrectionField* $_{\textit{IEEE1588}}$ *from the PTP packet which is in TimeInterval format.*
- *TimeInterval* $_{\textit{IEEE1588}}$ *is Integer64 scaledNanoseconds. The scaledNanoseconds member is the time interval expressed in units of nanoseconds and multiplied by* 2^{+16}. *For example, 2.5 ns is expressed as* $0000\ 0000\ 0002\ 8000_{16}$.

The CTOPs are used to calculate an intermediate/partial *CorrectionField* value:

- Partial CF = Original CF – RX Timestamp.

The partial result is placed in the 16B header for 1-step, and will be added by HW in the Tx MAC to the sampled TX timestamp to get the final *CorrectionField* result for the packet. The field offset is inserted by the CTOPs in the 1-step header to indicate to the Tx MAC where to place the final *CorrectionField* result.

To protect against RTC wraparound, the CTOPs are required to check if the ingress RTC timestamp is greater than half of its overall allowed value, and will set a wraparound flag in the 1-step 16B header if true.

- For 5B ingress timestamp mode, set Wraparound when RTC seconds > 128 seconds.

The CTOPs will also determine if the protocol (such as UDP IPv6) requires a checksum update for final packet transmission after inserting the updated correction field information. If required, the CTOPs will calculate a partial checksum (based on RFC 768) of the frame while zeroing out the 64-bit *CorrectionField*. The resulting intermediate checksum is placed in the Partial Checksum field of the 16B header to be passed to the TX MAC. The CTOPs will also insert the checksum field offset into the 16B header to indicate to the TX MAC the location in the packet to place the final checksum result; a flag is set by the CTOPs to enable the final checksum processing in the TX MAC.

Overall, the 1-step header format pre-pended by the CTOPs for the egress MAC interface is as follows:

Table 20-1: TX MAC 1-step header format

NAME	FORMAT	POSITION		DESCRIPTION
		BITS #	BYTE	
PARTIAL_CF	TimeInterval 64b	[63:0]	7-0	Intermediate Correction Field
CF_OFFSET	Byte index 14b	[77:64]	9-8	Offset of Correction Field inside the PTP frame
Reserved	2b	[79:78]		
CS_OFFSET	Byte index 14b	[93:80]	11-10	Offset of the checksum location inside the PTP frame
Reserved	2b	[95:94]		
PARTIAL_CS	Checksum 16b	[111:96]	13-12	Intermediate Checksum Field
Reserved		[119:112]	14	Unused
FLAGS	7 Bit flags	[127:120]	15	Bit 0 – Dynamic Step selection: '1' = 1-step Bit 1 – Checksum injection required. Bit 2 – Wraparound condition adjustment Bit 3 – Correction field start from odd byte. Bit 7:4 – Reserved.

20.4.1.1 Timestamping MAC Support 1-step IEEE1588

The 16B header contains a flag that indicates the timestamp type (1-step or 2-step), which is used in conjunction with the interface configuration (disabled, 1-step, 2-step, dynamic), to establish the timestamp mode for the egress MAC.

The RTC is sampled at the start of the packet transmission for packets requiring timestamp support. If the 1-step flag in the header is enabled while in dynamic mode, or if the port mode is 1-step, the sampled RTC value is translated in HW to TimeInterval format as defined for IEEE1588. The HW checks for wraparound of the RTC using the header bit/flag (Bit -2) and compensates accordingly if the sampled TX timestamp is less than half its overall allowed value (with flag set). The adjusted RTC TimeInterval value is added to the intermediate correction field provided in the original 16B header, and the result is inserted in the packet location indicated by the header’s offset location field. Dedicated HW engines perform the on-the-fly *CorrectionField* (and checksum) updates as required by the packet and indicated by the header.

The header checksum flag will indicate if the packet being transmitted requires an updated checksum in the protocol, such as for the UDP IPv6 encapsulation for PTP. The checksum is calculated after the new field (i.e. *CorrectionField*) is updated so the transmitted packet will have correct checksum. The checksum using the final correction field is calculated and added to the intermediate checksum that was calculated by the CTOPs and indicted in the 16B header. The adjusted value is inserted in the packet in the location indicated by the checksum offset field of the header.

The TX MAC will also calculate the correct MAC level CRC32 for the entire PTP frame after the all new fields are updated (i.e. correct Ethernet CRC).

20.4.2 Transmit Timestamping 2-step IEEE1588

Each egress MAC includes per port RTC timestamp logic which samples the RTC value for PTP transmitted packets and generates a message frame to the CTOPs containing the RTC and part of original PTP packet header (first 256B buffer).

The sampled RTC value is 40b LSB which are comprised of the 8b LSB of the RTC seconds counter and 32 bits of the nanoseconds counter.

20.4.2.1 Triggering Egress Timestamp

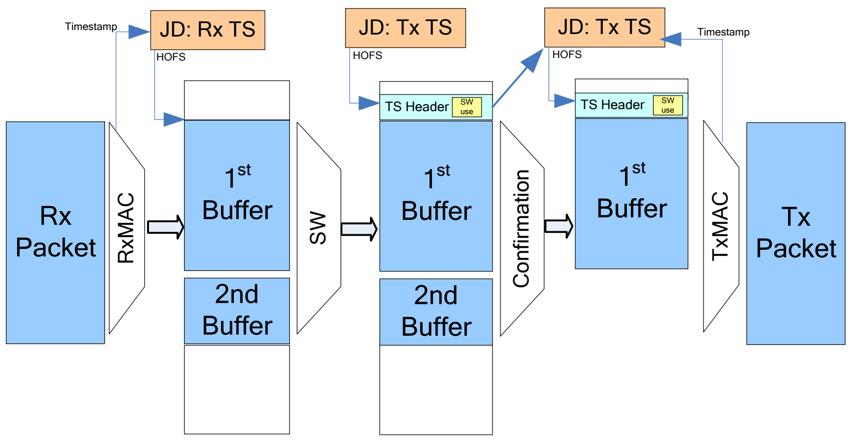
For a PTP transmitted packet, the CTOP attaches a 16-byte TS header to the beginning of the frame. This header is used to generate feedback confirmation packet Job Descriptor. The header is stripped at transmission.

The transmit MAC identifies the PTP marked packet timestamp request bit, removes the 16-byte TS header from the beginning of the frame, transmits the frame and samples the RTC. Then the MAC generates *Timestamp info* to the CTOPs containing the 5-bytes of the egress timestamp. The egress Timestamp info is copied to JD of the Confirmation feedback packet. The Confirmation feedback packet is truncated to just first buffer.

The CTOP application is required to process ingress and egress timestamps in order to calculate the Residence Time Interval. The application can store any required information in the first packet buffer before HOFS. The information usually required is the Receive timestamp and PTP flow identification.

The CTOPs process the Confirmation feedback packet and calculate the difference between the ingress and egress timestamps and keep/store this information for later use to add the difference in the *Correctionfield* in the follow-up packet. Since the follow-up packet uses the same flow, it is classified and identified by the CTOPs same as the original (i.e. Sync, Pdelay_Resp) PTP packet. The CTOPs retrieve the stored time difference returned from the MAC and use it for the follow-up packet by adding the value to the *Correctionfield*$_{\text{IEEE1588}}$. The follow-up message is then sent to the TM for transmission out of the interface; the follow-up packet does not require any additional TS headers or egress MAC processing.

Figure 20-2. Time stamped packet flow



20.4.2.2 Timestamp Message Format

The tables below illustrate a 16-byte PTP header for a frame that requires a timestamp by ports and Confirmation frame generation:

Table 20-2. Header for RTC time stamping (2-step format)

NAME	BITS #	DESCRIPTION
BD	b31:0	Buffer Descriptor points to Confirmation buffer for 2-step time stamping. Must be same as JD[BD]. Used to create confirmation packet JD.
BID	b41:32	Budget ID[7:0] for Confirmation frame JD/FD. Must be same as JD[BD]. Used to create confirmation packet JD.
Reserved	b43:42	
COS	b45:44	COS[1:0] of Confirmation frame JD/FD.
Reserved	b47:46	
HOFS	b55:48	Header Offset of Confirmation frame JD/FD. This header resides after the HOFS.
FB	b63:56	Free Bytes of Confirmation frame JD/FD. Free bytes are left at the end of the frame header data block.
SW Reserved	b119:64	Reserved for SW usage: PTP flow ID and Rx TS.
TS mode flag	b120	Per frame dynamic step selection: '0' – 2-step.
SW Reserved	b127:121	

Table 20-3. Confirmation TS frame JD fields

NAME	DESCRIPTION
BD [31:0]	Buffer Descriptor points to Confirmation buffer for 2-step time stamping. Copied from TS header.
BID[9:0]	Budget ID[7:0] for Confirmation frame JD/FD. Copied from TS header.
COS[1:0]	COS[1:0] of Confirmation frame JD/FD. Copied from TS header.
HOFS[7:0]	Header Offset of Confirmation frame JD/FD. Copied from TS header.
FB[7:0]	Free Bytes of Confirmation frame JD/FD. Free bytes are left at the end of the frame header data block. Copied from TS header.
EGRESS_TS_NS[31:0]	Timestamp: Nanoseconds
EGRESS_TS_SEC[7:0]	Timestamp: Seconds

Once the sampling is complete, the first frame buffer with updated header is sent back to the CTOPs on a feedback path through the Feedback Stack when FIFO entry is made available to the TM for another PTP packet (i.e. a credit is returned to TM).

The confirmation frame containing the egress timestamp is sent via PMU to CTOPs. The confirmation egress TS frame is a single buffer with data size of 256B-FB-HOFS.

The CTOPs application uses the information in the returned header to process and generate a follow-up frame as required for 2-step applications.

The egress timestamp is stored in a FIFO before being sent via the PMU. The TM special designated queue is used to protect this FIFO from overflow. The protection is implemented with a credit based mechanism allocating a credit for each PTP packet/message.

20.4.2.3 Egress Timestamp Confirmation Feedback Delivery Control

Each interface side implements an egress timestamp FIFO for confirmation packet descriptors.

In order to avoid the egress timestamp FIFO’s potential overflow, the TM must ensure that no more than eight IEEE1588 frames which require egress time stamping will be in transit to the FIFO.

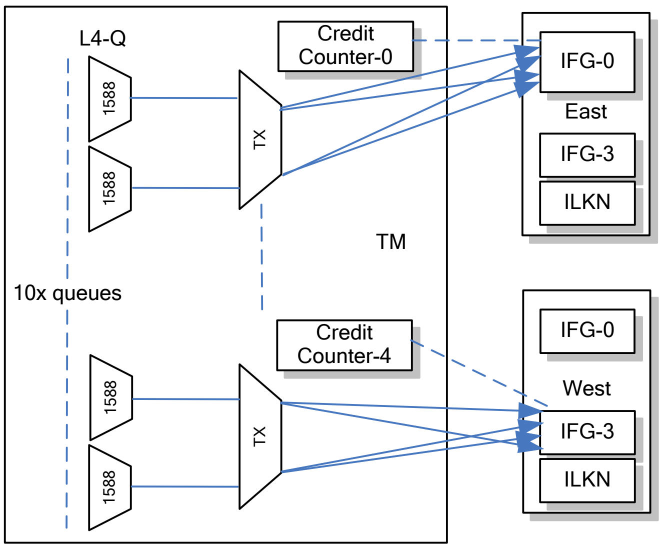
The FIFO overflow protection is done by counting the number of IEEE1588 packets scheduled to transmit that did not generate an egress timestamp feedback confirmation message to the PMU.

The overflow protection counters are implemented in the TM. This requires that 2-step IEEE1588 egress packets be transmitted via the TM dedicated queues.

Each TM engine implements a five FIFO overflow protection credit counters. Each credit-counter can service two TM-L4 IEEE1588 queues. Each credit-counter can be 1:1 associated with any one of five Interface Group on east or west side (per side four IFEs or ILKN).

The credit counter counts the number of packets that are in transit from the TM 1588 queues to the network interface MAC and were not transmitted to the network (i.e. did not generate an egress timestamp feedback message to the PMU).

Figure 20-3. Credit counter association



The TM (TM_QC) credit counter has a configurable threshold that stalls the TM 1588 queues scheduling until credits are recycled

> IEEE_1588_FLOW_0 <CREDITS> [4:0] – Threshold of credits of IEEE1588 frames scheduled and not recycled. This value must be set to 0x8.

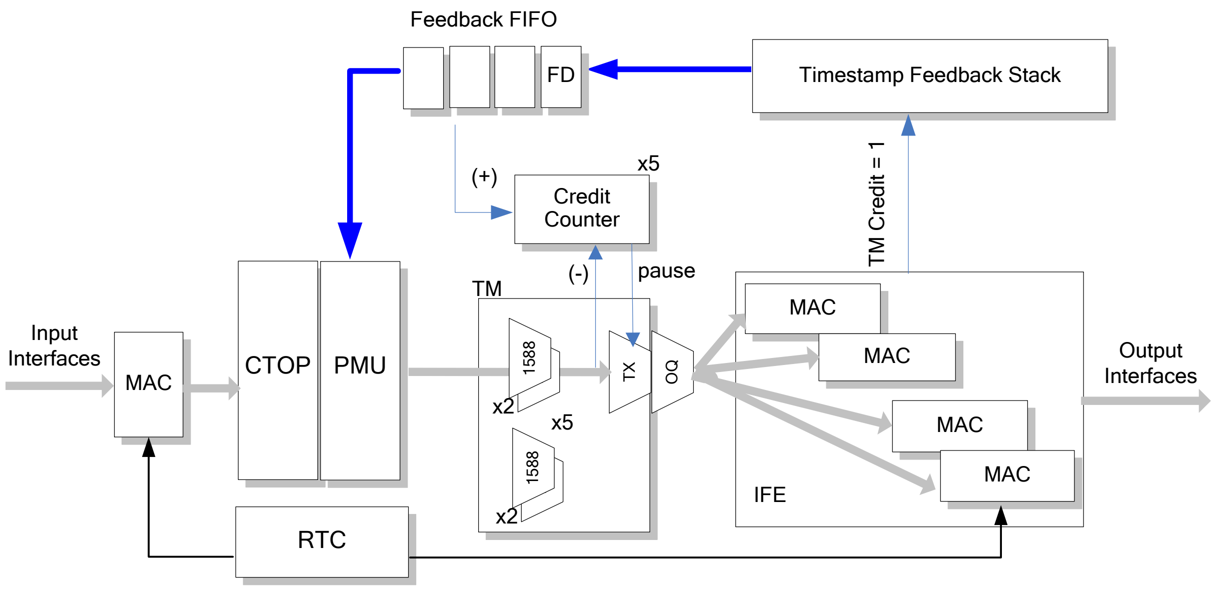
The credit counter increments when a packet is scheduled for transmission out of the 1588 queues and it decrements when an egress timestamp confirmation message is pushed to PMU.

The message delivery control flow (Figure 20-4. Credit counter mechanism):

1. PTP packet is sent to a TM 1588 dedicated queue for credit control.
2. The TM schedules a PTP packet from TM 1588 queues only when the credit counter allows. Upon scheduling, a credit unit is consumed.
3. The ingress timestamp is conveyed in a PTP confirmation message (JD is constructed based on PTP TS packet header).
4. The transmit MAC triggers timestamp confirmation FD queuing in the feedback stack.

5. When a PTP confirmation is generated and is pushed to PMU, the TM 1588 dedicated queue's credit counter recycles the credit.

Figure 20-4. Credit counter mechanism



The IEEE1588 SW identifies an egress timestamp message session by PTP Flow ID field written to TS header SW reserved 8B of the first buffer.

The ten L4-queues used for PTP are configurable per TM.

The credit counter assignment to IF group or ILKN is configurable. An IF group can be served by a single TM credit counter.

20.4.2.4 Redirecting 2-step Egress PTP Packet to Transmit Port

Since each TM is allocated up to ten L4 queues for 2-step IEEE1588 packets, this requires special management to redirect the PTP packet towards the transmit MAC or loopback interface.

The TM output queue that is mapped to transmit port is selected from 4K entry Packet Switch ID table. There are several modes to index the table as selected by JD QM mode. The QM mode for 2-step IEEE1588 is 0b010.

The JD-QM[2:0] field controls the TM Output Queue Mapping:

- QM[2:0] = 0x010 – Explicit JD-PSID selection.
  In this mode, SW puts an explicit PSID selection (EXPSID[12:0]) in the 16-bit INFO that explicitly selects an entry from the PSID table (from the portion typically used for L1 entities).

20.5 RTC Hardware Implementation

20.5.1 Real Time Counter for Timestamping

The system's Real Time Clock holding the local time required for timestamp can be updated by the control plane SW (CTOP or via PCIe), by the Slave MDC hardware operation, or by the 1PPS input. Allowing hardware configuration of the RTC from the MDC slave interface enables synchronized operation over several network processors and line cards in parallel.

The RTC counter is 64 bits which are separated into two 32-bit counters. The MSB 32 bits of the RTC are called the seconds counter and the LSB 32 bits of the RTC are called the nanoseconds counter. The two counters operate as a single 64-bit counter using the nanoseconds counter wraparound carry indication as an increment control for the seconds counter. Another 16-bit counter is used to save the RTC counter time residue with a resolution of 2^{-16} of a nanosecond. This 16-bit counter is advanced with the nanoseconds counter and saves the time increment residue (which allows for sub 25 ppm accuracy for any clock source driving the RTC counter). When working with an external reference clock, the residual portion is increased to 32-bits for increased accuracy (allowing 1 part per billion accuracy for any external reference clock).

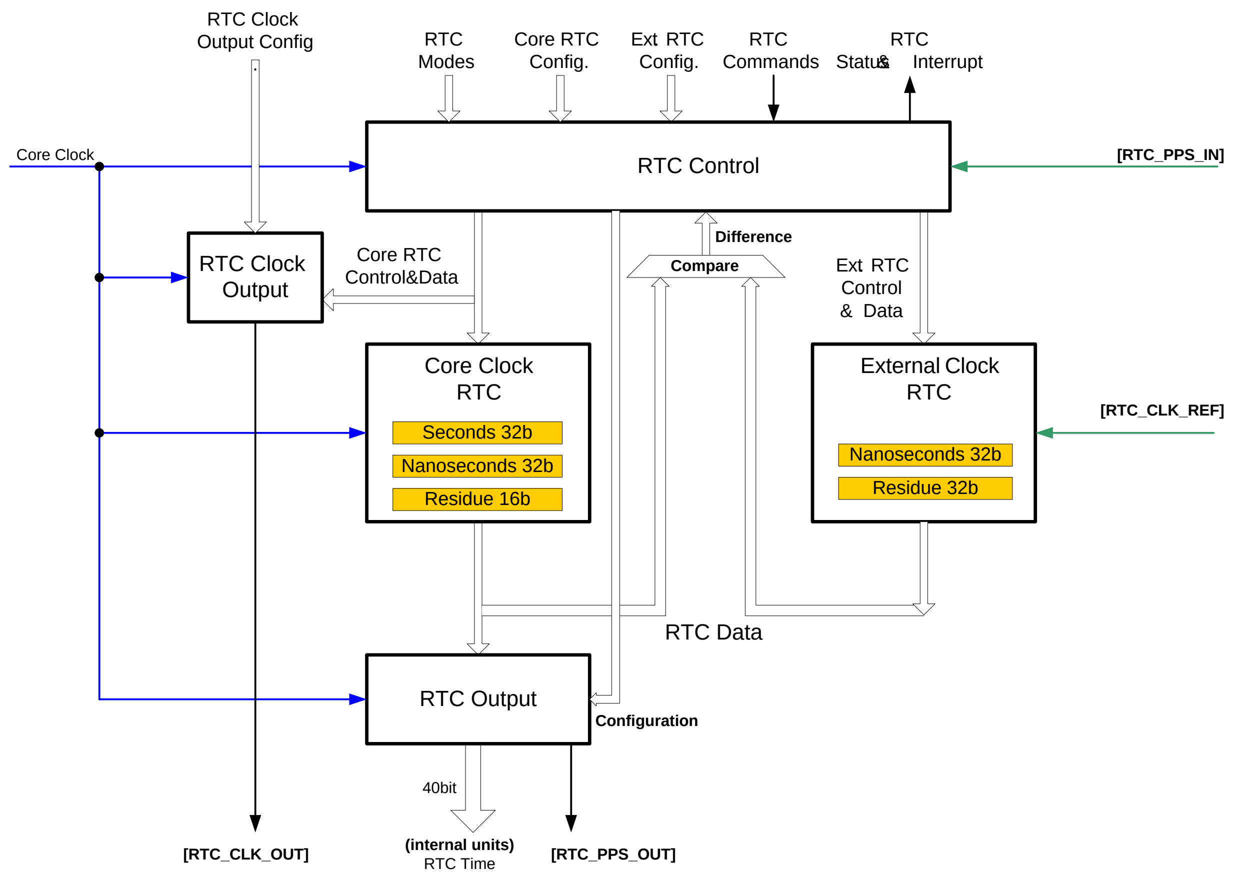
The LSB 32 bits nanoseconds counter wraparound to 0 can be configured to operate in any value between 1 to 4,294,967,295 (which is 2^{32}- 1 nanoseconds). The default value is 999,999,999 which represents a value of 1 second minus 1 nanosecond.

The RTC is advanced by the device's core clock. The time increment is configured by the control plane SW and is equal to the core clock cycle time with a resolution of up to 2^{-16} of a nanosecond. Each core clock, the counter is advanced by using the configured core clock time and the 16-bit residue counter.

Using an external clock source for the RTC is achieved indirectly by using a separate counter system that is running with the core clock and enabled by an external source clock trigger mechanism. The external clock source counter is similar to the RTC clock system but without the 32-bit seconds counter. The time increment is configured to the external clock cycle time. The seconds counter wraparound is the same as in the RTC counter system. The RTC counter system receives a periodic compare command after a configured number of external clock events and compares the nanoseconds time between its counter and the external clock counter. The time difference between the counters will be adjusted by the RTC clock system just like a usual time adjustment done by the local clock update (see Figure 20-5 below).

- *The 16 MSB of the RTC seconds counter (as defined by the IEEE 1588 standard) are maintained by the control plane software.*

Figure 20-5. RTC



20.5.1.1 Control Plane SW and Slave MDC Timestamp Read

The RTC contains local clock sampling logic that allows timestamp sampling into a buffer. The sampling to a 64 bit buffer allows atomic read of the timestamp related to an event that can be initiated by the control plane SW or external control through the slave MDC/MDI serial interface.

20.5.1.2 CTOPs Timestamp Read

There more than one method for CTOPs to get the RTC value with tradeoffs between precision accuracy and data width. All CTOPs clusters contain a local copy of the RTC.

- Per CTOP cluster RTC accessible 32-b value. The value is accessed as an AUX register by the LR instruction. This option has the best accuracy and has exactly the same fixed access delay for all CTOPs.
- A dedicated GETRTC accelerator instruction, which copies by atomic action the 64-bit RTC (sec and ns) to a specified CMEM address (in parallel, the 32-bits ns is written to the destination register). This instruction has lower accuracy as compared to LR for Aux.
- Either CTOPs or external PCIe CPU can always read RTC high and low 32b values of each cluster by memory mapped registers.

20.5.1.3 Local Clock Update

The update mechanism allows local clock synchronization to a remote node that serves as a time domain master. The update can be performed by the control plane SW as part of the IEEE protocol implementation, by the Slave MDC input or by the 1PPS input.

The local clock update mechanism is based on RTC configuration registers that can be written (or read) through a regular 32b register access operation. The register accesses different sections of the 64 bits RTC counters.

There are three RTC registers used to write/update the local clock, which can be accessed for read/write by the control plane SW or Slave MDC:

- 32-bit data register containing the data for read/write/update of the RTC seconds counter.
  See RTC_CMD_SECONDS Register [0x182].
- 32-bit data register containing the data for read/write/update of the RTC nanoseconds counter.
  See RTC_CMD_NSEC Register [0x181].
- 8-bit control register (see RTC_CMD Register [0x180]) that determines the operation initiated by the Control plane SW or Slave MDC.
  Operations include:
  - Load – sample the RTC counters.
  - Set – write configured time to the RTC counters.
  - Increment – increment the RTC counters by the configured time.
  - Decrement – decrement the RTC counters by the configured time.
  - Pause – suspend the RTC counters with the configured time.
  - 1 PPS – activation of 1PPS event through Host command.
  - Set Seconds – write configured time to the RTC seconds counter only.

20.5.1.3.1 RTC_CMD Register [0x180]

RTC block (ID = 0x483).

Table 20-4. RTC_CMD register

EZcp API: EZapiChannel_ConfigCmd_ExecuteRTCOperation

NAME	*BITS#*	*DESCRIPTION*	*INIT VALUE*
RTC_CMD_LD	b0 W	RTC Load Command, used to load the RTC time counter value into the RTC TIME registers (self cleared). A self cleared command bit, RTC counter time data will be read into the RTC TIME registers at the next clock (of issuing the command). RTC status will be ready immediately.	0x0
RTC_CMD_SET	b1 W	RTC Set Command, used to set the RTC time counter to a specific time value (self cleared). A self cleared command bit, RTC time data will be written to the RTC counter. The time parameters resides in the RTC Command Data registers. RTC status will be not ready until the command is finished.	0x0
RTC_CMD_INC	b2 W	RTC Increment Command, used to advance the RTC time counter by a specific time value (self cleared). A self cleared command bit, RTC time will be advanced. The advance time parameters resides in the RTC Command Data registers, RTC status will be not ready until the command is finished.	0x0
RTC_CMD_DEC	b3 W	RTC Decrement Command, used to reduce the RTC time counter by a specific time value (self cleared). A self cleared command bit, RTC time will be reduced. The reduced time parameters resides in the RTC Command Data registers, RTC status will be not ready until the command is finished.	0x0
RTC_CMD_PAUSE	b4 W	RTC Pause Command, used to delay the RTC time counter advancement by a specific time value (self cleared). A self cleared command bit, RTC time will be paused or advanced slower. The pause time parameters resides in the RTC Command Data registers, RTC status will be not ready until the command is finished. When the Pause parameter in the Data register is 0, then the command is to pause. When Data is not 0, then it advances slower than regular.	0x0
RTC_CMD_1PPS	b5 W	RTC 1PPS Command, used to synchronize the RTC Seconds Timer with a 1PPS event (self cleared). A self cleared command bit, RTC time will advance its seconds counter by 1 second and set the nanoseconds counter to a pre-defined value. This command is affective only when 1PPS mode is enabled.	0x0
RTC_CMD_SET_SEC	b6 W	RTC Set Seconds Command, used to set only the RTC Seconds counter to a specific time value (self cleared). A self cleared command bit, RTC Seconds data will be written to the RTC Seconds counter. The time parameters resides in the RTC Command Data registers. RTC operation is not affected, status will be ready and the RTC nanoseconds counter will continue its operation. Limitation - The Set Seconds command cannot be issued when a nanoseconds counter wraparound happens.	0x0
Reserved	b31:7 NA		0x0

20.5.1.3.2 RTC_CMD_NSEC Register [0x181]

Table 20-5. RTC_CMD_NSEC register

EZcp API: EZapiChannel_ConfigCmd_ExecuteRTCOperation

NAME	BITS#	DESCRIPTION	INIT VALUE
DATA	b31:0 W	RTC Command Nano Seconds Data Holds the RTC Set command nanoseconds data. Holds the RTC Increment/Decrement/Pause commands change data (in nanoseconds).	0x0

20.5.1.3.3 RTC_CMD_SECONDS Register [0x182]

Table 20-6. RTC_CMD_SECONDS register

EZcp API: EZapiChannel_ConfigCmd_ExecuteRTCOperation

NAME	BITS#	DESCRIPTION	INIT VALUE
DATA	b31:0 W	RTC Command Seconds Data Holds the RTC Set command seconds data. Holds the RTC Increment/Decrement/Pause commands change parameters (Maximum Inc/Dec & Minimum Pause).	0x0

20.5.1.4 RTC Synchronization Modes

The RTC synchronization to an external clock can be achieved via three input interfaces:

- RTC_REF_CLK – This is a free running reference clock. The clock adjusts the NPS RTC. The adjustment can be a singular event (jump forward/backward in time) or a gradual adjustment.
- 1-PPS – Pulse synchronized with RTC_REF_CLK signaling the adjustment of RTC seconds count and zeros the nanoseconds counter. The adjustment can be setting the seconds counter to a pre-set value or seconds auto increment. The adjustment timing is executed in synchronization with RTC_REF_CLK.
- Slave MDC/MDIO – Interface enables the external device to read/write the RTC and trigger a 1-PPS event. The 1-PPS trigger is a single MDIO transaction that needs to be loosely synchronized with the RTC_REF_CLK cycle.

The Gradual update mode should be enabled only with S/W Update operations (e.g. not when 1PPS is enabled).

When applying a S/W increment/decrement/pause update command (RTC_CMDINC / RTC_CMD_DEC / RTC_CMD_PAUSE), the update command should be applied only when the previous update command has been (gradually) completed.

In non-gradual update mode, this is being done by examining the RTC ready status RTC_STS[READY].

When in gradual update mode, the S/W must calculate the estimated finish time of the update command before issuing the next update command.

This estimation can be performed using the Gradual Mode configurations RTC_GRDL_UPDATE and RTC_GRDL_INTERVAL.

The estimation can be calculated by this simple equation: Update finish estimation time = RTC_GRDL_INTERVAL * ceil (RTC_CMD_NSEC / RTC_GRDL_UPDATE)

RTC_GRDL_UPDATE:	Maximum allowed update within a Gradual Mode Interval – nano-seconds.
RTC_GRDL_INTERVAL:	The interval time between each of the mini updates allowed in the Gradual mode – nano-seconds.

20.5.1.4.1 External Reference Mode

There are two RTC counters:

- Core RTC – Advances by Core clock with a configurable increment step **RTC_CORE_ADV[CFG_NSEC_ADV], RTC_CORE_RES[**CFG_RESIDUE].
- Reference RTC – Advanced by **RTC_CLK_REF** clock with a configurable increment step **RTC_EXTC_ADV[CFG_NSEC_ADV], RTC_EXTC_RES[**CFG_RESIDUE].

The Core RTC can be synchronized to RTC_CLK_REF input (Reference RTC). The synchronization event adjusts the Core RTC to match the Reference RTC.

There are two synchronization sub-modes:

- RTC advanced in each Core clock – REF-Synch mode.
  Enabled by **RTC_MAIN_CFG[EXT_CLK_MODE]**
- RTC advanced in External Clock steps – REF-Advance mode.
  Enabled by **RTC_MAIN_CFG[ADV_MODE_EXT_CLK]**

20.5.1.4.2 1-PPS Synchronization

The RTC can be synchronized to 1-PPS pulse on RTC_PPS_IN. This mode is enabled via **RTC_MAIN_CFG[1PPS_MODE]**. The mode enables the RTC counters seconds advance/set and nanoseconds wrap events by 1-PPS signal.

The RTC 1-PPS trigger can be used to set the RTC seconds/nanoseconds to a configured value when **RTC_MAIN_CFG[1PPS_SET_EN]** mode is enabled. The configured value should be written to **RTC_CMD_SECONDS[DATA]**, RTC_CMD_NSEC[DATA]. The exact time of the RTC update is synchronized by the RTC_CLK_REF rising edge.

For a single time event update, **RTC_MAIN_CFG[1PPS_SET_EN]** should be disabled after a 1-PPS pulse.

20.5.1.4.3 Slave MDC/MDIO RTC Adjustment

The external logic can control the NPS Core RTC:

- The RTC seconds/nanoseconds counter can be loaded.
- The RTC 40b can be read.
- The RTC 1-PPS can be triggered to load the seconds counter and zero the nanoseconds.
- The MDC/MDIO registers allow indirect access to RTC:

REGISTER ADDRESS	*OPERATION*
0x0	Write Address 31:16
0x1	Write Address 15:0
0x2	Write Data 31:16
0x3	Write Data 15:0
0x4	Read Address 31:16
0x5	Read Address 15:0
0x6	Read Data 31:16
0x7	Read Data 15:0
0x8	Trigger RTC commands: Data[0] – Latch RTC 64b for read Data[1] – 1PPS trigger

To access the RTC for setting the **RTC_CMD_SECONDS[DATA]**:

1. Write 0xFA00 to MDIO register 0x0 (Address [31:16]).
2. Write 0x2082 to MDIO register 0x1 (Address [15:0]).
3. Write Seconds[31:16] to MDIO register 0x2 (Data [31:16]).
4. Write Seconds[15:0] to MDIO register 0x2 (Data [15:0]).

The last data write (4) triggers the **RTC_CMD_SECONDS[DATA]** load.

- *Note: The Address[15:0] of RTC nanoseconds is 0x4781.*

When the seconds counter needs a periodical update each second, the device can execute a single write of Seconds[15:0] only (given that the other MDIO registers were unchanged).

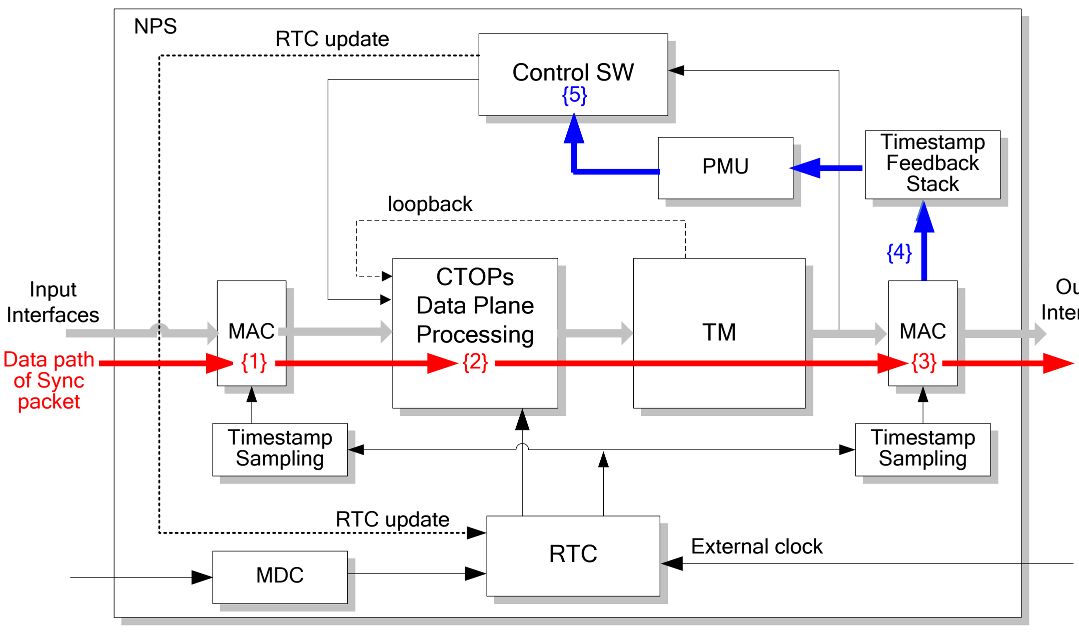
When the 1-PPS signal is not used, the device can trigger a 1-PPS event by MDIO write to register 0x8 with bit[1] set.

20.6 Basic 2-step Application Data Flow Example

An end-to-end 2-step transparent clock PTP application measures the local node delay which is added to a special correction field in the follow-up message. The delay calculation uses the timestamp data for both the ingress and the egress stages of the node that forwards the follow-up message to its destination.

The following application example describes a walkthrough of the data flow in the NPS as shown in the figure below.

Figure 20-6. Basic end-to-end application data flow example – sync packet data path



The end-to-end transparent clock node application needs to identify the Master to Slave Sync and follow-up messages to maintain timestamp tracking of the sync message and add the time difference between the ingress to egress timestamps to the follow-up message.

{1} An ingress timestamp attached to a Sync packet message when it is forwarded from the link input MAC to the CTOP engines.

{2} The CTOP data plane application identifies the Sync event message, its PTP flow and PTP slave destination node. The CTOPs look for an index in a hash table representing the PTP message flows.

If the session does not have a PTP flow ID in the hash table, the CTOPs will request an index from the index queue and will learn the session into the hash table while sending the packet to the TM. The PTP ID will be saved (as a result) in the hash table representing the flow session based on the packet’s 5-tupple data.

The CTOPs will write the PTP flow ID (from the PTP hash table) and Rx timestamp to the frame’s TS header. At this stage, the CTOPs can handle multicast Sync frames and build the Sync frames that need to be sent to the various IP destinations. In such a case, each Sync packet will have a different flow ID (and therefore a different PTP ID header) and could be handled differently by the control plane SW (keeps its own ingress to egress timestamp difference).

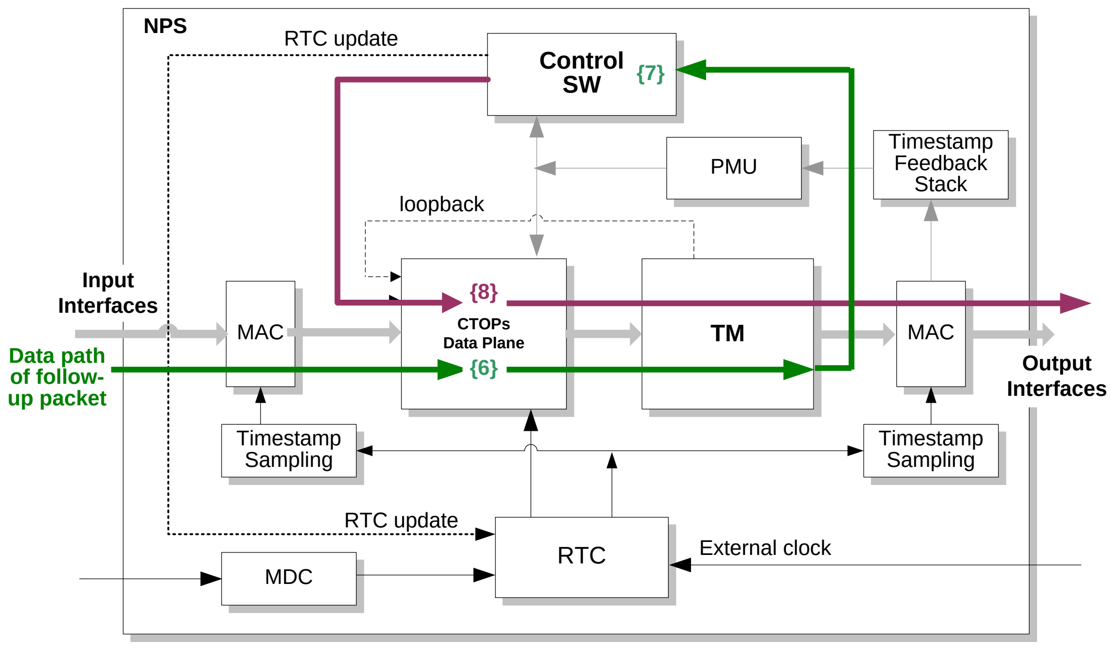
The CTOPs tag the packet as a packet that requires timestamp sampling at the link output MAC and forward the frame to the TM.

{3} The packet’s RTC timestamp is sampled when the link output MAC sends the frame.

{4} The frame header (which includes the input timestamp and the PTP flow ID), the output timestamp and the output port identification are then forwarded to the control plane SW via the Feedback packet.

{5} The control plane SW identifies the PTP flow (using the PTP ID in the header) and the Master to Slave connection, and calculates the time difference between the egress and ingress timestamp and keeps it for future use (when the follow-up message arrives).

Figure 20-7. Basic end-to-end application data flow example – follow-up packet data path



{6} When a Sync follow-up packet arrives, it is identified and classified by the CTOPs and then forwarded to the control plane SW through the TM block. Since the follow-up packet uses the same flow as the Sync packet it will get the same PTP ID in the TS header data as the Sync packet.

{7} The control plane SW finds the corresponding Sync packet's egress to ingress time difference and adds it to the time written in the correction field of the follow-up message.

When a follow-up message arrives to the control plane SW and egress timestamp feedback is missing (i.e. it is still in the TM output queue due to congestion, flow control, etc.), the control plane SW will send the follow-up message to the same TM output queue that the corresponding Sync message was sent and set a Transmit Confirmation request (TC) bit in the frame descriptor. In such a case, the frame will be looped back by the MAC and this will happen after the corresponding Sync message was sent by the MAC to the link.

{8} The follow-up packet is sent from the control plane SW to the CTOPs which handle the lower level protocol headers and link output headers.

The CTOPs send the follow-up message to the TM and output queue.

The follow-up message is then transmitted to the link as a normal packet without requiring output timestamping.

In case the original Sync packet uses a “1-step clock” scheme, it is possible for the CTOPs to change the packet to a “2-step clock” scheme and notify the control plane SW accordingly. In this case, the control plane SW will initiate the Sync follow-up message and write the ingress to egress time difference in the correction field of the follow-up message.

The CTOPs need to alter the original Sync packet data payload and change the protocol header data to indicate a “2-step clock” Sync packet.

20.7 Data Plane Only Processing

An end-to-end transparent clock PTP application implemented by the CTOPs data plane processing is used mainly when the number of PTP event messages traversing the NPS is very large and the control plane SW computational power is insufficient to handle the application load.

Most of the data flow and actions by the different NPS blocks are the same as in the previous example (Basic 2-step Application Data Flow Example), the only change is in handling of the egress timestamp which is done by the CTOPs data plane processing.

The following application example describes a walkthrough of the data flow in the NPS as shown in Figure 20-6.

Steps {1} through {4} are the same as Figure 20-6 in Section 20.6 above.

The packet's timestamp is sampled when the link output MAC sends the frame. The frame header (which includes the input timestamp and the PTP flow ID), the output timestamp and the output port identification are then forwarded to the CTOPs data plane processing.

The CTOPs data plane (instead of the control plane SW) identify the PTP flow (using the ID in the header) and the Master to Slave connection, and calculate the time difference between the egress and ingress timestamp and keep it for further use (when the follow-up message arrives).

When a Sync follow-up packet arrives, it is identified and classified by the CTOPs. Since the follow-up packet uses same PTP flow as the Sync packet it will get the same PTP ID in the header data as the Sync packet.

The CTOPs find the corresponding Sync packet's egress to ingress time difference and add it to the time written in the correction field of the follow-up message.

The CTOPs send the follow-up message to the TM and output queue.

The follow-up message is then transmitted to the link as a normal packet without requiring output timestamping.

When a follow-up message arrives and egress timestamp feedback is missing (i.e. it is still in the TM output queue due to congestion, flow control, etc.), the CTOPs will send the follow-up message to the TM with low priority and loopback the Sync packet to the CTOPs (in the hope that its related Sync message feedback will arrive). This loopback should be tagged with a TTL number to limit the number of loops for such a packet.

20.8 Basic End-to-End Application Data Flow Example in a Line Card System

The end-to-end transparent clock application may be implemented in a distributed system such as core routers, which are typically designed as a chassis holding several line cards. The ingress timestamp and the egress timestamp reside on different chips or different line cards. Nevertheless, the ingress to egress delay should represent the entire hardware delay regardless of the number of chips or line cards it traverses.

In a line card system, all the NPS devices' local clocks are kept in synchronization with the line card that is defined as the local clock master and maintains the local clock synchronization with the network time domain master.

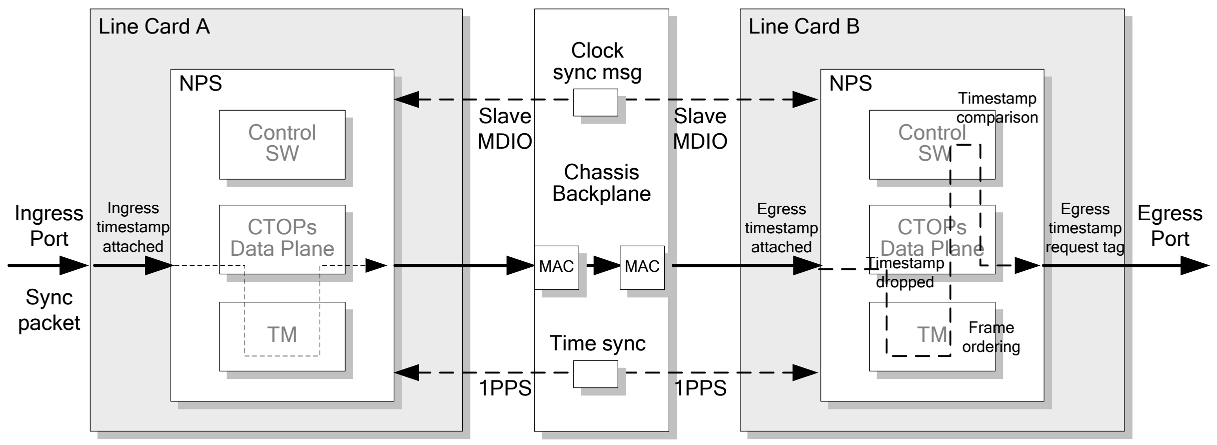
Synchronization between the line cards is accomplished using the NPS device's 1PPS input or the Slave MDC/MDIO interface.

The recommended method is to use the 1PPS event generated periodically by the master line card and feed the slave line cards with 1PPS synchronization event. When a 1PPS event is received by the NPS device on a slave line card, it is used to: (a) update time-of-date, or (b) simply to zero the nanoseconds counter and increment the seconds counter, or (c) generate an automatic update command (increment/ decrement) with the appropriate time delta between the local NPS device and the 1PPS event.

With the synchronization method using the Slave MDIO interface, the master line card periodically sends RTC sample command messages through the Slave MDIO interface to the NPS devices on all the other line cards. Since the transmission time of the Slave MDIO channel can be made fixed and deterministic for all devices, all the NPS devices (including the one that resides on the master line card) sample their local RTC time at exactly the same time. The control plane SW on the master line card then sends the master line card's RTC sampled data to the Control CPUs on all the slave line cards. The control plane SW on the slave line cards compare their local RTC sampled time to the master line card's RTC sampled time they received and will update/adjust their local RTC according to the difference.

Alternatively synchronization of the RTC on all line cards can be achieved by using an external RTC clock source which will be driven from the same clock master to all line cards. In this mode, the RTC needs to be adjusted only at reset time and the local RTC will follow the external dedicated clock sources from this point on.

Figure 20-8. Basic end-to-end application data flow example in a chassis based system



Operation of the end-to-end transparent clock application is split between the two NPS devices which reside on different line cards. The following application description assumes that Line Card A is used as the ingress port for the PTP messages while Line Card B is used as the egress port.

For a Sync packet received at the link input of Line Card A, the timestamp is attached. The CTOPs on Line Card A identify the Sync packet destination (on Line Card B), attach an PTP flow ID and 16-byte TS header to the frame and send the packet to Line Card B with Line Card A ingress timestamp (in the 16-byte header) but without a timestamp feedback request indication. The Sync packet is passed via the backplane MAC of Line Card A to the backplane input MAC of Line Card B.

At Line Card B, if the MAC connected to the backplane is configured not to add an ingress timestamp, the packet is forwarded to the CTOP as is.

The Sync packet is identified as a packet that was received from Line Card A and includes the Line Card A link ingress timestamp and PTP flow ID.

The CTOPs tag the packet with an egress timestamp sampling request indication and forward the frame to the link output MAC.

The packet’s timestamp is sampled at the link output MAC which also sends the timestamp feedback with the packet’s TS header (PTP flow ID and Line Card A input timestamp) to the control plane on Line Card B.

The control plane SW on Line Card B identifies the PTP flow and Master to Slave connection and calculates the time difference between the Line Card A ingress timestamp and Line Card B egress timestamp and saves it for future use (when the follow-up message arrives).

When the Sync follow-up message packet arrives (to Line Card A), it is forwarded to Line Card B. At Line Card B, the CTOPs forward the follow-up message to the control plane SW which finds the correlated Sync packet ingress to egress time difference and adds it to the time written in the correction field of the follow-up packet.

The follow-up message is then sent by the control plane SW of Line Card B and transmitted to the link as a normal packet without requiring output timestamping.

21. General Interrupt Controller

An interrupt controller aggregates services that are required to signal interrupts and process interrupt acknowledges. In NPS-400, the general interrupt controller block is called GIM.

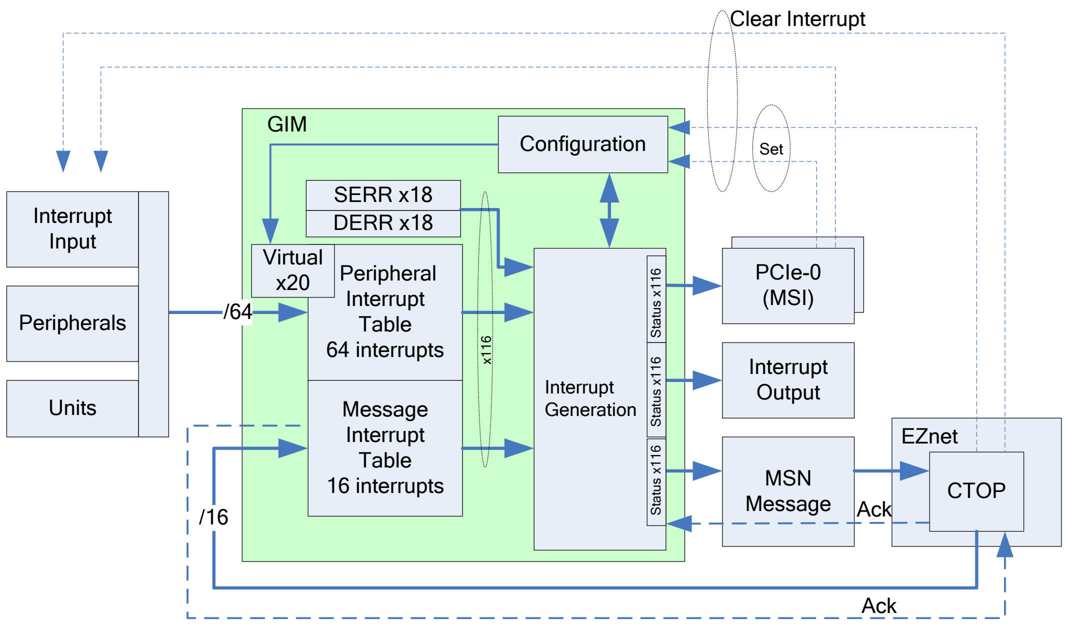
The interrupt sources are:

- Peripheral and Internal sources up to 64 interrupts:
  - Peripheral and internal units
  - Interrupt input
  - CTOPs/PCIe-Host via 20 GIM virtual interrupts, SW triggered
- CTOP message triggered 16 interrupts
- ECC interrupts for single (SERR) and double (DERR) errors.

The interrupt targets are:

- CTOPs (thread/core/cluster - unicast/multicast)
- PCIe interfaces (MSI)
- External interrupt output

Figure 21-1. GIM structure



The interrupt sources are aggregated into Interrupt Tables. There are 64 interrupts in the Peripheral Interrupt table and 16 interrupts in Message Interrupt table.

The Peripheral Interrupt Table serves all internal functional units and input signals.

The Message Interrupt table serves CTOP SW generated interrupts conveyed by messages.

The interrupts can be configured to target PCIe (PCIe-0 or PCIe-1), interrupt output or CTOP thread/core/cluster. The interrupt can target multiple CTOP targets when it is configured as multicast.

The interrupts targeting CTOPs are messaged via MSN EZnet messages. The message delivery is end-to-end protected with message acknowledgement. The Message SW triggered interrupts are also delivered via MSN messages.

The peripheral and internal unit interrupts are SW cleared by CTOPs or over PCIe by direct access to the unit's interrupt cause register and GIM cause register.

Out of the 64 peripheral interrupts, eight of the peripheral interrupts, range 63:44, are allocated as 20 Virtual Interrupts that can be triggered by SW access to the GIM configuration space. The Virtual Interrupts are accessible to CTOPs as well as the Host CPU via PCIe.

The NPS ECC soft error protection also generates Single (SERR) or Double (DERR) error interrupts that can be targeted to CTOPs or external CPU over PCIe/external interrupt output.

The SERR and DERR each contain a range of 18 interrupts. 16 of the interrupts are for CTOP clusters, bit per cluster, and two additional interrupts are for NPS data path SERR/DERR.

Each interrupt source supports configured functional modes:

- Level Sensitive – Generate interrupt while source is active. Memory-less since interrupt cause may disappear before it was serviced.
- Edge Sensitive – Set interrupt on cause state change (rising or falling). Interrupt request is registered until serviced or acknowledged. Stays set even when cause disappears.
- Clear Interrupt by CTOP Message Acknowledge (non-blocking) – The interrupt is cleared when interrupt message reaches the target CTOP. This mode effectively saves SW steps to clear the interrupt.
- Clear Interrupt by Service (blocking) – The interrupt stays set until it is cleared by SW.

MODE	***BLOCKING***	***NON-BLOCKING***
Level	Interrupt set while cause active. Needs to be SW serviced and cleared. May miss interrupt when cause disappears before message is generated.	Not useful. As auto clearing by message Ack occurs before service thus causing additional interrupts.
Edge	Interrupt set when cause transitions to active. Needs to be SW serviced and cleared. May miss cause transitions while interrupt is set.	Interrupt set when cause transitions to active. Auto-cleared by message acknowledge. Needs to be SW serviced. Low probability to miss cause transitions while interrupt is set.

The CTOP SW can trigger up to 16 message interrupts. A message must be acknowledged before the same interrupt cause can be triggered again by the same cluster. Thus, per cluster, up to 16 on-air (before Ack) interrupt messages can be triggered.

The interrupt to CTOP can specify one of two interrupt service routines (mapped to different bits in the CTOP IRQ register).

The GIM interrupt message generation is subject to priority arbitration as configured per interrupt source.

21.1 GIM Interrupt Table

The GIM maintains 64+16+18+18 interrupt entries in the table. The table configuration allows steering the interrupt source to target destination and setting functional modes.

The Interrupt entry supports several functional configurations:

- Enable/disable – The interrupt generation can be disabled.
- Sensitivity – Edge triggered or Level triggered interrupt.
- Polarity – Selects level interrupt generation on 0 or 1 state. Selects transition for edge sensitive interrupt.
- MSN message signaling is blocking/non-blocking – A non-blocking interrupt is cleared by message acknowledge. A blocking interrupt is cleared by access to GIM configuration space Status register.
- Interrupt signaling priority 0-3 – Sets the priority to generate an interrupt message.
- Target destination for interrupt:
  - Message signaling to CTOP SW:
    - Specified Thread in CTOP
      - The configured interrupt Type 0/1 allows the triggering of two different interrupt routines (mapped to different bits in CTOP IRQ register)
    - Specified CTOP core
    - Multicast – All CTOP threads in a specified CTOP. All CTOPs in a specified cluster. All threads in all CTOPs.
  - Interrupt output
  - PCIe-0 or PCIe-1

21.1.1 Interrupt Cause Status Indication

Each target destination – CTOPs, PCIe-0, PCIe-1, Interrupt output – has dedicated Interrupt cause status register that aggregates status of interrupts assigned to its destination.

Per destination there is a status register for Peripheral interrupts, Message interrupts, SERR and DERR.

Each interrupt entry holds the status: Valid (pending) / Invalid.

The level interrupts can be cleared by writing '1' to the interrupt cause register bit.

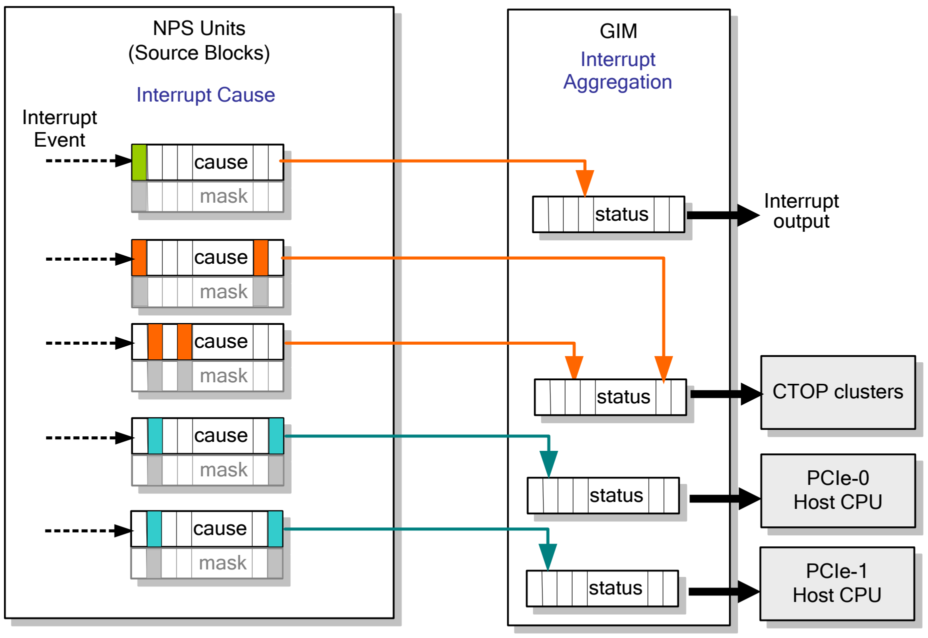
21.1.2 Peripheral and Unit Interrupts Mapping to GIM

The NPS supports a variety interrupt events/indications. The interrupt events are detected in the functional units (also referred to as “source blocks”). Each such event is propagated to the GIM. The GIM maps the sources to service target destinations in the Interrupt Cause Status register, allowing control of interrupt notification and propagation to CTOPs and external CPU / devices.

Each interrupt status is comprised of:

- 64b peripheral and units interrupt status register.
- 16b message interrupt status register
- 18b SERR interrupt status register
- 18 DERR interrupt status register

Figure 21-2. Interrupts



21.1.3 Source Units

Each functional unit block maintains a separate source event cause and mask bit for each detailed event (empty indication per-queue, ECC error indication per-memory, etc.). These are grouped into host registers according to event type (e.g. one register for empty queue bits, and another register for ECC error bits).

Source interrupt events are attributed into two types: transient and stable.

Transient events may occur (and disappear) sporadically. These can be discrete/momentary events (such as ECC errors, memory access errors), a transient level state (such as a resource queue being empty) or there may even be a change in a level state (such as link up/down status change). Indications of transient events are latched at the source block (setting the matching cause bit), along with any additional required information on the event (e.g. address in memory where ECC error occurred, link state before/after the link status change, etc.). This prevents flapping of the event and ensures that the SW is able to view a consistent snapshot of the event and any related data. Once the event is latched in the source block, the level/state of the latched cause bit is mirrored up to the GIM.

In contrast, stable events represent a state that will not change without CPU intervention, such as the existence of frames/messages for the host to read (message FIFO not empty, etc). Stable level events do not need to be latched. These states are reflected in a status register (bit per event/cause), and the matching signals are simply mirrored to the GIM as is.

For each source event, an independent cause bit (for transient events) or status bit (for stable events) is supported. Cause and status bits can be read by the SW. In addition, each cause bit can be cleared atomically (independently per bit) by the SW, allowing the SW to explicitly acknowledge that an interrupt event has been processed per-source event. When an interrupt event is acknowledged, the source block releases the latched cause (and any additional latched statuses), re-enabling further interrupts on the released event. In addition, a read/write mask bit is supported for each source event.

In cases where a number of instances of an event exist in the source block (e.g., bit per queue-empty indication, bit per ECC error indication), the source block aggregates the various events into a single indication to the GIM (e.g. at least one queue is empty; at least one memory had an ECC error).

Table 21-1. Source units

		bit 63:44	Virtual Interrupts [19:0]
		bit 43	INTERRUPT_IN[1]
		bit 42	INTERRUPT_IN[0]
		bit 41	East Configuration Bridge Error
		bit 40	East PCIe Controller
		bit 39	West Configuration Bridge Error
bit 23	East network ports 0	bit 38	West PCIe Controller
bit 22	East network ports 1	bit 37	West network ports 0
bit 21	East network ports 2	bit 36	West network ports 1
bit 20	East network ports 3	bit 35	West network ports 2
bit 19	SMI MDC/MDIO - 1	bit 34	West network ports 3
bit 18	East debug LAN status / Level	bit 33	West debug LAN status / Level
bit 17	East TM OQ-0 /Level	bit 32	West TM OQ-0 /Level
bit 16	East TM OQ-1 /Level	bit 31	West TM OQ-1 /Level
bit 15	East TM OQ-2 /Level	bit 30	West TM OQ-2 /Level
bit 14	East TM OQ-3 /Level	bit 29	West TM OQ-3 /Level
bit 13	East TM OQ-4 /Level	bit 28	West TM OQ-4 /Level
bit 12	East Interlaken MAC	bit 27	West Interlaken MAC
bit 11	East debug LAN Rx Ready / Level	bit 26	West debug LAN Rx Ready / Level
bit 10	East debug LAN Tx Done / Level	bit 25	West debug LAN Tx Done / Level
bit 9	East Configuration Bridge timeout	bit 24	West Configuration Bridge timeout
bit 8	Clock and Reset Generator Watchdog		
bit 7	SPI interface		
bit 6	UART		
bit 5	SMI MDC/MDIO - 0		
bit 4	Real Time Counter		
bit 3	South side external TCAM recovery		
bit 2	South side internal TCAM parity scan		
bit 1	North side external TCAM recovery		
bit 0	North side internal TCAM parity scan		

21.2 CTOP Interrupt Processing

The CTOP supports two interrupt levels

- Lower Priority – Level 1
- Higher Priority – Level 2

Interrupt levels disabled on entry to handler

- Level 1 entry disables Level 1
- Level 2 entry disables Level 2 and Level 1

Interrupt service static priority

- Level 2 (high priority) interrupts from the lowest number up to the highest, then
- Level 1 (low priority) interrupts from the lowest number up to the highest

All interrupts disabled on entry to exception handler. This allows exceptions to be taken from interrupt handlers (e.g. use of translated memory).

All interrupts force a switch to kernel mode. There is no access to interrupt state/flags from user mode.

The interrupt forces a change to machine state

- PC and STATUS32 are saved in interrupt link registers
- The PC is reset to the interrupt vector
- STATUS32-bits are set appropriately

Interrupt Cause registers are set appropriately

- The interrupt is acted on
- The instruction at Select is aborted, and the remainder of the pipeline is flushed
- Fetching restarts at the PC (now pointing to the interrupt vector)

SLEEP instruction allows programmer to send the CPU to sleep for a period of low power consumption. The CTOP is awakened on an interrupt.

21.3 GIM Interrupt to PCIe-Host CPU

The interrupt source can be configured to interrupt PCIe resident host CPU.

The GIM interrupt triggers PCIe MSI (Message Signaled Interrupt).

The MSI parameters are configurable:

- Physical function 0-3.
- Traffic Class 2b.
- Virtual Function 7b.

22. Power-up and Initialization

This section describes the NPS-400 power-up, reset sequence and device initialization steps.

The power-up and reset sequence is controlled by several functional steps:

Table 22-1. Power-up and reset sequence

NO.	STEP	COMMENTS
1.	Power-up sequencer.	Sequencer operates with core reference clock CLK_REF_P/N.
2.	Reset configuration sampling.	Refer to *NPS-400 Hardware Reference Manual* for values sampled at reset.
3.	Clock generation PLL configuration and stabilization.	The system's core clock PLLs configuration and TM clock PLLs configuration is sampled at reset.
4.	External ROM device read of the boot code.	The boot can be performed from one of two code partitions. The boot code load can be skipped.
5.	Boot-code execution by CTOP core(s).	The boot code can configure and enable the required interfaces.
6.	Operating system (Linux) copy. OS boot.	
7.	Functional mode.	Interface configuration.
8.	SW activity watchdog re-trigger.	

Reset interface inputs:

- PO_RESETn – Asynchronous power-on reset. Resets all logic and PLLs.
- HW_RESETn – Asynchronous HW reset. Resets all functional units to their initial state, except SAR (Sample at Reset) logic and PLLs.
- JTAG_TRSTn – JTAG controller reset.

Configuration values sampled at reset:

- Core PLL clock and TM PLL clock frequency. PLL bypass.
- Internal memory built in test (MBIST) enable.
- SPI modes.
- Boot-loader enable and boot-code partition (main/secondary).
- Watchdog enable.

Refer to the *NPS-400 Hardware Reference Manual* for values sampled at reset.

22.1 Power-up Sequence

Provided below is the NPS power up sequence:

1. The PO_RESET is released when CLK_REF_P/N is stable. The power-on state machine operates on the east-side CLK_REF_P/N (CLK1_REF_P/N) clock.
2. The initial configuration is sampled at reset (SAR) from dedicated I/Os.
3. The system core and TM PLLs are released from reset with valid configurations. PCIe PLL and DRAM PLL have constant initial configurations that can be SW re-configured.
4. The sequence machine waits until PLLs are locked on target frequency. The PLL can be bypassed for debug as per SAR value.
5. The internal SRAM memory repair is executed. This step can be skipped by a SAR value.
6. The internal blocks are released from reset.
7. The boot-loader state machine is activated to copy boot-code from external non-volatile memory to internal memory and wake a selected CTOP core to initially configure the device. The boot-loader can be disabled by a SAR value.
8. The designated CTOP core is released from reset to execute the boot-code.
9. The boot-code is responsible for configuring other PLLs. The boot-code services the watchdog to indicate successful completion of the initialization sequence. Later on the watchdog timer can be re-enabled by a device driver to monitor and protect the real time application execution.

22.1.1 Power-up Sequence Failures

The power-up and reset sequence may result in failure. The failure cause is latched in registers.

The debug interfaces can be used to read the device status and re-configure the device.

Table 22-2. Power-up sequence failure actions

FAILURE CAUSE	*REMEDY ACTION*	*FUNCTIONAL DEBUG INTERFACE*
Core PLL doesn't reach lock state	The power-up sequence stops. Validate PLL power supply is according to requirements. Activate PLL bypass mode via SAR value. Issue power-up. Complete sequence with PLL bypass clock.	While PLL isn't in lock state, the JTAG is functional. In PLL bypass mode, the power-up sequence is completed and Slave MDC/MDIO is functional.
TM PLL doesn't reach lock state	The power-up sequence finishes without performing memory MBIST. The error is logged in status registers. Validate PLL power supply is according to requirements. Activate PLL bypass mode via SAR value, or disable MBIST mode via SAR value. Issue power-up.	While PLL isn't in lock state, the JTAG is functional. When power-up sequence is completed, the functional interfaces are: JTAG, Slave MDC/MDIO. When boot-up completes, the PCIe and network interfaces are functional.
Internal memory test failed	The power-up sequence completes. Failed SRAM may affect some functionality. The MBIST cannot be re-initiated via configuration.	When power-up sequence is completed, the functional interfaces are: JTAG, Slave MDC/MDIO. When boot-up completes, the PCIe and network interfaces are functional.
SerDes MBIST failure	The power-up sequence completes. Failed SerDes may not function. The SerDes MBIST cannot be re-initiated via configuration.	When power-up sequence is completed, the functional interfaces are: JTAG, Slave MDC/MDIO. When boot-up completes, the PCIe and network interfaces are functional. (Failed SerDes may not function.)

FAILURE CAUSE	***REMEDY ACTION***	***FUNCTIONAL DEBUG INTERFACE***
Boot-loader failure	Boot from Slave-MDC/MDIO. Slave-MDC/MDIO can also be used for new boot code load into the boot flash device.	When power-up sequence is completed, the functional interfaces are: JTAG, Slave MDC/MDIO.

22.2 Software Controlled Reset

The SW can trigger reset to selected functional units. The SW can also shut down clocks to selected functional units.

The SW reset can be globally synchronized to issue a reset to several units with a common trigger. The SW selects each unit to participate in synchronized reset event, then the reset trigger activated to all selected units at once.

Table 22-3. Globally synchronized reset participants

Block	Globally Synchronized SW Reset	Per Block SW Reset	Configura-tion Reset	Clock Control	Description
EZnet & crossbars	Yes	Yes	Yes*	--	*Note: requires global reset/
Network Receive DMA (RxNDMA)	Yes	Yes	Yes	Yes	Per side: east/west
Network Transmit DMA (TxNDMA)	Yes	Yes	Yes	Yes	Per side: east/west
Network Receive IFs (RxIF)	Yes	Yes	Yes	Yes	Per side: east/west
Network Transmit IFs (TxIF)	Yes	Yes	Yes	Yes	Per side: east/west
Network ports	Yes	Yes	Yes	Yes	Reset per side: east/west Reset per side port group: 11-0. Reset per side port pair: 23-0.
Configuration bridge	Yes	--	--	--	Per side: east/west
DRAM controller	Yes	Yes	Yes	Yes	Per controller 0-11.
SPI	Yes	--	Yes**	Yes	**Note: single reset for both configuration and data flow.
Debug LAN	Yes	--	Yes	Yes	Reset per side: east/west
UART	Yes	Yes	--	Yes	Per clock domain
Buffer Management Unit (BMU)	Yes	Yes	Yes	--	Both sides of BMU are reset
Processor Management Unit (PMU)	Yes	Yes	Yes	Yes	Reset per PMU: east/west
Traffic Manager (TM)	Yes	Yes	Yes	Yes	Reset per side: east/west
I^2C	Yes	Yes	Yes	Yes	
Boot-loader SM	Yes	Yes	Yes	Yes	
Flow Control Unit (FCU)	Yes	Yes	Yes	--	Reset per side: east/west
General Interrupt Controller (GIM)	Yes	Yes	Yes	Yes	Reset per side: east/west
GCI	Yes	Yes	Yes	Yes	Reset per side: east/west
On-demand statistics	Yes	Yes	Yes	Yes	Reset per unit 0-7
Posted statistics	Yes	Yes	Yes	Yes	Reset per unit 0-3
Internal TCAM	Yes	Yes	Yes	Yes	Reset per side: north/south
External TCAM IF (ILKN-LA)	Yes	Yes	Yes	Yes	Reset per side: east/west
PCIe	Yes	Yes	Yes	Yes	Per PCIe

22.3 Initialization Sequence (boot load)

After the power-up sequence completes, the NPS device's functional units are released from reset. The CTOP cores are held in halt until initialization is completed.

The initialization is performed by reading initialization data from non-volatile serial memory via SPI interface (i.e. NOR-Flash, NAND-Flash) and writing either boot code into IMEM (initially viewed as non-partitioned flat memory) or configuration data into NPS registers.

The initialization sequence is:

1. Configure SPI master interface.
2. Read from serial memory 8B each time.
3. Check boot-code data integrity (embedded CRC).
4. Decode boot-code format: Op-code, Address, Data.
5. Check data block integrity.
6. Write decoded data to decoded NPS address (configuration register or internal memory).
7. Stop sequence on data error.
8. Last op-code command is to enable CTOP(s) and terminate sequence.

The initialization sequence terminates on errors in case of:

- Data integrity error due to CRC check.
- Data format error
- Time-out (if watchdog timer is enabled by SAR pins).
- Boot-code read image > 0.5MB.
- SPI interface error.

22.3.1 Serial Boot-code Memory

The SPI interface can operate with all four valid combinations of clock polarity and phase. Four communication modes are available (Mode 0, 1, 2, 3) that basically define the SCLK edge on which the MOSI line toggles, the SCLK edge on which the master samples the MISO line, and the SCLK signal steady level (i.e. the clock level, high or low, when the clock is inactive). Each mode is formally defined with a pair of parameters called 'clock polarity' (CPOL) and 'clock phase' (CPHA).

The desired SPI operational mode is sampled at rest (SAR).

The SPI frame format is Motorola. The Data size is 8b.

The SPI clock frequency is (Core frequency / 32).

The SPI master issues a Read Command (8'h03) to serial memory for each 8B of data and uses a 3B address.

The serial memory device must support at least 16MB. The boot-code size is up to 0.5MB.

22.3.2 Boot-code Load Options

The boot code supports two portions: Primary (main) and Secondary.

The SAR value determines which partition to load first.

When Primary is selected by SAR value, the initialization starts loading of the Primary partition and if the load attempts fail, it loads the Secondary partition.

When the Secondary partition load fails, the boot-load terminates.

The Primary partition 24b address is 0x0F0000.

The Secondary partition 24b address is 0x0F8000.

22.3.3 Boot-code Execution

The boot code partially configures the NPS device and loads execution code to IMEM of the designated set of CTOP cores. After boot load successful termination, a CTOP core is released to execute the loaded code.

The CTOP program can read the BOOTLOADER_CONF SAR value to determine the boot mode:

- Initialize PCIe and wait for external host CPU to configure the device.
- Initialize PCIe and complete OS boot from serial memory.
- Complete OS boot from serial memory.

The CTOP boot code initializes the DRAM memories and copies an image of the operating system to the DRAM. It then executes a final jump to an EMEM (DRAM) address to enable IMEM initialization and partitioning for the actual application.

A. Acronyms and Abbreviations

This section contains an alphabetical list of the acronyms, abbreviations and terms frequently used in this document. Terms highlighted are of particular relevance to the NPS product line.

10GBASE-KR	10Gbps that operates over a single backplane lane
ACK	ACKnowledge message
ACL	Access Control List
AES	Advanced Encryption Standard
Aging	On-chip mechanism to delete old entries from hash tables.
AH	Authentication Header protocol
ALU	Arithmetic Logic Unit; for computational functions
ARP	Address Resolution Protocol
ASID	Address Space Identifier
ATM	Asynchronous Transfer Mode
AVS	Automatic Voltage Scaling
BD	Buffer Descriptor; data structure describing a single frame buffer
BDR	Buffer Descriptor Ring
BID	Budget Identifier; in Buffer Management Unit
BLT	Buffer Link Table
BMA	Buffer Management Algorithm
BMC	Buffer Management Counter
BMU	Buffer Management Unit; manages pools of frame buffers and a pool of jobs, supports resource budget management allocation and release
BSD	Berkley Standard Distribution
BST | BIST	Built in Self Test
BT	Burst Tolerance
BW	Bandwidth
CAM	Content Addressable Memory
CAUI	100 GbE (10 x 10.3125Gbps) Attachment Unit Interface (electrical interface) (C Roman numeral for 100)
CB | CBFC	Class-based flow control
CBR	Constant Bit Rate
CBS	Committed Burst Size for srTCM
CDPI	Coarse DPI
CDR	Clock Data Recovery
CESR	Carrier Ethernet Switches and Routers
CIR	Committed Information Rate
CIU	Cluster Interface Unit
CLDMA	Cluster DMA; provides data transaction services to NPC threads
CMEM	Core’s Local Memory; in CTOP
CMT	Channel Mapping Table
CN	Congestion Notification
COS | CoS	Class of Service
CPPI	100 Gigabit Parallel Physical Interface
CSM	Context Switch Manager; in MTM
CTOP	EZchip’s C-programmable Task Optimized Processor; internal blocks
CTT	Connection Translation Table
DA	Destination Address
DCB	Data Center Bridging
DDP	Direct Data Placement
DDR3 SDRAM	Double Data Rate Type Three Synchronous Dynamic Random Access Memory
DDR4 SDRAM	Double Data Rate Type Four Synchronous Dynamic Random Access Memory
DFA	Deterministic Finite Automaton
DMA	Direct Memory Access
DPI	Deep Packet Inspection
DSLAM	Digital Subscriber Line Access Multiplexers
EBP	Explicit Burst Protection
EBS	Excess Burst Size
ECC	Error-Correcting Code
EDP	EMEM Frame Data Buffer Pool Controller; BMU index pool
ETMP	EMEM TM PD Buffer Pool Controller; BMU index pool
EFA	Ethernet Fabric Adapter
EIR	Excess Information Rate
EMEM	NPS External Memory
Entry	User-defined data in search structure, i.e. key, result and other associated data
EP	Endpoint; PCIe
EPC	Evolved Packet Core; 4G mobile
ESP	Encapsulation Security Payload protocol
ETMP	EMEM TM PD Buffer Pool Controller; BMU index pool
ETS	Enhanced Transmission Selection
EZcesr	EZchip carrier Ethernet switching and routing application

EZcp	EZchip Control Plane SDK; APIs for configuration and management of the NPS device; supplied with EZdk
EZdk	EZchip SDK is a comprehensive set of design and runtime tools developing both data-plane and control-plane applications for NPS
EZdp	EZchip Data Plane SDK; API library for data plane services; supplied with EZdk
EZdpi	EZchip deep packet inspection solution
EZetcamConfig	EZchip utility for use with external TCAM; supplied with EZdk
EZide	EZchip Eclipse™-based Integrated Development Environment; supplied with EZdk
EZ-ISA	EZchip Instruction Set Architecture
EZldk	EZchip Linux® development kit; supplied with EZdk
EZnet	NPS cross chip fabric to interconnect several internal blocks including CTOPs, DMAs and PMUs
EZof	EZchip OpenFlow
EZpci	EZchip high bandwidth PCIe interconnect
EZregex	EZchip PCRE compiler
EZsec	EZchip IPsec encryption/decryption and authentication
EZsim	EZchip NPS software simulator; supplied with EZdk
EZtap	EZchip utility; supplied with EZdk
EZtcp	EZchip TCP/IP protocol stack
FC	Flow Control
FCR	Flow Control Report; updates BMU
FCU	Flow Control Unit
FD	Frame Descriptor
FDPI	Fine DPI
FEC	Forward Error Correction
FIC	Fabric Interface Controller
FID	Flow ID for NPS Traffic Manager
FLT	Frame Link Table
FMT	Fixed Mapping Table
FPX	Floating Point Acceleration; in CTOP
FQ	Fair Queuing
GB	Gigabyte or grant bucket
GbE | GE	Gigabit Ethernet
Gbps	Gigabits per second
GCI	GigaChip Interface; high bandwidth, low latency memory interface for external memory
GIM	NPS general interrupt controller block
HBA	Host Bus Adapter
HEC	Header Error Checks
HFCBGA	High-performance Flip Chip Ball Grid Array
Host	Refers to an external control CPU
HT	HyperTransport
HW	Hardware
IC	Interrupt Controller
ICMEM	Internal Code Memory
ICMEMG	Internal Code Memory Global
ICMP	Internet Control Message Protocol
ICU	Input Classification Unit; performs input packet classification
ICV	Integrity Check Value
IDE	Integrated Development Environment
IDMEM	Internal Data Memory
IDMEMG	Internal Data Memory Global
IDMEMGIO	Internal Data Memory Global I/O
IDP	IMEM Frame Data Pool Controller; BMU index pool
IDS	Intrusion Detection Systems
IE | IFE	Interface Engine; internal blocks
IFG	Inter Frame Gap; also IPG
IFU	Interface Units, specifically RxIFU and TxIFU
IGMP	Internet Gateway Message Protocol
IJP	IMEM Job Pool Controller; BMU index pool
ILKN	Interlaken interface
ILKN-LA	Interlaken Look-Aside interface
ILP	Instruction Level Parallelism
IMEM	NPS Internal Memory
Interlaken | ILKN	An interconnect protocol optimized for high bandwidth and reliable packet transfers
Interlaken-LA | ILKN-LA	Interlaken Look-Aside; protocol suitable for short, transaction-related transfers
IPC	Inter Processor Communication
IPG	Inter Packet Gap; also IFG
IPI	Inter Processor Interrupts
IPS	Intrusion Prevention Systems
IPsec	Internet Protocol Security; protocol suite for securing IP communications
ISA	Instruction Set Architecture
ISR	Interrupt Service Routines
ISSU	In Service Software Update
ITMP	IMEM TM Pool Controller; BMU index pool
ITT	Initiator Transfer Tag
IV	Initialization Vector
JD	Job Descriptor; data structure tracking the state of all frames being processed
JID	Job Identifier; also Job ID; token by which one references a single job

Job	Concurrent tasks on multiple processors in the system
JSM	Job Status Manager; in MTM
JTLB	Joint Translation Lookaside Buffers
Key	Used to perform a lookup in a search (data) structure.
LAG	Link Aggregation Group
LBD	Linked Buffer Descriptor; an array of buffers descriptors
LCMEM	Local Code Memory
LDMEM	Local Data Memory
LIF	Logical Interface
LKP	Look-up in a search (data) structure
LL | LLFC	Link level flow control
LLP	Linear Line Protection
LPM	Longest Prefix Match
LRAM	Local RAM
LRU	Least Recently Used
MAC	Message Authentication Code or Media Access Control
MC	Memory Controller or multicast
MCN	Memory Core Network
MDDR	Modified Deficit Round Robin
MDN	Memory DMA Network
MEF	Metro Ethernet Forum’s Traffic Management Specification
MMF	Multi-Mode Fiber
MMU	Memory Management Unit; provides virtual memory addressing, memory protection and cache control
MPLS	Multiprotocol Label Switching
MPLS-TP	Multiprotocol Label Switching Transport Profile
MPO	Multiple-Fibre Push-On
Mpps	Million packets per second
MR	Multicast Replicator; TM internal block
MSI	Message Signaled Interrupt
MSID	Memory Space Identifier
MSN	Messaging Network
MSS	Maximum Segment Size
MSU	Messaging & Scheduling Unit; internal block
MTM	Multi Thread Manager; CTOP HW threads scheduler
MTP	Multiple-Fibre Pull-off
MTU	Maximum Transmission Unit; TCP/IP
NDMA	Network DMA, specifically RxNDMA and TxNDMA
NFA	Nondeterministic Finite Automaton
NP	EZchip Network Processor
NPC	Network Processor Cluster; clusters with each consisting of an array of CTOPs with local memory, L2 cache and a data bus interface unit
NPS	EZchip NP for Smart networks; EZchip product family
NPU	Network Processor Unit; such as EZchip’s NP and NPS product lines
NRT	Non Real Time traffic
OAM	Operations, Administration and Maintenance
OCP	Occupancy Congestion Policer; in TM
OFD	On-the-Fly Decisions
OLT	Optical Line Termination
OOB	Out-of-band
OQ	Output Queue; internal TM block
OTN	Optical Transport Network
PADDR	Physical address
PBD	Protection by Duplication
PBS	Peak Burst Size
PCIe	PCI Express; Peripheral Component Interconnect Express
PCS	Physical Coding Sublayer
PCR	Peak Cell Rate
PCRE	Perl Compatible Regular Expression
PD	Packet Descriptor or Priority Drop
PDR	Drop Precedence Report; updates BMU
PDU	Protocol Data Unit
PFC	Priority Flow Control
PFQ	Per Flow Queuing; also called TM
Phase	May be used refer to hardware revisions of the NP/NPS device.
PHB	Per Hop Behavior
PIB	Policy Information Base
PIR	Peak Information Rate
PMA	Physical Medium Attachment
PMD	Physical Medium Dependent
PMU	Processor Management Unit; handles packet ordering and scheduling
PPS	Packets per second
PQ	Physical Queue; internally managed
PRBS	Pseudorandom Binary Sequence
PS	Phantom size
PSID	Packet Switch Identifier for NPS Traffic Manager
PSOFS	PSID offset
PTE	Page Table Entry
PTP	Precision Time Protocol; may refer to the IEEE 1588 protocol
PUL	Power Up Library
QCN	Quantized Congestion Notification
QDR	Quad Data Rate

QoS	Quality of Service
RD	Raw Data
RDMA	Remote Direct Memory Access
RE2	Regular Expression Library
Regex	Regular Expression
Result	Key-associated data returned from the match in a search (data) structure.
RISC	Reduced Instruction Set Computing
RR	Round Robin
RT	Real Time
RTC	Real Time Clock
RTT	Round Trip Time
RxIFU	Receive Interface Unit; internal blocks
RxNDMA	Receive Network DMA; internal blocks
SA	Source Address
SAR	Sample At Reset, or Segmentation and Reassembly
SCM	Statistics Counter Manager; internal block
SCP	Statistics Code Profile; TM statistics
SCR	Sustained Cell Rate
SD	Search Descriptor; handler to a specific search structure data element
SD entry	Structure Definition entry defining a search structure data element
SDK	Software Development Kit
SDN	Software Defined Networking
SDRAM	Synchronous Dynamic Random Access Memory
SDRAM Controller	Memory controllers for DDR3/DDR4 SDRAM devices
SDT	Structure Definition Table; internal table defining the data structures
Search Structure	Data structure for lookups (e.g. table, hash)
SECDED	Single Error Correction Double Error Detection
SerDes	Serializer/Deserializer
SFP	1GbE small form-factor pluggable transceiver
SFP+	10GbE enhanced small form-factor pluggable transceiver that use an SFI interface
SFT	Stateful Flow Table
SGMII	Serial Gigabit Media Independent Interface
SLA	Service Level Agreement
SMF	Single-Mode Fiber
SMP	Symmetric multi-processing Linux®
SoC	System on a Chip
SRAM	Static random-access memory
srTCM	Single Rate Three Color Meter
SSL	Secure Sockets Layer
SSP	Search Structure Protection
SW	Software
TBS	Target Board Support
TCAM	Ternary Content Addressable Memory
TLB	Translation Lookaside Buffer; internal block
TLP	Thread Level Parallelism
TM	Traffic Manager; internal blocks
TOS	Type of Service
trTCM	Two Rate Three Color Meter
TTL	Time To Live
TxIFU	Transmit Interface Units; internal blocks
TxNDMA	Transmit Network DMA; internal blocks
UC	Unicast
UDM	Unified Data Memory; CTOP internal block
UTF-8	UCS (Universal Character Set) Transformation Format, 8-bit
VC	Virtual Channel
VOQ	Virtual Output Queuing
VPN	Virtual Private Network
VRF	Virtual Router Function
WFQ	Weighted Fair Queuing
WRED	Weighted Random Early Discard
WRR	Weighted Round Robin
XFP	10 Gigabit Small Form Factor Pluggable modules that use a XFI interface
XLAUI	40 GbE (4 x 10.3125-Gbps) Attachment Unit Interface (electrical interface) (XL Roman numeral for 40)
XLPPI	40 Gigabit Parallel Physical Interface
XSMI	10GE Serial Management Interface