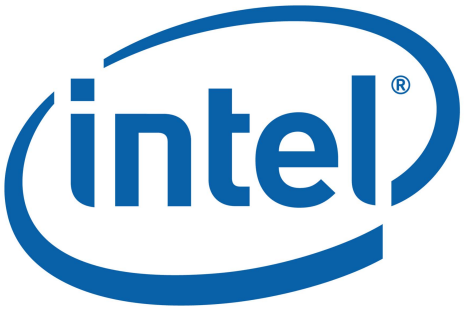


Intel® Ethernet Switch FM5000/ FM6000

1 Gb/2.5 Gb/10 Gb/40 Gb Ethernet (GbE) L2/L3/L4 Chip

Datasheet

Networking Division (ND)

Revision 3.3
November 2014
331496-001

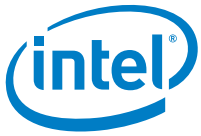
LEGAL

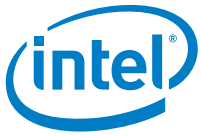
Revision History

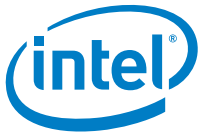
Revision	Date	Comments
3.3	November 10, 2014	General updates, including: • Section 1.1, "Part Numbering" — Revised description of package type to include FCBGA. • Table 1-2, "Definition of Terms" — Revised the cut-through latency value for the FlexPipe™ definition. • Section 3.3.10, "JTAG Pins" — Updated description of TRST_N pin. • Table 6-6, "EPL Standard Modes" — Updated SFI information. • Section 9.4, "SerDes Management" — Added notes regarding SBus IDs.
3.2	March 26, 2014	General updates, including: • Section 9.9, "LED Controller" — Updated LED controller state decoding in Table 9-7.
3.1	February 11, 2014	General updates, including: • Section 10.2.2, "Maximum Peak Current" — Revised maximum peak current values for AVDD in Table 10-3, Table 10-4 and Table 10-5. • Section 10.2.3, "Maximum Sustained Power" — Revised maximum sustained power values for AVDD in Table 10-6, Table 10-7 and Table 10-8. • Section 11.4, "Pin List Ordered by Name" — Correct pin errors in Table 11-2.
3.0	November 22, 2013	General updates, including: • Reordered chapter sequence, and renamed Section 2.0, "FM5000/FM6000 Capabilities" to Section 5.0, "Frame Processing". • Moved overview information from Section 5.0, "Frame Processing"and added to new Section 2.0, "Architecture Overview". • In Section 10.0, "Electrical Specification", updated power sequencing drawing and added REFCLK specification. • In Section 11.0, "Mechanical Specification", added legend to ballout diagram. • Replaced Figure 11-2, "1677-Ball Package Top/Side View"with new drawing containing notes for call-outs.
2.3	August 7, 2013	General updates, including: • Revised pin description tables in Section 3.0, "Pin Descriptions". • Revised Table 3-1, "Boot Mode Selection by GPIO[9..7] Pin Strap Latched at Reset". • Revised Section 10.0, "Electrical Specification". • Revised Section 11.0, "Mechanical Specification".
2.2	June 11, 2013	Initial release (Intel confidential).

NOTE: ***This page intentionally left blank.***

Contents







1.0 Introduction

The Intel® Ethernet and Switch series consists of the Intel® Ethernet Switch FM2000 series L2 switch chip platform, the Intel® Ethernet Switch FM4000 series multi-layer switch chip platform and the high-performance Intel® Ethernet Switch FM5000/FM6000 Series (FM5000/FM6000) multi-layer switch chip platform. This document provides the following information:

- Functional descriptions
- Frame processing pipeline
- Hardware design
- Pin interface
- Electrical specifications
- Package mechanical
- Thermal information

Note: For software details, see the *Intel® Ethernet and Switch Family Software API*.

1.1 Part Numbering

The FM5000/FM6000 provides a range of part numbers as listed in Table 1-1. Each part number is configured with a maximum core bandwidth that puts a limit on the sum of the maximum port bandwidth plus the maximum management bandwidth. The management bandwidth is the portion of the core bandwidth that can be used by Ethernet management ports and/or the CPU interface, and is limited to less than 20 GbE. The ports can be allocated in any manner as long as the maximum port bandwidth is not exceeded.

Table 1-1 Part Numbering

Part Number	Max Port Bandwidth	Max SGMII or 1 GbE Serdes Ports	Max XAUI Ports	Max 10 GbE Serial Ports	Max 40 GbE Ports
FM5224	240G	72	2	8	2
FM6324	240G	72	24	24	6
FM6348	480G	72	24	48	12
FM6364	640G	72	24	64	16
FM6724	240G	72	24	24	6
FM6764	640G	72	24	64	16

This document pertains to all variants of the FM5000/FM6000 series, although most references are specific to the 64-port 10 GbE version of the device. All devices are manufactured in a Flip-Chip Ball Grid Array (FCBGA), 1677-ball package.

1.2 Definitions

Table 1-2 Definition of Terms

Term	Definition
{A,B}	Denotes a bit concatenation of variable A or B where A is most significant bit field and B is least significant bit field. ***Note:*** {0,A} means that 0s are added to the left (most significant bits) to pad to the desired size while {A,0} means padded to the right.
Bit Numbering	Bit 0 is the least significant bit throughout the architecture (even if Ethernet standards and specifications suggest otherwise).
Byte	8 bits.
Double Word	64 bits.
EBI	External Bus Interface — Intel's term for a legacy address/data external bus interface to processor.
F32/F64/F96	Intel-proprietary inter-switch link tag, which is used to pass relevant management and control information from one Intel Ethernet and Switch Family device to another in a network. Not supported in the FM2000 series.
FlexPipe™	The name for the FM5000/FM6000 series high-performance flexible frame processing pipeline that operates at over one billion packets-per-second with less than 400 ns cut-through latency. Microcode can be used to provide flexible pipeline configurations.
FM2000	The first generation of the Intel® Ethernet and Switch Family. The FM2000 is a Layer 2 (L2) 10 GbE switch chip platform that forms the basis for many L2 switch product variants, including the FM2224, FM2212, FM2208, FM2112, FM2104, and FM2103.
FM4000	The second generation of the Intel® Ethernet and Switch Family. The FM4000 is an enhanced multi-layer 10 GbE switch chip platform that forms the basis for new switch product variants, all of which are pin-compatible with their original FM2000 series counterparts. The FM4xxx devices are full-featured L3 routing devices, and contain other enhancements, such as ACL's, congestion management, increased frame memory and more. The FM3xxx devices are similar to the FM4xxx, except that L3 routing is not included.
FM5000/FM6000	The third generation of the Intel® Ethernet and Switch Family. The FM5000/FM6000 enhances functionality and bandwidth while providing package options that are pin compatible with the FM4000 series. The FM5000/FM6000 L2/L3/L4 devices provide advanced CEE/DCB features for the Data Center.
GloRT	Intel-proprietary Global Resource Tag, which is used to pass global identification information from one device to another in a network. GloRT is the proper pronunciation, although other uses might appear in the document and have the same meaning. For example, glort, Glort, GloRT or GLORT.
Half Word	16 bits.
Logging	Logging refers to a copy of the frame sent to a local CPU for monitoring purposes.
Mirroring	Mirroring refers to a copy of the frame sent to another port for monitoring purposes.

Table 1-2 Definition of Terms (Continued)

Term	Definition
Packet or Frame	**Packet** — On a typical computer network, data is transmitted in the form of structured and modest-sized packets. Instead of transmitting arbitrary-length strings of data, structured packets Packet or Frame allow error checking and other relevant processing to occur on smaller easier-to-retransmit data. Packetized data also helps to alleviate traffic jams on the network when multiple nodes are contending for a shared network resource. **Frame** — While a packet is a small block of data, a Frame is the definition of how packets of data are defined and transported on a specific network. When sending data over a network, both sides of the connection must agree on a common frame format (e.g., when a frame starts, when a frame ends, padding, etc.) Combining terms, an Ethernet packet is sent onto an Ethernet interface using an Ethernet frame format. This document uses both terms interchangeably.
PCIe Express* (PCIe*)	PCIe Interface to a processor. Compliant to PCIe Gen 2.
RapidArray	The name for Intel's single output queued shared memory architecture that is used in the FM2000, FM4000 and FM5000/FM6000 series 10 GbE switch chip platforms.
Register Type	Registers are split into fields of the following types: RO: Read-Only (This register cannot be programmed by software. Typically reports a status.) RW: Read-Write (Reading returns the value written.) CW Clear-on-Write (Writing to the register clears the register.) CW1: Clear-on-Write-1 (Writing 1b to any bit clears that bit.) CR: Clear-on-Read (Reading the register clears the register.) RV: Reserved (For upward compatibility. Always write as 0b, ignore on read.) WO: Write-Only (The value written cannot be read back.)
Segment	A portion of a packet corresponding to one of the common architecturally-significant storage and processing chunks that has been defined in the architecture. For FM5000/FM6000 devices, a segment is 160 bytes.
Trapping	Trapping refers to special frames that are captured by the switch and redirected to a local CPU for processing.
Word	32 bits.
X[0..N]	Denotes an array with indexes that go from 0 through N, inclusively.
X[N:0] or X[0:N]	Denotes a bit range within a variable. The bit range [0:N] indicates that bit 0 is the most significant, while bit range [N:0] indicates that the bit 0 is the least significant.

NOTE: ***This page intentionally left blank.***

2.0 Architecture Overview

2.1 Overview

The FM5000/FM6000 main components are shown in Figure 2-1.

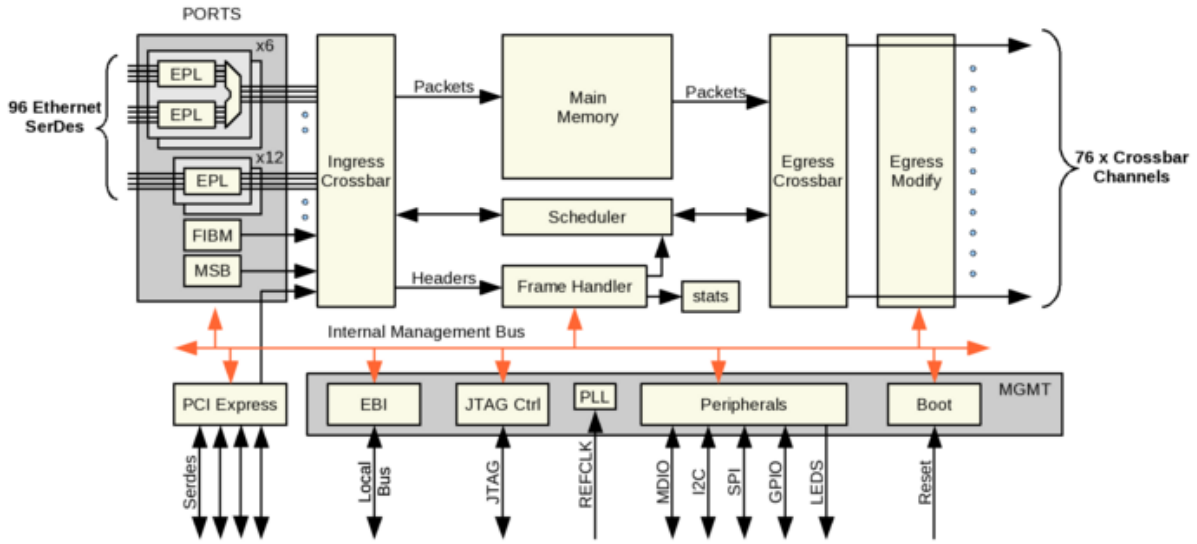


Figure 2-1 FM5000/FM6000 Block Diagram

At the switch ingress, the Ethernet Port Logic (EPL) parses incoming packets to extract the packet payload, which gets forwarded to the shared packet memory through a receive crossbar. In the reverse direction, the scheduler forwards packets from the shared packet memory through the transmit crossbar, to the egress modifier. The egress modifier might modify the packet before forwarding it to the EPL along with recomputing the CRC. All packet headers are processed by the frame handler.

2.1.1 EPL

EPL is an interface between the Ethernet SerDes and the internal data structures. It implements PCS, reconciliation layer, and the MAC layer. The EPL also implements PCS-type management, such as fault handling, auto-negotiation, etc. Various SerDes-to-channel mapping combinations are offered to enable support for 1GBASE-X, SGMII, XAUI, 10 GbE serial and 40 GbE MLD. There are 24 EPLs in the switch. Two arrangements are supported. The first 12 EPLs are arranged in pairs, in which each pair shares 40 GbE of bandwidth to the crossbar.

The next 12 EPLs are directly attached to the crossbar, in which each EPL has access to up to 40 GbE worth of bandwidth to the crossbar (4 x 10 GbE channels).

See Section 6.2 and Section 6.3 for more details concerning the 24 EPLs.

2.1.2 Ingress Crossbar

The role of the ingress crossbar is to receive a serial stream of data at 10 GbE and forward it to the main memory as segments. The segment size is 160 bytes. The first 112 bytes of the first segment are passed to the frame handler for packet processing, switching and/or routing decisions.

The ingress and egress crossbars support up to 76 full-duplex channels. Any channel can support a throughput of 10 GbE (four channels are combined together for 40 GbE support) but the total aggregate throughput is limited to 720 GbE.

2.1.3 Packet Memory

The main memory stores incoming packets from ingress EPLs and forwards them to egress EPLs upon request from the scheduler. The main memory size is 7.5 MB. Additional memory in other functional blocks brings the total packet and frame header memory to approximately 9.5 MB.

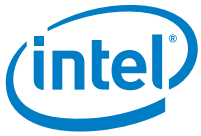
2.1.4 Frame Handler

The frame handler makes forwarding decisions based on the frame header received from the EPL. The forwarding information (port mask and modification data) is sent to the scheduler.

2.1.5 Scheduler

The scheduler manages free data segments, maintains receive and transmit queues and schedules packets for transmission. The free segments are forwarded as needed to the receive crossbar, which uses them to store incoming packets to the right location in the shared memory fabric. The scheduler keeps the list of the segments sent to each EPL/MSB/IPL and waits for the frame handler forwarding decision before placing the packet at the tail of the proper transmission queue.

The scheduler then applies advanced scheduling algorithms to decide which packet to forward to the EPL/MSB/IPL. It then sends a segment list to the transmit crossbar, which uses those segments to retrieve the packet from memory and forward the payload to the egress modifier.



2.1.6 Egress Crossbar

The egress crossbar receives pointers from the scheduler, reads the packets from memory and forwards them to the egress modifier for packet modification.

2.1.7 Egress Modifier

The egress modifier executes the instructions received from the scheduler to modify the packets as data gets transmitted. Examples of egress modification includes new DMAC/SMAC/VLAN and VLAN translation as well as adding/removing an MPLS header.

2.1.8 MSB

The MSB unit is an internal port to the switch fabric for EBI attached devices. The MSB receives frames from the EBI and forwards them to the switch or receives frames from the switch and forwards them to the EBI.

2.1.9 PCIe

The PCIe unit (fixed at internal port 0) is the primary interface for controlling the switch. Also, the PCIe has direct access to the switch fabric enabling packets to flow between the fabric and the host processor through a built-in DMA engine.

2.1.10 Management

The management system includes many entities: legacy external address/Data Bus Interface (EBI), JTAG controller, peripherals, boot controller and PLL.

2.1.11 Global Resource Tag (GloRT) Definition

GloRT is a 16-bit number that can be used to identify a specific port, link aggregation group, multicast group, management frame or any other packet destination in a single or multi-stage fabric.

The source GloRT is generally used to identify the logical source port of the frame as it enters a multi-stage switch fabric. The destination GloRT is generally the logical destination port or group of ports to where the frame exits the multi-stage fabric. Each FM5000/FM6000 must be configured with the following set of GloRTs.

- **Per-port Source GloRT** — Each time a frame arrives without an Inter-Switch Link (ISL) tag, this source GloRT is associated with the frame. If the frame's source MAC address is learned, this GloRT value is stored in the MAC address table.

- **CPU GloRT** — Each time a frame is trapped, logged or mirrored to the CPU, the top eight bits of the destination GloRT are set to this value. The bottom eight bits are set to a trap code value, chosen by hardware to indicate the reason for sending the frame to the CPU.

In addition, other feature-specific GloRTs can also be created on-the-fly using API commands:

- Rx mirror GloRT
- Tx mirror GloRT
- Multicast GloRT
- Link aggregation GloRT
- Load balancing group GloRT

The mapping from GloRT to a physical port involves a ternary CAM lookup. This enables the 16-bit GloRT space to be allocated in a fairly arbitrary manner. The only structure required by the mapping function consists of:

- A configurable bit range within link aggregation group GloRTs, used to identify individual physical port members.
- Fixed 8-bit CPU trap code field within the CPU GloRT.

2.1.12 Multi-chip and Intel® Tags

Multiple switches can be aggregated together to create a larger logical switch by connecting them through normal Ethernet ports, and by extending the Ethernet packet format to carry extra information. Intel defines three extra tag formats, F56, F64 and F96, for this purpose:

- The F56 tag is 7 bytes long and replaces the seven bytes of preamble between the SDF symbol and the first byte of the payload.
  - This tag is available on 10 GbE and 40 GbE links only.
- The F64 tag is 7 bytes long and is inserted at byte 12 (right after SMAC).
  - This tag replaces the VLAN tag.
- The F96 tag is 12 bytes long and is inserted at byte 12 (right after SMAC).
  - This tag replaces the VLAN tag.

Table 2-1 shows the F56 tag (first byte is preamble SDF symbol).

Table 2-1 F56 Tag

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SDF								FTYPE				SWPRI			
VLAN PRI			CFI	VLAN											
Source GloRT															
Destination GloRT															

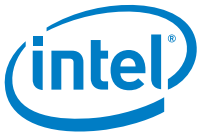


Table 2-2 shows the F64 and F96 tags.

Table 2-2 F64/F96 Tags

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FTYPE				SWPRI				USER							
VLAN PRI			CFI	VLAN											
Source GloRT															
Destination GloRT															
EXTRA (F96 only)															
EXTRA (F96 only)															

The FTYPE is:

0x0000 = Normal frame

0x1000 = Special delivery

NOTE: ***This page intentionally left blank.***

3.0 Pin Descriptions

The FM5000/FM6000 comes in several port bandwidth options. This section describes the largest port bandwidth configuration. The actual pin-list is presented in Table 11-1 and Table 11-2.

3.1 Pin Overview

Figure 3-1 shows an overview of the pins used for the FM5000/FM6000. Some part variants are restricted to a reduced number of Ethernet ports due to core bandwidth limitations. See Section 11.0, "Mechanical Specification" for additional pin information.

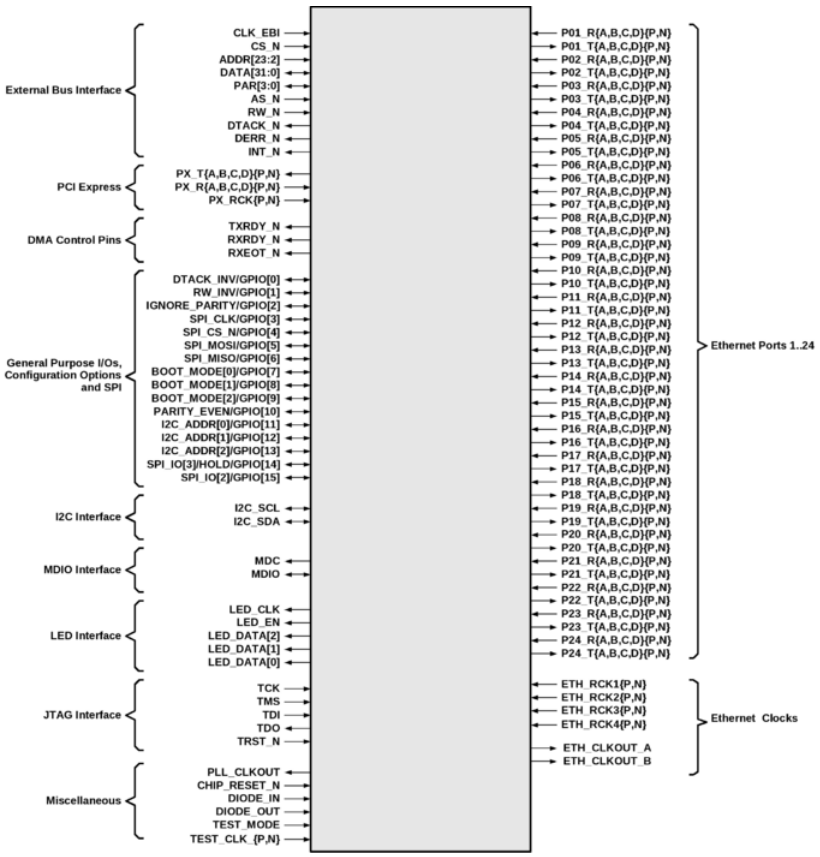


Figure 3-1 FM5000/FM6000 Pin Overview

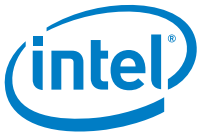
3.2 Signal Name Convention

The following signal name conventions are used:

- **Signal Mnemonic** — The signal mnemonic is a generic name used as a prefix to identify the function of this pin.
- **Negative Logic** — Signal with inverted logic (0=asserted, 1=deasserted) are designated by using the signal mnemonic following by _N suffix.
- **Differential Pairs** — Differential signals are designated by using the signal mnemonic followed by _P for the positive pin and _N for the negative pair.
- **Bus Designation** — A signal that is part of a bus is designated by using the signal mnemonic followed by [n] to designate the pin number in that bus. The entire bus or part of a bus is designated using the [M...N] designation. As an example, DATA[31..0] bus represents 32 pins designated DATA[0], DATA[1], etc...
- **Set** — A set of signals is referenced globally using name{M...N} or name{A,B,C,D}. The actual pin designation is formed by using the signal mnemonic and concatenating one of the values of the set. As an example, the pin set REFCLK{1..4}{A,B}{P,N} represents 16 pins designated REFCLK1AP, REFCLK1AN, REFCLK1BP, etc...
- **Direction** — The following pin direction types are used:
  - IN: Input to the chip
  - OUT: Output from the chip
  - IN/OUT: Might operate as input or output
  - OD: Open drain output
  - OC: Open collector output
  - Power: A pin used for power
  - Ground: A pin used for ground
  - Sense: A pin used for sensing (no input/output concept)
- **Standard** — The following pin input/output buffer types are used:
  - CML
  - LVCMOS
  - LVPECL

3.3 Detailed Pin Descriptions

The FM5000/FM6000 pins are listed in the following tables. GPIOs pins are latched when CHIP_RESET_N is de-asserted to provide default configurations.



3.3.1 Ethernet Port Pins

Pin Name	Direction	Type	Usage
P{1..24}_R{A,B,C,D}{P,N}	IN	CML	SerDes receive ports. There are four pairs per port. Each set {A,B,C,D} of four pairs can be mapped to LANE0..LANE3. In CML or to LANE3...LANE0 by software. Unused pins can be left floating; they are internally terminated.
P{1..24}_T{A,B,C,D}{P,N}	OUT	CML	SerDes transmit ports (i=[1..24]). There are four pairs per port. Each set {A,B,C,D} of four pairs can be mapped to LANE0.LANE3 or to LANE3...LANE0 by software. Unused pins can be left floating; they are internally terminated.
ETH_RCK[1..4]{P,N)	IN	LVPECL	Ethernet reference clocks for SerDes; 156.25 MHz.
ETH_CLKOUT_{A,B}	OUT	LVCMOS	Recovered clocks from SerDes.

3.3.2 PCIe Pins

Pin Name	Direction	Type	Usage
PX_R{A,B,C,D}{P,N}	IN	CML	PCIe receive lanes.
PX_T{A,B,C,D}{P,N}	OUT	CML	PCIe transmit lanes.
PX_RCK{P,N}	IN	LVPECL	PCIe reference clock. Must be 125 MHz.

3.3.3 Power Pins

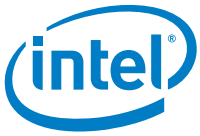
Pin Name	Direction	Type	Usage
VDD	Power	N/A	Variable core power supply that can be adjusted to support higher bandwidth.
VDDS	Power	N/A	Fixed core power supply.
VDD25	Power	N/A	TTL I/O power supply.
AVDD25	Power	N/A	LVPECL reference clocks power supply.
AVDD	Power	N/A	SerDes analog power supply.
VDDPLL	Power	N/A	Analog power for frame handler PLL.
VFB1, VFB2	Reserved	N/A	Must be tied to ground for normal operation. Permanent damage can occur if this pin is tied to a positive potential.
VSS	Ground	N/A	Ground.
VDDK	OUT	Power	Kelvin sense pin for the VDD power plane on the die. Leave open if not used.
VSSK	OUT	Power	Kelvin sense pin for the VSS power plane on the die. Leave open if not used.
VDDSK	OUT	Power	Kelvin sense pin for the VDD power plane on the die. Leave open if not used.
VSSK2	OUT	Power	Kelvin sense pin for the VSS power plane on the die. Leave open if not used.

3.3.4 External Bus Interface Pins

Pin Name	Direction	Type	Usage
CLK_EBI	IN	LVCMOS	Bus interface clock.
ADDR[23:2]	IN	LVCMOS	Address inputs.
DATA[31:0]	IN	LVCMOS	Data bus.
PAR[3:0]	IN/OUT	LVCMOS	Data parity.
CS_N	IN/OUT	LVCMOS	Chip Select (Active low). This pin is sampled when CHIP_RESET_N is de-asserted. If this pin is low, EBI is assumed to not be used and EBI_CLK is ignore and replaced internally with PCIE_REFCLK.
AS_N	IN	LVCMOS	Address Strobe (Input, active low).
RW_N	IN	LVCMOS	Read/Write. Defines the type of transaction (read or write) being requested. Polarity of this signal depends on the RW_INV strapping pin. When RW_INV is pulled down to ground, read is active high while write is active low. Conversely, when RW_INV is pulled up to VDD25, read is active low while write is active high.
DTACK_N	OUT	LVCMOS	Data Transfer Acknowledge. Indicates the completion of a data transfer. This signal is actively asserted for one cycle when data transfer is ready during read or latched during write and then actively de-asserted for 1 cycle, the cycle after, and finally tri-stated for all other cycles. Polarity of this signal is determined by DTACK_INV strapping pin. If DTACK_INV is pulled down to ground at reset, DTACK_N is active low. If DTACK_INV is pulled up to VDD25 at reset, DTACK_N is active high.
DERR_N	OUT	LVCMOS	Write Data Parity Errors. Only asserted (and valid) when DTACK_N asserted. Tri-stated otherwise.
INTR_N	Open drain	LVCMOS	Interrupt. This pin is active low and asserted each time an interrupt condition exists in the chip. The pin gets de-asserted once all interrupt sources have been cleared. Pull down current = 30 mA.

3.3.5 DMA Interface Pins

Pin Name	Direction	Type	Usage
TXRDY_N	OUT	LVCMOS	Transmit FIFO Ready. Asserted each time the switch can accept a new word of data for packet transmission from the CPU to the network.
RXRDY_N	OUT	LVCMOS	Receive Data Ready. Asserted each time the switch has data in its receive FIFO for the CPU.
RXEOT_N	OUT	LVCMOS	End of frame indication. Asserted while reading the last data word of a packet to indicate this is the last work of the packet.



3.3.6 GPIOs and Strapping Pins

Pin Name	Direction	Type	Usage
GPIO[0]/DTACK_INV	IN/OUT	LVCMOS	General Purpose I/O pin. This pin is also sampled when CHIP_RESET_N is de-asserted to determine DTACK polarity mode. If the pin is sampled low, DTACK_N is active low, if the pin is sampled high, DTACK_N is active high (operate as DTACK rather than DTACK_N).
GPIO[1]/RW_INV	IN/OUT	LVCMOS	General Purpose I/O pin. This pin is also sampled when CHIP_RESET_N is de-asserted to determine R/W polarity mode. If the pin is sampled low, read is active high while write is active low. Conversely, when sampled high, read is active low while write is active high.
GPIO[2]/IGN_PAR	IN/OUT	LVCMOS	General Purpose I/O pin. This pin is also sampled when CHIP_RESET_N is de-asserted to determine parity mode. If the pin is sampled high, parity on writes is not checked. If the pin is sampled low, parity on incoming writes is checked.
GPIO[3]/SPI_SCK GPIO[4]/SPI_CS_N GPIO[5]/SPI_MOSI GPIO[6]/SPI_MISO	IN/OUT	LVCMOS	Used for either SPI or General Purpose I/O pins. These pins are used as SPI when the switch retrieves its configuration from an external SPI EEPROM (if enabled) and as GPIO after. The pin directions when the interface is used as SPI are: SPI_CLK = Out SPI_CS_N = Out SPI_MOSI = Out SPI_MISO = In
GPIO[7]/BOOT_MODE[0] GPIO[8]/BOOT_MODE[1] GPIO[9]/BOOT_MODE[2]	IN/OUT	LVCMOS	General Purpose I/O pin. These pins are also sampled when CHIP_RESET_N is de-asserted to define the boot mode of the switch (see Table 3-1).
GPIO[10]/PARITY_EVEN	IN/OUT	LVCMOS	General Purpose I/O pin. This pin is also sampled when CHIP_RESET_N is de-asserted to determine the parity mode on the data bus after reset (high = even, low = odd). This pin must be pulled up or pulled down and cannot be left unconnected.
GPIO[11]/I2C_ADDR[0]	IN/OUT	LVCMOS	General Purpose I/O pin. This pin is sampled when CHIP_RESET_N is de-asserted to set the I^2C slave address of the switch.
GPIO[12]/I2C_ADDR[1]	IN/OUT	LVCMOS	General Purpose I/O pin. This pin is sampled when CHIP_RESET_N is de-asserted to set the I^2C slave address of the switch.
GPIO[13]/I2C_ADDR[2]	IN/OUT	LVCMOS	General Purpose I/O pin. This pin is sampled when CHIP_RESET_N is de-asserted to set the I^2C slave address of the switch.
GPIO[14]/DATA_HOLD/SPI_IO3	IN/OUT	LVCMOS	General Purpose I/O pin. This pin is sampled when CHIP_RESET_N is de-asserted to define DTACK and DATA behavior. If this pin is pulled up, DTACK and DATA (if read) are asserted when needed and remain asserted until CS is de-asserted. If this pin is pulled down, DTACK and DATA are asserted (if read) for one cycle, DTACK gets actively de-asserted for one cycle and tri-stated after that. This pin should be pulled up or down. It should not be left unconnected. Also used as IO[3] for SPI when operating in quad-pin mode.
GPIO[15]/SPI_IO2	IN/OUT	LVCMOS	General Purpose I/O pin. Also used as IO[2] for SPI when operating in quad-pin mode.

3.3.7 I²C Pins

Pin Name	Direction	Type	Usage
I2C_SCL	Open drain	LVCMOS	I²C Clock.
I2C_SDA	Open drain	LVCMOS	I²C Data.

3.3.8 MDIO Pins

Pin Name	Direction	Type	Usage
MDC	OUT	LVCMOS	MDIO Clock.
MDIO	Open drain	LVCMOS	MDIO data.

3.3.9 LED Pins

Pin Name	Direction	Type	Usage
LED_CLK	OUT	LVCMOS	Serial LED Clock.
LED_EN	OUT	LVCMOS	Serial LED Enable. Asserted only on the first bit of an 80-bit frame.
LED_DATA[2:0]	OUT	LVCMOS	Serial LED Data. These pins are latched at power up to control internal termination on the 2.5 V LVPECL reference clocks inputs. Using a pull down enables the termination, using a pull up disables the termination. These pins have weak internal pull down. LED_DATA[0] = Controls internal termination for ETH_RCK inputs. LED_DATA[1] = Controls internal termination for PX_RCK inputs. LED_DATA[2] = Controls internal termination for TEST_CLK inputs.

3.3.10 JTAG Pins

Pin Name	Direction	Type	Usage
TCK	IN	LVCMOS	JTAG Clock. Internal pull up.
TMS	IN	LVCMOS	JTAG Control. Internal pull up.
TDI	IN	LVCMOS	JTAG Data Input. Internal pull up.
TDO	OUT	LVCMOS	JTAG Data Output.
TRST_N	IN	LVCMOS	JTAG Reset input pin. Active low to reset for the JTAG port. The external Pdn value should be 470 Ω to 1 KΩ, with the internal pull-up impedance ranging from 25 KΩ to 50 KΩ. This provides a GND level to disable the JTAG port under normal operation when the JTAG testing function is not used.

3.3.11 Miscellaneous Pins

Pin Name	Direction	Type	Usage
CHIP_RESET_N	IN	LVCMOS	On the FM5000/FM6000 series, all strapping options are latched while the chip is in reset. Internal pull down.
DIODE_IN DIODE_OUT	Sense	Analog	Note that the temperature versus voltage might be different on the FM5000/FM6000 series.
PLL_CLKOUT	OUT	LVCMOS	Copy of PLL Output. Presence of this signal is controlled by PLL configuration register.
TESTMODE	IN	LVCMOS	Connect to GND for normal operation.
TEST_CLK_[P,N]	IN	LVPECL	Connect to GND for normal operation.
PAD_TRI_N	IN	LVCMOS	Leave open for normal operation.

3.4 Boot Mode Selection

Table 3-1 Boot Mode Selection by GPIO[9..7] Pin Strap Latched at Reset

BOOT_MODE				Mode	Usage
[2]	[1]	[0]	HEX		
0	0	0	0x0	Slave	CPU acts as I^2C, SPI, or EBI master to boot the FM5000/FM6000.
0	0	1	0x1	I^2C Master	Boot from I^2C serial boot ROM at address 0x50, image at offset 0.
0	1	0	0x2	Not Supported	
0	1	1	0x3	I^2C Master	Boot from I^2C serial boot ROM at address 0x51, image at offset 0.
1	0	0	0x4	SPI Master	Boot from SPI serial boot ROM, image address pointer at offset 0.
1	0	1	0x5		Boot from SPI serial boot ROM, image address pointer at offset 4.
1	1	0	0x6		Boot from SPI serial boot ROM, image address pointer at offset 8.
1	1	1	0x7		Boot from SPI serial boot ROM, image address pointer at offset 12.

NOTE: ***This page intentionally left blank.***

4.0 Power Up, Reset and Interrupts

This section describes how to power up and reset the FM5000/FM6000 device along with a description of the interrupt controller block.

4.1 Power

The power-up sequence steps are:

1. Apply power.
   - Ensure CHIP_RESET_N is maintained asserted as long as power is not valid.
2. Apply clock.
   - Ensure CHIP_RESET_N is maintained asserted for at least 1024 clock cycles.
3. De-assert CHIP_RESET_N.
4. Use a serial SPI Flash or I^2C EEPROM to boot the chip.
   - A serial SPI Flash or I^2C EEPROM is required for minimum setup.
5. Use the PCIe or EBI interface to boot and manage the switch (optional).
   - Use the FM5000/FM6000 API platform driver code to bring up the switch features.
   - Apply networking protocol software.

4.2 Reset

Figure 4-1 shows the reset structure.

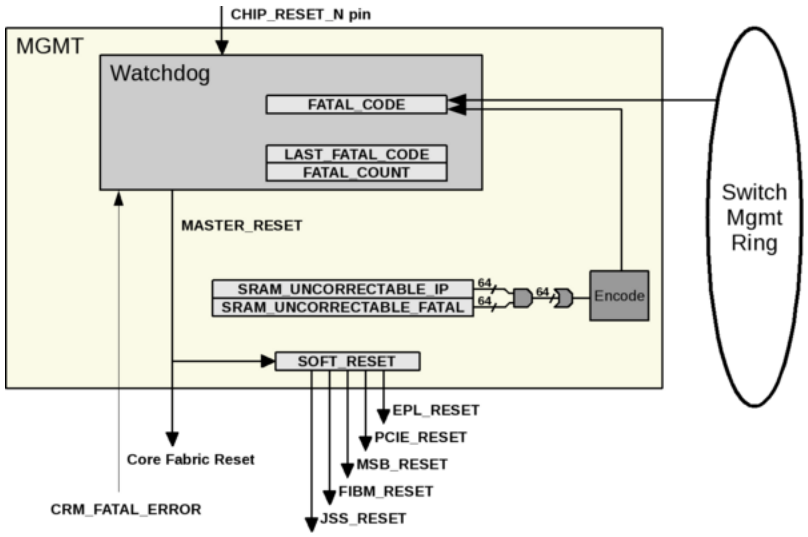


Figure 4-1 FM5000/FM6000 Reset Domains

The different control points are as follows:

CHIP_RESET_N (external pin)

Places the watchdog in reset when asserted. Afterwards, the watchdog automatically asserts the internal MASTER_RESET. The GPIO pins are also reset to input.

When CHIP_RESET_N is de-asserted, the GPIO pins are sampled and the watchdog module is immediately taken out of reset. The watchdog then releases the internal MASTER_RESET after a few cycles. It also initializes the FATAL_COUNT and FATAL_CODE registers to 0x0.

FATAL_CODE Register (internal register in watchdog)

There are three methods to write into this register:

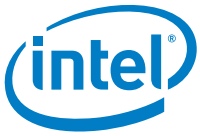
- Detecting an uncorrectable SRAM error (each uncorrectable SRAM is maskable).
- Timeout of CRM access.
- Direct write from any bus master to this register via a management ring master (PCIe, EBI, etc...).

When the register is written and the WATCHDOG_CFG enables reset upon write to FATAL code, the watchdog copies FATAL_CODE into LAST_FATAL_CODE, clears FATAL_CODE, increments FATAL_COUNT by one and waits 64 cycles before asserting MASTER_RESET.

MASTER_RESET (internal master reset)

The internal MASTER_RESET is an output of the watchdog circuit. It causes the management module and core fabric to be placed in reset when asserted, forcing those modules to regain their default state and reset all registers to their default value. Manageability also contains the SOFT_RESET register that controls the reset status of PCIe, MSB, JSS, and EPL modules. The JSS modules include a SerDes micro-controller and Serial Bus Ring (SBUS) master.

When MASTER_RESET is de-asserted, the management module boots the system by checking the status of the BOOT_MODE pins.



SOFT_RESET Register

The SOFT_RESET register controls the reset for different modules: EPLs, PCIe, JSS (SPICO/SBUS) and MSB. Its default value is to assert reset on all modules. Each module must be taken out of reset before the platform is accessed.

The entire boot process is listed in Table 4-1.

Table 4-1 Reset Process Detail

Step	Description	Details
1	Asserting Reset	Asserting CHIP_RESET_N or the self-asserting WATCHDOG causes the MGMT module to go in reset and, by consequence, causes all other modules to go into reset. All registers are reset to their default values. GPIO pins are reset to input mode.
2	De-asserting Reset	De-asserting CHIP_RESET_N or the self de-asserting WATCHDOG causes the following: • The GPIO pins are sampled, providing the initial configuration for various elements. • The internal MASTER reset control is kept asserted for 64 cycles and then self-clears.
3	Switch Initialization	De-assertion of the internal MASTER reset causes the boot controller to start to boot the chip. The boot controller does the following: • Transfers the content of the fusebox to the different modules. • Uses sampled BOOT_MODE pins to determine if booting form serial ROM is requested and what type of serial ROM is used.
4	Select Boot Method	Boot from serial ROM: • If booting from ROM is enabled, the boot controller reads and executes the instructions in the ROM until the last instruction (END) is retrieved. Boot from CPU: • If booting from ROM is disabled, the boot controller is stalled until boot instructions are received from the CPU through the BOOT_CTRL register.
5	Select Normal Operating Mode	Write 0xFFFFFFFF to SCAN_CHAIN_DATA_IN to put the core logic and the EPLs into normal operating mode.
6	Setup PLL	Initialize PLL and wait for PLL lock. The maximum lock time is 80 ms. The CPU might poll the PLL_STATUS register if desired to reduce waiting time.
7	Enable Modules	Take all modules out of reset (EPL, PCIe, MSB, SPICO/SBUS).
8	Perform FFU Slice Number Assignment	When booting from the serial boot ROM: • Send command BOOT (Initialize FFU Slice Numbers). • EEPROM fetching is on hold until completed. When booting from the CPU: • Write command Initialize FFU Slice Numbers into BOOT_CTRL:Command register. • Wait for BOOT_STATUS:CommandDone to return to 1b.
9	Perform Bank Memory Repair	When booting from the serial boot ROM: • Send command BOOT (Apply Bank Memory Repairs). • EEPROM fetching is on hold until completed. When booting from the CPU: • Write command Apply Bank Memory Repairs into BOOT_CTRL:Command register. • Wait for BOOT_STATUS:CommandDone to return to 1b.
10	Perform Freelist Initialization	When booting from the serial boot ROM: • Send command BOOT (“Initialize All Scheduler Freelists”). • EEPROM fetching is on hold until this is completed. When booting from the CPU: • Write command Initialize All Scheduler Freelists into BOOT_CTRL:Command register. • Wait for BOOT_STATUS:CommandDone to return to 1b.

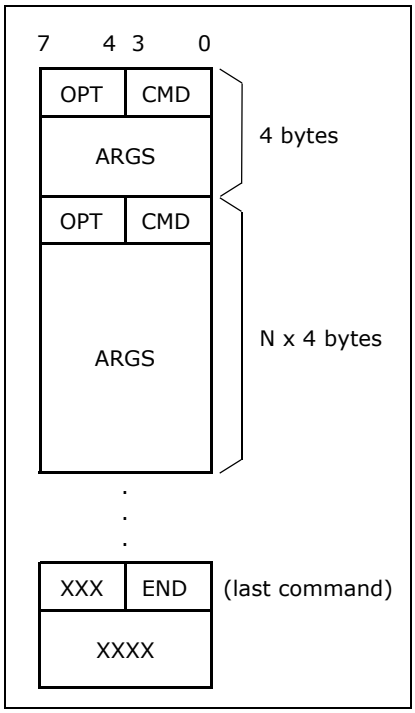
Table 4-1 Reset Process Detail (Continued)

Step	Description	Details
11	Start PCIe	If PCIe is used, BOOT ROM must setup PCIe SerDes and take PCIe out of reset.
12	Initialize Memory	This step could be: • Use CRM to setup memory table. • Launch CRM execution. • Wait for completion. Or: • Software writes memory manually.

4.3 Serial Boot ROM Format

The FM5000/FM6000 supports booting from a serial EEPROM attached on either the I^2C bus or the SPI bus. The SPI bus is much faster (up to 62.5 MHZ) and supports large memory sizes.

The serial boot ROM format is the same for both types of memories, and consists of a series of instructions encoded as follows.



The size of the instructions is at least 4 bytes long and always a multiple of four bytes. The first 4 bits make up the command, followed by a 4-bit option code to the command and then arguments to the command.

Note: If the FM5000/FM6000 is reset during boot, the EEPROM should also be reset by performing a power cycle.

The command encoding is detailed in Table 4-2.

Table 4-2 Boot ROM Command Encoding

Command	Code	Option	ARGS and Data
WRITE	1	NWORDS-1	ARGS[0] = ADDR[23:16] ARGS[1] = ADDR[15:8] ARGS[2] = ADDR[7:0] ARGS[3] = DATA[0][31:24] ARGS[4] = DATA[0][23:16] ARGS[5] = DATA[0][15:8] ARGS[6] = DATA[0][7:0] ARGS[7] = DATA[1][31:24] . . . ARGS[66] = DATA[15][7:0]
POLL	2	Not used	ARGS[0] = ADDR[23:16] ARGS[1] = ADDR[15:8] ARGS[2] = ADDR[7:0] ARGS[3] = DATA[31:24] ARGS[4] = DATA[23:16] ARGS[5] = DATA[15:8] ARGS[6] = DATA[7:0] ARGS[7] = MASK[31:24] ARGS[8] = MASK[23:16] ARGS[9] = MASK[15:8] ARGS[10] = MASK[7:0] ARGS[11] = MAX_RETRY[15:8] ARGS[12] = MAX_RETRY[7:0] ARGS[13] = RETRY_INTERVAL[15:8] ARGS[14] = RETRY_INTERVAL[7:0] ARGS[15] = not used ARGS[16] = JUMP_ADDRESS[23:16] ARGS[17] = JUMP_ADDRESS[15:8] ARGS[18] = JUMP_ADDRESS[7:0]
WAIT	3	Not used	ARGS[0] = TIME[23:16] ARGS[1] = TIME[15:8] ARGS[2] = TIME[7:0]
BOOT	4	Not used	ARGS[0] = Not used ARGS[1] = Not used ARGS[2] = boot command BOOT 4 Not used The boot commands defined are: 1 = Initialize FFU slice numbers 2 = Apply bank memory repairs 3 = Initialize all scheduler Freelists 4 = Initialize only the array Freelist 5 = Initialize only the head storage Freelist 6 = Initialize only the TXQ Freelist 7 = Initialize only the RXQ Freelist
LCNT	5	Counter 0 Counter 1	ARGS[0] = Not used ARGS[1] = COUNT[15:8] ARGS[2] = COUNT[7:0]
LOOP	6	Counter 0 Counter 1	ARGS[0] = JUMP_ADDRESS[23:16] ARGS[1] = JUMP_ADDRESS[15:8] ARGS[2] = JUMP_ADDRESS[7:0]

The polling function polls a register at regular intervals until a masked value is found or maximum retry count is reached. The boot stops if the maximum number of retries is reached without the expected value having been found. The function is typically used to issue a command (such as initializing a block of memory) and wait for completion.

```
retry = 0;
w = read_reg(ADDR);
while ( retry < MAX_RETRY && (w & MASK) != VALUE)
{
  retry++;
  wait_clock_count(RETRY_INTERVAL);
  w = read_reg(ADDR);
}
```

The LCNT and LOOP are used to do loops. The LCNT instruction loads one of the two 16-bit counters with a constant value. The LOOP instruction checks the counter value and, if not zero, decrements the counter and jumps to the address indicated. These instructions are intended to create a series of packets easily.

The waiting times (WAIT and RETRY_INTERVAL) are in PCIE_REFCLK clock cycles.

The data and addresses are always encoded Most Significant Byte (MSB) first. Examples illustrating the byte ordering are:

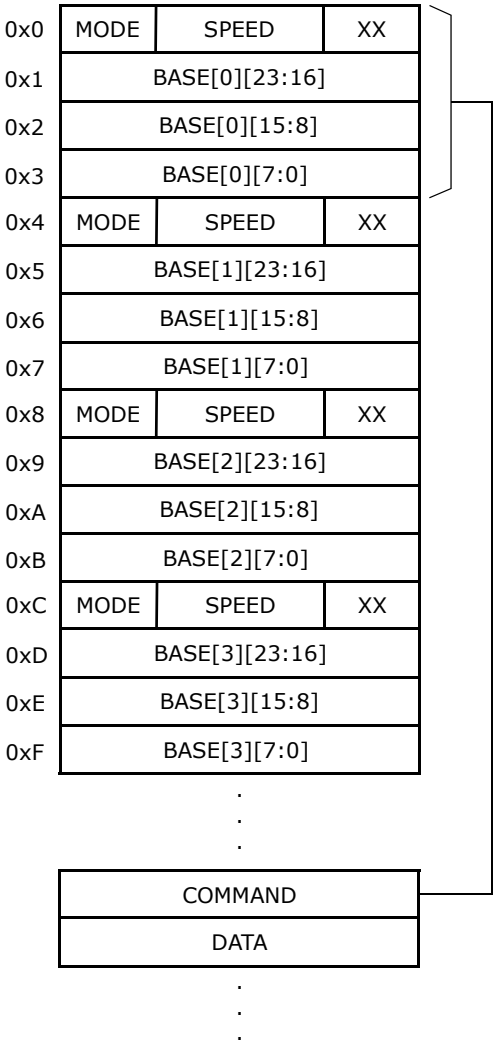
```
WRITE ADDR=0x40104 DATA=0x19 => [11 04 01 04 00 00 00 19]
DELAY 10 usec (03 00 04 E2)
BOOT CMD 1 (04 00 00 01)
```

The location of the image and the type of interface to use depends on BOOT_MODE pin strapping, as shown in Table 4-3.

Table 4-3 Boot Mode Pin Strapping

BOOT_MODE			Usage
[2]	[1]	[0]	
0	0	0	Boot from CPU only (not valid for PCIe operation only)/
0	0	1	Reserved.
0	1	0	Boot from I^2C serial boot ROM at address 0x51, image at offset 0.
0	1	1	Boot from I^2C serial boot ROM at address 0x50, image at offset 0.
1	0	0	Boot from SPI serial boot ROM, image address pointer at offset 0.
1	0	1	Boot from SPI serial boot ROM, image address pointer at offset 4.
1	1	0	Boot from SPI serial boot ROM, image address pointer at offset 8.
1	1	1	Boot from SPI serial boot ROM, image address pointer at offset 12.

If SPI serial boot ROM are used, the first four 32-bit words define the base address of each image. The image selection [0..3] depends on the BOOT_MODE strapping as follows:



The first byte defines the SPI operating mode and speed.

Table 4-4 SPI First Byte

Bits	Name	Description
2:0	DON'T CARE	N/A
5:3	SPEED	111b = 62.5 MHz 110b = 31.2 MHz 101b = 15.6 MHz 100b = 7.8 MHz 011b = 3.9 MHz 010b = 2.0 MHz 001b = 1.0 MHz 000b = 0.5 MHz
7:6	MODE	00b = Single (command code = 0x03) 01b = Dual (command code = 0x3B) 10b = Quad (command code = 0x6B) 11b = Single_fast (command code = 0x0B)

4.4 Interrupt Controller

4.4.1 Normal Interrupts

Interrupt handling includes the following registers:

- **Interrupt Pending Registers** — The interrupt pending registers (XXX_IP) contain one bit per interrupt source that gets set when an interrupt condition is detected. The interrupt condition is cleared by software by writing the corresponding bit to 1b, writing a 0b has no effect.
- **Interrupt Mask** — The interrupt mask registers (XXX_IM) have a direct correspondent bit-per-bit to the interrupt pending registers and is used to mask out the interrupt source if desired. A masked interrupt (setting bit to 1b) does not post an interrupt up in the hierarchy.
- **Interrupt Detect Registers** — The interrupt detect registers are a collective representation of the presence of an unmasked interrupt source in the system or sub-system. The register GLOBAL_INTERRUPT_DETECT is the top register.
- **Global Interrupt Mask Registers** — The global interrupt mask registers enable designers to select which global interrupt gets presented to which output.

Figure 4-2 shows the overall interrupt hierarchy.

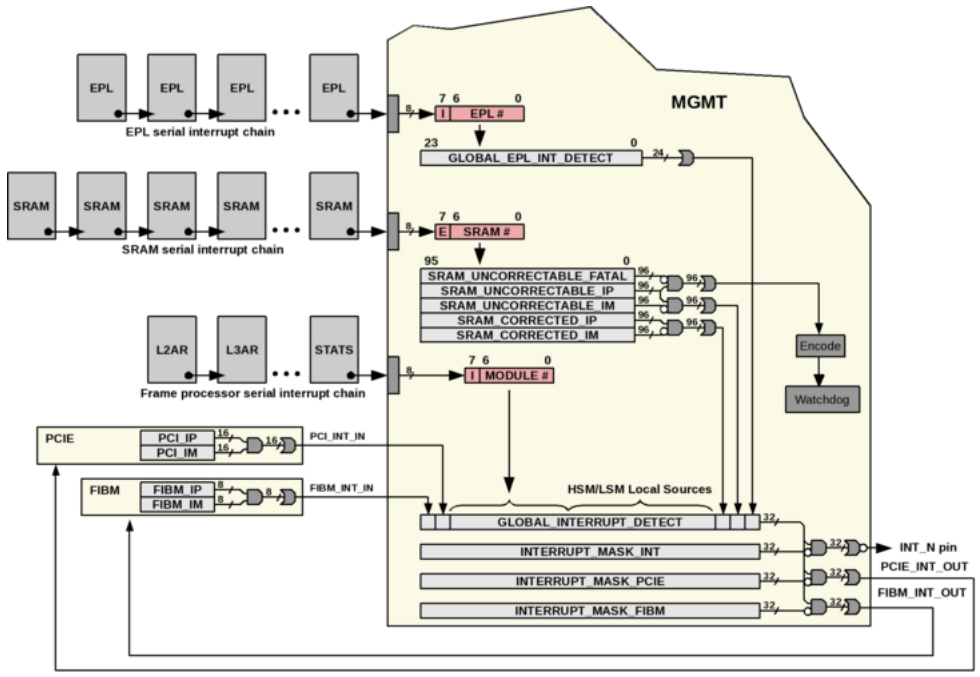


Figure 4-2 FM5000/FM6000 Interrupt Hierarchy

The elements in the hierarchy are listed in Table 4-5.

Table 4-5 Interrupt Sources

Module	Description
EPL	All EPL blocks are daisy-chained through a serial chain. The vector posted to manageability is an 8-bit vector with the lower 7 bits indicating the EPL number (only 0..24 are valid) and upper bit indicates if an interrupt is pending or not. The EPL generates a vector immediately after it is taken out of reset and also each time the EPL_IP becomes non-zero or goes to zero. If the vector has not been sent yet and a new condition arises that requires sending an updated vector (this could happen if the chain is delayed temporarily and an event create an interrupt or the software just cleared all interrupts in this EPL), the new vector replaces the one that has not left yet. This ensures that the latest state is reported to the manageability module. The manageability module, upon receiving the vector, sets or clears the corresponding bit in the GLOBAL_EPL_INT_DETECT register for that EPL. It also sets the EPL_INT bit in the GLOBAL_INTERRUPT_DETECT if any bit is set in the GLOBAL_EPL_INT_DETECT register.
FPP	All FPP modules are daisy-chained through a serial chain. The vector produced is an 8-bit vector with the lower 7 bits indicating the FPP module number and upper bit indicates if an interrupt is pending or not. A particular FPP module generates a vector each time an interrupt condition is raised or each time all interrupt conditions got cleared. An interrupt condition is detected each time an IP bit is set and corresponding IM bit is cleared. As for EPL, if the vector has not been sent yet and a new condition arises that requires sending an updated vector, the new vector replaces the one that has not left yet. This ensures that the latest state is reported to the manageability module. The manageability module, upon receiving the vector, sets or clears the corresponding bit in the GLOBAL_INTERRUPT_DETECT register for that module.

Table 4-5 Interrupt Sources (Continued)

Module	Description
SRAM	All SRAM blocks are daisy-chained through a serial chain and have ECC memory protection. An interrupt is posted each time an ECC error is detected while reading a particular location. The vector produced SRAM is an 8-bit vector with the lower 7 bits indicating the SRAM block number and upper bit indicates the error type. There are two types of errors detected: corrected errors and non-correctable errors. A corrected error event is an event where the type of error detected was correctable and corrected. A non-correctable error is an event where the type of error detected was not correctable. If the vector has not been sent yet and a new error is detected, the new vector can only replace an existing vector if the new vector is a non-correctable error. This guarantees reporting of all non-correctable errors. The manageability module, upon receiving the vector, either sets a bit in either SRAM_CORRECTED_IP or SRAM_UNCORRECTABLE_ERR depending of the type received. Manageability also sets the SRAM_C_ERR or SRAM_U_ERR bit in the GLOBAL_INT_DETECT each time an interrupt is posted and the corresponding mask bit is cleared.
MGMT Local Sources	Manageability includes local interrupt sources that get posted in the GLOBAL_INTERRUPT_DETECT each time they are active and not masked out. Those sources are GPIO, MDIO, I^2C, SOFT, CRM, and LCI.
PCIe	The PCIe can also be a source of interrupts. For example, GLOBAL_INTERRUPT_DETECT.
Posting Interrupts	Three methods are available: • INT_N pin • In-band PCIe Manageability provides three mask registers for each of these outputs enabling designers to select which interrupt gets posted by which method.

4.4.2 Fatal Interrupts

The switch is capable of detecting this type of event and resetting itself. This is accomplished by configuring the SRAM_UNCORRECTABLE_FATAL register to enable chip reset after detecting an uncorrectable memory error.

If the chip is reset for this reason, the LAST_FATAL_CODE contains the SRAM block that caused the error and the FATAL_COUNT is incremented.

Chip reset causes the serial EEPROM to reload so the switch can be reached.

5.0 Frame Processing

5.1 FM5000/FM6000 Capabilities

Table 5.1 describes the capabilities of the FM5000/FM6000.

Table 5-1 FM5000/FM6000 Product Family Features

Feature	FM5000/FM6000
Number TCAM rules	24 K
TCAM remapping	Yes
64 K binary search tree	Yes
Next hop table	Yes
Egress CoS queues	8
QCN support	Yes
TRILL support	Yes
Double VLAN tagging	Yes
IEEE 1588 support	Yes
Tunneling features	No
Q-in-Q PB support	No
MAC-in-MAC PBB support	No
Synchronous Ethernet	No

5.2 Application

The FM5000/FM6000 operates as a one-armed IP router combined with an Ethernet L2 switch (see Figure 5-1). In this architecture, incoming packets are first associated with a VLAN (using the VTAG tag if present, or by associating a default VLAN), and are then either switched within their respective VLANs or routed across VLANs or both. The decision to switch, route or drop depends on tables stored in the FFU (which includes a large ternary CAM to store IP route entries and access control lists), other tables located in the switch (such as MAC Address table) or tables located in the router (such as ARP table).

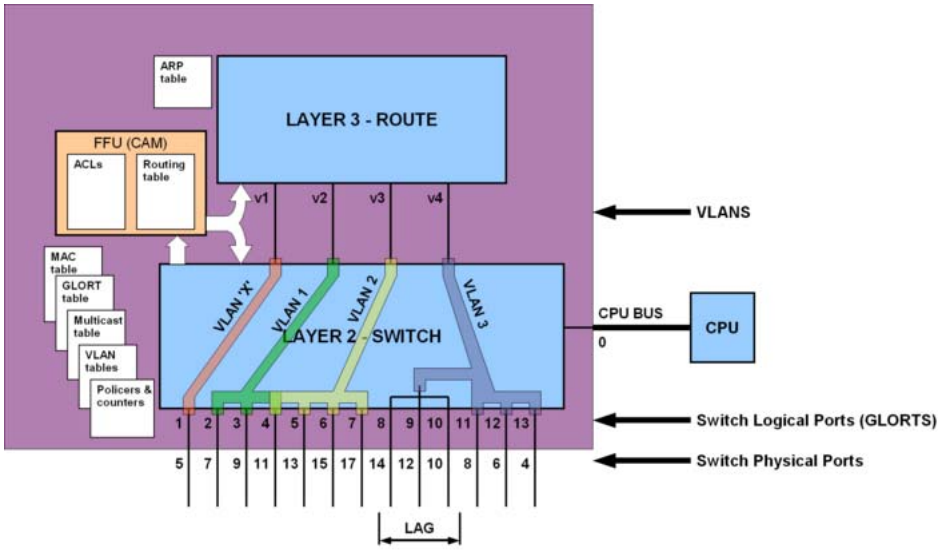


Figure 5-1 FM5000/FM6000 Switch/Router Concept

The switch includes a set of features such GloRT, distributed link aggregation, inter-switch tags and advance multicast distribution to enable a set of physical switches to operate as a single logical switch as shown in the figures that follow. Figure 5-2 shows a stack arrangement, while Figure 5-3 shows a fat-tree (Clos) architecture. Both examples enable exploitation of the full feature set through a F64 tag used on internal links. The F64 is described later in this section.

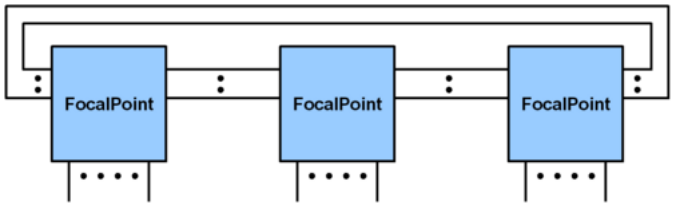


Figure 5-2 FM5000/FM6000 in a Stack Topology

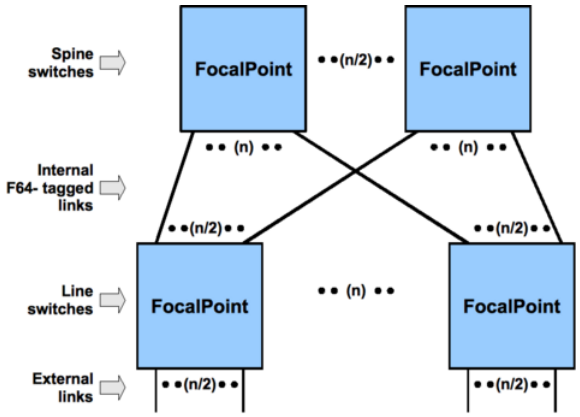


Figure 5-3 FM5000/FM6000 in a Tightly Coupled Clos Topology

5.3 Frame Processor

The FM5000/FM6000 series frame processor is designed to handle wire-speed L2/L3/L4 switching in the context of a single-chip or a multi-chip solution in a variety of topologies. The features offered are:

- Global Layer 3 (L3) routing over multiple devices in fat tree, ring or meshed topologies, with support for Equal Cost Multipath (ECMP) route selection
- L2 switching with optional automatic address learning and security
- Tunneling support for protocols such as TRILL, MPLS, VPWS, VPLS, Q-in-Q, MAC-in-MAC
- DCB support for PFC, ETS, QCN and DCBx
- Server virtualization support for protocols such as VEPA+
- Basic and extended Access Control Lists (ACLs) for L2/L3/L4 and deep packet inspection
- Snooping of IGMP v1, v2, and v3
- Link aggregation across multiple links using various sources of information from the frame header to derive the hashing function
- Trapping special frames
- Egress filtering and redirections including mirroring and logging
- Traffic policing with tricolor marking
- Congestion management
- IEEE 802.1ad provider bridging support
- Jumbo packet support (up to 15864 bytes)
- Cut through switching

The frame header pipeline is designed to process the following frame data:

- Source Port (7 bits)
- Source MAC Address (48 bits)

- Destination MAC Address (48 bits)
- VLANs (SVLAN and CVLAN) (24 bits)
- VLAN priority and CFI/DEI bit (4 bits)
- Ethernet Type (16 bits)
  - The frame Ethernet type is the first type field after VLAN and RLT tags.
- Type of IP packet (IPv4 or IPv6)
- Source IP (32 or 128 bits)
- Destination IP (32 or 128 bits)
- Tunneling Labels (32 bits) or MAC-in-MAC/TRILL Header (128 bits)
  - Only one or the other can exist in its given frame, not both.
- Layer 4 (L4) Source Port (16 bits)
- L4 Destination Port (16 bits)
- L4 4 Options (6 bits)
- L4 4 Protocols (8 bits)
- IP TOS (8 bits)
- IP TTL (8 bits)
- IP Flow ID (20 bits)
- Deep packet inspection:
  - Extra bytes extracted after the L4 decoding for IP packets and after L2 Ethernet type for non-IP packets are passed to the FFU for further processing.
- ISL tag:
  - Switch priority (4 bits)
  - Source GloRT (16 bits)
  - Destination GloRT (16 bits)
  - User info (8 bits)
  - Frame type

The frame processor produces the following information:

- All forwarding information, including a destination port mask, necessary for forwarding and multicast-replication of each frame.
- Data and directives needed for egress header modifications.

5.3.1 Frame Processing Pipeline

Figure 5-4 shows the basic pipeline structure of the FM5000/FM6000 frame processor. Processing begins at the parser, which receives the frame's first 112 bytes from the ingress crossbar and extracts relevant fields for the subsequent stages. All switching, routing, classification, and other forwarding decisions are made over a sequence of ingress stages up to the scheduler (SCHED). Once these decisions are made, the scheduler enqueues the frame in the shared memory (or possibly discards it) and later dequeues one or more copies for transmission. The final transformation of the frame takes place during egress transmission by the modify stage.

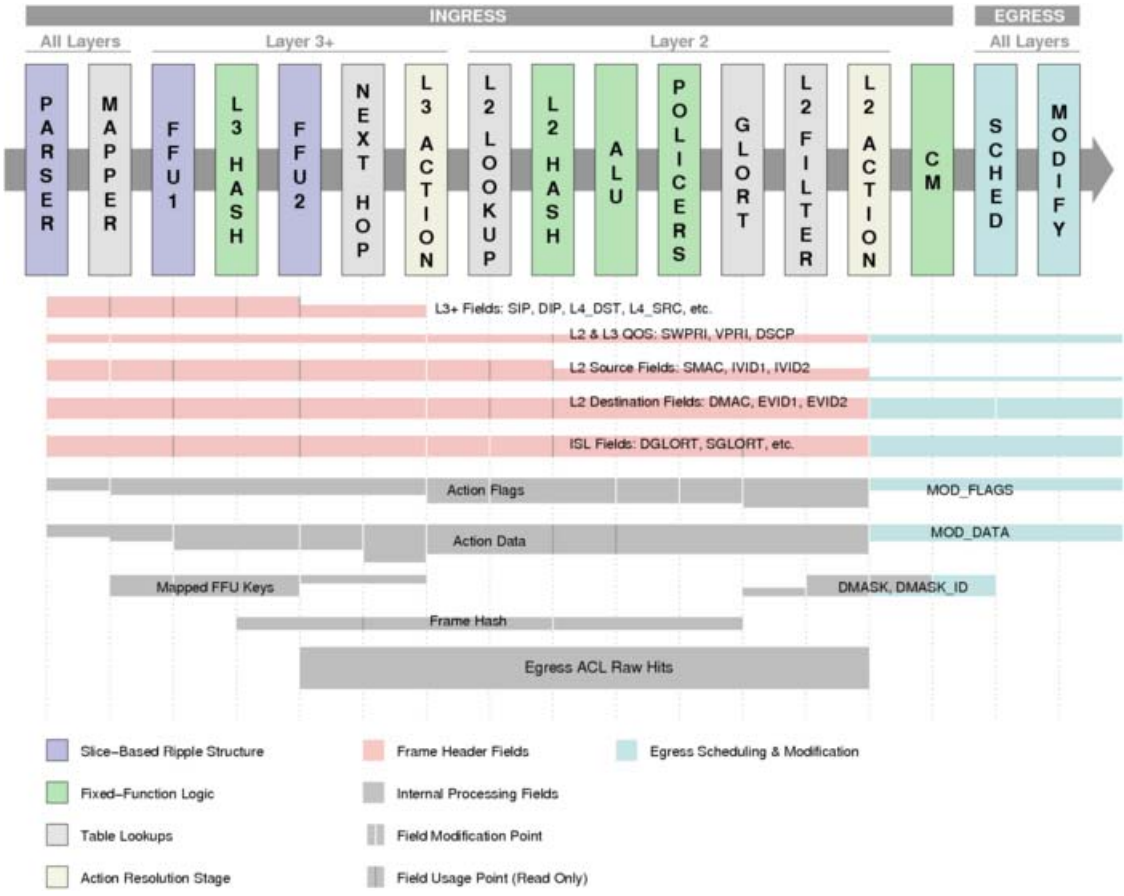


Figure 5-4 Frame Processing Pipeline

Figure 5-4 also shows the lifetime of various header and internal data fields in the pipeline. Generally, all L3+ and L2 ingress header fields are consumed during processing.

5.3.2 Frame Tail

Once each frame has fully ingressed, the following tail information is passed to the frame processor:

- Packet Length
- End of Frame Status (good CRC, bad CRC, symbol error, disparity error, oversize, undersize)

Some frame actions are defined at the time the header is processed (such as learning), but have to wait until the tail is received and the frame is confirmed to be a valid frame (good CRC and no error) before execution. These actions include:

- MAC address learning
- Policed update
- Packet-related interrupt to processor
- Ingress PAUSE execution
- Sampling frames for congestion notification
- Statistics counting

When possible, invalid frames are forwarded by the scheduler. However, in cut-through mode, a frame's tail might not arrive before egress transmission begins, making it impossible to discard the corrupted frame. In this case, the switch guarantees that such frames carry incorrect CRCs, and they can be counted specially by the frame statistics counters.

The EPL contains configuration options to select how to mark a frame (discard or forward) depending on the type of errors encountered. The default is set to discard each time the frame is erroneous for any reason.

5.3.3 Coloring

The EPLs are split into two colors (see table Table 5-2), and the maximum installed traffic for each color must not exceed 320 GbE. For EPL8s, the amount of active traffic per EPL8 (or a pair of EPL4s within that EPL8) is always limited to a maximum of 40 GbE. The maximum amount of active traffic per independent EPL4 is also limited 40 GbE. \

Table 5-2 Group Coloring

Group	Members	Maximum Bandwidth
Blue_EPLs	EPL[3], EPL[4], …..EPL[22], EPL[24]	320 GbE
Red_EPLs	EPL[1], EPL[2], …..EPL[21], EPL[23]	320 GbE

Figure 5-5 shows the color map for the switch.

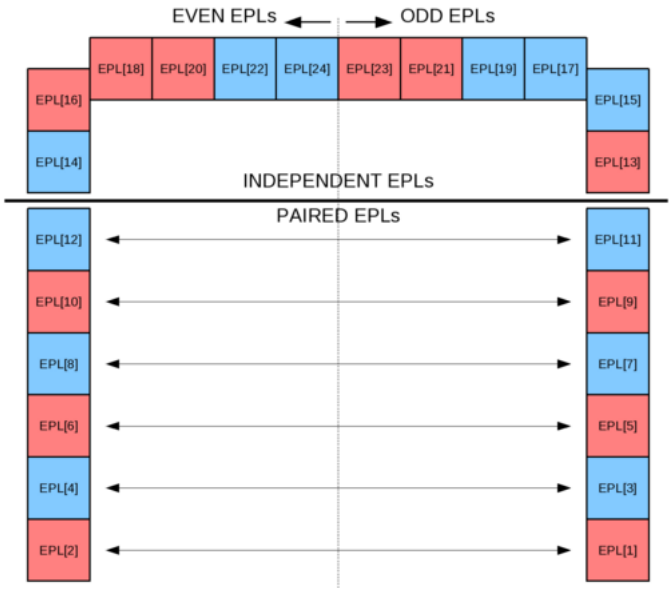


Figure 5-5 EPL Coloring

The color is related to bit 2 of the internal port number. If bit 2 of the internal port number is 0b, the port is blue, and if bit 2 is 1b the port is red.

5.4 Chip Management Logic

This section provides an overview of the FM5000/FM6000 management logic (see Figure 5-6). The management blocks are all connected through a fully provisioned 12 x 12 crossbar, which provides full bandwidth between multiple blocks simultaneously. As shown, the crossbar interface labels indicate whether a block act as an Initiator (I), Target (T) or both. All targets can be accessed by up to four initiators at the same time. The two exceptions are the frame handler manager, which can be accessed by up to five initiators, and the I^2C/CRM block, which can be accessed by up to 3 initiators.

The initiators issue MEM_READ and MEM_WRITE operations. The targets execute MEM_READ and MEM_WRITE operations. The MEM_READ operation requires a data response. The management targets contain scratch registers that are used to accumulate atomic data for entries larger than 32 bits. Each target has one atomic data scratch register per initiator, except for the I^2C/CRM block, which has only three sets of these registers. Each target also has a small FIFO that, when full, sends back pressure to the crossbar.

The crossbar back pressures an initiator if that initiator tries to push a transaction to a target that has a full FIFO.

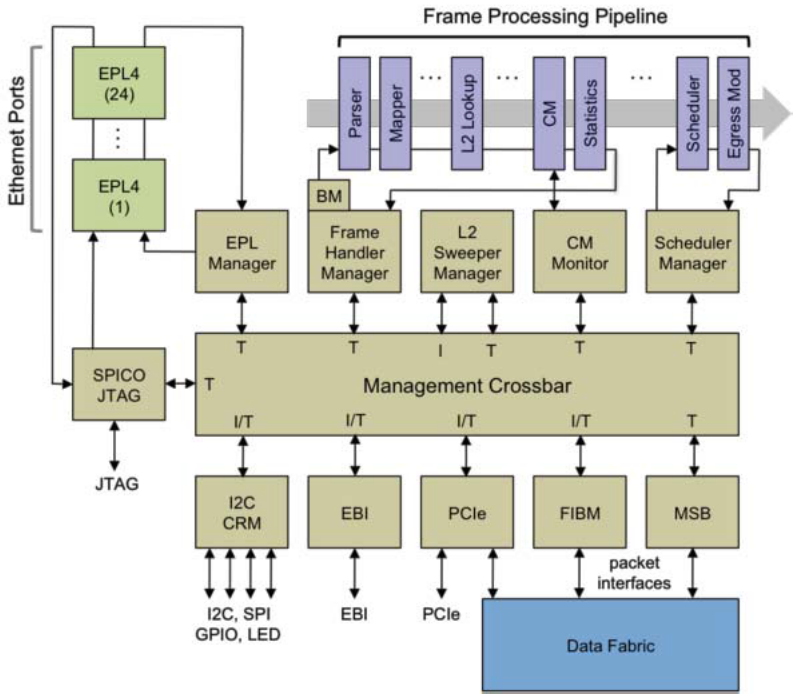


Figure 5-6 Management Logic

5.4.1 I²C/CRM Block

The Counter Rate Monitor (CRM) block is used to reduce the need for an external CPU to perform repetitive periodic operations in the switch. This block also contains I²C, SPI, GPIO and LED interfaces. This block acts as a initiator to control and monitor other targets, or as a target for other initiators. It also acts as an initiator and target at the same time.

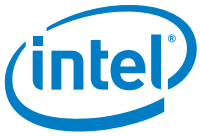
This block can be accessed by up to three management blocks simultaneously.

5.4.2 EBI Block

The EBI block provides a legacy parallel CPU interface to the switch and acts as a low latency management interface to external devices for applications such as congestion monitoring.

This block acts as an initiator to control and monitor other targets, as a target for other initiators, or as an initiator and target at the same time. The EBI block is accessed by up to four management blocks simultaneously.

The EBI block can be an initiator for I²C (if the FM5000/FM6000 is targeted by a remote I²C initiator to do a memory read or write) or for CRM, and uses a round-robin arbiter to determine which of these two devices can initiate a transaction.



5.4.3 PCIe Block

The PCIe block provides a high bandwidth CPU interface to the switch and has a packet interface to the data fabric that includes a data FIFO. This block also acts as an initiator to control and monitor other targets, as a target for other initiators, or as an initiator and target at the same time.

The PCIe block can be accessed by up to four management blocks simultaneously.

5.4.4 MSB Block

The Management Switch Bridge (MSB) block is a target packet interface to the data fabric. It is mainly used for packet transfers to the EBI block using a built-in FIFO.

This block can be accessed by up to four management blocks simultaneously.

5.4.5 SPICO JTAG Block

The SPICO micro-controller is a target device used to configure and monitor the SerDes portion of the EPL logic using a daisy-chained control loop. The operation of the micro-controller can be configured by other initiator devices connected to the management crossbar. It also contains the external JTAG interface.

This block can be accessed by up to four management blocks simultaneously.

5.4.6 EPL Manager

The EPL manager is used to configure and monitor the digital portion of the EPL logic. It uses a 16-bit bus that is connected to the EPL4 blocks through a daisy-chain control loop. The reads and writes are done through a special command that contains a port/register address followed by two 16-bit data words, providing 32-bit register access.

This is a target block that can be accessed by up to four management blocks simultaneously.

5.4.7 Frame Handler Manager

The frame handler manager is a target block and is accessed by up to five management blocks simultaneously. This is an important management block that is used to configure and/or monitor many of the key frame handler blocks. It operates through a 32-bit wide daisy-chain control loop using a command that contains a block/register address followed by one, two, three or four 32-bit data words. This enables up to a 128-bit wide register access.

When reading or writing to the various frame handler tables, these register access commands compete for part of the frame processing bandwidth. Because of this competition, a Bandwidth Manager (BM) block is used to ensure management traffic is best effort and does not impact the overall Ethernet frame processing bandwidth. Because of the daisy-chain bus structure and the potential delays caused by the bandwidth manager, posted writes are used by the initiators accessing this block.

5.4.8 L2 Sweeper Manager

The L2 sweeper manager is used for maintenance of the L2 lookup table providing hardware acceleration of common table-wide searches and entry transformations. It contains both an initiator and target interface to the management crossbar. The target interface is accessed by up to four initiators, which can configure and monitor the operation of the sweeper functions. The initiator interface uses the management crossbar to access the L2 lookup table through the frame handler manager.

This is the reason the frame handler manager supports up to five initiators simultaneously.

5.4.9 Congestion Management Monitor

The congestion management monitor is used to provide a direct, low-latency path to the registers in the congestion management block. This low latency is important for applications that use external devices such as CPUs or FPGAs that must receive very fast updates on the state of congestion in the switch.

This is a target block and can be accessed by up to four management blocks simultaneously.

5.4.10 Scheduler Manager

The scheduler manager is a target block accessed by up to four management blocks simultaneously. This block is used to configure and/or monitor several frame handler blocks including the schedulers, L3 multicast tables and the egress modification unit. It operates through a 32-bit wide daisy-chain control loop using a command that contains a block/register address followed by one, two, three or four 32-bit data words. This enables up to a 128-bit wide register access.

5.5 Parsing and Association

5.5.1 Parser

The FM5000/FM6000 parser processes each frame's incoming byte sequence according to programmed rules, mapping the frame's conditionally-formatted header fields into fixed hardware channels. At the hardware level, very little of this functionality is specific to Ethernet, IP, TCP, or any other protocol supported by the device. The parser's operation is defined almost entirely by microcode-specified transformations of its internal state and output channels.

At a high level, the parser can be viewed as an iterative state machine that consumes successive four-byte words of the frame on each iteration. In response to the incoming frame contents and its internal state, the parser maps the frame data to specific fields of a 88-byte output bus. It also records properties of interest about the frame by setting specific bits of a 40-bit output flags vector. Finally, it transforms an internal 32-bit register to preserve and update parsing state from one cycle to the next.

To maintain fully pipelined operation, the parser implementation is physically unrolled in hardware, with each parser slice representing one iteration of the parsing state machine as shown in Figure 5-7.

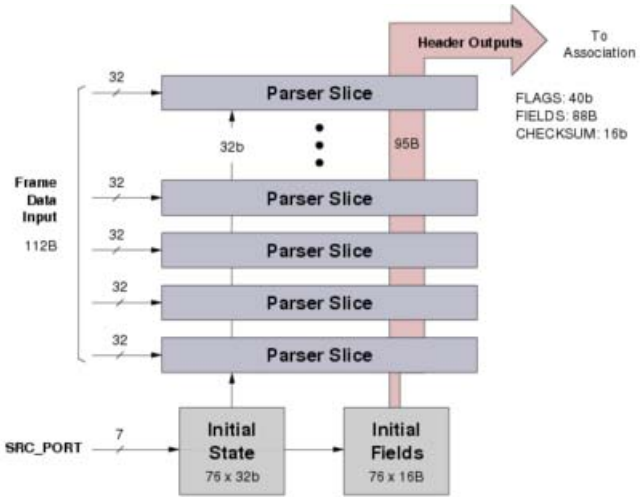


Figure 5-7 Parser Block Diagram

Each successive slice receives the next word of frame data and the prior slice's 32-bit state output. These two 32-bit quantities are combined into a 64-bit key and looked up in a TCAM, the result of which determines an action specifying the following: how to update the state vector, what flags to set, and how the frame data should be mapped to the slice's 88-byte output channel that ripples from slice-to-slice. At the end of the slice array, the 88-byte channel represents the frame's parsed header fields that are passed on to the association block.

The parser includes 28 slices, providing a maximum parsing visibility of 112 bytes into each frame. The 112 bytes includes any preamble bytes that the EPL might be configured to interpret as frame data.

The HEADER output structure contains the following channels:

- **HEADER.FLAGS** — 40 flag bits. Any slice can set any bit.
- **HEADER.FIELDS[0..43]** — 88 bytes of header fields, partitioned into 44 x 16-bit units. These output header field channels are divided into two categories:
  - — Initialized fields (FIELDS[0..3] and FIELDS[8..11]): Initialized with per-port defaults. These fields allow per-byte enables on overwriting the value.
  - — Power-gated fields (FIELDS[4..7] and FIELDS[12..43]): Not initialized. For power savings purposes, these fields are only active in the hardware if a slice assigns to them. If an unassigned field is referenced downstream in the pipeline, it assumes the value 0x0.
- **HEADER.CHECKSUM** — 16-bit ones-complement checksum. Calculated over specific 16-bit half-words of the frame payload, as identified by the parsing microcode.

The association stage that follows the parser maps the generic FIELDS channels to specific named channels representing their expected application usage in the frame processing pipeline.

5.5.2 Channel Initialization

As previously shown, the state vector's initial value is determined by the frame's physical ingress port. This enables different ports to support different parsing configurations, such as enabling F64-tagging or matching against a port-specific DMAC. It is expected that 8-16 bits of the state vector is likely available for such per-port configuration constants. Microcode would leave these bits unmodified through the slice array.

Note: The source port itself is not automatically included in the state vector, although the initial state table could be configured to include it at the cost of seven bits.

Eight of the 44 16-bit FIELDS channels are also initialized with values from a source port indexed table.

This mechanism is provided as a way to associate port-dependent default values with optional header fields such as VLAN IDs and ISL tag values. The other 36 FIELDS values are initialized to zero at the input to the slice array.

5.5.3 Parser Slice

Figure 5-8 shows the data flow and processing structures within each parser slice.

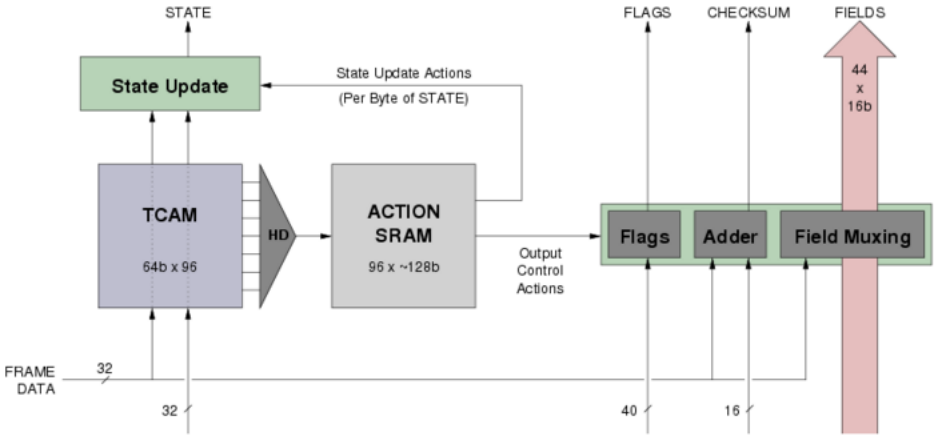


Figure 5-8 Parser Slice Data Flow and Processing

The CAM and RAM each contain 128 entries. Of all matching entries in the CAM, the highest-numbered one determines the index looked up in the action RAM. If there is no match, entry 0 is the hit index.

The action entry determined by this index specifies a set of state and header channel transformations (entry 0 must contain the default action when there is no hit). The state transformation action consists of four 12-bit operations, one for each byte (STATE8) of the state vector. Each STATE8 operation specifies one of four transformations:

Absolute GOTO	STATE8: = immediate
Relative GOTO	STATE8: = STATE8 + immediate
OVERWRITE	STATE8: = FRAME_DATA[8*n+7:8*n] + immediate
OVERWRITE2	STATE8: = FRAME_DATA[8*n+7:8*n]*2 + immediate

On the output header transformation side, the following set of actions are supported:

- **Flags** — Set any of the general-purpose bits in FLAGS[37:0]. Clearing bits are not supported.
- **Header Length Assertion** — In addition, control is provided for the special-purpose IncompleteHeader flag (FLAGS[38]): Each action specifies a byte count that is interpreted as an assertion on the number of FRAME_DATA bytes that must be valid. If the frame ends unexpectedly and the number of valid bytes does not match this value, the IncompleteHeader flag is set and parsing terminates.
- **Checksum** — Add FRAME_DATA[31:16] and/or FRAME_DATA[15:0] to CHECKSUM.
- **Fields Assignment** — Each 16-bit halfword from FRAME_DATA can be assigned to any FIELDS channel.
  - — Before assignment, each halfword might be barrel-shifted by some multiple of four bits.
  - — Assignment is enabled on a per-byte basis. Any unassigned bytes of any FIELDS channel is propagated to the next slice unmodified.
- **Parsing Termination** — This action might specify that parsing is complete, and no more frame bytes need to be examined. This is to optimize latency, so that the parser need not wait for the remaining bytes of the frame (though since the parser produces output in frame order, there might still be a delay if the previous frame needed to be parsed more deeply).

This is illustrated in Figure 5-9.

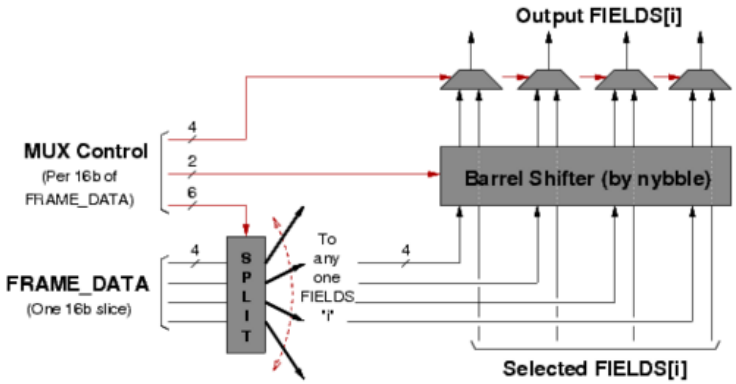


Figure 5-9 Parser Muxing Dataflow per FRAME_DATA Halfword

5.5.4 Parser Byte Ordering

Incoming packets are simple arrays of bytes. In most protocols, such as 802.3 and TCP/IP, the larger structures within the packet are transmitted most significant byte first. To facilitate software handling of the different fields, the parser follows this convention and maps incoming bytes into the CAM and output multiplexers with the first byte received loaded into the most significant byte of those structures. Output multiplexers can be programmed to swap the bytes (by rotating by eight bits) if required but normally does not do so.

This is illustrated in Figure 5-10. Note that the states are encoded differently.

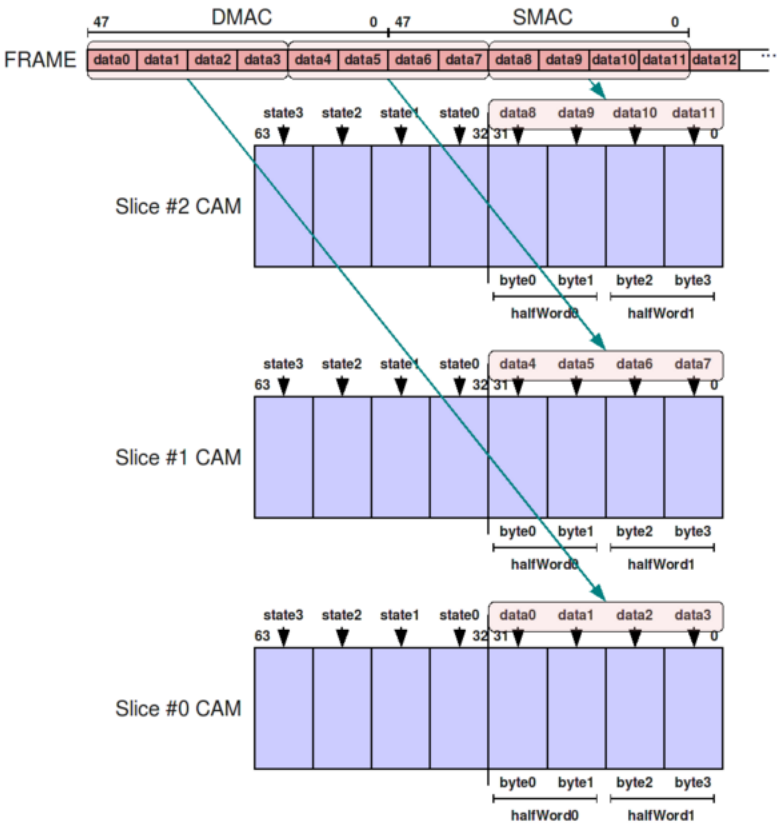
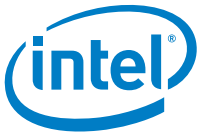


Figure 5-10 Big/Little Endian Parsing

Figure 5-10 does not include the shift transformation that can optionally be applied to each slice's FRAME_DATA parsing window. See the ShiftNextSlice action field listed in Table 5-3 for an explanation of this transformation.



5.5.5 Action Encoding

Each parsing slice's actions are encoded in its Action SRAM listed in Table 5-3.

Table 5-3 Parser Action SRAM Encoding

Field	Width	Description
StateOp0	2	Encodes state transformation operation per STATE8 byte of state vector. For N=0..3: 0 = STATE8 := STATE8 + StateValueN 1 = STATE8 := StateValueN 2 = STATE8 := FRAME_DATA[8 ·M+7:8 ·M] + StateValueN 3 = STATE8 := FRAME_DATA[8 ·M+7:8 ·M] ·2 + StateValueN All arithmetic is mod 256. The FRAME_DATA byte index M is calculated as: M=(N+StateFrameRot)%4 where StateFrameRot is a value 0..3.
StateOp1	2	
StateOp02	2	
StateOp3	2	
StateValue0	8	Immediate value for use with StateOp.
StateValue1	8	
StateValue2	8	
StateValue3	8	
StateFrameRot	2	Byte rotation amount to apply to the slice's frame data prior to applying state transformations.
SetFlags	38	Specifies flag bits to be set. FLAGS : = FLAGS | SetFlags. ***Note:*** Designers cannot directly set these flags: IncompleteHeader and ParityError.
Halfword0Dest	6	Specifies the index of the 16-bit output FIELDS channel to which the top (Halfword0) and bottom (Halfword1) sixteen bits of the slice's frame data is mapped. If Halfword0Dest = Halfword1Dest, assignments from the upper 16 bits of frame data have higher precedence.
Halfword1Dest	6	
Halfword0Rot	2	Specifies the nibble rotation to apply to the top (Halfword0) and bottom (Halfword1) 16 bits of the slice's frame data (one of four rotations that is applicable to both of the half-word's byte assignments.) Not supported for the top 8 parsing slices.
Halfword1Rot	2	
Byte0Enable	1	For each post-rotated byte of frame data, specifies whether to overwrite the corresponding output data FIELDS as indexed by Halfword XDest.
Byte1Enable	1	
Byte2Enable	1	
Byte3Enable	1	
Halfword0Add	1	Specifies whether the bottom and top 16 bits of frame data are added to CHECKSUM.
Halfword1Add	1	
ShiftNextSlice	3	Specifies an additive byte shift offset to apply to the next slice's frame (0..7). Slice number i sees the following four-byte sequence of frame data: `FRAME_DATA = FRAME_BYTES[4*i+3-window_shift:4*i-window_shift]` This is also referred to as the slice's parsing window. The three-bit window_shift state is propagated from slice-to-slice. By setting ShiftNextSlice to a non-zero value, the next slice's window_shift is advanced by the specified amount (modulo 8).
LegalPadding	2	Specifies the number of valid bytes of frame data (of four) that are required by this slice. The quantity is encoded in terms of padding bytes, or 4-num_valid_bytes. Hardware evaluates the following condition: `4-LegalPadding <= num_valid_bytes` If the test fails, the IncompleteHeader flag is set (bit 37), indicating that the frame ended prematurely. ***Note:*** The frame payload presented to the parser might include the final CRC word.

Table 5-3 Parser Action SRAM Encoding (Continued)

Field	Width	Description
TerminateAllowd	1	Set if the frame might legally terminate in the slice's parsing window, or in any un-parsed bytes that might precede the next slice's parsing window. ***Note:*** The latter condition might arise when window_shift is advanced beyond 7. If not set, and the next slice's parsing window contains no valid bytes, the IncompleteHeader flag is raised.
Terminate	1	Specifies that parsing must terminate at this slice. No further changes to FIELDS, FLAGS, or CHECKSUM takes effect following any rule that sets this bit to 1b.

The mapping of frame data to output FIELDS channel is precisely defined as follows:

```
data16[0] := BarrelShiftLeft({frame_data[0],frame_data[1]}, 4·Halfword0Roll)
data16[1] := BarrelShiftLeft({frame_data[2],frame_data[3]}, 4·Halfword1Roll)

FIELDS[Halfword0Dest][15:8] := Byte0Enable ? data16[0][15:8] : FIELDS[Halfword0Dest][15:8]
FIELDS[Halfword0Dest][7:0]  := Byte1Enable ? data16[0][7:0]  : FIELDS[Halfword0Dest][7:0]
FIELDS[Halfword1Dest][15:8] := Byte2Enable ? data16[1][15:8] : FIELDS[Halfword1Dest][15:8]
FIELDS[Halfword1Dest][7:0]  := Byte3Enable ? data16[1][7:0]  : FIELDS[Halfword1Dest][7:0]
```

5.5.6 Header Flags

Three bits of HEADER.FLAGS have specific fixed-function definitions within the parser (see Table 5-4.)

Table 5-4 Parser Header Flag Definitions

Flag	Bit	Description
ChecksumError	37	Set in microcode to enable checksum testing. Cleared if the final ones-complement sum of all halfwords identified by the Halfword{0,1}Add actions equals 0xFFFF; otherwise, remains set, indicating a bad checksum.
IncompleteHeader	38	Indicates that a slice's header length assertion failed.
ParityError	39	Indicates that a hardware parity error was detected in one of the slice SRAMs.

All other flag bits are set exclusively under microcode control using the SetFlags action. However, certain ACTION_FLAGS bits that originate as parser flags do have specific interpretation downstream in the pipeline.

By setting those flags, the parsing microcode can effectively control the operation of the relevant downstream fixed-function logic. These flag bits are described in Section 5.5.9.

5.5.7 Association Named Channels

The association stage is a thin layer of fixed-function mapping rules at the output of the parser. For the most part, this association function involves no more than mapping the parser's generic FIELDS channels (and in some cases, sub-fields of these channels) to specific named hardware channels. The only logical operations applied by the association stage involve the Quality-of-Service (QoS) fields as described in the sections that follow.

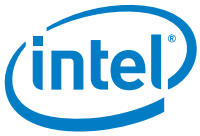


Table 5-5 lists the fixed mapping between the parser FIELDS outputs and the channels referenced downstream in the frame processing pipeline. Channels with a yellow background have some fixed-function interpretation by subsequent stages in the pipeline. The specific handling rules of all other fields are determined only by microcode and register configuration.

Table 5-5 Parser Fixed Mapping

Channel	Width	Parser FIELDS Channel Source[1]	Notes[2]
SRC_PORT	7		
QOS	24	0/11:8 = ISL_PRI 1/15:12 = L2_VPRI1 2/15:12 = L2_VPRI2 16/7:0 = L3_PRI 11/11:8 = W4	All QoS-related fields with special association rules. Contains the following fields that are derived from the frame header and per-port defaults. ***Note:*** All of these mappings from specific header fields are microcode-dependent. ISL_PRI (4 bits) — Priority from ISL tag (formerly Switch Priority in the FM4000 series). L2_VPRI1 (4 bits) — Priority (+CFI bit) from outer VLAN tag. L2_VPRI2 (4 bits) — Priority (+CFI bit) from inner VLAN tag. L3_PRI (8 bits) — Priority (TOS or traffic class) from IP header. ***Note:*** DSCP field is L3_PRI[7:2], CU is L3_PRI[1:0]. W4 (4 bits) — Generic priority available for configurable use. For example, might be assigned from an MPLS tag's EXP field.
ISL_FTYPE	3	0/15:14	FTYPE from the ISL tag. Has no fixed-function hardware interpretation.
ISL_VTYPE	2	0/13:12	
ISL_USER	8	0/7:0	
L2_VID1	12	1/11:0	Outer or S-Tag VID (source in same FIELD as QOS.L2_VPRI1).
L2_VID2	12	1/11:0	Inner or C-Tag VID (source in same FIELD as QOS.L2_VPRI2).
ISL_SGLORT	16	3	Guaranteed to be non-zero (by microcode).
ISL_DGLORT	16	4	
L2_DMAC	48	7 = L2_DMAC[47:32] 6 = L2_DMAC[31:16] 5 = L2_DMAC[15:0]	
L2_SMAC	48	14 = L2_SMAC[47:32] 13 = L2_SMAC[31:16] 12 = L2_SMAC[15:0]	
L2_TYPE	16	15	EtherType
L3_FLOW	20	(16/15:12,17/15:0)	Flow label (IPv6) (>>4 rotation of IPv6 header bytes 0..1 expected to align traffic class with FIELDS16[16][7:0], putting FlowLabel[19:16] in FIELDS16[16][15:12].)
L3_LENGTH	16	18	Total IP length (IPv4) or IP payload length (IPv6).
L3_TTL	8	19/15:8	
L3_PROT	8	19/7:0	Refers to the L4 protocol number (specified by the L3 header).

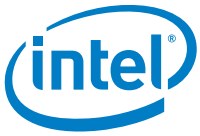
Table 5-5 Parser Fixed Mapping (Continued)

Channel	Width	Parser FIELDS Channel Source[1]	Notes[2]
L3_SIP	128	33 = L3_SIP[127:112] 32 = L3_SIP[111:96] 39 = L3_SIP[95:80] 38 = L3_SIP[79:64] 37 = L3_SIP[63:48] 36 = L3_SIP[47:32] 21 = L3_SIP[31:16] 20 = L3_SIP[15:0]	IPv4 SIP goes in L3_SIP[31:0]
L3_DIP	128	33 = L3_DIP[127:112] 32 = L3_DIP[111:96] 39 = L3_DIP[95:80] 38 = L3_DIP[79:64] 37 = L3_DIP[63:48] 36 = L3_DIP[47:32] 21 = L3_DIP[31:16] 20 = L3_DIP[15:0]	IPv4 SIP goes in L3_DIP[31:0]
L4_SRC	16	24	
L4_DIST	16	25	
FIELD16[A..I]	9 x 16-bits	26..27 = FIELD16{A,B} 8..11 = FIELD16{C,D,E,F} 40..42 = FIELD16{G,H,I}	Leftover generic header fields available for deep inspection and other configurable uses. FIELD16A (FIELDS[8]) and FIELD16B (FIELDS[9]) have special significance to the FFU. Specifically, they map to keys that can be assigned from action outputs at the Remap point midway through the slice array: FIELD16A[7:0] = LABEL8A FIELD16A[15:8] = LABEL8B FIELD16B = LABEL16 The FIELD16{A..I} channels have specific fixed-function interpretation when the parser sets the PAUSE_Frame flag (bit #34). For PAUSE frames, these channels are used to communicate the pause time(s) and per-class enable vector to the congestion management stage for PAUSE reception handling. ***Note:*** The FIELD16{C..F} initial values are programmable per port. FIELD16I is used by the QOS mapper, and is not available as an FFU key.
Unused	16	43	

1. Notation is n/hi:lo, indicating the corresponding channel is set from the parser HEADER.FIELDS[n][hi:lo] output.
2. The FIELDS[0..3] and FIELDS[8..11] initial values are programmable per port; FIELDS[4..7] and FIELDS[12..43] initial values are zero. FIELDS retains their value unless a particular field is changed by the parser.

In hardware, these fields are all named as previously listed. These names should be viewed as identifying their default interpretation by the frame processor. To support certain features (such as MAC-in-MAC or extra-deep inspection), some of these fields (like L3_DIP) might be overloaded for these alternate purposes. The fixed number of bits available in hardware for header field extraction represents one constraint on the number of frame processing features that the FM5000/FM6000 can simultaneously support.

These fields as parsed-and-associated and are referred to by the names previously mentioned. For example, any reference to L2_DMAC always means the value associated with the ingress frame. L3 action resolution has the capability to select a different DMAC for the egress frame(s), but if it does so, it annotates this new DMAC into a generic field within the ACTION_DATA channel. The L2_DMAC channel name always refers to the value produced at the output of parsing and association.



5.5.8 QoS Handling

The parser extracts and associates the following QoS-related fields:

- **ISL_PRI** (4 bits) — Primary QoS classification in the pipeline.
  - In the mapper stage, ISL_PRI can be mapped from each of the other priority fields with configurable precedence.
  - Association rules in the mapper are defined by a configurable CAM/RAM structure.
  - Nominally taken from FIELDS[0][11:8].
- **L2_VPRI1** (4 bits) — Primary L2 priority.
  - Assigned from FIELDS[1][15:12] via a per-port mapping table.
- **L2_VPRI2** (4 bits) — Secondary L2 priority.
  - Assigned from FIELDS[2][15:12] via a per-port mapping table.
- **L3_PRI** (8 bits) — L3 priority
  - Taken from FIELDS[16][7:0].
  - Nominal interpretation: TOS (IPv4) or traffic class (IPv6).
  - Top six bits (L3_PRI[7:2]) interpreted and transformed later in the pipeline as DSCP.
- **W4** (4 bits) — Extra 4-bit QoS field available for configurable use.
  - Assigned from FIELDS[11][11:8] via a per-port mapping table.
  - Potential application-dependent definitions:
    - MPLS EXP field
    - PBB BVLAN tag's PRI+CFI field.

In later stages in the pipeline, these QoS fields are always transformed together as a group. The 24-bit structured channel is referred to as QoS. As with other structured channels, its sub-channels are referenced using a dotted notation (such as QOS.ISL_PRI).

5.5.9 Action Flags

In addition to the fixed header fields, the parser's flag bits seed the ACTION_FLAGS channel that propagates throughout the subsequent stages of the frame processing pipeline. These flags convey specific properties of interest to downstream logic. In particular the L3 and L2 action resolution stages respond to the flags by applying different processing actions to the frame. The width of the ACTION_FLAGS vector begins as the parser's 40 bits of HEADER.FLAGS and expands from stage to stage in the pipeline as more conditions are evaluated.

To support basic Ethernet switch/router functionality, the parser microcode is expected to define the following set of flag bits (or one very similar to it). While the parsing stage itself only defines three fixed-function flag bits (37..39), other specific flags bits are interpreted by downstream fixed-function logic. These are highlighted in yellow in Table 5-6. All other flag definitions should be interpreted as suggestions for microcode programming.

Table 5-6 Parser Action Flags

Flag	Bit	Notes
Unbound	0	Available for configurable use.
ISL_RX_Tagged	1	Indicates that the ingress frame carried an ISL tag.
ISL_Type[1:0]	3:2	If ISL_RX_Tagged = 0x1, indicates the ISL tag format (F32, F64, F96).
ISL_FType[1:0]	5:4	Indicates the frame type as specified by the ISL tag or assigned.
L2_VLAN1_RX_Tagged	6	Has S-TAG; dependent on EtherType match.
L2_VLAN2_RX_Tagged	7	Has C-TAG; dependent on EtherType match.
L3_IsIPv4	8	Dependent on EtherType.
L3_IsIPv6	9	Dependent on EtherType.
ISL_IDGLORT_NonZero	10	Set if ingress DGLORT was non-zero.
ISL_ISGLORT_NonZero	11	Set if ingress SGLORT was non-zero.
L3_DIP_V4InV6	12	DIP[127:32] = 0xFFFF or DIP[127:32] = 0x0
L3-TTL_Expired	13	(ttl = 0x0 or ttl = 0x1).
L3_HeadFrag	14	Head fragment bit.
L3_DontFrag	15	Do not fragment bit.
L3_Options	16	Options present or not.
L3_Mcst	17	Derives from bit 40 of DMAC.
L3_Bcst	18	Derived by (DMAC = 1) comparison.
L3_Mcst	19	Hard-coded condition, depends on v4/v6 and top few bits of DIP.
PBB_RX_Tagged / MPLS_RX_Tagged[0]	20	Indicates whether the ingress frame was PBB tagged or the MPLS stack depth.
PBB_HeaderType / MPLS_RX_Tagged[1]	21	Identifies one of two PBB header types or indicates the MPLS stack depth.
L3_IPv6_HopByHop	22	
Unbound	23	Available for configurable use.
HdrOffsets[7:0]	31:24	Might encode L3 and L4 header offsets (measured in 32-bit words). L3AR provides a mux pathway to ACTION_DATA (and eventually MOD_DATA) so that these bits can be annotated in the egress frame(s). A likely encoding represents the L3 offset with HdrOffsets[2:0] and the L4 offset with HdrOffsets[7:3].
L2_VID1_Translate	32	Used by the mapper in some configurations. See Section 5.6.2, “VID Tables”
L2_VID2_Translate	33	Used by the mapper in some configurations. See Section 5.6.2, “VID Tables”
PAUSE_Frame	34	Set on pause frames. Instructs the congestion management stage to interpret FIELD16B as a pause time to apply to SRC_PORT. See Section 5.18.7, “Pause Frame Reception”
PAUSE_CBPFrame	35	Set on class-based pause frames, only relevant when PAUSE_Frame is also set. Causes FIELD16A and FIELD16{C..I}, in addition to FIELD16B, PAUSE_CBPFrame 35 to communicate PAUSE-related information for class-based PAUSE handling. See Section 5.18.7, “Pause Frame Reception”
Unbound	36	Available for configuration use.
ChecksumError	37	Indicates that the parser's header checksum calculation did not sum to 0xFFFF as required.
IncompleteHeader	38	Indicates that a slice’s header length assertion failed.
ParityError	39	Hardware parity error.

Note: These flags have mixed-case names since they refer to constant bit index numbers within this channel.

5.6 Mapper

The mapper stage contains a number of mapping structures, such as maps, CAMs, and range compares. It serves a number of purposes:

- Recognizes particular header field values of interest (such as IEEE reserved DMAC addresses, virtual router DMAC addresses, etc.).
- Compresses wide header fields into smaller identifier values when structure is present in those fields (such as TCP port ranges).
- Remaps the VID1 and/or VID2 (VLAN ID) numbering spaces for software purposes.
- Tags particular header field values of interest with configurable or fixed-function properties (such as the routable property associated with VLANs, source ports, and DMACs).
- Classifies each frame by assigning it a scenario value, used by the FFU.

The mapper stage transforms the raw frame header fields produced by the parser into a more compact form for FFU and action resolution processing. Without these compression tables, far more TCAM resources are required to implement commonly required matching conditions. Some mapper outputs are available only to the downstream action resolution stages. All other outputs are sent directly to the FFU, typically for use as TCAM and BST input keys.

Some of the CAMs in the mapper have two or sometimes even three seemingly redundant instances. The motivation for this redundancy is to provide independent classification sets for different software layers. For example, the three DMAC_CAMs are provided for independent use by microcode, ACLs, and routing.

In the structure diagrams shown in Figure 5-11, blue outputs correspond to flag bits that are aggregated in the ACTION_FLAGS channel. Red outputs are available only to the downstream action resolution stages: L2 Action Resolution (L2AR) and L3 Action Resolution (L3AR). All other outputs are sent directly to the FFU, typically for use as TCAM and BST input keys.

5.6.1 SRC_PORT_TABLE

This 76- x 31-bit mapping table produces three eight-bit ID outputs and two flag bits from the frame's physical source port number.

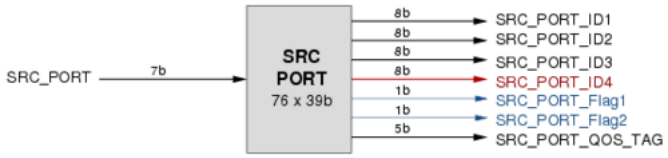


Figure 5-11 Source Port Mapper

The mapped SRC_PORT_ID values identify port classes that share common sets of FFU rules. Two of these IDs (ID1 and ID2) are available as keys to the FFU's primary CAM. The lower four bits of SRC_PORT_ID2 can also be referenced in the initialization of the FFU_DATA.W24 field.

The 8-bit SRC_PORT_ID3 is available in the FFU's SCENARIO_CAM and is intended for power-gating entire slices that are known to contain no applicable rules. SRC_PORT_ID4 is included in the L3AR CAM key.

The SRC_PORT_Flag{1,2} bits are available downstream in the pipeline as bits in the ACTION_FLAGS channel. In particular, SRC_PORT_Flag1 is available in the FFU TCAM slices' SCENARIO_CAM keys. Its expected use is to encode the L3-routed status of the ingress port. SRC_PORT_Flag2 has direct application within the mapper unit for the control of VID translation.

The 5-bit SRC_PORT_QOS_TAG value is used internally by the mapper's QOS_CAM structure. It is not propagated any further down the pipeline.

5.6.2 VID Tables

Two mapping tables are provided for classification and optional remapping of the VID1 and VID2 VLAN IDs produced by the parser.

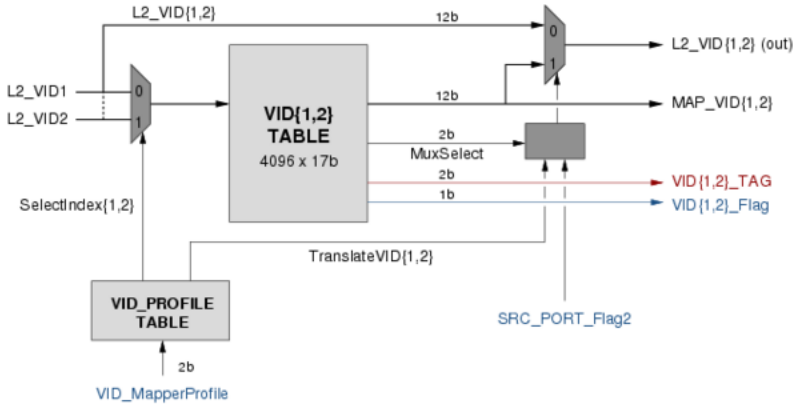


Figure 5-12 VID Mapping

The mapping behavior of these two tables is determined by a profile configured in MAPPER_VID_PROFILE_TABLE (see Table 5-7). The two-bit VID_MapperProfile field, set by the parser as ACTION_FLAGS[33:32], selects the mapping profile to apply to the frame.

The SelectIndex1 and SelectIndex2 bits in the profile select each table's input index. A value of 0 selects L2_VID1; a value of 1 selects L2_VID2. By default, VID1_TABLE is indexed by L2_VID1 and VID2_TABLE is indexed by L2_VID2.

The TranslateVID1 and TranslateVID2 bits in the VID_PROFILE_TABLE entry, one per VID table, controls the output muxing of the L2_VID{1,2} fields that is used downstream in the pipeline. The 2-bit MuxSelect field and the SRC_PORT_Flag2 bit also factors into this determination.

Table 5-7 Mapper VID Profile Table

MuxSelect	TranslateVIDx	SRC_PORT_Flag2	MUX Case
0	x	x	0
1	x	x	1
2	C	x	C
3	x	C	C

Note: "X" represents don't care values. "C" represents the value of the flag.

Regardless of the MuxSelect setting, both the final VID value and the MAP_VID value are available as FFU keys.

The VID1_TABLE and VID2_TABLE entries also specify TAG and flag values. The 2-bit TAG values are available as bits in the SCENARIO_CAM. The 1-bit flag values set bits in the ACTION_FLAGS channel, which are available for more flexible use in both L3AR and L2AR.

Flag bits are different from tag bits in that they can be redefined and overridden under software control by the action resolution SetFlags action. Flag bits can also control specific fixed-function logic in the pipeline, such as the L2_VIDx_Translate and SRC_PORT_Flag2 bits do in the mapper stage.

5.6.3 L2 CAM/RAM Mapping

Three CAM/RAM mapping structures are provided on the L2_DMAC and two on both the L2_SMAC channels and L2_TYPE. Each CAM/RAM structure produces 4-bit or 5-bit IDs that are available as keys to the FFU or action resolution stages. The highest-numbered matching entry in each TCAM determines the index to look up in the RAM, which gives the output ID number. If no TCAM entry matches, the ID from RAM entry 0 is returned.

The DMAC_RAM1 table is special in that it includes a 1-bit L2_DMAC_Flag in addition to the L2_DMAC_ID1 index. The flag bit sets the corresponding bit in ACTION_FLAGS, which is also available for muxing into the FFU's SCENARIO_FLAGS channel. The expected use for this flag is to indicate the L3-routed status of the DMAC. For example, it identifies the router's MAC address.

The DMAC-related structures are shown in Figure 5-13. The SMAC and TYPE pathways are similar, although neither includes any flag bits.

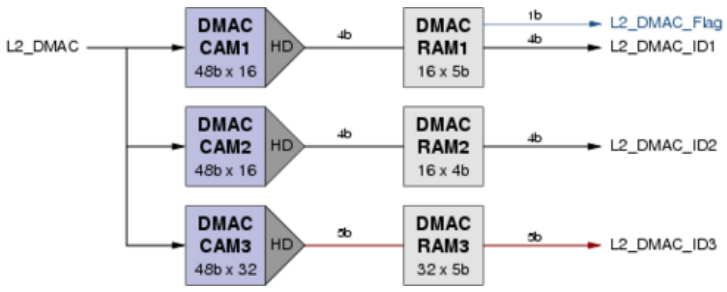


Figure 5-13 L2 mapping

The expected uses of these ID outputs are:

- **L2_DMAC_ID1**, **L2_DMAC_ID2**, **L2_SMAC_ID1** — Routing or ACLs
  - — There are no L2_SMAC_ID2.
- **L2_DMAC_ID3**, **L2_SMAC_ID3** — Microcode, such as:
  - — CPU MAC identification
  - — Matching against reserved IEEE MAC ranges
  - — GMRP, GVRP, LACP, other slow protocol traps
- **L2_TYPE_ID1** — ACLs
- **L2_TYPE_ID2** — Microcode, such as:
  - — Pause frame identification
  - — VCN/QCN frame identification

5.6.4 L3 CAM/RAM Mapping

A collection of CAM/RAM mapping structures are provided on the L3 header fields that commonly require exact and longest-prefix matching: DIP, SIP, and protocol. Three IDs are mapped from DIP and SIP each. Figure 5-14 shows DIP CAMs only, same circuit is duplicated for the SIP CAMs) and two IDs are mapped from the L3 protocol (L3_PROT).

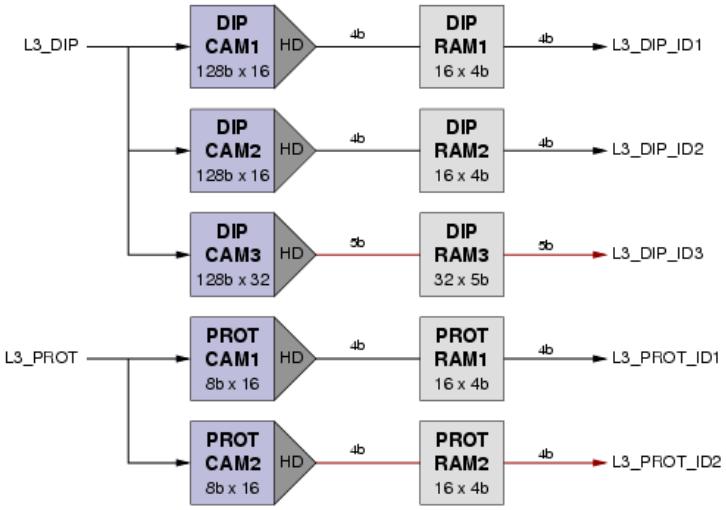
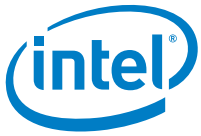


Figure 5-14 L3 Mapping

Anticipated uses of the DIP and SIP identifiers:

- **ID1** — Routing
- **ID2** — ACLs



- **ID3** — Identifies port backbone DMACs when DIP is overloaded for this purpose (PBB). Also available for general second-order parsing when the DIP and SIP fields are overloaded for other purposes.

The L3_PROT_IDs are intended for the following applications:

- **ID1** — ACL rule compression, especially in conjunction with the L4_SRC_COMPARE and L4_DST_COMPARE mapping structures.
- **ID2** — L4 traps in L3AR.

If no entry matches the input key in any of the L3 CAMs, an index value of zero is produced.

5.6.5 L3_LENGTH_COMPARE

The L3_LENGTH_COMPARE structure implements range-based binning of the IP length header field. It is configured with 16 lower-bound comparison values LowerBound[0..15]. The comparison structure generates its L3_LENGTH_BIN output as follows:

```
L3_LENGTH_BIN = 0
for i=0..15
    if (LENGTH_COMPARE[i].LowerBound ≤ L3_LENGTH)
       L3_LENGTH_BIN = LENGTH_COMPARE[i].Bin
```

The L3_LENGTH_BIN output is available as FFU and L3AR keys. It can be used to maintain statistics on user-defined length bins for MTU-based dropping decisions, or for length-dependent routing rules.

5.6.6 L4 Port Mapping

The L4_SRC_COMPARE and L4_DST_COMPARE structures provide similar range-based matching of the L4 source and destination port fields. They differ from the L3_LENGTH_COMPARE structure in the following respects:

- An additional exact-match test against L3_PROT is included in each comparison.
- An arbitrary 16-bit ID value is specified for the matching entry. If the entry is determined to match against the input L4_{SRC,DST}, the output L4_{SRC,DST}_ID value is assigned to the configured ID.
- If there is no match, the mapped output is set to the original input port value, rather than zero.
- Each entry has a configured valid bit which must be set to 1b for the entry to match.
- To match against arbitrary (lower or upper) ranges specified as pairs of entries located anywhere in the structure, an entry is only selected if the next entry does not match.

These properties enable the L4 port comparisons to support a variety of different usage models. For example, exact-match port comparisons can be entered alongside range-based comparisons, with only the mapped ID value then needed in FFU rules. For a given protocol, a list of single-sided range comparisons can be defined by entering LowerBound points in sorted order, or double-sided ranges can be defined by pairs of (lower or upper) entries entered in arbitrary order. In the latter usage model, the upper entry would have their valid bits set to zero.

The L4_SRC_COMPARE mapping function operates according to the following pseudo code:

```
L4_SRC_ID = L4_SRC
for i=0..63
    if (Valid[i] && LowerBound[i] ≤ L4_SRC && Protocol[i] == L3_PROT &&
            (LowerBound[i+1] > L4_SRC || Protocol[i+1] ≠ L3_PROT))
        L4_SRC_ID = ID[i]
```

The boundary-case index i=64 is evaluated as if LowerBound[64]=2^{16}. For example, LowerBound[64] > L4_SRC always evaluates true. The L4_DST_COMPARE function has an analogous definition.

5.6.7 FFU Initialization

The mapper is responsible for initializing all of the channels that propagate through the FFU slices. These include both the FFU's key input channels and its action output channels. Most keys are mapped in a fixed manner from the parser outputs and from the outputs generated by the mapping structures previously described. The fixed mapping from these fields to named FFU keys is specified in the FFU register section.

A handful of keys and all of the action data channels are initialized with a limited degree of static configurability, as described in the sections that follow.

5.6.8 SCENARIO_FLAGS

To minimize the SCENARIO_CAM key width, the FFU's scenario key includes only 16 of the ACTION_FLAGS' 45 flag bits. Each bit of the 16-bit SCENARIO_FLAGS is individually selected by the 96-bit SCENARIO_FLAGS_CFG register. This muxing is applied after all flag bits defined by the mapper structures have been set.

5.6.9 FFU Action Data

In many applications, the FFU action data channels must be initialized to meaningful values to properly handle the case that no action assigns a new value. Fields in the MAPPER_FFU_INIT register specify the static muxing of each channel's initialization source listed in Table 5-8.

Table 5-8 Mapper FFU Initialization Source

Field	Bits	Mux Select	Value	Input Source
FFU_DATA.W24	15:0	W24_MuxSelect	0	0 (constant)
			1	ISL_DGLORT
			2	ISL_SGLORT
			3	[SEC_PORT_ID2[3:0], L2_VID2][1]

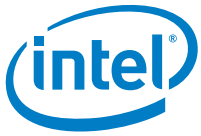


Table 5-8 Mapper FFU Initialization Source (Continued)

Field	Bits	Mux Select	Value	Input Source
FFU_DATA.W16A	11:0	W16A_MuxSelect1	0	0 (constant)
			1	L2_VID1
			2	L2_VID2
			3	MAP_VID1
			4	MAP_VID2
FFU_DATA.W16A	15:12	W16A_MuxSelect2	0	0 (constant)
			1	QOS.L2_VPRI1
			2	QOS.L2_VPRI2
FFU_DATA.W16B	11:0	W16B_MuxSelect1	0	0 (constant)
			1	L2_VID1
			2	L2_VID2
			3	MAP_VID1
			4	MAP_VID2
FFU_DATA.W16B	15:12	W16B_MuxSelect2	0	0 (constant)
			1	QOS.L2_VPRI1
			2	QOS.L2_VPRI2
FFU_DATA.W8A	7:0	W8A_MuxSelect	0	0 (constant)
			1	[QOS.ISL_PRI, QOS.W4]
			2	ISL_USER
			3	QOS.L3_PRI
FFU_DATA.W8b	7:0	W8b_MuxSelect	0	0 (constant)
			1	[QOS.ISL_PRI, QOS.W4]
			2	ISL_USER
			3	QOS.L3_PRI
FFU_DATA.W8c	7:0	W8c_MuxSelect	0	0 (constant)
			1	[QOS.ISL_PRI, QOS.W4]
			2	ISL_USER
			3	L3_TTL

1. L2_VID2 muxed by the VID mapping stage. In all cases, the top four bits of W24 is zero.

The FFU TAG channels are always initialized to zero.

5.6.10 QoS Mapping

A number of mapping functions are provided to reconcile frame QoS priorities across different protocol layers, VLANs, ports, and multi-chip switch domains. Generally, the ISL tag priority (ISL_PRI) is intended to be interpreted as the frame's canonical priority value. From this four bit value, the scheduling traffic class and shared memory partition are mapped prior to congestion management and egress scheduling policy evaluations.

The QoS mapping tables provided on ingress involve three stages of ISL_PRI determination:

1. Source port based normalization of all L2 priority fields. The L3 L3_PRI field is not normalized by port.
2. Layer-based normalization of all QoS fields.
3. CAM/RAM-based precedence selection of all QoS fields for the final ISL_PRI value. The parser's action flags and SRC_PORT_QOS_TAG are available as keys in this function.

Figure 5-15 shows the QoS mapping data flow.

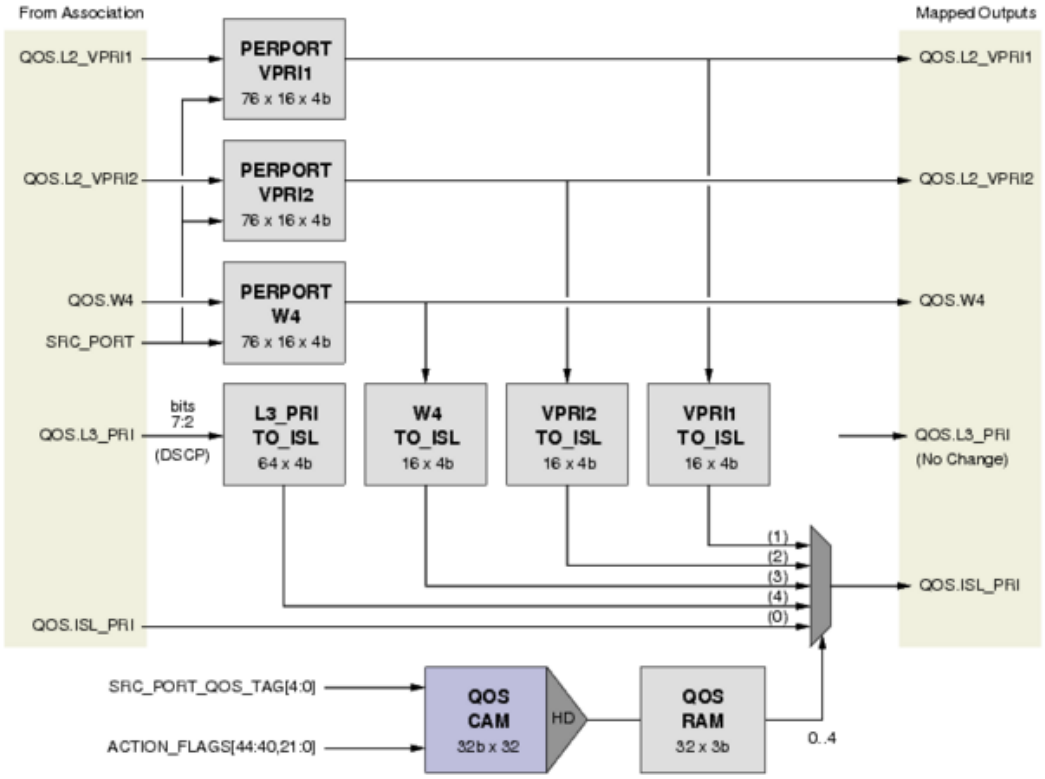


Figure 5-15 QoS Mapping

As with all other CAM structures in the FM5000/FM6000, if no QOS_CAM entry matches the input key, a value of zero is produced, which indexes QOS_RAM[0].

A second QOS_CAM/RAM structure (not shown in Figure 5-15) can be used to control an input assignment stage of the 4-bit QoS fields (L2_VPRI1, L2_VPRI2, and W4). Since the parser does not support masked assignment of fields smaller than eight bits, for some applications the hard-coded bit

assignments of the 4-bit QoS fields might be incorrect. To workaround such cases, the parser might extract the appropriate byte of the header to the FIELD16I channel, and the QOS_CAM/RAM mapping can be used to select the final assignment of these QoS nibble fields from the appropriate nibble of FIELD16I.

Both CAM/RAM structures share the same input key. In the register definitions, QOS_CAM1 and QOS_RAM1 refer to the CAM/RAM structure controlling the QoS nibble field assignments, while QOS_CAM2 and QOS_RAM2 control the ISL_PRI assignment.

Table 5-9 lists the mux pathways available for these QoS field assignments. The L2_VPRI1_Select, L2_VPRI2_Select, and W4_Select are each 2-bit fields mapped from QOS_RAM1.

Table 5-9 QoS Nibble Field Mux Cases

QoS Field	QOS_RAM1.FIELD_Select Case (per field)			
	0	1	2	3
L2_VPRI1	As parsed	FIELD16I[15:12]	FIELD16I[11:8]	FIELD16I[7:4]
L2-VPRI2	As parsed	FIELD16I[15:12]	FIELD16I[11:8]	FIELD16I[7:4]
W4	As parsed	FIELD16I[15:12]	FIELD16I[7:4]	FIELD16I[3:0]

These assignments are made prior to all uses of the QoS fields within the mapper. For example, QOS_PER_PORT_VPRI1, QOS_L3_PRI_TO_ISL, etc.

5.6.11 Mapper Outputs

Table 5-10 lists all new, non-flag outputs produced by the mapper. also lists the availability of these fields downstream in the pipeline.X

Table 5-10 Mapper Outputs

Field	Width	Downstream Consumers				
		FFU (S-key)[1]	FFU (key)	L3AR (key)	L3AR (mux)	L2AR (key)
SRC_PORT_ID1	8	No	Yes	No	No	No
SRC_PORT_ID2	8	No	Yes	No	No	No
SRC_PORT_ID3	8	Yes	No	No	No	No
SRC_PORT_ID4	8	No	No	Yes	No	No
L2_VID1	12	No	Yes	No	Yes	No
L2_VID2	12	No	Yes	No	Yes	No
MAP_VID1	12	No	Yes	Yes	No	No
MAP_VID2	12	No	Yes	Yes	No	No
VID1_TAG	2	Yes	No	No	No	No
VID2_TAG	2	Yes	No	No	No	No
L2_DMAC_ID1	4	Yes	Yes	No	No	No
L2_DMAC_ID2	4	No	Yes	No	No	No
L2_DMAC_ID3	5	No	No	Yes	No	Yes

Table 5-10 Mapper Outputs (Continued)

Field	Width	Downstream Consumers				
		FFU (S-key)[1]	FFU (key)	L3AR (key)	L3AR (mux)	L2AR (key)
L2_SMAC_ID1	4	No	Yes	No	No	No
L2_SMAC_ID3	5	No	No	Yes	No	Yes
L2_TYPE_ID1	4	No	Yes	No	No	No
L2_TYPE_ID2	4	No	No	Yes	No	Yes
L3_DIP_ID1	4	No	Yes	No	No	No
L3_DIP_ID2	4	No	Yes	No	No	No
L3_DIP_ID3	5	No	No	Yes	No	No
L3_SIP_ID1	4	No	Yes	No	No	No
L3_SIP_ID2	4	No	Yes	No	No	No
L3_SIP_ID3	5	No	No	Yes	No	No
L3_PROT_ID1	4	No	Yes	No	No	No
L3_PROT_ID2	4	No	No	Yes	No	No
L4_SRC_ID	16	No	Yes	No	No	No
L4_DST_ID	16	No	Yes	No	No	No
SCENARIO_FLAGS	16	Yes	Yes	No	No	No
QOS	24	No	Yes	Yes	Yes	Yes

1. -SCENARIO_CAM key.

Not listed in Table 5-10 are the FFU_DATA.* initialized channels that are consumed immediately by the FFU.

All mapper flag bits added to ACTION_FLAGS are listed in Table 5-11.

Table 5-11 Mapper Action Flags

Flag	Bit	Special Handling
SRC_PORT_Flag1	40	Available in FFU's SCENARIO_CAM keys.
SRC_PORT_Flag2	41	Controls VID1 and VID2 translation from VIDx_TABLEs.
L2_DMAC_Flag	42	Available in FFU's SCENARIO_CAM keys.
L2_VID1_Flag	43	Available in FFU's SCENARIO_CAM keys.
L2_VID2_Flag	44	Available in FFU's SCENARIO_CAM keys.

5.7 Frame Filtering and Forwarding Unit (FFU)

The FM5000/FM6000 FFU is a parallel, flexible frame classification unit characterized by the following high-level properties:

- Capability to compare up to 24,570 36-bit CAM rules against any frame. Up to 24 of these rules (one per 1024 set) can apply an action to the frame. Each action can take effect in parallel, changing different properties of the frame processing.
- Rules in each 1024-entry set (referred to as a slice) can be arbitrarily chained together to form slice chains with wider input keys. For example, chaining two slices together produces a 1024-entry set with a 72-bit key.
- Capability to compare up to 65,536 32-bit Binary Search Tree (BST) rules against any frame. BST rules are resolved to a one-to-four matching rule that can take the same action that an FFU CAM slice can.
- Rules in the BST can be constructed with 32, 64, 96 or 128-bit wide keys. Each BST slice can be partitioned into up to 16 partitions that share the same key mux and function.
- Based on the results of the first 12-Kb CAM rules, a remap stage can reassign certain key fields for comparisons in the subsequent 12-Kb CAM rules.
- Each rule might modify the contents of one of seven generically defined data channels whose meanings are derived from the microcode. Each rule can also set the encoding of the route and switch flags to control the inputs to the next hop table.
- Each rule can assign a tag to one of two independent tag channels, for later use by L3AR.

The FFU is primarily intended to implement Access Control Lists (ACLs) and L3 routing tables. However, the highly flexible interpretation of its action outputs by the L3AR stage enables many other features to be controlled by the FFU's rules. Examples include flow-based policing, encapsulated MAC address lookups, MPLS label matching, priority assignment, and control over egress header transformations.

Note: The FM5000/FM6000 only supports 12,288 rules and does not include the BST or the remapping stage.

5.7.1 Overview

The FM5000/FM6000 FFU has a slice-based structure, as shown in Figure 5-16. In general terms, the FFU receives a large collection of header-derived input keys from the mapper (96 bytes) and produces a small set of output actions by matching those keys against its internal rules. The keys are propagated from input-to-output for continued processing, possibly with certain fields transformed by remapping at the slice array midpoint.

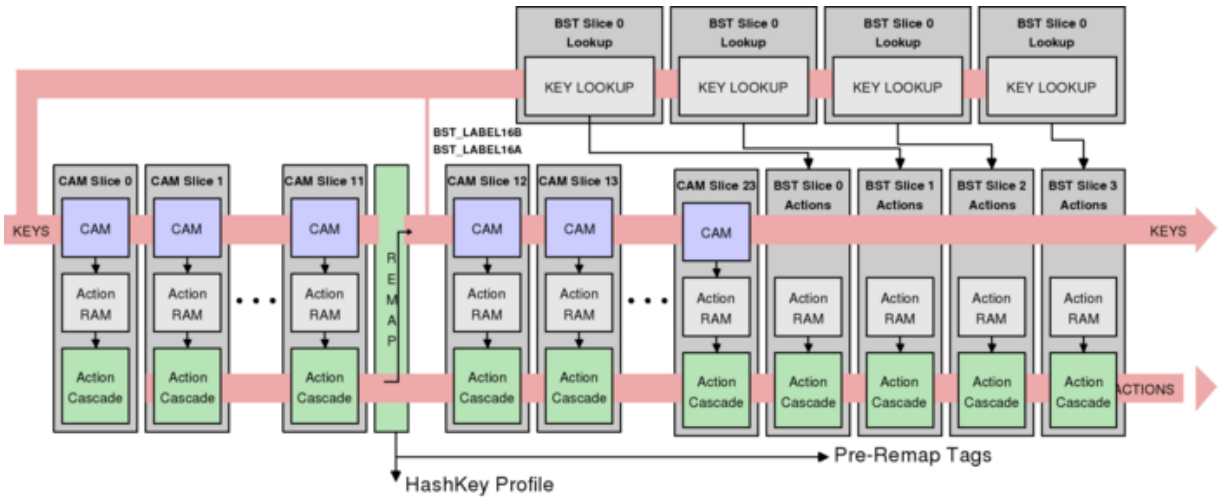


Figure 5-16 FFU Structure

The FFU's rules are divided into two categories:

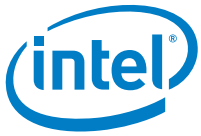
- **CAM Slices** — A collection of 1024 rules that share the same 36-bit input key derived from the frame's header and mapped fields. The key is matched against all 1024 rules in a ternary CAM, with a precedence hit detect function resolving the hits to a single winning rule. The winning rule assigns specific values to a set of action channels that are later interpreted by L3AR. There are 24 CAM slices, providing a total of 24,510 CAM-based rules.
- **BST Slices** — An additional collection of 16,383 rules implemented with a BST structure, intended primarily for longest prefix DIP matching. There are four BST slices, providing a total of 65,532 BST-based rules. Each BST slice supports a key width of 32 bits, but can be combined with up to three neighboring slices to support matching against keys as large as 128 bits. Any number of low-order bits of each key can be masked out of the matching, providing support for longest-prefix route matching. Within one BST slice, only one of the BST's 16-Kb rules match against any given frame. The winning rule specifies an action in the same manner as the CAM slices.

  Note: The BST keys are taken after the remap mid point so that the first half of CAM rules can influence the BST search.

5.7.2 Keys

There are 768 bits of header-derived keys that propagate unmodified from slice-to-slice. The keys are organized as follows:

- 8 x 32-bit keys
- 24 x 16-bit keys
- 16 x 8-bit keys
- 32 x 4-bit keys (upper or lower halves of the 8-bit keys, mapped in each slice).



Each slice (CAM or BST) selects some subset of these keys to match against. The CAM slices select a 36-bit key constructed from the keys previously mentioned. Similarly the BST selects a 36-bit key from the same set of possible keys to match against, but the top 4 bits can only be used to select a partition. The keys are limited to 32-bit within a partition.

Keys are selected in units of eight key bits, with special handling for the top four bits. The 8-bit keys have arbitrary alignment (each byte of selected key can be configured to take any 8-bit input key). The 16-bit keys have 16-bit alignment (each byte of selected key selects either the low or high byte of a 16-bit input key); the 32-bit keys have a single alignment per 32-bits of selected key. The top four key bits are selected among all low and high nibbles of the set of eight bit keys.

5.7.3 Scenario Key

Each CAM or BST slice has an additional 32-bit fixed alignment key which is applied to a 32-entry SCENARIO_CAM to give a 5-bit frame scenario. In the event of no match, the scenario is said to be invalid and all subsequent key comparisons in the associated CAM or BST slice are skipped. This feature can save substantial power when simple properties associated with the frame imply misses in large numbers of related FFU rules. For example, if the DMAC does not match any of the device's router addresses, the SCENARIO_CAM can be used to disable matching against all of the FFU's routing rules.

The processing of the frame scenario is similar between CAM and BST slices. In the event that there is a match in the scenario CAM for CAM slices, the 5-bit scenario value is used to index a 32-entry table (FFU_SLICE_SCENARIO_CFG) to retrieve a scenario configuration, which includes the following information:

- Mux selects of the CAM slice primary key
- Slice chaining properties
- Action chaining
- Case selection (2-bit)

The process for the BST slices is similar, the frame scenario is used to index a 32-entry table (FFU_BST_SCENARIO_CFG) to retrieve a slice configuration register, which includes the following information:

- Mux selects of the CAM slice primary key
- Slice chaining properties
- Action chaining
- Root keys selection
- Nibble selection.

For the remap point, the frame scenario is directly used to retrieve a remap profile:

- Update fields
- Setup hash profile

The structure of scenario to slice configuration is shown in Figure 5-17.

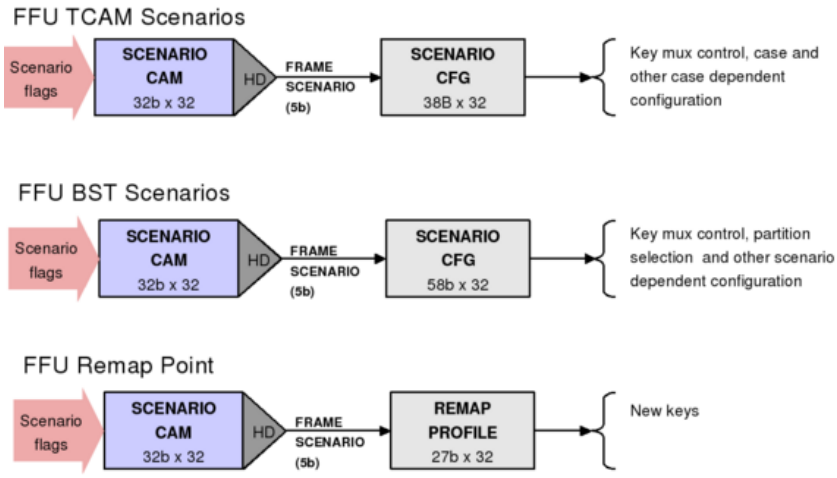


Figure 5-17 FFU Scenario CAM

The 32-bit scenario key bit definitions are listed in Table 5-12.

Table 5-12 FFU Scenario Key Bit Definitions

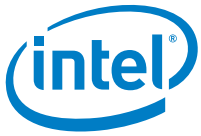
Field	Bits	Description
SCENARIO_FLAGS	15:0	Selected from flags using MAPPER_SCENARIO_FLAGS_CFG.
SRC_PORT_ID3	23:16	Mapped from MAPPER_SRC_PORT_TABLE.
L2_DMAC_ID1	27:24	Mapped from MAPPER_DMAC_CAM/RAM1.
VID1_TAG	29:28	Mapped from MAPPER_VID1_TABLE tag bits.
VID2_TAG	31:30	Mapped from MAPPER_VID2_TABLE tag bits.

Each slice has its own scenario CAM. For a particular case value, the scenario CAMs must be consistent with each other for correct operation. If for a particular case a set of slices implement a slice chain, each slice needs to have the same set of CAM rules to match scenarios to that case. The first slice in the chain must start the comparison while the last in the chain must specify the number of actions to take.

5.7.4 Action Channels

The FFU action channel is a 130-bit collection of data fields, some subset of which might be transformed by any slice. With the exception of two ACTION_FLAGS bits, the action fields are referred to as components of an FFU_DATA structured channel. Generally these channels are available for configurable interpretation by L3 action resolution, either as keys, mux inputs, or both. A handful of these channels play a specific fixed-function role in the subsequent next hop table lookup stage.

All FFU action channels ripple from one slice to the next. If more than one slice assigns the same action channel, the latest slice (numerically highest) one to assign the channel has precedence.



The 130 bits of action data are organized as follows:

- Two Action Flags bits (FFU_Route, FFU_Switch) identifying the forwarding action, if one is specified. These flag bits are mutually exclusive; the pair represents a one-hot encoding of the FFU's forwarding action.
- 24-bit DATA.W24 is primarily intended for forwarding purposes. If Route=1b, this field specifies the next hop table indexing. The FFU_Route and FFU_Switch flags are closely associated with this action channel, and all three fields are set together with the same precedence.
- Three generic 8-bit fields (W8A, W8B, W8C) support bit-masked assignment. Any masked bit can be preserved as is from an earlier slice's assigned value.
- Two generic 16-bit fields (DATA.W16A, DATA.W16B) support assignment masking on a four-bit granularity.
- Four 12-bit TAG fields (TAG1A, TAG1B, TAG2A, TAG2B) which, when assigned, fully overwrites an earlier assignment.

Any slice can assign one W field and one TAG field. Of the TAG fields, only slices 0..11 can assign TAG1A and TAG1B, and only slices 12..23 can assign TAG2A and TAG2B.

The CAM slices and the BST slices determine how they transform the action channels based on a lookup in an action SRAM. The action SRAM is indexed by the matching rule number, assuming a rule matched. The encoding of this SRAM is describe in the Actions section. If no rule matched in the CAM slice or BST slices, the SRAM lookup is skipped and all action channels are left as is.

5.7.5 Action Precedence

Each slice produces only a single command action and/or a tag action. When multiple rules in one slice match, the slice uses the rule with the highest index as the active rule. The actions associated with this rule are combined with the actions resulting from the rest of the slices in the FFU and the action from the BST slices. Multiple actions from different slices are resolved first by data channel and then by action precedence. Actions that modify different data channels are considered orthogonal and are applied in parallel. An example would be one slice specifying a change to the W16A channel while another slice specifies routing of the frame, changing the W24 channel.

A 3-bit precedence value is associated with each unit of action assignment. Any time an FFU rule specifies an action W8/W16/W24 or TAG assignment, it also specifies a precedence. The rule's action data overrides the input action value only if the specified precedence is equal to or greater than the action channel's input precedence value. As a result, if a slice assigns a specific DATA field a precedence value of 2, all subsequent CAM or BST slices with an action with precedence 1 has no effect on that data channel.

Precedence is evaluated and maintained per 1-bit, 4-bit, or 24-bit sub-fields of the DATA.W* channels, and per 12-bit TAG channel, as follows:

- W8{A,B,C} — Precedence is maintained per bit.
- W16{A,B} — Precedence is maintained per nibble.
- W24 — One precedence value is maintained for the entire channel.
- TAG{1,2}{A,B} — Precedence is maintained per 12-bit tag.

The W24 precedence value is also shared with the FFU_Switch and FFU_Route flags. The W24 channel and these two flag bits are all set atomically as one unit. Otherwise, the precedence granularity of these fields corresponds to the masking granularity of each field's assignment actions, as described in the sections that follow.

Note: A slice's placement in the slice array represents additional low-order bits of action precedence. If all slices are assigned with the same explicit precedence value, later slices in the array implicitly have higher precedence than earlier slices. The BST slices actions are applied at the end of the slice array, giving it highest implicit precedence.

The precedence value of zero has special hardware meaning. It is used to represent an unassigned action channel for power gating purposes. If a rule specifies an action assignment with precedence zero, the assignment has no effect. As a result, the FFU practically supports seven explicit precedence values (1 through 7) as well as ~5 bits of low-order precedence implied by slice placement.

5.7.6 CAM Slice Chaining

If more than 36 bits of key are needed, two or more adjacent slices can be chained or combined together to create wider condition keys. Two slices chained together support 1024 rules with 72-bit conditions, while three slices create 108-bit conditions, etc. There is no restriction on the number of slices that can be chained together, except that the chain cannot cross the mid-point. Therefore, up to 12 slices can be chained together to create 432-bit keys without incurring performance penalties or side effects).

Each FFU slice has 1024 CAM entries that produce a single bit of "hit." This produces a 1024-bit hit vector. The slice produces this hit vector only if its lookup is valid. This is true if the slice's LookupValid is true, there was a hit in its SCENARIO_CAM and the case specified by the scenario set either or both ValidLow and ValidHigh to true. If the action is valid for this slice, the highest index hit in the hit vector corresponds to the action SRAM index to activate. The action is only valid if the slice's ActionValid is set and the ActionLength is greater than zero.

When two slices are chained together, after the first slice's CAM is looked up, the hit vector is passed to the chained slice instead of being used to look up an action for that slice. If the second slice's lookup is valid, the hit vector of the previous slice is ANDed with the hit vector generated by looking up all 1024 rules in the second slice. This ANDed result becomes the hit vector for the slice chain. If the second slice's action is valid, this hit vector is used to lookup the action corresponding to the highest indexed rule that hit in both slices. Otherwise, the hit vector representing the combination of both hit vectors is passed to the next slice. If the next slice's lookup is not valid, the hit vector from the previous slice is passed through unchanged to the next slice.

Slice chaining is configured by the StartCompare bit in each slice's SCENARIO_CFG register. Slice chaining cannot extend over the remap stage; slice 12's StartCompare is treated as 1b regardless of its configured value.

5.7.7 CAM Slice Exclusion Sets

To create an ACL or routing table larger than 1024 entries, multiple hardware slices can be grouped together to produce at most a single action between them. On a per-case basis, a slice (or chain of slices) can be disabled if the previous slice (or chain of slices) was found to have an active action. This enables multiple consecutive slices to produce a single action, where the entries in earlier slices have precedence over entries in later slices, regardless of the precedence fields of later entries. Slices grouped in this manner are referred to as slice exclusion sets.

Slice exclusion sets are configured on a per-scenario and per-chain basis using the StartSet field in the FFU_SLICE_SCENARIO_CFG register. This bit must be set to 1b to start a new set and to 0b to continue the current set. This bit should be set along with the StartCompare bit that specifies the start of a hit chain.

Unlike slice hit chaining, exclusion sets can extend through the remap stage.

5.7.8 Action Chains

While each slice can only produce one action, each case can specify usage of multiple consecutive action SRAMs to specify more than one action for this set of conditions. This enables a rule set (usually chained with other slices) to specify multiple actions (up to three) that are to be activated in parallel. This can lead to efficiencies for rule sets that have wide condition keys. Instead of replicating the slice chains for each action, a single slice chain can specify a set of actions. Slices that have action SRAMs that are part of an action chain can have their condition CAM used in a separate slice chain, using both CAM and SRAM structures in the slice. A configuration example of this is represented in Figure 5-18.

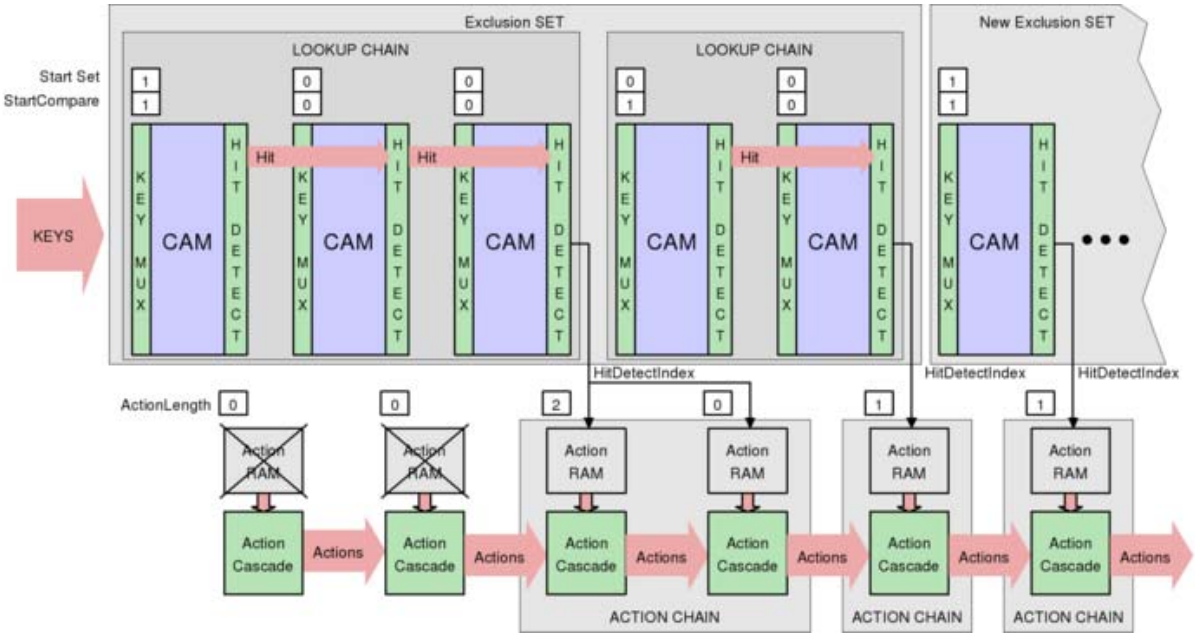


Figure 5-18 FFU Cascading Slices

Figure 5-18 shows two slice chains, one exclusion set and one action chain. This arrangement is for one specific case and could be different for another case.

The two slice chains are using slices 0-1-2 and slices 3-4 respectively and are part of one exclusion set. If the first chain hits, the second chain is skipped over. The first slice chain is using 2 consecutive action RAMs (2 and 3) enabling up to two actions to be set for a single hit. The second slice chain uses one action RAM (4) and can only define one action for any hit.

Only the slice starting the action chain needs to specify the number of action SRAMs to be used. It is considered a mis-configuration to have a slice that is participating in an action chain to also be involved in another action chain for a given case. If done, the new ActionLength overrides the previous length of action, setting the end of the action chain to be equal to the number of slices specified by the last ActionLength field greater than zero.

Each action SRAM in an action chain can specify a precedence of zero, which causes this action SRAM to not produce an action. Later action SRAMs in the chain are still evaluated. Not specifying an action implies that the action channels pass through the slice unmodified. The per-slice ActionValid disables the actions functionality, possibly in a defective slice, and all action fields are passed through unchanged.

Slices with actions disabled in this way are skipped over if earlier slices specify that their actions are to be chained.

5.7.9 Egress Actions

Figure 5-19 shows the linkage between the FFU and the Egress ACLs.

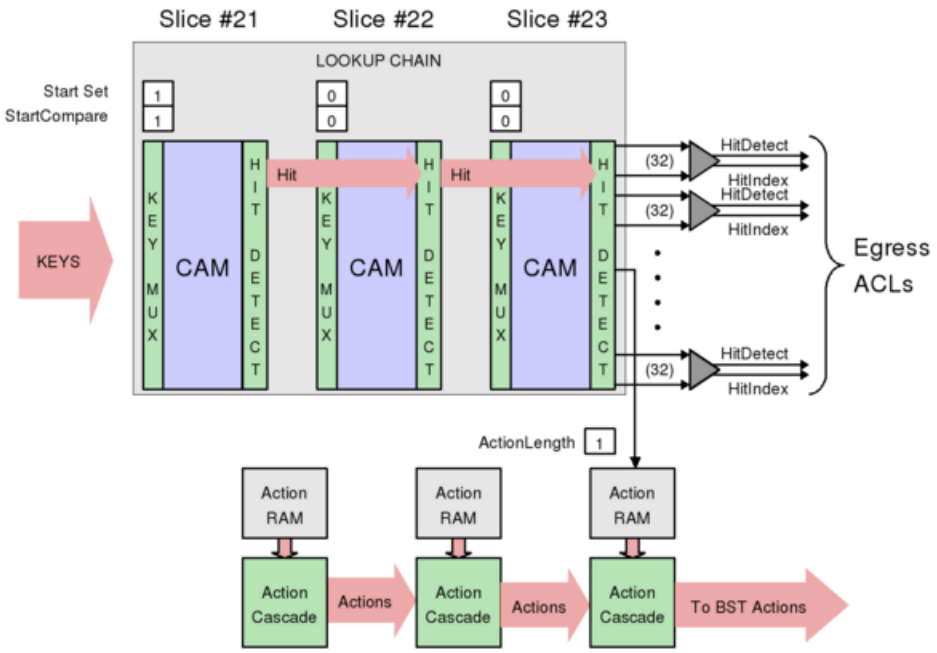


Figure 5-19 FFU Egress ACLs

The CAM rules in the FFU's last slice can be interpreted as egress actions by further conditioning their hit status on the forwarding destination mask determined later in the pipeline. The configured rules are partitioned into hit sets in units of 32 rules. In each set, only one rule can hit, providing up to 32 independent egress action lists comprising 32 rules each. The winning rule numbers, one per set, are preserved until the EACL stage of the pipeline where they are tested against the frame's destination mask to determine their final hit status. The partitioning of the last slice's rules into egress action lists is configured by the FFU_EACL_CFG register.

See Section 5.16 for a complete description of Egress Actions and their implementation of Egress ACLs.

Note: If any egress ACL rule is used, the scenario case for the last slice must be a hit to produce the right hit detect vector. If a frame is causing a miss in the scenario for this slice, the hit detect vector from the previous slice is carried through to the egress ACLs and results in unwanted egress action matches.

5.7.10 Remap Stage

Generally, the FFU keys are propagated unmodified from one slice to the next. The exception is at the remap stage, which offers the capability to set a small number of keys based on the action results from slices 0..11. The selection of which keys are remapped depends on a REMAP_PROFILE lookup table indexed by a SCENARIO_CAM, enabling the remapping to be conditioned on the type of frame passing through the FFU.

Table 5-13 FFU Remap Profile Fields

Field	Width	Description
Select_LABEL16	1	Selects source for 16-bit LABEL16 mux: 0b = FIELD16D (no change) 1b = DATA.W16A The LABEL16 is mapped to key FFU_BST_LABEL16B
Select_LABEL8A	2	Selects source for 8-bit LABEL8A mux: 00b = FIELD16C[7:0] (no change) 01b = DATA.W8A 10b = TAG1A[7:0] 11b = Reserved The combination of {LABEL8B,LABEL8A} is mapped to key FFU_BST_LABEL16A
Select_LABEL8B	2	Selects source for 8-bit LABEL8B mux: 00b = FIELD16C[15:8] (no change) 01b = DATA.W8B 10b = TAG1B[7:0] 11b = Reserved
Select_HASH_PROFILE	1	Selects hash key profile index: 0b = 0 (constant) 1b = DATA.W8A[1:0]

The DATA.W16A, DATA.W8A, and DATA.W8B values previously described refer to the state of these action channels after the TCAM slice immediately prior to the remapping stage (the 12th slice).

At the Remap stage, the state of the two 12-bit TAG channels is captured on the TAG1A and TAG1B outputs for use by L3AR. The TAG channels otherwise propagate unmodified to the second stage of the FFU and, at the final output of the FFU, are presented to L3AR as TAG2A and TAG2B.

Slice hit chaining cannot propagate over the remap point. At slice 12, all SCENARIO_CFG StartCompare values are treated as 1b regardless of their configured values. Exclusion sets and action chaining can be configured to propagate through the remap stage.

5.7.11 Action SRAM

Once a matching rule has been determined by a CAM or BST slice, the rule number is mapped through an action SRAM to determine a 42-bit action specification. The specification, encoded as described in the sections that follow determines how that rule transforms the action channel structure. The most generic entry format takes the following form listed in Table 5-14.

Table 5-14 FFU Action RAM

Field	Bits	Description
ActionData	23:0	Conditionally formatted action data.
Route	24	Specifies a route action, meaning the FFU_Route action flag is set and a lookup is done in the next hop table. Exclusive with the switch action, the FFU_Switch bit in ACTION_FLAGS is cleared if set.
TagData	36:25	12-bit tag value, overwrites tag A or tag B depending on TagCmd.
TagCmd	38:37	00b = Do not modify tags. 01b = RSVD. 10b = Set TAG nA. 11b = Set TAG nB, where n=1 for slice numbers 0..11 and n=2 for CAM slice numbers 12..23 and the BST slices.
Precedence	41:39	Explicit precedence assignment as previously described.

5.7.11.1 Route Action

When Route=1b, the route flag is set and a lookup in the next hop is implied. For this action, the contents of ActionData are interpreted as listed in Table 5-15.

Table 5-15 FFU Route Action Encoding

Field	Bits	Description
NextHopBaseIndex	15:0	Next hop table index.
NextHopRange	22:16	Entry range size.
NextHopEntryType	23	0b = Narrow 1b = Wide

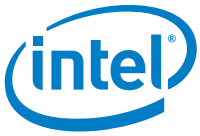
These values are assigned to the DATA.W24 channel for the next hop table's use. As a side effect of this action, the switch flag is cleared and the value of DATA.W24 as specified by any prior Route=0b action is be cleared. Thus, according to these semantics, the FFU can only switch or route a frame.

5.7.11.2 Switch Action

When Route=0b, a similar switch flag in bit 23 of ActionData distinguishes between two non-route actions. Switch=1b is intended for use as a direct destination GloRT assignment action when lookups in neither the next hop nor MAC table are necessary. In this case, the contents of ActionData have the following interpretation listed in Table 5-16.

Table 5-16 FFU Switch Action Encoding

Field	Bits	Description
SwitchData	22:0	Specifies the value to which DATA.W24[22:0] is assigned. DATA.W24[23] is set to 0x0.
Switch (1)	23	Encodes a switch action, setting the FFU_Switch flag. Assuming the action's precedence value is greater than or equal to the precedence of a prior route or switch action, the FFU_Switch flag is set to 1b, the FFU_Route flag is set to 0b, and DATA.W24 is set to the contents of ActionData[22:0].



5.7.11.3 All Other Actions

When Route=0b and Switch=0b, the remaining 22 bits of ActionData encode assignments to any one of the FFU action data channels. In this case, the contents of ActionData[22:0] has the following definition listed in Table 5-17.

Table 5-17 FFU Non-route Actions

Field	Bits	Description
Data	19:0	Payload interpreted based on Cmd. When assigning to FFU action data channels, precedence state is maintained and evaluated per minimum masked unit of data for each particular channel. Thus, DATA.W24 is assigned as a 24-bit unit; DATA.W16* fields are assigned with 4-bit granularity, and DATA.W8* fields are assigned with 1-bit granularity.
Cmd	22:20	Specifies one of six action channel assignments: 000b = Sets DATA.W24 from ActionData[19:0], Route=0, Switch=0. 001b = Sets DATA.W16A from ActionData[15:0] with nibble-masking controlled by ActionData[19:16]. 010b = Sets DATA.W16B from ActionData[15:0] with nibble-masking controlled by ActionData[19:16]. 011b = Sets DATA.W8A from ActionData[7:0] with bit-masking controlled by ActionData[15:8]. 100b = Sets DATA.W8B from ActionData[7:0] with bit-masking controlled by ActionData[15:8]. 101b = Sets DATA.W8C from ActionData[7:0] with bit-masking controlled by ActionData[15:8]. 110b = Reserved 111b = No action

The precedence of each action channel is maintained and evaluated per unit of masking. Thus, for any DATA.W* bit, the general expression describing its precedence and value transformation is:

```
if (ActionMask[i]==1 && ActionPrecedence ≥ PREC[i])
    DATA[i] = ActionValue[i]
    PREC[i] = ActionPrecedence
else
    DATA[i] and PREC[i] remain unchanged
```

where:

- `DATA[i]` represents any DATA.W* bit and `PREC[i]` represents its three-bit precedence value.
- `ActionMask[i]` is the corresponding assignment mask bit as implied by ActionData. For DATA.W24, the ActionMask is always 0xFFFFF; for DATA.W8*, it is ActionData[15:8]; for DATA.W16*, it is derived from ActionData[19:16] as follows:

  ```
  ActionMask = 0xF000 .ActionData[19] + 0x0F00 .ActionData[18] + 0x00F0
  .ActionData[17] + 0x000F .ActionData[16]
  ```

  This enables different rules to set different DATAW16* nibbles in an orthogonal manner.

5.7.12 BST Key Generation and Matching

In most respects, the four BST 16K-rule slices operate exactly like a 1K-rule CAM slice. For each BST slice, a BST_SCENARIO_CAM maps the FFU's SCENARIO_KEY to a 5-bit frame scenario. From each frame scenario, the BST slice's configuration is looked up in FFU_BST_SCENARIO_CFG[Case], specifying:

- Chaining of slices to perform lookup for larger keys (64, 96, 128 bits keys)
- Creating distinct exclusion sets to ensure one entry is selected across multiple slices
- Defining key muxing and masking control for the frame's key generation to be used in the BST search.
- Defining action RAM chain

The frame's BST key muxing is specified in units of eight bits for the lower 32-bit and in a unit of four bits for the final four bits, the muxing of each field encoded in the same manner as for the CAM slices. The BST slices also offer a nibble masking stage that follows the muxing.

The BST's frame key is generated from the header fields available after the remap point, so it is possible to use the first half of the FFU CAM slices to influence key search at BST.

Figure 5-20 shows one BST slice.

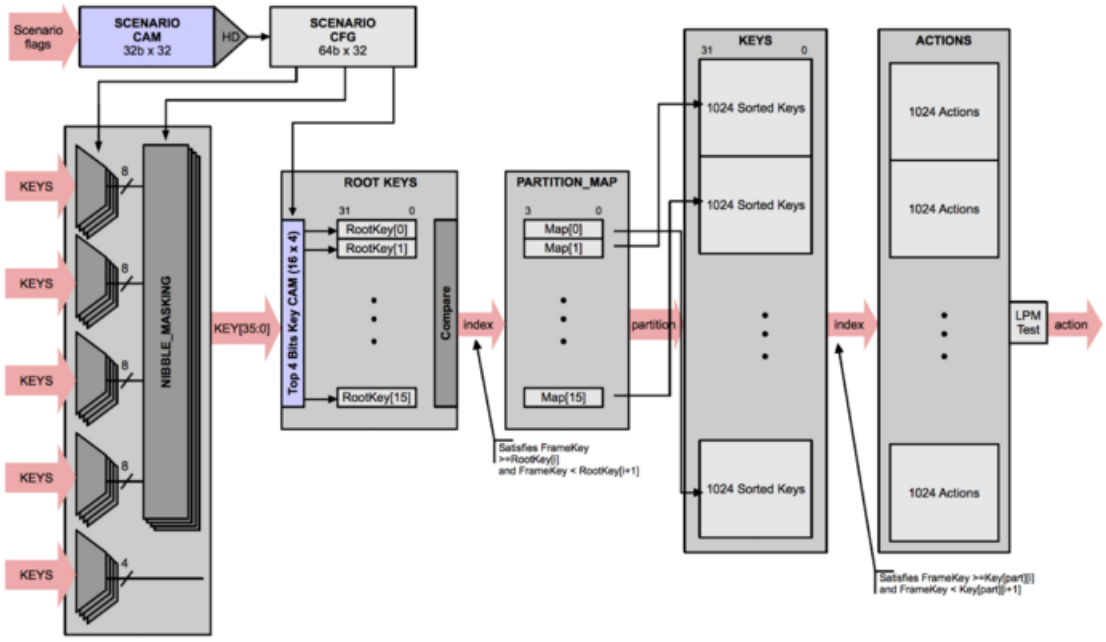
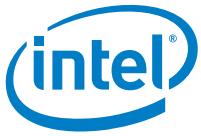


Figure 5-20 FFU BST Slice

As shown in Figure 5-20, the scenario CAM and scenario CFG is used to construct a frame key from the various keys, which can be masked nibble-by-nibble. The frame key is then compared to the set of root keys valid for this case.



A root key is valid for comparison if the top four bits of the key matches the corresponding CAM entry and the root key is set valid in the case configuration. The resulting root index is then used to retrieve a partition number from a partition map. The partition itself is a binary search structure that is walked through to locate the index i that brackets the frame key search. Software must ensure that the entries are written in order for both the root keys and within a 1-Kb partition.

```
KEY[slices,part,i].Value . FrameKey[31:0] < KEY[slices,part,i+1].Value
(last+1 is inifinity)
```

Note: KEY[*,*,0] is always 0 (cannot be changed). If the key is smaller than KEY[*,*,1], the resulting index i is zero and hardware considers this a miss (no action).

Slices can be combined to create larger key (64 bits, 96 bits, 128 bits) by using the StartCompare bit in the scenario configuration profile. Setting the StartCompare creates a new chain of slices starting at this slice and clearing this bit chains this slice to the previous one to create wider keys. The variable slices in the previous equation is the set of slices in a chain (1 slice for 32 bit, 2 slices for 64 bits, etc...). For equal keys, the switch always selects the highest index.

Exclusion sets can be defined to create large key tables that span over multiple slices (16-Kb, 32-Kb, 48-Kb, 64-Kb). This feature is controlled by the StartSet bit in the scenario configuration register. Setting this bit initiates a new exclusion set, clearing this bit extends the set over multiple slices (or set of slices for wider entries). During lookup, if a frame key falls in the current set of root keys, all further lookup is skipped in the follow-on slices (or chain of slices) that are part of the same exclusion set. The slices must be organized with the highest value keys in the first slice of the exclusion set.

Given the matching rule index [partiton,i], the BST action is then looked up in a 16K-entry FFU_BST_ACTION table. The encoding of this action RAM is identical to that previously specified for the CAM slices except for the addition of LPM field to validate the action. The BST's actions are applied after the final stage of the CAM slice array, so any BST actions overrides any conflicting CAM actions with the equal or lower precedence value. The FFU_BST_ACTION[*,*,0] are always zero and cannot be changed.

Note: The action is applied only if the frame key prefix matches the BST key prefix found (FrameKey & ~((1<<LPM)-1)) == (value & ~((1<<LPM)-1)), where the field LPM defines the number of bits that are ignored.

If no entry in the table matches the frame's key, no BST action lookup is performed and all action channels are left as-is in their state at the end of the CAM slice array.

Figure 5-21 shows the usage StartSet and StartCompare to chain multiple slices to create wider and larger tables.

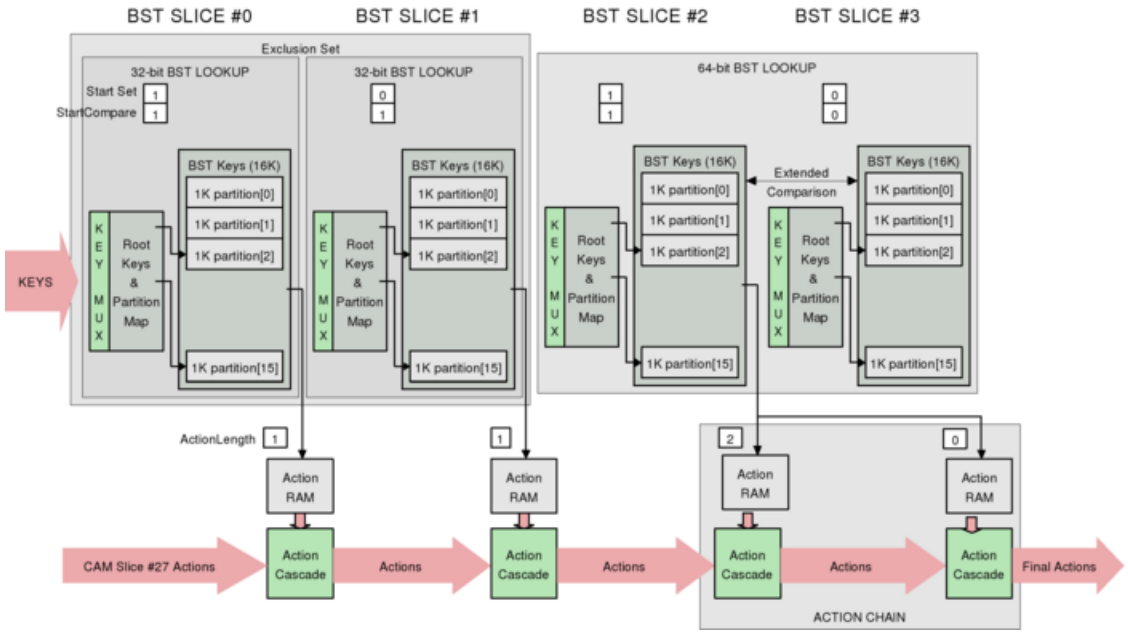


Figure 5-21 FFU Cascading BST Slices

5.7.13 Atomic Modifications

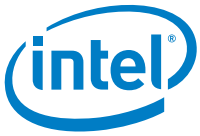
The following registers have special handling:

- For CAM slices:
  - FFU_SLICE_MASTER_VALID
- For BST slices:
  - FFU_BST_MASTER_VALID
  - FFU_BST_SCENARIO_VALID
  - FFU_BST_PARTITION_MAP
  - FFU_BST_ROOT_KEYS

Each write into these registers is stored into write-only shadow registers and remains in shadow until software commands those registers to be written into the active configuration. A read to these registers returns the current active configuration, not the value of the shadow register.

Software must use the FFU_ATOMIC_APPLY register to control the application of these registers. This register contains two bits. The first bit controls the special CAM slice register application FFU_SLICE_MASTER_VALID. The second bit controls the special BST slice registers FFU_BST_MASTER_VALID, FFU_BST_SCENARIO_VALID, FFU_BST_PARTITIONM_MAP, and FFU_BST_ROOT_KEYS[0..15] applications.

A value of 0b written to either bit is ignored. A write of 1b causes all of the corresponding CAM or BST registers to be set to their pending shadow values atomically between two frames. This feature cleanly enables swapping an existing BST or CAM configuration to a new one.



5.7.14 FFU Output

Many of the FFU keys do not persist beyond the last FFU slice. However, certain keys of interest are propagated on to the L3AR stage where they are available as CAM keys and ACTION_DATA mux inputs. The final action values are also sent to L3AR. If the route flag is set (such as the final forwarding action to take effect was a route action), FFU_DATA.ROUTE is sent to the next hop table instructing it to perform a lookup. If the route flag is not set, no lookup in the next hop table takes place.

All outputs of the FFU are listed in Table 5-18.

Table 5-18 FFU Outputs

Field	Width	Description
L2_DMAC	48	Preserved key for L2 lookup.
L2_SMAC	48	Preserved key for L2 lookup.
L2_VID1	12	Preserved key for L2 lookup.
L2_VID2	12	Preserved key for L2 lookup.
L3_LENGTH	16	Preserved key (for MTU checking).
L3_SIP	128	Preserved key (for ICMP redirect and configurable overloading).
L3_DIP	128	Preserved key (for deriving DMAC in IPv6 multicast routing case).
QOS	22	QoS channel, preserved un-modified.
SCENARIO	5	Preserved key from mapper.
L3_LENGTH_BIN	4	Preserved key from mapper.
HASH_PROFILE	4	Hash key profile number. Set by remap stage.
FFU_DATA.W8A	8	8-bit DATA8A output from the last slice.
FFU_DATA.W8B	8	8-bit DATA8B output from the last slice.
FFU_DATA.W8C	8	8-bit DATA8C output from the last slice.
FFU_DATA.W16A	16	16-bit DATA16A output from the last slice. Suggested sub-field definitions include: VID = 11:0 W4 = 15:12
FFU_DATA.W16B	16	16-bit DATA16B output from the last slice. Suggested sub-field interpretations are: VID = 11:0 W4 = 15:12
FFU_DATA.W24	24	Overloaded to carry the data associated with both the route and switch actions. Sent to the next hop table when routing. This field is also available as an L3AR key. Sub-field definitions include: GLORT = 15:0 W4 = 19:16
FFU_DATA.TAG1A	12	Value of action tag A following slice 13 (prior to remap point).
FFU_DATA.TAG1B	12	Value of action tag B following slice 13 (prior to remap point).
FFU_DATA.TAG2A	12	Value of action tag A following the BST_APPLY stage.
FFU_DATA.TAG2B	12	Value of action tag B following the BST_APPLY stage.
FFU_EGRESS_ACLS	32 x 6-bits	Egress ACLs hit detects (32 x (1-bit hit detect + 5-bit hit index).

All frame header fields and mapped FFU keys that were not previously listed are discarded at this point in the pipeline.

Additionally, the FFU sets the following flags in ACTION_FLAGS. The FFU guarantees that switch and route are mutually exclusive. See Table 5-19.

Table 5-19 FFU Action Flags

Field	Bits	Description
FFU_Switch	45	Set when the final forwarding action to take effect specifies Switch=1b.
FFU_Route	46	Set when the final forwarding action to take effect specifies Route=1b.
FFU_12aEqVID1	47	Indicates FFU_DATA.W16A[11.0]==L2_VID1.
FFU_12aEqVID2	48	Indicates FFU_DATA.W16B[11.0]==L2_VID2.

All outputs colored yellow indicate outputs that are produced or modified by the FFU.

5.8 Frame Hashing

The frame hash unit calculates and assigns a set of hash values to each frame based on a configured set of header keys. A total of six 16-bit hash values are calculated, three prior to L3AR and three after based on two dynamically selected hash key profiles.

5.8.1 L2 and L3 Frame Hashing Overview

Table 5-22 shows the architectural partitioning and data flow of the FM5000/FM6000's frame hashing in the larger frame processing pipeline.

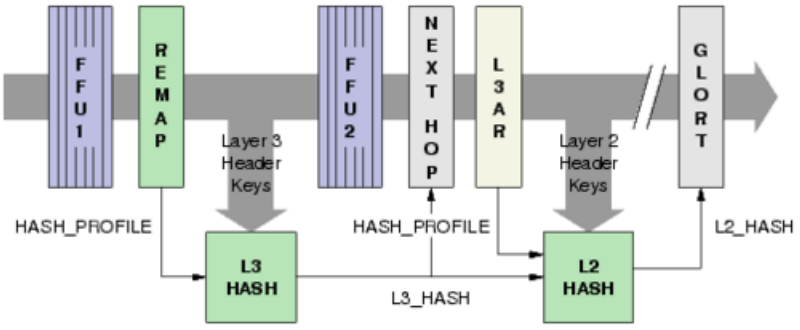
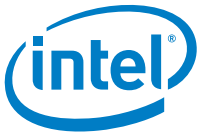


Figure 5-22 Hashing Blocks

The FM5000/FM6000 calculates two sets of frame hash values at two different points in the pipeline. The first stage calculation occurs prior to routing and other L3AR transformations of the frame's header fields. The second set of hash values is calculated after L3AR. The first stage calculation has full access to all of the frame's header fields as presented to the FFU, while the second stage calculation has a reduced set of header fields available to it.



The first stage hash calculation is primarily intended for ECMP and other L3 + load-balancing applications involving the next hop table. For that reason it is referred to as the L3 Hash, or L3_HASH. The second stage hash calculation is primarily intended for link aggregation and other link-layer load-balancing applications involving the destination mask table. It is referred to as the L2 hash, or L2_HASH.

Each hash value calculation operates on a fixed-size key constructed from the available set of header fields. The frame's hash key profile determines the selection and bit-masking of those header fields. 16 programmable hash key profiles are supported for each hash calculation stage. The FFU remap stage determines the first-stage hash key profile based on FFU actions and a SCENARIO_CAM lookup (see Section 5.7.10). By default, the L2 hash key profile is selected by the same index as the L3 hash key's profile, although the L3AR might override the index with its SetHashKeyProfile action.

The hash key generation logic supports symmetrization of certain source- and destination-specific fields. When enabled, symmetrization of these fields ensures that frames sent from a network host A to some other host B have the same hash value as frames sent from host B-to-A, assuming all other header fields are the same. This in turn guarantees that frames belonging to a given conversation follow the same path through the network, which might be desirable in some applications.

5.8.2 Hash Rotations

The FM5000/FM6000 hardware is capable of calculating up to three 16-bit hash values per frame: one for load balancing over next hop entries (L3_HASH) and two that can be used for load balancing over L2F table entries or for applying LAG filtering (L2_HASH_A and L2_HASH_B). To assure uncorrelated port selection over different switches in a network and across different hierarchical levels of hashing in multi-chip systems, each of these hash values can be selected from a set of thousands of independent hash functions that the FM5000/FM6000 supports. The particular independent hash function is referred to as the hash rotation, specified by a 20-bit HASH_ROT value.

The HASH_ROT value used in the calculation of L3_HASH is selected by the FFU remapping stage. The frame's FFU_REMAP_PROFILE entry specifies HASH_ROT as either a constant configured in the profile or indirectly as an assignment from {FFU_DATA.W8A[7:4],FFU_DATA.W16B}. The latter mode enables the hash function to be chosen by an FFU rule.

L3AR can subsequently modify HASH_ROT for use in the L2 hash. For each hash value, the L2 hash profile specifies whether to use the L3AR-modified HASH_ROT or a constant hash rotation number configured in HASH_LAYER2_PROFILE.

Different specific HASH_ROT values give hash functions of differing quality, in terms of hashing uniformity and cross-rotation independence. Selecting high-quality HASH_ROT values (16-bit mantissa, 4-bit exponent) is listed in Table 5-20.

Table 5-20 High-Quality HASH_ROT Values

51407, 15	29347, 14	3709, 11	23167, 13	4483, 12	3877, 11	18637, 14	20407, 14
27017, 14	5591, 12	2309, 11	26951, 14	62873, 15	31873, 14	50857, 15	3559, 11
49411, 15	34877, 15	4883, 15	4243, 15	15193, 13	57163, 15	49451, 15	5659, 12
22051, 14	64151, 15	64237, 15	64319, 15	13841, 13	48869, 15	52973, 15	22871, 14
50221, 15	62201, 15	3499, 11	49871, 15	52631, 15	27827, 15	28879, 14	14533, 13

5.8.3 Random Hashing

The FM5000/FM6000's hash function includes a 16-bit pseudo-random number generator. For load balancing applications that do not require frame order preservation within flows, the frame's hash value can be overridden by this random number. A randomize configuration bit per hash value specifies this behavior in each hash profile.

The random number generator has a period of 2,147,483,647 frames. The number is advanced on every frame that the FM5000/FM6000 processes with a randomized hash profile, without regard for source port, flow association, or any other frame-specific property.

5.8.4 L3 Key Generation

The L3 hash keys have the following header fields listed in Table 5-21.

Table 5-21 L3 Hash Keys

Field	Bytes	Masking Support	Symmetrization Set
L3_SIP	15:0	Per byte.	S1
L3_DIP	31:16	Per byte.	S1
QOS.L3_PRI	32	Per bit.	-
L3_FLOW	35:33	Per bit.	-
ISL_USER	36	Per bit.	-
L3_PROT	37	Per bit.	-
L4_SRC	39:38	Per bit.	S2
L4_DST	41:40	Per bit.	S2
FIELD16[B,A]	49:46	Per byte.	-
[LABEL16,LABEL8B,LABEL8A[1]	45:42	Per byte.	-

1. These are the FIELD16[D,C] keys following the FFU midpoint remapping stage.

The meaning of the symmetrization sets defined here are explained in the sections that follow.

5.8.5 L2 Key Generation

The L2 hash key is constructed from the following header fields listed in Table 5-22.

Table 5-22 L2 Hash Key Header Fields

Field	Bytes	Masking Support	Symmetrization Set
MA1_MAC[1]	5:0	Per bit.	S3
MA2_MAC[2]	11:6	Per bit.	S3
L2_TYPE	13:12	Per bit.	-
(EVID1,QOS.L2_VPRI1	15:14	Per bit.	-

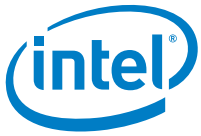


Table 5-22 L2 Hash Key Header Fields

Field	Bytes	Masking Support	Symmetrization Set
(EVID1,QOS.L2_VPRI2	17:15	Per bit.	-
ACTION_DATA.W16[A,B,C]	23:18	Per 2 bytes.	-

1. L3AR selects the configuration; nominally L2_DMAC.
2. L3AR selects the configuration; nominally L2_SMAC.

When the UseL3A or UseL3B hash profile bits are set to 1b, the 16-bit L3_HASH is XORed into the early stages of the L2_HASH_A or L2_HASH_B hash calculations, respectively. Thus, when so configured, the L2 hash key also includes the L3 hash key.

5.8.6 Symmetrization

Header fields belonging to the same symmetrization sets previously defined have an equal number of source and destination-specific bytes. Symmetrization is enabled per symmetrization set in each hash key profile.

When enabled, the source and destination key fields are derived from the header fields by a per-byte comparison:

```
foreach i in 0..num_bytes(field)-1:
  symmetrized_src_field[i] = MAX(src_field[i], dst_field[i])
  symmetrized_dst_field[i] = MIN(src_field[i], dst_field[i])
```

Where MAX and MIN refer to a simple numeric comparison.

5.8.7 Outputs

Three statistically independent 16-bit hash values, referred to as rotations, are calculated for each of the frame's L2 and L3 hash keys. The HASH_LAYER3_PROFILE and HASH_LAYER2_PROFILE registers specify which of these rotations are to be used by downstream logic. One L3 hash rotation and two L2 rotations are selected.

Table 5-23 Hashing Stage Output

Hash Value Output	Width	Description
L3_HASH_NEXTHOP	16	L3_HASH value used by the next hop table for ECMP-style load balancing.
L3_HASH	16	L3_HASH value sent to downstream frame processing stages, in particular to L3AR and the L2 hash function.
L3_HASH_A	16	L2 hash rotations, selected by the RotationA and RotationB fields, respectively, of HASH_LAYER2_PROFILE[profile]. Used for load balancing over L2F table entries and in the fixed-function LAG filtering stage. For more details, see Section 5.14, "GloRT Lookup".
L3_HASH_B	16	

5.9 Next Hop Table

The next hop table, formerly known as the ARP table, is a wide and deep mapping table indexed by the FFU's route action. Its primary role is to determine the next-hop L2 header in the context of IP routing, although in the FM5000/FM6000 it performs a more generalized next-hop header resolution function for a variety of different hierarchical network forwarding applications.

5.9.1 Overview

The general operation and capabilities of the next hop table are described as follows.

- Two entry types:
  - **Narrow** (64 bits) — Minimal entry size, optimized for standard IP routing with no support for additional tunneling features.
  - **Wide** (128 bits) — Double size entry. Supports additional fields for tunneling and other optional features at the expense of fewer total entries in the table.
- Capacity for 64-Kb narrow entries or 32-Kb wide entries. Each index in the table is individually selected to be either narrow or wide.

  Note: Entries 65534,65535 (narrow) and entry 32767 (wide) are not available and must not be used.
- Lookup specified by a 16-bit base index, a 1-bit entry size, a 16-bit hash value, and a 10-bit Equal Cost Multi-Path (ECMP) range size. Given these inputs, a specific table index is calculated, and either a narrow or wide entry is looked up and passed on to the L3AR stage.

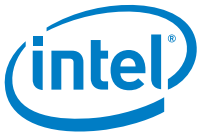
5.9.2 Input Interface

The FFU resolves a single routing action over its 28-Kb TCAM and 64-Kb BST rules. If a routing action is specified (FFU_Routed=1b), the routing command's FFU_DATA.W24 contents are passed to the next hop table and its 24 bits are interpreted as listed in.

Table 5-24 Next Hop Routing Command Contents

Field	Bits	Description
NextHopBaseIndex	15:0	Base index into NEXTHOP_TABLE.
NextHopRange	22:16	Encodes the range over which an entry is chosen based on the frame's L3_HASH value or from the low-order bits of its L3_DIP field.
NextHopEntryType	23	0b = Narrow 1b = Wide

In addition, a 16-bit L3_HASH value is provided for the lookup based on the frame's L3 hash calculation.



5.9.3 Index Calculation and Lookup

Given the previous inputs, a specific entry index is calculated and the entry is looked up in the table. In addition to the base index and range fields specified in the FFU routing command, the frame's L3 hash value and L3_DIP field can be used to identify the specific next hop table index to look up.

Case 1 (Hashed):

If NextHopRange[6:3] is any value other than 15, the index that is looked up is calculated using the frame's L3 hash value and division-based binning.

First, the entry range size is calculated over which the entry is uniformly selected.

```
range_size = (2 * NextHopRange[2:0] + 1) << NextHopRange[6:3]
```

By configuring NextHopRange[2:0]=7 and NextHopRange[6:3]=12, it is possible to select a range as large as 60-Kb. Values larger than 60-Kb can be encoded, but these lead to undesirable (nonuniform) aliasing over the 64-Kb entries in the table.

Next, based on the L3_HASH hash value, an index is selected from the calculated range of entries:

```
index = NextHopBaseIndex + (((L3_HASH * range_size) / 65536) << NextHopEntryType)
```

Case 2 (DIP-derived):

If NextHopRange[6:3]=15, the frame's hash value is not used. Instead, the index is directly mapped from the bottom NextHopRange[2:0]+1 bits of L3_DIP, as specified as follows:

```
index = NextHopBaseIndex + L3_DIP[NextHopRange[2:0]:0] << NextHopEntryType
```

In all cases, any index value exceeding 65535 aliases to NEXTHOP_TABLE[index%65536].

If NextHopEntryType=0, the 64-bit entry at NEXTHOP_TABLE[index] is looked up and passed on to L3AR. Otherwise, the two entries at NEXTHOP_TABLE[index & 16'hFFFE] and NEXTHOP_TABLE[(index & 16'hFFFE) + 1] are looked up, and the aggregate 128-bit wide entry is passed on.

Regardless of the entry type, the generic data fields specified in the lookup entry are grouped together into a NEXTHOP_DATA structured channel. The tag fields are passed on as a single NEXTHOP_TAG channel.

5.9.4 Narrow Entry Formats

The narrow entry type (NextHopEntryType=0) offers two entry format options for use in L3 action resolution. These two entry formats define different bit field partitions that are motivated by the needs of unicast and multicast routing applications. These entry aliases are intended to help simplify application programming. The same bits are available to L3AR regardless of which entry type is in use. For example, a mux input referencing the field NEXTHOP_DATA.W12A could be equivalently written (NEXTHOP_DATA.(W4,W8C,TAG[5:2]).

Table 5-25 Next Hop Unicast Table Entry Format

Field Name	Bits	Description
W16A	15:0	Available as an input source of egress DMAC selection.
W16B	31:16	
W16C	47:32	
W12A	59:48	Available as an input source for egress VID selection.
TAG[3:0]	63:60	Generic 4-bit tag field. A use example would be to use 1 bit to indicate if the entry is multicast or unicast.

Table 5-26 Next Hop Multicast Table Entry Format

Field Name	Bits	Description
W16A	15:0	Available as an input source for DGLORT selection.
W8A	23:16	Available for eventual assignment to MOD_DATA.W8A for egress frame modification. Expected use is Virtual Router ID.
W8B	31:24	Available for eventual assignment to MOD_DATA.W8B for egress frame modification.
W8C	39:32	Available for MOD_DATA.W8C and for QoS muxing.
W4A	43:40	Generic 4-bit field. Available as an input source for MTU_INDEX selection. Expected to be set to correspond to the minimum MTU over all VLANs associated with this multicast distribution. Also available for QoS muxing in L3AR.
TAG[7:4]	47:44	Generic tag bits available as L3AR key.
W12A	59:48	Available as an input source for egress VID selection.
TAG[3:0]	63:60	Generic 4-bit tag field. A use example would be to use 1 bit to indicate if the entry is multicast or unicast.

Note: Since many of the fields in these two entries have different names but reference the same bits, there is aliasing between the components of the NEXTHOP_DATA structured output channel. For example, NEXTHOP_DATA.W8A equals NEXTHOP_DATA.W16B[7:0], and TAG[7:4] equals W16B[15:12]. This aliasing does not occur on a wide entry lookup.

5.9.5 Wide Entry Format

A wide entry lookup occurs when the FFU specifies a NextHopEntryType value of one. The bit field partitioning of this entry format is formed by concatenating the two narrow-format entries. This provides a suitably diverse collection of field bit widths without requiring any unique hardware handling for this entry type.

Table 5-27 Next Hop Wide Table Entries

Field Name	Bits	Description
W16A	15:0	Available for MAC mapping and MOD_DATA.W16D.
W16B	31:16	Available for MAC mapping and MOD_DATA.W16E.
W16C	47:32	Available for MAC mapping and MOD_DATA.W16D.

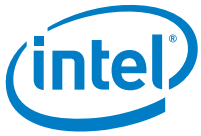


Table 5-27 Next Hop Wide Table Entries (Continued)

Field Name	Bits	Description
W12A	59:48	Available for VID mapping (in particular EVID1).
TAG[3:0]	63:60	Generic tag bits available as L3AR key.
W16D	79:64	Available for GloRT mapping and MOD_DATA.W16C.
W8A	87:80	Available for MOD_DATA.W8A. Expected use is to encode one of up to 256 destination virtual router IDs.
W8B	95:88	Available for MOD_DATA.W8B.
W8C	103:96	Available for MOD_DATA.W8C and for QoS muxing.
W4A	107:104	Available for MOD_DATA.W4, W8D, and for QoS muxing.
W4B	111:108	Available for MOD_DATA.W8D muxing.
W12B	123:112	Generic 12-bit field available for EVID2.
TAG[7:4]	127:124	Generic tag bits available as L3AR key.

5.9.6 Outputs

Based on the type of entry looked up in NEXTHOP_TABLE, either 62 or 124 bits of meaningful output data are passed on to L3AR, in the form of NEXTHOP_TAG (available as a key input) and the NEXTHOP_DATA structured channel. In addition, three fixed-definition flag bits are calculated and set in ACTION_FLAGS.

Table 5-28 Next Hop Data Path Outputs

Field Name	Width	Description
NEXTHOP_TAG	8	Taken from NEXTHOP_DATA.TAG fields.
NEXTHOP_DATA.W*	120	Contents of a narrow entry lookup are promoted to the width of the wide entry format with aliasing between fields as previously described.
NEXTHOP_IDX	16	Final table index used in the lookup. Sent to L3AR for muxing to ACTION_DATA.W16G, then to a RX_STATS.IDX16{A,B} channel in L2AR. It is expected that 64K saturating 1-bit counters are set by this index in the Statistics stage to flag next hop entries that have been recently accessed.

Table 5-29 Next Hop Action Flags

Field Name	Bits	Description
NEXTHOP_EntryType	49	Indicates the entry lookup specified by the FFU route action. 0b = Narrow 1b = Wide
NEXTHOP_12aEqVID1	50	Indicates NEXTHOP_DATA.W12A = L2_VID1.
NEXTHOP_12aEqVID2	51	Indicates NEXTHOP_DATA.W12B = L2_VID2.

5.10 L3 Action Resolution

The Layer 3 Action Resolution (L3AR) stage in the FM5000/FM6000 frame processing pipeline is responsible for a number of high-level packet handling decisions, including:

- Deciding whether to route, and if so, resolving the mechanics of this operation based on the output of the FFU and NEXT_HOP tables.
  - Selecting the next-hop VLAN/SMAC/DMAC.
  - Distinguishing between L3 unicast and multicast.
  - Checking TTL.
  - Identifying ICMP redirect conditions.
- Deciding whether to tunnel the packet (MPLS, MAC-in-MAC, etc.).
- Culling the set of header fields and other data associated with the packet for downstream use.
- Specifying the L2 lookup and filtering policies to apply to the packet.

5.10.1 Overview

The L3AR stage uses a TCAM/slice/mux architecture, which is implemented in microcode and has the following general properties:

- Total number of rules: 160
- Total number of TCAM slices sets: 5
- Number of rules per precedence set: 32
- Number of serial application stages: 5
- Total number of keys: 18
- Total key width: 251 bits
- Number of sequential actions: 6
- Number of output mux actions: 21

Figure 5-23 shows the basic components and data flow of the TCAM/slice/mux architecture.

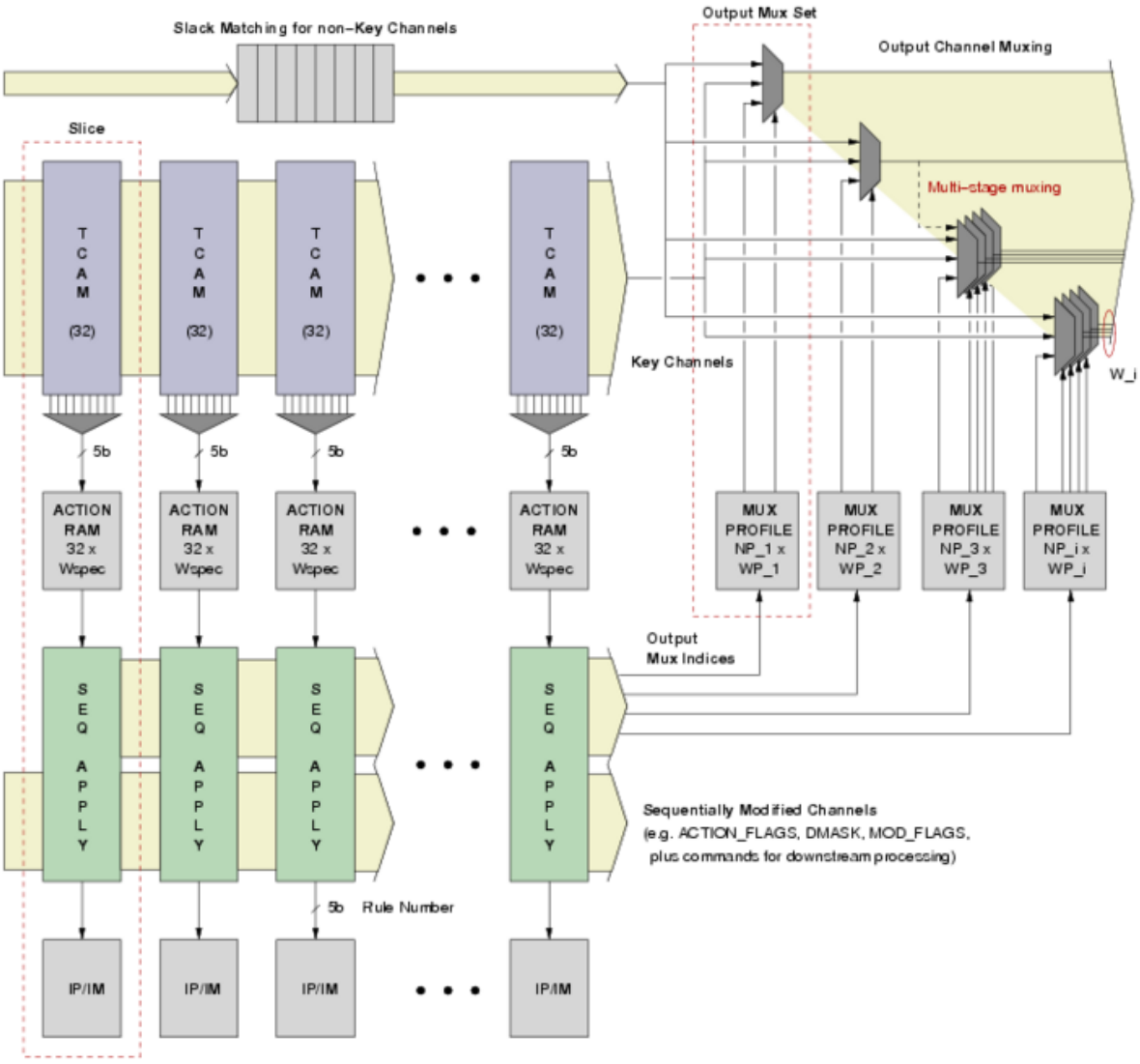


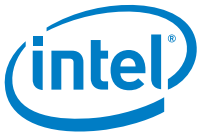
Figure 5-23 TCAM/Slice/Mux Architecture

5.10.2 Keys

Each L3AR slice has the following channel inputs available as TCAM keys as listed in Table 5-30.

Table 5-30 L3AR Input Keys

Key	Width	Description
ACTION_FLAGS	52	Contains the parser's HEADER_FLAGS as well as the results of fixed-function flag evaluations from the intervening stages. See Section 5.10.5 for the specific flag bits that are available at the L3AR stage of the pipeline. ***Note:*** The L3AR's SetFlags action enables all configurable flag bits to be redefined at this stage of the pipeline.
SRC_PORT	7	Source physical port number.
SRC_PORT_ID4	8	Mapper-associated SRC_PORT ID.
ISL_SGLORT	16	Source GloRT from ISL tag.
ISL_PRI	4	ISL priority as assigned by the QoS mapping stage.
L2_TYPE_ID2	4	Mapper-associated EtherType ID.
L2_DMAC_ID3	5	Mapper-associated DMAC ID.
L2_SMAC_ID3	5	Mapper-associated SMAC ID.
L3_DIP_ID3	5	Mapper-associated DIP ID.
L3_SIP_ID3	5	Mapper-associated SIP ID.
L3_PROT_ID2	4	Mapper-associated L4 protocol ID (from L3 header).
MAP_VID1	12	Mapped L2_VID1 value from MAPPER_VID1_TABLE.
MAP_VID2	12	Mapped L2_VID2 value from MAPPER_VID2_TABLE.
L3_HASH	16	L3 hash value.
FFU_DATA.W8{A,B,C}	24	FFU 8-bit DATA outputs (3 x 8-bits).
FFU_DATA.W16{A,B}_TOP	8	Top four bits of each of the FFU's 16-bit DATA outputs (2 x 4-bits).
FFU_DATA.TAG1{A,B}	24	FFU tag fields (2 x 12-bits), prior to the remapping point.
FFU_DATA.TAG2{A,B}	24	Tag fields (2 x 12-bits) from FFU output.
FFU_DATA.W24[23:16]	8	Top eight bits of the FFU's 24-bit route/switch action data payload.
NEXTHOP_TAG	8	NEXTHOP_TABLE entry tag.



5.10.3 Actions

L3AR supports the following actions. All implied action operands, such as valid bits, are not listed in Table 5-31. Each action is described in the Action Detail section that follows.

Table 5-31 L3AR Actions

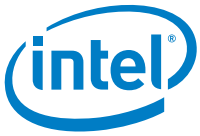
Action	Type	Configuration	Operands	Description
SetFlags	SMS	Value (52 bits) Mask (52 bits)	ACTION_FLAGS	Masked assignment of ACTION_FLAGS[51:0]. Changes accumulate serially from one application stage to the next. ***Note:*** Flags values seen in all L3AR slice TCAM keys is a copy of the ACTION_FLAGS channel at the input of the L3AR stage. Thus any changes applied by this action in a given slice is not visible to the TCAM keys of downstream slices.
SetAlu13CmdProfile SetALU46CmdProfile	SCS	ALU13_CMD_PROFILE (5 bits) ALU46_CMD_PROFILE (5 bits)	-	Sets command profiles for the ALU stage. One profile controls ALUs 1-3, the other controls ALUs 4-6. The ALU operand inputs are selected with a MuxOutput action.
SetL2LookupCmdProfile	SCS	L2L_CMD_PROFILE (4 bits)	-	Specifies the L2 lookup command profile.
SetDestMaskCmdProfile	SCS	DMASK_CMD_PROFILE (4 bits)	-	Specifies the DMASK Generation command profile, used in the L2F, EACL, and LAG stages of DMASK transformation.
SetTrapHeaderCmd	SC	ENABLE (1 bit) IDX (1 bit)	Many	Stores the packet's header fields in L3AR_HEADER_TRAP_DATA[HdrIdx.
SetHashKeyProfile	SD	Profile (4 bits)	HASH_PROFILE	If specified, overrides the L2 hash key profile index from its default value (the L3 hash key profile index) to the specified ProfileIdx value.
MuxOutput_DGLORT MuxOutput_SGLORT	MCM	Profile (5 bits)	Many	Specifies the mux-and-masking profile for two 16-bit output channels (DGLORT and SGLORT). Thirty-two profiles are supported for each.
MuxOutput_W8ABCD MuxOutput_W8E MuxOutput_W8F	MCM	Profile (5 bits)	Many	Specifies the mux-and-masking profiles for six 8-bit ACTION_DATA output channels partitioned into three mux sets. Thirty-two profiles are supported for each set. All mux sets have MCM-type assignment, enabling constant-assignment of arbitrary bits of each 8-bit field.
MuxOutput_W16ABC MuxOutput_W16DEF MuxOutput_W16GH	MCO	Profile (5 bits)	Many	Specifies the mux profiles for seven 16-bit ACTION_DATA output channels partitioned into three mux sets. Thirty-two profiles are supported for each of the ABC/DEF/GH action data channel sets. As an MCO-type action, each 16-bit field must be muxed or assigned in its entirety; bit-masking is not supported for these ACTION_DATA fields.
MuxOutput_MA1_MAC MuxOutput_MA2_MAC	MCO	Profile (5 bits)	Many	Mux control actions for the L2 lookup keys' MAC fields. 32 profiles are supported for each.

Table 5-31 L3AR Actions (Continued)

Action	Type	Configuration	Operands	Description
MuxOutput_MA_FID	MCO	Profile (5 bits)	Many	Mux control action for the L2 lookup keys' FID fields. The muxing of both MuxOutput_MA_FID MCO profile (5b) many lookup keys (L3AR_MA1_FID and L3AR_MA2_FID) are controlled together from one of 32 mux profiles. ***Note:*** These MAC table key FID values produced by L3AR might be overridden by L2 lookup's FID mapping logic.
MuxOutput_VID	MCO	Profile (5 bits)	Many	Mux control action for all ingress and egress VID fields. 32 profiles are supported. This action implements the final routing decision (namely the assignment of EVID1 and EVID2).
MuxOutput_CSGLORT		Profile (5 bits)	Many	
MuxOutput_HASH_ROT	MCO	Profile (5 bits)	Many	Mux control action for the HASH_ROT channel used by the L2 hash function. The HASH_ROT channel is initially set by the FFU Remapping point and by default is left unmodified. By assigning to HASH_ROT from the output of the FFU or a next hop entry, it is possible to override the L2 hash function based on an FFU classification or route action.
MuxOutput_ALU13_OP MuxOutput_ALU46_OP	MCO	Profile (5 bits)	Many	Mux control action for the ALU operand channels. The 12 operand channels (two per ALU) are muxed in two sets, one for ALUs 1-3, the other for ALUs 4-6. 32 profiles are supported for each.
MuxOutput_POL1_IDX MuxOutput_POL2_IDX MuxOutput_POL3_IDX	MCO MCM	Profile (5 bits)	Many	Mux control action for the policers index channels. Policer banks 1 and 2 have MCO-type index muxing; Policer bank 3 has MCM-type index muxing.
MuxOutput_QOS	MCM	Profile (5 bits)	Many	Mux control action for all five QoS fields (24 bits total). All QoS fields share the same muxing profile. The profile also specifies the source QoS field that indexes each of the two POLICER_QOS_MAP{1,2} tables. 32 profiles are supported.

All actions are applied serially from one precedence set to the next. An action specified by an earlier (lower-numbered) precedence set can be overridden by a later precedence set's action.

Note: Most non-mux actions defined here only specify commands or command profiles for downstream units in the pipeline. Such commands are executed once by the downstream logic, based on the final state of the command, regardless of how many actions overwrite the command state.



5.10.4 Outputs

Table 5-32, Table 5-33 and Table 5-34 list the outputs that are consumed, modified, and produced by L3AR.

Rows colored yellow indicate outputs that are modified by L3AR. All others are new channels produced by the L3AR stage.

This section defines:

1. The behavior of actions that have special handling within the L3AR stage.
2. The mux fan-in pathways defined for each output mux action.

Table 5-32 L3AR Generic Fields

Field	Width	Notes
ACTION_FLAGS[51:0]	41	Modified by SetFlags.
ACTION_DATA.W8{A..D}	32	General communication to L2AR and (ultimately) MOD_DATA initialization.
ACTION_DATA.W8E	8	For CPU code assignment.
ACTION_DATA.W8F	8	For L2AR QoS key visibility and QoS field remapping.
ACTION_DATA.W16{A..G}	112	General communication to L2AR and (ultimately) MOD_DATA initialization.

Table 5-33 L3AR Evolution of Names Frame Header Fields

Output Field	Primary Input Field	Width	Notes
QOS	QOS	34	Modified by MuxOutput_QOS, width increased from 24 to 34 bits due to new MAP{1,2}_IDX channels.
QOS.MAP1_IDX		4	Index for 16-entry POLICER_QOS_MAP1 table. (new at the L3AR stage).
QOS.MAP2_IDX		6	Index for 64-entry POLICER_QOS_MAP2 table. (new at the L3AR stage).
DGLORT	ISL_DGLORT	16	Modified by MuxOutput_DGLORT.
SGLORT	ISL_SGLORT	16	Modified by MuxOutput_SGLORT.
CSGLORT	ISL_SGLORT	12	Canonical SGLORT field, set by MuxOutput_CSGLORT.
IVID1	L2_VID1	12	Set by MuxOutput_VID.
IVID2	L2_VID2	12	
EVID1	L2_VID1	12	
EVID2	L2_VID2	12	

Table 5-34 L3AR Outputs Consumed Before L2AR

Output Field	Width	Notes
L2L_CMD_PROFILE	4	L2 Lookup command profile, from SetL2LookupCmdProfile.
ALU13_CMD_PROFILE	5	Command profile for ALUs 1..3, set by SetAlu13CmdProfile.
ALU46_CMD_PROFILE	5	Command profile for ALUs 4..6, set by SetAlu46CmdProfile.
DMASK_CMD_PROFILE	4	DMASK generation command profile, from SetDestMaskCmdProfile.
HASH_PROFILE	4	L2 Hash profile, set by SetHashProfile.
HASH_ROT	20	Layer 2 hash rotation selection.

Table 5-34 L3AR Outputs Consumed Before L2AR (Continued)

Output Field	Width	Notes
MA1_MAC	48	MA1 lookup (DMAC) MAC key field, set by MuxOutput_MA1_MAC.
MA2_MAC	48	MA2 lookup (SMAC) MAC key field, set by MuxOutput_MA2_MAC.
L3AR_MA1_FID1	12	MA{1,2} lookup FID{1,2} fields, which can be overridden in L2 Lookup, set by MuxOutput_MA_FID.
L3AR_MA1_FID2	12	
L3AR_MA2_FID1	12	
L3AR_MA2_FID2	12	
ALU1_X	16	Operands to ALU stage, set by MuxOutput_ALU_OP.
ALU2_X	16	
ALU3_X	16	
ALU1_Y	16	
ALU2_Y	16	
ALU3_Y	16	
ALU4_X	16	
ALU5_X	16	
ALU6_X	16	
ALU4_Y	16	
ALU5_Y	16	
ALU6_Y	16	
POL1_IDX	12	Policer indices, per bank. Set by MuxOutput_POL{1,2,3}_IDX.
POL2_IDX	12	
POL3_IDX	10	

5.10.5 SetFlags

The SetFlags action provides bit-level control over the 52-bit ACTION_FLAGS channel, based on the value and mask operands:

```
ACTION_FLAGS?Œ = ACTION_FLAGS & Mask | Value
```

The output ACTION_FLAGS' value is initialized with ACTION_FLAGS and ripples serially from one action stage to the next. Thus, any bits or fields not overwritten with a SetFlags action remain as seen by the L3AR ACTION_FLAGS key.

The transformation of ACTION_FLAGS that occurs in L3AR represents an important frame processing action for two reasons:

- Identifies frame handling cases based on data fields that are not propagated to L2AR (such as L3 header fields and mapper classification indices).
- Begins the process of formulating ACTION_FLAGS for its eventual destination as MOD_FLAGS, FORWARD_FLAGS, TAIL_FLAGS, and STATS_FLAGS. For a better interpretation of these flags bits, see Section 5.17.7, "Output Flags".

The ACTION_FLAGS channel is 52 bits wide at L3AR. Fixed-function circuitry in the stages that follow expand the channel by another 16 bits before it is passed to L2AR.

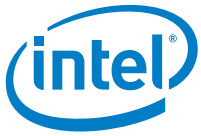


Table 5-35 L3AR Action Flags

Bit Range	Usage Notes
23:0	Originally set from HEADER_FLAGS, these bits eventually become MOD_FLAGS. Therefore, conditions relevant for egress processing need to be flagged in this range.
32:24	Scratch bits in HEADER_FLAGS recommended for propagating intermediate conditions between parsing, L3AR, and L2AR. These bits eventually become FORWARD_FLAGS bits that control specific fixed-function features of the scheduler.
33	StrictDestGlort. Specifies the DGLORT's default strict handling in the GloRT Lookup stage. Determines whether the DGLORT maps to a single DMASK entry, or the frame's L2_HASH value is used in the mapping.
35:34	Flag bits that ultimately control fixed-function PAUSE frame reception logic. Since the parser is critically involved in identifying and parsing these PAUSE frames, it is unlikely that L3AR could use these flag bits for any other purpose over the middle stages of the pipeline
39:36	Flag bits that carry error status conditions from the parser. Bit 39 (ParityError) has a fixed definition throughout the pipeline since any stage with an SRAM can set this bit when an unrecoverable parity error is detected. It is acceptable for microcode to collapse these three error cases to the single bit 39 at L3AR, freeing up three flag bits for other purposes.
51:40	Flag bits set by fixed-function circuitry in the mapper, FFU, or next hop stages. These bits might or might not be relevant to L3AR, depending on the application. It is likely that many of these bits could be redefined for other purposes at this stage.

Note: SetFlags has arbitrary control over the bits in ACTION_FLAGS. Care should be exercised when setting bits that are interpreted with fixed-function logic.

5.10.6 TrapHeader

The TrapHeader action provides a flexible way to save the header fields of a particular frame of interest for later software inspection. When a rule specifies the TrapHeader action by setting TRAP_HEADER_ENABLE to 1b, it identifies one of two storage registers (TRAP_HEADER_IDX) for the frame’s header data. Any subsequent frames that are header-trapped to the same storage index overwrites the previous frame's contents.

Two storage registers are provided mainly to simplify software coherency concerns. It's expected that software would ping-pong between the two registers according to the following procedure:

1. Receive interrupt for rule X specifying TrapHeader with TRAP_HEADER_IDX=A.
2. Change TRAP_HEADER_IDX of all TrapHeader rules to TRAP_HEADER_IDX=1-A.

   Note: Any other headers trapped during this period while TRAP_HEADER_IDX still equals A overwrites the earlier trapped header(s).
3. Read and interpret contents of TRAP_HEADER_DATA[A].
4. Return to step 1, now looking at TRAP_HEADER_IDX=1-A.

The following header fields and other information associated with the frame are stored in TRAP_HEADER_DATA[TRAP_HEADER_IDX].

Table 5-36 L3AR Trap Header Data Fields

Field	Bit Width	Definition
L3_SIP	128	Frame header fields as produced by the parser.
L3_DIP	128	
L3_PROT	8	
L4_SRC	16	
L4_DST	16	
IVID1	12	As muxed by MuxOutput_VID.
EVID1	12	
NEXTHOP_IDX	16	Next hop table index, if a lookup was performed (undefined otherwise).

In addition, the L3AR rule number that last set the TRAP_HEADER_ENABLE command bit to 1b is stored in L3AR_TRAP_HEADER_RULE[TRAP_HEADER_IDX].The TrapHeader action is provided mainly for ICMP redirect support. However, it might also be useful for FPGA-assisted load balancing and CN algorithms.

5.10.7 MuxOutput

The tables that follow list the mux pathways available for the L3AR MuxOuput actions. These actions are described in an abbreviated form, with the table columns indicating the following properties:

- **Set Name** — A descriptive identifier associated with the output mux set. These names correspond to names that appear in the action RAM MuxOutput specification fields.
- **Type** — Indicates the type of the mux action. A type MCO action implies that the MUX_PROFILE entry contains a constant *Value* field of width *Width*. A type MCM action's MUX_PROFILE entry additionally includes a mask field.
- **N_{pro}** — Number of profile entries defined for this output (span of allowable *Src* values).
- **Output#** — For multi-output mux sets, a numeric ID is assigned to each output channel in the set to unify the MUX_PROFILE encoding formats.
- **Output Channe**l — Name of the output channel to be muxed.
- **Width** — Bit width of the output channel.
- **N_{src}** — Maximum number of input source channels supported in the mux profile encoding for the particular output channel.
- **Src#** — Specific source select values that are configured in the output's profile entry.
- **Source Channel** — Input channel that is mapped to the output channel when the specific *Src* value is configured in the mux profile entry.

For all MCO cases, as described in the Action Resolution section, the maximum *Src* value always selects the constant value configured in each mux profile entry.

5.10.7.1 GloRTs

Typically, the L3AR muxes would assign the final SGLORT and the initial (flood) DGLORT values associated with the frame.

Table 5-37 L3AR GloRT Data

Set Name	Type	N_{pro}	Output Channel	Width	N_{src}	Src#	Source Field
DGLORT	MCM	32	DGLORT	16	8	0	ISL_DGLORT (post-association)
						1	FUU_DATA.W24[15:0]
						2	FUU_DATA.W16A
						3	FUU_DATA.W16B
						4	NEXTHOP_DATA.W16A
						5	NEXTHOP_DATA.W16D
SGLORT	MCM	32	SGLORT	16	8	0	ISL_SGLORT (post-association)
						1	FUU_DATA.W24[15:0]
						2	FUU_DATA.W16A
						3	FUU_DATA.W16B
						4	NEXTHOP_DATA.W16D

5.10.7.2 Action Data W8{A..D}

Generic ACTION_DATA channels that ultimately become MOD_DATA fields on the forward channel:

Table 5-38 L3AR Generic Action Data

Set Name	Type	N_{pro}	Channel Number	Channel	Width	N_{src}	Src#	Source Field
W8{A..D}	MCM	32	0	ACTION_DATA.W8A	8	4	0	NEXTHOP_DATA.W8A
							1	FFU_DATA.W8A
							2	NEXTHOP_TAG
			1	ACTION_DATA.W8B	8	4	0	NEXTHOP_DATA.W8B
							1	FFU_DATA.W8B
							2	ACTION_FLAGS[23:16]
			2	ACTION_DATA.W8C	8	4	0	NEXTHOP_DATA.W8C
							1	FFU_DATA.W8C
							2	ACTION_FLAGS[31:24]
			3	ACTION_DATA.W8D	8	4	0	NEXTHOP_DATA.{W4B,W4A}
							1	FFU_DATA.W8A
							2	FFU_DATA.W8B
							3	FFU_DATA.W8C

5.10.7.3 Action Data W8{E,F}

Table 5-39 L3AR Generic Communication to L2AR

Set Name	Type	N_{pro}	Channel Number	Output Channel(s)	Width	N_{src}	Src#	Source Field
W8E	MCM	32	0	ACTION_DATA.W8E	8	8	0	FFU_DATA.TAG1A[7:0]
							1	FFU_DATA.TAG1B[7:0]
							2	FFU_DATA.TAG2A[7:0]
							3	FFU_DATA.TAG2B[7:0]
							4	NEXTHOP_TAG
W8F	MCM	32	1	ACTION_DATA.W8F	8	8	0	(QOS.W4, QOS.ISL_PRI) (post-MuxQOS)
							1	(QOS.L2_VPRI1, QOS.L2_VPRI2) (post-MuxQOS)
							2	QOS.L3_PRI (post-MuxQOS)

5.10.7.4 Action Data W16{A..C}

Output mux sets defined for L2 hash selection, MOD_DATA initialization, and other configurable uses:

Table 5-40 L3AR W16 Data

Set Name	Type	N_{pro}	Output Channel(s)	Width	N_{src}	Src#	Source Field
W16ABC	MCO	32	ACTION_DATA.W16A	16	8	0	FFU_DATA.W16A
						1	FFU_DATA.TAG1A (12 bits)
						2	FFU_DATA.TAG2A (12 bits)
						3	L2_VID1 (12 bits)
			ACTION_DATA.W16B	16	8	0	FFU_DATA.W16B
						1	FFU_DATA.TAG2B (12 bits)
						2	FFU_DATA.TAG2B (12 bits)
						3	L2_VID2 (12 bits)
			ACTION_DATA.W16C	16	4	0	FFU_DATA.(W8C,W8B)
						1	NEXTHOP_DATA.W16D
						2	L3_CHECKSUM

Table 5-40 L3AR W16 Data (Continued)

Set Name	Type	N_{pro}	Output Channel(s)	Width	N_{src}	Src#	Source Field
W16DEF	MCO	32	ACTION_DATA.W16D	16	8	0	DMAC[15:0]
						1	NEXTHOP_DATA.W16A
						2	NEXTHOP_DATA.W16D
						3	L3_DIP-derived[15:0] (L3 multicast case)
						4	L3_SIP[15:0] (overloaded for DMAC)
			ACTION_DATA.W16E	16	8	0	DMAC[31:16]
						1	NEXTHOP_DATA.W16B
						2	L3_DIP-derived[31:16] (L3 multicast case)
						3	L3_SIP[31:16] (overloaded for DMAC)
			ACTION_DATA.W16F	16	8	0	DMAC[47:32]
						1	NEXTHOP_DATA.W16C
						2	L3_DIP-derived[47:32] (L3 multicast case)
						3	L3_SIP[47:32] (overloaded for DMAC)
W16GH	MCO	32	ACTION_DATA.W16G	16	8	0	FFU_DATA.TAG2A (12 bits)
						1	FFU_DATA.TAG2B (12 bits)
						2	FFU_DATA.W16A
						3	FFU_DATA.W16B
						4	NEXTHOP_DATA.W16D
						5	NEXTHOP_IDX
			ACTION_DATA.W16H	16	8	0	FFU_DATA.TAG1A (12 bits)
						1	FFU_DATA.TAG1B (12 bits)
						2	FFU_DATA.W16A
						3	NEXTHOP_DATA.W12A (12 bits)
						4	NEXTHOP_DATA.W12B (12 bits)
						5	NEXTHOP_DATA.W16A
						6	NEXTHOP_DATA.W16D

5.10.7.5 L2 Lookup Channels

L3AR is responsible for specifying the frame's final VID associations, the MAC table lookup keys, and the VLAN/STP filtering table indices for L2 filtering.

Table 5-41 L3AR VID Associations

Set Name	Type	N_{pro}	Channel Number	Output Channel(s)	Width	N_{src}	Src#	Source Field
MA1_MAC	MCO	32	0	MA1_MAC (DMAC)	48	8	0	L2_DMAC
							1	NEXTHOP_DATA.(W16C,W16B,W16A)
							2	L3_DIP-derived (L3 multicast case)[1]
							3	L3_DIP[47:0]
							4	L3_SIP[47:0][2]
							5	*Undefined*
							6	*Undefined*
							7	Profile-specified constant
MA2_MAC	MCO	32	0	MA2_MAC (SMAC)	48	4	0	L2_SMAC
							1	L3_SIP[95:48][2]
							2	*Undefined*
							3	Profile-specified constant
MA_FID	MCO	32	0	L3AR_MA2_FID1	12	4	0	IVID1
							1	FIELD16C[11:0][3]
			1	L3AR_MA2_FID2	12	4	0	IVID2
							1	{FIELD16D,FIELD16C}[23:12][3]
			2	L3AR_MA1_FID1	12	4	0	EVID1
							1	FIELD16C[11:0][3]
							2	NEXTHOP_DATA.(W8A,W16A)[11:0][3]
			3	L3AR_MA1_FID2	12	4	0	EVID2
							1	{FIELD16D,FIELD16C}[23:12][3]
							2	NEXTHOP_DATA.W12A

Table 5-41 L3AR VID Associations (Continued)

Set Name	Type	N_{pro}	Channel Number	Output Channel(s)	Width	N_{src}	Src#	Source Field
VID	MCO	32	0	IVID1	12	8	0	L2_VID1
							1	L2_VID2[4]
							2	NEXTHOP_DATA.W12A
								FFU_DATA.W16A[11:0]
							4	FFU_DATA.W16B[11:0]
			1	IVID2	12	8	0	L2_VID2
							1	L2_VID4[4]
							2	NEXTHOP_DATA.W12A
							3	FFU_DATA.W16A[11:0]
							4	FFU_DATA.W16B[11:0]
			2	EVID1	12	8	0	L2_VID1
							1	L2_VID2[4]
							2	NEXTHOP_DATA.W12A
							3	FFU_DATA.W16A[11:0]
							4	FFU_DATA.W16B[11:0]
			3	EVID2	12	8	0	L2_VID2
							1	L2_VID1[4]
							2	NEXTHOP_DATA.W12A
							3	FFU_DATA.W16A[11:0]
							4	FFU_DATA.W16B[11:0]
							5	NEXTHOP_DATA.W12B
CSGLORT	MCM	32	0	CSGLORT	16	2	0	ISL_SGLORT[5]
							1	L3_HASH[6]
							2	SGLORT[7]

1. Assignment from the IPv6 L3_DIP EUI-64 field is as follows:

```
MA1_MAC[23:0]= L3_DIP[23:0]
MA1_MAC[47:24] = L3_DIP[63:40] ^ 0x020000 (toggle bit 57, U/L flag)
```

2. Available for overloading with encapsulated (inner) DMAC/SMAC fields.
3. Intended use: PBB ISID.
4. Provided as a way to swap VID1 and VID2 fields, due to concern that the parser might not be able to assign these appropriately in the context of double-8100 tagged frames.
5. Canonical, relies on MCM masking support.
6. Provided for (SMAC, Hash(SIP)) security support.
7. Output of L3AR_SGLORT_PROFILE transform.

5.10.7.6 Layer 2 Hash Rotation

The HASH_ROT value initially set by the FFU remapping point for the L3 hash calculation might be overridden by L3AR.

Table 5-42 L3AR HASH_ROT Operands

Set Name	Type	N_{pro}	Output Channel(s)	Width	N_{src}	Src#	Source Field
HASH_ROT	MCO	32	HASH_ROT_MANTISSA	16	4	0	ISL_DGLORT (post-association)
						1	FFU_DATA.W24[15:0]
						2	FFU_DATA.W16A
			HASH_ROT_EXPONENT	4	4	0	FFU_DATA.W16B
						1	NEXTHOP_DATA.W16A
						2	NEXTHOP_DATA.W16D

5.10.7.7 ALU Operands

The six ALU operand pairs (X, Y) are muxed in two sets. All ALUs have identical sources of X and Y channels inputs.

Table 5-43 L3AR ALU Operands

Set Name	Type	N_{pro}	Output Channel(s)	Width	N_{src}	Src#	Source Field
ALU13_OP	MCO	32	ALU1_X ALU2_X ALU3_X	16	16	0..15	See X operand in Table 5-44.
			ALU1_Y ALU2_Y ALU3_Y	16	16	0..15	See Y operand in Table 5-44.
ALU46_OP	MCO	32	ALU4_X ALU5_X ALU6_X	16	16	0..15	See X operand in Table 5-44.
			ALU4_Y ALU5_Y ALU6_Y	16	16	0..15	See Y operand in Table 5-44.

Each ALU's X and Y operands are selected from the following mux sources:

Table 5-44 Sources for ALU X and Y Operands

Src#	X Source Channel	Y Source Channel
0	FFU_DATA.W16A	FFU_DATA.W16B
1	L3_LENGTH	NEXTHOP_DATA.{W8B,W4A}
2	CSGLORT[1]	NEXTHOP_DATA.{W8B,W8C}
3	SGLORT[2] DGLORT	NEXTHOP_DATA.W12A
4	<SGLORT[15:8],DGLORT[15:8]>[3]	NEXTHOP_DATA.W16A
5		ACTION_DATA.W16H
6	L3_HASH[15:0][4]	EVID1
7	L3_CHECKSUM	EVID2
8	L3_TTL (8b)	MA1_GLORT (after L2 lookup)
9	SRC_PORT (7b)	MA2_GLORT (after L2 lookup)
10	IVID1	L2L_ETAG1 (12b, after L2 Lookup)
11	IVID2	L2L_ETAG2 (12b, after L2 lookup)
12	L3_DIP[15:0]	L2_HASH_B[4]
13	L3_DIP[31:16]	L2_HASH_RANDOM
14	*Unused (gated)*	
15	*Per-profile constant value*	

1. For comparison of the canonical SGLORT to the MA1_GLORT for port security.
2. Provided for mapping to L2F_DMASK_IDX channels to support SGLORT-based distribution trees. These bits identify a DMASK in L2F that would prune the distribution appropriately for this FM5000/FM6000's forwarding hop.
3. Similar to footnote in that it provides support for (S,D)GLORT-based distribution trees. Notation indicates bit interleaving: for i=0..7:

```
L3AR_DMASK_IDX1[2  i] = SGLORT[8+i]
L3AR_DMASK_IDX1[2  i+1] = DGLORT[8+i]
```

4. Provides pathways between the frame's hash values and L2F indexing and/or the L2AR key.

5.10.7.8 Policer Indices

One output mux set is defined per policer bank to select the bank's source counter/policer index. For a discussion of how these indices are used, see Section 5.13, "Policers".

Table 5-45 L3AR Policer Indices

Set Name	Type	N_{pro}	Channel Number	Output Channel(s)	Width	N_{src}	Src#	Source Field
POL1_IDX	MCO	16	0	POL1_IDX	12	16	0..7	From L3AR inputs
							8..9	From L2 lookup outputs
							15	(Profile constant)
POL2_IDX	MCO	16	0	POL2_IDX	12	16	0..7	From L3AR inputs
							8..9	From L2 lookup outputs
							15	(Profile constant)

Table 5-45 L3AR Policer Indices (Continued)

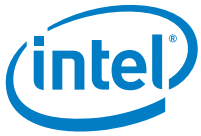
Set Name	Type	N_{pro}	Channel Number	Output Channel(s)	Width	N_{src}	Src#	Source Field
POL3_IDX	MCM	16	0	POL3_IDX	10	8	0	SRC_PORT
							1	FFU_DATA.TAG1A
							2	FFU_DATA.TAG1B
							3	FFU_DATA.TAG2A
							4	FFU_DATA.TAG2B
							5	ALU1_Z[9:0][1]
							6	ALU2_Z[9:0][1]
							7	ALU3_Z[9:0][1]

1. Muxed downstream in the pipeline. The MCM mask and constant value do not apply for this mux case. The entire ALU3_Z value is selected instead.

The specific source channel mappings for policer banks 1 and 2 are defined as follows:

Table 5-46 L3AR Policer Index Source

Src	Source Channel (Bank 1)	Source Channel (Bank 2)	Description
0	FFU_DATA.TAG2A	FFU_DATA.TAG2B	FM4000 series compatible.
1	FFU_DATA.TAG1A	FFU_DATA.TAG1B	Tag values prior to remapping point.
2	FFU_DATA.W16A[11:0]	FFU_DATA.W16B[11:0]	Alternative FFU index source, Enables use of TAGs for other purposes.
3	IVID1	EVID1	As muxed after L3AR.
4	IVID2	EVID2	As muxed after L3AR.
5	ITAG1	ETAG1	From L2L_{E,I}VID1_TABLE. Enables policing per ingress/egress VLAN (or classes of VLANs).
6	ITAG2	ETAG2	From L2L_{E,I}VID2_TABLE. Enables policing per ingress/egress VLAN (or classes of VLANs).
7	MAC1_TAG	MAC2_TAG	Zeroed if !HIT.
8	ALU1_Z[11:0]	ALU4_Z[11:0]	Result of ALU 1.
9	ALU2_Z[11:0]	ALU5_Z[11:0]	Result of ALU 1.
0	ALU3_Z[11:0]	ALU6_Z[11:0]	Result of ALU 1.



5.10.7.9 QoS

The MCM-type mux action defined for the QoS collection of fields is provided in the sections that follow in a somewhat different format from those previously defined.

Table 5-47 L3AR QoS Channel MUX Profiles

Field	Width	Description
Value	24	Constant value to OR into the output post-masking and post-muxing.
Mask	24	Mask to AND with the mux output post-muxing.
Select_QOS.ISL_PRI	3	Selects mux input for ISL_PRI.
Select_QOS.L2_VPRI1	3	Selects mux input for L2_VPRI1.
Select_QOS.L2_VPRI2	3	Selects mux input for L2_VPRI2.
Select_QOS.L3_PRI	3	Selects mux input for L3_DSCP.
Select_QOS.W4	3	Selects mux input for the generic W4 field.
Select_QOS.MAP{1,2}_IDX	3	Selects the source QoS field for each of the policers' QOS_MAP{1,2} mark-down mapping tables.

Table 5-48 L3AR MUX Source Field Definitions per QoS Subfield

Src	Source Channel					Description
	ISL_PRI	L2_VPRI1	L2_VPRI2	W4	L3_PRI	
0	ISL_PRI	L2_VPRI1	L2_VPRI2	W4	L3_PRI	Default.
1	FFU_DATA.W16A[15:12]				FFU_DATA.W8A	Bottom 12 bits intended for VID.
2	FFU_DATA.W16B[15:12]				FFU_DATA.W8B	Bottom 12 bits intended for VID.
3	FFU_DATA.W8C[3:0]				FFU_DATA.W8C	Generic QoS association.
4	FFU_DATA.W8C[7:4]				{FFU_DATA.W8C[7:4], L3_PRI[3:0]}	Generic QoS association.
5	NEXTHOP_DATA.W8C[3:0]				NEXTHOP_DATA.W8C	Generic QoS assignment from wide-entry NEXTHOP_DATA.
6	NEXTHOP_DATA.W4A					4-bit generic wide-entry NEXTHOP_DATA source.
7	W4	L2_VPRI2	L2_VPRI1	L3_PRI[7:4]	W4	Cross-field assignment.

5.11 L2 Lookup

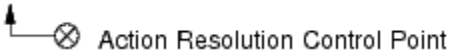
The L2 lookup stage performs source and destination MAC address lookups, which are essential for Ethernet forwarding and learning. The 64-Kb-entry MAC address hash table (MAC table) maps each frame's destination MAC address and, optionally, its source MAC address to corresponding source and destination GloRTs. These GloRTs are then processed by downstream stages to determine a L2 forwarding distribution (from the destination GloRT) and security and learning actions (from the source GloRT).

The MAC table is unique in the chip for the complexity of its management and in its support for hardware self-modification. It supports the following features:

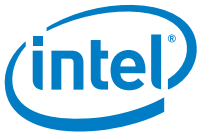
- Partitioning of the MAC address space with a 24-bit Forwarding ID (FID).
- Four different entry precedence levels, which, for example, can be used to distinguish learned entries from software-locked entries.
- Two statically configured table modes trading off number of entries (32-Kb versus 64-Kb) for performance (fully-provisioned versus opportunistic source MAC lookups).
- A flexible writeback mechanism, which, under the control of the L2AR stage, provides automatic MAC address learning and aging. The mechanism can also be used to record and report entries on which security violations are observed.
- A sweeper mechanism providing hardware acceleration of common table-wide searches and entry transformations.
- Support for reporting all table changes to software through an event notification FIFO.

The diagrams that follow make use of the following conventions:

- In the L2 lookup stage, as in other stages of frame processing logic, fixed-function operations are controlled by command bits mapped from a profile value specified by L3AR. The L3AR SetL2LookupCmdProfile action specifies an L2L_CMD_PROFILE value which is then mapped to specific command bits by the L2L_CMD_PROFILE_TABLE registers. In the diagrams that follow, points controlled by these mapped command bits are indicated with the following notation:



- All other inputs can be considered part of the frame pipeline data path. In most cases, they are indirectly controlled by L3AR (in the form of muxing, selection, etc.).
- Red coloring indicates internal non-configurable data dependencies in the L2 lookup data flow.



5.11.1 Basic Architecture

A 48-bit MAC address is too wide to map by a direct lookup table. Whereas application considerations motivate the use of TCAM and BST structures for the L3 address lookups, properties specific to the L2 lookup motivate a hash table implementation. The hash table architecture provides an excellent area and power efficiency for large table sizes with wide exact-match keys, especially when two lookups per frame must be performed with one (the source lookup) requiring only best-effort provisioning.

The FM5000/FM6000's MAC table is organized as 16 sets, each with 4096 entries. The sets can optionally be paired for performance reasons so that in some applications it effectively offers only eight 4-Kb sets. Thus, subject to key hashing statistics, the table offers an absolute maximum of either 64-Kb or 32-Kb entries, depending on its performance mode. Unlike typical software hash table implementations, the MAC table uses an independent hash function to calculate each set's key index. Doing so provides the best possible statistical table use.

Each MAC table entry consists of a 72-bit key, a 2-bit precedence tag, and 36 bits of data payload. To perform a lookup, the input key is first hashed to give the 12-bit index read in each set. The entries from all 16 (or eight) sets are then read, and among all entries with matching keys, the one with the highest precedence value is returned. Up to two keys can be looked up per frame, although in the table's maximum capacity configuration (64-Kb entries), only the first (destination) lookup is guaranteed to succeed.

Throughout this datasheet the two lookup operations are referred to as MA1 and MA2. In the MAC table's expected application usage, the MA1 lookup searches for the destination MAC key and the MA2 lookup searches for the source MAC key.

The MAC table key has the bit structure shown in Table 5-49.

Table 5-49 MAC Table Key Structure

Field	Width	Description
MAC	48	Set by L3AR's MuxOutput action.
FID1	12	Primary forwarding ID. Identifies the forwarding domain to which the entry belongs, nominally mapped from VID1 by the VLAN tables in the L2 Lookup stage.
FID2	12	Secondary forwarding ID, nominally mapped from VID2 in a manner symmetric to FID1.
Prec	2	2-bit precedence tag associated with the key. This field is not specified as part of the input key, but it does play an important role in the matching operation.

The data payload of each entry has the bit structure shown in Table 5-50.

Table 5-50 MAC Table Data Structure

Field	Width	Description
GLORT	16	Primary output result of the lookup operation. Uniquely identifies the virtual port entity associated with the entry's MAC and FID fields.
TAG	12	Software-defined tag field available as an input key to the L2AR stage.
DATA	8	Software-defined data field available as an input to data muxes in the L2AR stage.

5.11.2 FID Mapping

The 12-bit FID1 (spanning-tree forwarding ID) and FID2 fields associated with each key are mapped from either the EVID{1,2} channels (for the MA1 lookup) or IVID{1,2} (for the MA2 lookup), as muxed by L3AR.

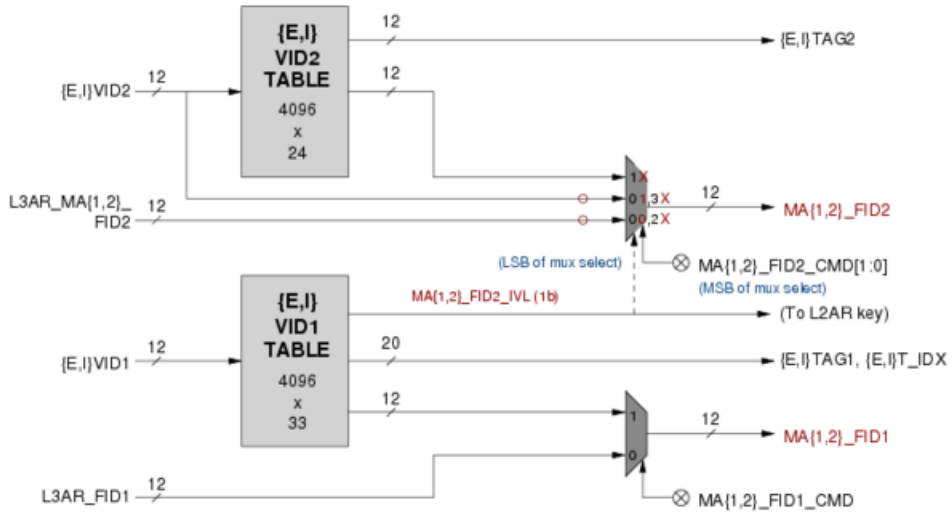


Figure 5-24 L2 Lookup FID Mapping

As shown in Figure 5-24, the FID fields can also be taken directly from the L3AR_MA{1,2}_FID{1,2} values as specified by L3AR, and in the case of FID2, a bit from the corresponding FID1_TABLE entry might select either the mapped FID2 or the L3AR_MA{1,2}_FID2 value on a per-VID1 basis. The MA1_FID2_CMD and MA2_FID2_CMD bits control this behavior as follows. Although noted here for MA1_FID2_CMD, the MA2 case is similar.

Table 5-51 L2 Lookup FID Table

MA1_FID2_CMD[1:0]	MA1_FID2_IVL	MA1_FID2
0	0	L3AR_MA1_FID2
0	1	EVID2
1	X	MA1_FID2_TABLE[EVID2]
2	X	L3AR_MA1_FID2
3	X	EVID2

In addition to the FID fields needed for the MAC table lookup operations, generic VID-associated TAG and IDX fields are mapped in these tables. These ITAG{1,2}, ETAG{1,2}, and {I,E}T_IDX fields are available to downstream units, such as the ALUs, L2F stages, and L2AR, for a variety of microcode-programmable uses.

5.11.3 Performance Versus Capacity

The MAC table supports two modes of operation, statically configured by the FullTableMode field in MAC_TABLE_CFG. When FullTableMode is set to 1b, the table is organized as 16 independent 4-Kb-entry sets. In this mode, the table provides sufficient access bandwidth to support MA1 lookups at the maximum frame rate. However, in this mode, the table is unable to also perform MA2 lookups at the maximal frame rate. Instead the table relies on average-case traffic loads and statistical memory bank alternation in the implementation of each set to support two lookups per frame.

When FullTableMode is set to 0b, the table's 16 sets are grouped into pairs, providing eight dual-ported 4-Kb sets. The redundancy of this mode enables the MAC table to sustain MA1 and MA2 lookups at the maximum frame rate at the expense of reducing the total table capacity to 32-Kb entries. In this split-mode of operation, the addressing of the table is aliased such that any hardware or software write to any index (i <32,768) applies to both paired sets.

5.11.4 Key Precedence

The role of key precedence in the MAC table's lookup function is shown in Figure 5-25.

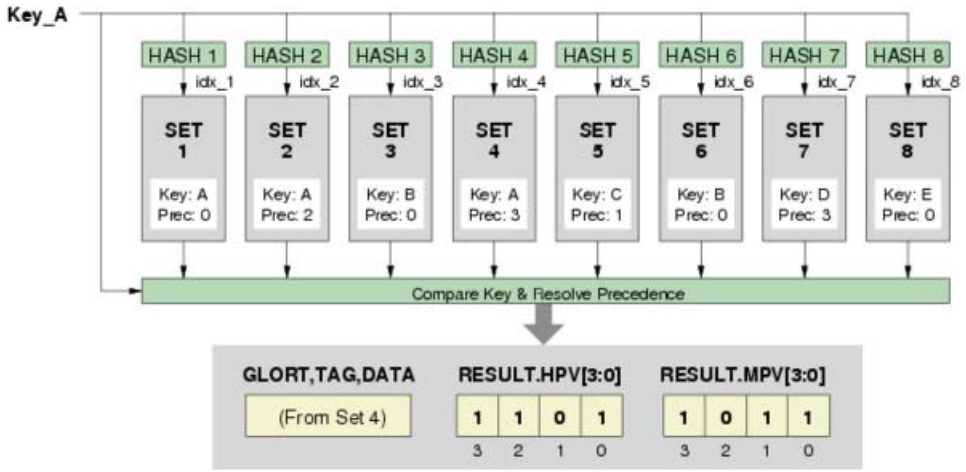


Figure 5-25 MAC Table Lookup Function

Among all entries with matching keys, the entry with the highest precedence value is returned. In this example, this is set 4's entry with a precedence value of 3, while all other matching sets have lower precedence values.

Figure 5-25 also shows the Hit-Precedence-Vector (HPV) and Miss-Precedence-Vector (MPV) values that are produced with each lookup operation. These outputs indicate which precedence classes are represented among all the entries that matched (HPV) and did not match (MPV) the input key. The HPV and MPV vectors are passed on to L2AR so that it can:

1. Determine how to interpret the lookup's hit result, if there was one.
2. Decide whether to clobber an existing entry when writing the MA2 key back to the table.

In cases of precedence ties, the higher numbered set always wins. For example, when key B is looked up with the set state as shown in Figure 5-25, the payload associated with set 6's entry is returned (not set 3's).

5.11.5 Source Lookup Writeback

Unlike any other table in the device, the MAC table has the capability to write new entries and update old entries without software intervention. This capability is referred to as entry writeback. The feature is only supported on the MA2 (source MAC) lookup operation.

Following a successful MA2 lookup, two indices are recorded: the index corresponding to the hit entry whose data was returned by the lookup (hit index), and the index of the lowest precedence entry that did not match the key (miss index). Based on higher-level application programming, the L2AR stage determines whether the MA2 key should be written back to one of these two indexed locations. To write the entry back to its existing location in the table (at its hit index), L2AR sets the MAC_WriteBack flag in its TAIL_FLAGS output. To write the entry into the miss index location (intended to be the best available location for a new entry), L2AR sets the MAC_WriteNew bit in TAIL_FLAGS.

An example application of the WriteBack behavior is to implement age bit refresh updates. The WriteNew behavior is used to learn new entries in the table or to record notable exceptional events, such as SMACs on which security violations are observed.

In the split table mode of operation, each index maps to two sets, one in each half of the table. Each time an entry is written back into the table, it is stored identically in both halves so on subsequent lookups, both MA1 and MA2 lookups see the new or updated entry.

Note: The hit and miss indices derived from the MA2 lookup are not exposed to the L2AR stage and therefore might not be reassigned by microcode.

5.11.6 Output Handling

Figure 5-26 shows the handling of the MAC table's MA1_GLORT and MA2_GLORT outputs. Channels colored red are produced and consumed internally within the L2 lookup block.

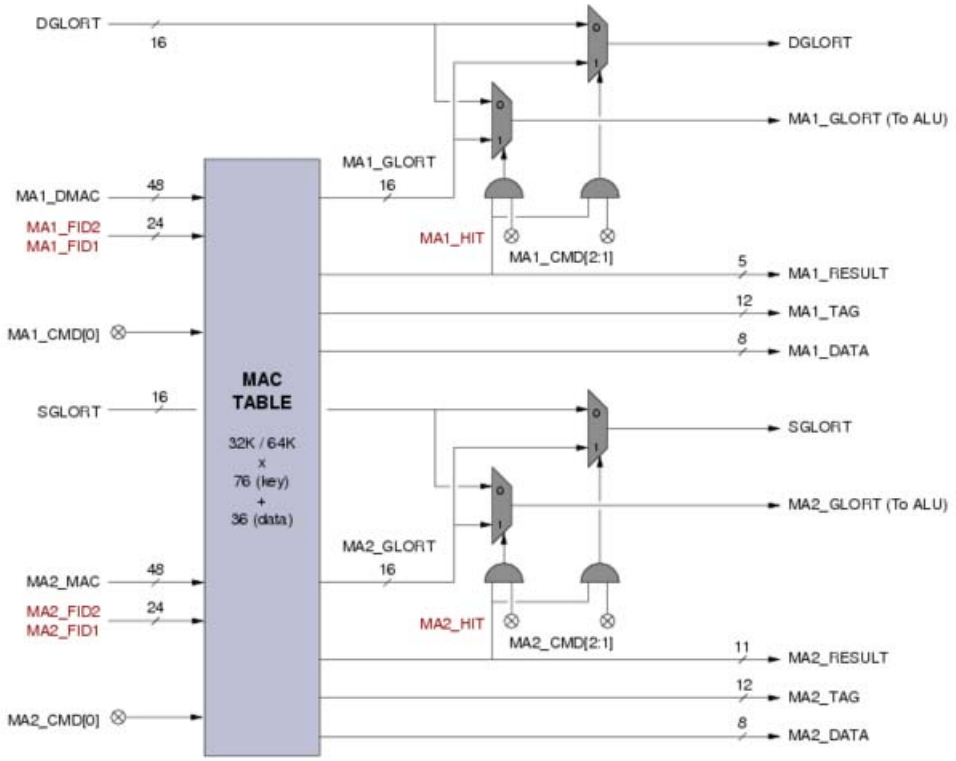


Figure 5-26 MAC Table GloRT Outputs

The MA1 (DMAC) and MA2 (SMAC) lookup pathways are very similar. As previously explained, the two lookup operations differ in only two respects:

- MA1 does not support a best-effort lookup mode, whereas the MA2 lookup can be no better than best-effort. This distinction is only relevant in the full-table mode of operation. Otherwise, the table is provisioned with sufficient bandwidth for both lookups.
- Only the MA2 lookup supports entry writeback.

Otherwise, outside of the MAC table, the MA1 and MA2 pathways are completely symmetric. Based on the L3AR-specified command bits, the GloRT output of a lookup hit is assigned to the DGLORT or SGLORT output channels. An independent control bit determines whether the entry's GloRT field is assigned to the ALU Y operand pathway.

The interface presented to the rest of the FPP does not depend on the table's split versus full mode configuration. The only observable effects of the table's configuration relate to the capacity and performance of these lookup operations.

5.11.7 Command and Result Encodings

The MA1_CMD and MA2_CMD fields are specified with each frame by L3AR. As with all other L2 lookup command bits, they are mapped from the frame's L2L_CMD_PROFILE and have the following encodings:

Table 5-52 L2 Lookup Command Encodings

Bit	Value	Definition
0	0b	Do not perform lookup.
	1b	Must perform lookup (MA1) or attempt to perform best-effort lookup (MA2).
1	0b	Map input DGLORT/SGLORT to output DGLORT/SGLORT unconditionally.
	1b	Map DGLORT/SGLORT from lookup result if there was a matching entry.
2	0b	Map the input DGLORT/SGLORT to the ALU Y operand channel. ***Note:*** This mapping is inconsequential unless the L3AR block has correspondingly selected the L2L_MAn_GLORT input as an ALU input operand.
	1b	Map the L2 Lookup's Y ALU operand channel from the lookup GloRT result if there was a hit, or from the input DGLORT/SGLORT otherwise.

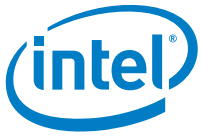
In Figure 5-26, the MA1_RESULT and MA2_RESULT outputs represent information about the status of the lookup operations that is passed on to L2AR on the following channels:

Table 5-53 L2 Lookup Action Channels

Sub-Field	Width	Definition
Lookup	1	One indicates whether a lookup was performed. For MA1, this is the same as the input MA1_CMD. For MA2, this equals MA2_CMD unless the lookup was skipped.
HPV	4	HPV. Indicates which precedence levels had matching entries.
MPV	4	(MA2 Only) MPV. Indicates which precedence levels had non-matching entries. ***Note:*** In general, HPV != ~MPV.
WriteBackEnabled	1	(MA2 Only) Indicates whether this key's lookup entry might be written back to the table. Writeback can be disabled due to two reasons: 1. Management access (or sweeper) clobber protection. 2. A previous table entry write associated with this frame's source port has not yet completed. This bit is communicated to L2AR on ACTION_FLAGS[MA2_WriteBackEnabled].
WriteNewEnable	1	(MA2 Only) Indicates whether a new entry may be written into the table associated with this key. A new entry may be disallowed for the following reasons: 1. Management access (or sweeper) clobber protection. 2. A previous table entry write associated with this frame's source port has not yet completed. This bit is communicated to L2AR on ACTION_FLAGS[MA2_WriteNewEnabled].

The MA1_HIT and MA2_HIT intermediate signals referenced in Figure 5-26 are calculated from each lookup operation's HPV vector and the corresponding value of HitPrecMask in the MAC_TABLE_CFG register. The HitPrecMask configuration determines whether a matching entry of each precedence level is considered a valid hit in the table. It is calculated as:

```
HIT = (HPV & HitPrecMask != 0)
```



Typically, HitPrecMask is set to 0xC0 such that the hit condition corresponds to an OR over the top two bits of the HPV vector. For example, MA1_RESULT.HPV[3:2]=0b. The lowest precedence level, corresponding to HPV[0], is reserved for invalid (or empty) entries, so is not included in the hit determination. The second lowest precedence level, corresponding to HPV[1], might be reserved for provisional entries. For example, entries that have been learned into the table but are not yet validated for use.

The unit preserves up to two indexes; one for updates and one for new entries. The update index is the index for the highest precedence hit. For new entry index is the index of the lowest precedence miss. The pseudo code for MA2 lookup is as follows (assume N is 16 for full mode and 8 for half mode).

```
foreach i (0..N)
  if ( the lookup matches the table value,
       AND HitPrecedenceMask[prec] is set AND
       and this is the highest hit precedence seen so far )
           Save hit address
           Set MA2_WriteBackEnabled
  else if ( the HitPrecedenceMask[prec] is zero AND
       this is the lowest miss precedence seen so far )
            Save the miss address
            Set MA2_WriteNewEnabled
```

The MA2_WriteBackEnabled and MA2_WriteNewEnabled bits are produced by the MA2 lookup and added to ACTION_FLAGS.

5.11.8 Direct Management Access

The MAC table memory space is divided into 16 4-Kb ranges, one for each FullTableMode set. The key and payload fields are accessed atomically as a single four-word unit. In terms of four-word entry indices, sets and entries within the set are addressed as follows:

```
   set = idx[15:12]
entry = idx[11:0]
```

In split table mode, the top-most address bit is ignored. Writes with idx[15] as either 0b or 1b atomically affects both sets idx[14:12] and idx[14:12]+0x8. Reads can also be issued to either aliased set in this mode.

5.11.9 Table Sweepers

To accelerate common MAC table management operations, an autonomous sweeper mechanism is provided for software use. The mechanism consists of two sweeper timers that sequence through all entries in the table at independently configured rates. Each entry read from the table is evaluated against the keys of up to 16 sweeper rules divided between the two timers. Any matching rule applies one or more configured actions to the entry.

A sweeper can be configured to process all entries in the table (or some configured subset of entries). Alternatively, a sweeper can be configured to iterate over a specified range of MAC table entries indefinitely.

Once a sweeper timer has processed the stop index entry, it posts an interrupt in the SWEEPER_IP register.

Each sweeper rule's match condition is configured as a standard TCAM (KeyInvert, Key) pair that it applies to the following fields:

- MAC Entry key (MAC, FID1, FID2, Prec)
- MAC Entry payload (DATA, TAG)
- Sweeper timer number (Timer:1)
- MAC Entry error bit (Error:1), indicating whether the entry contains an uncorrectable ECC error. A sweeper rule applies its action to a given entry only if all fields of the key match.

The sweepers support three types of actions:

- **Field Replacement** — Any of the matching entry's GLORT, DATA, TAG, or Prec fields can be replaced by a configured constant value. The constant assignment is restricted by a bit mask per field:

  $FIELD_{new}$ = FIELD & ReplaceMaskFIELD | ReplaceValueFIELD

- **Entry Reporting** — Matching entries can be posted to a 512-entry sweeper FIFO. The entry index and the sweeper rule number are recorded in each FIFO entry. If the 512-entry FIFO is full when a sweeper attempts to post a new entry, the sweeper state machine blocks it until space is available. Software dequeues entries from the FIFO by writing to the SWEEPER_FIFO_HEAD register.
- **Masked Field Decrement** (DATA field only) — In addition to a constant masked value assignment, the 8-bit DATA field supports a decrement-by-1 transformation. With this action, the complete expression specifying the transformed DATA value becomes:

  $DATA_{new}$ = DATA & ReplaceMaskDATA | ReplaceValueDATA |
  ((DATA & DecrementMaskDATA) - 1) & DecrementMaskDATA

The two sweeper timers have independent configurations and can operate concurrently. Their rate of iteration through the table is configured by the L2LookupPeriod0 and L2LookupPeriod1 fields in the LSM_SWEEPER_CFG register. The sweepers have lower priority access to the table than the hardware's lookup and writeback operations, so the actual sweep rate might be slower than expected. Basic state and statistics information is reported in the SWEEPER_STATUS register to facilitate debugging.

More than one sweeper rule can transform the same table entry. Each sweeper rule is evaluated and applied sequentially. The following example shows the behavior of a single sweeper timer when it accesses some MAC table entry at index idx:

1. Read Entry X = (MAC_TABLE_KEY[idx], MAC_TABLE_PAYLOAD[idx]); initialize X' = X.
2. For each Sweeper rule R = 0..15:

   If R.matches(X) and R.matches(Timer):

   - X' = R.transformEntry(X')
   - Set ReportMask[R] = R.ReportMatch

   Else:

   - Set ReportMask[R] = 0
3. Write X' back to (MAC_TABLE_KEY[idx], MAC_TABLE_PAYLOAD[idx]).
4. If ReportMask != 0:
   - Wait for space in SWEEPER_FIFO
   - Post (idx, ReportMask) to SWEEPER_FIFO

5. Depending on the sweeper timer configuration, do one of the following:
   - Proceed to the next table index idx next;
   - Halt, optionally raising an interrupt.

The SWEEPER register definitions provide more details regarding their operation and actions.

5.11.10 Table Access Arbitration

- The MAC table might be accessed by multiple independent sources in the FM5000/FM6000. Specifically, the following accesses can arise at any time and must be arbitrated by the table:
- Direct software R/W access
- Frame header MA1 and MA2 lookups
- Hardware entry writeback, triggered by frame tail arrivals
- Sweeper read-modify-write access

The direct software and the sweeper accesses share the management interface to the MAC table. Together they share any excess bandwidth that the MAC table has available after lookups and write backs are serviced. Accesses between software and the sweepers is fairly arbitrated.

Among head lookups and tail write-back operations, the MA1 lookup is given higher-priority access over write-back access. Regardless of the lookup load, the write-back mechanism is guaranteed to receive at least $1/128^{th}$ of the table bandwidth, and shares any excess bandwidth fairly with management and the sweepers. In split table mode, the MA2 lookup is unconditional and shares the MA1 lookup's prioritization. In full table mode, the MA2 lookup is performed in a best-effort fashion and is given lowest priority access.

Since write-back operations have a lower priority than the MA1 (and possibly MA2) lookups and tail write backs are not guaranteed to complete. However, at least one write-back request originating from each ingress port is buffered indefinitely until the lookup event rate is low enough to service the pending writeback operations. Under anomalous conditions, this could result in large delays from the time frames are forwarded by the switch to the time the MAC table is updated with the same frames' newly-written MA2 MAC addresses.

All hardware-initiated entry write backs are written atomically with respect to software and sweeper writes. In some cases, pending write-back operations can be invalidated to avoid overwriting a software or sweeper write that conflicts with the pending writeback index.

The sweepers issue read-modify-write accesses to the table and as a result expose a window of time (between the read and the write) when a conflicting software write might be subsequently overwritten. To avoid this hazard, a write indirection mechanism is provided that allows software to safely sequence its write operations with respect to the sweeper state machines. Each master in the management domain has a dedicated pair of L2L_SWEEPER_WRITE_COMMAND and L2L_SWEEPER_WRITE_DATA registers that it can use to specify an index and the data value to write to the MAC table. The sweepers issue these write operations to the MAC table on behalf of the master between its own read-modify-write accesses.

5.12 ALU

The ALU stage performs various arithmetic and logic operations on inputs produced earlier in the pipeline. It replaces a handful of fixed-function evaluations in the FM4000 series pipeline, such as the MTU check, and provides the flexibility to implement a number of optional features.

The ALU stage is viewed as a special key evaluation stage for L2AR. The ALU operations provide a richer set of key matching functions than the TCAM's simple masked-compare-to-constant test. However, the ALU functionality has a much larger implementation cost than a TCAM rule, so its matching capability is restricted to a small and aggressively muxed set of keys.

5.12.1 Overview

The ALU stage includes six functional units, referred to as ALU1..6. Each functional unit either adds, subtracts, or XORs two unsigned 16-bit input operands (X and Y) and generates overflow and equal-to-zero flags based on the value of its Z output. A number of transformations can be applied to the input operands:

- Bit-masking, to limit inputs to a specific bit width or to combine disjointed bit slices from the two operands.
- Mapping through a 16-entry by 16-bit table (Y operand only), providing operand compression when table entry bit resources are limited. In this case, only the bottom four bits of the Y operand are used.
- Swapping of X and Y operands.
- Barrel roll, to align fields from user-defined table entry formats (only supported on ALUs 1 and 4).

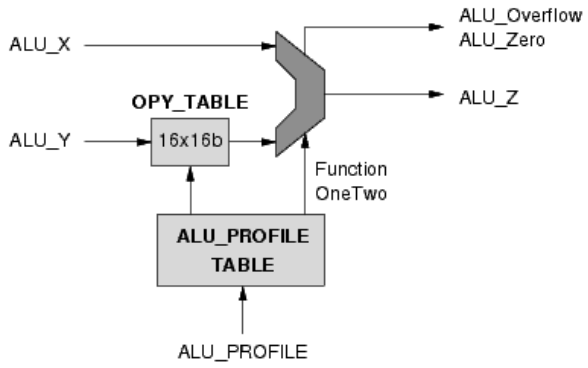


Figure 5-27 ALU Functional Unit

The functional units produce a 16-bit unsigned data output, passed on as potential inputs to the policers and L2AR units. Two result status bits (indicating equal-to-zero and overflow) are added to ACTION_FLAGS. These bits are available as L2AR keys and can be used to indicate equality or comparison relationships between the input operands.

The X input operand comes directly from L3AR mux outputs. The source of the Y input operand is also fully specified by L3AR, but in some cases its value is taken from the outputs of the L2 lookup stage. See Section 5.10.7 for details.

All ALU function units support two's complement arithmetic. For checksum calculation purposes, ALUs 2 and 5 additionally support ones-complement arithmetic, as indicated by the OneTwo control signal shown in Figure 5-27.

Table 5-54 lists the different capabilities of the six ALU functional units.

Table 5-54 ALU Capabilities

Function	ALU Number					
	1	2	3	4	5	6
Barrel roll on X and/or Y	✓	–	–	✓	–	–
Bit-masking of X and/or Y	✓	✓	✓	✓	✓	✓
Mapping of Y[3:0]	✓	✓	✓	✓	✓	✓
Swapping of X and Y	✓	✓	✓	✓	✓	✓
X + Y (two's complement)	✓	✓	✓	✓	✓	✓
X + Y (one's complement)	–	✓	–	–	✓	–
X ^ Y (XOR)	✓	✓	✓	✓	✓	✓

5.12.2 Inputs

The ALU stage receives two sets of inputs, one for ALUs 1..3, the other for ALUs 4..6. Each set consists of a command profile index (ALU13_CMD_PROFILE and ALU46_CMD_PROFILE) specified by L3AR as well as the input operands for the functional units.

Table 5-55 ALU Inputs

Channel	Width	Description
ALU13_CMD_PROFILE	5	Command profile numbers specified by the L3AR SetAluCmdProfile action.
ALU46_CMD_PROFILE	5	
ALU{1,2,3}_X	3 x 16-bits	Operand input #1 for each ALU functional unit. These all come directly from L3AR without modification.
ALU{4,5,6}_X	3 x 16-bits	
ALU{1,2,3}_Y	3 x 16-bits	Operand input #2 for each ALU functional unit. For some cases specified by the L3AR Output Mux action (see Section 5.10.7), this channel is set from outputs of L2 Lookup operations.
ALU{4,5,6}_Y	3 x 16-bits	

For each ALU functional unit, either the ALU13_CMD_PROFILE or ALU46_CMD_PROFILE input indexes a 32-entry ALU n_CMD_TABLE[0..31]. The 44-bit command returned by this table fully specifies the operation of each functional unit.

5.12.3 Command Encoding

Each ALU receives a 45-bit ALU_CMD_PROFILE-mapped command input, encoded as in Table 5-56:

Table 5-56 ALU Command Encoding

Field	Bits	Description
SwapXY	0	Specifies whether to swap input operands X=ALU_Y, Y=ALU_X. Following, all operands refer to their post-swapped values. This provides support for ALU_X-ALU_Y as well as ALU_Y-ALU_X, among other functional variations.
RollX	4:1	Specifies barrel rolling of X operand: X = (X << RollX) | X[15:(16-RollX)] Ignored for all ALUs other than 1 and 4.
MaskX	20:5	Specifies the post-roll masking of X: X = X & MaskX
RollY	24:21	Specifies barrel rolling of Y operand: Y = (Y << RollY) | Y[15:(16-RollY)]
MaskY	40:25	Specifies the post-roll masking of Y: Y = Y & MaskY
MapY	41	Specifies whether to map Y through Y_TABLE: Y = MapY | Y_TABLE[Y[3:0]] : Y
Function	43:42	Specifies the function to execute: 00b = ADD (X + Y) 01b = SUB (X - Y) 10b = XOR (X ^ Y) 11b = NOP (To save power, no output is produced)
OneTwo	44	Specifies ones or twos complement arithmetic. 0b = Twos compliment 1b = Ones compliment Only ALUs 2 and 5 support a OneTwo value of 1.
OrIntoL2ARKey	45	Specifies whether the ALU's Z output must be ORed into the corresponding ALU13_Z or ALU46_Z output for use in the L2 Action Resolution TCAM key.

Note: The sequence of operand transformations is swap, then roll, then mask, then map.

The following abbreviated notations are used to represent common ALU configuration profiles:

X + Y:	Function=ADD with OneTwo=0b.
X - Y:	Function=SUB with OneTwo=0b.
X (+1) Y:	Function=ADD with OneTwo=1b.
X (-1) Y:	Function=SUB with OneTwo=1b.
X <<(5,12) Y:	RollX=5, MaskX=0xFE0, RollY=0, MaskY=0x1F, Function=ADD.
X>Y:	SwapX=0, Function=SUB, OneTwo=1b. Overflow bit is result.
X==Y:	Function=SUB, Zero bit is result.

5.12.4 Outputs

Each ALU produces a 16-bit ALUn_Z result. The results are available downstream in the pipeline for the following uses:

- Mapping to policer indices (see SetPolicerIdxSrc action).
- Selection for L2F table indexing (ALU{1,2,3,4}_Z outputs). See Section 5.15.3.
- L2AR ALU13_Z and ALU46_Z TCAM keys. One ALUn_Z output from each 3-ALU set (1..3 and 4..6) can be selected for use as an L2AR TCAM key field.
- L2AR Output Channel Muxing. All ALU{1..6}_Z outputs are available as mux sources for a variety of MOD_DATA and STATS output channels. See Section 5.17.9

Table 5-57 shows a summary of 16-bit output channels.

Table 5-57 ALU Z Outputs

Output	Width	Description
	6 x 16-bits	16-bit result from each ALU functional unit.
ALU13_Z	16	OR over a selected set of ALU{1,2,3}_Z (ALU13_Z) and ALU{4,5,6}_Z (ALU46_Z) outputs. One bit in each ALU_CMD_TABLE selects whether each ALU's output is ORed into the corresponding ALU13_Z or ALU46_Z output. These channels are available as L2AR key fields.
ALU46_Z	16	

The ALU stage also produces two action flag bits per functional unit, as shown in Table 5-58.

Table 5-58 ALU Flag Outputs

Flag	Bits	Definition
ALU1_Overflow	54	Overflow output. For example, with Function=SUB, indicates X<Y when OneTwo=0b or X>Y with OneTwo=1b.
ALU2_Overflow	56	
ALU3_Overflow	58	
ALU4_Overflow	60	
ALU5_Overflow	62	
ALU6_Overflow	64	
ALU1_Zero	55	Equal-to-zero output. Evaluation of ALU_Z[i]=0b. When OneTwo=1b, this also evaluates to 1b if ALU_Z[i]=0xFFFF or ALU_Z[i]=0b since both are representations of zero in ones-complement arithmetic.
ALU2_Zero	57	
ALU3_Zero	59	
ALU4_Zero	61	
ALU5_Zero	63	
ALU6_Zero	65	

5.13 Policers

The policer stage of the FM5000/FM6000 FPP implements per-flow frame counting, tri-color marking, and drop-based rate-limiting. The early stages of the pipeline (FFU through L3AR) classify incoming frames and assign them to up to one policer index per policer bank. The policer stage looks up the count or rate profile state associated with those indices and passes the results on to L2AR.

Rules configured in L2AR determine whether the frame is to be dropped, recolored (a QoS transformation), or left as is. L2AR also determines whether the policer state associated with each frame assigned to it is to be credited with the bytes (or count) associated with the frame.

5.13.1 Overview

The policer stage contains two 4096 double-entry banks and one 1024 single-entry bank. The 4-Kb banks each provide two rate limiters or counters per frame, while the 1-Kb banks provide a single rate limiter or counter per frame. The 4-Kb banks are intended for flow-based tri-color policing and ACL counting. The 1-Kb bank is provided as a mechanism to limit the event rates of particular frame forwarding behaviors such as flooding or trapping to the CPU, possibly on a per-ingress-port basis.

Each 32-bit entry unit can be configured to operate in one of four modes:

- Data rate policing
- Frame rate policing
- Byte counter
- Frame counter

The policing functions are implemented as token buckets configured by a steady-state rate limit and a burst capacity. The token bucket count is replenished periodically and decremented when frames are credited to the bucket. When the bucket count goes to zero (or is negative), the token bucket is out of profile, and this status is indicated to L2AR so the frame can be dropped or down-marked appropriately.

Both entries in the 4-Kb banks can be configured as token buckets enforcing different rate limits to implement differentiated services tri-color marking (RFC 4115). In the terminology of tri-color marking, the two rate limits typically represent committed and excess rate profiles. In this mode of operation, the down-marking of the frame's DSCP status (and/or other QoS fields) is decided by L2AR, operating on:

1. The rate status flags reported by the policer evaluation.
2. The down-marked QoS fields and flags generated by the mark-down tables described in Section 5.13.6.

In counter mode, the entries operate as 32-bit rollover counters that increment each time frames are credited to the entry by L2AR. If both entries of a 4-Kb bank are configured to have the same counter type, the 32-bit state is combined to provide a single 64-bit frame or byte counter.

5.13.2 Evaluation, Reporting, and Crediting

For a given frame, any address in each of the two 4-Kb banks and 1-Kb bank can be accessed, evaluated, and updated. The entries are indexed by the L3AR's POL1_IDX, POL2_IDX, and POL3_IDX output channels. Thus, a total of five rate limiters or counters can be applied to any single frame, selected by three independent indices. Among these five entries, the out-of-profile rate status is reported to L2AR on the following action flag bits:

- **POL1_OverRate{1,2}** — Token bucket status of the two entries accessed in the first 4-Kb bank.
- **POL2_OverRate{1,2}** — Token bucket status of the two entries accessed in the second 4-Kb bank.
- **POL3_OverRate** — Token bucket status of the entry accessed in the 1-Kb bank.

If any of these token buckets have zero or negative token counts, its OverRate flag is set to 1b. If any of the entries are configured as counters, their OverRate flags are always reported as zero.

L2AR is responsible for setting a credit tail flag per token bucket or counter entry. By setting this flag, the bytes or count associated with the frame are credited to the token bucket or counter state. For a token bucket, this means the token count state is decremented; for a counter, the count state is incremented. The amount by which the state changes depends on the configured frame-rate/count versus byte-rate/count mode of the entry. A frame is not eligible for crediting to a token bucket or counter entry if is dropped due to congestion management. In such an event, all credit tail flags are overridden to zero.

The 32-bit signed token bucket state values saturate if/when they reach their maximum negative value (-2^{31}). Counter state values are unsigned and roll over to zero upon exceeding $2^{32}-1$.

The policer stage is located quite late in the pipeline, after L2 lookup. This provides L3AR with a wide selection of sources for its counter/policer index selection. This makes it possible to count or police any of the following:

- FFU rules (maintaining full FM4000 series compatibility)
- DMAC or SMAC lookups
- Ingress or egress VLANs

For more specific information about policer index selection, see Section 5.10, “L3 Action Resolution”.

5.13.3 Entry Formats

Each policer entry consists of 32 bits of token bucket or counter state, addressed by the POLICER_STATE registers, as well as another 32 bits of static configuration, addressed by the POLICER_CFG registers. The 32-bit configuration state is encoded as shown in Table 5-59

Table 5-59 Policer Configuration Format

Field	Width	Description
Mode	1	Specifies either counter mode or token-bucket policer mode. 0b = Counter mode 1b = Policer mode

Table 5-59 Policer Configuration Format (Continued)

Field	Width	Description
Unit	1	Specifies either frames or bytes as the unit to be counted or rate-limited. 0b = Frames 1b = Bytes
RateMantissa	4	The token bucket steady-state rate limit, encoded in floating point form. Only relevant when mode is set 1b. The rate unit depends on the configured unit value, and on the policer sweeper period. Rate = RateMantissa << RateExponent [bytes or frames per sweeper_period]
RateExponent	5	
CapacityMantissa	4	The token bucket burst capacity, encoded in floating point form. Only relevant when mode is 1b. Capacity = CapacityMantissa << CapacityExponent [bytes or frames] The absolute maximum supported capacity value is the maximum signed token bucket state value, or 2^{31}-1.
CapacityExponent	5	
TAG	12	Generic data tag available for encoding actions to perform by L2AR in conjunction with the entry's rate status. The top four bits of the TAG are available as key fields in the L2AR slice TCAMs. These are expected to encode the action type. The bottom 10 bits are available for QoS field assignment in L2AR. All 12 TAG bits are also available for STATS and MOD_DATA channel mux sources so the field can be used as a general 12b-to-12b mapping pathway.

In token bucket mode, the 32 bits of state associated with each entry encodes a signed token count in units of either bytes or frames, depending on the Unit configuration. Normally software does not have any reason to write to this state, although write access is supported.

In counter mode, the 32 bits of state encodes an unsigned count that rolls over to zero on overflow. When the two entries of a 4-Kb bank address are both encoded to have Mode=1b (count) and the same unit value, the second entry's 32-bit state field (count2) increments each time the Count1 counter overflows, thus providing a single 64-bit counter.

5.13.4 Token Bucket Dynamics

The operation of an ideal token bucket is straightforward. The bucket's token count is incremented continuously at the configured rate, saturating when it reaches the bucket's configured capacity.

When a rate-limited frame ingresses the switch, the token bucket state is evaluated. If the token count is equal to or larger than the frame's length, the token bucket is said to be in profile; otherwise, the bucket is reported to be out of profile (OverRate flag is set). In either case, the length of the frame (or 1 in frame-rate mode) is subtracted from the token count.

The dynamic behavior of the FM5000/FM6000's token buckets deviates from this ideal definition due to a number of architectural properties:

- Token bucket state evaluations are performed immediately when frames ingress (at head-of-frame), before the length of the frame is known.
- In data-rate mode, bucket token count decrement is performed in discrete amounts at the tail of each ingress segment (such as every 160 bytes or, on the last segment, less). In frame-rate mode, the bucket token count is decremented on the tail of the frame's first segment.
- Token bucket counts are incremented in discrete lump sums every sweeper period. The average sweeper period is guaranteed to match the configured value (set by PolicerPeriod in the LSM_SWEEPER_CFG register), but some variability might arise from one period to the next.

The cumulative effect of these non-ideal properties is to introduce potential inaccuracies in the adherence to the token bucket's capacity parameter. Selecting a sufficiently large capacity avoids most of these errors. Since many of these errors only arise in corner-case traffic conditions, it might be acceptable to configure a lower capacity and accept occasional burst rate violations.

The following system parameters have a quantitative impact on the capacity minimum bounds and errors:

- **$f_{sweeper}$** — Sweeper frequency in GHz (inverse of sweeper_period). A configured constant, nominally 3e-6 (3 KHz).
- **N_{ports}** — Number of port tokens in the segment scheduler (typically 64 to 74).
- **N_{src}** — Maximum number of ingress ports that can receive frames evaluated by the token bucket. In some applications this can be exactly one; in others, it can be as large as all ports in the switch.
- **$N_{policer}$** — Maximum number of policer entries that the sweeper must process over all three policer banks. This is configured in the FC_MRL_UNROLL_ITER register discussed below. Nominally 4095.
- **R** — Configured rate, in bytes or frames per sweeper period. With a nominal sweeper period of 333 µs, a 10 Gb/s rate would have R = 407 KB [per sweeper_period].
- **ΔT** — Maximum token refresh jitter (±) that can arise. For the FM5000/FM6000, this is about 0.45, worst-case.

$$\Delta T = 0.5 \ N_{ports} \ N_{policer} \ f_{sweeper}$$

Table 5-60 lists the minimum capacity bounds and errors that arise from the FM5000/FM6000's token bucket dynamics.

Table 5-60 Policer Capacity Bounds

Parameter	Worst-case Value	Description	Typical Case [1]
$\Delta C_{overshoot}$	N_{src} * MTU	Maximum capacity overshoot beyond the configured value. This is a fundamental unavoidable error term. It is asymmetric in the sense that the observed burst capacity always falls in the range $[C, C+\Delta C_{overshoot}]$. ***Note:*** In frame-rate mode, MTU becomes one.	1522 Bytes
$C_{min,static}$	R	Minimum capacity needed to avoid systematic rate errors throughout operation.	41 KB
$C_{min,dynamic}$	$R\ (1+\Delta T)/(1-\Delta T)$	Minimum capacity needed to avoid errors due to traffic pattern-dependent effects.	108 KB
$\Delta C_{dynamic}$	$R * \Delta T$	Maximum error (±) in observed token bucket capacity if C is configured in the regime $C_{min,static} < C < C_{min,dynamic}$. This *only* occurs when the ingress rate R_i falls in the following range: $R < R_i < R/(1-\Delta T) \approx 1.82 * R$	18 KB

1. The example provided has the following properties:
   125 MB/s rate limit (1 Gb/s)
   N ports=64
   N src=1

Note: The C min bounds scale linearly with the desired rate limit, R.

Operating in the $C_{min,static} < C < C_{min,dynamic}$ regime can be entirely reasonable in many applications. The effect is occasional violations of the specified burst capacity, usually much less than the worst-case $R * \Delta T$. In addition to the ingress rate falling within the specified range, the errors are furthermore dependent on the switch experiencing wide fluctuations in the chip-wide aggregate frame rate.

Table 5-61 lists the dynamic range of rates that the token buckets support.

Table 5-61 Token Bucket Dynamic Range Rates

Parameter	Formula	Description	Typical Case
R_{min}	$f_{sweeper}$	Minimum rate. Set by configuring the smallest rate possible, namely one byte per sweeper period.	3 KB/s
R_{max}	$2^{31} (1-\Delta T)/(1+\Delta T) f_{sweeper}$	Maximum rate. Limited by the dynamic range of the token bucket credit counter. To avoid clipping caused by traffic-dependent capacity overruns, the maximum capacity must be set smaller by a factor of $(1- \Delta T)/(1+\Delta T)$. Since $R \leq C$, this constraint also limits the maximum rate that is reliably sustained. With nominal settings, R_{max} greatly exceeds the maximum aggregate data rate of the switch.	2,400 GB/s

5.13.5 Sweeper Configuration

The policer sweeper is a background hardware process responsible for periodically replenishing each policer's token bucket count. The sweeper processes one address in all three banks per access cycle. It therefore normally requires 4095 access cycles to complete one sweep through all entries, referred to as sweeper_period or $f_{sweeper}$ (its inverse, measured in GHz).

Two configuration parameters influence the behavior of the policer sweeper and affect the dynamics of the policers:

- **FC_MRL_UNROLL_ITER.Cycles** — Identifies the maximum index over all three policer banks that contains an entry with mode set to 1b (token bucket). This nominally is 4095, the highest index number. However, the C_{min} bounds previously detailed can be reduced by using fewer than the maximum number of policers, packing those policer entries at the low end of the policer address range, and then specifying a minimal MaxPolicerIndex value for this parameter.
- **LSM_SWEEPER_CFG.PolicerPeriod** — Regulates sweeper_period. As the MaxPolicerIndex is reduced, PolicerPeriod can be commensurately reduced by a factor of MaxPolicerIndex/4095. PolicerPeriod must not be set too low or else the upper policer entries might not be updated reliably.

5.13.6 QoS Mark-Down Mapping

The policers and discrete rate limiters determine whether a frame belongs to a flow that is out of profile. The policer stage also includes two QoS mapping tables, POLICERS_QOS_MAP1 and POLICERS_QOS_MAP2, that determine how such frames' QoS values are transformed for tri-color marking purposes. The tables produce two fields: a first-order (First) and a second-order (Second) marked-down value. Both mark-down values are provided to L2AR so it can select the appropriate final QoS value based on the results of the policer evaluations.

Each QOS_MAP table is indexed by a QoS input value selected by L3AR (QOS.MAP1_IDX or QOS.MAP2_IDX) and contains entries explicitly configured with the two First and Second mark-down fields. It is up to software to ensure that the table is configured to satisfy the following properties:

```
MAP[MAP[IDX].First] = MAP[IDX].Second
IDX ∈ Red or IDX ∈ Yellow ⇒ MAP[IDX].First ∈ Red and MAP[IDX].Second ∈ Red
IDX ∈ Green ⇒ MAP[IDX].First ∈ Yellow and MAP[IDX].Second ∈ Red
IDX ∈ Green or IDX ∈ Yellow ⇒ IDX ≠ MAP[IDX].First
IDX ∈ Red ⇒ IDX == MAP[IDX].First
```

The green, yellow, and red terms represent equivalency sets of QoS fields associated with the corresponding markings. Given the previously described properties, L2AR can infer the color of the initial QoS state IDX (and therefore of the First and Second mark-down cases) based on the following evaluations:

```
IDX == MAP[IDX].First ⇒ IDX ∈ Red
IDX ≠ MAP[IDX].First and MAP[IDX].First == MAP[IDX].Second ⇒ IDX ∈ Yellow
IDX ≠ MAP[IDX].First and MAP[IDX].First  ≠ MAP[IDX].Second ⇒ IDX ∈ Green
```

The previous two independent terms, QOS_Map_Eq01 (IDX == MAP[IDX].First) and QOS_Map_Eq12 (MAP[IDX].First = MAP[IDX].Second), are calculated and passed on to L2AR as action flag bits.

Figure 5-28 shows the hardware structure of the POLICER_QOS_MAP1 table.

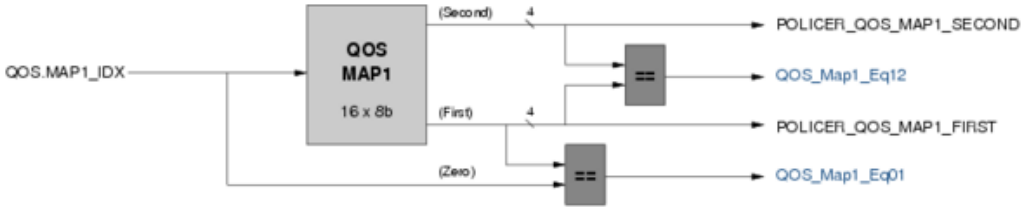


Figure 5-28 Policer QOS Map Table

The second POLICER_QOS_MAP2 table is similar, except it contains 64 entries and its First and Second mark-down fields are each eight bits wide. POLICER_QOS_MAP1 is intended for marking down any of the four bit QoS fields (such as QOS.ISL_PRI or QOS.L2_VPRI1) while the POLICER_QOS_MAP2 supports marking down the wider QOS.L3_PRI field (like DSCP) as well as any of the 4-bit fields.

5.13.7 Outputs

Each policer bank communicates the status of its indexed rate profiles by setting five action flag bits. The output comparisons of the QoS mark-down tables set an additional four bits in the action flags vector. All of these status bits are available as L2AR TCAM keys.

Table 5-62 Policer Outputs

Flag	Bit Index	Definition
POL1_OverRate1	61	Indicates that the corresponding policer entry that was indexed and evaluated by the corresponding POL{1,2}_IDX channel is out-of-profile. Is always be zero for counter entries.
POL1_OverRate2	62	
POL1_OverRate1	63	
POL1_OverRate2	64	
POL1_OverRate	65	
QOS_Map1_Eq01	66	Encodes QOS.MAP1_IDX == POLICER_QOS_MAP1[QOS.MAP1_IDX].First
QOS_Map1_Eq012	67	Encodes POLICER_QOS_MAP1[QOS.MAP1_IDX].First == POLICER_QOS_MAP1[QOS.MAP1_IDX].Second
QOS_Map1_Eq01	68	Encodes QOS.MAP2_IDX == POLICER_QOS_MAP2[QOS.MAP2_IDX].First
QOS_Map1_Eq12	69	Encodes POLICER_QOS_MAP2[QOS.MAP2_IDX].First == POLICER_QOS_MAP2[QOS.MAP2_IDX].Second

The policers also generate 60 bits of TAG outputs available for L2AR TCAM matching and muxing. In addition, the QoS mark-down tables produce four new QoS values that, on the basis of the previous rate limiter status evaluations, L2AR might select as the new egress QoS level.

Table 5-63 Policer QoS Outputs

Field	Width	Definition
POL1_TAG1	12	Tag fields associated with each policer entry.
POL1_TAG2	12	
POL2_TAG1	12	
POL2_TAG2	12	
POL3_TAG	12	
POLICER_QOS_MAP1_FIRST	4	First-order mark-down mapping of QOS.MAP1_IDX.
POLICER_QOS_MAP1_SECOND	4	Second-order mark-down mapping of QOS.MAP1_IDX.
POLICER_QOS_MAP2_FIRST	8	First-order mark-down mapping of QOS.MAP2_IDX.
POLICER_QOS_MAP2_SECOND	8	Second-order mark-down mapping of QOS.MAP2_IDX.

5.14 GloRT Lookup

Conceptually, the FM5000/FM6000 is a switch-router that operates on a virtualized physical layer. According to this abstraction, ports can represent LAGs, remote or attached CPUs, or internal loopback entities such as the IPL. More significantly, these virtualized ports can map to physical ports spread across any number of FM5000/FM6000 instances. These FM5000/FM6000 instances form a multi-chip domain, unified by a fabric tag. These virtualized ports are referred to as Global Resource Tags, or GloRTs.

The GloRT Lookup stage in the pipeline provides the linkage between the GloRT-virtualized ports and the FM5000/FM6000 instance-specific physical ports. Given a destination GloRT from the L2 lookup stage, the GloRT Lookup stage is responsible for producing a destination port mask index (abbreviated DMASK_IDX). The index is then passed on to the L2 filtering stage for the final mapping to an 76-bit destination port mask.

5.14.1 Overview

The GloRT Lookup function consists of three parts:

1. Matching the DGLORT in a 1-Kb-entry TCAM/RAM structure, which provides an index into the destination port mask table, along with other associated information.
2. Mapping the DMASK_IDX in a 16-Kb-entry destination mask table to obtain the destination port mask. This mapping can, optionally, involve the frame's L2_HASH value, useful for load balancing, link aggregation, and other stochastic frame forwarding applications. This hash-based transformation of DMASK_IDX is referred to as LAG pruning.
3. Hash-based filtering of the destination mask, primarily intended for link aggregation. This function is referred to as LAG filtering.

These DMASK_IDX mapping steps are shown schematically in Figure 5-29.

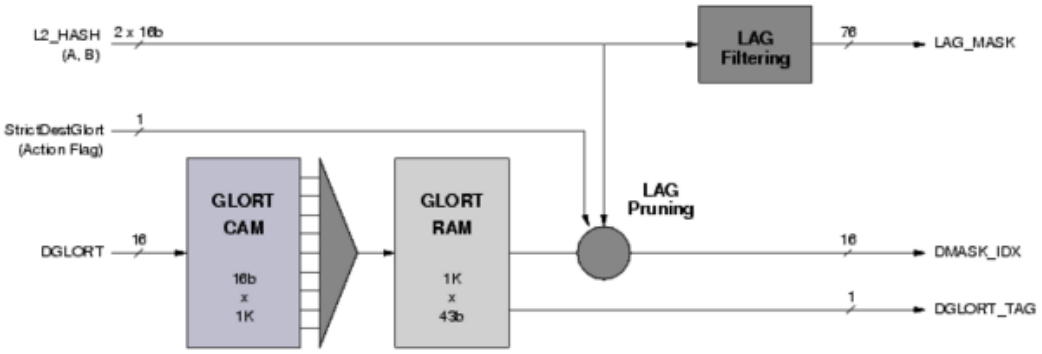


Figure 5-29 GloRT Lookup Block Diagram

The LAG filtering mask (LAG_MASK) is generated by the GloRT lookup stage since it is closely related to the LAG pruning calculation. The mask is passed on to L2 filtering where it can be applied to the final destination port mask. L2AR also has access to the LAG_MASK so it can re-apply LAG filtering in case it further transforms the destination port mask.

Two of four 16-bit L2 hash rotations calculated by the L2_HASH unit are available for use in the GloRT Lookup stage, as mapped by the HASH_ROTATION_CFG register. In the GloRT lookup stage these mapped rotations are referenced as rotations A and B, and are selected per GloRT entry (in the GLORT_RAM) or LAG group (in the GLORT_LAG_TABLE).

5.14.2 GloRT CAM and Table

The first-stage mapping of the DGLORT through the GLORT_CAM and GLORT_TABLE structures is a standard TCAM/RAM lookup operation. Each of the GLORT_CAM's 1024 entries stores a (Key, KeyInvert) pair. The highest-numbered entry i that satisfies the condition provides the index for the GLORT_TABLE lookup. If no entry hits in the CAM, the index is set to zero. Thus, regardless of how entry zero is configured, it behaves as if Key[0]=0xFFFF and KeyInvert[0]=0xFFFF.

```
(DGLORT & Key[i] == DGLORT) & (~DGLORT & KeyInvert[i] == ~DGLORT)
```

Each GLORT_TABLE entry stores the following fields:

Table 5-64 GloRT Table Entries

Field	Width	Description
HashCmd	2	Determines how the StrictDestGlort flag and L2_HASH value are used when mapping the destination GloRT to the forwarding destination mask. The GloRT is said to be *strictly* mapped if its the frame's hash value is ignored in this function. Four mapping cases are defined: 00b = Strictly mapped only if the StrictDestGlort Action Flags bit is set to 1; otherwise the L2_HASH value is used in both the LAG Pruning and LAG Filtering DMASK mapping steps (see Section 5.14.3 and Section 5.14.4). 01b = Same as case 0, except LAG Filtering is always strict (see Section 5.14.4). 10b = Not strict, overriding the StrictDestGlort flag. 11b = Strict regardless of StrictDestGlort.
DMaskBaseIdx	16	Base index into the L2F_GLORT_DMASK_TABLE, which maps to a forwarding destination port mask following the hashing transformations. ***Note:*** The L2F_GLORT_DMASK_TABLE is abbreviated as DMASK_TABLE.
RangeSubIndexA	8	Defines the position and size of a sub-index A taken from the DGLORT value. The first four bits (RangeSubIndexA[3:0]), OffsetA, give the starting bit position within DGLORT of the sub-index; the second four bits (RangeSubIndexA[7:4]), LengthA, give the width of the sub-index field.
RangeSubIndexB	8	Defines the position and size of a second sub-index B taken from the DGLORT value. Interpreted in the same manner as RangeSubIndexA.
DMaskRange	7	Specifies a LAG pruning range size of 1-Kb to 16-Kb.
HashRotation	1	Specifies one of two L2_HASH rotations for the LAG pruning operation (either rotation A or B).
DGlortTag	1	1-bit tag associated with the DGLORT for use as an L2AR key.

5.14.3 LAG Pruning

The mapping of DMaskBaseIdx to DMASK_IDX involves a complicated set of fixed-function rules motivated primarily by multi-chip link aggregation considerations. For the Strict=0b (hashed) case, the index is calculated according to the following equation:

```
DMASK_IDX = DMaskBaseIdx +
            (LagBin(L2_HASH[HashRotation],DMaskRange[2:0],DMaskRange[6:3]) <<
            LengthA) + SubIndexA
```

where SubIndexA = DGLORT[(LengthA+OffsetA-1):OffsetA].

The LagBin function distributes the indices over a range of entries specified by the L2_HASH value and the DMaskRange specification.

```
LagBin(x,Select[2:0],Shift[3:0]) = (x % (2 Select + 1)) << Shift | x[Shift-1:0]
```

With this binning function, DMASK_IDX is distributed uniformly over the following range of entries:

```
range_size = (2 * DMaskRange[2:0] + 1) << DMaskRange[6:3].
```

Note: This binning function is isomorphically equivalent to the modulo operator. For a fixed DMaskRange and x=0..2^{16}-1, LagBin(x, DMaskRange) might be 1-to-1 mapped to x%range_size.

The purpose of including LengthA and SubIndexA in the index calculation is to most efficiently use the GLORT_CAM and L2F_GLORT_DMASK_TABLE (DMASK_TABLE) resources. A GLORT_TABLE entry can be thought of as specifying a template for deriving an index location from a matching DGLORT. The template can be shared among multiple DGLORTs (each corresponding to a LAG) to better deal with the under-provisioning of the GLORT_CAM compared to the DMASK_TABLE.

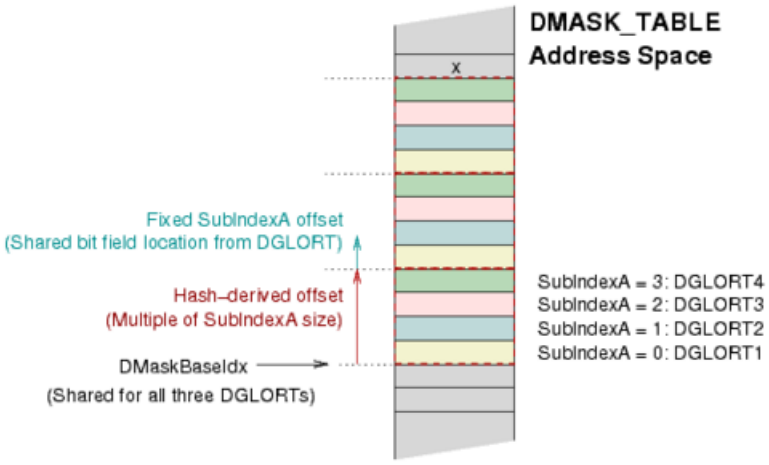


Figure 5-30 GloRT Lookup LAG Pruning Example

The specific form in which LengthA and SubIndexA factors into the index calculation enables LAGs with sizes that are not a power of two to efficiently pack into the DMASK_TABLE (see Figure 5-30). In this example, four DGLORTs, each representing 3-port LAGs, map to the same GLORT_TABLE entry. Thus, they all share the same DMaskBaseIdx, LengthA, and their SubIndexA are taken from the same bit slice from their DGLORT values. However, the next available DMASK_TABLE entry is only 3 x 4 = 12 entries above DMaskBaseIdx (marked x), whereas a more intuitive index calculation formula would put the next available location at 4 x 4 = 16 entries above DMaskBaseIdx.

For Strict=1b entries, the sub-index in the calculation previously presented that is dependent on the frame's hash value, [namely LagHash(L2_HASH, DMaskRange)], is taken directly from the DGLORT. This enables strict GloRTs to represent specific physical ports that might otherwise belong to a LAG or PSG. For these strict DGLORTs, the index calculation becomes:

```
DMASK_IDX = DMaskBaseIdx + (SubIndexB << LengthA) + SubIndexA
```

where SubIndexB = DGLORT[(LengthB+OffsetB-1):OffsetB].

This first-stage mapping of the DGLORT over a range of DMASKs based on a frame hash value is primarily intended for pruning a LAG GloRT's distribution over a multi-chip domain. A subsequent filtering of the DMASK returned by the GLORT_DMASK_TABLE in L2 filtering provides a second-stage resolution of each LAG DGLORT to a single destination physical port.

The use of these mechanisms to implement multi-chip link aggregation and other traffic load-balancing features are described further in Application Analysis sections.

5.14.4 LAG Filtering

The LAG filtering function has the responsibility of selecting at most one physical port of each LAG to which the frame should be forwarded. Generally, it is expected that the DMASK produced by the DMASK_TABLE has ones in all bit locations corresponding to external physical ports belonging to each LAG in the frame's forwarding distribution. The LAG filtering operation involves ANDing this DMASK with a LAG_MASK that leaves (at most) only a single one of these external ports one in the destination port mask. A basic example is shown in Figure 5-31.

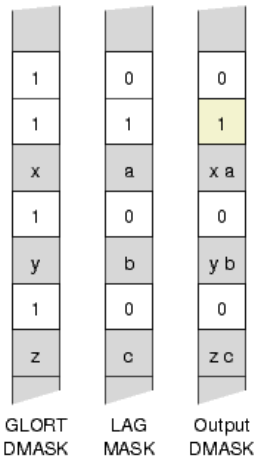
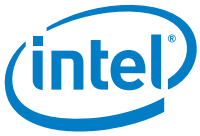


Figure 5-31 GloRT Lookup LAG Filtering Example



The white bits in the DMASK represent ports that are members of some LAG of interest. All bits are set to one in the DMASK returned by the DMASK_TABLE (labeled GLORT_DMASK), indicating that the frame is to be forwarded to this LAG. Based on the frame's hash value, the LAG_MASK is calculated to set only one bit of the LAG's port members to one, so the frame ends up being forwarded to only one destination port (indicated in yellow).

There are two different methods to define a LAG_MASK:

rev A: LAG_PORT_TABLE

rev B and above: LAG_FILTERING_CAM

In both cases, for LAG pruning and LAG filtering to work together in a coherent fashion, it is important that they both use the same hash values and binning functions. The resulting LAG_MASK is also passed on to L2AR for use with the TransformDestMask action (see Section 5.17.6).

5.14.4.1 Rev A: LAG_PORT_TABLE

For the rev A silicon, each bit of the LAG_MASK is calculated based on per-port configured properties and the frame's hash value. The LAG_PORT_TABLE register specifies the following per-port properties:

Table 5-65 GloRT Lookup Port LAG Properties

Field	Width	Description
LagSelec	3	Defines the total number of ports belonging to the LAG. The actual LAG size is equal to: LagSize = (2*LagSelect+1)<<LagShift
LagShift	3	
Index	7	Identifies the particular member index associated with the port. No two ports belonging to the same LAG should be assigned the same index number.
HashRotation	1	Selects one of two L2_HASH rotations to use in the filtering function. The hash key and rotation must be configured the same for all ports belonging to the same LAG. Value zero specifies rotation A, value one rotation B.

Given these properties, each bit p of the LAG_MASK is calculated as

```
LAG_MASK[p] = Strict | (LagBin(L2_HASH(HashRotation),LagSelect,LagShift) == Index)
```

where LagBin is defined as for LAG pruning.

5.14.4.2 Rev B+: LAG_FILTERING_CAM

For the rev B silicon, the filtering stage was updated to use a 76-entry CAM (one entry per PORT) where the 76-bit match result is the LAG_MASK.

```
LAG_MASK[p] = Strict | (LAG_FILTERING_CAM[p].Key & key == key &&
                        LAG_FILTERING_CAM[p].KeyInvert & ~key == ~key)
```

The 64-bit key presented to the CAM has the following structure:

Name	Width	Bits
hashA	8	7:0
modulo(hashA,3)	2	9:8
modulo(hashA,5)	3	12:10
modulo(hashA,7)	3	15:13
modulo(hashA,9)	4	19:16
modulo(hashA,11)	4	23:20
modulo(hashA,13)	4	27:24
modulo(hashA,15)	4	31:28
hashB	8	39:32
modulo(hashB,3)	2	41:40
modulo(hashB,5)	3	44:42
modulo(hashB,7)	3	47:45
modulo(hashB,9)	4	51:48
modulo(hashB,11)	4	55:52
modulo(hashB,13)	4	59:56
modulo(hashB,15)	4	63:60

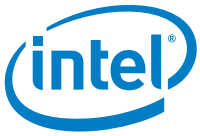
If a port is not part of any LAG, the actual content of hash A and B for that port must be set to all ones (ignored).

```
# Never filtered
LAG_MASK[p].key       = 0xFFFFFFFF_FFFFFFFF
LAG_MASK[p].keyInvert = 0xFFFFFFFF_FFFFFFFF
```

If the port is part of a LAG where packets must be distributed according to hashA, then hashB part of the key must be set to all ones (thus ignored) while the hashA part must be set such that each port in that LAG watches for a particular pattern of the value/modulo combination. Vice versa if hashB is to be used and hashA is to be ignored.

For power of two group size, distribute using the hash value directly.

```
# Ports 1..4 are in a 4-port LAG using hash rotation A
LAG_FILTERING_CAM[1].Key       = 0xFFFFFFFF_FFFFFFFC
LAG_FILTERING_CAM[2].Key       = 0xFFFFFFFF_FFFFFFFD
LAG_FILTERING_CAM[3].Key       = 0xFFFFFFFF_FFFFFFFE
LAG_FILTERING_CAM[4].Key       = 0xFFFFFFFF_FFFFFFFF
LAG_FILTERING_CAM[1].KeyInvert = 0xFFFFFFFF_FFFFFFFF
LAG_FILTERING_CAM[2].KeyInvert = 0xFFFFFFFF_FFFFFFFE
LAG_FILTERING_CAM[3].KeyInvert = 0xFFFFFFFF_FFFFFFFD
LAG_FILTERING_CAM[4].KeyInvert = 0xFFFFFFFF_FFFFFFFC
```



For an odd size group, distribute using the pre-computed modulo for the desired rotation.

```
# Ports 11..19 are in a 9-port LAG using hash rotation A
LAG_FILTERING_CAM[11].Key       = 0xFFFFFFFF_FFF0FFFF
LAG_FILTERING_CAM[12].Key       = 0xFFFFFFFF_FFF1FFFF
LAG_FILTERING_CAM[13].Key       = 0xFFFFFFFF_FFF2FFFF
....
LAG_FILTERING_CAM[19].Key       = 0xFFFFFFFF_FFF8FFFF
LAG_FILTERING_CAM[11].KeyInvert = 0xFFFFFFFF_FFFFFFFF
LAG_FILTERING_CAM[12].KeyInvert = 0xFFFFFFFF_FFFEFFFF
LAG_FILTERING_CAM[13].KeyInvert = 0xFFFFFFFF_FFFDFFFF
....
LAG_FILTERING_CAM[19].KeyInvert = 0xFFFFFFFF_FFF7FFFF
```

For even size group not a power of two, use a combination of hash value and pre-computed modulo to create the distribution. As an example, if ports 10..15 are in a 6-port LAG (2 x 3) on hash rotation A, use 1 bit of HashAValue and use the 2 bits of HashAModule3 to create the 6-way uniform distribution:

```
LAG_FILTERING_CAM[10].Key       = 0xFFFFFFFF_FFFFFCFE
LAG_FILTERING_CAM[11].Key       = 0xFFFFFFFF_FFFFFCFF
LAG_FILTERING_CAM[12].Key       = 0xFFFFFFFF_FFFFFDFE
LAG_FILTERING_CAM[13].Key       = 0xFFFFFFFF_FFFFFDFF
LAG_FILTERING_CAM[14].Key       = 0xFFFFFFFF_FFFFFEFE
LAG_FILTERING_CAM[15].Key       = 0xFFFFFFFF_FFFFFEFF
LAG_FILTERING_CAM[10].KeyInvert = 0xFFFFFFFF_FFFFFEFF
LAG_FILTERING_CAM[11].KeyInvert = 0xFFFFFFFF_FFFFFEFE
LAG_FILTERING_CAM[12].KeyInvert = 0xFFFFFFFF_FFFFFDFF
LAG_FILTERING_CAM[13].KeyInvert = 0xFFFFFFFF_FFFFFDFE
LAG_FILTERING_CAM[14].KeyInvert = 0xFFFFFFFF_FFFFFCFF
LAG_FILTERING_CAM[15].KeyInvert = 0xFFFFFFFF_FFFFFCFE
```

5.14.5 Outputs

The GloRT lookup stage produces the following channel outputs:

Table 5-66 GloRT Lookup Outputs

Field	Width	Description
DMASK_IDX	16	Specifies a 76-bit DMASK for L2 filtering. Propagated to the forward channel where it identifies the L3 multicast replication index in the frame scheduler's mid stage.
LAG_MASK	76	LAG filtering mask. Passed to L2AR for use with its TransformDestMask action. Needed for mirroring to LAGs.
DGLORT_TAG	1	Tag associated with the DGLORT's DGLORT_TABLE entry. Might indicate that the DMASK as returned by the L2F_GLORT_DMASK_TABLE was zero. For example, that the frame was dropped due to an invalid DGLORT.)

5.15 Destination Mask Generation

5.15.1 Overview

One of the primary purposes of the FPP between the GloRT lookup and scheduler stages is to determine the specific set of physical destination ports to which each frame is forwarded. The forwarding distribution is represented by a 76-bit destination mask (DMASK). If bit i of this mask is set to 1b, at least one copy of the ingress frame is forwarded to port number i. A number of mapping, filtering, and other transformation steps are involved in the determination of each frame's final DMASK. The specific nature of many of these steps are application-dependent. An example DMASK generation sequence might be the following:

1. Map initial DMASK from the GLORT_DMASK_IDX returned by the GloRT lookup.
2. Filter the DMASK by the VLAN membership mask of the frame's egress VID.
3. Filter the DMASK by the frame's egress spanning tree.
4. Filter the DMASK such that the frame is forwarded to only one port of each destination LAG.
5. Zero the DMASK if the frame is determined to have violated any security policies.
6. Override the DMASK to include only the CPU port if the frame should be trapped.
7. OR additional ports into the DMASK for mirroring or logging purposes.
8. Filter the DMASK for congestion management purposes.

Each stage of the DMASK generation section of the pipeline is, to varying degrees, microcode-configurable in how it transforms the DMASK. Each stage includes a custom-function sub-stage that determines a DMASK transformation command which is then applied by a shared DMASK transformer sub-stage. The structure of the DMASK generation pipeline is shown in Figure 5-32.

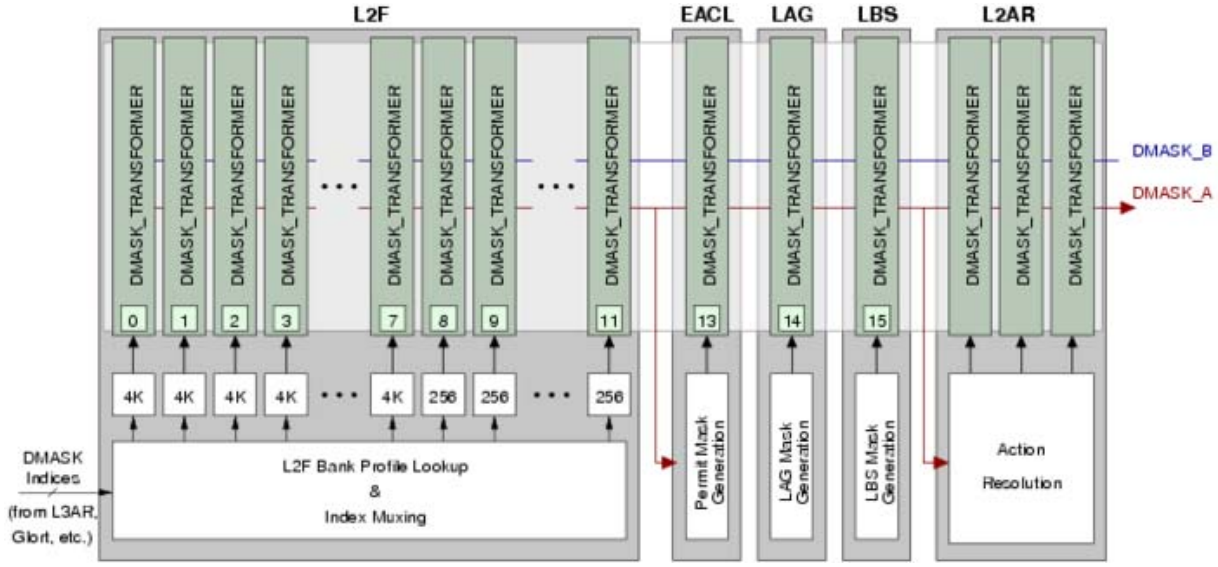
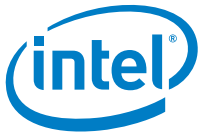


Figure 5-32 DMASK Generation Pipeline



As shown in Figure 5-32, the DMASK generation occurs in five sections:

1. **Layer 2 Filtering (L2F)** — Responsible for performing all DMASK-related mapping transformations. At a minimum, an initial DMASK is mapped by one or more L2F tables from a DMASK index determined by the L3AR or GloRT lookup stages. Typically, other filtering masks are applied to the DMASK based on the frame's VLAN membership, source port, spanning tree and other indexed associations. The L2F section includes eight 4-Kb-entry mapping tables and four 256-entry tables.
2. **Egress ACLs (EACL)** — Responsible for determining and applying the EACL actions identified by the final FFU TCAM stage based on the state of the DMASK at this point in the pipeline. In particular, the permit/deny action produces a permit mask which further filters the frame's forwarding.
3. **LAG** — Responsible for selecting exactly one port of each link aggregation group to which the frame is forwarded, based on the L2 hash value.
4. **Loopback Suppress Filtering (LBS)** — Responsible for preventing packets from being looped back to the port they come from.
5. **L2AR** — Responsible for applying all final DMASK transformations that are a function of the frame's header and the microcode-programmed forwarding policies. Each L2AR slice maps a DMASK transformation command based on a typical CAM/RAM architecture shared with other Action Resolution stages of the frame processing pipeline.

The common DMASK transformer function of each stage operates on up to two DMASK channels, referred to as DMASK_A and DMASK_B. These channels run the entire length of the DMASK generation pipeline. DMASK_A is the primary DMASK; its final value at the output of L2AR is propagated on to the congestion management and scheduler stages, whereas DMASK_B terminates. DMASK_B is intended as a scratch register for propagating transitory DMASK operands from stage to stage. For example, the LAG_MASK produced by the LAG stage can be mapped to DMASK_B so that LAG filtering can later be applied to any additional LAGs added by L2AR to DMASK_A.

5.15.2 DMASK Transformer

The DMASK transformer function is shared among all DMASK generation stages. Its purpose is to transform two 76-bit input DMASK channels, A and B, according to a 7-bit input command and an input M DMASK operand. It supports a wide variety of 3-input, 2-output logical operations (see Table 5-67).

If the input DMASK_A is non-zero and the application of the command results in an output DMASK_A' that is zero, an 8-bit DROP_CODE that ripples from stage-to-stage is set to a value associated with the current stage. The final DROP_CODE value identifies which transformation, if any, caused the frame to be dropped. It is presented as a TCAM key to L2 and statistics action resolution stages to enable these frames to be appropriately counted for system diagnostics purposes.

Figure 5-33 shows the structure of each DMASK transformer.

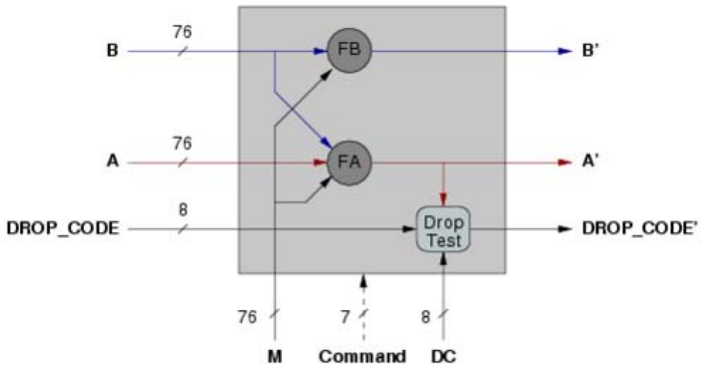


Figure 5-33 DMASK Transformer

- Either A or B outputs might depend on the M input.
- Output A takes B as an input, but not vice versa.

The input command is split between a 4-bit CmdA for the FA function and a 3-bit CmdB for the FB function. The transformation commands supported and their encodings are as follows.

Table 5-67 DMASK Transformation Commands

Value	CmdA Function	CmdB Function
0	A' = A	B' = B
1	A' = M	B' = M
2	A' = A & M	B' = B | M
3	A' = A | M	Power-gate B' output
4	A' = A & (M | B)	—
5	A' = A | M & B	—
6	A' = A & B | M	—
7	A' = B & M	—
8	A' = (A | M) & B	—
9	A' = A & B	—
10	A' = A | B	—
11	A' = B	—
12	A' = B | M	—
13	A' = A | B | M	—
14	A' = A & B & M	—
15	A' = A & M | B	—

Only the DMASK_B channel supports power gating. Each time this scratch channel is not needed, it should be turned off to save power by specifying a CmdB of four.

In any of these cases, if all input channels are unavailable, for example due to power-gating of B or due to not performing an L2F table lookup, the command reverts to case zero. That is, the assignments become either A' = A or B' = B, depending on the channel.

Otherwise, if the operands are only partially valid, the missing terms are treated as either all-one's or all-zero's in the transformation equations previously listed. If the term is combined in an AND expression (&), the unavailable mask is interpreted as all-ones, while in an OR expression (|), it is interpreted as all-zeros.

At the beginning of the DMASK generation pipeline, the rippling channels are initialized as follows:

- DMASK_A: All-ones
- DMASK_B: Power-gated
- DROP_CODE: 0

Figure 5-34 shows some example transformation functions supported by the DMASK transformer.

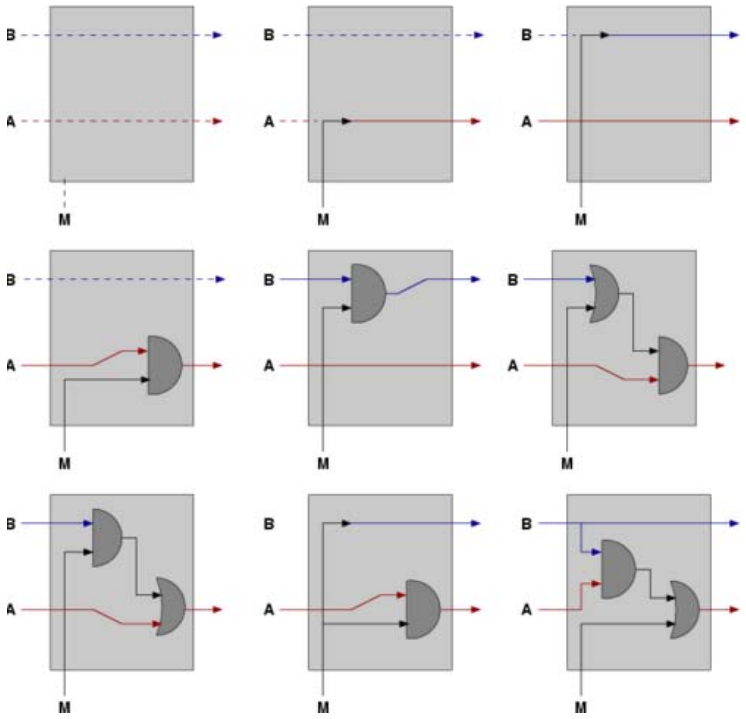


Figure 5-34 DMASK Example Transformations

5.15.3 L2 Filtering Tables

The initial L2F phase of DMASK generation consists of a sequence of 13 table lookups that each provide 76-bit masks (M) and command operands to DMASK transformers. Each stage transforms four rippling channels:

- **DMASK_A, DMASK_B** (76 bits) — Manipulated by the stage's DMASK transformer as previously described.
- **DROP_CODE** (8 bits) — Set by the DMASK transformer each time the DMASK_A channel transitions from a non-zero to a zero value. An L2F stage ID (1 through 13) is assigned to either the high or low four bits of this channel, depending on a bit of the profile command. The unassigned nibble is left as is.
- **ISTATE** (13 bits) — Each time a table lookup is performed, producing a 76-bit mask (M), the stage is set ISTATE[n] to M[SRC_PORT], where n corresponds to the stage number (0 through 12). The final 13-bit ISTATE value is presented to L2AR as a TCAM key field.

All L2F tables are functionally identical except for their sizes. The first nine tables have 4096 entries, while the last four have 256 entries.

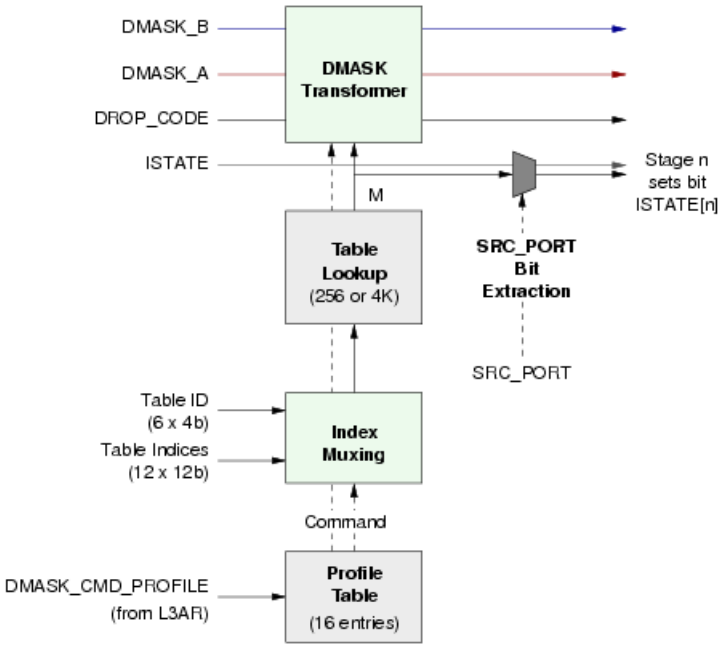


Figure 5-35 DMASK Layer 2 Filtering Tables

The lookup index used for each table lookup, as well as other necessary command information, is determined by L2F_PROFILE_TABLE[n,DMASK_CMD_PROFILE], where n is the L2F stage number. The DMASK profile number, specified by L3AR's SetDestMaskCmdProfile action, is shared among all L2F stages. The profile number identifies one of 16 command profiles that might be independently configured per stage.

The profile table of each L2F stage produces the following 18-bit command:

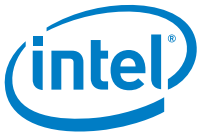


Table 5-68 DMASK L2F Command

Field	Width	Description
IndexSelect	4	Selects a table index from a pool of 12 possibilities: • SRC_PORT — For source-based and multi-chip loopback suppression • IVID1, IVID2, EVID1, EVID2 — For VLAN membership and STP indexing • L3AR_DMASK_IDX1, L3AR_DMASK_IDX2, L3AR_DMASK_IDX3 — For direct L3AR indexing, such as L3_DMASK_IDX. • L2L_ET_IDX, L2L_IT_IDX — Mapped indices from L2 lookup (intended for STP indexing from VID1) • GLORT_DMASK_IDX — Primary L2 destination mask index from the GloRT lookup. • ALU3_Z — Arithmetically generated, available for configurable use.
TableSelect	3	Selects one of six table identifier values. The selected 4-bit table identifier channel must match the value specified by TableID for a lookup to be performed. If the channel value does not match TableID, no lookup is performed, regardless of the value of CmdLookup. Available TableSelect channels include: • GLORT_DMASK_IDX[15:12] • L3AR_DMASK_IDX3[15:12] • MA1_TAG[11:8] • MA2_TAG[11:8] • ALU3_Z[15:12] • DGLORT_TAG (one bit) • constant 0 This mechanism enables index ranges larger than 4-Kb to be distributed over arbitrary sets of L2F stages.
TableID	4	Value to which the selected 4-bit table identifier channel is compared. A lookup is performed only if the channel value matches TableID.
CmdLookup	1	Controls whether a table lookup is performed or not. 0b = No lookup performed 1b = Lookup performed. When CmdLookup is set to 1b, a lookup only occurs if TableID matches the corresponding four bits selected by TableSelect.
CmdA	4	DMASK Transformer DMASK_A command.
CmdB	3	DMASK Transformer DMASK_B command.
DropCodeSelect	1	To support some degree of non-linearity in the drop sequencing among L2F stages, each configuration stage can set either DROP_CODE[7:4] or DROP_CODE[3:0] according to the value of this field. 0b = If set to 0b, and DMASK_A goes to zero due to the action of this stage, DROP_CODE[3:0] is set to DropCode and DROP_CODE[7:4] is left as is. 1b = DROP_CODE[7:4] is set to DropCode and DROP_CODE[3:0] is left as is.
DropCode	4	Constant value to assign to either DROP_CODE[3:0] or DROP_CODE[7:4] when DMASK_A is zeroed.

5.15.4 EACLs

The EACL stage immediately follows the L2 filtering stages. The EACLs take, as their input, the state of DMASK_A at the output of L2F. One result of the EACLs is a permit DMASK that is normally expected to be ANDed with DMASK_A. For flexibility and uniformity, the permit DMASK is applied using a DMASK transformer. The transform commands are determined by a lookup in EACL_PROFILE_TABLE, indexed by DMASK_CMD_PROFILE.

5.15.5 LAG Filtering

The LAG_DMASK calculated by the GloRT Lookup stage is applied to the DMASK channels immediately after the EACL stage. Its DMASK transformer commands are determined by LAG_PROFILE_TABLE, also indexed by DMASK_CMD_PROFILE.

5.15.6 LBS Filtering

The LBS DMASK M is computed at this stage. It is initially set to all ones (frame accepted on all ports). The source GloRT assigned to the frame being switched is then compared to a canonical source GloRT assigned to each port, and each matching port is subtracted from the DMASK. Similarly to other DMASK transforms, the DMASK transformer commands are determined by LBS_PROFILE_TABLE, indexed by DMASK_CMD_PROFILE. The canonical GloRT for each port is defined in LBS_CAM[0..75] where a match for a given port removes the port from the DMASK.

5.15.7 Outputs to L2AR

Table 5-69 lists the DMASK-related outputs generated by the L2F, EACL, and LAG stages that are passed on the to L2AR, the final stage of configurable DMASK processing.

Table 5-69 DMASK-related Outputs

Channel	Width	Description
DMASK_A	76	Primary L2 destination port mask channel.
DMASK_B	76	Secondary DMASK channel,. Usually set to LAG_DMASK so any additional ports added to DMASK_A by L2AR might be LAG-filtered.
DROP_CODE	8	Assigned when DMASK_A becomes zero (indicating the frame is to be dropped to all destinations) to identify which filtering stage caused the drop. Initializes the L2AR DROP_CODE field. The DROP CODE is set at the first of DMASK_A getting to zero and is not changed after even if the DMASK_A is returned to a non-zero value and zero again.
ISTATE	13	One bit per L2F table, selected as the SRC_PORT bit from the table lookup output. Expected to contain Ingress VLAN and spanning tree state associated with the frame's source port and IVID1/IVID2. Presented to L3AR as a TCAM key for policy handling.
ACTION_FLAGS[DMASK_SingleDest]	1	Tests whether |DMASK_A|==1, where |DMASK_A| represents the total number of bits set to 0x1 in DMASK_A immediately prior to L2AR.

5.16 Egress ACLs

The EACL unit is responsible for resolving the application of egress access rules. The features supported are:

- Up to 1024 egress rules
- Up to 32 access control lists
- Allocation of rules to list in set of 32 rules
- Arbitrary assignment of ports to list
- Two actions implemented in fixed hardware:
  - Permit/deny action
  - Count action
- Four configurable actions:
  - Three are shared actions between lists
  - One can be uniquely defined per list

5.16.1 Functional Description

Figure 5-36 shows the overall EACL unit and its interaction with the other surrounding units.

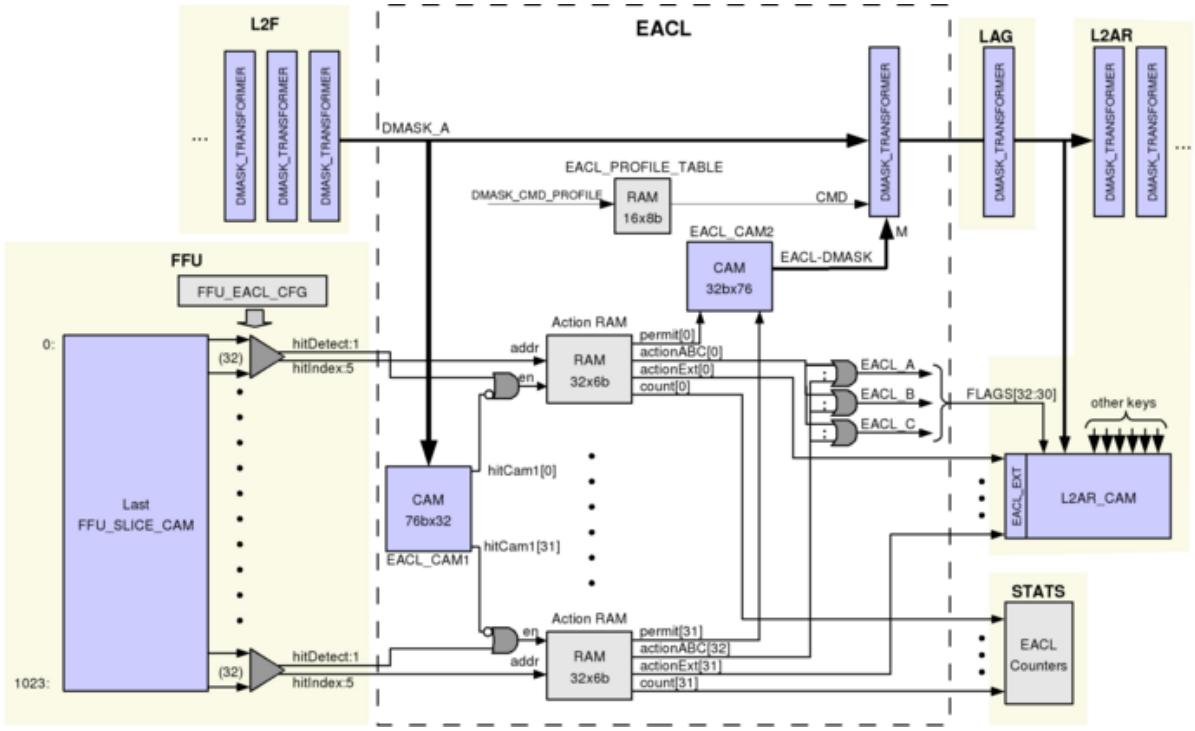


Figure 5-36 EACL Block Diagram

The EACL starts at the FFU. The rules must be loaded into the FFU and must terminate at the last slice of the FFU. This slice contains up to 1024 rules, any portion of which can be used for either ingress or egress ACLs, in sets of 32 rules (aligned to a multiple of 32). If all 1024 rules are used for egress, up to 32 sets of 32 rules can be defined. By cascading up to 16 sequential sets, a given egress ACL can be constructed to contain as many as 512 rules. ACLs cannot be cascaded between the lower 16 sets and the upper 16 sets.

Using the FFU_EACL_CFG register, each set outputs a hitDetect signal to indicate if a rule was hit within the cascade and a hit-index pointing to the highest priority rule. The FFU_EACL_CFG register defines two bits per set:

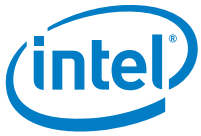
- **Valid** — Indicates whether each set is used for EACLs.

  Defines whether each 32-rule set is used for EACL purposes. If Valid is set to 0b, the set's EACL hit detect function always returns 0b and the value of its StartList is ignored.

- **StartList** — Indicates whether each set starts a new list or continues the previous list.

  Specifies the partitioning of rules between EACLs in units of 32-rule sets. If StartList for set i is 1b, only one rule is permitted to hit in all sets starting from set i until the next valid set j>i with StartList[j]=1b. Within each list, rules in lower-numbered sets have precedence. However, contrary to other FFU hit detect functions, within a given set, higher-numbered rules have precedence.

Each EACL's preliminary hitDetect, as determined by the FFU, is then filtered based on the content of the forwarding destination port mask from L2F. The egress rule is canceled if the destination mask doesn't include any port member of this egress ACL.



The EACL set's port membership is programmed in EACL_CAM1, one entry for each set. The entry programmed in the CAM is actually the inverse of the membership so that ports that are not members must be ignored. The destination mask from L2F is presented as a key to the CAM and a hitCam1 bit is generated for each set. The bit is set to 1b if the destination mask does not include any port member of this EACL set. This bit is then used to cancel application of the egress rule. As an example, if ports 4, 5, 6, 7 are member of EACL set 16, the entry in the EACL_CAM1[16] must be:

```
EACL_CAM1[16][0].KeyInvert = 0xFFFFFFFF_FFFFFFFF;
EACL_CAM1[16][0].Key       = 0xFFFFFFFF_FFFFFF0F;
EACL_CAM1[16][0].KeyInvert = 0xFFF;
EACL_CAM1[16][0].Key       = 0xFFF;
```

The resulting filtered hit-detect and the hit-index are then used to retrieve actions from a 32x6-bit SRAM for a given set. There are 32 banks of 32x6-bit RAM, one bank per set. The actions can be any of the following:

- permit(1)/deny(0)
- count(1)/no-count(0)
- actionA
- actionB
- actionC
- actionExt

The default actions if no hit is detected for a set are the following:

- permit(1)
- no-count(0)
- no-actionA(0)
- no-actionB(0)
- no-actionC(0)
- no-actionEXT(0)

The 32 permit/deny actions are then sent as a key to a second CAM (EACL_CAM2) to produce a permit DMASK. To achieve this, each entry in the CAM is programmed to define the set list applicable for a given port. The entry j must be set to look for a 1b for set i if this set applies to port j and for a don't care bit if this set doesn't apply for this port. The resulting output hit vector from the CAM is a permit DMASK that is forwarded to a dedicated DMASK transformer (see Section 5.15.2), to be processed with the DMASKs from L2F stages to produce a new DMASK set. The command to the DMASK transformer comes from the EACL_PROFILE_TABLE entry indexed by the DMASK_CMD_PROFILE coming from L3AR.

The 32 count actions are sent to STATS for counting. The STATS module includes 32 parallel EACL counters.

The actionA, actionB and actionC are generic shared actions. The 32 actionA (one per set) are ORed together and available as a single bit EACL-A of the action flags and thus available as a key to any L2AR slices. The same process is applied for actionB and actionC. An example usage might be for implementation of a log, mirror or trap action. If two Egress ACLs both hit on different rules and both command a log action, the L2AR receives one log bit and can implement the proper action for that frame to be logged.

The actionExt bits are available as a per-set action definition. The 32 actionExt bits are presented as EACL_ACTION_EXT to all L2AR slices enabling each L2AR slice to define a distinct action per list. Rule i in slice j must match EACL_ACTION_EXT[i] if the configuration bit in L2AR_EACL_EXT[j][i] is set.

5.16.2 Registers

The configuration includes the following elements:

Table 5-70 Egress ACL L2AR Keys

Register/Table	Usage
FFU_EACL_CFG	Enables sets and defines cascading. Located in FFU unit
EACL_CAM1	Defines port-list per set
EACL_ACTION_RAM	Defines actions for each rule for each set
EACL_CAM2	Defines set list per port
EACL_PROFILE_TABLE	Defines the DMASK transformation command.

5.17 L2 Action Resolution

The Layer 2 Action Resolution (L2AR) stage performs the final configurable processing of each frame prior to scheduling. It is responsible for finalizing all forwarding decisions and for formulating the important forward channel, which encodes all egress scheduling and modification directives. Functions performed by this unit include:

- All policy-based trapping, mirroring, and dropping decisions.
- Precedence resolution among all applicable forwarding policies.
- Special frame classification (such as reserved IEEE DMAC handling).
- Security handling based on SMAC lookup result.
- EACL handling.
- All MAC table learning/update handling, including formulating the entry.
- Final QoS determination and policer-based coloring transformations.
- Selecting frame header fields for statistics counters.
- Formulation of MOD_DATA for egress modification handling.
- Congestion notification frame sampling.
- Ingress rate limiting.

5.17.1 Overview

The L2AR stage functionality is implemented using microcode. Figure 5-37 shows the overall architecture.

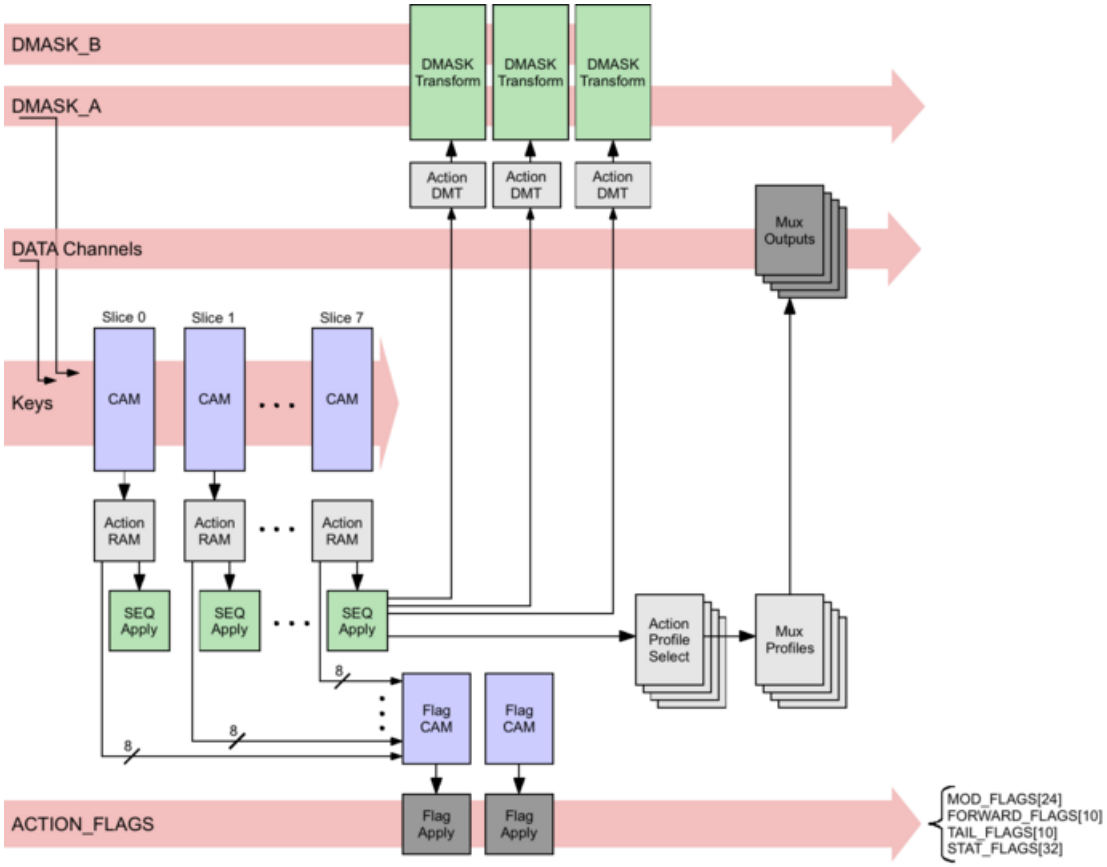


Figure 5-37 L2AR Block Diagram

The L2AR stage has the following generic properties:

- 512 rules
  - — Implemented in 8 precedence sets of 64 rules each
  - — Total key width per rule of 378 bits
- Two stages of action flags updates
- Three stages of DMASK transform
- 18 mux output blocks

An L2AR rule is defined by its key matching state (value and mask for all keys) and one or more actions to apply to the frame if the rule is selected. A rule is selected (or wins) if:

1. Its key matching state successfully matches against an ingress frame's corresponding key values.

2. No higher-precedence (higher-numbered) rule in the same extended precedence set also matches.

A precedence set is a collection of 64 sequential rules from which only one can match at a time. Configurable precedence sets can be chained together to create larger extended precedence sets. The total number of eight precedence sets put a limit on the maximum number of rules that can simultaneously apply to a given frame. Chaining precedence sets together reduces this parallelism.

5.17.2 Keys

The following set of frame properties are available as matching conditions to each L2AR rule. For each of these keys, a value and mask is configured for TCAM-based comparison:

Table 5-71 L2AR Keys

Key	Width	Notes
DMASK_A	76	State of the DMASK_A channel after LAG filtering. This field has bitwise-OR matching semantics. That is, if any DMASK_A bit is set in the key (Key[i]=1b and KeyInvert[i]=0b), the entire field matches. Used typically for egress mirrors.
SMASK	76	Bit vector over ingress ports matched against decoded SRC_PORT (such as 1<<SRC_PORT). Used for a number of purposes: • Ingress mirrors • General per-SRC_PORT dependent processing
ACTION_FLAGS	76	Action flags as calculated through the pipeline. Action flags for specific bit definitions.
ACTION_DATA.W8F	8	Intended as a generic 8-bit QoS L2AR key as a substitute for all 22 QoS bits. L3AR muxes some subset of the QoS channel into this field, (QOS.W4, QOS.ISL_PRI) by default.
POL1_TAG1_TOP	4	Top four bits of each entry's TAG value from POLICER_CFG_4K[0].
POL1_TAG2_TOP	4	
POL2_TAG1_TOP	4	Top four bits of each entry's TAG value from POLICER_CFG_4K[1].
POL2_TAG2_TOP	4	
POL3_TAG_TOP	4	Top four bits of the TAG value from POLICER_CFG_1K.
DROP_CODE	8	Drop code associated with DMASK_A at the input of L2AR. Identifies the L2F, EACL, or LAG stage (if any) that zeroed DMASK_A.
L2F_ISTATE	13	Ingress STP/VID state, as returned by L2 filtering tables. Used for: • Ingress filtering • Controlling learning
DGLORT_TAG	1	Tag associated with the destination GloRT's DGLORT_TABLE entry.
ALU13_Z	16	OR over selected ALU{1,2,3}_Z outputs as specified by ALU_CMD_TABLE[0..2,ALU13_CMD_PROFILE].OrIntoL2ARKey.
ALU46_Z	16	OR over selected ALU{4,5,6}_Z outputs as specified by ALU_CMD_TABLE[3..5,ALU46_CMD_PROFILE]OrIntoL2ARKey.
MA1_FID2_IVL	1	MA1 FID2 mux control bit. Can be interpreted as 12th bit of ETAG1 in some configurations.
MA2_FID2_IVL	1	MA2 FID2 mux control bit. Can be interpreted as 12th bit of ITAG1 in some configurations.
MA1_LOOKUP	1	Was a MA1 lookup (DMAC) performed?
MA1_HPV	4	MA1 lookup HPV.
MA1_TAG	12	Tag associated with the MA1 entry (DMAC).

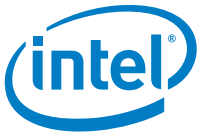


Table 5-71 L2AR Keys (Continued)

Key	Width	Notes
MA2_LOOKUP	1	Was a MA2 lookup (SMAC) performed?
MA2_HPV	4	MA2 lookup HPV.
MA2_MPV	4	MA2 lookup MPV used for entry write-back control (including learning).
MA2_TAG	12	Tag associated with the MA2 entry (SMAC).
L2L_ETAG1	12	Tag associated with EVID1.
L2L_ETAG2	12	Tag associated with EVID2.
L2L_ITAG1	12	Tag associated with IVID1. Used to implement per-VLAN configuration: • Reflect loopback-suppression • TrapIGMP
L2L_ITAG2	12	Tag associated with IVID2.
DGLORT	16	ISL tag destination GloRT (post-MAC table).
SGLORT	16	ISL tag source GloRT (post-MAC table).
L2_DMAC_ID3	5	Highest precedence match in mapping stage DMAC_CAM3. Used for: • CPU MAC address trapping • 802.1x trapping • GMRP and GVRP trapping • LACP trapping • Other reserved IEEE multicast trapping • BPDU trapping/dropping ***Note:*** By convention, 0xF should be reserved to indicate no match (all mask bits set to zero), leaving 15 usable values.
L2_SMAC_ID3	5	Highest precedence match in mapping stage SMAC_CAM[16..31]. Provided mostly for symmetry with DMAC. By convention, 0xF should be reserved to indicate no match (all mask bits set to zero).
L2_TYPE_ID2	4	Highest precedence match in mapping stage TYPE_CAM[16..31]. Used for pause and other special frame type classification. By convention, 0xF should be reserved to indicate no match (all mask bits set to zero).
ISL_USER	8	User bits from ISL tag

5.17.3 EACL Extended Actions

The EACL extended actions are described in the EACL section. There are up to 32 egress extended actions, one per egress ACL. Each action is a one-bit condition, that if enabled, defines if the rule matches on an extended action or not. The bit vector table L2AR_EACL_EXT[0..15] controls if the extended action is used or not. If the bit i of L2AR_EACL_EXT[n] is 0b, the extended action from list i/2 is ignored for slice n. If the bit is set to 1b, the rule i in slice n only hits if the extended action is 1 for list i/2.

5.17.4 Actions

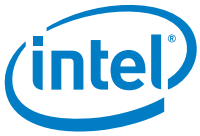
Each L2AR rule supports any of the following set of actions (defined in L2AR_RAM):

Table 5-72 L2AR Actions

Action	Width	Description
FLAGS_TAG	8	Sets an 8-bit tag to be used in two-stage action flag transform. Each slice has a unique 8-bit set available to this slice.
TransformDestMask	1	Indicates whether this rule requests a DMASK transformer action.
DMT_PROFILE	5	Defines the DMASK transformer profile to use if a DMASK transformer action is requested.
DMT_NEXT_STAGE	1	If set to 1b, the DMT stage controlled by this rule or any rules in subsequent slices are incremented. Applies regardless of the value of TransformDestMask. The DMT stage number begins at zero and saturates at two.
SetCpuCode	1	Enables CPU code assignment action.
SetTrapHeader	1	Enables header trapping action.
SetMirror[0..3]	4x1	Enables mirror action.
MuxOutput_QOS	1	Enables QoS output action.
MuxOutput_MA_WRITEBACK	1	Enables MAC table write back output action.
MuxOutput_DGLORT	1	Enables DGLORT output action.
MuxOutput_W16AB	1	Enables W16{A,B} output action.
MuxOutput_W16CDEF	1	Enables W16{B,C,D,E,E} action.
MuxOutput_W8ABCDE	1	Enables W8{A,B,C,,D,E} action.
MuxOutput_W4	1	Enables W4 action.
MuxOutput_VID	1	Enables VID outputs action.
MuxOutput_DMASK_IDX	1	Enables DMASK_IDX output action.
MuxOutput_STATS_IDX5AB	1	Enables STATS_IDX5{A,B} output action.
MuxOutput_STATS_IDX5C	1	Enables STATS_IDX5C output action.
MuxOutput_STATS_IDX12A	1	Enables STATS_IDX12A action.
MuxOutput_STATS_IDX12B	1	Enables STATS_IDX12B action.
MuxOutput_STATS_IDX16A	1	Enables STATS_IDX16A output action.
MuxOutput_STATS_IDX16B	1	Enables STATS_IDX16A action.

For all actions, with the exception of flags tag, the sequential apply resolves which action takes precedence. The sequential apply proceed as follows:

- Within each precedence set, only one rule can win and actions are recovered for this rule.
- Between precedence sets, the enabled actions of winning rules are evaluated sequentially in precedence set order (lower number first). The higher precedence set always win.



For flags tag, each precedence set output a unique 8-bit value for this slice. This tag is then used during the action flag updates.

The DMT action profile has special handling. There are three DMT actions for the entire L2AR unit but a given rule can only specify one DMT action. The DMT transformer selected depends on the accumulated DMT_NEXT_STAGE bits. for example, each rule that set DMT_NEXT_STAGE causes this rule and any following one to pick the next DMT transformer (saturation at DMT transformer 2). The initial DMT transformer used is zero.

All other actions are split between set actions (such as SetCpuCode) and select multiplexer output profile actions. The index {slice,rule} of the winning rule for this action is used to index the corresponding L2AR_ACTION table to recover a final value for set actions or an intermediate profile index for the multiplexer outputs actions. The multiplexer output profile index is then used to recover the multiplexer output profile.

Table 5-73 L2AR Actions

Action	Type	Configuration	Operands	Description
SetCpuCode	Set	CPU_CODE (8 bits)	ACTION_DATA.W8E	Set CPU code.
SetMirror[0..3]	Set	MIR_RX (1 bit) MIR_TX (1 bit) MIR_TRUNC (1 bit) MIR_MAP_PRI (1 bit)	_	Specifies the mirroring command for any of the four mirroring mechanisms. Setting either MIR_RX or MIR_TX to 1b enables the mirror. The MIR_TRUNC and MIR_MAP_PRI parameters influence how the mirror frame is generated and scheduled.
	Set	ENABLE (1 bit) IDX (1 bit)	_	If the final state of TRAP_HEADER_ENABLE is 1b, specific frame data fields are stored in L2AR_HEADER_TRAP_DATA[TRAP_IDX].
SetDestGlort	MCM	Profile (5 bits)	DGLORT	Update DGLORT.
MuxOutput_W16AB	MCM	Profile (5 bits	SGLORT ACTION_DATA	MOD_DATA.W16AB muxing.
MuxOutput_W16CDEF	MCO	Profile (5 bits)	Many	MOD_DATA.W16* muxing.
MuxOutput_W8ABCDE	MCM	Profile (5 bits)	Many	MOD_DATA.W8* muxing.
MuxOutput_W4	MCM	Profile (4 bits)	Many	MOD_DATA.W4 muxing.
MuxOutput_VID	MCO	Profile (5 bit)s	{I,E}VID{1,2}	L2_VID{1,2} muxing for IP multicast handling and egress modification.
MuxOutput_DMASK_IDX	MCO	Profile (5 bits)	L3AR_DMASK_IDX3 GLORT_DMASK_IDX	IP multicast DMASK_IDX muxing.
MuxOutput_QOS	MCM	Profile (5 bits)	QOS	Assigns one of 32 mapping profiles for the 24-bit QoS collection of fields.
MuxOutput_MA_WRITEBACK	MCO	Profile (5 bits)	Many	MA_TABLE writeback entry muxing.
MuxOutput_STATS_IDX5AB MuxOutput_STATS_IDX5C	MCO	Profile (5 bits)	Many	Muxing for statistics counter 5-bit index channels.
MuxOutput_STATS_IDX12A MuxOutput_STATS_IDX12B	MCO	Profile (5 bits)	Many	Muxing for statistics counter 12-bit index channels.

5.17.5 Action Flags

The 76-bit ACTION_FLAGS vector is an accumulation of properties, conditions, and other tests evaluated throughout the FPP. One of the L2AR's primary purposes is to determine a final handling disposition for the frame based on the status of all these conditions.

The first 40 bits of ACTION_FLAGS originate from the parser. Bits 51:40 come from the mapper, FFU, and next hop table stages. Note that L3AR might arbitrarily overwrite any flag bits it has access to, namely bits 51:0. Bits 74:52 are set by fixed-function evaluations in the logic between the L3 and L2AR stages: L2 lookup, the ALUs, and L2 filtering. Bit 75 is not used by the preceding stages of the pipeline and therefore is always zero at the input of L2AR.

Table 5-74 L2AR Action Flags

Flag	Bit	Notes
L3AR Flags	51:0	Propagated unmodified from L3AR, except for: Bit 30 = EACL_A Bit 31 = EACL_B Bit 32 = EACL_C
MA2_WriteBackEnabled	52	Indicates the MA2 lookup entry in the MAC table can be written back.
MA2_WriteNewEnabled	53	Indicates whether a new entry might be written into the MAC table associated with the MA2 key.
ALU1_Overflow	54	Overflow output from ALU1. Indicates X<Y if the ALU operation was SUB with OneTwo=0b. Indicates X>Y if the ALU operation was SUB with OneTwo=1b.
ALU1_Zero	55	Indicates the output from ALU1 is zero.
ALU2_Overflow	56	Overflow output from ALU2. Indicates X<Y if the ALU operation was SUB with OneTwo=0b. Indicates X>Y if the ALU operation was SUB with OneTwo=1b.
ALU2_Zero	57	Indicates the output from ALU2 is zero.
ALU3_Overflow	58	Overflow output from ALU3. Indicates X<Y if the ALU operation was SUB.
ALU3_Zero	59	Indicates the output from ALU3U is zero.
ALU4_Overflow	60	Overflow output from ALU4. Indicates X<Y if the ALU operation was SUB.
ALU4_Zero	61	Indicates the output from ALU4 is zero.
ALU5_Overflow	62	Overflow output from ALU5. Indicates X<Y if the ALU operation was SUB with OneTwo=0b. Indicates X>Y if the ALU operation was SUB with OneTwo=1b.
ALU5_Zero	63	Indicates the output from ALU5 is zero.
ALU6_Overflow	64	Overflow output from ALU6. Indicates X<Y if the ALU operation was SUB.
ALU6_Zero	65	Indicates the output from ALU6 is zero.
DMASK_SingleDest	66	Indicates that exactly one bit is set in DMASK at the input of L2AR.
POL1_OverRate1	67	Indicates the corresponding policer entry is out-of-profile. See Section 5.13, "Policers".
POL1_OverRate2	68	
POL2_OverRate1	69	
POL2_OverRate2	70	
POL3_OverRate	71	
POLICER_QOS_MAP1_EQ01	72	Indicates the input index to POLICER_QOS_MAP1 equals the first-order mapped output.
POLICER_QOS_MAP1_EQ12	73	Indicates the POLICER_QOS_MAP1's first-order and second-order mapped outputs are equal.

Table 5-74 L2AR Action Flags (Continued)

Flag	Bit	Notes
POLICER_QOS_MAP2_EQ01	74	Indicates the input index to POLICER_QOS_MAP2 equals the first-order mapped output.
POLICER_QOS_MAP2_EQ12	75	Indicates the POLICER_QOS_MAP2's first-order and second-order mapped outputs are equal.

The L2AR is also responsible for updating the actions flags and generate a new set of output flags for consumption in follow on stages of the pipeline. The FLAGS_TAG is available to help define the final action flags. The 8-bit FLAGS_TAG are unconditionally set by the winning rule of each slice. If a slice as no hit, the FLAGS_TAG are set to zero for that slice. The result is 64 bits of tagging information, which is then presented as a key to two 64bx76 CAMs.

The output of each CAM lookup is a 76-bit array, one bit per action flag indicating if the 64-bit FLAGS_TAG key presented match a certain pattern. The match/miss is then used to modify the action flags using the L2AR_FLAGS_CAM_POLARITY and L2AR_FLAGS_CAM_VALUE as operands. The action flag apply logic is a flag to the value defined in L2AR_FLAGS_CAM_VALUE[i]{flag} if there is a match and polarity is zero, or a miss for polarity one. Otherwise the action flag remains unchanged.

```
foreach i (0..1)
  match = CAM_lookup(L2AR_FLAGS_CAM[i],FLAGS_TAG)
  foreach flag (0..75)
    if ( match{flag} ^ L2AR_FLAGS_CAM_POLARITY[i]{flag} )
      ACTION_FLAGS{flag} = L2AR_FLAGS_CAM_VALUE[i]{flag}
    else
      ACTION_FLAGS{flag} = ACTION_FLAGS{flag}
```

At the output of the L2AR flag update stages, bits 23:0 are sent on the forward channel as MOD_FLAGS, with eventual configurable interpretation by the egress modify stage. Bits 33:24 are also sent on the forward channel and influence the frame's scheduling behavior, such as store-and-forward and mirroring. Bits 43:34 are preserved by the FPP as TAIL_FLAGS and these bits influence how the frame is handled once its tail arrives. Bits 75:44 are sent to the statistics unit where they are interpreted by the stats action resolution stage for frame counting purposes.

5.17.6 TransformDestMask

The L2AR TransformDestMask action shares the same DMASK transformer function as all stages in the DMASK generation pipeline. The L2AR action RAM specifies the CmdA, CmdB, M, and DC operands that are used to transform the DMASK_A and DMASK_B channels that ripple from slice-to-slice. At the end of the L2AR slices, DMASK_A becomes the final destination forwarding mask, which can be further modified only by congestion management. DMASK_B propagates no further than the last L2AR slice.

The final DROP_CODE value is available as mux inputs to the MOD_DATA.W8E and the 5-bit STATS_IDX5{A,B,C} channels.

The DMASK transformer function supports a number of potential DMASK transformations. One expected usage mode is to drop or add ports to the forwarding distribution while LAG-filtering any additional ports using DMASK_B. Table 5-75, provided for informational purposes only, lists some possible operations supported in this usage mode.

Table 5-75 L2AR DMASK Action Commands

Transformed DMASK_A	Use
DMASK_A | M	Used for the default case (NOP) with ACTION_DMASK=0b, or to extend the forwarding distribution to additional output ports.
DMASK_A & M	Exclude (drop) ports from distribution.
M	Override DMASK with action-specified value.
DMASK_A | M & LAG_MASK	Include additional ports in distribution, with LAG filtering.
M & LAG_DMASK	Override DMASK with action-specified value, with LAG filtering.
(DMASK_A | M) & LAG_DMASK	Apply alternate LAG filtering mask to DMASK, possibly with additional ports included from M (prior to LAG filtering).
DMASK_A & LAG_DMASK | M	Apply alternate LAG filtering mask to DMASK, possibly with additional ports included from M (after LAG filtering).

Note: M represents the L2AR action mask, referenced as ACTION_DMASK in the register definitions.

5.17.7 Output Flags

The L2AR output flags are partitioned into four sets based on the recipient stage:

- **MOD_FLAGS[23:0]** — Microcode-defined directives influencing the frame's egress modification.
- **FORWARD_FLAGS[9:0]** — Forwarding directives to the scheduler.
- **TAIL_FLAGS[9:0]** — Directives for the frame's tail handling.
- **STATS_FLAGS[31:0]** — Microcode-defined frame properties of interest to the statistics stage.

All bits of FORWARD_FLAGS have fixed-function interpretation by the scheduler. Table 5-76 shows the definition of these flags.

Table 5-76 L2AR FORWARD_FLAGS Definition

Field	Bits	Description
L2_VID_Select	0	Selects between L2_VID1 or L2_VID2 for the L3-multicast egress VID comparison check in MCAST_POST. Also selects whether such frames' L2_VID1 or L2_VID2 field is overridden by the new VID obtained from the MCAST_VLAN_TABLE.
L2_TAG	1	A one-bit tag used in MCAST POST to control loopback suppression. It is compared to the one-bit TAG in MCAST_VLAN_TABLE and only if the two bits match the frame that is eligible for loopback suppression.
SNF_Override	2	If set to 0b, the frame is forced into store-and-forward mode only if required by the SAF_MATRIX lookup (see Section 5.19, "Packet Replication".)
CN_SampleEligible	3	Controls whether the frame is eligible for congestion notification sampling (see Section 5.18.10, "Congestion Notification Frame Sampling"). If set to 1b, the frame is forced unconditionally into store-and-forward mode.
TX_Trunc	4	Setting this flag instructs the scheduler to forward only the first segment of the frame (up to 160 bytes) to all egress ports that have MCAST_TX_TRUNC_MASK[p] set to 1b. The flag only applies to normally-forwarded frames (such as non-mirrored frames). Truncation of mirrored frames is controlled by the SetMirror action. Additionally, the TX_Trunc bit is propagated to the egress modification stage, where is available in the modify slice TCAM key. Microcode can interpret the flag in a configurable manner to further truncate individual egress frames to specific lengths less than 160 bytes.

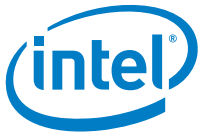


Table 5-76 L2AR FORWARD_FLAGS Definition

Field	Bits	Description
TX_IgnoreError	5	By default, any store-and-forwarded frame is dropped if a CRC or other line error (referred to as a tail error) is detected during ingress. Cut-through frames are never dropped. If this bit is set, such store-and-forwarded error frames are forwarded as normal. For example, they are not dropped.
Reserved	9:6	Reserved.

The TAIL_FLAGS field contains flag bits that are preserved until all bytes of the frame are received. At that point, with the frame's length and error status known, all tail processing functions are then executed based on the state of these tail flags. The TAIL_FLAGS channel has the following fixed-function bit definitions, as shown in Table 5-77.

Table 5-77 L2AR TAIL_FLAGS Definition

Flag	Bit	Description
PAUSE_Frame	0	Indicates a pause frame, either port- or class-based.
PAUSE_CBPFrame	1	Indicates if the pause frame is a class-based pause or a normal pause frame.
POL1_Credit1	2	Controls whether the frame (and its associated byte length) is credited to the corresponding policer entry's counter or token bucket. These bits are overridden to zero if the frame is dropped by congestion management.
POL1_Credit2	3	
POL2_Credit1	4	
POL2_Credit2	5	
POL3_Credit	6	
MA_WriteBack	7	Indicates the MA table entry, as constructed by the MA_WRITEBACK mux set, should be written back into the MAC table, overwriting the entry that was looked up with the MA2 key. This directive is only respected if the MA2_WriteBackEnabled flag was set.
MA_WriteNew	8	Indicates a new MA table entry, as constructed by the MA_WRITEBACK mux set, should be written into the MAC table. This directive is only respected if the MA2_WriteNewEnabled flag was set and MA_WriteBack is set to 0b. For any given frame, it is illegal to both update an existing entry and write a new entry in the MAC table.
Reserved	9	Reserved.

5.17.8 SetMirror

Generating mirror frames is controlled by the SetMirror[0..3] actions. The FM5000/FM6000 supports four mirror mechanisms that are independently enabled by these actions, each capable of generating one mirrored copy of the source frame. Thus, by enabling all four mirrors, up to four mirror frames can be generated from any given ingress frame.

Each mirror action specifies whether its mirror frame is generated based on the ingress state of the frame (Rx mirroring) or the egress state of the frame (Tx mirroring). Although L2AR might enable each mirroring action based on any condition it likes, static register configuration downstream in the pipeline impose the following constraints on Tx mirroring:

- Tx mirroring is enabled only when the DMASK distribution includes one or more of a specific set of ports, configured by CM_TX_MIRROR_SRC[mirnum].SrcMask.

- The Tx mirror frame is generated according to the VLAN replication and egress modification rules applicable to a single canonical Tx mirror source port, configured redundantly by the CanonicalSrcPort fields of MCAST_MIRROR_CFG[mirnum] and MOD_TX_MIRROR_SRC.

No such constraints are imposed on Rx mirroring. If both Rx and Tx mirroring modes are enabled, the final mode depends on the following condition:

```
FINAL_DMASK & CM_TX_MIRROR_SRC[mirnum].SrcMask != 0
```

That is, Tx mirroring takes precedence. The FINAL_DMASK is the DMASK following all drop decisions applied by the congestion management stage.

Two forwarding modes are supported per mirror:

- **Explicit Forwarding** — In this mode, when L2AR enables a mirror, it controls where the mirror frame is sent to by setting the appropriate bit(s) of DMASK_A. For the mirror frame to be generated correctly, any bits set must also be set in the ExplicitDestMask field as configured in the CM_TX_MIRROR_DEST and MCAST_MIRROR_CFG registers. This forwarding mode is most useful for mirroring to a LAG; however, it imposes the following two restrictions:
  - On a given egress port, either the source frame or the mirror frame might be sent, bot not both.
  - If more than one explicit mirror is enabled for a given frame, their destination masks must not overlap.
- **Overlaid Forwarding** — In this mode, L2AR does not modify DMASK_A when it enables the mirror. Instead, the destination mirror port is statically specified by the OverlayDestPort field of MCAST_MIRROR_CFG. During multicast replication, a mirror copy of the frame is synthesized and sent on OverlayDestPort, in addition to any normally forwarded copies of the frame sent on that egress port. This forwarding mode must be used when ports cannot be dedicated exclusively for mirroring.

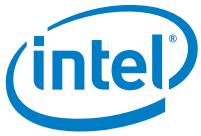
Both modes of mirror forwarding operate independently, although it is expected that only one would be used per mirror mechanism. Overlaid forwarding is enabled by setting OverlayEnabled to 1b in the CM_TX_MIRROR_DEST and MCAST_MIRROR_CFG registers. Explicit forwarding is enabled each time ExplicitDestMask is a non-zero.

Each mirror supports two optional features that are enabled by the SetMirror action:

- **Truncation** — If the MIR_TRUNC parameter is set to 1b, only the first segment (160 bytes) of the frame is forwarded. Egress modification rules might further reduce the frame length less than 160.
- **Priority Mapping** — If the MIR_MAP_PRI parameter is set to 1b, the overlay-forwarded mirror frame (if enabled) is scheduled with the traffic class OverlayTC, as configured in MCAST_MIRROR_CFG, not with the original traffic class of the source frame. The MIR_MAP_PRI parameter has no effect for explicitly forwarded mirror frames.

The following set of registers specify static mirroring configuration parameters:

- **CM_MIRROR_DEST[0..3]**
  - ExplicitDestMask (76 bits)
  - OverlayEnabled (1 bit)
- **CM_TX_MIRROR_SRC[0..3]**
  - SrcMask (76 bits)



- **MCAST_MIRROR_CFG[0..3]**
  - — ExplicitDestMask (76 bits)
  - — OverlayEnabled (1 bit)
  - — OverlayDestPort (7 bits)
  - — OverlayTC (4 bits)
  - — CanonicalSrcPort (7 bits)
- **MOD_TX_MIRROR_SRC**
  - — CanonicalSrcPort[0..3] (7 bits)

All fields sharing the same name must be set to consistent values.

In egress modification, the MIR_RX, MIR_TX, and MIR_NUM fields in its TCAM key identify mirror frames and enable microcode to format these frames appropriately. Most modify mapping tables are indexed by DST_PORT, which for Tx-mirrored frames is set to the mirror's CanonicalSrcPort. This causes the frame to be modified according to the rules applicable to the canonical Tx mirror source port, not the physical port to which the mirror frame is transmitted; in modify, the physical egress port is TX_PORT, and is expected to influence some aspects of the mirror frame's modification, such as ISL-tagging).

The memory use of overlay-mirrored frames is tracked in dedicated virtual queues for purposes of congestion management. Explicitly-mirrored frames are subject to the memory accounting and drop watermarks as normal frames. In either case, the mirror frames might be dropped due to watermark evaluations, so they might not always be reliably generated as specified by the SetMirror action.

5.17.9 MuxOutput

One of the primary purposes of the L2AR stage is to formulate the configurable components of the forward channel. Some of these fields are set indirectly by actions such as TransformDestMask (DMASK) and SetFlags (MOD_FLAGS). However, many fields, notably all 11.5 bytes of MOD_DATA, are specified directly with MuxOutput actions.

One additional MuxOutput set is defined for specifying the payload of the MAC table entry that can be optionally written back to the table.

All L2AR MuxOutput actions have types of either MCM (supporting masked constant assignment) or MCO (supporting a profile-specified constant override value), with the exception of the very simple (type M) DGLORT mux action. Since the DGLORT mux action operates on fields that can already be set in a highly configurable manner (using SetDestGlort and SetCpuCode actions), nothing more than a type M simple mux action is necessary. See Section 5.17.4 for a description of these action types.

Some of the output sets support a large number of profiles (96-128) to enable application of arbitrary constants on a per-port basis.

5.17.9.1 MOD_DATA Outputs

Table 5-78 shows the modified data outputs.

Table 5-78 L2AR MUX Output Configurations

Set Name	Type	N_{pro}	Output Field(s)	Width	N_{src}	Src	Source Field
W16A	MCM	128	MOD_DATA.W16A	16	8	0	SGLORT
						1	CSGLORT
						2	ACTION_DATA.W16A
						3	ALU1_Z
						4	ALU4_Z
W16B	M	0	MOD_DATA.W16B	16	8	0	DGLORT
						1	DGLORT | CpuCode
						2	ACTION_DATA.W16B
						3	ALU3_Z
						4	ALU6_Z

Table 5-78 L2AR MUX Output Configurations (Continued)

Set Name	Type	N_{pro}	Output Field(s)	Width	N_{src}	Src	Source Field
W16CDEF	MCO	32	MOD_DATA.W16C	16	8	0	ACTION_DATA.W16C
						1	ACTION_DATA.W16G
						2	ACTION_DATA.W16H
						3	DMASK[15:0]
						4	ALU2_Z
						5	ALU5_Z
			MOD_DATA.W16D	16	8	0	MA1_MAC[15:0]
						1	ACTION_DATA.W16D
						2	ACTION_DATA.W16G
						3	ACTION_DATA.W16H
						4	ALU3_Z
						5	ALU6_Z
						6	DMASK[31:16]
			MOD_DATA.W16E	16	8	0	MA1_MAC[31:16]
						1	ACTION_DATA.W16E
						2	ACTION_DATA.W16G
						3	ACTION_DATA.W16H
						4	ALU1_Z
						5	ALU4_Z
						6	DMASK[47:32]
			MOD_DATA.W16F	16	8	0	MA1_MAC[47:32]
						1	ACTION_DATA.W16F
						2	ACTION_DATA.W16G
						3	ACTION_DATA.W16H
						4	ALU2_Z
						5	ALU5_Z
						6	DMASK[63:48]

Table 5-78 L2AR MUX Output Configurations (Continued)

Set Name	Type	N_{pro}	Output Field(s)	Width	N_{src}	Src	Source Field
W8ABCDE	MCM	32	MOD_DATA.W8A	8	4	0	ACTION_DATA.W8A
						1	ALU1_Z[7:0]
						2	ALU2_Z[7:0]
						3	ALU3_Z[7:0]
			MOD_DATA.W8B	8	8	0	ACTION_DATA.W8B
						1	MA1_DATA
						2	ALU1_Z[15:8]
						3	ALU2_Z[15:8]
						4	ALU3_Z[15:8]
						5	POL1_TAG1[7:0]
			MOD_DATA.W8C	8	8	0	ACTION_DATA.W8C
						1	DMASK[71:64]
						2	ALU4_Z[7:0]
						3	ALU5_Z[7:0]
						4	ALU6_Z[7:0]
						5	CN_SRC_TX[1]
						6	POL2_TAG1[7:0]
			MOD_DATA.W8D	8	8		ACTION_DATA.W8D
							DMASK[79:72]
							ALU4_Z[15:8]
							ALU5_Z[15:8]
							ALU6_Z[15:8]
							POL3_TAG[7:0]
							ISL_USER
			MOD_DATA.W8E	8	8	0	CpuCode
						1	ACTION_DATA.W8E
						2	DropCode
						3	ALU4_Z[7:0]
						4	ALU5_Z[7:0]
						5	ALU6_Z[7:0]

Table 5-78 L2AR MUX Output Configurations (Continued)

Set Name	Type	N_{pro}	Output Field(s)	Width	N_{src}	Src	Source Field
W4	MCM	16	MOD_DATA.W4	4	16	0	ACTION_DATA.W8D[3:0]
						1	ACTION_DATA.W8D[7:4]
						2	ACTION_DATA.W8E[3:0]
						3	ACTION_DATA.W8E[7:4]
						4	ALU1_Z[3:0]
						5	ALU2_Z[3:0]
						6	ALU3_Z[3:0]
						7	MA1_TAG[11:8]
						8	POL1_TAG1[11:8]
						9	POL2_TAG1[11:8]
						10	POL3_TAG[11:8]

1. The values of these mux cases are not known by L2AR. If selected, these values are set downstream by the congestion notification sampling stage.

5.17.9.2 Named Forward Channel Outputs

Table 5-79 shows the other forward channel output fields:

Table 5-79 L2AR Forward Channel Output Fields

Set Name	Type	N_{pro}	Output Field(s)	Width	N_{src}	Src	Source Field
VID	MCO	32	L2_VID1	12	4	0	EVID1
						1	IVID1
						2	ALU3_Z[11:0]
			L2_VID2	12	4	0	EVID2
						1	IVID2
						2	ALU6_Z[11:0]
DMASK_IDX	MCO	32	DMASK_IDX	12	4	0	L2F_DMASK_IDX
						1	ACTION_DATA.W16H
						2	ALU3_Z

5.17.9.3 QoS

The MuxOutput_QOS action sets the profile number that is used for a Type-MCM transformation of the QOS fields.

Table 5-80 L2AR Profile Table (QOS_PROFILE_TABLE[0..31])

Field	Width	Description
Src_ISL_PRI	4	Selects mux source for ISL_PRI.
Src_L2_VPRI1	5	Selects mux source for L2_VPRI1.
Src_L3_PRI	4	Selects mux source for L3_PRI.
Src_L2_VPRI2	5	Selects mux source for L2_VPRI2.
Src_W4	5	Selects mux source for generic W4 field.
Mask	24	Mask to AND with the output post-muxing.
Value	24	Constant value to OR with the output post-masking and post-muxing.

Table 5-81 L2AR Source Channel Values

Src	Source Channel					Description
	ISL_PRI	L2_VPRI1	L2_VPRI2	W4	L3_PRI[1]	
0	ISL_PRI	L2_VPRI1	L2_VPRI2	W4	L3_PRI	Default.
1	POLICER_QOS_MAP1_FIRST					Assign from four-bit POLICER_QOS_-MAP1_FIRST output (first-order mapping).
2	POLICER_QOS_MAP1_SECOND					Assign from four-bit POLICER_QOS_-MAP1_SECOND output (second-order mapping).
3	POLICER_QOS_MAP2_FIRST					Assign from eight-bit POLICER_QOS_-MAP2_1 output (first-order mapping). For all but L3_PRI, top four bits of the mapped value are ignored.
4	POLICER_QOS_MAP2_SECOND					Assign from eight-bit POLICER_QOS_-MAP2_2 output (second-order mapping). For all but L3_PRI, top four bits of the mapped value are ignored.
5	QOS.W4					Map any QoS field from generic QOS.W4 (BPRI, MPLS EXP, etc.
6	ACTION_DATA.W8F[3:0]					Assign from bottom four bits for all output fields. L3_PRI is assigned the all eight bits of W8F.
7	ACTION_DATA.W8F[7:4]					Assign from top four bits. Undefined for L3_-PRI.

Table 5-81 L2AR Source Channel Values (Continued)

Src	Source Channel					Description
	ISL_PRI	L2_VPRI1	L2_VPRI2	W4	L3_PRI[1]	
8	POL1_TAG1[7:4]				POL1_TAG1[5:0]	Policer entry TAG assignment cases.
9	POL1_TAG2[7:4]				POL1_TAG2[5:0]	
10	POL2_TAG1[7:4]				POL2_TAG1[5:0]	
11	POL2_TAG2[7:4]				POL2_TAG2[5:0]	
12	POL3_TAG[7:4]				POL3_TAG[5:0]	
13	POL1_TAG1[9:6]	POL1_TAG1[3:0]			—	
14	POL1_TAG2[9:6]	POL1_TAG2[3:0]			—	
15	POL2_TAG1[9:6]	POL2_TAG1[3:0]			—	
16	POL2_TAG2[9:6]	POL2_TAG2[3:0]			—	
17	POL3_TAG[9:6]	POL3_TAG[3:0]			—	

1. All assignments to L3_PRI from fields of widths less than eight bits are MSB-aligned. For example, POL1_TAG1[5:0] sets L3_PRI[7:2]. In these cases, unassigned low-order bits are left unmodified unless explicitly assigned by a (Mask,Value) profile constant.

5.17.9.4 MAC Table Write-Back

MAC write-back is used to write learned addresses back into the L2 lookup table.

Table 5-82 L2AR MAC Table Write-Back

Set Name	Type	N_{pro}	Output Field(s)	Width	N_{src}	Src	Source Field
MA_WRITEBACK	SC	32	MA_WRITEBACK.PREC	2	0	0	Constant value.
	MCM	32	MA_WRITEBACK.GloRT	16	4	0	MA2_GLORT
						1	CSGLORT
						2	ISL_SGLORT
						3	ALU1_Z
			MA_WRITEBACK.TAG	12	4	0	MA2_TAG
						1	ACTION_DATA.W16A[11:0]
						2	{ACTION_DATA.W16A[11:8], MA2_TAG[7:0]}
			MA_WRITEBACK.DATA	8	4	0	MA2_DATA
						1	ACTION_DATA.W8B
						2	ACTION_DATA.W8C
						3	ACTION_DATA.W16A[7:0]

Note: The combination of {MA2_TAG[11:8],MA2_DATA} can be used as a 12-bit tunnel ID. For this purpose, the ACTION_DATA.W16A[11:0] is mapped to {MA2_TAG[11:8],MA2_DATA} at learning and the {MOD_DATA.W4,MOD_DATA.W8B} is mapped to recovered {MA1_TAG[11:8],MA1_DATA} for forwarding.

5.17.9.5 Statistics Index Channels

A number of frame properties are sent from L2AR to the statistics counters where they are categorized and assigned to specific counter index values for counting. Some of these properties are sent over fixed-function channels, such as the Rx ingress port number, the frame's byte length, and the tail error status. Others are selected in a configurable manner by L2AR:

- **STATS_FLAGS** — Set by the L2AR SetFlags action, split off from ACTION_FLAGS.
- **STATS_IDX*** — Muxed from channels available at the input to L2AR. Available as Rx Per-Index and CounterNum source values for statistics counter index selection.

The mux definitions for the statistics index channels STATS_IDX{5A,5B,5C,12A,12B,16A,16B} are listed in Table 5-83.

Table 5-83 L2AR MUX Definitions for Statistics

Set Name	Type	N_{pro}	Output Field(s)	Width	N_{src}	Src	Source Field
STATS_IDX5AB	MCO	32	RX_STATS.IDX5A RX_STATS.IDX5B	5	16	0	QOS.ISL_PRI
						1	QOS.L2_VPRI1
						2	QOS.L2_VPRI2
						3	QOS.W4
						4	L2F_ITAG1[4:0]
						5	L2F_ITAG2[4:0]
						6	L2F_ETAG1[4:0]
						7	L2F_ETAG2[4:0]
						8	MA1_TAG[4:0]
						9	MA2_TAG[4:0]
						10	ACTION_DATA.W8E[4:0]
						11	ALU1_Z[4:0]
						12	DropCode[4:0] (IDX5A) ALU4_Z[4:0] (IDX5B)
						13	QOS.TC (as mapped by CM_TC_MAP in congestion management).
						14	QOS.RXMP (as mapped by CM_RXMP_MAP in congestion management).
STATS_IDX5C	MCO	32	RX_STATS.IDX5C	5	16	0..14	Same as for STATS_IDX5B.

Table 5-83 L2AR MUX Definitions for Statistics (Continued)

Set Name	Type	N_{pro}	Output Field(s)	Width	N_{src}	Src	Source Field
STATS_IDX12A STATS_IDX12B	MCO	32	RX_STATS.IDX12A RX_STATS.IDX12B	12	32	0	IVID1
						1	IVID2
						2	EVID1
						3	EVID2
						4	L2F_ITAG1
						5	L2F_ITAG2
						6	L2F_ETAG1
						7	L2F_ETAG2
						8	MA1_TAG
						9	MA2_TAG
						10	ACTION_DATA.W8E
						11	POL1_IDX
						12	POL2_IDX
						13	ALU2_Z[11:0]
						14	ALU3_Z[11:0]
						15	ALU6_Z[11:0]
						16	POL1_TAG1
						17	POL2_TAG1
						18	POL3_TAG
STATS_IDX16A STATS_IDX16B	MCO	16	RX_STATS.IDX16A RX_STATS.IDX16B	16	16	0	ACTION_DATA.W16A
						1	ACTION_DATA.W16B
						2	ACTION_DATA.W16C
						3	ACTION_DATA.W16G
						4	ACTION_DATA.W16H
						5	ALU1_Z
						6	ALU4_Z
						7	ALU5_Z
						8	MOD_DATA.W16A (Final SGLORT).
						9	MOD_DATA.W16B (Final DGLORT).

Note: All three 5-bit IDX5{A,B,C} indices have the same set of mux input channels. However, IDX5A and IDX5B share a single profile, while IDX5C has its own. This makes MuxOutput_STATS_IDX5C the best action to use for direct index assignments from L2AR rules.

5.18 Congestion Management

The data produced by L2AR represents, in part, the frame's desired egress forwarding distribution and scheduling policy. The Congestion Management (CM) stage is then responsible for implementing drop and sample decisions that are intended to protect the switch, the network, and particular classes of traffic from the undesirable consequences of network congestion.

The FM5000/FM6000 provides the following mechanisms for managing congestion:

Accounting of memory use by Rx and Tx memory partitions

A particular frame belongs to exactly one Rx memory partition (RXMP) and one Tx memory partition (TXMP). Its membership is mapped from QOS.ISL_PRI. All drop, pause, and sampling decisions are applied within the domain of these independent memory partitions.

Private and drop watermarks defined for each ingress port (Rx) per RXMP

As frames arriving from a particular (Rx and RXMP) pair begin to consume too much of the shared frame memory, this mechanism enables subsequent frames to be head-dropped before they are queued. Conversely, any (Rx and RXMP) consuming less than its private watermark enables is guaranteed space in the frame memory regardless of the ingress congestion due to other ports.

Global per-RXMP drop watermarks

These watermarks are applied to any frames whose (Rx and RXMP) usage is above their respective private watermarks.

Global drop watermark

This watermark is applied to any frame.

Per-port, per-TXMP soft drop watermarks

These watermarks probabilistically drop frames queued to (Tx and TXMP) destinations that are consuming too much of the shared memory. A probabilistic mechanism is employed on Tx to avoid reserving excessive amounts of memory to maintain fair egress scheduling.

Generation of PAUSE frames in response to rising Rx memory consumption

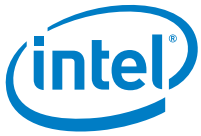
Two styles of pause frames can be generated, one for standard IEEE 802.3 per-port pause, the other for IEEE 802.1Qbb PFC. Port-based pause frames are generated based on Xon/Xoff watermarks referenced to total Rx port and total shared memory usage. Class-based pause frames are controlled by Xon/Xoff watermarks referenced to per-(Rx and RXMP) and total RXMP memory levels. As long as any Xoff watermark is exceeded, the corresponding port's pause frame is repeated at some timed resend interval configured per port.

Periodic generation of PAUSE frames for link partner rate limiting

Pause pacing is supported on any port or pause class. When enabled, pairs of PAUSE-ON/OFF frames are generated periodically to limit the rate at which the link partner can transmit on the given port or pause class.

Tx per-port and per-class reaction to ingress PAUSE frames

Pause frames identified by the parser might provide the egress scheduler with timed per-port or per-TC pause times. The mechanism supports Xon messages with pause payload-specified IEEE-standard time values, as well as immediate-response Xoff messages. Each pause frame can specify these parameters over a vector of up to eight traffic classes. The configurability of the parser provides support for standard IEEE 802.3 PAUSE frames, IEEE 802.1Qbb PFC PAUSE, as well as other user-defined frame formats.



Periodic frame sampling per-(TX,TXMP) queue

Running byte counters are maintained per (Tx and TXMP) queue. Every configured interval, up to a minimum period of 1 µs, frames might be mirrored to a particular GloRT destination. If the destination is a companion device, the mirrored frames can be reformatted as congestion notification sample frames and returned to the FM5000/FM6000 to be forwarded back to the original source. This mechanism can be used to generate IEEE 802.1Qau QCN messages.

Per-Tx port, per-TXMP watermark-based interrupt notification

Each (Tx and TXMP) queue supports high/low watermarks tied to interrupt notification bits. When the corresponding queue level is above a high watermark, an IP high-limit bit corresponding to that queue is set. Once the level is below a low watermark, the IP low-limit bit is set. The interrupt bits are stored in an 80-bit wide register. This mechanism might be used by an external device to generate virtual output queuing Xon/Xoff messages.

Mirror accounting and tracking

Up to four mirror commands can be attached to a single frame. The CM module keeps track of the memory use for each mirror in a manner similar to what it does for Tx queues. For example, through a per-mirror, per-TXMP usage and soft drop watermarks. The mirror event is dropped by canceling the mirror command.

5.18.1 Linkage to the Frame Processing Pipeline

The CM stage is the last stage of ingress frame processing before frames are queued, multicast-replicated, and scheduled for transmission. It immediately follows the L2AR stage and has the last opportunity to drop frames before they are stored in the shared frame memory.

The CM stage primarily operates on the following channels from L2AR:

- **ISL_PRI** — This 4-bit value represents the frame's comprehensive QoS priority for purposes of queue mapping and watermark drop evaluations. In CM, three new QoS fields are mapped from ISL_PRI that factor into these forwarding policies: a Traffic Class (TC) number and two shared memory partition numbers, one for ingress (RXSMP) and one for egress (TXMP). These parameters are described in detail as follows.
- **DMASK** — Drop and sampling decisions made by the CM are implemented as transformations of the frame's destination port mask. For example, if the frame memory is full and no more frames can be allowed into the switch, the DMASK is zeroed. If the frame is to be sampled for congestion notification purposes, the DMASK is augmented with a bit corresponding to the appropriate sampling destination.
- **CM_FLAGS** — Compared to earlier stages in the FPP, the CM stage includes a large amount of highly specific fixed-function logic. A set of flag bits set by L2AR provide per-frame control over these operations.

  Note: Currently this only includes CN_SampleEligible, bit 30 of ACTION_FLAGS.

Figure 5-38 shows a high-level view of the sequence of QoS mapping, sample, and drop operations applied by the CM stage:

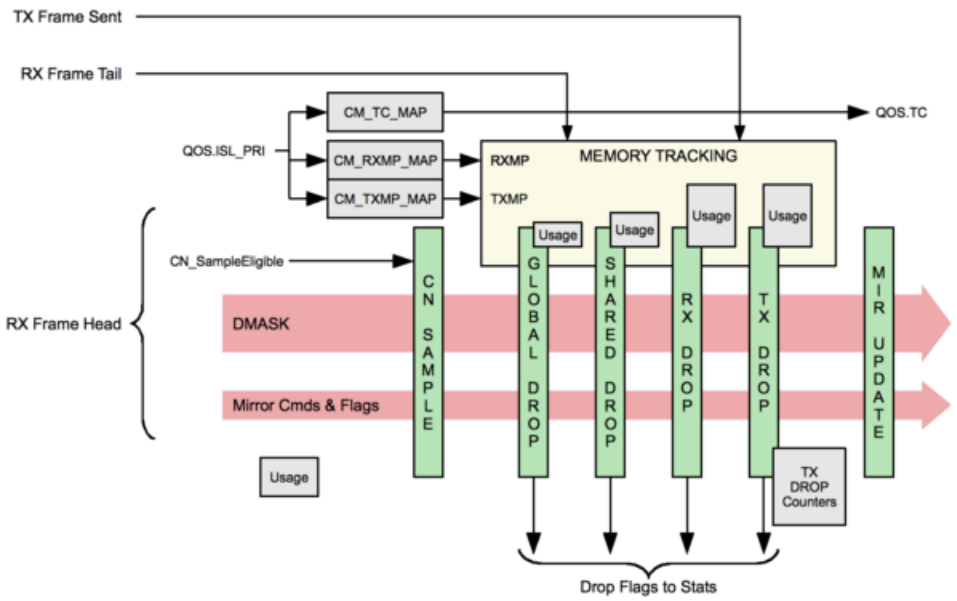


Figure 5-38 Sequence of QoS Mapping

The CM_TC_MAP[0..15], CM_RXMP_MAP[0..7], and CM_TXMP_MAP[0..11] registers define the mapping from the FM5000/FM6000's 16 ISL priorities (QOS.ISL_PRI) to 12 TCs, and 12 Rx and Tx memory partitions (RXMP and TXMP). The TC influences egress scheduling policy and is passed on to the scheduler. The RXMP and TXMP quantities are used locally for memory usage tracking, drop watermark evaluation, and flow-control. All three QoS values can be propagated to egress modification with the appropriate QoS mux selection in L2AR.

In addition to the forwarding-related frame processing transformations applied to each frame, the CM stage implements the following other background functions related to memory management and monitoring:

- **Flow Control** — Pause frame handling and generation.
- **Egress Traffic Shaping** — For architectural reasons, the egress scheduler's traffic shaping is configured and controlled by the CM stage.
- **Interrupt-based TX queue monitoring** — Interrupt notification based on (Tx and TXMP) queue levels.

5.18.2 Memory Management

The FM5000/FM6000 fundamentally is a shared memory switch. Thus its central frame memory is accessible to all ports on a first-come, first-served basis. To control undesirable crowding of this memory by different QoS TCs, mechanisms are provided for dividing the memory into independently managed partitions. This limits the exposure of one QoS class to congestion experienced by another.

Two partitions are supported, one associated with each frame's ingress port (Rx), the other with each frame's egress port(s) (Tx). Eight Rx and 12 Tx memory partitions are supported, referred to respectively as RXMP and TXMP. These partitions are overlaid in the sense that each track independently measures memory consumption.

For congestion management purposes, a port-partition pair, (Rx and RXMP) or (Tx and TXMP), defines a frame's ingress or egress queue. A frame's RXMP and TXMP values are mapped independently from its ISL_PRI, although it is expected that the RXMP map is defined over independent sets of TXMPs. For example, a frame’s RXMP could be mapped uniquely from its TXMP. The TXMP map is also expected to exactly match the TC map, but for flexibility purposes is not unified with CM_TC_MAP.

Each memory partition tracks usage by port and by total usage. The Rx memory partitions additionally support a per-port private watermark, which allocates a protected quantity of memory per Rx port. As long as frames from a particular (Rx and RXMP) queue consume no more than their allocated private memory, they are guaranteed to never be dropped.

Note: Exceptions might occur when the system of CM watermarks is mis-configured such that the total absolute frame memory is unexpectedly exceeded.

Also, global memory usage is tracked and a global watermark is used to drop any frame above this level. The watermark must be set at the top of memory minus worst case in-flight segments expected. If the system runs out of segments, the MAC is back pressured and overflow conditions are reported by the EPL.

A number of drop, flow-control, and notification watermarks are defined over the memory levels tracked by this system. All drop watermarks are evaluated at the head of each frame, before the full length of the frame is known. Thus a skid-pad memory buffer must be reserved above all drop watermarks to allow for in-flight frames.

Only frames that are not dropped are queued and credited to their corresponding memory usage level(s). The frame is accounted for on the receive side (CM_RX_xxxx_USAGE) and on the transmit side (CM_TX_xxxx_USAGE) as segments are allocated on ingress or freed on egress. A multicast frame is credited to its ingress port's level once on the receive side and to every possible destination on the transmit side. On egress, frames are credited in an L2-replicated sense. Thus, regardless of whether one or 4000 L3 copies of a frame are sent to a particular Tx port, the level associated with that Tx port reflects only the single source frame that is replicated.

Memory accounting is reported in the CM_xxx_USAGE registers. The units of accounting are frames and 160-byte segments for global and Rx usage counters, and 480-byte segments for Tx counters. They are incremented for frames that are scheduled for transmission (forwarded somewhere), and are decremented once the frame has been transmitted or discarded by the scheduler (due to timeout or port down).

The global usage registers and receive usage registers are decremented only when the segment or frame is freed. For example, when successfully transmitted or discarded on all possible transmit queues. The transmit usage registers are decremented upon transmission of each 480-byte segment. Frames that are discarded before being queued (such as leave the CM stage with a DMASK of zero) are not counted as they are not stored in memory.

The Tx port numbers 76..79 are virtual ports used only for mirror frames. Each of the FM5000/FM6000's four mirror mechanisms is mapped to one of these virtual ports when it is configured to use overlaid forwarding. For more information about overlaid forwarding, see Section 5.17.8, "SetMirror".

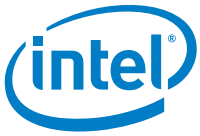
For congestion management purposes, each mirror mechanism has two queues on each virtual port corresponding to the two MIR_MAP_PRI cases. In all Tx CM registers indexed by Tx port number and a QoS value (either ISL_PRI or TXMP), the QoS value is taken from the corresponding mirror's MIR_MAP_PRI value.

For example, for mirror ports 76..79, the CM_PORT_TXMP_USAGE[TX,TXMP] register has the following definition:

- **CM_PORT_TXMP_USAGE[76..79,0]** — Memory usage of mirror 0..3 frames with MIR_MAP_PRI[MIRROR_NUM]=0b.
- **CM_PORT_TXMP_USAGE[76..79,1]** — Memory usage of mirror 0..3 frames with MIR_MAP_PRI[MIRROR_NUM]=1b.
- **CM_PORT_TXMP_USAGE[76..79,2..11]** — Unused.

Figure 5-39 Memory Management Usage Tracking

Register	Description	Tracks
CM_GLOBAL_USAGE	Tracks global memory usage, regardless of memory partitioning. This represents the sum over all RXMPs.	Number of frames and number of 160-byte segments.
CM_RXMP_USAGE[0..11]	Tracks the total memory usage of each RXMP. Used for the RXMP-based Tx soft drop mechanism.	Number of 480-byte segments.
CM_SHARED_RXMP_USAGE[0..11]	Tracks usage of shared (non-private) memory per RXMP. A segment or frame is included in this counter only if its source port has exceeded its corresponding private watermark. If the egress transmission of a particular frame causes its ingress source port to drop below its private watermark, only the segments required to bring the port's usage below the watermark is decremented from this counter.	Number of frames and number of 160-byte segments.
CM_PORT_RXMP_USAGE[0..75,0..11]	Tracks per-port, per-RXMP memory usage. If multiple ISL_PRIs map into the same RXMP, memory usage for frames in these priorities are tracked together.	Number of frames and number of 160-byte segments.
CM_GLOBAL_TXMP_USAGE[0..11]	Tracks total memory usage per TXMP.	Number of 480-byte segments.
CM_PORT_TXMP_USAGE[0..79,0..11]	Tracks per-port, per-TXMP memory usage. If multiple ISL_PRIs map into the same TXMP, memory usage for frames in these priorities are tracked together.	Number of 480-byte segments.



5.18.3 Watermarks

All watermarks listed in Table 5-84 are evaluated with greater-or-equal-to semantics. Thus, a watermark value of zero evaluates true for all frames handled by the CM stage regardless of memory use.

Table 5-84 Memory Management Watermarks

Register	Description	Limits
CM_GLOBAL_WM	Defines an upper-bound limit to memory use for new frames. This limit is typically set to the maximum memory minus the maximum frame size times the number of active ports.	Total frames and 160-byte segments.
CM_SHARED_RXMP_WM[0..15]	Defines the upper-bound limit of shared RXMP memory per switch priority. The watermark is mapped from the frame's ISL priority and is compared against the corresponding CM_SHARED_RXMP_USAGE level. This limit is typically set to the desired RXMP size minus an allowance for in-flight frames and all per-port private allocations.	Per-RXMP frames and 160-byte segments.
CM_PORT_RXMP_PRIVATE_WM[0..75,0..11]	Defines the size of each Rx port's private memory per RXMP. Indexed by (Rx and RXMP).	Per-(Rx and RXMP) frames and 160-byte segments.
CM_PORT_RXMP_HOG_WM[0..75,0..15]	Defines the maximum CM_PORT_RXMP_USAGE beyond which frames belonging to the corresponding (RX,RXMP) queue are dropped. Indexed by (RX,ISL_PRI) so different ISL priorities belonging to the same RXMP are limited to different levels of memory consumption.	Per-(RX,ISL_PRI) frames and 160-byte segments.
CM_PORT_TXMP_PRIVATE_WM[0..79,0..15] CM_PORT_TXMP_HOG_WM[0..79,0..15]	Defines watermarks on (Tx and TXMP) egress queues, indexed by (Tx and ISL_PRI). Tx dropping is disabled until PrivateSegmentLimit is exceeded, at which point the queue becomes eligible for probabilistic soft dropping. All frames are dropped when the queue's usage exceeds HogSegmentLimit. Two soft dropping mechanisms can be enabled by setting the SoftDropOnPrivate or SoftDropOnRxmpFree configuration bits defined in CM_PORT_TXMP_PRIVATE. If SoftDropOnPrivate is set to 1b, frames are dropped with 50% probability once the CM_PORT_TXMP_USAGE level exceeds PrivateSegmentLimit. If SoftDropOnRxmp is set to 1b, frames are dropped with either 50% or 100% probability, depending on the proportion of free RXMP_USAGE space the Tx queue's CM_PORT_TXMP_USAGE level consumes. These mechanisms are described in more detail in the sections that follow. If the frame's (Rx and RXMP) usage is below its Rx private watermark, this drop mechanism is disabled entirely and the frame is guaranteed to be queued. Indexed by (Tx and ISL_PRI) for ports 0..75. Indexed by (76+MIR_NUM, MIR_MAP_PRI[MIR_NUM]) for the mirror overlay ports 76..79.	Per-(TX,ISL_PRI) 480-byte segments.
CM_RXMP_SOFT_DROP_WM[0..15].{SoftDropSegmentLimit, SoftDropSegmentLimitJitterBits, HogSegmentLimit}	Configures Tx soft dropping based on RXMP usage. Each time the total RXMP usage exceeds SoftDropSegmentLimit plus a random jitter term, Tx soft dropping is enabled such that the Tx ports consuming the most memory are penalized most. Frames are dropped with 100% probability if the RXMP usage exceeds HogSegmentLimit. These watermarks are mapped by ISL_PRI and compared against the corresponding CM_RXMP_USAGE level (measured in 480-byte segments).	Per-ISL_PRI 480-byte segments.

Table 5-84 Memory Management Watermarks (Continued)

Register	Description	Limits
CM_PORT_TXMP_IP_WM[0..79,0..11].{Max, Min}	Defines per-(TX,TXMP) maximum and minimum watermarks that control when bits are set in the CM_PORT_TXMP_ABOVE_IP and CM_PORT_TXMP_BELOW_IP interrupt registers.	Per-(Tx and TXMP) 480-byte segments.
CM_PORT_TXMP_SAMPLE_WM[0..79,0..11]	Defines per-(Tx and TXMP) sampling period for frame sampling.	Per-(Tx and TXMP) 480-byte segments.
CM_GLOBAL_TXMP_SAMPLE_CMD[0..11]	Defines per-TXMP command when sampling is required.	N/A

All memory management registers and watermarks can be changed by software at any time while there is traffic being switched through the fabric, with the exception of the following watermark registers:

- **CM_PORT_RXMP_PRIVATE_WM**
- **CM_RXMP_MAP**
- **CM_TXMP_MAP**
- **CM_TC_MAP**

Changes to these registers, while frames are in flight through the switch, might result in unpredictable frame dropping and/or persistent memory loss.

5.18.4 Rx Watermark Evaluation

Figure 5-40 through Figure 5-42 show the different receive watermarks and usage counters, per-port-per-RXMP, per-RXMP, global.

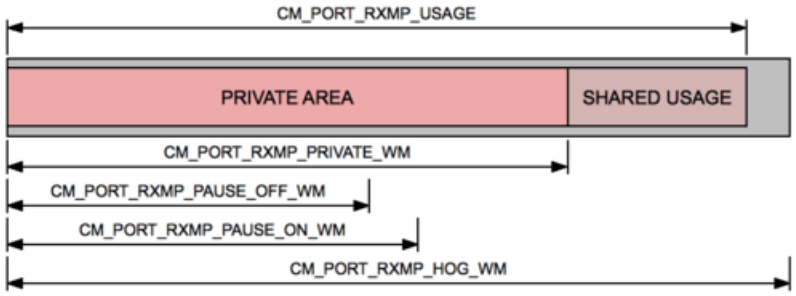


Figure 5-40 Receive Usage and Watermarks

Figure 5-41 shows the watermarks and usage counters per RXMP.

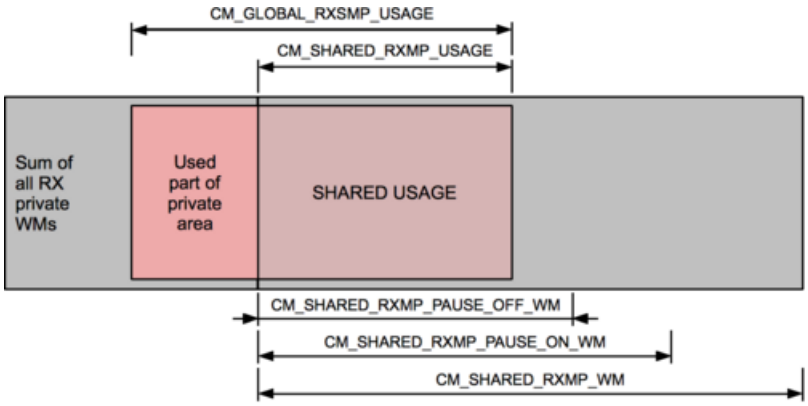


Figure 5-41 SMP Usage and Watermarks

Figure 5-42 shows the overall watermarks and usage for the entire memory.

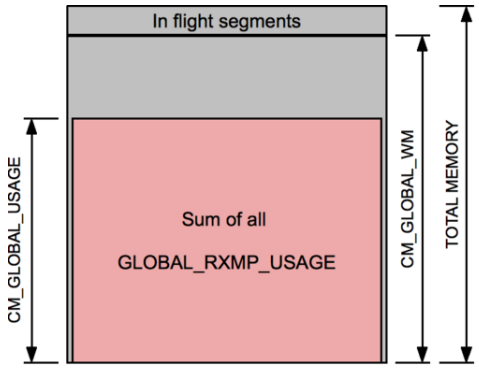


Figure 5-42 SMP Usage and Watermarks

For reception, a frame gets dropped if there is no private space available and the frame exceeds the hog or SMP shared or global watermarks:

```
overRxPrivate = CM_PORT_RXMP_USAGE[RX,RXMP].SegmentCount ≥
                    CM_PORT_RXMP_PRIVATE_WM[RX,RXMP].SegmentLimit |
                CM_PORT_RXMP_USAGE[RX,RXMP].FrameCount ≥
                    CM_PORT_RXMP_PRIVATE_WM[RX,RXMP].FrameLimit

overRxPort    = CM_PORT_RXMP_USAGE[RX,RXMP].SegmentCount ≥
                    CM_PORT_RXMP_HOG_WM[RX,ISL_PRI].SegmentLimit |
                CM_PORT_RXMP_USAGE[RX,RXMP].FrameCount ≥
                    CM_PORT_RXMP_HOG_WM[RX,ISL_PRI].FrameLimit

overRxShared  = CM_SHARED_RXMP_USAGE[RXMP].SegmentCount ≥
                    CM_SHARED_RXMP_WM[ISL_PRI].SegmentLimit |
                CM_SHARED_RXMP_USAGE[RXMP].FrameCount ≥
                    CM_SHARED_RXMP_WM[ISL_PRI].FrameLimit

overGlobal    = CM_GLOBAL_USAGE.FrameCount ≥
                    CM_GLOBAL_WM.FrameLimit |
                CM_GLOBAL_USAGE.SegmentCount ≥
                    CM_GLOBAL_WM.SegmentLimit

outOfMemory   = [Set when the Scheduler runs out of segments. Never
                 happens with proper CM_GLOBAL_WM configuration.]

if ( overRxPort | (overRxPrivate & overRxShared) | overGlobal | outOfMemory )
   // Cancel sending the frame anywhere
   dmask = 0
```

5.18.5 Tx Watermark Evaluation

Figure 5-43 figure shows the different transmit watermarks and usage counters per port per TXMP.

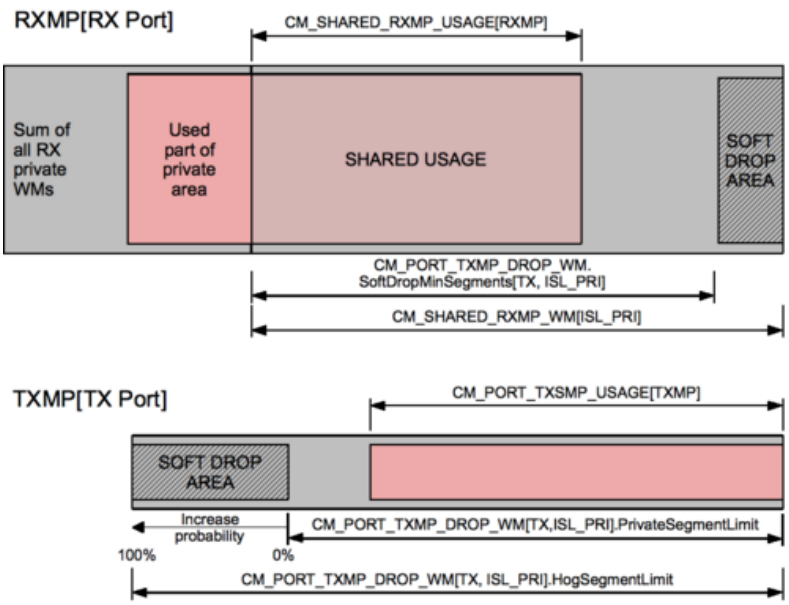
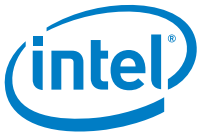


Figure 5-43 Transmit Usage and Watermarks



For transmission, a frame cannot be forwarded to a particular port, even if it should be according to the destination mask, if the transmit queue is consuming too much memory. A (private and hog) watermark pair is defined per (Tx and ISL_PRI) queue, which controls this filtering. As long as the queue's memory usage (CM_PORT_TXMP_USAGE[TX,TXMP]) is below the private watermark (CM_PORT_TXMP_PRIVATE_WM[TX,ISL_PRI].PrivateSegmentLimit), the frame is queued to the specified destination. If memory use is above the hog watermark, the frame is not queued. Between the private and hog watermarks, two soft drop evaluations are performed to determine one of three drop probabilities (0%, 50%, or 100%).

The drop probability is calculated as follows:

Table 5-85 TX Drop Probability Calculation

When SoftDropOnPrivate=1b:

Regime	Drop Probability
Private > usage	0%
Hog > usage ≥ Private	50%
usage ≥ Hog	100%

When SoftDropOnRxmpFree=1b:

Regime	Drop Probability
Private > usage	0%
Hog ≥ free/2 > usage ≥ Private	50%
Hog ≥ free > usage ≥ free/2	100%
Hog > usage ≥ free	100%
usage ≥ Hog	100%

When both mechanisms are enabled, the larger of the two drop probabilities is applied.

The RXMP-based free parameter is a measure of the free switch-wide memory available to the (Tx and TXMP) queue that it might be sharing among other ports and other TXMPs:

```
free = CM_RXMP_SOFT_DROP_WM[ISL_PRI] - CM_RXMP_USAGE[RXMP]
```

The RXMP-based soft drop probability is further guarded by a CM_RXMP_SOFT_DROP_WM watermark comparison against the CM_RXMP_USAGE:

```
RxmpSoftDropEnabled = CM_RXMP_USAGE[RXMP] ≥ CM_RXMP_SOFT_DROP_WM[ISL_PRI] + RandomJitter
```

If RxmpSoftDropEnabled evaluates false, the RXMP-based soft drop probability remains at 0%. RandomJitter is an additive random term between 0 and 7 bits wide, configured by CM_RXMP_SOFT_DROP_WM[ISL_PRI].SoftDropSegmentLimitJitterBits.

When the RXMP-based soft drop mechanism is enabled, it is expected that the RXMP and TXMP maps are defined such that the RXMPs are supersets of TXMPs. That is:

$$RXMP\_MAP(isl\_pri_1) == RXMP\_MAP(isl\_pri_2) \Longrightarrow TXMP\_MAP(isl\_pri_1) == TXMP\_MAP(isl\_pri_2)$$

The Tx drop watermark evaluation is made more complex by its interaction with mirroring. If a frame is flagged with one or more mirror commands, the CM has to account for the mirrors' overlay-forwarded frames in special virtual ports (numbered 76..79), referred to as mirror overlay ports. If the mirror overlay ports fail their Tx drop evaluations, a post-filtering stage must disable any Tx-mirroring that is no longer applicable.

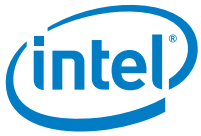
The process followed is:

- From the CM stage on, the DMASK is augmented with four extra bits corresponding to the overlay-forwarded frames queued due to each mirror mechanism.

- The memory consumed by the 76..79 mirror overlay ports is then tracked in terms of two virtual queues per mirror, one for frames overlay-mirrored with MIR_MAP_PRI=0b, the other for overlay-mirrored frames with MIR_MAP_PRI=1b. This provides a mechanism to independently control the memory consumption (by Tx watermarks) of these two classes of mirror frames.
- Tx drop watermark evaluations are then performed over the 80-bit augmented DMASK. The DMASK is filtered based on the memory usage of each destination queue (or virtual queue).
- All DMASK destinations filtered in this manner due to the Tx drop watermarks are counted in the per-queue 64-bit CM_PORT_TXMP_DROP_COUNT counters.

The exact pseudo-code is:

```
// Augment DMASK
dmask1[75:0] = dmask
foreach M in (0..3)
    if ( (MIR_RX[M] | MIR_TX[M]) & CM_MIRROR_DEST[M].OverlayEnabled )
        dmask1[76+M] = 1

// Determine status of RXMP-based soft dropping
rxmpSoftDropEnabled =
    CM_RXMP_USAGE[RXMP].SegmentCount ≥
     CM_RXMP_SOFT_DROP_WM[ISL_PRI].SoftDropSegmentLimit +
     random(CM_RXMP_SOFT_DROP_WM[ISL_PRI].SoftDropSegmentLimitJitterBits)

// Soft drop becomes more aggressive as remaining RXMP free space drops
rxmpFree = CM_RXMP_SOFT_DROP_WM[ISL_PRI].HogSegmentLimit -
           CM_RXMP_USAGE[RXMP].SegmentCount

// Evaluate TX Drop Watermarks
foreach TX in (0..79)
    txDrop[TX] = 0

    if ( dmask1[TX] == 1 & overRxPrivate )
        // Per-(TX,ISL_PRI) drop watermarks
        overTxPrivate = CM_PORT_TXMP_USAGE[TX,TXMP] ≥
                        CM_PORT_TXMP_PRIVATE_WM[TX,ISL_PRI]
        overTxHog     = CM_PORT_TXMP_USAGE[TX,TXMP] ≥
                        CM_PORT_TXMP_HOG_WM[TX,ISL_PRI]

        if ( overTxHog )
            txDrop[TX] = 1

        else if ( overTxPrivate )
            if ( SoftDropOnRxmpFree==1 & rxmpSoftDropEnabled )
                // Soft drop based on total RXMP usage
                if ( CM_PORT_TXMP_USAGE[TX,TXMP] ≥ rxmpFree )
                    txDrop[TX] = 1
                else if ( CM_PORT_TXMP_USAGE[TX,TXMP] ≥ rxmpFree/2 )
                    txDrop[TX] = random(1)

                else if ( SoftDropOnPrivate==1 )
                    // Soft drop based only on TX Private Watermark
                    txDrop[TX] = random(1)

    // Count frame if dropped
    if ( txDrop[TX] )
        CM_PORT_TXMP_DROP_COUNT[TX]++

// Apply TX drop decisions
dmask[79:0] = dmask1[79:0] & ~txDrop[79:0]
```



In the previous pseudo-code, TXMP is mapped from CM_TXMP_MAP[QOS.ISL_PRI] and RXMP is mapped from CM_RXMP_MAP[QOS.ISL_PRI]. The random(N) function returns an N-bit random value uniformly distributed over 0 and 2^N-1. All TXMP or ISL_PRI indexing above for the mirror overlay ports 76..79 actually use the MIR_MAP_PRI[M] setting for the mirror; so the usage or watermark index is either zero or one depending on MIR_MAP_PRI[M].

5.18.6 Update Mirror Commands

Once the 80-bit augmented DMASK has been appropriately filtered due to the Tx drop watermarks, the DMASK and mirror commands are updated to cancel any mirroring that is no longer applicable:

- Disable mirroring if any Rx watermark (HOG/RXMP/GLOBAL) is exceeded or if the shared memory runs out of free segments.
- Disable mirroring if the destination mirror ports failed their drop watermark evaluations. This applies to either the mirror-overlay or explicit port destinations, depending on the mirror forwarding mode.
- Disable Tx-mirroring if the final DMASK does not include Tx mirrored port(s). The pseudo-code is:

```
foreach M in {0..3}

overlayEnabled           = CM_MIRROR_DEST[M].OverlayEnabled
explicitInitiallyEnabled = (MIR_TX[M] | MIR_RX[M])
explicitEnabled          = explicitInitiallyEnabled

// Cancel overlay mirroring if the virtual queue watermark is exceeded
// or if the global watermark is exceeded
if ( txDrop[76+M] == 1 | overGlobal | outOfMemory)
    overlayEnabled = 0
    dmask[76+M] = 0

// Cancel explicit mirroring if the explicit mirror ports were dropped
( if dmask[75:0] & CM_MIRROR_DEST[M].ExplicitDestMask == 0 )
    explicitEnabled = 0


if ( !overlayEnabled && !explicitEnabled )
    MIR_RX[M] = 0
    MIR_TX[M] = 0

// Cancel TX mirroring if final DMASK doesn't include the TX mirrored set
if ( MIR_TX[M] == 1 && (dmask & CM_TX_MIRROR_SRC[M]) == 0 )
    MIR_TX[M] = 0

// Make final selection between RX or TX mirroring; TX has precedence
if ( MIR_TX[M] == 1 )
    MIR_RX[M] = 0

// Cancel mirroring if neither RX nor TX mirroring remains enabled
if ( MIR_RX[M] == 0 && MIR_TX[M] == 0 )
    dmask[76+M] = 0
    if ( explicitInitiallyEnabled )
        dmask[75:0] = dmask[75:0] & !CM_MIRROR_DEST[M].ExplicitDestMask
```

Under all circumstances, if the final 80-bit augmented DMASK is zero, the frame is dropped and ingress is not consume any shared memory resources.

5.18.7 Pause Frame Reception

The FM5000/FM6000 can be configured to respond to a variety of pause frames received from its link partners. The handling of these frames in the FPP proceeds as follows:

1. The parser identifies the frame as one of two pause types:
   - Port-based: One pause time is applied to all of the port's TCs.
   - Class-based: A unique pause time is applied to each TC.
2. Parser microcode is expected to extract the relevant pause time values by writing them to the FIELD16{A..I} channels. The parser marks frames for pause handling by setting the PAUSE_Frame and PAUSE_CBPFrame action flag bits appropriately (bits 34 and 35). The FIELD16{A..I} channels are unconditionally sent to the CM stage and are interpreted as pause control values based on the final state of the pause flags.
3. All other stages of the frame processing pipeline should be configured to ignore and not forward these pause frames, as identified by the PAUSE_Frame flag. Downstream stages that have control over the action flags (L3AR and L2AR) might optionally override these flags to refine the parser's initial classification decision.
4. Upon receiving the frame tail, fixed-function logic verifies that the pause frame was received without error.
5. The pause times are applied to the port's egress scheduling policy. Periodically, based on the port's configured speed, pause quanta are subtracted from these times. As long as the times remain positive, transmission on the (port and TC) remain halted.

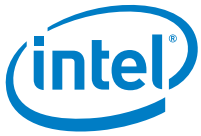
The pause-related flags have the following fixed-function interpretation:

- **PAUSE_Frame** — Indicates a pause frame, either port- or class-based. By setting this flag, the parser activates the pause reception logic and the pause times parsed from the frame are applied to the port's egress scheduling policy.
- **PAUSE_CBPFrame** — Controls whether a single pause time is taken from the value parsed into FIELD16B and applied to all TCs, or whether per-TC pause times are applied from FIELD16{B..I} (8 16b), based on the class vector enable bits in FIELD16A[7:0], as detailed in the sections that follow.

When the PAUSE_Frame flag is set, the FIELD16{A..I} channels are interpreted by the pause reception logic in the following fixed-function manner, depending on the value of the PAUSE_CBPFrame flag:

Table 5-86 Pause Reception Logic

Channel	PAUSE_CBPFrame == 0	PAUSE_CBPFrame == 1
FIELD16A[7:0]	—	Bit i causes the corresponding pause time value to be applied to pause class i. The pause class i is mapped to the appropriate internal TC by the CM_TC_PC_MAP registers.
FIELD16B	Pause time applied to all TCs.	Pause time applied to class 0, if enabled by FIELD16A[0].
FIELD16C	—	Pause time applied to class 1, if enabled by FIELD16A[1].
FIELD16D	—	Pause time applied to class 2, if enabled by FIELD16A[2].
FIELD16E	—	Pause time applied to class 3, if enabled by FIELD16A[3].
FIELD16F	—	Pause time applied to class 4, if enabled by FIELD16A[4].
FIELD16G	—	Pause time applied to class 5, if enabled by FIELD16A[5].
FIELD16H	—	Pause time applied to class 6, if enabled by FIELD16A[6].
FIELD16I	—	Pause time applied to class 7, if enabled by FIELD16A[7].



The parser is responsible for mapping the relevant fields from the received pause frames into these channel fields, which are overloaded in this case for pause signaling.

For each port, each pause class is associated with one or more scheduling TCs by configuring the PauseClass[0..11] fields in CM_TC_PC_MAP[0..75]. Assigning CM_TC_PC_MAP[port].PauseClass[tc] to pc enables pause class pc to pause TC (tc) on port (port).

Each port's pause state and time counters are maintained by a pause sweeper process. The pause sweeper services each TC on each port at a uniform sweeper_period as configured by SWEEPER_CFG.PausePeriod. The actual real time a particular TC pauses in response to some pause_time value received is determined by the following relations:

```
real_pause_time = pause_time * pause_quanta
pause_quanta = MultiplierMantissa * 2^MultiplierExponent / Divisor * sweeper_period
```

The MultiplierMantissa, MultiplierExponent, and divisor parameters are all specified in the per-port CM_PAUSE_CFG[0..75] registers (all prefixed by PauseQuanta).

Since the sweeper services all ports at the same rate, the pause time values received from the ingress pause frames must be scaled to adjust for each port's different speed (pause quanta). Table 5-87 lists the multiplier and divisor settings for the common port speeds supported by the FM5000/FM6000, assuming the default sweeper_period of 96 ns.

Table 5-87 Multiplier and Divisor Settings

Port Speed	Pause Quanta	MultiplierMantissa	MultiplierExponent	Divisor
40 Gb/s	12.8 ns	1	1	15
10 Gb/s	51.2 ns	1	3	15
1 Gb/s	512 ns	1	4	3
100 Mb/s	5.12 µs	5	5	3
10 Mb/s	51.2 µs	26	6	3

Relative to each port's unit of pause quanta, the resolution of the FM5000/FM6000's response to pause time values diminishes with increasing port speed. The absolute unit of response is sweeper_period, or 96 ns. This represents ~8 pause quanta for a 40 Gb/s port, or ~2 for a 10 Gb/s port.

5.18.8 Pause Frame Generation

The FM5000/FM6000 can be configured to generate pause frames in response to changes in switch memory use. One set of pause-on/off watermarks is defined per (Rx and RXMP) queue and generate pause frames in based on the queue's memory use. Another set of pause-on/off watermarks is defined per RXMP and generates pause frames in based on total shared RXMP memory use. The latter shared RXMP watermarks can be enabled or disabled on a per (Rx and RXMP) queue basis. Together, these pause watermarks provide support for three pause-based flow-control models that might be selected per (Rx and RXMP) queue:

- **Per-Queue Pause** — In this model, the per-queue watermarks are used to rigidly partition the RXMP memory among all ports. A queue is paused each time its usage exceeds its allowance.
- **Shared RXMP Pause** — Under this model, a port is paused once the shared RXMP level becomes too large. However, each queue can be granted a private memory allowance and is protected from pause as long as it does not exceed that allowance.

- **No Pause** — By disabling both the per-queue and the shared pause mechanisms, pause can be completely disabled for a given queue. Instead, drop watermarks and/or congestion notification mechanisms are used to manage congestion experienced by the queue.

Both sets of watermarks are defined over 160-byte segment and frame use levels. The register definitions of these watermarks are listed in Table 5-88.

Table 5-88 Pause Watermarks

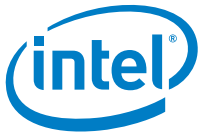
Register	Description	Units
CM_PORT_RXMP_PAUSE_ON_WM[0..75,0..11] CM_PORT_RXMP_PAUSE_OFF_WM[0..75,0..11]	Defines per-(Rx and RXMP) pause ON and pause OFF watermarks. If either frame or segment ON-watermark is exceeded, PAUSE-ON frame generation is enabled. Only after both frame and segment OFF-watermarks are satisfied the PAUSE-ON generation cease and a PAUSE-OFF frame are generated. These watermarks are indexed by (RX,RXMP) and compared against CM_PORT_RXMP_USAGE.	Per-(Rx and RXMP) frames and 160-byte segments.
CM_PAUSE_CFG[0..75].SharedPauseEnable[0..11]	Enables or disables the shared RXMP pause mechanism for each queue. By default, the shared pause watermarks are disabled for all queues.	
CM_SHARED_RXMP_PAUSE_ON_WM[0..11] CM_SHARED_RXMP_PAUSE_OFF_WM[0..11]	Defines per-RXMP pause-ON and pause-OFF watermarks compared against the corresponding CM_SHARED_RXMP_USAGE level. Determine a pause status for each RXMP, which might result in a pause frame being generated on an (Rx and RXMP) queue depending on the queue's per-(Rx and RXMP) watermarks and SharedPauseEnable configuration.	Per-RXMP frames and 10-byte segments.

When both of these pause-ON watermarks are enabled and exceeded, the port (or queue) enters a paused state and a notification is issued to the scheduler to generate a PAUSE-ON frame. The type of pause frame to generate (port-based or class-based) is configured in a per-port manner by the CM_PAUSE_CFG[port].PauseType parameter. At that time, a pause_resend_count value associated with that port is initialized to CM_PAUSE_CFG[port].PauseResendInterval. Thereafter, as long as both pause-OFF watermarks remain exceeded, the port (or queue) remains in the paused state and the pause_resend_count is decremented. Each time the count drops to zero, another PAUSE-ON frame is generated and pause_resend_count is reset to PauseResendInterval.

If at any time the relevant memory levels drop below any of a port's (or queue's) pause-OFF watermarks, a notification is issued to the scheduler to generate a PAUSE-OFF frame. A PAUSE-OFF frame (either port-based or class-based) has the same format as the PAUSE-ON frame, but has all pause times set to zero.

For class-based pause, the pause classes marked on generated pause frames are mapped from RXMP on a per-port basis. The mapping semantics, defined by the CM_PC_RXMP_MAP registers, enables each pause class to represent the status of exactly zero or one RXMPs. If a pause class has no RXMP mapped to it, it always remains in the pause-OFF state. Since the FM5000/FM6000 supports 12 RXMPs but only 8 pause classes, not all RXMPs can be used to control the pause status of a particular port.

Each port's CM_PAUSE_CFG[port].PauseType parameter controls not only the format of pause frames generated, but the precise timing of when its PAUSE frames are generated. For port-based pause (PauseType=0b), any port in a paused state continues resending PAUSE-ON frames as long as any of its RXMP queues exceed their corresponding Pause-OFF watermarks. For class-based pause (PauseType=1b), pause states are maintained individually per RXMP (such as per queue or class). Thus, a PAUSE-OFF frame can be generated for one RXMP queue while another RXMP' queue belonging to the same port still exceeds a pause-OFF watermark.



The following pseudo-code specifies the pause-related watermark evaluations and how these evaluations result in pause frame generation:

```
// Evaluated periodically as memory levels change:
foreach RX in {0..75} {
  // Determine new pause state
  new_pause_state[0..7] = pause_state[RX,0..7]
  foreach RXMP in {0..11} {
      if ( CM_PORT_RXMP_USAGE[RX,RXMP] ≥ CM_PORT_RXMP_PAUSE_ON_WM[RX,RXMP] &&
           (CM_SHARED_RXMP_USAGE[RXMP] ≥ CM_SHARED_RXMP_PAUSE_ON_WM[RXMP] ||
            !SharedPauseEnable[RX,RXMP])) {
        new_pause_state[RXMP] = 1
      }
      if ( CM_PORT_RXMP_USAGE[RX,RXMP] < CM_PORT_RXMP_PAUSE_OFF_WM[RX,RXMP] ||
           (CM_SHARED_RXMP_USAGE[RXMP] < CM_SHARED_RXMP_PAUSE_OFF_WM[RX,RXMP]) &&
           SharedPauseEnable[RX,RXMP]) {
        new_pause_state[RXMP] = 0
      }
}
// Map RXMP to PC (pause class)
foreach PC in {0..7} {
    RXMP = CM_PAUSE_PC_RXMP_MAP[RX].RXMP[PC]
    if (RXMP < 12) new_pc_state[PC] = new_pause_state[RXMP]
    else new_pc_state[PC] = 0
}
// Determine whether PAUSE-ON or PAUSE-OFF frame is to be generated
if ( // Port-based PAUSE:
     CM_PAUSE_CFG[RX].PauseType == 0 &&
     (pc_state[RX,0..7] == 0x00 && new_pc_state[0..7] != 0x00 ||
      pc_state[RX,0..7] != 0x00 && new_pc_state[0..7] == 0x00) ||
     // Class-based PAUSE:
     CM_PAUSE_CFG[RX].PauseType == 1 &&
     pc_state[RX,0..7] != new_pc_state[0..7]) {
  SendPauseFrame(RX, new_pc_state[0..7])
  }
  pause_state[RX,0..11] = new_pause_state[0..11]
  pc_state[RX,0..7] = new_pc_state[0..7]
}
```

Upon receiving PAUSE-ON or PAUSE-OFF frame generation notices, the scheduler transmits the requested pause frame at the next available opportunity. Fundamentally, this might introduce a delay of up to one MTU due to pending frame transmission on the egress port.

The format of the pause frames generated based on the PAUSE-ON/OFF events is controlled by egress modification microcode. The egress modification slices receive the following information from which they generate the appropriate pause frame:

- **PAUSE** — (TCAM key) Indicates that a pause frame needs generated.
- **PAUSE_Tag** — (TCAM key) 2-bit format tag mapped by Tx port number from MOD_TX_PORT_TAG[0..75]. One of these tag bits is expected to distinguish port- versus class-based pause format, redundantly configured as in CM_PORT_CFG[0..75].PauseType.
- **PAUSE_CBP_VEC** — (TCAM key and data field source) 8-bit vector identifying the pause classes with non-zero pause times (such as pause-ON state per pause class). For port-based pause frames (PAUSE_CBP=0b), the vector is either be all ones (indicating pause-ON) or all zeros (indicating pause-OFF). Corresponds to the new_pc_state variable in the previous pseudo-code.

- **PAUSE_TIME{1,2}** — (Data field source) 16-bit pause times available for use in the generated pause frame. Mapped from the destination port number by MOD_TX_PAUSE_QUANTA[port]. Microcode selects which of the two pause times to map to which pause classes.

For each pause frame scheduled, egress modification receives a 16-byte frame fragment consisting of all zeros. Microcode is responsible for using the previous information, as provided by CM, to format the pause frame appropriately. In particular, this involves padding the frame length to 64 bytes.

The pause frames generated by this mechanism are not included in any CM memory usage counters, nor do they factor into any bandwidth accounting for DRR or egress rate limiter scheduling.

5.18.9 Pause Pacing

In addition to watermark-driven pause frame generation, the FM5000/FM6000 supports periodic pause frame generation based on a configured rate per (port, pause class) pair. This pause pacing support is controlled by the following register fields:

- **CM_PAUSE_CFG[PORT].PausePacingMask** — 8-bit mask enabling pause pacing on each pause class.
- **CM_PAUSE_PACING_CFG[PORT,PC].PausePacingTime** — Specifies the time that the pause class remains un-paused (permitted to transmit) every PauseResendInterval.

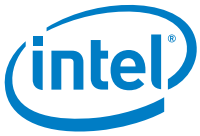
When pause pacing is enabled on at least one pause class of a given port, a pair of PAUSE-ON/PAUSE-OFF frames are generated periodically every PauseResendInterval. At the beginning of each resend interval, a PAUSE-ON frame is generated pausing the classes whose bits are one in PausePacingMask. Each class remains paused until the resend counter drops below its PausePacingTime, at which point a PAUSE-OFF frame is generated (assuming the class does not need to remain paused due to exceeding its pause memory watermarks). After the resend interval time has fully expired, another PAUSE-ON frame is generated and the cycle repeats.

Pause pacing and watermark-driven pause can be enabled concurrently on the same port and pause class. However, any time the port spends paused due to memory watermarks is not explicitly credited against the port's pause pacing time. Thus, the configured PausePacingTime represents the maximum amount of time that the link partner is permitted to send on the pause class.

From the perspective of egress modification, the pause frames generated for pause pacing are indistinguishable from those generated due to memory watermark evaluations.

5.18.10 Congestion Notification Frame Sampling

The FM5000/FM6000 provides a mechanism for periodic sampling of frames based on the number of segments enqueued to a particular (Tx and TXMP) queue. A segment counter is maintained per queue, incremented on every ingress 480-byte segment belonging to a frame enqueued to the (Tx and TXMP) destination. When the counter exceeds some SampleThreshold number of segments, the threshold amount is subtracted from the segment count and the next frame enqueued are sampled.



```
On each frame head handled by the sampler:
   If (SampleCount ≥ SampleThreshold)
     Sample Frame
     SampleCount = min(SampleCount - SampleThreshold, SampleThreshold)
     Update SampleThreshold

At the end of each ingress 480-byte frame segment:
    SampleCount++
```

In the absence of discrete frame length effects, an ideal sampler samples a frame each time SampleCount exactly equals SampleThreshold. For such an ideal sampler, SampleCount is always be reset to zero each time a frame is sampled. However, uncertainty due to discrete frame lengths as well as evaluation timing uncertainties are present in the sampler implementation. These uncertainties motivate the SampleCount - SampleThreshold subtraction to keep the long-term sample rate accurate.

The sampler implementation cannot sample frames faster than one per 2 µs (500 KHz). Under conditions of absolute maximum congestion (sustained N:1 traffic), a sample's overshoot amount (SampleCount - SampleThreshold) can be as large as ~168 KB. This number represents a fundamental minimum bound on SampleThreshold to guarantee avoiding sample rate saturation. Smaller values of SampleThreshold are supported but might result in a slower than expected sample rate and/or undersampling under conditions of extremely heavy congestion. In the previous pseudo-code, this occurs when SampleCount saturates at SampleThreshold due to the minimum evaluation.

The SampleThreshold amount is updated dynamically after each frame is sampled based on the per-queue configuration of CM_PORT_TXMP_SAMPLE_PERIOD. The configuration comprises a deterministic 16-bit SegmentPeriod parameter and a random 4-bit SegmentJitter parameter:

$$SampleThreshold_{next} = SamplePeriod + Random \% 2^{SegmentJitter}$$

SegmentJitter selects the number of random bits to add to the deterministic sample period. Any number of bits between 0 to 12 bits are supported. The long-term average sample period is SamplePeriod + $2^{SegmentJitter-1}$.

Note: All quantities involved in the calculation of the sample threshold are in units of 480-byte segments.

Frames sampled in this manner serve as a measure of congestion experienced on each queue, suitable for use by congestion notification algorithms such as IEEE 802.1Qau Quantized Congestion Notification. With a constant SegmentPeriod, frames are sampled at a rate proportional to the level of congestion.

Frames are sampled by using one of the FM5000/FM6000's four mirrors. The mirror number, command, and map-priority properties are configured statically for each TXMP in the CM_SAMPLING_MIRROR_CFG registers. If a frame is identified for sampling, the corresponding mirror command is applied to the frame using the specified mirror number.

The use of FM5000/FM6000's mirroring mechanism enables sampled frames to be forwarded to a single destination port, configured statically per TXMP. It is expected that this static sampling destination corresponds to an off-chip companion device responsible for:

1. Reformatting the sampled frame in accordance with the relevant congestion notification protocol.
2. Returning the frame back to FM5000/FM6000 so it can be forwarded to the original sender.

A frame is be sampled if its CN_SampleEligible flag (ACTION_FLAGS bit 30) is set to 1b. The CN_SampleEligible flag has no effect on the counting of frame segments. The (Tx and TXMP) segment counters include all segments enqueued, regardless of the status of the frames' CN_SampleEligible

flags. Frames dropped due to any of the Rx watermarks (GLOBAL, RX_HOG, or RX_SHARED) or due to the out-of-memory protection mechanism are not eligible for sampling. Frames dropped for any reason do not count towards a queue's segment sample count.

For a given ingress frame, only one (Tx and TXMP) segment counter is evaluated. If a frame with CN_SampleEligible set to 1b it has more than one bit of its DMASK set to 1b, the segment counter belonging to the lowest-numbered Tx port with DMASK[TX]=1b is evaluated. Normally, it is expected that all such multicast/broadcast/flooded frames have CN_SampleEligible set to 0b.

When a frame is sampled, MOD_DATA.W8C is set to the source Tx port number if L2AR selects this mux case. This value is referenced as CN_SRC_TX in MOD_DATA outputs. If L2AR selects this mux case and the frame is not sampled, MOD_DATA.W8C is set to 0xFF.

The sampling period is configured in the CM_PORT_TXMP_SAMPLING_PERIOD registers, indexed by TX_QIDX.

5.18.11 Interrupt Notification

A pair of interrupt bits, with associated min/max watermarks, are defined per (Tx and TXMP) queue as an accelerated mechanism for notifying off-chip devices of congestion. Each time a queue's memory use exceeds its maximum watermark, the queue’s above the interrupt bit is set. Likewise, each time a queue's memory use drops below its minimum watermark, the below interrupt bit is set. These watermarks are configured per TX_QIDX in the CM_PORT_TXMP_IP_WM registers.

This interrupt mechanism is supported over all Tx queues, including the virtual mirror ports. The interrupt bits are exposed in the CM_PORT_TXMP_{ABOVE,BELOW}_IP registers as 80-bit pending vectors arrayed per-TXMP. A corresponding set of CM_PORT_TXMP_{ABOVE,BELOW}_IM mask registers are also defined. Each time any of these interrupt pending bits is set 1b and is not masked, the interrupt is propagated to the CM_INTERRUPT_DETECT and GLOBAL_INTERRUPT_DETECT registers.

5.19 Packet Replication

Replicating a frame to many destinations is a fundamental requirement for any Ethernet switch. There are three operations needed:

- Replicating the frame to multiple ports
- Replicating the frame for mirroring purpose
- Replicating the frame multiple times on the same port across different VLANs

Finally, a fourth operation is added to this unit.

- Delete frames that have exceeded their time-out values

These requirements are implemented in steps:

1. Frames first go through the FPP for a forwarding decision.
2. Frames are temporarily queued into a RXQ per TC waiting for FPP forwarding decision.
3. Frames are then queued into transmit queues waiting for transmission.
4. Frames are scheduled for transmission and are dropped if too old.

5. Frames are sent to the egress modifier unit.
6. Frames are removed from transmit queue when the last copy needed on this port was sent.
7. Segments are returned to the free pool when frames are no longer in any transmit queue.

Figure 5-44 shows the process.

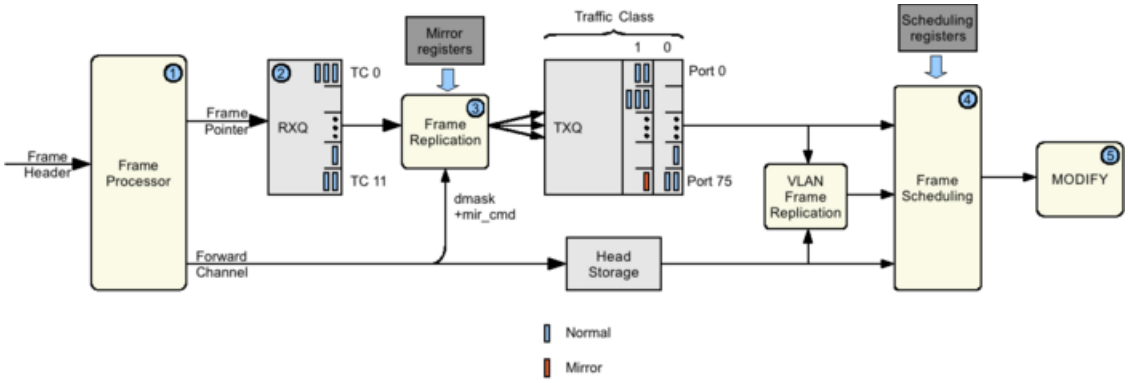


Figure 5-44 Packet Replication

The frame's payload is stored only once in the main memory, and only the pointer to the frame is actually replicated to the different queues. Mirrored frames are queued as well, so a single given port might have up to five copies of the same frame (original and four mirrors) in its transmit queue.

For VLAN replication, from a queue usage perspective, frames are replicated across different VLANs on the same port (as required for some IP multicast routing scenarios). They are queued only once even if replication is required multiple times. From a scheduling perspective, each replicated frame is treated like an individual frame. For example, shaping and prioritization applies to each frame that is replicated and the replicated frames might be interleaved with frames from other higher priority queues.

5.19.1 Frame Replication

Frames freshly received are temporarily placed into a receive queue with a time tag attached. The frame remains in the receive queue until forwarding instructions are received from the FPP.

Next, the MCAST_MID unit uses the destination mask and the mirroring commands received as part of the forwarding instructions to replicate the ingress frame pointer to the proper transmit queues. It executes the following replication procedure:

1. Scan through all ports in the destination mask. For every bit set, scan through the four mirror commands and, for each active mirror command with overlay disabled, check if the explicit destination mask for this mirror is set for this port. If this is the case, enqueue a mirror frame for this mirror. If no explicit mirror was found for this port, enqueue a normal copy.
2. Scan through the four mirror commands for this packet and, for each active mirror command with overlay enabled, place a copy of the frame into the overlay transmit port allocated for this mirror.

Also, when the frame is queued into a transmit queue, a VLAN list pointer is attached to the frame to inform the Frame VLAN replication unit (MCAST_POST) that a replication is possibly needed. This is true for both un-mirrored and Tx mirror copies, ensuring that the mirrored frames contain all copies as with the normal frames. For Rx mirror copies, the VLAN pointer list is null and no VLAN replication is done.

Table 5-89 lists the registers used in this case.

Table 5-89 Packet Replication Register Usage

Register	Field	Width	Function
MCAST_MIRROR_CFG[0..3]	ExplicitDestMask	76	Defines the explicit destination mask for non-overlay mirrors.
	OverlayEnabled	1	Defines if this is an overlay mirror or not.
	OverlayDestPort	7	Defines the output port for an overlay mirror.
	OverlayTC	4	Defines the new TCs if requested for the overlay mirror.
	CanonicalSrcPort	7	Defines the port that is used as a reference for TXmirrors.
MCAST_DEST_TABLE	DestMask	76	Defines if there is a VLAN replication list for each port. The index for this table is supplied as part of the forwarding information received from the FPP.
	MulticastIndex	16	Defines the base index in the VLAN multicast table.

5.19.2 Frame VLAN Replication

Some applications, like L3 multicast and RBridge, require packets to be replicated across multiple VLANs for the same port. This is accomplished by this unit using the MCAST_VLAN_TABLE. The table is indexed from information retrieved from the MCAST_DEST_TABLE.

There are two masks involved in the process:

- The actual destination mask supplied by the FPP
- The destination mask included in the MCAST_DEST_TABLE

The actual destination mask supplied by the FPP is used to determine on which port a packet is transmitted. The destination mask included in the MCAST_DEST_TABLE is used to determine the number of VLAN lists in the MCAST_VLAN_TABLE and, thus derive an actual index for the head of the VLAN list for a given destination port.

If the actual destination mask is one for a given port and the MCAST_DEST_TABLE mask is zero for the same port, it means that there isn't any VLAN replication required for this port and the native copy gets sent (see Figure 5-45).

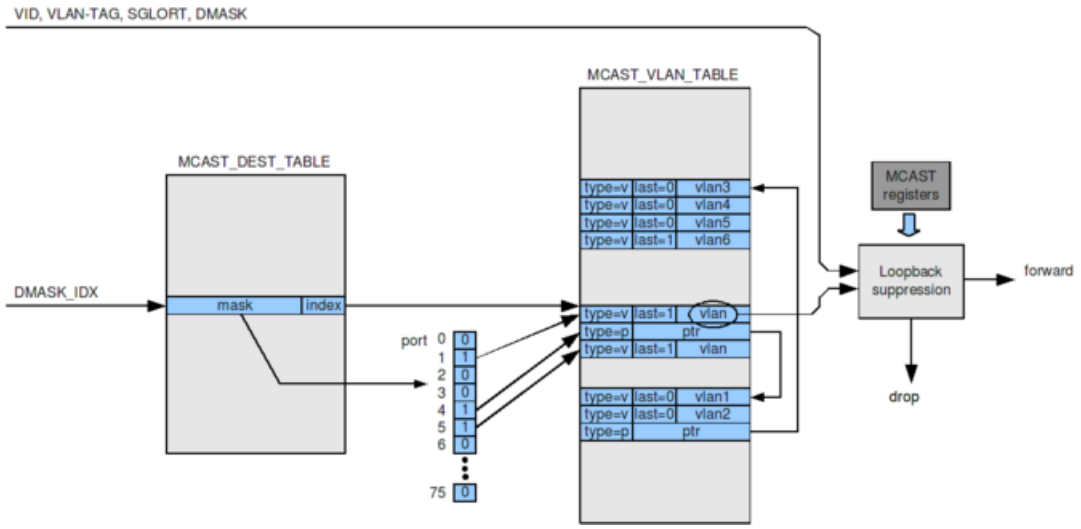
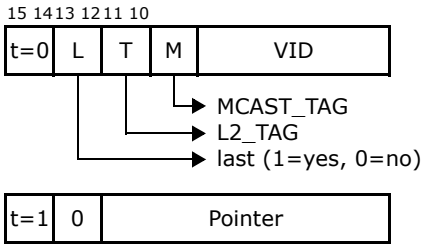


Figure 5-45 VLAN Packet Replication Indexing

The entries in the MCAST_VLAN_TABLE can be of two types:

- **Pointer** — Points to a new location for the rest of the VLAN list.
- **VLAN** — VLAN entries which include the following fields:
  - **VID** — A 12-bit VLAN ID.
  - **LAST** — Indicates if this is the last item of the list.
  - **L2_TAG** — A 1-bit tag involved in the loopback suppression evaluation.
  - **MCAST_TAG** — A 1-bit tag available for microcode interpretation in the modify slice TCAM key

The actual format of each entry is:



The MCAST_POST also includes a generic multicast loopback suppress. This avoids transmitting a frame back to the port it came from if the VLAN is unchanged.

The loopback suppress requires the following information:

- A 1-bit L2_Tag attached to the frame supplied by FPP
- A 16-bit source GloRT attached to the frame

- A 1-bit L2_Tag supplied with the VLAN entry
- A register to define the canonical GloRT of the egress port
- A single bit attached to the frame identifying the egress VID to use (VID1 or VID2).

The goal of loopback suppression is to compare the VID/TAG of the entry to the VID/TAG supplied with the frame, and if they match, suppress sending the frame to the canonical port on which the frame was received. The expressions are:

```
glort-mask = MCAST_LOOPBACK_SUPPRESS[egress_port].mask;
csglortX = MCAST_LOOPBACK_SUPPRESS[egress_port].glort;
csglortF = frame.sglort & glort-mask;
vidX = MCAST_VLAN_TABLE[ptr].VID;
vidF = (frame.l2_vid_select) ? frame.L2_VID2 : frame.L2_VID1
tagX = MCAST_VLAN_TABLE[ptr].L2_TAG;
tagF = frame.L2_TAG
drop = (csglortX == csglortF) & (vidX == vidF) & (tagX == tagF)
```

If the frame is forwarded, the frame VID (L2_VID1 or L2_VID2 depending on L2_VID_SELECT) gets updated with the VID recovered from the MCAST_VLAN_TABLE. If the VID ended up the same, either because the new VID is the same or because the entire MCAST_VLAN_TABLE was skipped, the flag L2_VID_EQUAL is set.

5.20 Scheduler

The scheduler is responsible for the following tasks:

- Manage the free lists.
- Allocate segments from the free pool to ports as needed.
- Receive data segments from ports and placing them in a receive queue.
- Forward frame header data to frame handler.
- Receive the forwarding decision from the frame processing pipeline.
- Dequeue frames from the received queue and enqueue them into as many transmit queues as defined by the FPP.
- Apply the egress scheduling algorithm to schedule frames.
- Free frames as they are transmitted.
- Return segments to the free pool once all copies of the frame have been transmitted or deleted.

Each FM5000/FM6000 egress port supports up to 12 Class of Service (CoS) queues, or TCs, that provide a variety of programmable scheduling and shaping options. As described earlier, the TC is mapped from the final ISL_PRI priority field at the output of the L2AR stage. The TCs are further partitioned into collections of various scheduling groups, on which the egress scheduling algorithm operates. The egress scheduling algorithm supports the following features:

Strict Priority Scheduling

- Strict-high classes are scheduled before all frames of lower priority.
- Strict-low classes are scheduled only if all higher classes have no frames to send.
- Consecutive queues can be assigned to the same strict priority group.

Deficit Round-Robin (DRR) Scheduling

- Groups of sequential traffic classes share a guaranteed minimum bandwidth allocation.
- Lower priority groups are scheduled only if all higher priority groups either have no frames to send or have exceeded their minimum bandwidth guarantees.
- Equal priority groups are scheduled in round-robin order until a group has no frames to send or has exceeded its minimum bandwidth guarantee.
- Within a DRR scheduling group, higher-priority classes are strictly preferred over lower-priority classes.

Traffic Shaping

- Groups of TCs can be assigned a maximum egress bandwidth limit.
- Queuing delay, not discard, is used to enforce the bandwidth limits.
- Consecutive TCs can be assigned to the same shaping group.

Figure 5-46 shows an example implementation of egress queuing and shaping. This is intended as a guide for understanding the scheduling algorithm, which is described later in this section.

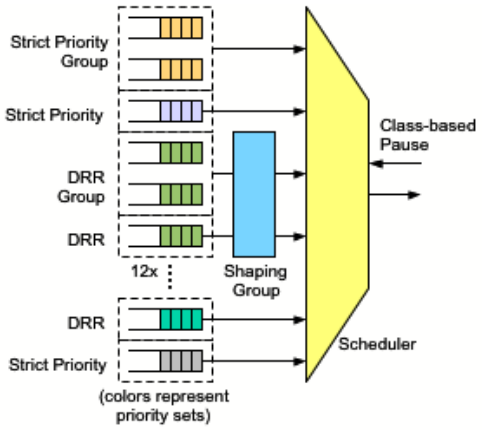


Figure 5-46 Example Egress Scheduler Configuration

5.20.1 Group Eligibility

The scheduler determines which scheduling groups are eligible for transmission by considering the pause status of each class and the current bandwidth consumption of each group. More precisely, a group is eligible if both of the following conditions hold:

1. It contains at least one class that is not paused, and that class has at least one frame queued for transmission.
2. The group's bandwidth use is under its shaping limit. For example, the shaping group g that contains the scheduling group i has $B_g < UB_g$.

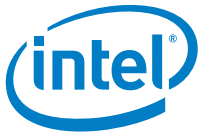
The pause state of a class is controlled by receiving pause frames from the port's link partner. The FM5000/FM6000 supports two types of pause frames: standard IEEE 802.3 pause and IEEE class-based pause frames. Receiving a standard pause frame causes all of the port's TCs to become ineligible for a period of time specified by the frame's pause value. If the pause time is zero, all paused TCs immediately become un-paused. Class-based pause frames have similar semantics, except a pause frame can apply to a specific set of TCs, and each class is assigned its own pause time.

The second eligibility condition represents an internally generated source of egress queue pausing. Each shaping group's cumulative bandwidth use is tracked over a configured time window. If the group consumes too much of the port's bandwidth, it exhausts its bandwidth credit and the constituent scheduling groups become ineligible. The shaping function is implemented using a leaky bucket algorithm, which is described later in this section.

5.20.2 Class Selection

Given a non-empty set of eligible groups as defined in the previous section, the scheduler selects a single group from which the next frame is scheduled. A Deficit Counter (DC) is used within each DRR group. A value Q is used, which is also known as DRR weight. it is a measure of the proportion of a port's bandwidth allocated to a given DRR group. A strict priority group can be considered similar to a DRR group, but with an infinite DC value. Such high-numbered (high priority) classes are said to have strict-high priority and low-numbered classes have strict-low priority. Following, the subscript i represents a specific DRR group.

1. Any DRR groups in the eligibility set with $DC_i \leq 0$ are pruned from the eligibility set as long as there is at least one group in the eligibility set with $DC_i > 0$.
2. If all DRR groups in the eligibility set have $DC_i \leq 0$, then Q i is added to each DC_i, marking the beginning of a new DRR scheduling round. The LastDRRGroup state is reset to -1.
3. Among all eligible groups, the scheduler selects the group that matches the earliest of the following rules. When multiple groups match, the group that is numerically larger is chosen.
   a. The group has a strict-high prioritization.
   b. LastDRRGroup, unless the value is -1.
   c. A group belonging to a different and numerically higher priority set than LastDRRGroup, or anyDRR group if LastDRRGroup is -1.
   d. A DRR group i belonging to the same priority set as LastDRRGroup, with i < LastDRRGroup.
   e. A DRR group i belonging to the same priority set as LastDRRGroup, with i > LastDRRGroup.
   f. Any group in the eligibility set.
4. If the selected group is a DRR group, LastDRRGroup is set to that group number. If is not, LastDRRGroup is set to -1.



The next scheduled frame is then chosen from the class with the highest *inner priority* that satisfies all of the following conditions:

- Belongs to the selected scheduling group.
- Has at least one frame to be scheduled.
- Is not paused.

Each TC has one bit of explicitly configured inner priority, plus lower-order priority implied from its TC number. If InnerPriority(c_1) > InnerPriority(c_2), then c_1 is always chosen over c_2 if it satisfies the previous conditions. If InnerPriority(c_1) = InnerPriority(c_2), the higher-numbered class is chosen.

5.20.3 Algorithm Notes

The FM5000/FM6000 scheduling algorithm, as previously described, has the following characteristics:

- Once a particular DRR group is chosen for scheduling, the group continues to be serviced until one of the following conditions arises:
  - Its deficit count becomes zero or negative.
  - It becomes ineligible (due to emptying its queue or due to pause).
  - A strict-high priority group becomes eligible.
  - A strict-low priority group is only scheduled if none of the other groups, including DRR groups, are eligible. A frame being scheduled from such a group implies that all DRR groups have maximum deficit counts and are waiting for eligible frames to send.
- When the DRR scheduling switches to a different priority set, the round-robin pointer state within the prior priority set is lost. Thus, the (non-strict) group prioritization only influences the iteration order through the DRR groups within a given scheduling round and it does not provide independent DRR schedulers. This feature is intended to reduce the scheduling latency of particular DRR groups relative to others. The latency preference only applies when the lower-priority groups are not perennially backlogged.
- The inner priority and class number represent a nested strict prioritization of classes belonging to the same DRR group. Two classes mapped to the same scheduling group is guaranteed a minimum egress bandwidth, represented by the group's Q_i (in the absence of strict-high priority frames). However, the class with the higher inner priority starves the lower-priority class as long as it remains eligible. In most applications, this nested strict priority behavior is not needed, and the scheduling groups can be left at their default one-class-per-group configuration.
- Configuring two classes to have strict priority gives the same scheduling behavior in all of the following scenarios:
  - Both classes belong to the same scheduling group.
  - Each class belongs to its own unique scheduling group and priority set.
  - Each class belongs to its own scheduling group, but share the same priority set.

5.20.4 Deficit Round-Robin

The terms Deficit Weighted Round-Robin and Deficit Round-Robin are both widely used and reference the same scheduling algorithm. Intel uses the latter term to be consistent with the original publication of this algorithm (by M. Shreedhar and G. Varghese in 1995.)

The original specification of the DRR algorithm assumed store-and-forward switching: the length of the current frame determines whether the next frame in the same queue can be transmitted or not. However, in the FM5000/FM6000 as in any other high-performance cut-through switch, the next scheduling decision must be made in a speculative manner, before the length of the previously scheduled frame is known. The DRR implementation therefore must delay its decrement of the scheduled frame's deficit count relative to the standard algorithm. Intel refers to this variant of the DRR algorithm as Delayed Deficit Round-Robin (DDRR).

Given a set of per-queue DRR quanta Q i (all greater than or equal to 2*MTU), the guaranteed minimum bandwidth of a particular group, as a proportion of the port's total egress bandwidth, can be expressed:

$$LB_i = \frac{Q_i}{\sum Q_i}$$

If any Q_i is less than 2*MTU, the minimum bandwidth allotted to each group becomes dependent on the actual frames queued.

Note: The presence of strict-high classes in the scheduler configuration also disturb the DRR groups' lower bound bandwidth guarantees.

5.20.5 Bandwidth Shaping

In this section, the shaping function has been described as a simple comparison between a measured bandwidth per shaping group, B_g, and some configured upper-bound rate limit, UB_g. Groups remain eligible for scheduling as long as their bandwidth remains lower than the bound: $B_g < UB_g$. This is a simplification of the actual bandwidth shaping algorithm. The actual shaping function is implemented using a token bucket algorithm, which has the following characteristics:

- Every unit of time, $1/UB_g$ bytes of credit (tokens) are added to the bucket.
- Credit is subtracted from the bucket's token count as bytes belonging to the shaping group egress the device.
- When the token count goes to less than or equal to zero, all associated scheduling groups become ineligible.
- The capacity of the bucket determines the maximum amount of burst that is permitted in the shaping group's egress bandwidth profile. Credit stops accumulating in the token bucket once this capacity is reached.

In the FM5000/FM6000, the token bucket parameters have the definitions listed in Table 5-90.

Table 5-90 Scheduler Token Bucket Parameters

Symbol	Mnemonic	Definition
T	Time Unit	The period of the pause sweeper process that maintains the bandwidth shaping state. Nominally 96 ns.
C	Capacity	Token bucket capacity, specified in units of 64 bytes.
r	Rate	Number of bytes credited every sweeper period T, in units of 1/16 bytes.

The UB_g parameter previously used as a proportion of a port's egress bandwidth, is r/(T*R), where R is the total bandwidth of the egress port (nominally 1.25 bytes/ns for a 10 Gb/s port). With these parameters, the Maximum Burst (MB) a shaping group can transmit (starting from an idle, fully credited, state) is:

$$MB = C + MTU + R \cdot T + r$$

This burst occurs over MB/R ns. The additional MTU term is seen in the worst case because scheduling decisions are made without regard for the lengths of frames. A maximum length frame can be scheduled when a group's token count is at its minimum positive value (one), causing the count to go negative by up to MTU equals one. This is a requirement of any cut-through switch. The non-ideal condition only occurs on short time scales; the scheduler guarantees that the UB_g bound is satisfied on time scales larger than 2 times MB/R.

As in the DRR implementation, an inter-frame gap plus preamble byte count penalty (of 20 bytes) is subtracted from the credit count at the end of each transmitted frame. This correction properly models the higher bandwidth cost of scheduling small frames versus large frames. These penalty bytes should be considered when interpreting the maximum burst byte count calculation described previously in this section.

The bandwidth shaping mechanism supports egress rate limits over a range of 635 KB/s to 5+ GB/s with capacity values of 64 bytes to 4 MB.

5.20.6 Frame Timeout

The scheduler implements a mechanism for discarding frames that have resided in frame memory for too long a time. Upon ingress, each frame is assigned a global timestamp value that increases at a rate configured by the FRAME_TIME_OUT register. The egress scheduling policy takes the timestamp at the head of each Tx queue into consideration when scheduling frames. For each queue, if the difference between the head frame's timestamp and the current global timestamp exceeds the frame timeout period, the frame is discarded.

Frame timeouts are serviced with strict-high priority relative to all normal frame scheduling policies. In particular, a port's pause and bandwidth shaping status has no effect on the servicing of frame timeouts. If multiple traffic classes on a given port contain out-of-time frames, the higher-numbered TCs are serviced with strictly higher priority than the lower-numbered TCs.

Each frame segment discarded in this manner costs a minimum-size frame scheduling time slot on the affected port, resulting in a ~67 ns * cell (frame_length/160) loss of egress bandwidth on that port. In the corner case that the entire frame memory is filled to its maximum segment capacity with frames enqueued to a single paused port, the scheduler requires 3.3 ms to drain the memory. Therefore, to guarantee reliable timeout, the timeout period should not be configured less than this value.

At the high end, the timeout period is limited only by the bit width of the FRAME_TIME_OUT register. Thus, any value up to the maximum of 72 minutes is supported.

The frame timeout mechanism does not guarantee a high degree of precision in the timeout period. In the worst case, the actual timeout period experienced by a given frame could be ±50% the configured value.

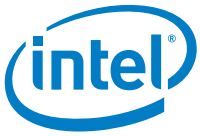
Stalling on the EBI frame reception interface might, under some circumstances, inhibit the timing out of frames enqueued for the EBI port. To avoid lost frame timeouts, the EBI interface must dequeue frames from the switch no slower than one frame every half of the configured frame timeout period (specifically, one frame per FRAME_TIME_OUT.timeOutMult * 1024 * PCIE_REFCLK_PERIOD).

5.20.7 Configuration Registers

Table 5-91 lists the configuration registers relevant for egress scheduling.

Table 5-91 Scheduler Configuration Registers

Register Name	Description
CM_TC_MAP	Maps switch priority to TC (an arbitrary 16-to-12 mapping table). ***Note:*** This is global for all ports.
ESCHED_DRR_CFG ESCHED_CFG_1 ESCHED_CFG_2	Defines the scheduling group and priority set boundaries as bit vectors over TCs. Also specifies the per-class strict priority attribute. These mappings can be configured uniquely per-port. ***Note:*** These two registers are intentionally redundant and must be configured identically for the scheduling to operate correctly.
ESCHED_CFG_3	Specifies one bit of inner priority per TC.
CM_BSG_MAP	Specifies per-port mappings from TC to shaping group.
ESCHED_DRR_Q	Defines the DRR quantum per group per port. Additionally, in ESCHED_DRR_CFG, each TC can be configured to interpret all frames as having zero length, providing the behavior of an infinite quantum value.
ERL_CFG	Defines the egress rate limiter (bandwidth shaping) parameters per shaping group, per port.
ERL_USAGE	Stores the current token bucket credit count per shaping group.
FRAME_TIME_OUT	Specifies the global frame timeout period.



5.20.8 Definition of Terms

The following is a list of terms used in this section.

Table 5-92 Scheduler Definition of Terms

Quantity	Notation	Definition
Traffic Class	c	Identifies one of 12 egress queues from which the frame is scheduled. Also known as fabric priority and scheduler priority. Mapped from the frame's switch TCs c priority (via the CM_TC_MAP table.) The TC numeric value is significant when multiple classes are allocated to the same DRR group. In that context, higher numbered classes are strictly preferred over lower numbered classes.
Scheduling Group	i	A set of TCs that share a DRR minimum bandwidth allocation. Groups are defined as contiguous sets of TCs. (That is, for i = {c1, c2, ... cn }, c2 = c1+1, c3 = c2+1, etc.). In the absence of strict-high prioritized frames, the total egress bandwidth over these classes is guaranteed to never fall below the minimum limit (Q i), assuming the group stays collectively backlogged. Like bandwidth shaping groups, DRR groups are defined as a series of contiguous TCs.
Bandwidth Shaping Group	g	A set of egress TCs for a given port that share a common maximum bandwidth allocation. The total egress bandwidth over these classes is guaranteed bandwidth shaping g group to never exceed the group's upper-bound limit (UB_g). Shaping groups are defined by a class-to-group (many-to-one) mapping. Although hardware supports a fully general mapping, shaping groups should be configured to be super sets of scheduling groups.
Priority Set	s	Contiguous TCs of equal priority form a priority set. A priority set is preferred in scheduling if its TC members have higher numeric values than another priority set s priority set's. Classes belonging to the same scheduling group must be configured to have the same priority.
Strict Priority	SP_c	An attribute configured per TC. When set, the TC has effectively infinite DC_g, it preempts lower-priority classes, and it is preempted by strict priority SP c higher-priority classes. The strict priority attribute must be uniformly set within each priority set. Additionally, it should only be assigned to contiguous sets starting from the outermost ones (numbers 0 and 7). Such high-numbered (high priority) classes are said to have strict-high priority; low-numbered classes have strict-low priority.
Inner Priority	—	One bit of priority assigned per TC. If c_1 and c_2 belong to the same scheduling group and InnerPriority(c_1) > InnerPriority(c_2), then c_1 is serviced with strict inner priority - high priority relative to c_2. Among all TCs of equal inner priority in a given scheduling group, higher-numbered classes are preferred over lower-numbered classes.
DRR Group	i	A scheduling group that is set to be non-strictly scheduled. That is, SP_c =0b for all classes c in the group. DRR groups are scheduled according to a deficit weighted DRR group i round-robin algorithm each time all strict-high classes have no eligible queued frames.
DRR Quantum	Q_i	A measure of the proportion of a port's bandwidth allocated to a given DRR group. Also known as DRR weight. The scheduler attempts to provide no less than this proportion of bandwidth to the associated DRR group. Q_i is measured in bytes and must be no smaller than twice the maximum frame size of the network (MTU).
Deficit Count	DC_i	Dynamic state indicating the portion of DRR quantum remaining in a scheduling round. The deficit count is decremented as frame segments egress the device. The counter deficit count DC_i is incremented by Q_i at the end of a scheduling round. DC_i is signed and never greater than Q_i.
Scheduling Round	—	A scheduling iteration over all TCs. The end of a scheduling round occurs when no DRR group can be scheduled without replenishing its deficit count.
Average Bandwidth Use	B_g	The average egress bandwidth use of a particular shaping group, as measured over some period of time.
Upper-bound Bandwidth Limit	UB_g	For each shaping group, the egress scheduler guarantees that $B_g \leq UB_g$.

5.21 Egress Modification

The final frame processing stage of the FM5000/FM6000 pipeline is egress modification. This stage performs the complementary function of the parsing and association stage at the beginning of the pipeline, as it reformats the frame header from a combination of fixed-function and generic data fields and processing flags. As with the parser, very little of this unit's functionality is specific to Ethernet or the higher-layer protocols supported by the FM5000/FM6000. All behavior of use to the application layer must be configured with microcode.

5.21.1 Basic Properties

The FM5000/FM6000's modify architecture provides the following egress modification capabilities:

- Ability to apply up to 20 modification commands to each frame (see Section 5.21.5). Up to 56 bytes of frame data can be inserted or replaced.
  - For ports operating at 10 GbE or below, only the first 160 bytes can be modified. The rest of the payload has to be passed unmodified.
  - For ports operating at 20 GbE or above, only the first 80 bytes can be modified. The rest of the payload has to be passed unmodified.
  - When replacing or inserting frame data, the new frame data might come from the generic forwarding data fields generated by L2AR at the conclusion of ingress processing, from local tables in the unit, or from constants specified in modification rules. The data from local tables available in egress modifications are:
  - 12 bits of CAM key bits per destination port (76 x 12-bit table).
  - One bit of tagging state per (VID1, DST_PORT) pair (4 Kb x 76-bit table).
  - One bit of tagging state per (VID2, DST_PORT) pair (4 Kb x 76-bit table).
    - This table can be re-purposed to provide extra 64-bit mapped data
  - 12-bit data mapped from either DST_PORT or TX_PORT. (76 x 12-bit table).
  - 2 x 8-bit configured to be mapped from DST_PORT, TX_PORT (2 x 76 x 8-bit tables).
  - 4 x 16-bit configured to be mapped from VID1, VID2 or generic 12-bit index from L2AR (4x4Kx16b tables).
  - 4 bits of egress VPRI1 mapping by (VID1, DST_PORT) pair (76 x 16 x 4-bit table).
  - 4 bits of egress VPRI2 mapping by (VID2, DST_PORT) pair (76 x 16 x 4-bit table).
- Ability to overwrite frame data with 2-bit assignment granularity and 4-bit rotation alignment per byte of new data.
- Limited support for calculating or updating a two-byte ones-complement checksum header field. In some applications, this enables IP, TCP, or UDP checksums to be set to properly reflect changes made to those protocols' packet headers.
- Support for numerically decrementing bytes of the frame. Provided primarily to support decrementing the IP or MPLS TTL when routing.
- Global drop and pre-rule drop options.

5.21.2 Top Level Organization

Although both the parser and the modify units share a slice-based architecture, the modify stage is considerably more complicated than the parser. Its processing relies on a fixed-function input mapping stage, two different flavors of slices, and a high-performance serial modification engine at its output.

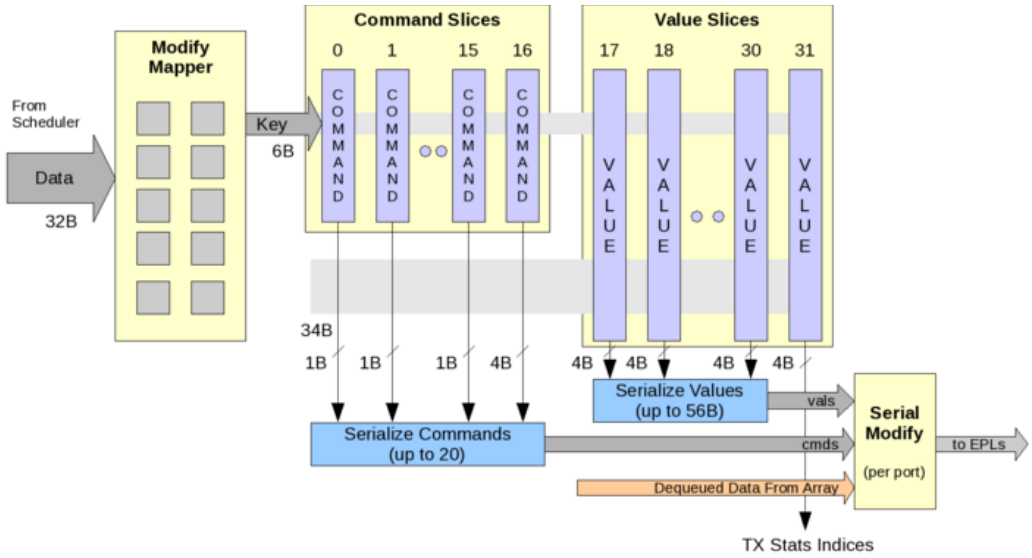


Figure 5-47 Top-Level Egress Modification Architecture

There are three major stages in the modify architecture:

- **Modify Mapper** — Receives the modify channel from the scheduler, which includes MOD_DATA, MOD_FLAGS, QoS, and other fields that have been generated or transformed by the scheduler. Some of these fields are transformed by Tx port number to map the values to the appropriate egress destination, and produce key and data operands for the modify slices.
- **Modify Slices** — Uses a TCAM/RAM slice structure to generate a set of modification commands. Thirty-two CAM slices are available to store rules; the first 17 store rules for commands, the next 14 store rules for data to egress modifier and the final one is for data to stats.
- **Serial Modify** — Performs the final transformation of the egress frame data and applies the serialized commands to the byte-serial frame data dequeued from the Array. Each command can modify one or more bytes of the frame, and each command might require a variable number of command operand values (zero, one, or more). Once the modification commands have been exhausted, any remaining bytes of frame data are passed unmodified to the port, which is responsible for link-layer transmission and recalculating the frame's CRC.

5.21.2.1 Data from Scheduler

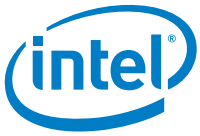
The data from the scheduler is listed in Table 5-93. Fields with a gray background are produced by the scheduler. Fields with a yellow background are transformed by the scheduler prior to the egress modify unit.

Table 5-93 Egress Modification Data from Scheduler

Field	Width	Description
TX_PORT	7	Physical Tx port number; actual egress port number of the frame.
MIR_RX	1	If set, indicates the frame is an Rx-mirror frame.
MIR_TX	1	If set, indicates the frame is a Tx-mirror frame. MIR_RX and MIR_TX are never both set.
MR_NUM	2	Set to indicate the mirror number.
TRUNC	1	Specifies whether frames are to be truncated. For mirrored frames, depends on Forward.MIR_TRUNC; for normal frames, depends on Forward.TX_TRUNC and TX_TRUNC_MASK, configured statically in post.
MAP_PRI	1	Specifies whether a mirror frame should be mapped to a new ISL priority. Depends on Forward.MIR_MAP_PRI.
L2_VID1	12	Egress VLAN1 ID. Potentially updated by frame replicator if L2_VID_SELECT is zero.
L2_VID2	12	Egress VLAN1 ID. Potentially updated by frame replicator if L2_VID_SELECT is one.
L2_VID_EQUAL	1	Indicates if the frame after MCAST replication has the same VID{1,2} as the one supplied by the FPP. This is used by modifier to cancel routing modifications for frames that are multicasted on the same VLAN they come from. (Equals zero if no IP multicast VLAN table lookup was performed.)
MCAST_TAG	1	1-bit tag from IP multicast VLAN table entry. (Equals zero if no IP multicast VLAN table lookup was performed.)
L2_TAG	1	1-bit tag supplied by L2AR for comparison purpose in multicast VLAN table.
QOS	24	QoS-related fields. Some of these have fixed definitions, others are configured to be defined. Grouped together since they are opaque to the scheduler and transformed as a group.
MOD_FLAGS	24	Configured to be generated modification flags opaque to the scheduler. The meaning is microcode defined.
MOD_DATA	140	Configured to be generated modification data fields opaque to the scheduler. The meaning is microcode defined.
PAUSE	1	Set if a PAUSE frame should be generated.
PAUSE_CBP_VEC	8	Specifies the mask of TCs to pause for a class-based-pause frame. If bit i is 0b, the class is un-paused (sending a quanta of zero). If class-based-pause is not enabled in CM_PAUSE_CFG.PauseType, this vector is either all zeros (0x00) or all ones (0xFF).

5.21.2.2 PAUSE Generation

When the CM PAUSE generation logic determines that a PAUSE frame must be sent on a particular port, the scheduler synthesizes an all-zero 64-byte frame to be formatted appropriately by egress modification. For PAUSE frames, the previous data fields sent from the scheduler are all zero, with the exception of TX_PORT, PAUSE, and PAUSE_CBP_VEC. Various fields of the modify slices' TCAM key and data fields are overloaded for PAUSE generation, as described in the sections that follow.



5.21.3 Modify Mapper

The modify mapper is split into a series of tables and transforms that uses data received from the scheduler to construct data for consumption by the modify slices. More specifically, the mapper prepares the following data:

- Key to modify command and value slices
- Data channels to modify value slices

The tables in Modify Mapper are:

- **MOD_TX_PORT_TAG[0..75]**
- **MOD_DST_PORT_TAG[0..75]**
  - These tables are indexed by the actual Tx port or the original intended destination port (DST_PORT). In a non-mirror case, the TX_PORT and the DST_PORT are the same. In a mirror case, the TX_PORT is the actual output port while the DST_PORT was the intended port. The width of the TX_PORT_Tag is 2 bits while the width of the DST_PORT_Tag is 10 bits. For PAUSE frames, an additional 2-bit PAUSE_Tag (indexed by TX_PORT) is provided. The tags can be used as a key and represent per-port configuration options such as inter-switch tagging type and class-based pause.
- **MOD_L2_VPRI1_TX_MAP[0..75]**
- **MOD_L2_VPRI2_TX_MAP[0..75]**
  - Used to transcode QOS.L2_VPRI1 or QOS.L2_VPRI2 per port.
- **MOD_TX_PAUSE_QUANTA[0..75]**
  - For PAUSE frame generation, two PAUSE_TIME values are mapped from DST_PORT. For normal port-based PAUSE frames, only one value is used. For class-based PAUSE frames, microcode determines which of the two pause times to apply to each pause class.
- **MOD_MIN_LENGTH[0..75]**
  - Used to configure the minimum packet size. Shorter frames get padded. The length configured must exclude the CRC (4 bytes).
- **MOD_MAP_DATA_W8A[0..75]**
- **MOD_MAP_DATA_W8B[0..75]**
  - General purpose tables returning one byte. Indexing is configurable between TX_PORT or DST_PORT.
- **MOD_MAP_DATA_W12A[0..75]**
  - General purpose table returning 12 bits. Intended use is to define a default VID per port; it is combined with the remapped L2_VPRI1 to create a 16-bit VLAN tag.
- **MOD_MAP_DATA_W16A[0..4095]**
- **MOD_MAP_DATA_W16B[0..4095]**
- **MOD_MAP_DATA_W16C[0..4095]**

- **MOD_MAP_DATA_W16D[0..4095]**
  - — General purpose tables returning 16 bits of data. Indexing is configurable between TX_PORT, DST_PORT, MOD_T1_IDX, MOD_T2_IDX, MOD_DATA.W16C, L2_VID1 or L2_VID2.
    - MOD_T1_IDX is {QOS.W4[3:0],MOD_DATA.W8A[7:0]}
    - MOD_T2_IDX is {MOD_DATA.W4[3:0],MOD_DATA.W8B[7:0]}
    - For PAUSE frames, any selected index other than TX_PORT or DST_PORT is treated as zero.
- MOD_L2_VLAN2_TX_TAGGED (alias MOD_MAP_DATA_V2T)
  - — Dual purpose table; VLAN2 tagging or general data.

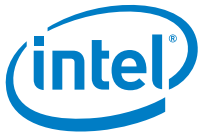
The table MOD_L2_VLAN2_TX_TAGGED (alias MOD_MAP_DATA_V2T) as dual purpose; operation mode is configured globally. It can be used as a 1-bit per-port VLAN2 tagging table (76 bits entry, MOD_L2_VLAN2_TX_TAGGED) to control VLAN2 tagging per port, or as an extra 64-bit new frame data (72-bit entries, MOD_MAP_DATA_V2T) to provide more frame data (typically large tunnel). If used as an extra 64-bit frame data, the entry can be used to update any of the following fields:

- **MOD_DATA_W16A[7:0]**
- **MOD_DATA_W16A[15:8]**
- **MOD_DATA_W16B[7:0]**
- **MOD_DATA_W16A[15:8]**
- **MOD_DATA_W8B**
- **MOD_DATA_W8C**
- **MOD_DATA_W8D**
- **MOD_DATA_W8E**

The upper 8 bits of the entry indicates if the entry received from L2AR is kept as is (zero) or if the entry is updated with the value from this table (one).

The modify mapper also includes the following registers:

- **MOD_MAP_DATA_CTRL**
  - — Defines the method to index the different tables.
- **MOD_TX_MIRROR_SRC**
  - — For Tx-mirror frames, defines the original intended destination port (called DST_PORT) being mirrored.
- **MOD_MAP_DATA_V2T_CTRL**
  - — Control usage of MOD_L2_VLAN2_TX_TAGGED/MOD_MAP_DATA_V2T as either a 1-bit per-port VLAN2 tagging table or as extra 64-bits of frame data and defines the method to index this table.



5.21.4 Modify Slices

The modify slices are constructed using 32 CAM slices for rules, 20 RAM banks for defining commands and 15 RAM banks for data. The CAM slices are used as follows:

- **CAM slice 0..15** — For commands 0..15, a single CAM rule in each CAM slice enables one command, defined in MOD_COMMAND_RAM[0..15].
- **CAM slice 16** — For commands 16..20, a single CAM rule in this CAM slice enables up to four commands, defined in MOD_COMMAND_RAM[16..19].
- **CAM slice 17..30** — For egress data 0..56, a singe CAM rule in each CAM slice enables up to 4 bytes of data, defined in MOD_VALUE_RAM[0..13].
- **CAM slice 31** — For stats data 0..3, a singe CAM rule in each CAM slice enables up to 4 bytes of data, defined in MOD_VALUE_RAM[14].

5.21.4.1 TCAM Key

The TCAM Key used in all modify slices is defined in the following table:

Table 5-94 Egress Modification Modify Slices Key

Field	Bit(s)	Description
PAUSE	0	Indicates egress PAUSE frames.
MIR_RX	1	Indicates if this is an Rx-mirror copy or not. Exclusive with MIR_TX.
MIR_TX	2	Indicates if this is a Tx-mirror copy or not. Exclusive with MIR_RX.
MIR_NUM	4:3	Mirror number, only applicable if MIR_RX or MIR_TX is set.
MAP_PRI	5	Indicates mirror frames are to be mapped to a new priority.
TRUNC	6	Indicates frame is to be truncated.
L2_TAG	7	The L2_TAG bit in FORWARD_FLAGS as set by L2AR.
L2_VID_EQUAL	8	Flag set by MCAST_VLAN to indicate if the MCAST_VLAN entry was equal to the egress VID for this frame.
MCAST_TAG	9	Tag bits from the MTable.
TX_PORT_Tag	11:10	Lookup in MOD_TX_PORT_TAG table by TX_PORT.
DST_PORT_Tag	21:12	Lookup in MOD_DST_PORT_TAG table by DST_PORT (the original Tx port of TX-mirrored frames).
L2_VLAN1_TX_Tagged	22	Lookup in MOD_L2_VLAN1_TX_TAGGED table by (DST_PORT, L2_VID1).
L2_VLAN2_TX_Tagged	23	Lookup in MOD_L2_VLAN2_TX_TAGGED table by (DST_PORT, L2_VID2).
MOD_FLAGS	47:23	MOD_FLAGS from L3AR.

When PAUSE=1b, the MOD_FLAGS portion of the key is overloaded with the PAUSE-related fields listed in Table 5-95 (all other MOD_FLAGS bits are zero for these frames).

Table 5-95 MOD_FLAGS definition for PAUSE Frames

Field	Bit(s)	Description
PAUSE_Tag	25:24	Lookup in MOD_TX_PORT_TAG table by TX_PORT. Can be used to configure class-based PAUSE format.
PAUSE_CBP_VEC	33:26	8-bit pause class vector for class-based PAUSE frames.

5.21.4.2 Modify Command Slices

The command slices are shown in Figure 5-48.

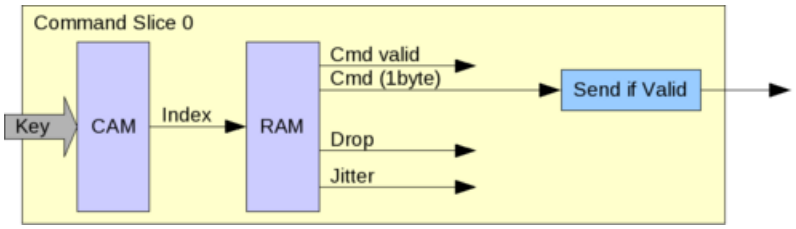


Figure 5-48 Egress Modification Command Slices

There are 16 command slices. Each command slice contains a 32x48-bit CAM (MOD_COMMAND_CAM) followed by a 32x16-bit RAM (MOD_COMMAND_RAM) and can produce at most one command. The key is searched for in all CAMs. If there is a match in a particular CAM block, the highest matching entry is selected and the corresponding RAM entry is read. The RAM entry contains the following fields:

- Command is valid
- Command
- Jitter
- Drop

The command is sent to the egress modifier only if it is valid.

The drop flag indicates to drop the frame. Any slice can force the frame to be dropped and the frame is forwarded if and only if all drop flags of all matching entries are set to zero.

The jitter value is accumulated across all 16 slices and used to control the jitter buffer in the egress serial modify.

Note: Not all commands are necessarily executed, the egress modifier skips the last commands and data if the frame terminates before the command is executed. An example would be a series of skip to byte 72 and then insert 4 bytes data. In this example, if the frame received as only 64 bytes, the insert instruction is not executed.

5.21.4.3 Modify Value Slices

The value slices are similar to the command slices with the addition of a data selection and data modification circuit as shown in Figure 5-49.

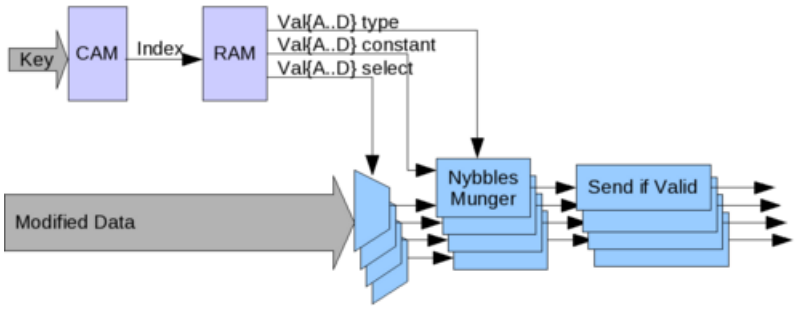


Figure 5-49 Egress Modification Value Slices

There are 15 value slices in total that can produce up to 60 bytes of frame data (4 bytes per slice, 56 bytes go to egress modifier, 4 bytes go to stats). Each value slice contains a 48-bit x 32-entry CAM (MOD_VALUE_CAM) followed by a 32 x 64-bit RAM (MOD_VALUE_RAM).The key is searched in all CAMs.

If there is a match in a particular CAM block, the highest matching entry is selected and the corresponding RAM entry is read. The RAM entry contains the following fields for each data byte:

- **DataSelect** (5 bits) — Identifies one of 24 source data channels.
- **Type** (3 bits) — Controls the validity and transformation of each value byte prior to being sent to serial modify.
- **Constant** (8 bits) — Constant data involved in the output transformation.

Data bytes are identified A through D where A is sent to modifier first and D is sent last. Bytes whose type is zero (invalid) are omitted from the serial value byte stream.

The DataSelect field selects one of 24 source fields for the corresponding value byte, as listed in Table 5-96.

Table 5-96 Egress Modification Data Select Fields

Field	Width	Data Select	Not PAUSE Frame	PAUSE Frame	Comment
W8[0]	8	0	{QOS.W4, QOS.ISL_PRI}	PAUSE_CBP_VEC	
W8[1]	8	1	{QOS.L3_DSCP, QOS.L3_CU}	Unused (0x0)	Packed as in TOS byte
W8[2]	8	2	MOD_MAP_DATA.W8A	MOD_MAP_DATA.W8A	
W8[3]	8	3	MOD_MAP_DATA.W8B	MOD_MAP_DATA.W8B	
W8[4]	8	4	{MOD_DATA.W4, 0x0}	Unused (0x0)	
W8[5]	8	5	MOD_DATA.W8A		
W8[6]	8	6	MOD_DATA.W8B		
W8[7]	8	7	MOD_DATA.W8C		

Table 5-96 Egress Modification Data Select Fields (Continued)

Field	Width	Data Select	Not PAUSE Frame	PAUSE Frame	Comment
W16[0]	16	16	{L2_VPRI1_TX, L2_VID1}	PAUSE_TME1	VPRI1/VID1 packed as in VLAN tag
W16[1]	16	17	{L2_VPRI2_TX, L2_VID2}	PAUSE_TME2	VPRI1/VID2 packed as in VLAN tag
W16[2]	16	18	{L2_VPRI1_TX, MOD_MAP_DATA.W12A}	{0x0, MOD_MAP_DATA.W12A}	Packet as in VLAN tag.
W16[3]	16	19	MOD_MAP_DATA.W16A	MOD_MAP_DATA.W16A	For PAUSE, indices other than TX_PORT are taken to be zero.
W16[4]	16	20	MOD_MAP_DATA.W16B	MOD_MAP_DATA.W16B	
W16[5]	16	21	MOD_MAP_DATA.W16C	MOD_MAP_DATA.W16C	
W16[6]	16	22	MOD_MAP_DATA.W16D	MOD_MAP_DATA.W16D	
W16[7]	16	23	MOD_MAP_DATA.W16E	MOD_MAP_DATA.W16E	
W16[8]	16	24	MOD_MAP_DATA.W16F	MOD_MAP_DATA.W16F	
W16[9]	16	25	MOD_DATA.W16A	Unused (0x0)	
W16[10]	16	26	MOD_DATA.W16B		
W16[11]	16	27	MOD_DATA.W16C		
W16[12]	16	28	MOD_DATA.W16D		
W16[13]	16	29	MOD_DATA.W16E		
W16[14]	16	30	MOD_DATA.W16F		
W16[15]	16	31	{MOD_DATA.W8D,MOD_DATA.W8E}		

For the 8-bit source values, each value is assigned directly:

```
Val{A..D} = W8[Val{A..D}_DataSelect]
```

For the 16-bit source values, the particular eight bits assigned depends on the field:

```
ValA = W16[ValA_DataSelect][15:8]
ValB = W16[ValB_DataSelect][7:0]
ValC = W16[ValC_DataSelect][15:8]
ValD = W16[ValD_DataSelect][7:0]
```

Thus, the most significant byte of the 16-bit fields are serialized first (and therefore transmitted first in the frame), before the least significant byte.

The selected byte is then transformed on a per-nibble basis using their respective constant and type fields.

The selected data byte is represented as 0xPQ and the constant byte from MOD_VALUE_RAM is represented as 0xXY.

```
Type
0: byte is invalid (not serialized)
1: 0xXY (constant)
2: 0xPQ (data)
3: 0xQP (data rotated)
4: 0xXQ (bottom nybble of data on bottom)
5: 0xXP (top nybble of data on bottom)
6: 0xPY (top nybble of data on top)
7: 0xQY (bottom nybble of data on top)
```

5.21.4.4 Transmit Disposition Flags

MODIFY produces a set of disposition flags that are used by the scheduler, frame processing pipeline and statistics. The disposition flags informs those units that the frame has been sent and what happens to the frame enabling those units to update their state accordingly.

The following disposition flags (TX.DISP_FLAGS) bits are defined:

0 = (RX_ERR)	Indicates the frame, on ingress, experienced an RX error (corresponding to RX.DISP != 0). For cut-through frames, this flag is set on any frame that also experiences an internal uncorrectable ECC error, as reported by ECC_ERR. However, RX.DISP is always reliably reported.
1 = (OOM)	The frame was truncated due to an out-of-memory error during frame reception.
2 = (LBS)	The frame was not transmitted due to the MCAST_LOOPBACK_SUPPRESS test.
3 = (TIMEOUT)	The frame was not transmitted due to the frame timeout mechanism.
4 = (ECC_ERR)	The frame experienced an uncorrectable ECC error in the array or scheduler or modify data memories. For cut-through frames, this error flag masks the reporting of a concurrent RX_ERR condition, if one applies. If the frame has not yet been started, it is dropped, with the exception of MOD_MAP_IDX12A. for this table, the frame is simply not dropped.
5 = (DROPPED)	The frame was not transmitted due to one or more of the previous conditions.
6 = (TX_FREE)	Identifies the frame as the last copy to be replicated on TX.PORT due to the MCAST_VLAN_TABLE and/or mirroring mechanisms.
7 = (RX_FREE)	Identifies the last copy of the source Rx frame to be replicated over all egress ports. Segments belonging to the source Rx frame are returned to the free pool.

5.21.4.5 Statistics Interface

The MODIFY unit provides two sets of information to statistics: an updated frame length, and a set of indexes.

The updated length is also sent to the scheduler shapers, to enable them to compute bandwidth accurately. The updated frame length is computed by accumulating the differences implied by each command to the original frame length assuming all commands are executed, and then sent to the statistics unit. The updated frame length reported is accurate for most normal modifications but not all of them. The condition under which the frame length is accurate are:

- The ingress frame is long enough that all the specified modification gets applied.
- Only the first 80 bytes of the ingress frame is modified, and not all of the first 80 bytes is deleted.
- MOD_MIN_LENGTH is set to <= 80 bytes.

The Tx indexes are derived from the last value slice in the modify stage and are sent to the statistics action resolution stage. These statistics value slices are identical to the other value slices in every respect, except their four value bytes are not serialized. Instead, they are mapped into two sets of 4-bit and 12-bit indices as follows (the IDX12A support a remap function):

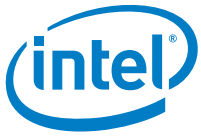
```
From slice 15:
TX.IDX4A  = ValA[7:4]
TX.IDX12A = MOD_MAP_IDX12A[{ValA[3:0], ValB[7:0]}]
TX.IDX4B  = ValC[7:4]
TX.IDX12B = {ValC[3:0], ValD}
```

Note: These statistics index slices are referenced as MOD_VALUE_{CAM,RAM}[15] in the register definitions. All source channels and output transformations are supported by these slices.

5.21.5 Serial Modify

Each port has a dedicated serial modify unit that processes egress frame data in a byte-serial manner (1.25 GHz for 10 Gb/s ports). The unit's operation is controlled by the serialized command stream produced by the modify slices. Commands are consumed and interpreted until they are exhausted or until reaching maximum modifiable size. Nine commands are supported:

- **SKIP** — Used to leave bytes of the frame unchanged. Takes a repeat count spanning 1..32 to enable skipping over many bytes of the frame with a single command. [zero value bytes required].
- **INSERT** — Inserts 1 to 16 bytes into the frame. [1..16 value bytes required].
- **DELETE** — Deletes 1 to 16 bytes of the frame. [zero value bytes required].
- **REPLACE** — Replaces 1 to 16 bytes of the frame. [1..16 value bytes required].
- **REPLACE_MASKED** — Replaces specific bits of a byte with bits from the value byte v, as specified by the dybble mask dybmask in the command byte. Each bit i of the output byte R is transformed from the input L as follows: `R[i] = (dybmask[i/2] ? v[i] : L[i])`. [1 value byte required].
- **DECREMENT** — Decrements a byte of the frame. 0x00 becomes 0xFF. [zero value bytes required].



- **DECREMENT_INSERT** — Inserts a new byte into the frame after first decrementing the new value by one. [one value byte required].
- **DECREMENT_REPLACE** — Replaces a byte of the frame after first decrementing the new value by one. [one value byte required].
- **CHECKSUM** — Updates a 2-byte ones-complement checksum such as used in IPv4 or TCP. This depends on accumulated changes to the frame resulting from other commands (each of the other command types carries a flag to indicate that the checksum should be accumulated), as well as an additional 2-byte delta provided on the value stream:

```
egress_checksum = (ingress_checksum + ingress_bytes_accumulated - egress_bytes_accumulated - delta)
```

The checksum accumulator state is cleared after this command or at end-of-frame. The accumulated bytes must form 16b byte-pairs that are 16b-aligned to the checksum.The checksum cannot be modified if it is the last 2 bytes of the frame or last 2 bytes of the maximum number of bytes modifiable. [2 value bytes required]

5.22 Statistics

The FM5000/FM6000 maintains 32-Kb frame counters that provide management software with statistical information about the state of the switch and of the network in general. In contrast to the FM2000 series and the FM4000 series, these frame counter resources are highly configurable, enabling software to track any frame properties that are of interest to the larger application context. The counters are flexible enough to support many of the network monitoring-related IETF RFCs, such as RFC 2819 (RMON), RFC 3273 (High-Capacity RMON), RMON 2613 (SMON), RFC 4502 (RMON2), RFC 2863 (Interfaces MIB), and RFC 4293 (MIB for IP).

5.22.1 Overview

The FM5000/FM6000's frame statistics counters are distributed over 16 identical 2-Kb-entry counter banks and 64 discrete counters. Each 2-Kb bank contains 2048 64-bit counter state elements supporting three individually-configurable modes:

- 64-bit frame count
- 64-bit byte count
- 64 x 1-bit saturating counters

Each discrete counter maintains both a 64-bit frame count and a 64-bit byte count.The byte count accumulates Rx frame lengths only, while the frame count counts either Rx or Tx frames however it is configured.

With these counter resources, any particular Rx frame can be counted by a maximum of 144 counters (16 bank counters, 64 discrete frame counters, and 64 discrete byte counters); any Tx frame can be counted by a maximum of 80 counters (16 bank counters and 64 discrete frame counters).

The frame properties tracked by the statistics counters are defined by the programming of a series of CAMs and RAMs in the statistics action resolution stage. These properties are mapped dynamically per frame based on information generated and selected by L2AR, egress modification, CM, and other fixed-function sources in the switch.

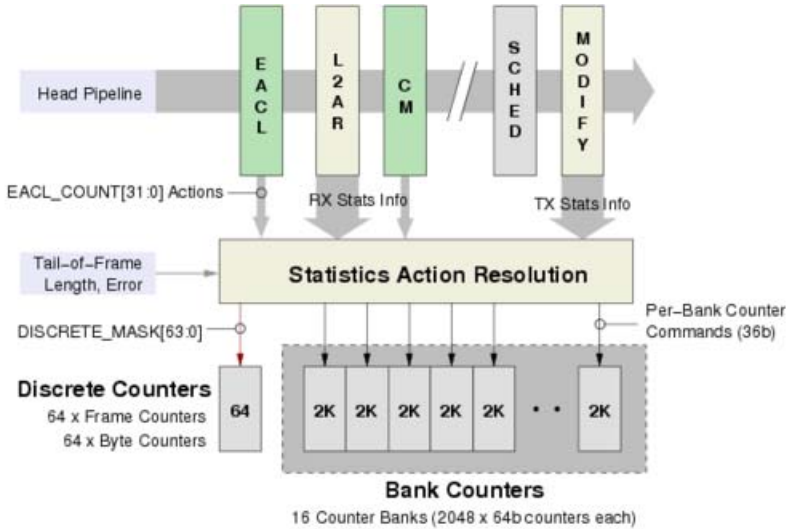


Figure 5-50 Statistics Counter Architecture

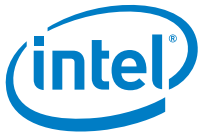
The determination of which counters to update on each frame takes place in a statistics action resolution (STATS_AR) classification stage. The STATS_AR stage applies a number of CAM and RAM mappings similar to those applied in the L2 and L3AR stages. It receives input key and index values from the frame processing pipeline and uses a CAM/RAM/MUX structure to map these inputs to 16 counter commands, one per counter bank, and a 64-bit discrete counter enable vector (DISCRETE_MASK[63:0]).

The statistics action resolution stage processes up to one ingress (Rx) and one egress (Tx) frame per cycle in parallel. An ingress frame is processed once it has been fully received by the switch. The frame's length and error status are combined with head information generated by L2AR and CM, and this bundle of data is presented to the statistics action resolution stage for processing.

On egress, the modify stage is responsible for generating key, index, disposition, and length information associated with each Tx frame it processes. Once the frame fully egresses, modify passes this information to the statistics unit for processing.

The statistics action resolution stage receives the following set of Rx and Tx frame properties:

- Rx Port (7 bits)
- Rx Frame Length (14 bits)
- Rx Disposition (2 bits)
- Rx Stats Key (43 bits)
- Rx Stats Flags (32 bits)
- Rx Egress ACL count actions (32 bits)
- Rx Stats Index data (71 bits)
  - 3 x 5-bit index channels (RX.IDX5{A,B,C})
  - 2 x 12-bit index channels (RX.IDX12{A,B})
  - 2 x 16-bit index channels (RX.IDX16{A,B})



- Tx Port (7 bits)
- Tx Frame Length (14 bits)
- Tx Replication Status (2 bits)
- Tx Disposition flags (6 bits)
- Tx Stats Key (40 bits)
- Tx Stats Index data (32 bits)
- 2 x 4-bit index channels (TX.IDX4{A,B})
- 2 x 12-bit index channels (TX.IDX12{A,B})

These fields are detailed further later in this section.

Software configuration of the statistics action resolution structure determines how this information is mapped to individual counters in the 16 2-Kb counter banks. This mapping operation results in generating 16 counter commands, one per bank, in the following form:

- **Mode** (2 bits) — Specifies one of three counter modes:

  0 = Do not count (bank remain idles, saving power).

  1 = Add the specified amount to the 64-bit counter state element.

  2 = Set bit number Amount[5:0] (one-bit saturating count mode).

- **Index** (11 bits) — Identifies the counter within the bank to update. The bottom bit of this index is referred to as the counter color for reasons that are explained later in this section.
- **Amount** (16 bits) — Specifies the amount to add to the specified counter (when Mode=1) or the bit index to set (when Mode=2). Generally, for Mode=1, this is either 1 (for frame counting) or else the Rx or Tx frame length (for byte counting). However, other amounts can be selected, such as pause times.

5.22.2 Action Resolution Structure

The architecture of the statistics action resolution stage is shown in Figure 5-51.

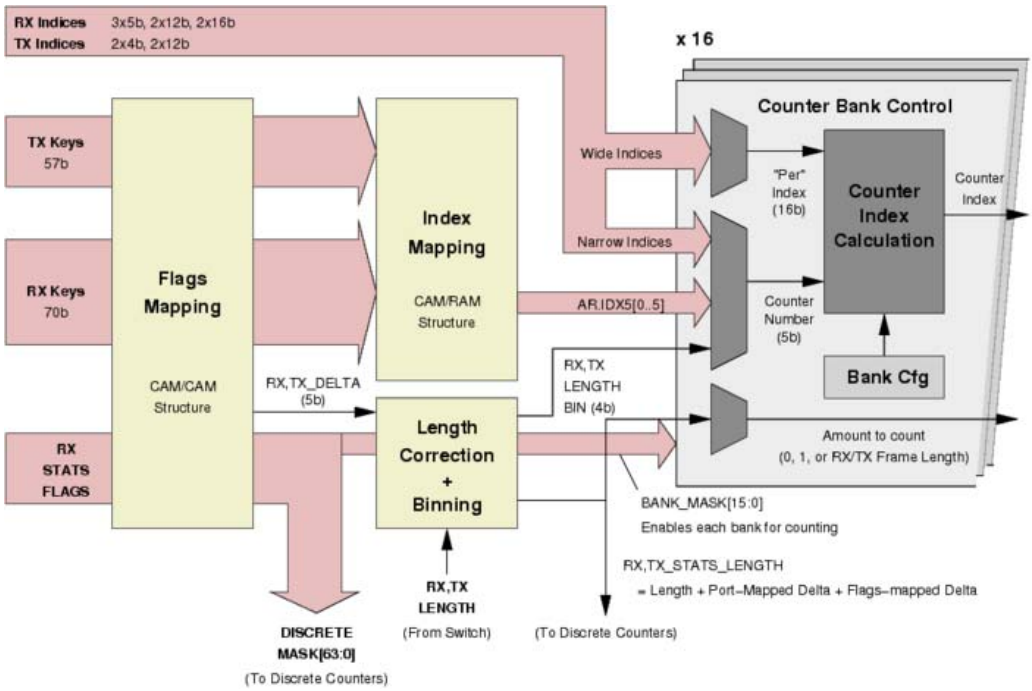
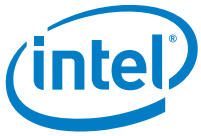


Figure 5-51 Statistics Action Resolution

It includes five functional structures:

- **Flags Mapping Stage** — Transforms a 64-bit STATS_FLAGS input to a 90-bit output STATS_FLAGS. The output flags enable the discrete and bank counters and specify Rx/Tx frame length correction terms.
- **Index Mapping Stage** — Generates six 5-bit AR.IDX5 indices from a set of CAM/RAM slices. These indices are available for muxing in each counter bank's index computation.
- **Length Correction and Binning** — Due to the inclusion or omission of ISL tags, preambles, CRCs, IEEE 1588 timestamps, etc., the frame length as seen by the switch data path might not be the appropriate length to count for statistics purposes. This stage applies two 5-bit signed delta amounts to each frame length to correct for these factors. The stage also bins the Rx and Tx frame lengths by comparing them to 16 configurable thresholds. The 4-bit bin values are available to the bank counter index calculation.
- **Counter Bank Control** — Each counter bank's index, counter mode, and amount to increment is computed based on static action resolution configuration.
- **Port Mapping** (not previously shown) — The RX_PORT and TX_PORT inputs are each mapped through 76-entry tables to provide 8-bit RX_PORT_TAG and TX_PORT_TAG CAM keys as well as RX_PORT_DELTA and TX_PORT_DELTA frame length adjustment terms.

These functions are specified in more detail in the Statistics Action Resolution section.



5.22.3 Per-Port Counters

It is expected that many applications calls for a large number of 64-bit per-port counters, as required by RMON, SMON, and other network statistics standards. The STATS_AR's bank index calculation function provides general support for such a port-based indexing mode. Effectively, the index of each per-port 64-bit counter is calculated as:

```
CounterIdx64 = 76 * CounterNum + Port
```

where port is either the Rx or Tx port number and CounterNum is a value spanning 0 to 25 taken from a selected 5-bit index channel. A bank configured in this manner provides up to 26 counters per Rx port or 25 per Tx port. The CounterNum source channels can come from L2AR (for Rx counters), from egress modify (for Tx counters), or from one of six AR.IDX5 channels assigned locally by the STATS_AR Index mapping function.

If all counter banks are allocated to per-port counters, this gives an absolute maximum of 416 counters per port, organized as independent groups of 16×26 counters. For each ingress or egress frame, one counter in each group can be incremented. For FM4000 compatibility, 10 of these groups are dedicated to per-port counting and the remaining six would be available for other purposes, such as per-VLAN counting.

5.22.4 Discrete Counters

In addition to its 16 banks of 2-Kb counters, the statistics stage also includes 64 discrete counters. Whereas the banked counters only provide a maximum per-frame counter parallelism of 16 (one counter per bank), all 64 discrete counters can increment in parallel per frame. Half of these counters have a fixed-function definition, and are controlled by the EACL count action.

The discrete counters are controlled by a 64-bit DISCRETE_MASK channel produced by the STATS_AR flags mapping function. Each bit of the DISCRETE_MASK controls a pair of counters: one counts frames and the other counts Rx frame length. If any bit DISCRETE_MASK[i] is set to 1b, the corresponding counter pair number i increments. If a discrete counter pair is configured to count Tx frames, the frame length counter value is meaningless.

The bottom 32 bits of DISCRETE_MASK are initialized by the 32-bit STATS_FLAGS channel produced by L2AR.The top 32 bits of DISCRETE_MASK are initialized by the EACL_COUNT[31:0] action result produced by the EACL stage. The DISCRETE_MASK is then transformed by the flags mapping stage, specified in the sections that follow.

The discrete counters are defined in the registers STATS_DISCRETE_COUNTER_FRAME and STATS_DISCRETE_COUNTER_BYTE. Each counter value is 64 bits wide with atomic access.

5.22.5 Counter Performance

Each 2-Kb bank counter can be incremented at a maximum rate of 550 MHz. Thus, per-port properties can be tracked by each bank counter with no performance corner-cases since the maximum frame rate of even a 40 GbE port (60 MHz) is well below this level. Throughout the FM5000/FM6000, bit two of the port number (RX.PORT[2] or TX.PORT[2]) is guaranteed to strictly alternate under maximum frame rate conditions, so in each 2-Kb statistics bank this bit is used to stripe counter updates across two 1-Kb sub-banks, with each sub-bank capable of sustaining a 550 MHz maximum update rate.

Switch-aggregate properties (such as activity on a particular VLAN) might see event rates exceeding 550 MHz, so pairs of counters must be allocated to count such properties in a fully-provisioned manner. Each counter is said to be colored in this case, where one counter tracks the aggregate property as seen on red ports while the other tracks the aggregate property as seen on blue ports. Including either RX.PORT[2] or TX.PORT[2] in the counter index calculation provides this color-alternation. To determine the total aggregate count, software must read and add the counts of the two colored counters.

Alternatively, aggregate counters can be left uncolored as long as the aggregate counter rate does not exceed 550 MHz. If these counters are mis-configured such that they are exposed to rates greater than 550 MHz, the entire frame processing pipeline is limited to 550 MHz and frame dropping can result.

Each discrete counter is fully-provisioned to count both frames and bytes at the maximum switch aggregate frame rate.

5.22.6 Port Mapping

As a preprocessing step before the other STATS_AR functions, the RX_PORT and TX_PORT values are mapped through 76-entry classification tables to provide 8-bit RX_PORT_TAG and TX_PORT_TAG fields for use as keys in the flags and index mapping CAMs. The bits in these tag fields are expected to encode properties such as:

- Per-port enables of particular counter groups (FM4000 STATS_CFG compatibility).
- Per-port selection between different input index fields (such counting by TC versus VPRI).

The tags are provided as a compression facility to reduce the number of CAM rules that would otherwise be needed.

The RX_PORT_MAP and TX_PORT_MAP tables also provide two 5-bit frame length correction terms, RxPortDelta and TxPortDelta, used by the length correction and binning function described later in this section.

5.22.7 Input Keys

The TCAMs in the index and flags mapping functions are provided the following 127-bit key:

Table 5-97 Statistics Input Keys

Field	Width	Description
RX_VALID	1	Indicates valid ingress frame information.
RX.PORT_TAG	8	Mapped ingress port tag.
RX.DISP	2	Ingress frame error status, encoded as follows: 00b = No error 01b = Framing error 10b = Unused 11b = CRC error
RX.KEY	59	Ingress stats key as asserted by L2AR.
RX.KEY.MOD_FLAGS	18	MOD_FLAGS as sent on the forward channel.
RX.KEY.STAT_FLAGS[15:0]	16	Statistics flag bits set by L2AR's SetFlags action. The upper 16 STATS_FLAGS[31:16] bits are not available in the TCAM key. These bits RX.KEY.STATS_FLAGS[15:0] 16 are only used for DISCRETE_MASK generation in the flags mapping stage.

Table 5-97 Statistics Input Keys (Continued)

Field	Width	Description
RX.KEY.DROP_CODE	8	Final drop code produced by L2AR's SetDropCode action.
RX.KEY.CPU_CODE	8	Final CPU code produced by L2AR's SetCpuCode action.
RX.KEY.RXMP	4	Rx shared memory partition number assigned to the frame.
RX.KEY.CM_FLAGS	5	Each bit encodes a CM drop condition that applies to the frame: 0 = Dropped by the global privileged watermark. 1 = Dropped by global SMP watermarks. 2 = Dropped by the Rx hog watermark. 3 = Dropped (to all destinations) by Tx hog watermarks. 4 = Dropped due to the shared memory running out of segment pointers.
TX.VALID	1	Indicates valid egress frame information.
TX.PORT_TAG	8	Mapped egress port tag.
TX.DISP_FLAGS	8	Egress frame transmit disposition. For details, see Section 5.21.4.4, "Transmit Disposition Flags".
TX.MOD_KEY	40	TCAM key from the modify slices. For details, see Section 5.21.4, "Modify Slices".

Note: Total key width = 127 bits.

5.22.8 Flags Mapping

The flags mapping function takes the 127-bit RX/TX STATS_AR frame key and a 64-bit STATS_FLAGS input produced by the head pipeline (associated with the Rx frame) and produces the 64-bit DISCRETE_MASK, a 16-bit BANK_MASK, plus two 5-bit length correction fields. This function is implemented as a two-stage CAM transformation, similar to each of the L2AR's two flags CAM stages, as shown in Figure 5-52.

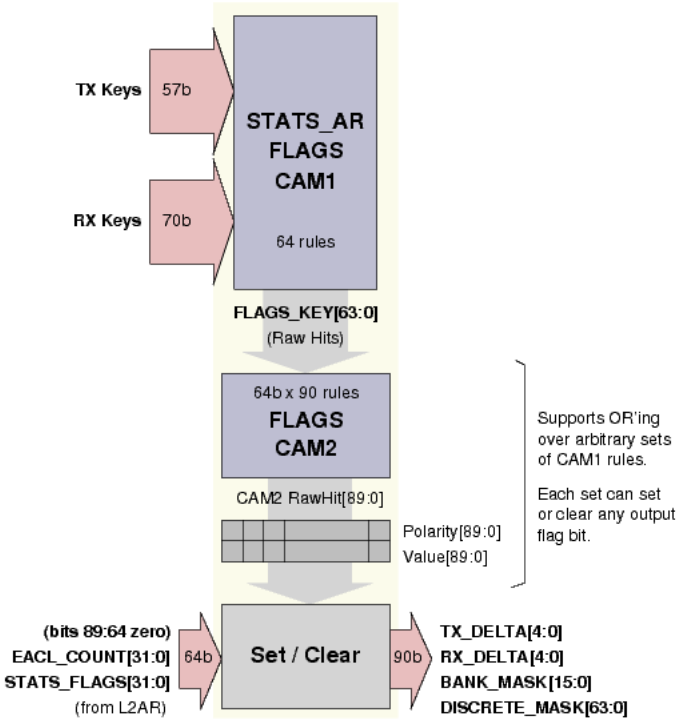


Figure 5-52 Statistics Action Resolution Flags Mapping

The two-stage CAM transformation enables sets of STATS_AR_FLAGS_CAM1 rules to be defined such that when any rule in the set matches, a particular set of STATS_FLAGS bits are set to a specified constant value. For example, one application would be to set DISCRETE_MASK[i1] when either STATS_AR_FLAGS_CAM1[r1] or STATS_AR_FLAGS_CAM1[r1+1] matches the input key.

5.22.9 Index Mapping

The index mapping function comprises six TCAM slices, each of which performs the following CAM/RAM transformation:

1. Matches the 127-bit STATS_AR frame key in its STATS_AR_IDX_CAM[slice] TCAM, producing the hit index (hit_idx) of the highest matching rule.
2. 2. Maps STATS_AR_IDX_RAM[slice,hit_idx] to provide the 5-bit value of AR.IDX5[slice].

If no rule in the slice matches, the corresponding AR.IDX5[slice] is invalid (power-gated) and prevents any bank that depends on the index from counting.

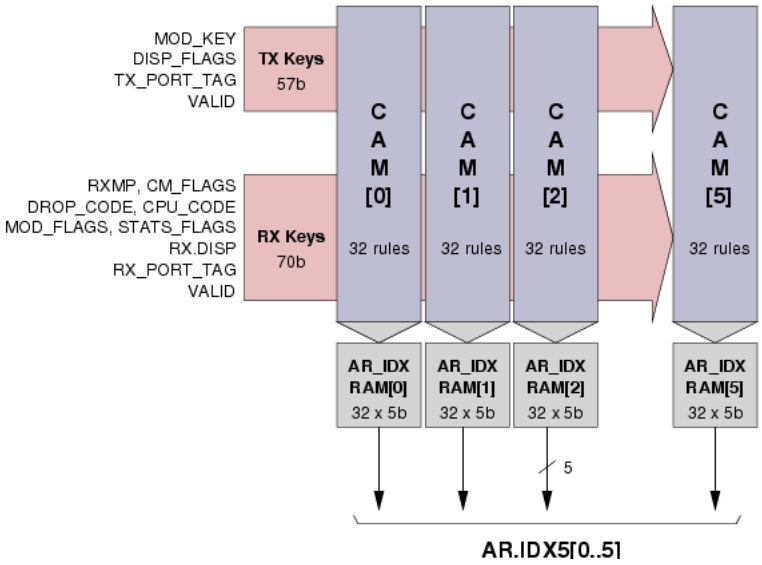


Figure 5-53 Statistics Action Resolution Index Mapping

5.22.10 Length Correction and Binning

All statistics frame length counters operate on RX and TX frame lengths that are corrected by two delta values: {RX,TX}.STATS_LENGTH = {RX,TX}.LENGTH + {RX,TX}_PORT_DELTA + {RX,TX}_DELTA, where

- **{RX,TX}.LENGTH** — Frame length as seen by the switch datapath.This might or might not include preamble bytes, CRC bytes, or IEEE 1588 timestamp bytes.
- **{RX,TX}_PORT_DELTA** — 5-bit signed correction values produced by the {RX,TX}_PORT_MAP tables.
- **{RX,TX}_DELTA** — 5-bit signed correction values produced by the flags mapping stage.

The resulting {RX,TX}.STATS_LENGTH values are then passed through a configurable range compare stage, similar to the MAPPER_LENGTH_COMPARE structure in the mapper, to recover the final bin to be used for size binning in Rx and Tx.

```
RX.LENGTH_BIN = 0
rxlength = RX.LENGTH + RX_DELTA;
for i=0..15
  if (STATS_AR_RX_LENGTH_COMPARE[i].LowerBound ≤ rxLength)
    RX.LENGTH_BIN = STATS_AR_RX_LENGTH_COMPARE[i].Bin

TX.LENGTH_BIN = 0
txLength = TX.LENGTH + TX_DELTA;
for i=0..15
  if (STATS_AR_TX_LENGTH_COMPARE[i].LowerBound ≤ txLength)
    TX.LENGTH_BIN = STATS_AR_TX_LENGTH_COMPARE[i].Bin
```

This enables these length values to be binned among 16 arbitrary ranges. In the bank index muxing stage, the binned length values are referred to as RX.LENGTH_BIN and TX.LENGTH_BIN, each 4 bits wide

5.22.11 Counter Bank Control

Each bank is configured by a static 34-bit BANK_CFG register. The fields in this register determine how the bank's counter index, counter mode, and increment amount is calculated.

Table 5-98 Statistics Bank Profile

Field	Width	Description
CounterType	1	Identifies the counter entry type: 0b = 64-bit rollover counter. 1b = 64 x 1-bit saturating counter.
Amount	2	Determines the amount to add to the indexed counter: 00b = Add 0 (do not count case). 01b = Add 1 (64-bit frame counter or 1-bit saturating counter). 10b = Add RX.LENGTH + RX_DELTA (64-bit byte counter). 11b = Add TX.LENGTH + TX_DELTA (64-bit byte counter).
Select_CounterNum	4	Mux select for up to 16 CounterNum channels described in the sections that follow.
Select_PerChannel	3	Mux select for up to eight PerChannels described in the sections that follow.
PerLowBit	4	Parameters relating to the counter index calculations as described in Section 5.22.12.
PerHighBit	4	
PerSpanMantissa	5	
PerSpanExponent	4	
ColorCase	2	
BankID	6	

Each bank i (for i=0..15) is enabled for counting by the BANK_MASK[i] bit generated by the flags mapping stage. If BANK_MASK[i]=0b, the configuration in BANK_CFG[i] has no effect and the bank remains inactive (power-gated).

The calculation of the counter index is complex due to the many different modes and counter types that are supported. The section that follows details the index generation function.

5.22.12 Counter Index Generation

Depending on configuration, selected input indices might identify individual bits in a 2-Kb bank (saturating 1-bit counter mode) or they might span ranges as large as multiple 2-Kb banks (such as to implement per-VLAN counters). Further complicating the definition are coloring considerations and optimizations to keep the per-bank index computation logic to a minimum.

All indices listed in Table 5-99 are phrased in relation to a generalized 22-bit Counter ID. The following quantities are involved in the counter ID computation.

Table 5-99 Statistics Counter ID Quantities

Parameter	Width	Description
CounterNum	5	A counter number corresponding to the property being counted. For example, octets belonging to a particular ISL_PRI.
PerChannel	16	The channel representing the quantity by which CounterNum is counted per. For example, PerChannel might be RX.PORT, in which case the assigned counter numbers are counted per rx port.
PerLowBit PerHighBit	4 + 4	Generally, the numeric per index field to be used for the counter ID calculation might be embedded in the PerChannel. These bit indices identify the appropriate slice as PerChannel[PerHighBit:PerLowBit]. A constant zero per index might be encoded by setting PerHighBit < PerLowBit.
PerSpan	5 + 4	The meaningful numeric span of the per index, represented in floating point form as a 5-bit mantissa and a 4-bit exponent. For example, when PerChannel is RX.PORT or TX.PORT, PerSpan is 76 (PerSpanMantissa=19, PerSpanExponent=2).
ColorCase	2	Identifies the coloring directive associated with the counter. Determines how the PerChannel is transformed for coloring (as detailed by the Decolor() function) and what PerColor value, if any, to associate with the channel. Four cases are defined as follows: 00b = Do not color 01b = Explicit by RX.PORT 10b = Explicit by TX.PORT 11b = Implicit (only meaningful when PerChannel is RX.PORT or TX.PORT) Implicit is normally used for per-port counters while explicit is used for aggregate counters (such as VLAN counters where ingress counters would use explicit RX.PORT and egress uses explicit TX.PORT).
PerColor	1	Either RX.PORT[2] or TX.PORT[2], depending on ColorCase. Not relevant if ColorCase is zero. This is an implied parameter derived by ColorCase. For example, it is not explicitly set in the bank's configuration.
CounterType	1	Identifies the counter type (64b rollover versus 1b saturating), as previously defined.

In a simplified sense, the counter ID is derived by a simple calculation:

```
CounterID = CounterNum * PerSpan + PerIdx
```

Unfortunately, the actual hardware calculation is not so simple. Coloring, field encodings, and counter type considerations make the computation considerably more complicated. First, a PerID is calculated as follows:

```
PerID = Decolor(PerChannel)[PerHighBit:PerLowBit]
```

The Decolor() transformation extracts the coloring port bit (bit 2) from RX.PORT or TX.PORT if applicable:

```
if ColorCase == 3 // For PerChannel == RX.PORT or TX.PORT
  Decolor(PerChannel) = {PerChannel[15:3],PerChannel[1:0]}
else
  Decolor(PerChannel) = PerChannel
```

Once decolored, an RX.PORT PerChannel is handled as ColorCase=1 (color by RX.PORT) and a TX.PORT PerChannel is handled as ColorCase=2 (color by TX.PORT) in the same manner as any other PerChannel selection. The ColorCase=3 handling should only be selected when PerChannel is either RX.PORT or TX.PORT.

Next, PerIdx64 (16b) and BitIdx (6b) quantities are calculated in a manner dependent on ColorCase and CounterType.

ColorCase>0?	CounterType	PerIdx64	BitIdx
0	0	PerID[15:0]	No
0	1	PerID[15:6]	PerID[5:0]
1	0	{PerID[14:0], PerColor}	No
1	1	{PerID[15:6], PerColor}	PerID[5:0]

Finally, BankID (5b) and CounterIdx64 (11b) quantities are calculated as:

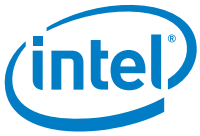
```
CounterID64 = (PerSpan * CounterNum) + PerIdx64
BankID = CounterID64[15:11]
CounterIdx64 = CounterID64[10:0]
```

The conceptual counter ID can be considered to be the 3-tuple (BankID, CounterIdx64, BitIdx), where BitIdx is only relevant for CounterType =1b. The identified counter is incremented so long as BankID matches BANK_CFG[bank].BankID.

As an example, to implement per-port counters, a bank's configuration parameters are configured as follows:

- **Select_PerChannel** — Set to RX.PORT or TX.PORT as appropriate.
- **Select_CounterNum** — Chosen as desired from one of the 5-bit {RX,TX,AR}.IDX5 channels.
- **PerSpanMantissa** — Set to 19.
- **PerSpanExponent** — Set to 2.
- **ColorCase** — Set to 3.
- **CounterType** — Set to 0.
- **BankID** — Set to 0.

Note: If any input channel selected by the bank configuration is invalid (power-gated), the bank counter index is also invalid and no counter in the bank counts the frame.



5.22.13 Bank Index Muxing

All counter banks are identical and select their input CounterNum and PerChannel input indices based on static bank configuration from a pool of all indices available. However, due to implementation cost considerations, only specific subsets of indices are available to each bank as detailed in the sections that follow.

5.22.13.1 CounterNum Channel Sources

The CounterNum channel mux pathways are listed in Table 5-100. The select encodings are uniform over all banks, but certain select cases are invalid for specific banks. Selecting an invalid choice for a particular bank result in a zero CounterNum value.

Table 5-100 Counter Channel MUX Pathways

Select#	Source	Banks 0-to-15
0	AX.IDX[0]	Yes
1	AX.IDX[1]	Yes
2	AX.IDX[2]	Yes
3	AX.IDX[3]	Yes
4	AX.IDX[4]	Yes
5	AX.IDX[5]	Yes
6	RX_LENGTH_BIN	Yes
7	RX.IDX5A	Yes
8	RX.IDX5B	Yes
9	RX.IDX5C	Yes
10	RX.IDX12A	Yes
11	TX_LENGTH_BIN	Yes
12	TX.IDX4A	Yes
13	TX.IDX4B	Yes
14	TX.IDX12A	Yes
15	TX.IDX12B	Yes

5.22.13.2 Per-Index Channel Sources

Mux pathways for the PerChannel source channels used in each bank's Counter ID calculation are listed Table 5-101.

Table 5-101 Counter Source Channels

Select#	Source	Banks 0-to-19
0	RX.PORT	Yes
1	TX.PORT	Yes
2	RX.IDX12A	Yes
3	RX.IDX12B	Yes
4	RX.IDX16A	Yes
5	RX.IDX16B	Yes
6	TX.IDX12A	Yes
7	TX.IDX12B	Yes

Note: This matrix becomes sparser as the design cost is better understood.

5.22.14 Atomicity

Unlike FM4000, all management accesses to the FM5000/FM6000's counters are atomic with respect to the 64 bits of each counter. All 16 banks share a single 64-bit cached read register with the same read semantics as other multi-word register caches elsewhere in the FM5000/FM6000. Specifically, each time management reads from an address that does not match the cached counter's address or else from the address of the cache entry's least significant word, the cache is refilled and the requested word is returned. Otherwise, the cached word is returned directly without reading the counter state.

5.22.15 Clearing Counters

To support selective clearing of 1-bit saturating counter state, the statistics bank counters have Clear-on-Write-1 (CW1) access semantics. A write with a non-zero value clears the counter bits corresponding to the 1b's in the value written. A write of 0b has no effect.

6.0 Ethernet Port Logic (EPL)

There are 24 EPL blocks in the FM5000/FM6000, each containing four SerDes lanes and four Media Access Controllers (MACs). One MAC is 40 GbE capable; the other three MACs are 10 GbE capable. For more details, see Section 11.5, "EPL Blocks".

6.1 Overview

Each EPL includes four MACs and four SerDes lanes for external connections (see Figure 6-1). SerDes lanes are numbered A through D; MACs and their corresponding channels are numbered 0 through 3. Each MAC is connected to the internal ports of the switch fabric through a 1.25 GHz x8-bit channel.

The switch supports 24 EPLs or 96 SerDes lanes that can support up to 72 Ethernet ports by using port mapping (see Section 6.2.2)

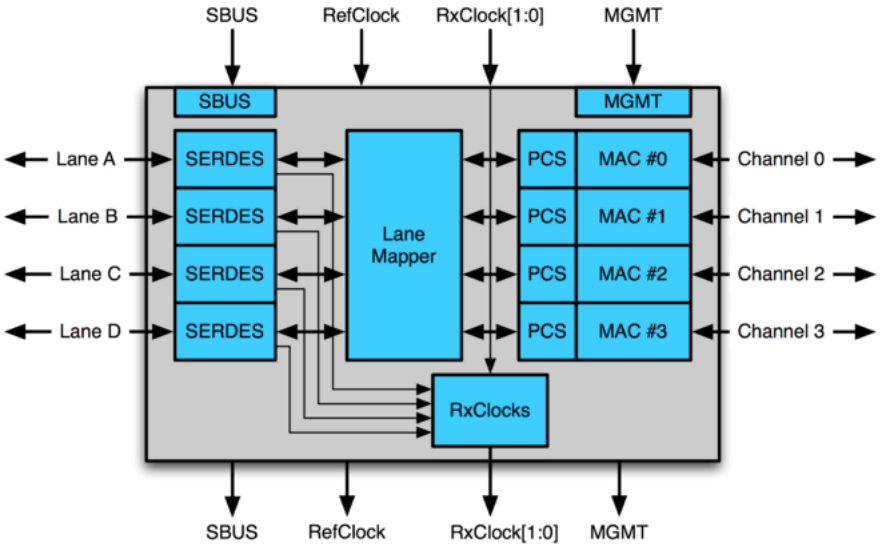


Figure 6-1 EPL Block Diagram

The EPL includes the following elements:

SerDes

- Physical interface including the following functions:
  - Serial transmit and receive
  - Symbol locking
  - Built-in Self Test (BIST)
  - 8b/10b encoding and decoding
  - Lane polarity reversal

PCS

- Reconciliation layer which includes the following functions:
  - Lane alignment
  - Lane ordering reversal
  - Frame boundary decoding
  - 64/66 encoding and decoding
- Clause 73, 37 and SGMII auto-negotiation

MAC

- Packet receive and transmit.
- CRC validation and CRC computation
- Programmable handling of preamble

Management interface

- SBUS interface-to-SPICO micro-controller
- Management interface to processor

Clocking

- One reference clock to SerDes (156.25 MHz)
- Two recovered receive clocks

EPLs are daisy-chained together for management purposes (see Figure 6-2).

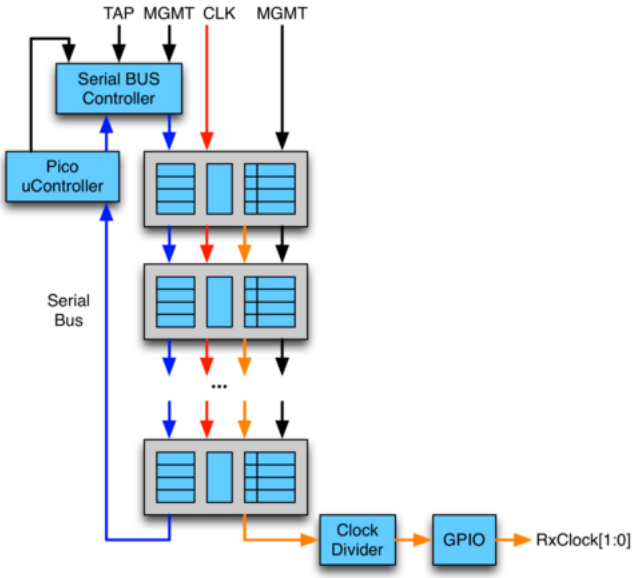


Figure 6-2 EPL Management

There are two management accesses:

- Memory mapped for common Ethernet port registers.
- Serial bus for less frequently used registers.
  - The serial bus is accessible via the SBUS controller (see LSM).
  - A micro-controller is also present in the chain to perform SerDes management.

6.2 Port Mapping

6.2.1 Port Numbering

At the chip level, the internal ports are numbered 0 through 75 and attached to the various interfaces; EPLs, MSB and PCIe. The 24 EPLs are geometrically split odd/even with odd on right and even on left.

Twelve of the EPLs (EPL[13..24], independent EPL known as EPL4) are fully connected to the crossbars and operate independently of each other, whereas the remaining twelve EPLs (EPL[1..12], paired EPL known as EPL8) are arranged in pairs, in which an odd EPL is paired with an even EPL (see Figure 6-3).

EPL4 is often named for the 12 independent EPLs and EPL8 often named for the remaining 12 paired EPLs.

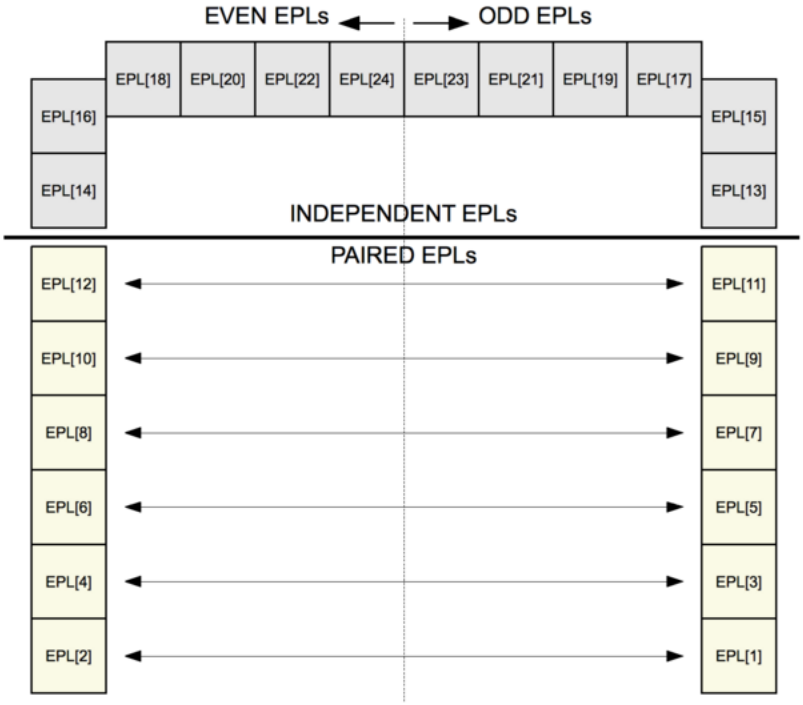


Figure 6-3 EPL Floor Plan (Bottom View; Pin Facing)

The exact mapping of physical interfaces at the chip level to internal ports is listed in Section 6.2.2.

6.2.2 Port Mapping Using the Channel

To connect with the internal switch fabric, an independent EPL or a paired EPL has four channels [0..3] to connect via a crossbar to the internal switch fabric. (See Table 6-1 and Figure 6-4).

Table 6-1 FM5000/FM6000 Port Mapping

Interface	Internal Port	Interface	Internal Port
PCIe host port	0	EBI host port (MSB)	1
Reserved	2	Loopback	3
EPL[1..2], Channel #0	4	EPL[1..2], Channel #1	5
EPL[1..2], Channel #2	6	EPL[1..2], Channel #3	7
EPL[3..4], Channel #0	8	EPL[3..4], Channel #1	9
EPL[3..4], Channel #2	10	EPL[3..4], Channel #3	11
EPL[5..6], Channel #0	12	EPL[5..6], Channel #1	13
EPL[5..6], Channel #2	14	EPL[5..6], Channel #3	15
EPL[7..8], Channel #0	16	EPL[7..8], Channel #1	17
EPL[7..8], Channel #2	18	EPL[7..8], Channel #3	19

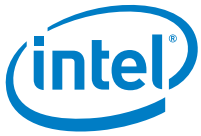


Table 6-1 FM5000/FM6000 Port Mapping (Continued)

Interface	Internal Port	Interface	Internal Port
EPL[9..10], Channel #0	44	EPL[9..10], Channel #1	45
EPL[9..10], Channel #2	46	EPL[9..10], Channel #3	47
EPL[11..12], Channel #0	48	EPL[11..12], Channel #1	49
EPL[11..12], Channel #2	50	EPL[11..12], Channel #3	51
EPL[13], Channel #0	52	EPL[13], Channel #1	53
EPL[13], Channel #2	54	EPL[13], Channel #3	55
EPL[14], Channel #0	40	EPL[14], Channel #1	41
EPL[14], Channel #2	42	EPL[14], Channel #3	43
EPL[15], Channel #0	56	EPL[15], Channel #1	57
EPL[15], Channel #2	58	EPL[15], Channel #3	59
EPL[16], Channel #0	20	EPL[16], Channel #1	21
EPL[16], Channel #2	22	EPL[17], Channel #3	23
EPL[17], Channel #0	24	EPL[18], Channel #1	25
EPL[17], Channel #2	26	EPL[17], Channel #3	27
EPL[18], Channel #0	28	EPL[18], Channel #1	29
EPL[18], Channel #2	30	EPL[18], Channel #3	31
EPL[19], Channel #0	32	EPL[19], Channel #1	33
EPL[19], Channel #2	34	EPL[19], Channel #3	35
EPL[20], Channel #0	36	EPL[20], Channel #1	37
EPL[20], Channel #2	38	EPL[20], Channel #3	39
EPL[21], Channel #0	60	EPL[21], Channel #1	61
EPL[21], Channel #2	62	EPL[21], Channel #3	63
EPL[22], Channel #0	64	EPL[22], Channel #1	65
EPL[22], Channel #2	66	EPL[22], Channel #3	67
EPL[23], Channel #0	68	EPL[23], Channel #1	69
EPL[23], Channel #2	70	EPL[23], Channel #3	71
EPL[24], Channel #0	72	EPL[24], Channel #1	73
EPL[24], Channel #2	74	EPL[24], Channel #3	75

Each channel interface consists of Tx/Rx lanes as a path to the crossbar and supports a maximum data rate of 10 Gb/s. Each lane contains two differential pairs in two directions: Tx pair and Rx pair. Each pair is an AC-coupled differential pair in an opposite polarity of wires: positive wire and negative wire.

Each channel is mapped to an internal port number of the switch fabric, but can be flipped within an EPL as described in Section 6.2.3.

Vice versa, each EPL offers four outbound SerDes lanes [A..D] to an external Physical Media Access (PMA) or Physical Media Device (PMD) called PHY or SFP+ modules. The EPL implements many features of multiple network standard interfaces, and the EPL lanes can perform and comply with transmitting or receiving an electrical interface of the defined standard. See Section 6.3 for a detailed description of supported operating modes.

6.2.3 Default Lane Reversal and Polarity Inversion Inside the Package

For some of the EPLs, the inter-connection from the EPL channels on the internal crossbar to the external output SerDes lane of the EPL on package are reversed by default, resulting in the following external SerDes lane to EPL internal channel mapping table (see Table 6-2). For example, EPL[1]'s physical connection to the package pins is reversed resulting in lanes A, B, C, D matching channels 3, 2, 1, 0 instead of 0, 1, 2, 3. This default configuration can be changed in FM5000/FM6000 platform driver code to take effect of the default setting the EPL register.

Note: The lane crossover (lane swapping) is not supported among these four lanes such as A-C and B-D as well as lane A-B and lane C-D.

Randomly swapping those four lanes within an EPL is not possible. For example, A:C:B:D or D:A:B:C, or C:A:D:B.

However, the polarity of the positive and negative of each lane can be inverted through a register setting change. The FM5000/FM6000 platform API driver code change in Platform_port.c is required to make the corresponding register effective.

Platform_port.c in FM5000/FM6000 platform API driver code:

```
#define RV_NONE      FM_LANE_REVERSE_NONE
#define RV_TXLN      FM_LANE_REVERSE_TX
#define RV_RXTX      FM_LANE_REVERSE_RX_TX
#define INVERT_RX    FM_POLARITY_INVERT_RX
#define INVERT_NO    FM_POLARITY_INVERT_NONE
```

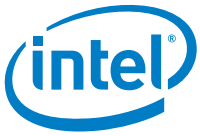


Table 6-2 EPL Channel Mapping

Interface	Flipped?	Lane-to-Channel Mapping
EPL[1]	Yes	Lanes {A,B,C,D} = Channels {3,2,1,0}
EPL[2]	Yes	Lanes {A,B,C,D} = Channels {3,2,1,0}
EPL[3]	No	Lanes {A,B,C,D} = Channels {0,1,2,3}
EPL[4]	Yes	Lanes {A,B,C,D} = Channels {3,2,1,0}
EPL[5]	No	Lanes {A,B,C,D} = Channels {0,1,2,3}
EPL[6]	No	Lanes {A,B,C,D} = Channels {0,1,2,3}
EPL[7]	No	Lanes {A,B,C,D} = Channels {0,1,2,3}
EPL[8]	Yes	Lanes {A,B,C,D} = Channels {3,2,1,0}
EPL[9]	No	Lanes {A,B,C,D} = Channels {0,1,2,3}
EPL[10]	No	Lanes {A,B,C,D} = Channels {0,1,2,3}
EPL[11]	No	Lanes {A,B,C,D} = Channels {0,1,2,3}
EPL[12]	Yes	Lanes {A,B,C,D} = Channels {3,2,1,0}
EPL[13]	Yes	Lanes {A,B,C,D} = Channels {3,2,1,0}
EPL[14]	No	Lanes {A,B,C,D} = Channels {0,1,2,3}
EPL[15]	No	Lanes {A,B,C,D} = Channels {0,1,2,3}
EPL[16]	No	Lanes {A,B,C,D} = Channels {0,1,2,3}
EPL[17]	Yes	Lanes {A,B,C,D} = Channels {3,2,1,0}
EPL[18]	No	Lanes {A,B,C,D} = Channels {0,1,2,3}
EPL[19]	No	Lanes {A,B,C,D} = Channels {0,1,2,3}
EPL[20]	Yes	Lanes {A,B,C,D} = Channels {3,2,1,0}
EPL[21]	No	Lanes {A,B,C,D} = Channels {0,1,2,3}
EPL[22]	Yes	Lanes {A,B,C,D} = Channels {3,2,1,0}
EPL[23]	No	Lanes {A,B,C,D} = Channels {0,1,2,3}
EPL[24]	Yes	Lanes {A,B,C,D} = Channels {3,2,1,0}

6.2.4 EPL Port Pairing

As shown in Figure 6-3, the EPLs 1 through 12 are arranged in pairs where each pair shares the same four internal ports to the crossbar. Figure 6-4 shows how the internal port sharing is done.

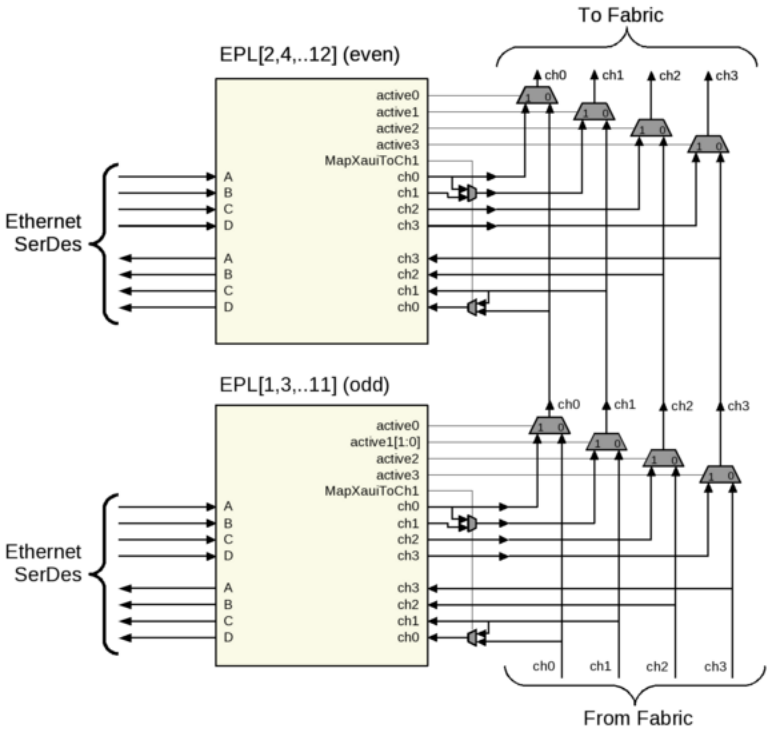


Figure 6-4 EPL Pairing

This arrangement supports the following configurations:

- Active/standby of 40 GbE, 4 x 10 GbE or 4 x 1 GbE
  - An EPL in active mode can forward packets to the switch fabric or receive packets from the switch fabric. An EPL in standby mode can establish and maintain an active link with its link partner but cannot receive nor transmit any packet. The change from active to standby is under software control using a selector in the EPL to control the multiplexers installed between the EPLs and the crossbars.
  - The active/standby selection can be set independently for the 4 x 10 GbE or 4 x 1 GbE.
- Support of two independent XAUI ports

  Note: XAUI is only available on channel 0 of EPL4. Software must appropriately set the mapping.

The mapping of each pair of channels is done through a special channel mapper controlled by a 6-bit code coming from an EPL. The internal channels of a paired EPL can be swapped dynamically without a system soft reset. For example, EPL[1] channel #0 connects to a 10GBASE-T PHY and EPL[2] channel #0 connects to a 10 GbE SFP+ module. Enabling the channel swapping between EPL[1] and EPL[2] at run-time by a register setting is possible and takes immediate affect without a performing a system soft reset.

Table 6-3 EPL8 Channel Mapping

Active				Channel 0	Channel 1	Channel2	Channel 3	Note
3	2	1	0					
0	x	x	x	Bypass	x	x	x	
1	x	x	x	Active	x	x	x	
x	0	x	x	x	Bypass	x	x	
x	1	x	x	x	Active	x	x	See note that follows
x	x	0	x	x	x	Bypass	x	
x	x	1	x	x	x	Active	x	
x	x	x	0	x	x	x	Bypass	
x	x	x	1	x	x	x	Active	

Note:

- MapXauiToCh1 = 0b:
  - Map EPL ch1 to fabric ch1
  - Map fabric ch0 to EPL ch0
- MapXauiToCh1 = 1b:
  - Map EPL ch0 to fabric ch1
  - Map fabric ch1 to EPL ch0
  - Behavior is undefined if set to 1b for other mode than XAUI

6.3 Mode of Operation

Each EPL supports up to four MACs, four SerDes lanes output and four channels to the internal fabric. The switch supports 24 EPLs for a total of 96 SerDes lanes output. However, the number of channels to the internal fabric 72 Ethernet ports are limited to 72 channels.

Mapping of MAC channels to the internal fabric Ethernet ports follow these rules:

- 48 MACs of 12 independent EPLs (EPL4) have full connectivity to the fabric for a total of 48 channels.
- 48 MACs shared among 12 paired EPLs (EPL8) connect to the internal fabric for a total of 24 channels.
- Optional SGMII ports can operate in 1 GbE or 2.5 GbE mode. In 2.5 GbE mode, the bit rate is 3.125 Gb/s.

Table 6-4 EPL Modes

Modes	MAC#0	MAC #1	MAC #2	MAC #3	Normal SerDes Bit Rate	Encoding
	A	B	C	D		
SGMII[1]	Yes	Yes	Yes	Yes	1.25 Gb/s or 3.125 Gb/s	8b/10b
1000BASE-nn 2500BASE-nn	Yes	Yes	Yes	Yes	1.25 Gb/s or 3.125 Gb/s	8b/10b
XAUI[1]	Yes A:B:C:D or D:C:B:A	No	No	No	4 x 3.125 Gb/s	8b/10b
10GBASE-nn, SFP+	Yes	Yes	Yes	Yes	10.3125 Gb/s	64b/66b
40GBASE-R	Yes A:B:C:D or D:C:B:A	No	No	No	4 x 10.3125 Gb/s	64b/66b
24GBASE-R Rate is 24.242 Gb/s	Yes A:B:C:D or D:C:B:A	No	No	No	4 x 6.25 Gb/s	64b/66b
20GBASE-R	Yes A:B or B:A	No	No	No	2 x 10.3125 Gb/s	64b/66b

1. Optional.

If the port uses more than one lane, the lane ordering (normal A-B-C-D or reverse D-C-B-A) is programmable on a per-port direction (see Section 6.2.3). The lane polarity (+/- or -/+) is always programmable on a per-port, per-lane direction (see Section 6.2.3). The maximum speed on a specific MAC is always 10 GbE after decoding (or before encoding) except for MAC 0 which is 40 GbE capable. Also, if a single MAC handles more than one lane, each lane must be running at the same speed and come from the same link partner (the MACs themselves might run at different speed in respect to each other). Table 6-5 lists the lane speed for each lane.

Table 6-5 EPL Lane Speed

Mode	MAC #0	MAC #1	MAC #2	MAC #3
Four independent MACs	1.25-10.3125 Gb/s	1.25-10.3125 Gb/s	1.25-10.3125 Gb/s	1.25-10.3125 Gb/s
One MAC	1.25-3.125-10.3125 Gb/s			

The EPL is compliant to the standards listed in Table 6-6.

Table 6-6 EPL Standard Modes

Mode	#SerDes	Standard	Encoding
SGMII[1]	1	Cisco*	8b/10b
1000BASE-X	1	IEEE Clause 36,37	8b/10b
1000BASE-KX/2500BASE-KX	1	IEEE 802.3ap	8b/10b
XAUI[1]	4	IEEE 802.3ap	8b/10b
10GBASE-KX4	4	IEEE 802.3ap	8b/10b
10GBASE-CX4	4	IEEE 802.3ak	8b/10b
10GBASE-R (KR)	1	IEEE 802.3ap	64b/66b

Table 6-6 EPL Standard Modes (Continued)

Mode	#SerDes	Standard	Encoding
SFP+ Direct Attach Copper[2]	1	SFF 8431	64b/66b
SFP+ SFI Interface (10GBASE-SR)[3]	1	SFF 8431	64b/66b
20GBASE-R	2	IEEE 802.3ba (only first two lanes used)	64b/66b
40GBASE-KR4	4	IEEE 802.3ba	64b/66b
QSFP 5M Direct Attach (40GBASE-CR4)	4	IEEE 802.3ba	64b/66b
QSFP PMD Service Interface (40GBASE-SR4)	4	IEEE 802.3ba	64b/66b

1. Optional.
2. For SFP+ Direct Attach applications, Intel recommends using 24 gauge direct attach cables that are 5 meters or less.
3. Deterministic jitter generation is 150 mUI maximum verses SFI specification of 100 mUI maximum.

6.4 Reference Clock

The EPL requires one 156.25 MHz reference clock per group of 6 EPLs. This reference clock is suitable for 1.25 GbE, 3.125 GbE, 6.25 GbE and 10.3125 GbE. The arrangement is shown in Table 6-7.

Table 6-7 EPL Clock Mode

EPL Clock Source	EPLs					
ETH_REFCLK1	1	3	5	7	9	11
ETH_REFCLK2	2	4	6	8	10	12
ETH_REFCLK3	13	15	17	19	21	23
ETH_REFCLK4	14	16	18	20	22	24

Figure 6-5 shows the reference clock internal termination scheme. These resistors can be disabled by strapping LED_DATA[0]. The LED_DATA[0] pin is latched when CHIP_RESET_N is de-asserted, and has an internal pull down that defaults to internal termination. If this pin is externally pulled up (such as with a 5 KΩ resistor to VDD25), the REFCLK input has its internal termination disabled.

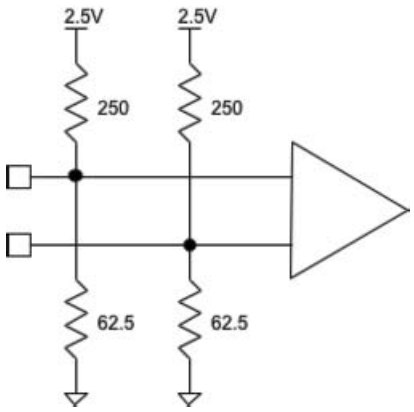


Figure 6-5 Ethernet Reference Clock Internal Termination

6.5 SerDes Characteristics

The SerDes operates at any of the following speeds:

- 10.3125 GbE, 64b/66b encoding
- 6.25 GbE, 64b/66b encoding
- 3.125 GbE, 8b/10b encoding
- 1.25 GbE, 8b/10b encoding

6.25 GbE is a non-standard mode of operation that can be used. For example, to create a 24.24 GbE 4-lane MLD link between FM5000/FM6000 devices.

The following is a list of SerDes characteristics:

- Programmable pre and post emphasis
- Decision Feedback Equalization (DFE)
- Symbol lock
- Eye measurement
- Auto adjustment via dedicated micro-controller
- KR DFE training
- Loopbacks

6.5.1 DFE Tuning and Emphasis

The receiver DFE tuning and transmit emphasis is done automatically for KR and done as part of the KR training sequence.

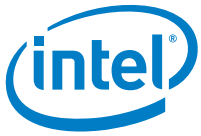
The DFE tuning and transmit emphasis is static for 1.25 Gb/s.

The DFE tuning is either static or dynamic for 3.125 GbE and 5.15625 GbE and 10.3125 GbE XFI.The dynamic tuning is done by the SPICO* micro-controller while static is done by software. The emphasis is always static for 3.125 GbE and 5.15625 GbE and it always done by software.

6.5.2 Pattern Generator/Comparator

Each SerDes lane has one pattern generator and comparator. The patterns supported are:

- Polynomial
  - PRBS: $x^{7}+x^{6}+1$ (ITU V.29)
  - PRBS: $x^{15}+x^{14}+1$ (ITU-T O.151)
  - PRBS: $x^{23}+x^{18}+1$ (ITU-T O.151)
  - PRBS: $x^{31}+x^{28}+1$
  - PRBS: $x^{11}+x^{9}+1$ (IEEE Std 802.3ap-2007)



  - — PRBS: x ^{9}+x ^{5}+1 (fiber channel)
- Fix pattern
  - — 10/20/40 bits

Both link partners must be configured with the same pattern for comparator to work properly. Once configured, the comparator is capable to self synchronize to the incoming data and also include a counter to count the number of errors detected. The software should first check synchronization has been detected and then reset the counter before monitor a bit error rate.

6.5.3 Loopbacks

The SerDes supports two loopbacks:

- Near loopback where the Tx serialized output is looped back into the Rx serial input.
- Far loopback where the received parallel data is looped back into the transmit parallel data.
  - — The data is placed in a small FIFO for clock boundary crossing but the Tx must be traceable to the same clock source as the link partner for this feature to operate properly.

6.5.4 Eye Measurement

The SerDes support eye measurement.

6.6 Recovered Clocks

The FM5000/FM600 offers support for up to two recovered clocks. The two recovered received clocks are daisy-chained from one EPL to the next, passed to two clock dividers at the end of the chain, and presented to two dedicated pins.

Within each EPL, two identical circuits enable clock selection. Each circuit consists of a 5:1 multiplexer to select one of the four local sources (one from each SerDes) or the clock of the adjacent EPL and a divide-by-2 or divide-by-4 to divide the local clock selected. The divider does not apply for pass-through clocks. If all EPLs are in pass-through, no recovered clock is available. The recovered clock from EPL is rippled through to the manageability module where the final dividers are located.

```
ClkOut = BitRate/20 x (M/N) / DIV (3.125G or 1.25GG, use divide-by-2)
ClkOut = BitRate/80 x (M/N) / DIV (5.15625G or 10.3125G, use divide-by-4)
```

Where M must be smaller than N.

For example, for 10.3125 GbE, if M=32, N=33 and DIV=15625, then:

10.3125 GbE/80 x (32/33) / 15625 => 8 KHz

6.7 Auto-Negotiation

The FM5000/FM6000 supports auto-negotiation as defined in clause 37 and clause 73 and the industry de-facto SGMII auto-negotiation mode. The clause 73 is a standalone layer while the clause 37/SGMII are attached to the PCS layer and controlled through the normal memory-mapped management interface.

6.7.1 Clause 73

Clause 73 uses a relatively low-frequency to encode auto-negotiation messages. Each message is 48 bits long and is encoded in a 106-bit frames using a Manchester encoding scheme (see Figure 6-6). These frames are exchanged between link partners before the link comes up because the speed and mode of operation is not determined yet. The clause 73 requires a 125 MHz reference clock.

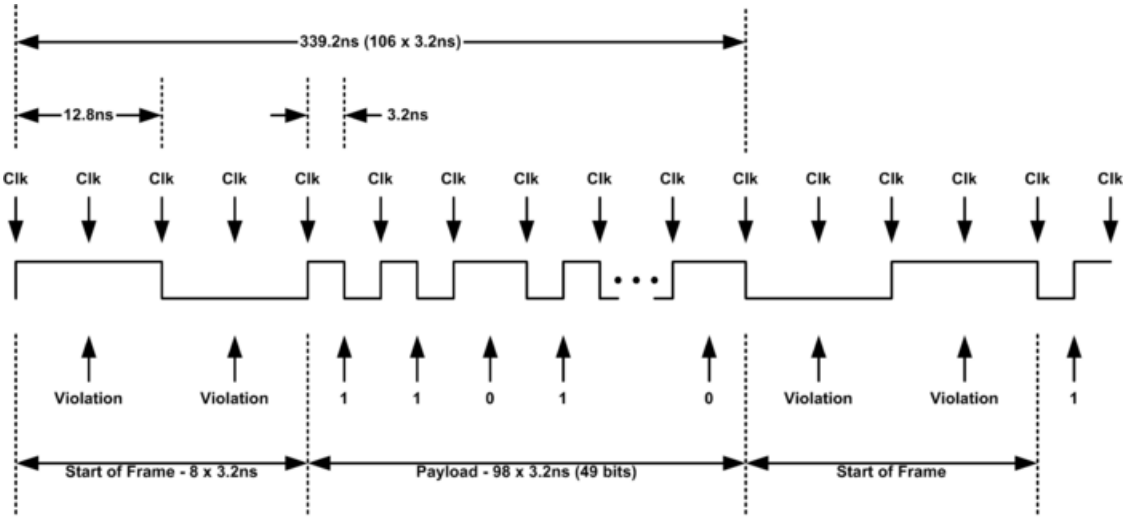
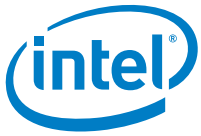


Figure 6-6 EPL Auto-Negotiation Clause 73

The auto-negotiation process is detailed in clause 73 and summarized here:

1. The AN state machine sends the mandatory control word frame repetitively setting the ACK bit to 0b. The control word includes an ID, which is a pseudo random number set by the hardware that is static for the duration of this negotiation. The control word includes an NP bit to indicate if the sender has more pages to send.
2. The AN expects to receive the mandatory control word and waits for three consecutive identical copies of the control word to be received.
3. The AN sends an acknowledge (ACK set to 1b) and echoed the ID from the link partner.
4. The AN waits for an ACK with an echoed ID equal to the one sent. The receiver checks for NP and if set enters the next page negotiation.
5. If neither the local end nor the link partner has more page to send, the AN completes.



6. If one or both has next pages to send, they do so at that point starting first by sending the ACK bit to 0b and then waiting for a valid next page before sending the next page. If one side as no next page to send, it sends the NULL page.

The control word format is listed in Table 6-8.

Table 6-8 EPL Clause 73 Control Word Format

Bits	Name	Transmitter	Receiver
4:0	Selector	Fixed (00001). Cannot be set by software.	Checked by hardware.
9:5	Echoed Nonce	Transmitter echoes received transmitted Nonce in this field once three valid identical CW are received.	Checked by hardware to be equal to the transmitted "transmitted Nonce" (if ACK is 1).
12:10	Pause Abilities	Set by software.	Pause abilities. Read and interpreted by software once negotiation exchange completes.
13	Fault	Set by software.	Fault at link partner. Read and interpreted by software once negotiation exchange is completes.
14	ACK	Software presets if needed and hardware sets at appropriate time.	Checked by hardware.
15	NP	Software presets if needed and hardware sets at appropriate time. Set by hardware once three valid identical CW are received.	Checked by hardware.
20:16	Tx Nonce	ID for this negotiation session. Set by hardware after each restart.	Link partner ID for this negotiation session.
45:21	Mode Abilities	Abilities of this transmitter. Set by software.	Abilities of link partner. Read and interpreted by software.
47:46	FEC Abilities	FEC abilities of this transmitter. Set by software.	FEC abilities of link partner. Read and interpreted by software.

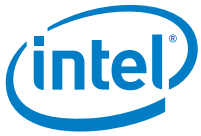
The next page format is listed in Table 6-9.

Table 6-9 EPL Clause 73 Next Page Format

Bits	Name	Transmitter	Receiver
10:0	Message	Set by software.	Read and interpreted by software once negotiation exchange completes.
11	Toggle	Set by software.	Checked by hardware to be toggling.
12	ACK2	Set by software.	Read and interpreted by software once negotiation exchange completes.
13	MP	Set by software.	Read and interpreted by software once negotiation exchange completes.
14	ACK	Software presets if needed and hardware follows once three valid identical CW are received.	Checked by hardware.
15	NP	Software presets if needed and hardware follows once three valid identical CW are received.	Checked by hardware.
47:16	Message	Set by software.	Read and interpreted by software.

The details of the algorithm are as follows:

1. Software sets the MAC in AN73 mode.
   - Software announces its local abilities by loading the control word into the AN_73_BASE_PAGE_TX register. The NextPage bit in the CW must be set if there are other pages to transmit.
   - Software enables AN, starts a timer and waits.
2. The switch exchanges the CW with its partner.
   - The switch transmits the AN_73_BASE_PAGE_TX as the control word repetitively with ACK bit to 0b.
   - The switch waits for three identical valid control words and stores them into AN_73_BASE_PAGE_RX.
   - When three identical valid control words are received, the switch echoes the NONCE received into its own CW message and sets the ACK bit to 1b. If the switch detects the NONCE received is equal to the NONCE transmitted, the switch posts a *An73TransmitDisable* interrupt and stops and waits for software intervention, which changes the NONCE to a new random value. If the NONCE is different, the switch waits for an ACK to be received.
   - When an ACK is received, the switch posts an *An73CompleteAcknowledge* interrupt.
3. The software detects the *An73CompleteAcknowledge* interrupt that indicates that a page has been received and can now read the AN_73_BASE_PAGE_RX register (or the AN_73_NEXT_PAGE_RX register. If there are next pages, the software must load the AN_73_NEXT_PAGE_TX register with the next page to send and set the TxNextPageLoaded bit. If there is no next page, go to step 6.
4. The switch exchanges the NP with its partner:
   - The switch detects the write to the TxNextPageLoaded bit, clears it, and starts to transmit the next page. It first transmits with the ACK bit set to 0b until the hardware receives three identical NextPage messages.
   - The switch detects three identical NextPage messages, then sets its ACK bit to 1b and stores the receive page into the AN_73_NEXT_PAGE_RX register.
   - The switch waits for three consecutive ACKs. Afterwards, the message in AN_73_NEXT_PAGE_RX is updated.
   - When an ACK is received, the switch posts an *An73CompleteAcknowledge* interrupt.
5. Steps 3 and 4 are repeated until all pages have been exchanged.
6. The switch posts an *An73AnGoodCheck* interrupt to indicate that the exchange of pages completed and then waits for the link to come up.
7. Software detects the interrupt and then applies the highest common denominator.
   - Software is responsible for programming SerDes at the proper speed, load the PCS mode negotiated into the AN73 PCS mode register and set the appropriate timer for this mode to sync.
8. The switch posts an *An73AnGood* when the link comes up in time or posts an *An73TransmitDisable* interrupt if the link didn't come up before the timer run out.
9. Software detects *An73AnGood* and now declares the AN completed. If it was an *An73TransmitDisable*, the state machine automatically returns to step 2.



6.7.2 Clause 37

Clause 37 is a sub-mode of the 1000BASE-X operating mode. It enables exchanging abilities with the remote device such as duplex mode, flow control or fault signaling.

The auto-negotiation process is detailed in clause 37 and summarized here:

1. The AN state-machine sends the mandatory control word frame repetitively setting the ACK bit to 0b. The control word includes an NP bit to indicate if the sender has more pages to send.
2. AN expects to receive the mandatory control word and waits for three consecutive, identical copies of the control word to be received.
3. AN sends an acknowledge (ACK set to 1b) and echoed the ID from the link partner. The message is sent for at least 10 ms.
4. AN waits for an ACK for at least 10 ms. The receiver then checks for NP and if set enters the next page negotiation.
5. If neither the local end nor the link partner has more pages to send, AN completes.
6. If one or both has next pages to send, they do so at that point starting first by setting the ACK bit to 0b and then waiting for a valid next page before sending the next page. If one side as no next page to send, it sends the NULL page.

The 1000BASE-X has the ability to enter into clause 37 either manually via software intervention or automatically upon link down detection or by detecting AN messages from remote devices. If programmed to start clause 37 upon link down, clause 37 announces a remote fault and maintains the announcement until the link is up. The remote fault is cleared when the link comes up and the handshake completes.

The control word format for clause 37 is listed in Table 6-10.

Table 6-10 EPL Clause 37 Control Word Format

Bits	Name	Transmitter	Receiver
4:0	Selector	Set by software.	Checked by software.
6:5	Duplex (set by software)	Duplex abilities. Read and interpreted by software once negotiation exchange completes.	
8:7	Pause	Set by software.	Pause abilities. Read and interpreted by software once negotiation exchange completes.
11:9	Reserved	Set by software.	Checked by software
13:12	Fault	Set by software or hardware.	For transmit, either set by software or hardware. For receive, always interpreted by software.
14	ACK	Software presets if needed and hardware sets at the appropriate time. Set by hardware once three valid identical CW are received.	Checked by hardware.
15	NP	Software presets if needed and hardware sets at the appropriate time. Set by hardware once three valid identical CW are received.	Checked by hardware.

The next page format is listed in Table 6-11.

Table 6-11 EPL Clause 37 Next Page Format

Bits	Name	Transmitter	Receiver
10:0	Message	Set by software.	Read and interpreted by software once negotiation exchange completes.
11	Toggle	Set by software.	Checked by hardware for toggling
12	ACK2	Set by software.	Read and interpreted by software once negotiation exchange completes.
13	MP	Set by software.	Read and interpreted by software once negotiation exchange completes.
14	ACK	Software presets if needed and hardware follows once three valid identical CW are received.	Checked by hardware.
15	NP	Software presets if needed and hardware follows once three valid identical CW are received.	Checked by hardware.

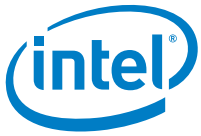
The control word format for SGMII is listed in Table 6-12.

Table 6-12 EPL Clause 37 SGMII Control Word Format

Bits	Name	Transmitter	Receiver
0	1	Set by software to 1b.	Report 1. Validated by software.
9:1	Reserved	Set by software to 0b.	Report 0. Validated by software.
11:10	Speed	Set by software to 0b.	Report speed: 00b = 10 Mb/s 01b = 100 Mb/s 10b = 1000 Mb/s 11b = Reserved Applied by hardware.
12	Duplex	Set by software to 0b.	Duplex mode. Validated by software. Hardware ignores this value and always assumes full duplex.
13	Reserved	Set by software to 0b.	Checked by hardware.
14	ACK	Software presets if needed and hardware sets at the appropriate time. Set by hardware once three valid identical CW are received.	Checked by hardware.
15	LinkUp	Set by software.	Checked by software

The details of the algorithm are as follows:

1. Software sets the MAC in 1000BASE-X mode.
   - Software announces its local abilities by loading the control word into the AN_73_BASE_PAGE_TX register. The NextPage bit in the CW must be set if there are other pages to transmit.
   - Software enables AN37, starts a timer and waits.
2. The switch exchanges the CW with its partner.
   - The switch transmits the AN_37_BASE_PAGE_TX as the control word, repetitively with ACK bit to 0b.



- The switch waits for three identical valid control words and stores them into AN_37_BASE_PAGE_RX.
- When an ACK is received, the switch posts an *An37CompleteAcknowledge* interrupt.

3. The software detects the *An73CompleteAcknowledge* interrupt indicating that a page has been received and can now read the AN_37_BASE_PAGE_RX register (or AN_37_NEXT_PAGE_RX register). If there are next pages, software must load the AN_37_NEXT_PAGE_TX register with the next page to send and flips the ToggleNpLoaded bit. If there are no next pages, go to step 6.
4. The switch exchanges the NP with its partner
   - The switch detects the change into the ToggleNpLoaded bit and starts to transmit the next page. It first transmits with the ACK bit to 0b until hardware receives three identical NextPage messages.
   - The switch detects three identical NextPage messages, sets its ACK bit to 1b and stores the receive page into the AN_37_NEXT_PAGE_RX register.
   - The switch waits for three consecutive ACKs. The message in AN_37_NEXT_PAGE_RX is updated.
   - When an ACK is received, the switch posts an *An73CompleteAcknowledge* interrupt.
5. Steps 3 and 4 are repeated until all pages are exchanged.
6. The switch posts an *An73IdleDetect* interrupt to indicate that the exchange of pages completed.
7. Software detects the interrupt and then applies the highest common denominator.

The exchange can potentially be initiated by a link down event. The algorithm in this case is:

- Software has pre-configured the AN for this event.
- The switch detects a link down, starts to transmit the CW indefinitely with remote fault set.
- The switch detects a link up, clears the remote fault and starts the process previously described.

The exchange could potentially be initiated by the link partner. The algorithm in this case is:

- Software has pre-configured the AN for this event.
- The switch detects arrival of three consecutive AN37 messages.
- The switch starts the process previously described.

6.7.3 SGMII

SGMII is a de-generated mode of the 1000BASE-X clause 37. It enables a 10/1000/1000BASE-T PHY to announce to the switch the negotiated values with its link partner. In this mode, the switch is passive and only replies, it never initiates.

The auto-negotiation process is detailed in the Cisco SGMII specification and summarized here:

1. AN waits for a message from the PHY and operates normally until such message is received.
2. The AN state-machine receives three consecutive identical messages from the PHY.
3. The AN starts a Tx timer and sends an ACK for the length of this timer.
4. The AN restarts the Tx timer if the message receives changes but otherwise continue operation.

5. After the TX timer completes, the AN checks if there is a change in the announcement versus the current mode. If yes, it posts an interrupt to announce the change. If this is a speed change, the AN can be programmed to automatically apply the speed change without software intervention.

The control word format for SGMII is listed in Table 6-13.

Table 6-13 EPL SGMII Control Word Format

Bits	Name	Receiver
0	1	Set by software and the PHY to 1b. Should be validated by software but otherwise ignored by the switch.
9:1	Reserved	Set by software and the PHY to 0b. Should be validated by software but otherwise ignored by the switch.
11:10	Speed	Set by software to 0b for the switch. The PHY reports the speed through this field: 00b = 10 Mb/s 01b = 100 Mb/s 10b = 1000 Mb/s 11b = Reserved The switch can be programmed to apply the speed change automatically. If the speed is not 1000 Mb/s, 100 Mb/s or 10 Mb/s, AN restarts.
12	Duplex	When set by software to 0b, the PHY reports the value negotiated with remote. Should be validated by software but otherwise ignored by the switch. The switch only supports full duplex and does not do anything different regardless of the value of this bit.
13	Reserved	Set by software and the PHY to 0b. Should be validated by software but otherwise ignored by the switch.
14	ACK	Software set to 0b and hardware sets at the appropriate time. Checked by hardware.
15	LinkUp	Set by the PHY. Checked by switch. This bit is included in the overall link up / link down status reported by the MAC to the software.

The algorithm is:

1. Software has pre-configured the AN for this mode.
   - Software sets a transmit timer value (TX repeat time).
   - Software enables AN and waits.
2. The switch operates normally until three identical messages are received.
   - The switch starts a timer and transmits the CW for the duration of that timer. The ACK bit is also set.
   - The switch waits for the ACK bit to be set to 1b while running the Tx timer; the updated word is stored in AN_37_NEXT_PAGE_RX.
   - After the Tx timer terminates and if ACK was received, hardware posts an AN_DONE interrupt and either applies the speed (if programmed to do so) and then posts a link up/down event (if it changes).
3. Software detects the interrupt and applies the values detected if needed.

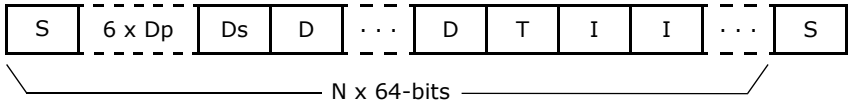
6.8 Physical Coding Sub-layer (PCS)

This section provides details about the FM5000/FM6000 (PCS).

6.8.1 40GBASE-R4

The PCS layer for 40GBASE-R4 (KR4/SR4/CR4) is designed to meet the IEEE 802.3ba, clause 80, 81, 82, 83 and 84.

The frame format in 40 GbE mode follows. The value of Dp (Data preamble) is 0x55 (symbol D21.2), the value of Ds (Data start) is 0xD5 (symbol D21.6). The only mode of operation is to accept frames that start with S, 6xDp and Ds as follows.



These frames are chopped into 64-bit chunks and the number of /I/ bytes added must be such that the S byte always start a new 64-bit chunk.

The 64-bit chunks are then encoded into a 66-bit block that are distributed to four lanes using a round-robin distribution as follows. An alignment marker is inserted after 16383 columns.

LANE 0	66b block n	66b block n+4	Align Marker	66b block n+8
LANE 1	66b block n+1	66b block n+5	Align Marker	66b block n+9
LANE 2	66b block n+2	66b block n+6	Align Marker	66b block n+10
LANE 3	66b block n+3	66b block n+7	Align Marker	66b block n+11

The addition of the alignment marker can cause a decrease of throughput, but is compensated by removing one column of IFG within the next 16383 columns.

The alignment marker includes Block Parity Checking (BIP) that verifies the parity of one lane over every 16383 blocks. The transmitter always generates the correct BIP and the receiver verifies it and reports any error in the BIP error counter. BIP checking does not start until the lane alignment has been achieved.

The lane ordering is automatically discovered using the alignment marker, and the mapping of lanes from an external device to the switch might follow an arbitrary lane mapping.The EPL has the capability to override the automatic lane ordering with a manual method.

The receiver lane deskew logic is designed to handle lane skew up to 1856 bits (180 ns).

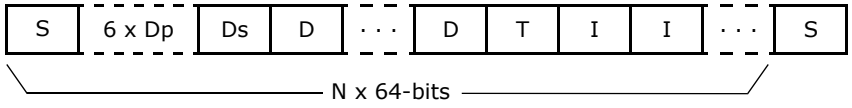
The normal bit rate is 10.3125 Gb/s. An alternative rate of 5.15625 Gb/s is also supported in this mode.

6.8.2 20GBASE-R2

The 40GBASE-R can also be used in a special non-standard two-lane mode (20GBASE-R). This mode follows the same standard except lanes 2 and 3 are not used. All other attributes of 40GBASE-R remains the same. For example, markers for lane 0 and lane 1 are as described in the 40GBASE-R specification.

6.8.3 10GBASE-R

The frame format in 10GBASE-R (KR/SR/CR) serial mode is as follows. The value of Data preamble (Dp) is 0x55 (symbol D21.2); the value of Data start (Ds) is 0xD5 (symbol D21.6). The default mode of operation is to accept only frames that starts with S, 6xDp and Ds.



Configurable options in the MAC enables various preamble size (4, 8, 16, 12) and enables preamble to be treated as data.

The normal rate for this interface is 10.3125 Gb/s. An alternative rate of 5.15625 Gb/s is also supported for this mode.

6.8.4 10GBASE-X

The frame format in 10GBASE-X (XAUI/KX4/CX4) mode is as follows. The value of Dp is 0x55 (symbol D21.2); the value of Ds is 0xD5 (symbol D21.6). The default mode of operation is to accept only frames that starts with S, 6xDp and Ds.

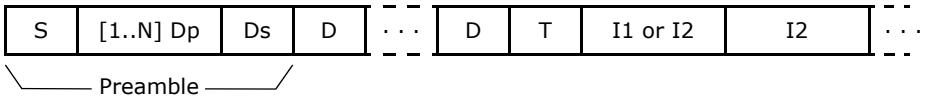
LANE 0	S	Dp	D	D	· · ·	D	A	R	K
LANE 1	Dp	Dp	D	D	· · ·	T	A	R	K
LANE 2	Dp	Dp	D	D	· · ·	K	A	R	K
LANE 3	Dp	Dp	D	D	· · ·	K	A	R	K

Configurable options in the MAC enables various preamble size (4, 8, 12, 16) and enables preamble to be treated as data.

Disparity is applied per lane. Sequences of K/A/R columns follow IEEE specifications.

6.8.5 1000BASE-X Frame Format

The frame format in 1000BASE-X mode (or SGMII) is listed in Table 6-13.



The 1000BASE-X doesn't allow preamble to be use as data. The preamble can be of any size on ingress but must at least have a /S/ symbol, one or more /Dp/ and a /Ds/ symbol. The receiver first checks for /S/, then skips all data bytes until /Ds/ is detected and then starts to capture the frame. For egress, the PCS transmitter terminates the current idle sequence before starting a new frame, so the preamble is reduced by 1 byte (removing one /Dp/) if the MAC had presented the frame to the PCS layer in the middle of an idle sequence. The minimum IFG configurable is four bytes that can get reduced to 3 byes (/S/, /Dp/, /Ds/) due to alignment of idle sequence.

In the 100 Mb/s mode, PCS searches for /S/ and then samples incoming data every 10 cycles. In the 10 Mb/s mode, PCS samples incoming data every 100 cycles.

6.8.6 Link Status

The PCS layer reports if a link is up or down. The link status are fully de-bounced with a different internal timer for link going down and link going up. An interrupt is posted each time the link status changes. A link is declared up when:

- Symbol locks has been achieved
- Lane alignment has been achieved
- Valid idle sequences are received
- No local faults are reported
- No remote faults are reported

Any of these conditions can be enabled or disabled by software. The default is for all conditions to be enabled.

6.8.7 FSIG

The 10 GbE and 40 GbE PCS function is capable of enabling a software configurable FSIG sequence transmission. The same PCS has the ability to receive an FSIG sequence at any time regardless of whether the FSIG transmission sequence is enabled or not. No interrupts are offered for receiving FSIG, the last value received is simply saved in a register.

6.8.8 IFGs

On transmit, it is necessary for the transmitter to modify the length of the Inter-frame Gap (IFG) to align the Start control character to the next boundary of 32 bits for 10 GbE (first octet of the frame must be on lane 0) and 64 bits for 40 GbE.

This is accomplished using one of the following two ways for both 1 GbE and 10 GbE:

- **Guarantee minimum IFG** — The MAC inserts additional idle characters to align the start of preamble on a four byte boundary (0, 1, 2 or 3 bytes added) and then proceeds in adding IFGs in group of 4. If the unit is configured for minimum of x12, the number of IFGs could actually be 12, 13, 14, 15 depending of frame size.
- **Meet average minimum IFG** — Alternatively, the transmitter might maintain the effective data rate by sometimes inserting and sometimes deleting idle characters to align the Start control character. When using this method, the RS must maintain a Deficit Idle Count (DIC) that represents the cumulative count of idle characters deleted or inserted. The DIC is incremented for each idle character deleted, decremented for each idle character inserted, and the decision of whether to insert or delete idle characters is constrained by bounding the DIC to a minimum value of zero and maximum value of three. This might result in inter-frame spacing observed on the transmit XGMII that is up to three octets shorter than the minimum transmitted inter-frame spacing specified in Clause 4; however, the frequency of shortened inter-frame spacing is constrained by the DIC rules. The DIC is only reset at initialization and is applied regardless of the size of the IPG transmitted by the MAC sub-layer. An equivalent technique might be employed to control RS alignment of the Start control character provided that the result is the same as if the RS implemented DIC as previously described.

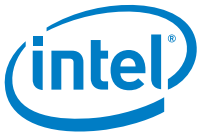
The FM5000/FM6000 devices support both methods.

For the guarantee minimum IFG, the minimum value supported by the transmitter is eight for 10GBASE-R/20GBASE-R/40GBASE-R and four for SGMII/1000BASE-X/10GBASE-X. However, this value might not guarantee reliable operation as there might not be enough idles for the link partner to maintain integrity of the communication depending how the link partner implemented clock compensation. For the meet average minimum IFG, the minimum value supported by the transmitter is 9 for 10GBASE-R/20GBASE-R/40GBASE-R and 5 for SGMII/1000BASE-X/10GBASE-X. The recommended mode of operation is 12 with DIC enabled.

The 20 GbE/40 GbE interface includes the following extra functions:

- The Start control character must be aligned to the next 64-bit boundary, so the IFGs can only be added in increment of eight, the DIC is designed to handle this case.
- An additional alignment marker column is periodically compensated for by deleting columns of /I/ between the markers. This is supported on receive and on transmit. For transmit, the compensation is done by configuring the clock compensation circuit to include idle deletions at a rate corresponding to inserted markers.

An extra circuit for clock compensation is also available that cuts IFGs periodically. The period is programmable between [1..65536].



6.8.9 Changing PCS Mode

PCS can be changed from any mode to any mode on a port-by-port basis and is configured in EPL_CFG_B register. The correct method is to first set to PCS_DISABLE and then change to the desired mode. The correct method to switch to 40 GbE and out of 40 GbE is to first set the PCS mode to DISABLE on all four ports, then update EPL_CFG_B.QplMode to the desired configuration and then update the EPL_CFG_B again with the desired PCS mode.

6.9 MAC

The MAC layer is responsible for the following actions:

In the receive direction:

- Detect start of frame and strip out pre-amble.
- Write the entire frame payload into memory except CRC.
- Compute CRC on frame payload:
  - The scheduler has pushed segment pointers ahead of time.
  - Writes the packet to memory as it is received.
- Validate the CRC and frame length at the end of frame and indicate to the frame control if the frame received is good or bad. A frame is bad if any of the following condition are met:
  - Frame has a bad CRC.
  - Frame is too long. In this case the frame is terminated within a few bytes after the maximum length configured has been reached. The MAC then checks the CRC of oversized packets and records the statistics of how many long packets were received with correct CRC and how many were received with bad CRC.
  - Frame is too short or has a symbol errors. In this case, the frame is terminated immediately and marked as bad, the MAC also enters a mode where it ignores the data received from the PCS until the minimum frame size has been reached. This prevents the MAC from overloading the switch with tiny frames.
  - There was an overrun to the main data memory while the frame was written.
  - The link went down while a frame was received.

In the transmit direction:

- The MAC receives the new frame (modified if needed) from the egress modifier.
- The transmitter first transmits the preamble, unless the preamble is part of packet payload, then transmits the packet payload and computes a new CRC as the frame is being transmitted. If the frame is shorter than the minimum allowed and CRC is to be added to the frame, the frame gets padded with zeros after the last byte of the payload has been transmitted, up to minimum number of bytes minus 4, then the CRC is added to the packet. If the packet is shorter than the minimum by less than 4 bytes, the packets gets truncated to minimum packet size minus 4 such that adding a valid CRC makes the frame exactly match the minimum frame size.

- The computed CRC replaces the original CRC in the frame. The CRC is inverted if the frame is flagged as bad by the egress modifier. This can happen if the frame was originally received bad but was already cut-through or if an irrecoverable ECC error occurred while the frame was retrieved from memory. Padding to minimum frame size is done regardless if the frame is being transmitted with a good or a bad CRC.

The MAC includes the following general transmit/receiver options:

Transmitter

- Force the port to drain all packets regardless of link status. The drain is applied immediately and might cut short an in-flight frame.
- Force the port to hold any packets regardless of link status. The hold is applied cleanly at the beginning of a frame.
- Port drains all packets automatically when link is down.
- Port holds transmission when link is down.

Receiver

- Drain all packets.
- Force all packets to be bad (debug mode).
- Force all packets to be good (debug mode).

6.9.1 Preamble and CRC Optional Processing

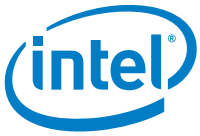
The FM5000/FM6000 MAC has optional features for the processing the preamble and CRC computations enabling the switch to support custom hardware. The options are configurable per MAC and only available for 10 GbE and 40 GbE interfaces. These options are not available for SGMII or 1000BASE-X.

Preamble Options

- Enable the receiver to either strip the preamble (default) or pass it to fabric.
- Enable the transmitter to either generate preamble locally or pass through what comes from the fabric. If the selection is to pass through the preamble, the transmitter forces the first byte of the pre-amble to an SDF symbol.

CRC Options

- The CRC computation has two options: the CRC computation might be started at preamble or right after preamble. The default is after preamble as defined by IEEE. If the CRC is started at preamble, the first byte of the frame is replaced with a configurable fixed value before the CRC is computed. The option of starting CRC at preamble is only allowed if the preamble is captured and passed to the fabric.



6.9.2 Packet Generation

The MAC includes a CJAPT packet generation capabilities for testing XAUI and 1000BASE-X, the CJPAT isn't available for KR and KR4. The CJPAT pattern is:

- START/PREAMBLE/SFD:
  - <FB> 55 55 55 55 55 55 D5
- Payload:
  - 0B for 1 Octet (lane 0);
  - 7E for 3 Octets (lanes 1, 2, 3);
  - 7E for 524 Octets—Low Density Transition Pattern;
  - F4 for 4 Octets—Phase Jump;
  - EB for 4 Octets—Phase Jump;
  - F4 for 4 Octets—Phase Jump;
  - EB for 4 Octets—Phase Jump;
  - F4 for 4 Octets—Phase Jump;
  - EB for 4 Octets—Phase Jump;
  - F4 for 4 Octets—Phase Jump;
  - AB for 4 Octets—Phase Jump;
  - B5 for 160 Octets—High Density Transition Pattern;
  - EB for 4 Octets—Phase Jump;
  - F4 for 4 Octets—Phase Jump;
  - EB for 4 Octets—Phase Jump;
  - F4 for 4 Octets—Phase Jump;
  - EB for 4 Octets—Phase Jump;
  - F4 for 4 Octets—Phase Jump;
  - EB for 4 Octets—Phase Jump;
  - F4 for 4 Octets—Phase Jump;
  - 7E for 528 Octets—Low-Density Transition Pattern;
  - F4 for 4 Octets—Phase Jump;
  - EB for 4 Octets—Phase Jump;
  - F4 for 4 Octets—Phase Jump;
  - EB for 4 Octets—Phase Jump;
  - F4 for 4 Octets—Phase Jump;
  - EB for 4 Octets—Phase Jump;
  - F4 for 4 Octets—Phase Jump;
  - AB for 4 Octets—Phase Jump;

  - B5 for 160 Octets—High-Density Transition Pattern;
  - EB for 4 Octets—Phase Jump;
  - F4 for 4 Octets—Phase Jump;
  - EB for 4 Octets—Phase Jump;
  - F4 for 4 Octets—Phase Jump;
  - EB for 4 Octets—Phase Jump;
  - F4 for 4 Octets—Phase Jump;
  - EB for 4 Octets—Phase Jump;
  - F4 for 4 Octets—Phase Jump.
- RC
  - BD 9F 1E AB
- IFG
  - <FD> followed by 11 idles

6.9.3 Reception Errors

The receiver checks various conditions while it is receiving a frame and posts an end-of-frame status to the switch fabric at the end of the frame. The end-of-frame status could be one of the following:

- The frame was received with a good CRC and no other errors were encountered.
- The frame was received with a bad CRC.
- The frame reception was aborted for one of the following reasons:
  - Frame is smaller than minimum size
  - Oversized
  - Disparity decoding error
  - Invalid symbol
  - Unexpected control symbol
  - Malformed preamble
  - Link fault detected
  - Link went down
  - MAC mode change
  - Overflow

Some of the conditions previously enumerated can be individually enabled or disabled by software and might also be available as interrupts. Table 6-14 lists all conditions and what type of control is available for each one of them and if an interrupt is available to detect this particular condition.

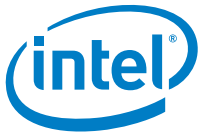


Table 6-14 EPL Conditions (1)

Error Type	Interrupt	Control
Bad CRC	Yes	Software can disable CRC checking.
Undersized	No	Software can configure minimum size.
Oversized	No	Software can configure maximum size.
Disparity error[1]	Yes	Software can disable disparity error checking.
Invalid symbol[1]	Yes	Always enabled. Software can select between two dispositions: 1. Terminate frame immediately. 2. Replace data with 0xFE and continue to receive frame until end of frame detected. ***Note:*** If disposition 2, the CRC is most likely invalid.
Unexpected control symbol	Yes	Always enabled.
Link fault	Yes	Always enabled.
Preamble error	No	Always enabled.
Link fault detected	No	Always enabled.
MAC mode change		Always enabled.
Overflow	No	Always enabled.

1. The disparity errors and symbol errors interrupts can also be raised during idle periods.

The switch includes a centralized statistics module that uses the end-of-frame condition and the frame length to maintain per-port counters such as runt, undersized, frame size binning, etc. However, due to the limited end-of-frame status posted to the fabric, some error conditions are accounted for in the MAC itself.

6.9.4 Counters

The MAC includes the following counters:

- Overflows (8-bit, saturates)
  - — When the data couldn't be pushed to the fabric
- Under-flows (8-bit, saturates)
  - — When the data didn't arrive fast enough from the fabric
- Jabber packets (32-bit, rollover)
  - — Oversized packets that end up having a bad CRC or a framing error at the end.
- Oversized packets (32-bit, rollover)
  - — Oversized packets that end up having a good CRC at the end.
- Runt packets (32-bit, rollover)
  - — Runt packets that end up having a bad CRC or a framing error at the end.
- Undersized packets (32-bit, rollover)
  - — Undersized packets that end up having a good CRC at the end.

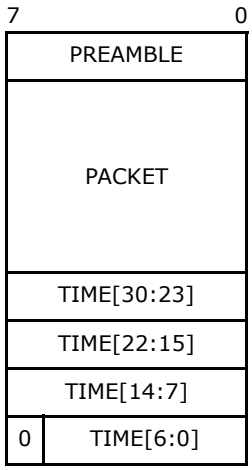
- Number of LOCAL FAULT events: (16-bit, rollover)
  - Incremented every time fault state machine switches to a LOCAL FAULT condition.
- Number of REMOTE FAULT events: (16-bit, rollover)
  - Incremented every time fault state machine switches to a LOCAL FAULT condition.
- Decoding error counter (32-bit rollover)
  - For 20 GbE/40 GbE only, counts the number of times the BIP checking failed
  - For 10GBASE-R, counts the number of BER errors
  - For 10GBASE-X and 1000BASE-X or SGMII, counts the number of disparity/code errors

Note: The error counter does count individual errors as long as they are separated in time by at least 4 valid symbol.

6.9.5 Time Stamping for IEEE1588

Each EPL has a 31- bit clock counter register that is incremented at each EPL clock (coming from Management PLL1, nominal 337.5 MHz). This register is used for time stamping. All EPLs use the same clock source and are all phase locked to each other but might be offset relative to each other by ±3 clocks cycles. The time register is readable by software at any time.

On ingress, each MAC can be configured to replace the CRC with the time stamp for all packets received (bit 7 of the last byte is set to 0b). The exact encoding is:



This time stamp becomes part of payload and is included in the packet transmitted to the local CPU enabling the CPU to recover the exact arrival time of this packet. The time stamp is captured at the time the CRC is replaced. Software has to subtract the time to receive the packet to determine the arrival time at the beginning of the packet. This feature is optionally enabled in the MAC_CFG register.

On egress, if this feature is enabled, the MAC must be configured to replace the last four bytes of the payload with a valid CRC (or a bad CRC if the frame was received with a bad CRC or an uncorrectable memory error was detected for this packet). But before doing so, the MAC examines bit 7 of the last byte and, if it is set to 1b, captures the ingress time from the packet and the current time into two

32-bit registers and raises an interrupt. This interrupt enables the CPU to recover the precise time at which a packet exited the switch and also associates this event with a packet sent by the CPU. There is only one egress capture time register per MAC, so the CPU should either issue only one frame requiring time capture at a time, and read the captured time after an interrupt before emitting a new packet requiring time stamping, or have an associated circuit (FPGA on EBI bus by example) to quickly capture the time event before another one occurs.

Note: The ingress and egress time stamps have to be cleared before a new event is captured (writing any value to the MAC_1588_STATUS clears the register), this ensures that both time stamps are coherent and coming from the same packet.

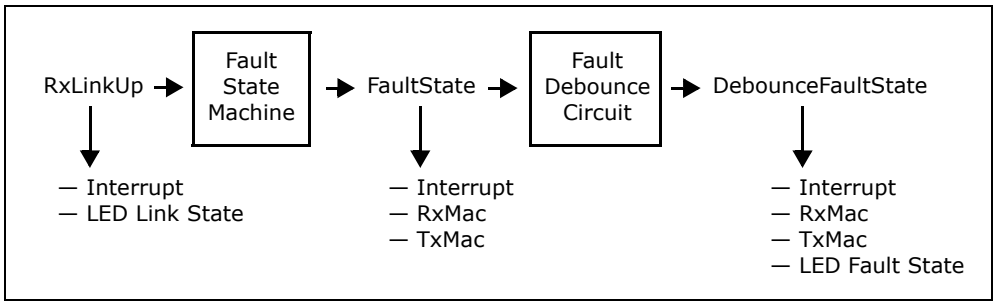
6.10 Status and Interrupts

The MAC might produce the following interrupts/status:

- Signal present/absent on each lane
- Symbol lock gain or lost on each lane
- AN completion on each MAC
- Link up/down on each MAC
- Fault gain/lost on each MAC

6.11 Link State and Fault Conditions

The EPL link state processing is as follows:



RxLinkUp represents the status of the link as perceived by the PMA and PCS layer. The exact conditions included in this state are listed in Table 6-15, where some of these conditions are required while others can optionally be included or excluded by software.

Table 6-15 EPL Conditions (2)

Condition	Mode	Inclusion	Notes
SerDes Ready	All	Optional	
SerDes Signal Detect	All	Optional	Signal detection includes a filter to smooth out possible glitches.
Symbol/Block Size	All	Required	
Lanes Alignment	20G/40GBASE-R 10GBASE-X	Required	
Idle Detection	All	Optional	Detection of at least one idle per time period. This also includes LF and RF for 10 GbE/20 GbE/40GBASE-R.
PHY Reports Link Up	SGMII	Required	
HiBer Detection	10G/20G/40GBASE-R	Optional	

The RxLinkUp state is then forwarded to the fault state machine, which reacts in the following way:

- If RxLinkUp is set to 0b (down), the fault state machine sets the FaultState to LOCAL FAULT
- If the RxLinkUp is set to 1b (up), the fault state machine follows the IEEE defined fault state machine and sets the FaultState to:
  - LOCAL FAULT - if receiving persistent local faults status from remote
  - REMOTE FAULT - if receiving persistent remote faults status from remote
  - NO FAULT - if it didn't received faults from remote

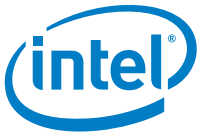
The FaultState condition then goes through a debouncing circuit to clean up any temporary fault conditions and produces a DebouncedFaultState. The debouncing circuit defines two timeouts (UpTimeout and DownTimeout), which are configured individually. The timer used depends on the transitions listed in Table 6-16:

Table 6-16 Debouncing Fault Timer

From	To	Timeout
Local Fault	Remote Fault	UpTimeout
Local Fault	No Fault	UpTimeout
Remote Fault	No Fault	Zero (instantaneous)
Remote Fault	Local Fault	DownTimeout
No Fault	Remote Fault	DownTimeout
No Fault	Local Fault	DownTimeout

Note: The usage of two timers enables the system to react at different times under fault or no-fault conditions.

The RxLinkUp, FaultState and DebouncedFaultState are all available in the port status and can each be used as an interrupt source, though only the DebouncedFaultState should really be required from a software perspective.



The FaultState and DebouncedFaultState are also sent to the TxMAC and RxMAC that can be programmed to react automatically to the different fault conditions listed in Table 6-17. For TxMAC in drain mode, packets from the fabric are drained and not transmitted to the line, for TxMAC hold mode, packets from fabric are on hold while the fault condition persist. The drain/hold mode can also be forced manually overriding the state conditions listed in Table 6-17.

Table 6-17 TxMAC Fault

FaultState	DebouncedFaultState	TxMAC	RxMAC
Local Fault	No	Send Remote Fault, hold/drain packets	Drain any incoming frames.
No	Local Fault	Send Remote Fault, hold/drain packets	Drain any incoming frames.
Remote Fault	Remote Fault	Send idles, hold/drain packets.	Drain any incoming frames.
Remote Fault	No Fault	Send idles, hold/drain packets.	Drain any incoming frames.
No Fault	Remote Fault	Not possible.	Not possible.
No Fault	No Fault	Send frames normally.	Accept incoming frames.

NOTE: ***This page intentionally left blank.***

7.0 PCIe Interface

The FM5000/FM6000 supports 4-lane PCIe Gen 1 v2.0(2.5GT/s) / Gen 2 v2.0(5.0GT/s) interface that connects to the core switch fabric through an internal port.

7.1 Overview

The PCIe interface is a PCIe Gen 1 v2.0(2.5GT/s) / Gen 2 v2.0(5.0GT/s) compatible end-point that supports the following features:

- Only one function (Function 0) is supported.
- Only one virtual channel (VC0) is supported.
  - PCIe only supports one packet queue.
- Only one traffic class (TC0, best effort).
- One, two or four lanes each operating at 2.5 Gb/s or 5.0 Gb/s mode.
  - Speed and number of lanes must be auto-negotiated.
  - Lane reversal and lane polarity are statically configured.
  - Lane bifurcation not supported.
- Support the following transactions:
  - TARGET
    - Configuration read/write.
    - Non-posted reads and non-posted read-lock.
    - The device makes no distinction between a read and a read-lock, both are executed exactly the same way.
    - Non-posted writes and posted writes.
    - The device supports up to 256 bytes of posted writes.
    - Non-posted writes are queued the same way as posted writes.
    - The device always preserves ordering for all read/write transactions.
  - INITIATOR
    - Interrupt messages.
    - Non-posted reads (DMA packet transfer).
    - Posted writes (DMA packet transfer).

- Scattered-gathered DMA packet transfer.
  - — Packets can be fragmented over multiple buffers.
- In-band reset and out-of-band reset.
  - — In-band reset is commanded via local registers in the PCIe module.
  - — Out-of-band reset is commanded via the HSM module.
- No advanced sleep mode.
- Byte swaps for big endian processors.
- 32-bit/64-bit address decoding supported.
- Response to switch PAUSE frame.

The block diagram of the module is shown in Figure 7-1.

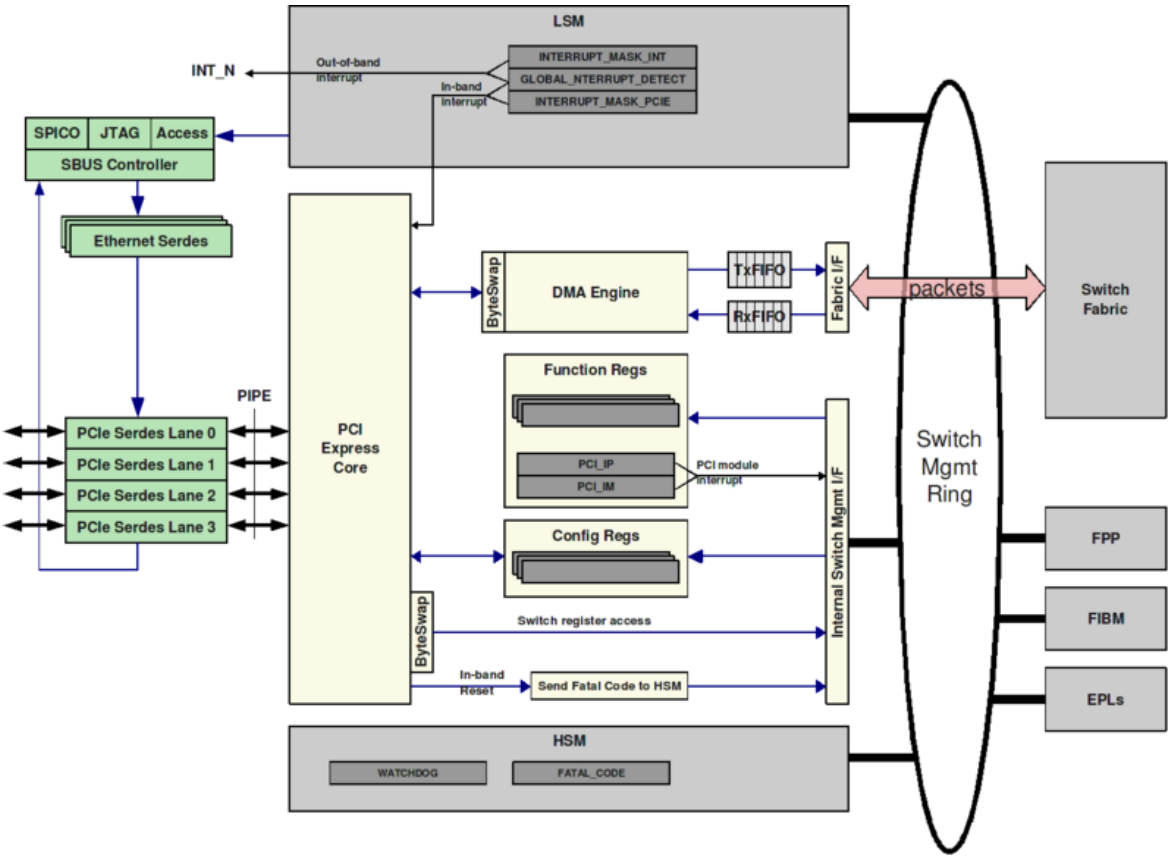


Figure 7-1 PCIe Block Diagram

The block contains two register files: one for the PCIe configuration space and one for the control of the internal functions of the module itself. The configuration register file follows the standard PCIe configuration space and is accessible via configuration read/write cycles from a PCIe host or via any master agent on the internal bus (such as EBI or I^2C). The block register file contains registers to control the extended functions of this block such as the DMA packet agent. This register file is accessible via normal read/write cycles from a PCIe host or via any master agent on the internal bus.

The packet DMA engine is responsible for transferring incoming packets or outgoing packets from the internal port to the host memory. The packet DMA engine initiates read/write accesses to host memory on its own to either retrieve/update buffer descriptors or to transfer packets. The packet DMA engine contains small FIFOs that are used to buffer packet payloads between internal port and PCIe to ensure pipelines are kept busy.

The generic switch interrupt pin is also made available to this block to command emission of interrupt messages when an interrupt condition exists in the switch. This block is also a source of interrupts on its own, which are routed internally.

7.2 Power Up

At power up, the PCIe differential pairs are in a reset state and the PCIe interface is inactive until the PCIe differential pairs are initialized. This requires the PCIe block of the switch to be initialized via an external ROM, both SPI Flash and I^2C EEPROM are supported and can be selected by boot mode set via the FM5000/FM6000 GPIO[9..7] settings.

7.3 Access to SerDes

SerDes is accessed through the LSM module.

7.4 Reference Clock

The PCIe interface requires one 125 MHz reference clock. Figure 7-2 shows the reference clock internal termination scheme. These resistors can be disabled by strapping the LED_DATA[1]. The LED_DATA[1] pin is latched when CHIP_RESET_N is de-asserted, and has an internal pull-down that defaults to internal termination. If this pin is externally pulled up (such as with a 5 KΩ resistor to VDD25), the REFCLK input has its internal termination disabled.

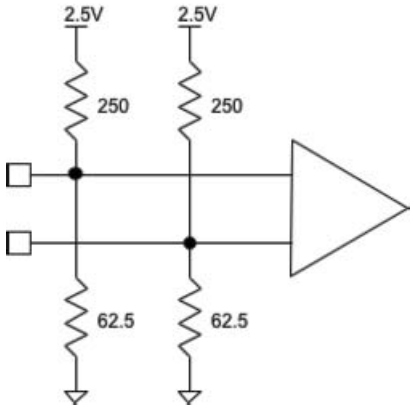


Figure 7-2 PCIe Reference Clock Internal Termination

7.5 In-Band Reset and Link Down Events

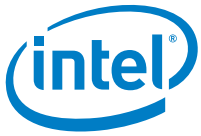
The PCIe module supports in-band reset. The in-band reset is asserted when a 2 ms continuous TS1 reset ordered-set is detected. The module can be configured to support one of the following three options:

- Reset the PCIe core only (warm reset)
  - The core fabric continues to operate normally while the PCIe core is reset.
- Reset the PCIe core and disable DMA (default value)
  - Disabling DMA caused any in-flight packets to the fabric to be terminated with an immediate tail error and causes any packets from fabric to be drained. The core fabric continues to operate normally while the PCIe core is reset.
- Send a fatal code to the watchdog circuit.
  - The HSM watchdog stores this code in the FATAL_CODE register and commands an internal general reset of the chip, which causes the entire chip to reboot as if the RESET_N pin was asserted (except the watchdog circuit itself). Resetting the switch also resets the PCIe SerDes and PCIe core and those need to be reprogrammed via serial boot or via EBI as detailed in previous section. This causes all traffic to stop while the switch is reset.

Also, the PCIe link can be restarted (link up/down) without affecting the switch fabric. The module can be configured to support one of the two operating modes when a PCIe link down event is detected:

- Reset the PCIe core only (warm reset)
- Reset the PCIe core and disable DMA (default value)

The link down operating mode and in-band reset operating mode are configurable independently.



7.6 Interrupts

Both out-of-band and in-band interrupts are supported. The out-of-band uses the INT_N pin while the in-band interrupt uses MSI capability.

The interrupt requests from different sources within each module of the switch are stored into local registers inside each module and can be masked. If a particular interrupt is not masked, a single interrupt pending bit is sent to the LSM and stored in the GLOBAL_INTERRUPT_DETECT register.

The LSM provides two INTERRUPT_MASK registers that enable designers to select which interrupt sources are posted on the out of band interrupt and which are posted on the in-band interrupts. For in-band interrupts, the MSI must have been enabled for the interrupt to be posted.

The PCIe module can be a source of interrupts as any other module in the switch. The interrupt sources are detailed in PCI_IP and can be masked using PCI_IM. If a PCIe internal interrupt is generated and not masked, a PCI interrupt pending is sent to LSM as for any other module.

The different internal interrupt sources are:

- Buffers sent or received
- Link up/down
- Pause on

7.7 Power Management

The device only supports the two mandatory power states:

- D0: ON
- D3: OFF

The device does not support low power modes. In the D3 state, the PCIe block enters a quiet period but the rest of the switch isn't affected. Software must shut down circuits manually before entering into D3 if power reduction is desired.

7.8 Byte Swapping

Byte swapping is available to speed up PCIe accesses for big endian processors. The PCIe is naturally little endian and designed for little endian processors such as Intel® x86 family. When a big-endian processor is used, there are two choices, preserve byte addresses or preserve byte ordering in a scalar. If preservation of byte addresses is the only option offered on the processor, all 32-bit accesses are byte swapped and software must swap back the bytes to restore the integrity of the scalar before using these values.

To address this problem, the PCIe interface includes two byte swapping mechanisms at separate locations to reduce software load on the processor. There is a byte swap mechanism for target transactions and a byte swap mechanism for the locally initiated transactions issued by the DMA controller. Both are enabled by software by writing to the PCI_ENDIANISM register. Any non-zero value in this register enables the byte swap mechanisms.

The byte swap mechanisms for target transactions simply swap the bytes (byte 3 swapped with byte 0 and byte 2 swapped with byte 1) for all transactions. For this to work properly, all transactions initiated by the host must be 32-bits accesses. The byte swap mechanisms for the DMA engine are more complicated. The byte swap mechanisms only swap bytes when accessing 32-bit words and are disabled when transferring packet payloads.

Note: There is no byte swap mechanism for the configuration registers themselves. These registers are typically accessed once and there is no significant performance degradation if the byte swapping has to be done by software.

7.9 32-bit/64-bit Addressing

The PCIe module supports 32-bit and 64-bit addressing. The device reports 64-bit mode in PCI_BAR0[2:1] and requires a 64-bit address to be loaded in PCI_BAR0/PCI_BAR1. The device accepts 32-bit addressing and 64-bit addressing for target transactions (3 Dword/4 Dword transaction format) if the upper 32-bit address is zero; otherwise, only 64-bit addressing (4 Dword format) is accepted.

Similarly, when the PCIe module initiates a transaction (DMA controller), the address size is 32-bit if the target address is in the first 4 Gb of address space and is 64-bit otherwise.

7.10 Registers

7.10.1 PCIe Configuration Space

The PCIe configuration space is listed in Table 7-1, which conforms to the normal configuration space in defined in PCIe specification.

Table 7-1 PCI Configuration Space

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0
0x0	Device ID		Vendor ID	
0x4	Status		Command	
0x8	Class Code			Revision ID
0xC	BIST	Header Type	Latency Timer	Cache Line Size
0x10	BAR 0 (address_low)			
0x14	BAR 1 (address_high)			
0x18	BAR2 (unused)			
0x1C	BAR2 (unused)			
0x20	BAR2 (unused)			
0x24	BAR2 (unused)			
0x28	CardBus CIS pointer			
0x2C	Subsystem Device ID		Subsystem Vendor ID	
0x30	Expansion ROM Base Address (unused)			

Table 7-1 PCI Configuration Space (Continued)

Byte Offset	Byte 3	Byte 2	Byte 1	Byte 0
0x34	Reserved			Cap Ptr (0x40)
0x38	Reserved			
0x3C	Max Latency	Min Grant	Interrupt Pin	Interrupt Line
0x40	Power Management Capabilities		Next_Cap_ptr (0x50)	PM Cap ID (0x01)
0x44	PM Data			
0x50	MSI Capabilities		Next_Cap_ptr (0x60)	MSI Cap ID (0x05)
0x54	MSI Address High			
0x58	MSI Address Low			
0x5C	PCIe Capabilities		MSI Data	
0x60			Next_Cap_ptr (0x00)	PCIe Cap ID (0x10
0x64	PCIe Link Capabilities			
0x68	PCIe Device Status		PCIe Device Control	
0x6C	PCIe Link Capabilities			
0x70	PCIe Link Status		PCIe Link Control	
0x74:0xFF	PCIe Reserved (all 0x0)			
0x100:0xFF	PCIe Reserved (all 0x0)			

The Vendor ID field is 0x8086. The device ID is 0x155B.

Only Base Address 0 (BAR0) is used. The window size is 24 MB. The address offset relative to base is equal to the register multiple by four.

```
BAR0[0x1100*4] => PCI_CTRL
BAR0[0x380000*4] => L2L_MAC_TABLE_LOOKUP
```

7.10.2 PCIe Control Registers

The PCIe module control registers are listed here:

- **PCI_ENDIANISM** — Controls little/big endian for packet payload.
- **PCI_COMMAND** — Controls DMA engine state for transmit and receive.
- **PCI_STATUS** — Reports DMA engine status.
- **PCI_COASLESCING** — Controls interrupt rate.
- **PCI_RX_BD_BASE**, **PCI_RX_BD_END**, **PCI_TX_BD_BASE** and **PCI_TX_BD_END** — Define descriptor list locations.
- **PCI_IP**, **PCI_IM** and **PCI_COALESCING** — Interrupt mask and coalescing configuration.
- **PCI_CURRENT_TX_DATA_PTR**, **PCI_CURRENT_RX_DATA_PTR**, **PCI_CURRENT_TX_BD_PTR** and **PCI_CURRENT_RX_BD_PTR** — Monitoring information.
- **PCI_IP** and **PCI_IM** — Interrupt status and interrupt mask.

- **PCI_TX_FRAME_LEN** — Defines minimum/maximum frame length.
- **PCI_SIZE** — Defines DMA block transfer size.
- **PCI_DMA_CFG** — Defines DMA configuration; enable/disable fabric tag location and size.
- **PCI_FRAME_TIMEOUT** — Defines the maximum time for a frame to be waiting for resources.
- **PCI_STAT_COUNTER** — Counts PCIe error events. General frame counters for PCIe port available from STATS unit.
- **PCI_CORE_CTRL** — Core control options.
- **PCI_CORE_DEBUG** — Various debug registers.

7.10.2.1 Command Register

The Command register is used to control the DMA engine packet processor operation.

Table 7-2 PCIe DMA Command Definitions

Name	Value	Description
PCI_TX_START	0x0001	Instructs the FMPCI Tx packet processor to begin processing descriptors from the Transmit Descriptor Table.
PCI_RX_START	0x0002	Instructs the FMPCI Rx packet processor to begin processing descriptors from the Receive Descriptor Ring.
PCI_TX_STOP	0x0003	Stops the Tx packet processor and resets the descriptor index to zero.
PCI_RX_STOP	0x0004	Stops the Rx packet processor and resets the descriptor index to zero.
PCI_TX_POST	0x0005	Informs the Tx processor that new descriptors have been written to the Tx descriptor ring. The number of packets processed is included in argument and used to determine the received packet counter, used for coalescing only.
PCI_RX_POST	0x0006	Informs the Rx packet processor that new descriptors have been written to the Rx descriptor ring. The number of packets processed is included in argument and used to determine the received packet counter, used for coalescing only.
PCI_TX_SUSPEND	0x0007	Ask the Tx to go idle at the end of the current frame regardless if there are more BDs ready to transmit. This command has no effect if the transmitter has not been started. The transmitter can be restarted using a TX_PORT command.
PCI_RX_SUSPEND	0x0008	Ask the Rx to go idle at the end of the current frame regardless if there are more BDs ready to receive data and there is data available. This command has no effect if the receiver has not been started. The RX can be restarted using a PCI_RX_POST command.
PCI_TX_DRAIN	0x0009	Signal Tx to go into a drain mode after the current in-flight frame. All data from the CPU is drained for now on until a TX_STOP/TX_START command sequence executes. While in drain mode, the BDs and frame data is still read from the CPU but not transmitted to the fabric. A TX_POST command or TX_SUSPEND command cancels the drain mode. The command is intended to enable a software device driver to be shut down cleanly as soon as possible.
PCI_RX_DRAIN	0x000A	Signal Rx to go into a drain mode immediately. The current frame in transit is terminated with an error code and all data from the fabric is drained for now on until any Rx command executes.

7.10.2.2 Status Register

A read from the PCI_STATUS register returns the following information listed in Table 7-3.

Table 7-3 PCIe DMA Status Register Read Format

Name	Bits	Description
TxState	2:0	Indicates the current Tx processor status: 000b = Stopped 001b = Running 010b = Idle 011b = Draining 100b = Pause All other values are reserved.
RxState	3:2	Indicates the current Rx processor status: 00b = Stopped 01b = Running 10b = Idle 11b = Draining

7.10.2.3 Descriptor List Boundaries

7.10.2.3.1 Receive Descriptor Table Base Address

This 64-bit register specifies the physical address in host memory of the base of the Receive Descriptor Table.

This register is read each time the Rx processor starts after a reset (either by PCI reset or by writing the FMPCI_RESET_RX command).

Note: The address must be aligned to a 32-byte boundary.

7.10.2.3.2 Receive Descriptor Table End Address

This register defines the end address of the Receive Descriptor Table. The Rx descriptor logic compares the current pointer after it was incremented to that value and then resets the pointer to the base address if found greater or equal.

7.10.2.3.3 Transmit Descriptor Table Base Address

This 32-bit register specifies the physical address in host memory of the base of the Transmit Descriptor Table.

This register is read each time the Tx processor starts after a reset (either by PCI reset or by writing the FMPCI_RESET_TX command).

Note: The address must be aligned to a 32-byte boundary.

7.10.2.3.4 Transmit Descriptor Table End Address

This register defines the end address of the Transmitter Descriptor Table. The Tx descriptor logic compares the current pointer after it was incremented to that value and then resets the pointer to the base address if found greater or equal.

7.10.2.4 Interrupt Status Register

The PCIe block posts an interrupt when the DMA engine has transferred a packet (Rx or Tx) or, in the case of a coalescing interrupt, when there were a certain number of packets transferred or a waiting time was exceeded.

The DMA Engine Interrupt Status register is listed in Table 7-4.

Table 7-4 PCIe DMA Interrupt Status Register

Name	Bits	Type	Description
Tx	0	RW1C	The TX packet processor completed processing on a Tx descriptor.
Rx	1	RW1C	The Rx packet processor completed processing an Rx descriptor.
Coalescing	2	RW1C	The set of Tx or Rx descriptors were processed or the coalescing waiting timer expired.

7.10.2.4.1 Interrupt Mask Register

The DMA Engine Interrupt Mask register enables interrupt sources to be masked.

7.10.2.4.2 Coalescing

The control registers define the parameters for interrupt coalescing. It defines the latency to post an interrupt message for Tx and Rx interrupts and the maximum number of packets transferred before posting an interrupt.

7.11 Packet DMA Engine

The packet DMA engine transfers packets between the core switch and the host memory following a list of buffer descriptors. There is a list of buffer descriptors for transmit and a list for receives. The number of buffer descriptors per list must be a power of two and the base must be aligned to a 32-byte boundary (the size of a buffer descriptor) as shown in Figure 7-3.

The packet DMA engine supports fragmentation packets into multiple buffers in transmit and receive.

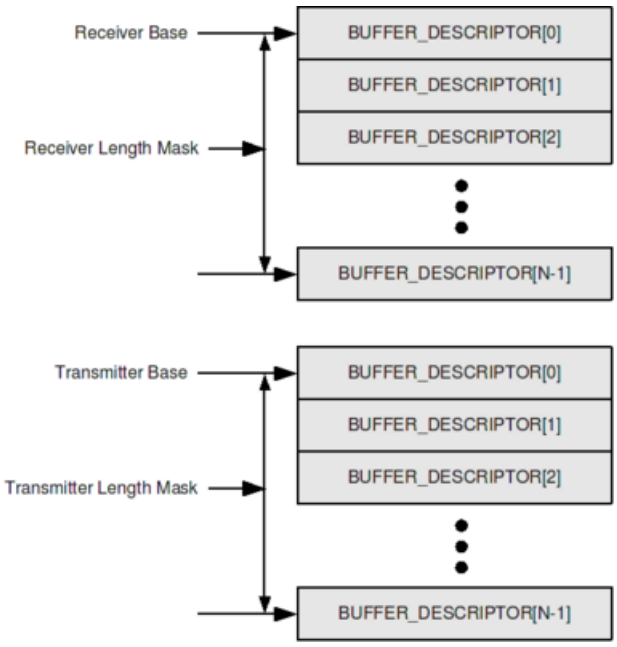


Figure 7-3 PCIe DMA Buffer

7.11.1 Buffer Descriptors

The DMA transfer for transmit and receive uses a 16-byte buffer descriptor structure as listed in Table 7-5.

Table 7-5 PCIe DMA Buffer Descriptor Structure

Offset	Byte 3	Byte 2	Byte 1	Byte 0
0x00	Buffer/Data Length		Unused	Status
0x04	Buffer Address Low			
0x08	Buffer Address High			
0x0C	Reserved			
0x10	Reserved			
0x14	F64 Tag			
0x18	F96 Tag (continued)			
0x1C	Reserved			

7.11.1.1 Status

The status field is listed in Table 7-6and is used to report the state of this buffer descriptor.

Table 7-6 PCIe DMA Buffer Descriptor Status Field

Name	Bits	Description
READY	0	Written by software, read by hardware. The descriptor was prepared by software and is ready for processing by the Rx/Tx packet processors.
STOP	1	Written by software, read by hardware. The Tx/Rx packet processor transitions to the STOPPED state after processing this descriptor.
DONE	2	Written by hardware, read by software. The descriptor was used by hardware and ready to be read by software.
EOP	3	Written by software on transmit to indicate that this buffer is the last buffer of this packet. Written by hardware on receive to indicate that this buffer is the last buffer of the packet.
ERR	5:4	Written by hardware on receive to indicate if an error was detected. The possible values are: 0x0 = This frame had no error. 0x1 = An unrecoverable ECC error occurred while the packet was retrieved from switch fabric main memory. 0x2 = Packet was truncated due to a timeout. Written by software on transmit to flag an error code to the fabric. The possible values are: 0x0 = Frame is good. 0x1 = Frame has bad CRC. 0x2 = Frame has framing error. The value 0x1 and 0x2 are for testing purpose. The value 0x2is also automatically posted if the DMA is stopped in a middle of a frame and the frame had to be cut short to the fabric (incomplete frame).

7.11.1.2 Buffer Length

For both Rx and Tx descriptors, this is the size of the buffer pointed to by the Buffer Address field in bytes.

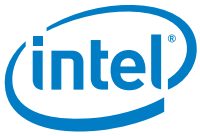
The Rx packet processor reads this field to determine the size of the buffer prior to saving data into the buffer, the buffer length is updated when the buffer was consumed. The Tx processor reads this field to determine how many bytes to transmit.

7.11.1.3 Buffer Address

This is the address of the buffer in which the data is to be placed by the Rx processor or from which data is to be transmitted by the Tx processor. This buffer is contained in host memory and the address is that returned by passing a Linux* kernel virtual address to the dma_map_single function call.

The buffer address is always 64 bits. If the upper 32 bits are set to 0x0, PCIe initiates a 3 Dword transaction (32-bit addresses); otherwise, it generates a 4 Dword transaction (64-bit address).

There is no alignment on the address except that it must not cross a 4 Gb boundary.



7.11.1.4 F64 Tag

This field is the F64 tag that was received with the packet or has to be transmitted along with the packet. Two sizes are supported; 64-bit or 96-bit and two locations are supported; at offset 0 and at offset 12. The actual F64 tag format is microcode dependent and the actual format is ignored by the DMA controller except for Pause packets.

The exact format for the FM5000/FM6000 microcode is defined as part of this specification and varies slightly depending of the location of the tag.

The F-tag format at offset 0 is listed in Table 7-7.

Table 7-7 F-tag Format for Offset 0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
USER								FTYPE		VTYPE		SWPRI			
VLAN PRI			CFI	VLAN											
Source GloRT															
Destination GloRT															

The F-tag format at offset 12 is listed in Table 7-8.

Table 7-8 PCIe F-tag Format at Offset 12

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FTYPE		VTYPE		SWPRI				USER							
VLAN PRI			CFI	VLAN											
Source GloRT															
Destination GloRT															
EXTRA (if F-tag is 96 bits)															
EXTRA (if F-tag is 96 bits)															

Packets that are transmitted and received to the switch fabric must contain a proprietary F64 F-tag. Packets read from memory or written to memory do not have the F-tag present in the payload. On switch egress (going to CPU) the DMA engine removes the F-tag from the packet and saves it in all BDs used for this packet. On switch ingress (coming from CPU), the DMA engine reads the F-tag defined in the first BD of the packet and inserts it into the packet before sending it to the fabric. The F-tag includes the VID and VPRI of the packet.

There are two options for the location of the F-tag within the packet. If the F-tag location is configured for offset 12, the normal VLAN tag is absent in the packet memory, and the software device driver must pass this information to the kernel out-of-band using the content of the F-tag. If the F-tag is configured for offset 0, the normal VLAN tag could be made available in the packets to the CPU using switch tagging options.

Using an F-tag at offset 12 is not recommended for new designs and will be phased out over time.

7.11.2 Packet Processing Overview

The Tx and Rx packet processors read descriptors from a ring in host memory and process these descriptors in order until they exhaust their respective rings. The processors have identical state machines. Software can start and stop the processors by writing commands to the Command register. The states are STOPPED, RUNNING, IDLE, DRAIN (and PAUSE for TX).

Note: DMA terminates incomplete frames to the fabric with a framing error trailing code if the DMA cannot complete the packet (no end).

In the STOPPED state, the packet processor is not processing descriptors and the descriptor index (array index into the descriptor table) is zero. Software causes the packet processor to start processing descriptors by writing the START command to the Command register. At this transition, the packet processor reads the Descriptor Ring Base Address and Descriptor Ring Length Mask registers and resets the Descriptor Index Register.

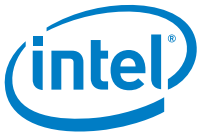
The packet processor reads descriptors and processes them until it finds a descriptor with either a DONE bit set or a RDY bit reset in the descriptor status field. As the processor fetches each descriptor it increments the Descriptor Index by 16, masks the result with the Descriptor Index Mask and reads the next descriptor.

When the packet processor finds a descriptor with the DONE bit set or the RDY bit reset, it stops reading packets, and transitions to the IDLE state. The packet processor enters the RUNNING state again when software writes a POST command to the command status register. At that point the hardware returns to the RUNNING state and starts by re-reading the last packet descriptor that was read.

The DMA engine raises interrupts only at frame boundaries. It does not generate an interrupt if a buffer was consumed, in receive or transmit, and this buffer doesn't terminate a packet.

Furthermore, the DMA engine has a coalescing feature to reduce the number of interrupts to the host processor. This feature delays interrupt posting for transmit or receive when either sufficient number of packets are transmitted or received or if the waiting time was exceeded. The coalescing can be independently set for both Tx and Rx.

The exact process is as follows:



```
uint16 packetCountRx;
uint16 packetCountTx;

if ( startTX or stopTX )
packetCounterTx = 0

if ( startRX or stopRX )
packetCounterRx = 0

if ( packetTransmitted )
setTxInterrupt()
packetCounterTx++
if ( packetCounterTx > coalescingPacketCountLimit )
   setCoalescingInterrupt()

if ( packetReceived )
setRxInterrupt()
packetCounterRx++
if ( packetCounterRx > coalescingPacketCountLimit )
   setCoalescingInterrupt()

if ( commandTxPost )
if ( packetCounterTx > value )
packetCounterTx -= value
else
    packetCounterTx = 0

if ( commandRxPost )
if ( packetCounterRx > value )
packetCounterRx -= value
else
    packetCounterRx = 0

if ( hostRequestClearingCoalescingInterrupt )
  timer = 0
  clearCoalescingInterrupt()

at each CoreClock/128:
if ( packetCounterTx != 0 or packetCounterRx != 0 )
   if ( timer > coalescingTimeout )
       setCoalescingInterrupt()
   else
       timer++
else
    timer = 0
```

7.11.3 Fabric Congestion Management

The PCIe block is a fabric port like any other external Ethernet port and supports the same congestion management features and also supports reacting to receiving PAUSE frames from the fabric. For example, it stops sending traffic to fabric after receiving a PAUSE-ON packet and restarts on a PAUSE-OFF frame. However, there are a few differences compared with a normal external Ethernet port.

7.11.3.1 PAUSE Detection

The PCIe uses the content of the F-Tag to determine if the frame is a PAUSE-ON or a PAUSE-OFF frame. The fabric egress microcode sets the first word of the F-Tag to 0xFFFFxxxx for PAUSE-ON and 0x00FFxxxx for PAUSE-OFF. The PCIe port uses this information to detect receiving PAUSE-ON or PAUSE-OFF packets and no other criteria (packet length, MAC address, etc...) is used. If the first word of the F-Tag doesn't match any of these two words, this is assumed to be a normal packet.

Note: The PAUSE frames are passed to the CPU, which would most likely drop them.

7.11.3.2 PAUSE Reaction

PCIe stops transmitting packets to the fabric after receiving a PAUSE-ON (going to PAUSE state) and resumes transmission after receiving a PAUSE-OFF (going out of PAUSE state). The PAUSE state clears itself after 500 ms from when the last PAUSE-ON was received, or if the transmitter is stopped and then restarted. The pause state is not cleared if the transmitter is placed in and out of suspend mode. If a current frame was in flight, its transmission is terminated before the PAUSE-ON is applied. The pause state is reported in the PCI_IP register and can be manually cleared.

8.0 External Bus Interface (EBI)

The EBI controller is designed to interface with simple parallel local buses. This interface is backward compatible with the FM4000 series EBI controllers.

8.1 Overview

The EBI supports the following signals:

- Bus interface:
  - ADDR
  - DATA
  - AS_N
  - CS_N
  - RW_N
  - DTACK_N
  - DERR_N
  - PAR
  - EBI_CLK
- DMA control:
  - RXRDY_N
  - RXEOT_N
  - TXRDY_N
- Other pins:
  - INTR_N

The maximum EBI_CLK rate is 66 MHz.

The EBI is configured through configuration pins available in the LSM. The strapping options for CPU are:

- **CS_N** — Defines if EBI is enabled or not. If this pin is held low when CHIP_RESET_N is de-asserted, the switch assumes that EBI is not used and all EBI pins (including EBI_CLK) are then ignored.
- **DTACK_INV** — Defines the polarity of DTACK.
- **RW_INV** (configuration pin for polarity of read/write) — Defines the polarity of RW_INV.

- **IGNORE_PARITY** — Defines if parity is used or not.
- **PARITY_EVEN** — Defines the parity type.

8.2 Bus Timing

The CPU bus interface timing diagram is shown in Figure 8-1 (assumes DTACK_INV and RW_INV are both set to 0b).

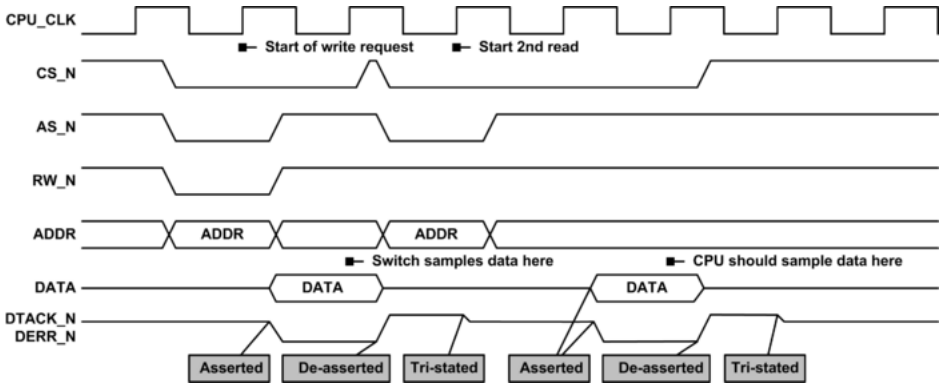


Figure 8-1 CPU Bus Timing Diagram

The CPU bus interface is synchronous and the switch samples/drives at rising edge of the CPU clock and follows a simple protocol:

- A cycle starts when CS_N and AS_N are both asserted (low). The ADDR and RW_N signals are also sampled on the same cycle to determine the register addressed and if the cycle is a read or a write.
  - The interpretation of RW_N depends on the strapping of the pin RW_INV. If the RW_INV is strapped to GND, then RW_N = 1b is READ and RW_N = 0b is WRITE. If the RW_INV is strapped to VDD25, then RW_N = 1b is WRITE and RW_N = 0b is READ.
- For a write cycle, the data is always sampled on the next cycle. Furthermore, the switch has a small write FIFO and asserts a DTACK_N signal on the next cycle as well if this write FIFO is not full; otherwise, the DTACK_N is delayed until this FIFO has room to store the data. The minimum cycle time is two clock cycles.
  - DERR_N is actively driven at the same time as DTACK_N is asserted. It is asserted (logic low) if the data parity is incorrect and de-asserted (logic high) if the data parity is correct.

    Note: The switch internally cancels the write operation if the parity is incorrect.
- For a read cycle, the switch delays asserting DATA and DTACK_N until data is available and drives both signals on the same clock cycle. The minimum cycle time is three clock cycles.
  - The parity on the data bus is presented at the same time as the data.
- After the read or the write cycle completes, the switch actively de-asserts DTACK_N for one cycle and then tri-sates the DTACK_N signal.

  Note: The switch can detect a new read or write cycle on the same cycle on which DTACK_N is actively de-asserted or on any follow-on cycles.

- The bus cycle latency from cycle start to DTACK_N being asserted is varied depending on the register accessed and the traffic load in the switch. The worst case is estimated to be 5 µs.

8.2.1 Using DATA_HOLD

The CPU bus interface also supports a data holding mode (asserted with DATA_HOLD strapping option) where the DTACK_N (for read and write cycles) and DATA (for read cycles) are maintained asserted until a chip select is de-asserted. The DTACK_N and DATA are then immediately tri-stated coincidentally with the de-assertion of chip select. This mode is intended for situations where a fixed length cycle is needed, the length of the cycle must be greater than the worst delay expected on DTACK_N.

Figure 8-2 shows the effect of the DATA_HOLD option.

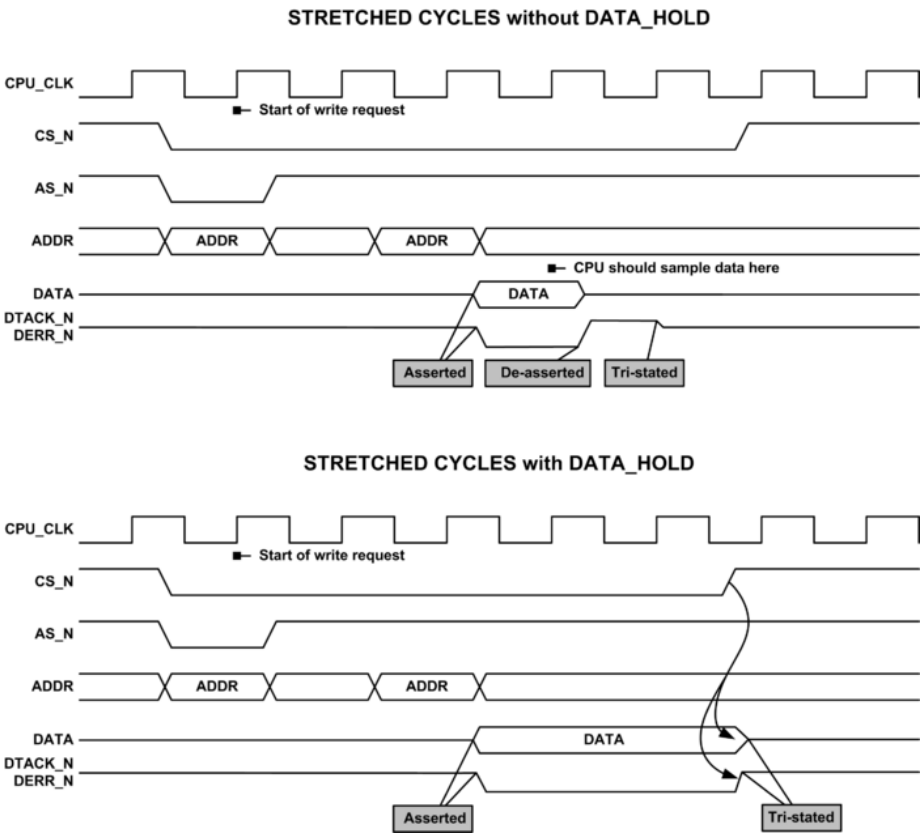


Figure 8-2 Effect of EBI DATA_HOLD on CPU Cycles

8.3 Atomic Accesses

The frame handler includes tables that are wider than 32 bits, which need to be accessed atomically. For example, the data must be written as one single large word and not as multiple smaller 32-bit words. This ensures that the table doesn't contain an intermediate incorrect value at any point in time or that software doesn't read a false value. The switch includes extra circuitry to hold the data words into temporary registers before performing the actual read or write. The exact process is the following.

For write:

- Software should write the entry starting with the least significant word and terminate with the most significant word.
- Hardware stores the least significant words into a temporary cache and then issues a write into the table when the most significant word is written using the content of the temporary registers to complete the entry.
- Software can accelerate loading an entire table with the same value (0 by example) by writing the least significant words only once and load successive entries by writing only the most significant word.

For read:

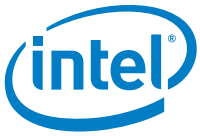
- Software might read an entry in any order.
- Hardware reads a table entry each time the index or the table is different from the last index or table read or written, and saves all words read into a temporary cache and then returns the particular word addressed by the software. If the index and table are the same as the last index and table, the content of the cache is used to return the word addressed.
- Software does not have to read all words if they are not needed.

There is only one set of temporary registers per bus master (one for PCIe, one for EBI) for this purpose. Accesses to an atomic table might be interleaved with accesses to non-atomic tables or single registers that are outside of the frame handler without causing problems to either type of access as long as it is understood that the non-atomic accesses might occur out of sequence with the atomic accesses.

8.4 Little and Big Endian Support

All registers, regardless of their width, are 32-bit aligned in the memory map. As an example, a 64-bit register is not necessarily aligned on a 64-bit boundary. And all registers greater than 32 bits are accessed least significant word first. As an example, a 128-bit register would be accessed in the following manner:

```
Address X+0: DATA[31:0]   (least significant word)
Address X+1: DATA[63:32]
Address X+2: DATA[95:64]
Address X+3: DATA[127:96] (most significant word)
```



Any large entity in a single large register (such as a 48-bit MAC address) is stored as a multiple of 32-bits where each 32-bit entity contains up to four bytes and where the least significant byte is assumed to be mapped using the least significant 8 bits of the 32-bit word. For example, the default IEEE assigned LACP frame is stored as follows:

MAC Address = 0x0180C2000002 (LACP FRAME)

	31 24	23 16	15 8	7 0
Address X+0	0xC2	0x00	0x00	0x02
Address X+1			0x01	0x80

The address is transmitted most significant byte first, starting by 0x01 in this case and terminating with 0x02.

This is the natural encoding for a little-endian processor such as the Intel x86 processor family and any large register might be accessed directly. In the case of a big-endian processor, the content of any memory must be accessed 32-bits at a time and reassembled into a 64-bit word manually.

Accessing a 64-bit register with a little-endian processor example follows:

```
long long macAddress = 0x0180C2000002LL;
long long *register_ptr;

*register_ptr = macAddress;
```

Accessing a 32-bit register with a bit-endian processor:

```
long long macAddress = 0x0180C2000002LL;;
long long *register_ptr;

*(((unsigned int *) register_ptr)+0) = macAddress;
*(((unsigned int *) register_ptr)+1) = macAddress >> 32;
```

There are only a few registers where this type of access is required and the effect on the CPU is usually negligible (the Intel API takes care of this).

However, the transfer of packets is not negligible and poses a challenge because the byte ordering within memory is not the same between a little- and big-endian processor. To avoid the processor having to swap bytes, the switch offers an option for byte ordering in the LCI_CFG register.

This bit only affects the interpretation of byte positions within the packet payload words sent to or received from the CPU. In the big endian configuration, successive bytes of a packet must be stored by placing the first byte in the most significant byte location of memory, moving toward the least significant byte. In the little endian configuration, successive bytes of a packet must be stored in the opposite sense, from least significant byte to most. In the case of 32-bit quantities transmitted over a 32-bit bus, how the CPU handles little or bit endian does not matter since all bit fields are defined explicitly. Thus the Tx command and Rx status words (and all other registers in the Intel® Ethernet Switch Family) are defined the same for both little-endian and big-endian CPUs, independently of the LCI_CFG setting.

8.5 CPU Frame Transfer

The FM5000/FM6000 supports packet transfers between the CPU bus interface and the switch fabric as if it were an external Ethernet port. Since the interface only supports slave mode operation, it cannot store or retrieve packet data directly to or from the CPU host memory. Instead, an external bus interface master must individually write or read each word of a packet being sent or received. This external bus master does not need to be the CPU itself. The FM5000/FM6000 series defines three additional bus signals that provide compatibility with certain standalone dual-channel DMA controllers, such as the PLX 9056. Such an arrangement is shown in Figure 8-3.

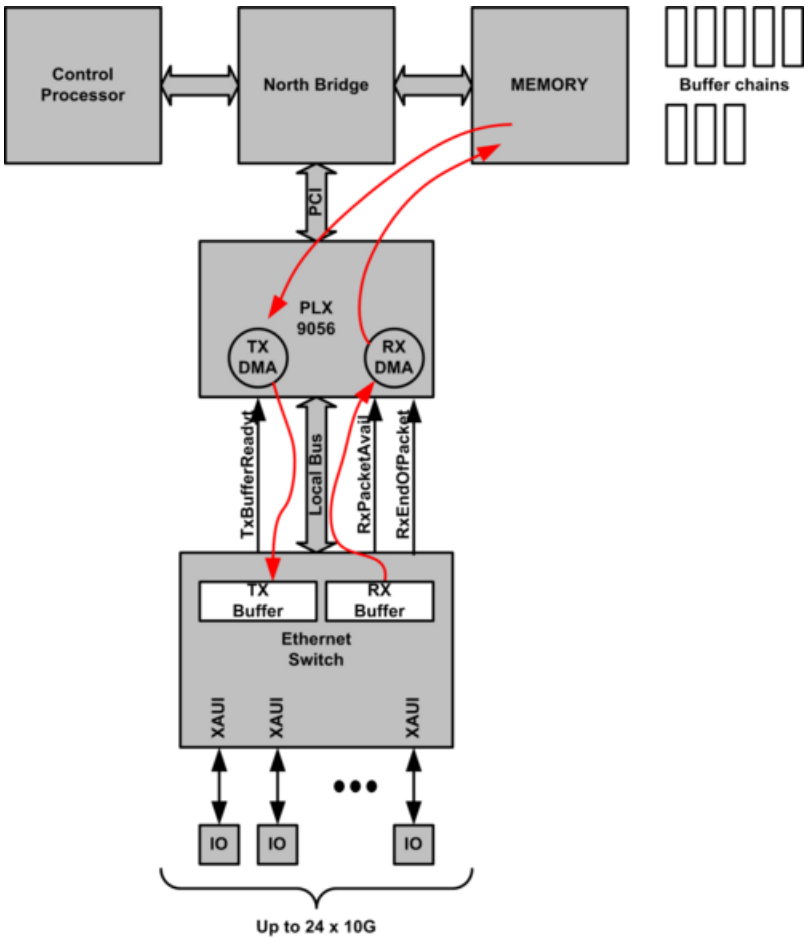
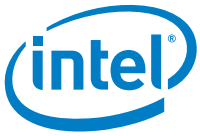


Figure 8-3 DMA Transfer via EBI



8.5.1 Packet Transmission via EBI

The packet format for transmissions on EBI is listed in Table 8-1.

Table 8-1 EBI Tx Packet Format

Word #	Description
0	CONTROL
1	PAYLOAD_WORD[0]
2	PAYLOAD_WORD[1]
...	...
N	PAYLOAD_WORD[N-1]

The first word (CONTROL) defines the length of the packet in bytes. The control word is not sent to the fabric. The following words are the packet payload and are sent to the fabric. Padding (zeros) are added if the length is smaller than minimum configured.

The general protocol for transmitting a packet through the switch is the following:

1. Determine that the transmit pathway is ready to receive a packet by either reading the TxReady bit of the LCI_STATUS register (non-DMA mode) or by observing the status of the TXRDY_N external pin (DMA mode). If this signal is inactive, the bus master must wait, possibly polling TxReady, until the transfer FIFO is ready. This condition never persists for any prolonged length of time (more than a few bus cycles).
2. Write the packet control word to the LCI_TX_FIFO register. The control word identifies the packet length.
3. Write frame payload words to the LCI_TX_FIFO register.

The packets provided to the FM5000/FM6000 must include a properly formatted F64 or F56 ISL tag.

The format of the Tx command word is listed in Table 8-2.

Table 8-2 Format of the EBI TX Command Word

Bit Field	Field Name	Description
15:0	RSVD	Reserved. Write 0x0.
29:16	Length	Length of the packet in bytes.
31:30	RSVD	Reserved. Write 0x0.

8.5.2 Packet Reception via EBI

The packet format for reception on EBI is listed in Table 8-3.

Table 8-3 EBI Tx Packet Format

Word #	Description
0	PAYLOAD_WORD[0]
1	PAYLOAD_WORD[1]
...	...
N-1	PAYLOAD_WORD[N-1]
0	STATUS

The last word (STATUS) defines the length of the packet and if the packet had an error or not.

The general protocol for receiving a packet from the switch is as follows:

- Determine that the switch has received a packet for the CPU, either by observing that the RXRDY_N external pin is low or by polling RxReady in the LCI_STATUS register or by relying on the NewFrameRecv interrupt in LCI_IP. The interrupt is set when a new packet is received and is ready to be read by the CPU on the EBI Interface.
- Read successive words of the packet from the LCI_RX_FIFO register. The last word received is a status word indicating the byte length and error status of the packet. The last word is marked by having the EOT bit set in the LCI_STATUS register before the word is read from LCI_RX_FIFO and by having by asserting RXEOT_N external pin while the word is read from the LCI_RX_FIFO (see Figure 8-4).

The packet received include and F64 or F56 ISL tag. The fields of this tag encode information useful for software:

- Type of frame (normally forwarded versus trapped).
- Source of the packet (in the Source GloRT).
- If the frame was trapped to the CPU, the reason for trapping (in the Destination GloRT).
- Associated system priority and VLAN.

The FM5000/FM6000 supports two levels of padding.

- Level 1 padding is done in the MSB to make the packet 32-bits aligned. This is always done by the hardware automatically regardless of HostPadding bit in LCI_CFG register.
- Level 2 padding is done in the HSM to make the packet 64-bits aligned if the HostPadding bit in LCI_CFG is set to 1b.

The format of the Rx status word is listed in Table 8-4.

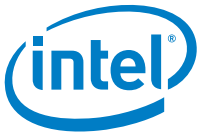


Table 8-4 Format of the EBI RX Status Word

Bit Field	Field Name	Description
0	Error	A value of 1b indicates the packet was corrupted in switch memory due to a parity error. The FM5000/FM6000 series discards any packet addressed to the CPU that is received from the network with a bad CRC.
15:1	RSVD	Reserved. Written as 0x0.
29:16	Length	Length of the packet in bytes.
31:30	RSVD	Reserved. Written as 0x0.

To avoid unreliable frame timeout on packets enqueued to the EBI port, the CPU must dequeue any waiting frames at a rate no slower than one per FRAME_TIME_OUT.timeOutMult * 1024 * PCIE_REFCLK_PERIOD.

8.5.3 Packet Transfer

The EBI packet transfer module supports little-endian and big-endian data encoding to facilitate handling of packets in little-endian and big-endian processors. The control of the data encoding is defined in the LCI_CFG register. The following tables list the encoding for packet transmission and reception in both data encoding modes.

8.5.3.1 Little Endian Packet Transfer

Table 8-5 EBI Little Endian Transmission Format

Phase	31:24	23:16	15:8	7:0
Command Word	Length		0	
Payload	frame[3]	frame[2]	frame[1]	frame[0]
Payload	. . .			
Payload	No	No	No	frame[Length-1]

Table 8-6 EBI Little Endian Reception Format

Phase	31:24	23:16	15:8	7:0
Payload	frame[3]	frame[2]	frame[1]	frame[0]
Payload	. . .			
Payload	No	No	No	frame[Length-1]
Status Word	Length		0	rxStatus

8.5.3.2 Big Endian Packet Transfer

Table 8-7 EBI Big Endian Transmission Format

Phase	31:24	23:16	15:8	7:0
Command Word	Length		0	
Payload	frame[3]	frame[2]	frame[1]	frame[0]
Payload	. . .			
Payload	frame[Length-1]	No	No	No

Table 8-8 EBI Big Endian Reception Format

Phase	31:24	23:16	15:8	7:0
Payload	frame[3]	frame[2]	frame[1]	frame[0]
Payload	. . .			
Payload	frame[Length-1]	No	No	No
Status Word	Length		0	rxStatus

8.6 Packet Transfer DMA Timing

Packet transmission with an external DMA controller is shown in Figure 8-4. The external TXRDY_N signal reflects the state of the TxReady bit the LCI_STATUS register. It is asserted each time the FM5000/FM6000 is ready to accept a word from the CPU. The DMA controller can write data words to LCI_TX_FIFO as long as this signal is asserted.

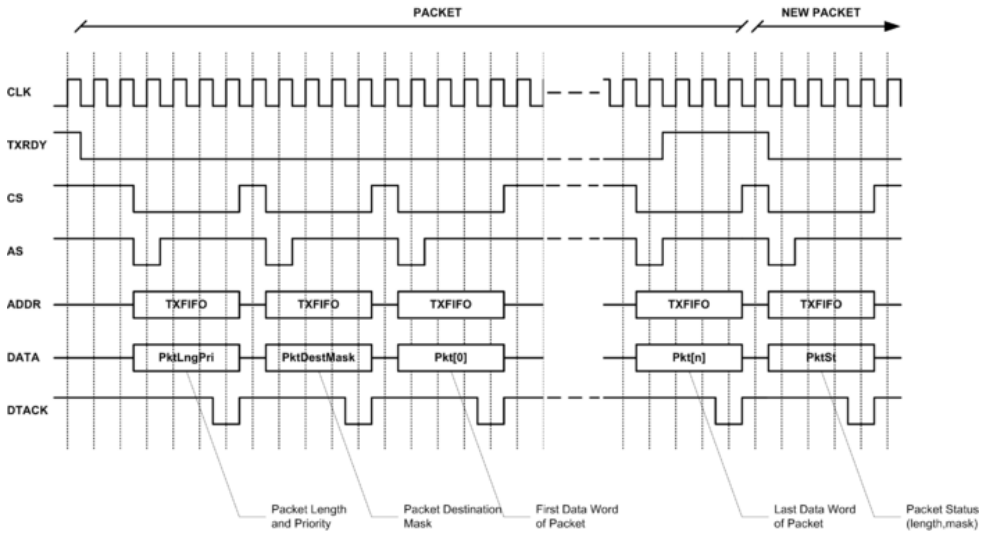


Figure 8-4 DMA Packet Transmission Using EBI

The timing of packet reception with an external DMA controller is shown in Figure 8-5. The RXREQ_N signal reflects the state of the RxReady bit in the LCI_STATUS register. It is asserted each time the FM5000/FM6000 has a data word ready to be read from LCI_RX_FIFO. The DMA controller can read LCI_RX_FIFO as long as this signal is asserted. When the last word of the packet transfer is being read on the bus (the LCI_RX_STATUS word), the FM5000/FM6000 asserts the RXEOT_N signal to indicate the end-of-transfer condition. The EOT signal notifies a DMA controller with buffer chaining support to close the current buffer and proceed to the next one in its descriptor list.

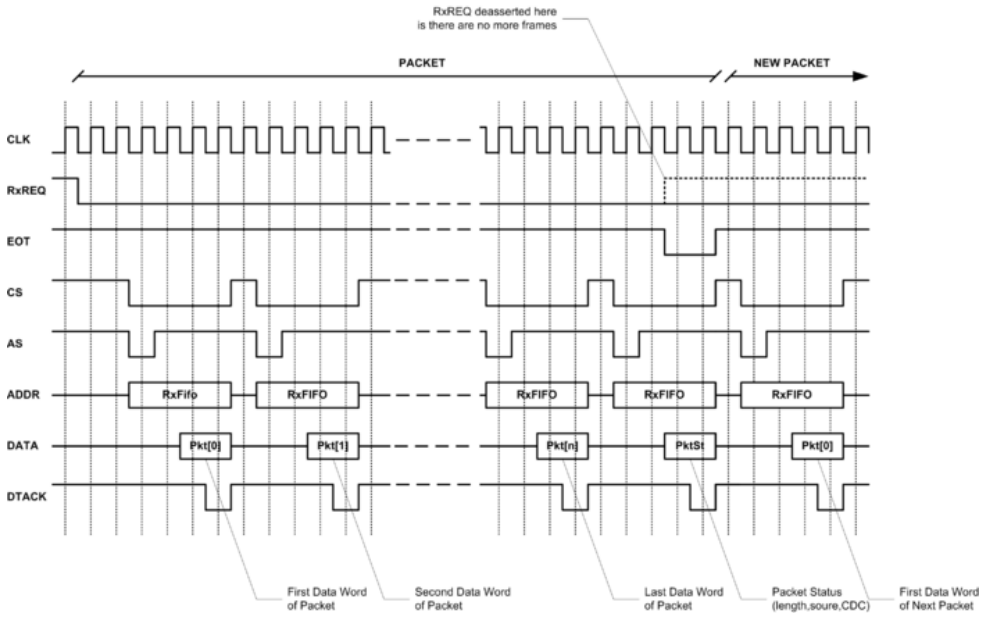


Figure 8-5 DMA Packet Reception Using EBI

NOTE: ***This page intentionally left blank.***

9.0 Peripherals

There are several peripheral interfaces available in the FM5000/FM6000 that can be used for purposes such as reading boot configurations, monitoring and controlling external PHYs, or driving external status LEDs.

9.1 Overview

Peripherals include the following elements:

- Clocking
- Counter Rate Monitor
- SerDes Management
- I^{2C} Controller
- MDIO Controller
- General Purpose IO (GPIO) Controller
- SPI Interface
- LED Controller
- JTAG Interface

9.2 Clocking

The FM5000/FM6000 clock distribution is shown in Figure 9-1.

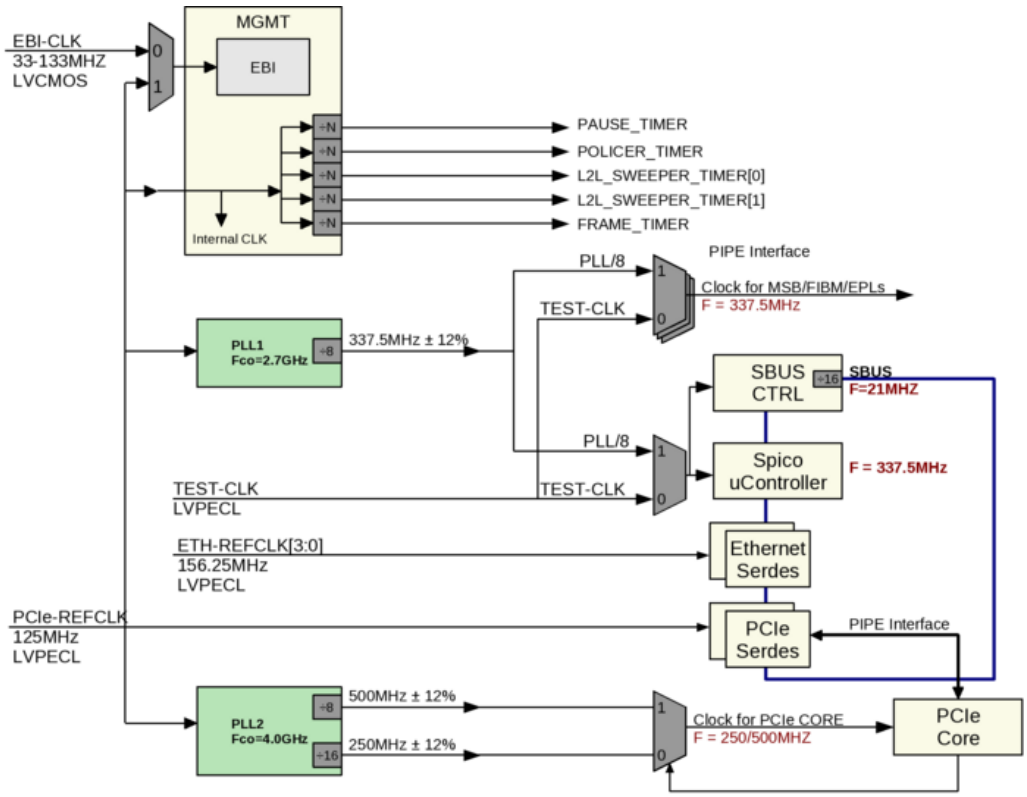
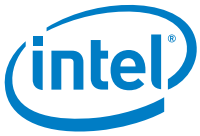


Figure 9-1 Switch Clocking

There are seven input clocks possible:

- Four required Ethernet reference clocks:
  - 156.25 MHz
  - LVPECL
  - One reference clock used per set of 6 x EPLs
- Required PCIe reference clock:
  - 125 MHz
  - LVPECL
- EBI clock:
  - For local bus interface
  - Only required if EBI used, ground both EBI_CLK and CS_N if not used
  - LVCMOS
- Test clock:
  - For test only
  - LVPECL



Note: The LVPECL reference clocks input support an internal termination that can be turned off. See LED_DATA[2:0] pins for details (Section 3.3.9).

The switch has two programmable PLLs: one for generating 337.5 MHz and one for generating 250 MHz and 500 MHz. Both PLLs have default configurations to produce the right frequency after reset and do not need to be re-configured. The default configuration for each can be overridden if desired using the PLL_CTRL register. The multipliers and dividers for each PLL must be programmed such that the PLL operates within ±12% of its core frequency (2.7 GHz for PLL1 and 2.0 GHz for PLL2). The PLL1 provides the normal clock source for MSB, SPICO, EPLs MACs, and SBUS controller modules. A set of clock multiplexers for each of these modules enables selecting an alternate external clock source if desired. The PLL2 provides the normal source for PCIe core. In this case, two clocks are produced and are in sync to each other; 500 MHz and 250 MHz. These clocks are used for speed negotiation between Gen1 and Gen2 PCIe.

Note: The PLL2 can potentially be reconfigured but it is not advised to do so as it has to match the PCIe serdes rate.

For EBI, the selection depends on the state of the CS_N pin at reset. If this pin is low, it indicates that the EBI is not used and the EBI module is clocked using the PCIE_REFCLK. If the pin is set high, it indicates that the EBI is used and the EBI module is clocked using the EBI_CLK. For the other modules, the selection is done by software via the PLL_CTRL register, which is also accessible via the JTAG interface.

PLL1 configuration:

```
Fco = (PCIE_REFCLK/divider) x 2 x ( Multiplier )
Fout = Fco / 16

Default multiplier => 54 (PLL_CTRL.Multiplier1)
Default divider => 5 (PLL_CTRL.Divider1)

Fco = (125MHZ/5) x 2 x 54 => 2.7GHZ
Fout = 2.7 / 8 => 337.5MHZ
```

PLL2 configuration:

```
Fco = (PCIE_REFCLK/divider) x 2 x ( Multiplier )
Fout1 = Fco / 8
Fout2 = Fco / 16

Default multiplier => 16 (PLL_CTRL.Multiplier2)
Default divider => 1 (PLL_CTRL.Divider2)

Fco = (125MHZ/1) x 2 x 16 => 4.0GHZ
Fout1 = 4.0 / 8 => 500.0MHZ
Fout1 = 4.0 / 16 => 250.0MHZ
```

The manageability module also produces the following reference timers used by other modules:

- PAUSE
- POLICERS
- L2 LOOKUP SWEEPERS
- FRAME TIMEOUT

The frame timeout period is programmed in FRAME_TIME_OUT register while the other timers are programmed in the SWEEPER_CFG register.

9.3 Counter Rate Monitor

The FM5000/FM6000 provides general counter rate monitoring and memory management functions to reduce the need for the CPU to perform repetitive periodic operations in the switch. This module's primary purpose is counter/state monitoring, but it includes other services as well. The services offered are:

- Monitor rate counter changes (too fast or too slow)
- Report state changes
- Monitor queue sizes
- Report max queue sizes
- Copy memory blocks
- Initialize memory blocks
- Compute checksums

The CRM structure is divided into four sets of registers:

- 2048 data set memory
- 64 commands definition registers
- Control register
- Interrupt registers

The control and status registers are used to control the CRM globally. The control register is used to start and stop the CRM and define the index range of the sequence of commands to execute. The status register is used to define the next command to execute (before start) or report current one being executed (while running) and the current state (running/stopped) of the CRM module.

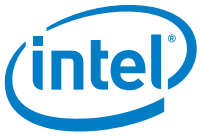


Table 9-1 CRM Control/Status Registers

Register	Sub-field	Width	Definition
CRM_CTRL	Run	1	Start or stop the CRM. Stopping might require some time as the current command has to complete.
	FirstCommandIndex	6	Index of first command of the sequence.
	LastCommandIndex	6	Index of first command of the sequence.
	Continuous	1	Defines if the sequence of commands is executed continuously or only once. 0b = Only once 1b = Continuously If executed only once, the run bit is cleared at the end of the sequence.
	TickPrescale	4	Defines (in power of two) the pre-scaler to apply to the PCIE_REFCLK to produce the reference time for the CRM.
CRM_STATUS	Running	1	(READ ONLY) Indicates if CRM is running or stopped. 0b = Stopped 1b = Running ***Note:*** Stopping the CRM (turning Run bit to 0) might be delayed until the current command being executed completes.
	CommandIndex	6	If stopped, this is the index to the next command to execute. If stopped because the sequence completed, the command index is CommandIndex 6 reset to the FirstCommandIndex. The value must be set to any command in the sequence before starting the CRM. If running, this is the index of the current command being executed.
CRM_TIME	Tick	32	Current tick counter.

The command registers are used to define commands. These commands can have any data size set associated with each one. When running, the CRM executes the sequence of commands defined in the CRM_CTRL register, once or repetitively. Each command has its own interrupt bit in CRM_IP that can be used to post an interrupt the processor. Each interrupt is maskable using the CRM_IM register.

It is possible for the CPU, at any time, to stop the current monitoring process and execute a special command (or series of commands). The process would be:

1. Stop CRM
2. Wait for CRM to indicate it has stopped.
3. Save the index of the next command to execute.
4. Load a temporary list of commands (possibly by using unused command space).
5. Launch execution of that new program (single execution), wait for its completion by polling corresponding CRM_IP bit for this command (or waiting for interrupt).
6. Re-active the previous program at the place it was stopped.

The CRM can only be stopped at a command boundary. If a single command is going through a large data set, the CRM might take a while before stopping.

Table 9-2 lists the CRM commands register set.

Table 9-2 CRM Commands Register Set

Register	Sub-field	Width	Definition
CRM_COMMAND[63:0]	Command	3	Defines the command to execute (see Table 9-3).
	DataIndex	11	Pointer of the data section for this command. Not used for all commands.
	Count	20	Defines the number of registers to walk through by this command. If the command needs a data section, the count is limited to Count 20 2048 maximum (total size of data set). If the command doesn't need a data section (such as a memory set), the limit is 1,048,575. The count is the number of registers, not the number of bytes or words.
CRM_REGISTER[63:0]	BaseAddress	22	First register address.
	Size	2	Defines register size (0=32-bit, 1=64-bit, 2=96-bit, 3=128-bit).
	BlockSize1Shift	4	Defines register block size in power of 2. Block size is 1<<BlockSize1Shift.
	Stride1Shift	4	Defines stride to next block in power of 2. Stride is 1<<Stride1Shift.
	BlockSize2Shift	4	Defines second level register block size in power of 2. Block size is 1<<BlockSize2Shift.
	Stride2Shift	4	Defines second level stride to next block in power of 2. Stride is 1<<Stride2Shift.
CRM_PERIOD[63:0]	Interval	32	Defines time interval before executing this command. The timebase (CRM tick) is a pre-scale PCIE_REFCLK (1 to 65536 in power of two). A value of 0x0 is as fast as possible.
	LastTick	32	Defines last time the command was executed.
CRM_PARAM[63:0]	Param	32	Defines the parameters for each command. Interpretation depends on the command.

The data set is a two dimensional array of 32-bit data, CRM_DATA[0..2047,0..1]. The first index is the data pointer (as defined in the CRM_COMMAND register) while the second is interpreted depending of the command used. Table 9-3 lists the commands.

Table 9-3 CRM Command Definition

#	Command	Definition	CRM_PARAM	CRM_DATA[0]	CRM_DATA[1]	Interrupt
0	Memory Set	Initialize a memory block.	Value to set[1]	N/A	N/A	Set after initialization completes.
1	Memory Copy	Copy a memory block from one location to another location.	Destination register (least significant 22 bits used only)	N/A	N/A	Set after copy completes.
2	Monitor Rate Increase	Increment counter if a register changed by more than a certain limit. If the register size is greater than 32, CRM reads the entire register, but uses only the least significant 32 bits for this operation.	Limit	Last value	Count	Set if at least one counter in the data set exceeded the last value.

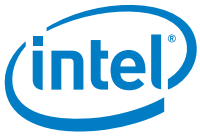


Table 9-3 CRM Command Definition (Continued)

#	Command	Definition	CRM_PARAM	CRM_DATA[0]	CRM_DATA[1]	Interrupt
3	Monitor Changes	Check for change and save it. If the register size is greater than 32, CRM reads the entire register, but uses only the least significant 32 bits for this operation.	Mask	Last value	Tick at change	Set if at least one counter in the data set changed.
4	Save Max	Compare value to last and save if max. If the register size is greater than 32, CRM reads the entire register, but uses only the least significant 32 bits for this operation.	Mask	Maximum value	Tick at change	Set when a new maximum is found.
5	Monitor Rate Stagnant	Increment counter if register change by less than a certain limit. If the register size is greater than 32, CRM reads the entire register, but uses only the least significant 32 bits for this operation.	Limit	Last value	Count	Set if at least one counter in the data set is not incrementing fast enough.
6	Count if greater or equal	Count if greater or equal. Increment counter if register is greater or equal to limit. If the register size is greater than 32, CRM reads the entire register, but uses only the least significant 32 bits for this operation.	Mask	Limit	Count	Set if at least one counter in the data set is increased.
7	Compute Checksum	Read table and compute a 32-bit 1's complement checksum. If the register size is greater than 32 bits, each 32-bit word is added to the checksum.	Expected checksum	N/A	N/A	Set if checksum computed does not match checksum expected.

1. The CRM_PARAM register is a 32-bit register, for the SET MEMORY command, this value is replicated to match the size of the set (32,64,96,128).

A single command might be interested in a set of registers that is not contiguous in memory. For this reason, the register definition includes two levels of block sizes and strides that enable a single command to efficiently be performed over a large set of registers.

The basic pseudo code of the overall CRM module is:

```
while (1)

if ( Run )

# Check if time to run this command
if ( tick - CRM_PERIOD[CommandIndex].LastTick > CRM_PERIOD[CommandIndex].Interval )
   if ( tick - CRM_PERIOD[CommandIndex].LastTick > CRM_PERIOD[CommandIndex].Interval*2 )
      CRM_PERIOD[CommandIndex].LastTick = tick
   else
      CRM_PERIOD[CommandIndex].LastTick += CRM_PERIOD[CommandIndex].Interval

   baseAddress = CRM_REGISTER[CommandIndex].BaseAddress
   count0 = 0
   count1 = 0
   count2 = 0
   if (registerSize == 0)
      size0=1
   else if (registerSize == 1)
```

```
      size0=2
   else if (registerSize == 2)
      size0=4
   else if (registerSize == 3)
      size0=4
   size1 = 1<<CRM_REGISTER[CommandIndex].BlockSizeShift1
   size2 = 1<<CRM_REGISTER[CommandIndex].BlockSizeShift2
   stride1 = 1<<CRM_REGISTER[CommandIndex].StrideShift1
   stride2 = 1<<CRM_REGISTER[CommandIndex].StrideShift2

   foreach "i" in (0..CRM_COMMAND[CommandIndex].Count)

      # Compute address
      address = baseAddress + count0 * size0 + count1 * stride1 + count2 * stride2

      # Execute command at address "address"
      switch (CRM_COMMAND[CommandIndex].Command)
         case 0:
            # Set memory
         case 1:
            # Copy memory
         case 2:
            # Monitor rate too fast
         case 3:
            # Monitor changes
         case 4:
            # Monitor max
         case 5:
            # Monitor rate too slow
         case 6:
            # Count changes greater than limit
         case 7:
            # Add register content to checksum

      # Advance counter
      count0++
      if ( count0 == size1 )
         count0 = 0
         count1++
         if ( count1 == size2 )
            count1 = 0
            count2++

# Pass to next command
if ( CommandIndex = LastCommandIndex )
   CommandIndex = FirstCommandIndex
else
   CommandIndex++
```

9.4 SerDes Management

All SerDes (96 for Ethernet and 4 for PCIe) are linked together using a low frequency SBUS.

Note: The SBus IDs for the PCIe SerDes are in the range 1-4, while the SBus IDs for the EPL SerDes are in the range 5-100. The offset of 1 for the PCIe SerDes, and the offset of 5 for the EPL SerDes are incorporated in the SerDes register definitions. This means that the SerDes macro definitions expect an index in the range 0-3 for the PCIe SerDes, and an index in the range 0-95 for the EPL SerDes.

The controller for the serial bus is located in the JSS unit along with a micro-controller and a JTAG controller.

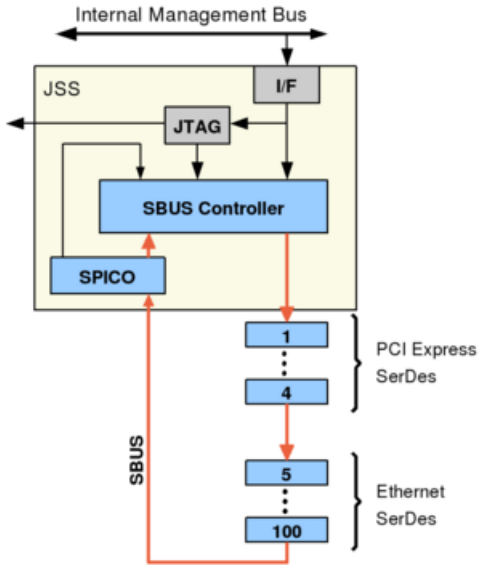


Figure 9-2 SerDes Serial Bus Interface

The SBUS controller supports three masters; host (via manageability module), JTAG (for manufacturing purpose) and a local micro-controller. The local micro-controller is also part of the SBUS, enabling the host to access the micro-controller (such as downloading code, reading or writing registers, etc.).

The role of the micro-controller is to provide SerDes management and free up the host processor from timing critical tasks such as DFE tuning.

Note: The SPICO micro-controller should be accessed using the reserved SBus ID 0xFD, while the SBus controller should be accessed using the reserved SBus ID 0xFE.

The SBUS is a slow bus with 101 end points. As a result, the access time to any register in any SerDes is long, up to 6 µs. To prevent the internal management bus of the switch to be on hold for that long, the manageability module provides a command register that latches a command issued from a host (local or remote) and frees up the internal bus while the command is being executed. The host might poll the command register at a later time to see if the command executed. The host must wait for a previous command to complete before a new one can be issued.

The command register contains:

- Device (1..4 = PCIe SerDes, 5..96 = Ethernet SerDes, 101 = SPICO).
- Register (0..255)
- Operation (read, write, reset)
- Execute (transition from 0 to 1 starts the command)
- Busy (0 = idle, 1 = executing a command)

The transaction on the serial bus starts as soon as the command is written. A read back from the register returns a busy bit to indicate that the command executed. The busy bit is cleared when the command execution completes.

The software access for a read back is:

```
// Start command
SBUS_COMMAND = (EXECUTE_BIT) + (READ << 16) + (device << 8) + (register);

// Wait for command to complete
while (SBUS_COMMAND & BUSY_BIT) yield();

// Read data
data = SBUS_RESPONSE;

// Clear register for next command
SBUS_COMMAND = 0;
```

The software access for a write command is:

```
// Write data
SBUS_REQUEST = data;

// Write command
SBUS_COMMAND = (EXECUTE_BIT) + (READ << 16) + (device << 8) + (register);

// Wait for command to complete
while (SBUS_COMMAND & BUSY_BIT) yield();

// Clear register for next command
SBUS_COMMAND = 0;
```

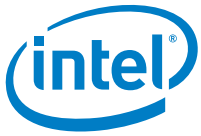
Writes to command registers are ignored while a command executes. Reading any register returns the current value; however, the SBUS_RESPONSE content is undefined while a READ command executes.

The switch management bus supports multiple masters in the system (PCIe, EBI, etc...) and a problem might arise if two masters try to read different SerDes registers.

Local processor on PCIe	Local processor on EBI
SBUS_COMMAND = read(A)	
.	SBUS_COMMAND = read(B)
.	.
.	.
.	.
.	y = SBUS_RESPONSE
.	
x = SBUS_RESPONSE	

In this case, the command read(B) is ignored and the variable y contains the result of read(A), which is not the desired objective. This problem is common for many registers in the switch and there is no logic in the switch to prevent this. To avoid this problem, only one master should issue a read or write command at a time and complete it before any other master issues a read command.

The SBUS_CFG register defines the reset state of the SBUS controller and the clock ratio. The clock ratio should be set to 4.



9.4.1 SPICO Micro-controller

The SBUS_SPICO register controls the SPICO controller.The SPICO controller can be in any of the following three states at any time.

Reset

SPICO controller is in reset and all internal circuits reset to the their default state. Software cannot be downloaded while the SPICO controller is in this state.

Disabled

SPICO controller is out of reset but the micro-processor is not running. Software might be downloaded while the SPICO is in this state.

Enabled

SPICO controller is out of reset and executing code.

The bootstrap process is:

- Place the SPICO in reset (default after chip reset)
  - SBUS_SPICO.Reset = 1b
  - SBUS_SPICO.Enable = 0b
- Take SPICO out of reset
  - SBUS_SPICO.Reset = 0b
- Download software
  - Instructions to be supplied, uses a series of SBUS_COMMAND=WRITE (device = 101,...)
- Start micro-controller
  - SBUS_SPICO.Enable = 1b

9.4.2 SerDes Registers

The SerDes registers are documented in the register section. The operations for each type of SerDes, PCIe or Ethernet, are documented in the PCIe and Ethernet Port Logic sections, respectively.

9.4.3 Device Address to Serdes Map

Table 9-4 lists the SBUS to SerDes/EPL mapping as well as the SBUS ordering on the serial bus. Both the EPLs and the PCIe module have four SerDes each. The PCIe SBUS is the first one on the ring and covers the SBUS address range 1 to 4, followed by EPL[1] SBUS, which covers the address range 5 to 9 and so on.

Table 9-4 SBUS to SerDes/EPL Mapping

Interface	SBUS Order	SBUS Address
PCIe	1	1
EPL[1]	2	5
EPL[3]	3	9
EPL[5]	4	13
EPL[6]	5	17
EPL[16]	6	21
EPL[17]	7	25
EPL[18]	8	29
EPL[19]	9	33
EPL[20]	10	37
EPL[14]	11	41
EPL[9]	12	45
EPL[11]	13	49
EPL[13]	14	53
EPL[15]	15	57
EPL[21]	16	61
EPL[22]	17	65
EPL[23]	18	69
EPL[24	19	73
EPL[2]	20	77
EPL[4]	21	81
EPL[6]	22	85
EPL[8]	23	89
EPL[10]	24	93
EPL[12]	25	97

9.5 I²C Controller

The FM5000/FM6000 contains one I²C controller. The I²C controller supports the following features:

- 100 KHz or 400 KHz operation.
  - The speed is programmable through the I2C_CFG register and the switch supports a larger set.
  - Slave mode enabling external masters to read/write registers in the chip.
  - All registers are accessible except for the LCI_TX_FIFO and LCI_RX_FIFO registers. Those have special handling and are not accessible from the I²C. As a result, it is not possible to send or receive frames via I²C.
- Master mode enabling the switch to access external I²C devices.
- Configurable I²C slave address.
- Boot configuration from I²C serial EEPROM.
- Bus arbitration.
  - The I²C controller attempts to gain the bus only once and returns an error if it didn't succeed. Software needs to retry at a later time.
- 7-bit addressing mode.

The I²C controller doesn't support the following features found in some I²C devices:

- SMBus
- General call address.
- 10-bit addressing mode.
- Start byte.
- Mix-speed mode.
- High-speed mode.

Figure 9-3 shows the basic read and write accesses.

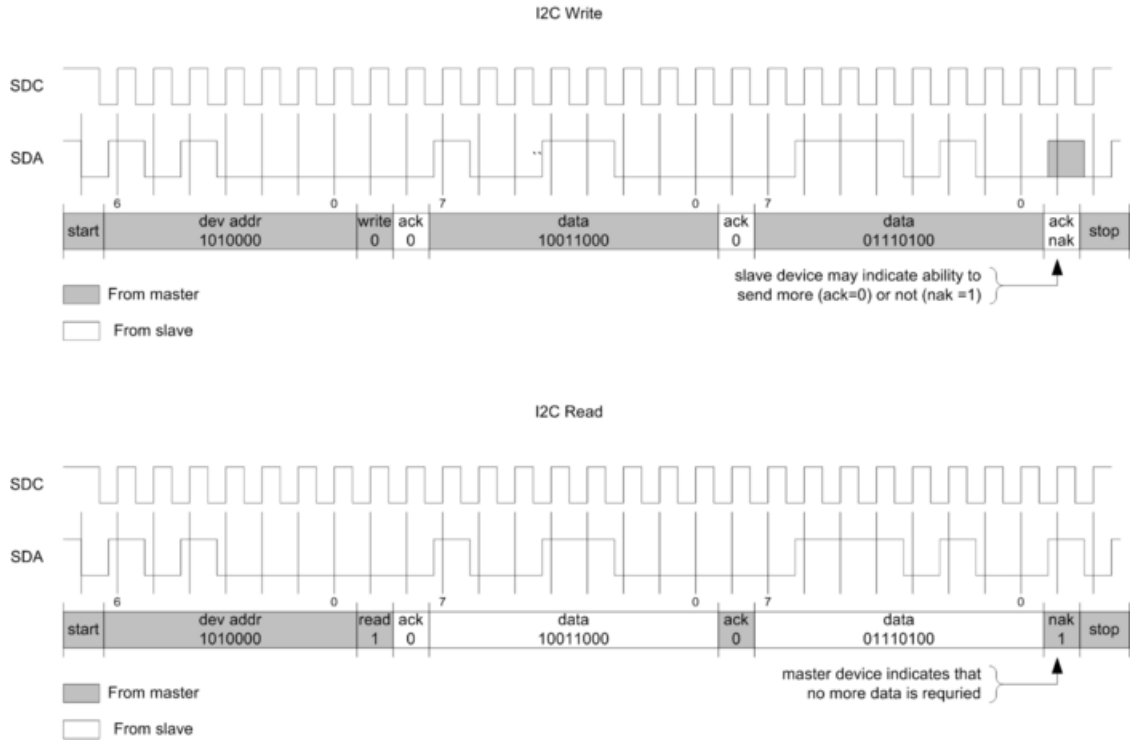


Figure 9-3 I2C Basic Accesses

The SDA and SDC pins are open drain with external pull ups. This structure has the disadvantage of creating slow rising edges that can potentially be incorrectly interpreted as a double transition. To prevent this, the FM5000/FM6000 incorporates a digital filtering circuit to ensure that only complete transitions of either SDC or SDA are detected. The filter operates by sampling the SDA and SDC using the EBI clock rate and only reports transitions that are stable for at least 16 cycles.

The I^2C controller supports a timeout of 100 µs (1/10 of the I^2C clock rate) that aborts the current cycle and returns all pins to 1.

In the slave mode, an external agent on the I^2C bus can access any register of the chip as the CPU does.

The slave mode has the following characteristics:

- Slave address is user configurable and defaults to 1000xxx (0x40-0x47) where xxx is defined by the I2C_ADDR configurations strapping pins (derived from pull up or pull down on the DMA pins).
- All write accesses start with a 3-byte address field to indicate which address (register or table) to read followed by N 4-byte data words. Address and data must be sent MSB first. Data words are written into consecutive addresses starting with the address given. The address is incremented at the end of the data transfer. The master is at liberty to write any number of words and it is assumed that the master never attempts to write into an illegal address. It is possible for a write access to only include the 3-0 byte address without any data words. This is used to load an address into the chip for an eventual read (note that the address is saved only if 3 address bytes are sent).

- All read accesses return consecutive 4-byte words starting with the last address used during a write cycle. The actual register is latched just before the first byte of the word that was sent. The master is at liberty to read any number of words and terminates with a NAK on the last byte desired, which could any byte.
- The controller supports restart cycles (a stop-start).

These characteristics are shown in Figure 9-4.

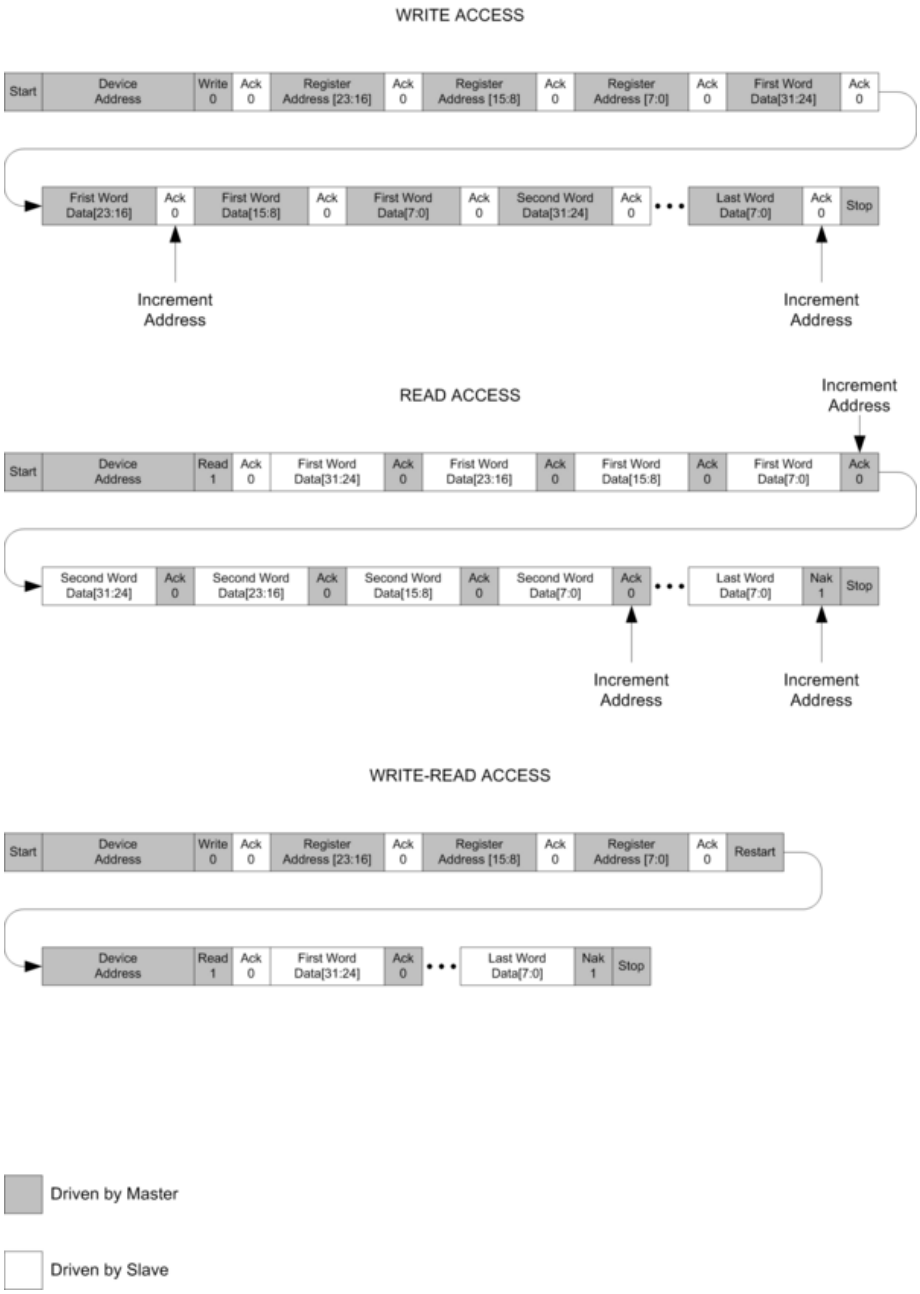


Figure 9-4 Access to Internal Registers via I2C

In the master mode, the I^{2}C controller is capable of issuing automatic I^{2}C accesses:

- **Write** — Can write up to 8 bytes
- **Write-read** — Can write up to 4 bytes and receive up to 4 bytes
- **Read** — Can read up to 8 bytes

The I^{2}C registers are:

- **I2C_CFG**
  - **Enable** (1 bit) — Defines if the switch answers to an I^{2}C address from an external master or not
  - **Address** (7 bits) — Defines the address to which the switch answers
  - **Divider** (12 bits) — Defines clock rate as a divider of the PCIE_REFCLK base clock (which is CPU_CLK divided by 2)
  - **Filter** (5 bits) — Defines the number of clocks for data to be stable before being recognized. This register is used to filter glitches on I^{2}C with bad slew rate.
- **I2C_DATA_W** (32 bits) — Contains the first word for write or write-read commands.
- **I2C_DATA_RW** (32 bits) — Contains the second word for the write or the read for the write-read or read commands.
- **I2C_CTRL** — Used when I^{2}C controller is master
  - **Address** (8 bits) — Defines the address of the device to access (lower bit ignored)
  - **Command** (2 bits) — Execute command write, write-read, read, null.
  - **LengthW** (4 bits) — Defines the number of bytes to send (0..12)
  - **LengthR** (4 bits) — Defines the number of bytes to read (0..12)
  - **LengthSent** (4 bits) — Number of bytes actually written (0..LengthW). Defines the length sent. It is shorter than LengthW if and only if a NAK is received prematurely.
- **Status** (4 bits) — Defines the status of the command (completed, aborted, etc...)

The possible transactions are shown in Figure 9-5.

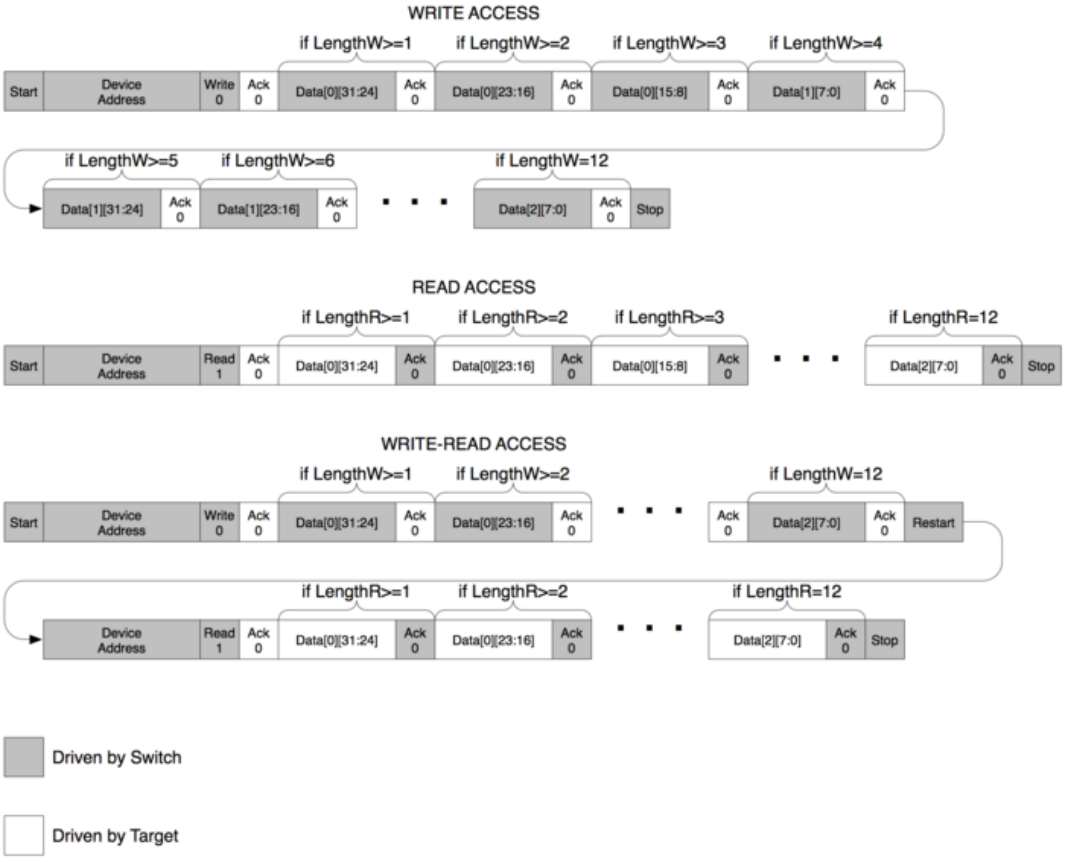


Figure 9-5 Combine I2C Accesses

9.6 MDIO Controller

The FM5000/FM6000 contains one MDIO controller. The MDIO controller is designed to enable the CPU to access MDIO devices through the switch. Features include:

- Support for clauses 22 and 45.
- Master only. The switch cannot be the target for any access.
- 3.3V/2.5V level compatibility only. An external level converter is required for access to 1.2V MDIO as defined in the IEEE802.3ae specification.

The clause 22 format is shown in Figure 9-6.

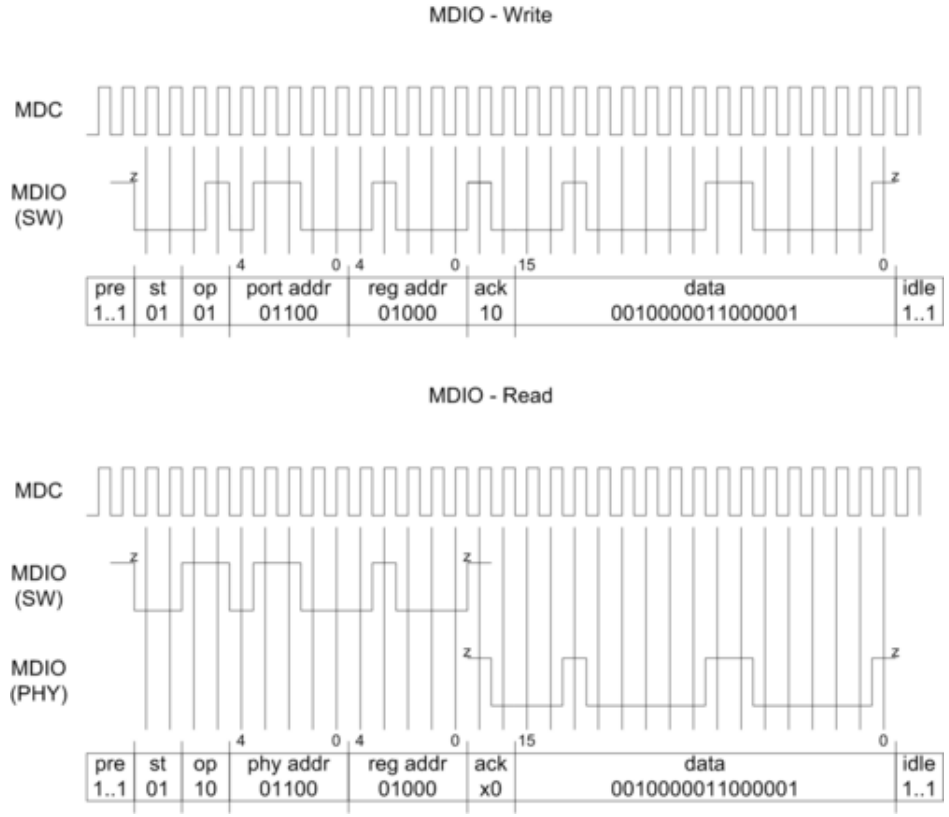


Figure 9-6 Clause 22 MDIO Transaction Format

Clause 45 frame format is shown in Figure 9-7.

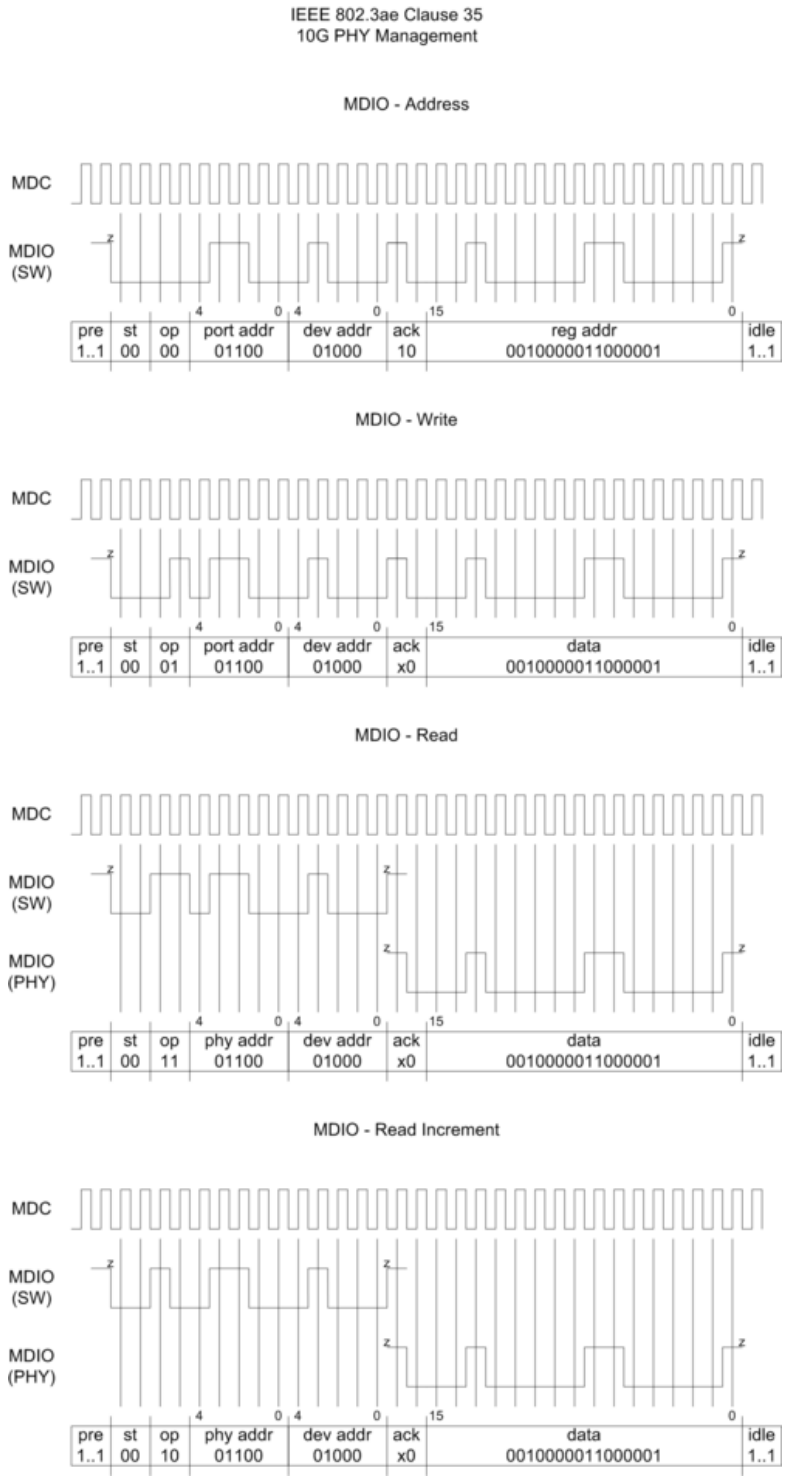


Figure 9-7 Clause 45 MDIO Transaction Format

The register settings that support these transactions are:

- **MDIO_CFG**
  - **Divider** (12 bits) — Defines the clock divider (from PCIE_REFCLK clock)
  - **Preamble** — Defines if the 32-bit pre-amble is always sent or not.
- **MDIO_DATA** — The data sent or read. Only 16 bits.
- **MDIO_CTRL**
  - **PHY Address** (5 bits)
  - **Device Address** (5 bits) — Note that in 1 GbE mode this field becomes the register field.
  - **Register Address** (16 bits) — Note that in 1 GbE mode this field is unused.
  - **Command** (2 bits)
    - **Null**: Do nothing
    - **Write**: Send register address frame and then write frame.
    - **Sequential-read**: Send read command frame only.
    - **Random-read**: Send register address frame followed by a read frame.
  - **Device Type** (1 bit) — Defines if the frame format is compatible with clause 22 (0) or clause 45 (1). For the clause 22, the commands sequential-read and random-read are exactly equivalent.
  - **Status** — Returns the status of the command.

9.7 General Purpose IO (GPIO) Controller

The GPIO controller is 16-bits wide and supports:

- Input, output, open-drain.
- Interrupts per bit
  - Configurable for low to high or high to low or both.

The following registers are defined for the controller:

- **GPIO_DATA**
- **GPIO_CTRL**
- **GPIO_IP**
- **GPIO_IM**

All GPIO pins are defaulted as inputs after reset and sampled to determine default hardware options. If BOOT_MODE is set to SPI, the SPI controller is enabled and GPIO[3,4,5] become outputs immediately superseding the default state and also superseding the GPIO_CTRL configuration for those bits until boot completes. Similarly, if software enables the SPI controller, the same pins GPIO[3,4,5] are taken over by the SPI controller as well, and the GPIO_CTRL setting is ignored for those pins.

Table 9-5 lists the GPIO pin strapping options and default reset states.

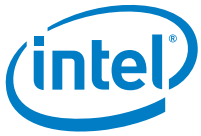


Table 9-5 GPIO Strapping Options

GPIO Pin	Strapping Option	After Reset SPI Disabled	After Reset SPI Enabled
0	DTACK_INV	Input	Input
1	RW_INV	Input	Input
2	IGNORE_PARITY	Input	Input
3	N/A	Input	Output (SPI_CLK)[1]
4	N/A	Input	Output (SPI_CS_N)[1]
5	N/A	Input	Output (SPI_MOSI/SPI_IO0)[1]
6	N/A	Input	Input (SPI_SPI_MISO/SPIO_IO1)
7	BOOT_MODE[0]	Input	Input
8	BOOT_MODE[1]	Input	Input
9	BOOT_MODE[2]	Input	Input
10	PARITY_EVEN	Input	Input
11	I2C_ADDR[0]	Input	Input
12	I2C_ADDR[1]	Input	Input
13	I2C_ADDR[2]	Input	Input
14	DATA_HOLD	Input	Input (SPI_IO3]
15	Not used	Input	Input (SPI_IO2]

1. return to input after boot completes unless boot command include enabled SPI_CFG.

9.8 SPI Interface

9.8.1 Overview

The SPI interface is an optional serial interface that can be used by the switch to upload a default configuration after reset. The SPI interface is multiplexed with general purpose I/Os and is automatically activated if the BOOT_MODE strapping options are set to boot from SPI. After configuration, the general purpose I/O pins revert to normal operation. The SPI interface also offers a management interface enabling a CPU, local or remote, to initiates transactions on the SPI bus. The SPI interface supports normal read mode (single pin), a dual-pin mode and a quad-pin mode. Speed can be up to 62.5 MHz (1/2 of 125 MHz reference clock), the default speed is 500 KHz.

The signals are listed in Table 9-6.

Table 9-6 SPI Signals

Signal Name	Signal Direction	Signal Description
SPI_CS_N	Out	SPI chip select (active low).
SPI_SCK	Out	Clock for SPI interface. Maximum is 62.5 MHz.
SPI_MOSI/IO0	Out	Serial Data Output (MOSI, Master-Out- Slave-In, since the FM5000/FM6000 switch is master). Connect to serial data input of serial EEPROM/FLASH. Also used as a serial data input (IO0) when operating in dual-pin mode.

Table 9-6 SPI Signals (Continued)

Signal Name	Signal Direction	Signal Description
SPI_MISO/IO1	In	Serial Data Input (MISO, Master-In-Slave-Out, since the FM5000/FM6000 switch is master). Connect to serial data output of serial EEPROM/FLASH. Also used as a second serial data input (IO1) when operating in dual-pin mode.
SPI_IO2	In	Serial Data IO2. Used as a third serial data input in quad-pin mode.
SPI_IO3	In	Serial Data IO3. Used as a fourth serial data input in quad-pin mode.

9.8.2 Boot

The SPI configuration upload is a two step process. First it retrieves the base address of the boot image and second it retrieves and executes the boot image. The actual location of the base address of the image depends on the BOOT_MODE strapping option and enables multiple images (up to four) to be stored in the same Flash.

Figure 9-8 shows the two transactions being performed in a single-pin mode.

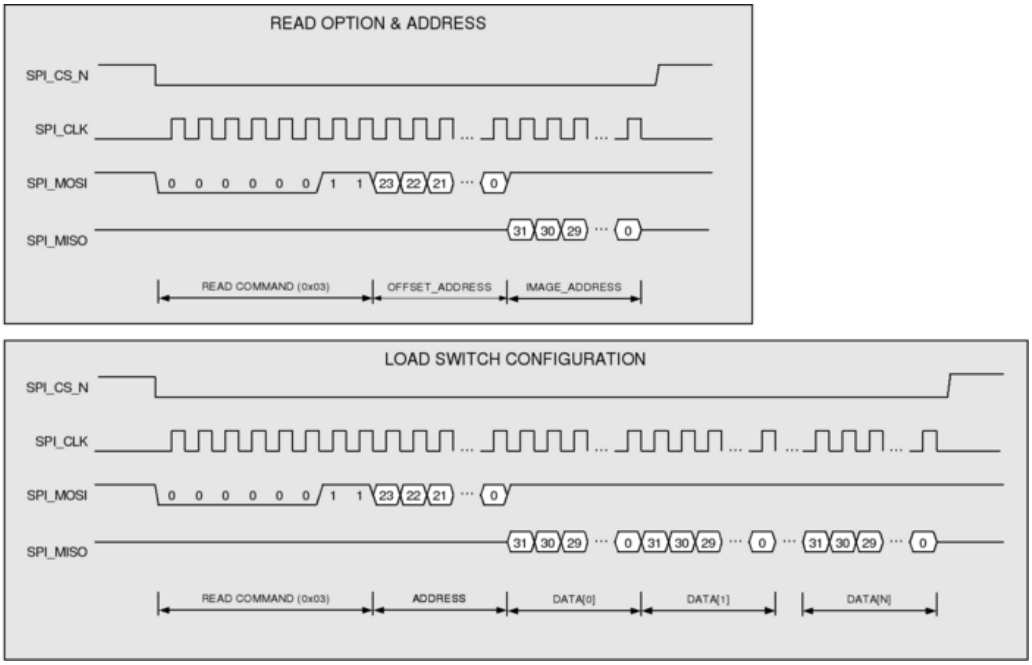


Figure 9-8 SPI Boot Access

Figure 9-9 shows the same two transactions with the second transaction as a request to be in dual-pin mode.

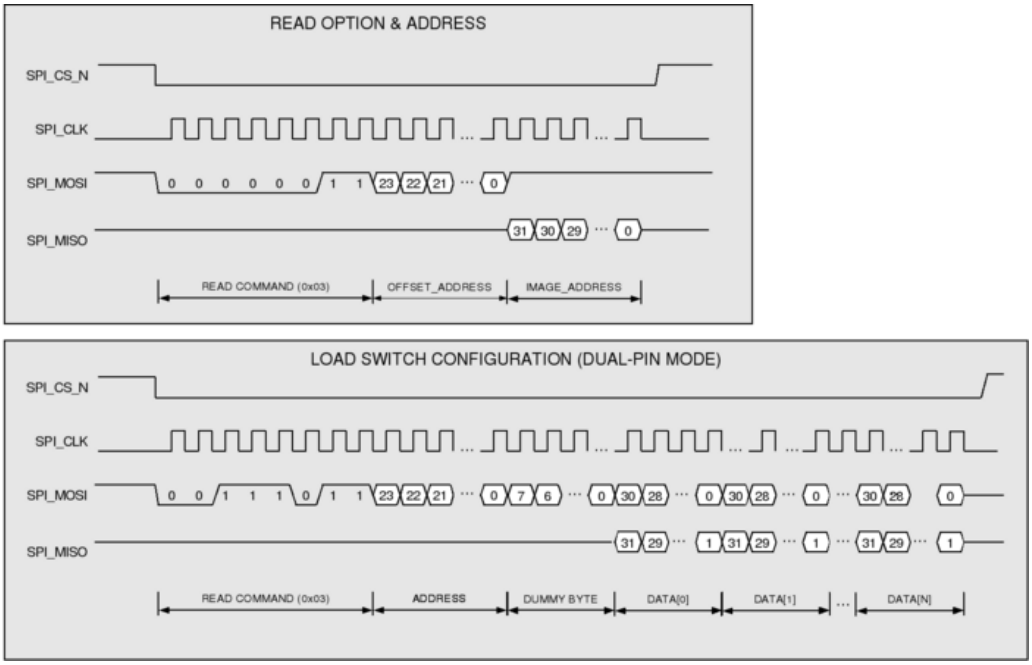


Figure 9-9 SPI Boot Access (Dual-pin Mode)

The exact process is:

- Switch is reset, BOOT_MODE is set to SPI for image 0,1,2 or 3.
- Switch configure GPIO to select special SPI pins.
- Switch asserts SPI_CS_N and send a read command (0x03) at address 0, 4, 8, 12 depending of the image selected.
  - Data is driven on the negative edge the clock.
  - Most significant bit is sent first.
  - Rate is 1/1024th of reference clock (122 KHz).
- Switch reads 32-bit word to recover base address of image selected.
  - First 8 bits defines speed (bit 5-3) and mode (7:6).
  - Next 24 bits define offset to read instructions.
- Switch de-asserts and re-asserts SPI_CS_N and sends a read command at address previously recovered.
  - Command is 0x03 for single-pin read, 0x0B for fast-read, 0x3B for dual and 0x6B for quad.
- Switch read data words (32 bits at the time) until end of configuration.
  - Data is sampled on the positive edge of the clock.
- De-activate SPI_CS_N, SPI_CLK and return GPIO to former mode.

9.8.3 Management

The SPI controller also includes capability for the host (local or remote) to issue commands on the SPI bus.

The SPI_CTRL Register controls the interface:

- **ENABLE** — Indicates if the controller is enabled or not
- **FREQ** — Defines speed of operation
- **CMD[3:0]** — Command to execute
- **SHIFT_METHOD** — Single, dual or quad
- **DATA_SIZE** — 1, 2, 3, 4 bytes (4 bytes coded as 00b)
- **HEADER_SIZE** — 1, 2, 3, 4 bytes (4 bytes coded as 00b)

Register SPI_HEADER contains the header, register SPI_TX_DATA contains the data to shift out, and register SPI_RX_DATA contains the data shift in.

Note: Enabling the SPI controller overrides control of GPIO/SPI pins from the GPIO controller. The shift method is only applicable for data read.

The command is a 4-bit structure:

Bit 0 = Indicates if a header must be shifted out

Bit 1 = Indicates if an 8-bit idle must be shifted out (used for turn-around in read-fast, read-dual, read-quad modes)

Bit 2 = Indicates if data must be shifted in/out

Bit 3 = Indicates if the SPI_CS_N must be deasserted or left asserted (set to 0b to leave asserted).

Figure 9-10 shows a management request with all four steps enabled, each step can potentially be disabled.

The command must be cleared (all 4 bits set to 0b) before a new command can be issued.

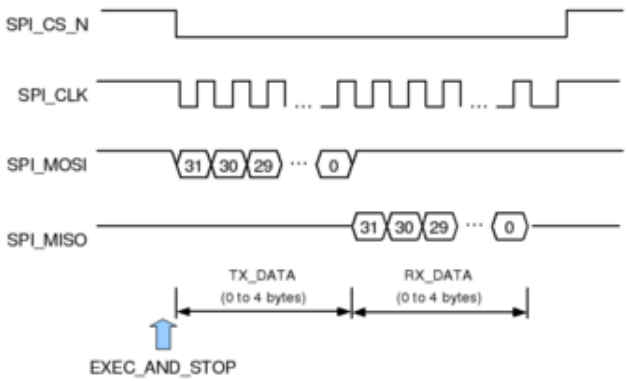


Figure 9-10 SPI Single Word Access

As an example, to read at a random address in an SPI-based EEPROM, the instruction to send is 0x03 followed by the address to read (total 32 bits) and followed by 32 bits of data read. Execution is as follows:

```
SPI_HEADER = 0x03000000 + <addr> # Set command (high byte) and address
SPI_TX_DATA = <don't care>       # Set command (high byte) and address
SPI_CTRL.Enable = 1              # Enable controller
SPI_CTRL.Freq = 0;               # Max rate
SPI_CTRL.ShiftMethod = 0;        # Single pin mode
SPI_CTRL.HeaderSize = 0;         # 4 bytes header
SPI_CTRL.DataSize = 0;           # 4 bytes data size
SPI_CTRL.Cmd = 4'b1101;          # Send Header, Get Data, Deselect when done
---wait 1.024us---
data = SPI_RX_DATA               # Data read
SPI_CTRL.Cmd = 4'b0000;          # Clear command
```

To read sequence of four words in quad-bit mode without de-selecting between words:

```
SPI_HEADER = 0x6B000000 + <addr>  # Set command and address
SPI_CTRL.Enable = 1               # Enable controller
SPI_CTRL.Freq = 0;                # Max rate
SPI_CTRL.ShiftMethod = 2;         # Quad pin mode
SPI_CTRL.HeaderSize = 0;          # 4 bytes header
SPI_CTRL.DataSize = 0;            # 4 bytes data size
SPI_CTRL.Cmd = 4'b0111;           # Send Header, Idle 1 byte, Get Data, Keep selected when done
---wait 0.75us---
data = SPI_RX_DATA                # Data word 0
SPI_CTRL.Cmd = 4'b0000;           # Clear command
SPI_CTRL.Cmd = 4'b0100;           # Get more data
---wait 0.12us---
data = SPI_RX_DATA                # Data word 1
SPI_CTRL.Cmd = 4'b0000;           # Clear command
SPI_CTRL.Cmd = 4'b0100;           # Get more data
---wait 0.12us---
data = SPI_RX_DATA                # Data word 2
SPI_CTRL.Cmd = 4'b0000;           # Clear command
SPI_CTRL.Cmd = 4'b1100;           # Get more data, deselect when done
---wait 0.12us---
data = SPI_RX_DATA                # Data word 3
SPI_CTRL.Cmd = 4'b0000;           # Clear command
```

Figure 9-11 shows the timing diagram for this sequence.

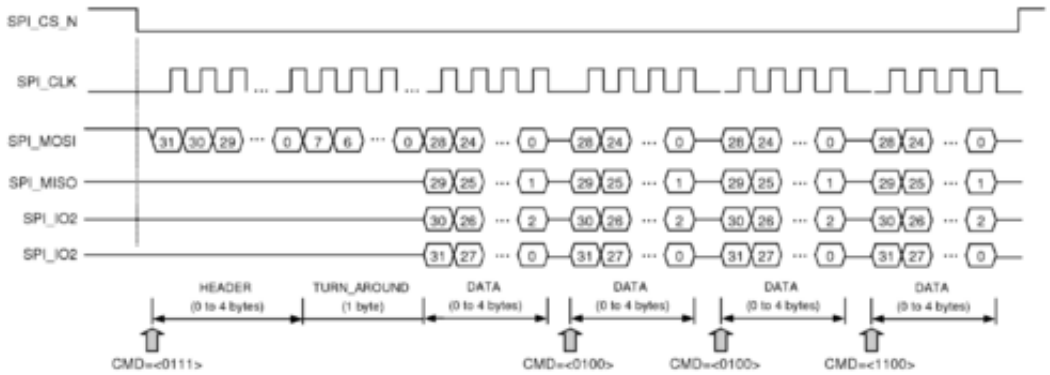


Figure 9-11 SPI Multiple Words Access

9.9 LED Controller

The LED interface consists of five signals: LED_CLK, LED_DATA0, LED_DATA1, LED_DATA2, and LED_EN.

Those five signals are used to transmit three bits of status data per MAC over the time multiplexed data pins.

The LED interface is controlled via the LED_CFG register which defines:

- **LED_FREQ** (24 bits) — A divider to derive the LED_CLK from the PCIE_REF_CLK (125 MHz). The exact frequency is equal to PCIE_REF_CLK/(LED_FREQ+1)/2. A value of zero is not supported.
- **LED_ENABLE** (1 bit) — Controls if the LED's are enabled (1b) or disabled (0b)

The overall timing diagram is shown in Figure 9-12. Each MAC is identified with a number X.Y where X is the EPL number (1 through 24) and Y is the MAC number within that EPL (0 through 3).

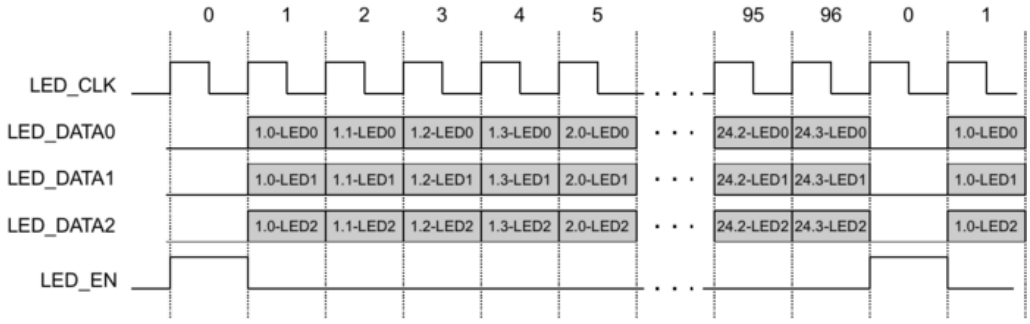


Figure 9-12 LED Timing Diagram

The LED_EN is asserted for one clock cycle to mark the beginning of a 97 clock cycle. Each successive clock cycle carries the three bits per MAC (24x4 = 96). Table 9-7 contains the 3-bit LED state encoding.

Table 9-7 LED Controller State Decoding

DATA0	DATA1	DATA2	State
0	0	0	Port is in reset.
0	0	1	Port is down, this state reflects the non-debounced link down state.
0	1	0	Port has detected remote fault.
0	1	1	Port has detected local fault.
1	0	0	Link is up, no packets transmitted or received since last sample.
1	1	1	Packet received since last sample.
1	x	1	Packet transmitted since last sample.

9.10 JTAG Interface

The JTAG controller is compliant to the IEEE 1149.1-2001 specification and provides the following basic external chip debug features:

- Access to an identification register.
- Access to the boundary scan.
- Access to the internal scan chains.
- Ability to Clamp and HighZ all outputs (except SerDes).

The maximum frequency of operation is 40 MHz.

The supported operations of these registers are:

- **Load IR** (instruction register)
- **Capture** — Initializes/captures/freezes value of register.
- **Shift**— Serially shifts in/out value into/out of register.
- **Update**— Validates the contents of the register. For example, logic can now use the new value for its internal operation.

The JTAG reset domain is separate and independent from the chip reset domain.

9.10.1 Tap Controller

The tap controller is a finite state machine of 16 states controlled by the 5-pin JTAG interface. It is defined by IEEE 1149.1-2001. Supported JTAG instructions are listed in Table 9-8.

Table 9-8 JTAG Instructions

Instruction	Code (6b)	Description
ICODE	0x01	Selects the identification register.
SAMPLE/PRELOAD	0x02	Selects the boundary scan register. Sample input pins to input boundary scan register and pre-loads the output boundary scan register.
EXTEST	0x03	Selects the boundary scan register. Output boundary scan register cells drive the covered output pins. Input boundary cell registers sample the input pins.
HIGHZ	0x06	Selects the bypass register and sets all covered output pins to high impedance.
CLAMP	0x07	Forces a known value on the outputs, but uses the bypass register to shorten scan length.
BYPASS	0x3F	

See the *Intel® Ethernet Switch FM6000 Series – Boundary Scan Description Language (BSDL)* file for boundary scan chain information.

NOTE: ***This page intentionally left blank.***

10.0 Electrical Specification

This section describes the electrical specifications for the FM5000/FM6000.

10.1 Absolute Maximum Ratings

Table 10-1 Absolute Maximum Ratings

Symbol	Parameter	Min	Max	Units
VDD	Variable core voltage	-0.3	1.3	V
VDDS	Fixed core voltage	-0.3	1.3	V
AVDD	SerDes analog voltage	-0.3	1.3	V
VDD25	LVCMOS power supply	-0.3	3.0	V
AVDD25	LVPECL (REFCLKs) power supply	-0.3	3.0	V
VDDPLL	PLL analog power supply	-0.3	1.3	V
-	Operating case temperature under bias	-	+115	C
-	Storage temperature	-65	+150	C
-	ESD	-2000	+2000	V

10.2 Recommended Operating Conditions

Table 10-2 Power Supply Voltages

Recap Parameter	Symbol	Min Voltage (V)	Max Voltage (V)	Comments
Core Voltage 1/Core Voltage 2	VDD/VDDS	0.91	1.14	
PLL Analog Power Supply	VDDPLL	1.05	1.15	
SerDes Analog Voltage	AVDD	0.95	1.05	No SerDes at 10 GbE.
SerDes Analog Voltage	AVDD	1.05	1.15	Any SerDes at 10 GbE.
LVCMOS and LVPECL Supplies	VDD25, AVDD25	2.25	2.75	

Notes:

1. VDD and VDDS voltages must be set to nominal values stored in the on-chip fuse box. These values should be read by the software and the power supplies adjusted accordingly.
2. The tables that follow also assume that 10 GbE SerDes are used, so the nominal AVDD supply is set to 1.10V.
3. These devices only operate over the commercial temperature range of 0 to 85 °C case temperature.

10.2.1 Voltage Scaling

Intel uses voltage scaling to satisfy device performance targets. When each part is tested, the required nominal VDD and VDDS supply voltages are programmed into an on-chip fuse box. The nominal voltages can range from 0.95V to 1.10V, with performance guaranteed over a ±40 mV span around each s specific nominal value. The system design must have the capability to adjust both VDD and VDDS based on the values programmed in the fuse box.

Note: The highest performance devices might also have high leakage current. For such parts, the maximum sustained power numbers are observed at lower VDD and VDDS values than listed in the tables that follow. The nominal VDD and VDDS values of those parts are therefore determined more by power constraints than performance constraints.

10.2.2 Maximum Peak Current

Maximum peak current limits are provided in the sections that follow as guidance for system power supply design. Under worst-case operating conditions and anomalous transient traffic loads, the current needs on the VDD and VDDS supply domains might spike to very high levels. These transient conditions generally do not arise for longer than a few milliseconds in production environments, if at all. However, to guarantee correct device operation, the system power supply should be provisioned to deliver these worst-case peak currents.

The device current requirements are highly sensitive to instantaneous traffic load, and in particular to the distribution of packet lengths the switch must process. Smaller packets require more energy to process than longer packets.

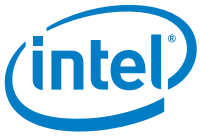
One consequence of this property is very high dI/dt characteristics. The device is capable of swinging from idle to peak current conditions over very small windows of time. Under pathological lab test conditions dI/dt can be 135 A/250 ns. For this reason, Intel strongly recommends the use of fast-response, multi-phase power supplies. Best industry design practices should be employed to achieve low impedance supply networks (below 1 mΩ) and adequate decoupling capacitance. Proper choice of capacitance and a power supply with <5 μs response time can support all conditions.

The typical current values listed represent typical current transients that might arise with a typical worst-case distribution of network traffic typical of many applications: a 1:15 ratio of 64:256-byte packets with, on average, 60 bytes of IFG (3x minimum). current values correspond to the theoretical worst-case traffic condition: continuous 64-byte packets with minimum IFG on all ports with no internal blocking in the switch.

Table 10-3 Maximum Peak Current[1] for FM6324, FM6724 and FM5224

Supply Domain	Symbol	Max Voltage (V)	Maximum Peak Current (A)	
			Typical	Margined
Core Voltage 1	VDD	VDD(nom)+40 mv	55.5	98
Core Voltage 2	VDDS	VDDS(nom)+40 mv	38.0	47.0
SerDes and PLL Supplies	AVDD, VDDPLL	1.15	9.3	11.0
LVCMOS and LVPECL Supplies	VDD25, AVDD25	2.75	0.6	0.6

1. Please contact Intel if intending to use a port configuration that requires more than 66 scheduler tokens.



Notes: Typical numbers assume 75 °C case temperature.
Margined numbers assume 85 °C case temperature.

Table 10-4 Maximum Peak Current[1] for FM6348

Supply Domain	Symbol	Max Voltage (V)	Maximum Peak Current (A)	
			Typical	Margined
Core Voltage 1	VDD	VDD(nom)+40mV	91.0	175.0
Core Voltage 2	VDDS	VDDS(nom)+40mv	60.5	77.5
SerDes and PLL Supplies	AVDD, VDDPLL	1.15	10.8	12.5
LVCMOS and LVPECL Supplies	VDD25, AVDD25	2.75	0.6	0.6

1. Please contact Intel if intending to use a port configuration that requires more than 66 scheduler tokens.

Notes: Typical numbers assume 75 °C case temperature.
Margined numbers assume 85 °C case temperature.

Table 10-5 Maximum Peak Current[1] for FM6364 and FM6764

Supply Domain	Symbol	Max Voltage (V)	Maximum Peak Current (A)	
			Typical	Margined
Core Voltage 1	VDD	VDD(nom)+40mV	116.0	232.0
Core Voltage 2	VDDS	VDDS(nom)+40mv	88.0	105.0
SerDes and PLL Supplies	AVDD, VDDPLL	1.15	11.7	13.5
LVCMOS and LVPECL Supplies	VDD25, AVDD25	2.75	0.6	0.6

1. Please contact Intel if intending to use a port configuration that requires more than 66 scheduler tokens.

Notes: Typical numbers assume 75 °C case temperature.
Margined numbers assume 85 °C case temperature.

10.2.3 Maximum Sustained Power

Intel characterizes maximum sustained power as a requirement for system TDP. Over short time spans, instantaneous power might see excursions to the maximum peak values (previously listed) in response to fluctuations in supply voltage, synchronized bursts of minimum-size packets, and other transitory conditions. However, over thermal time scales, with realistic worst-case traffic and power supply behavior, the steady-state maximum power dissipated by the device is bounded by the values specified in the sections that follow.

Typical and margined power values are provided in the section that follows. Both are obtained from measurements over silicon parametric variance using a realistic worst-case use model. The typical values assume the voltage levels are held at nominal values, allowing for transient excursions, while the margined values are characterized at max voltage with no ripple. The margined values are further buffered to allow for the full range of silicon variability that the manufacturing process might, theoretically, produce.

The traffic use model assumes all features are enabled in the device in their maximum power configurations. It assumes the following distribution of packets on all active ports: 5% 64-byte, 75% 256-byte, 20% idle. Packets are assumed to require maximally deep header parsing (IPv6 or similar).

Table 10-6 Maximum Sustained Power Dissipation for FM6324, FM6724 and FM5224

Supply Domain	Symbol	Voltage (V)		Maximum Sustained Power (W)	
		Min	Max	Typical	Margined
Core Voltage 1	VDD	0.95-1.10	+40mV	37.4	51.9
Core Voltage 2	VDDS	0.95-1.10	+40mV	29.0	41.6
SerDes and PLL Supplies	AVDD, VDDPLL	0.95	1.15	10.2	12.7
LVCMOS and LVPECL Supplies	VDD25, AVDD25	2.25	2.75	1.3	1.6
Total Power				73	102

Notes: Typical numbers assume 75 °C case temperature.
Margined numbers assume 85 °C case temperature.

Table 10-7 Maximum Sustained Power Dissipation for FM6348

Supply Domain	Symbol	Voltage (V)		Maximum Sustained Power (W)	
		Min	Max	Typical	Margined
Core Voltage 1	VDD	0.95-1.10	+40mV	64.6	78.4
Core Voltage 2	VDDS	0.95-1.10	+40mV	45.1	64.5
SerDes and PLL Supplies	AVDD, VDDPLL	1.10	1.15	11.8	14.4
LVCMOS and LVPECL Supplies	VDD25, AVDD25	2.50	2.75	1.3	1.5
Total Power				120	154

Notes: Typical numbers assume 75 °C case temperature.
Margined numbers assume 85 °C case temperature.

Table 10-8 Maximum Sustained Power Dissipation for FM6364 and FM6764

Supply Domain	Symbol	Voltage (V)		Maximum Sustained Power (W)	
		Min	Max	Typical	Margined
Core Voltage 1	VDD	0.95-1.10	+40mV	85.1	101.8
Core Voltage 2	VDDS	0.95-1.10	+40mV	57.8	79.7
SerDes and PLL Supplies	AVDD, VDDPLL	1.10	1.15	12.9	15.5
LVCMOS and LVPECL Supplies	VDD25, AVDD25	2.50	2.75	1.3	1.4
Total Power				156	195

Notes: Typical numbers assume 75 °C case temperature.
Margined numbers assume 85 °C case temperature.

10.3 Thermal Characteristics

The FM5000/FM6000 devices are designed to work over the commercial temperature range of 0-85 °C case temperature. Typical data center incoming air temperature is 35 °C. As can be seen in Figure 10-1, the thermal resistance from ambient to case can be maintained at 0.2C/W using a heat sink with airflow above 15 CFM. As shown in Table 10-8, the maximum sustained power for the FM6000 series is 195 W. This means that for 35 °C incoming air, the case temperature will be about 75 °C, providing some margin for temporary power increases.

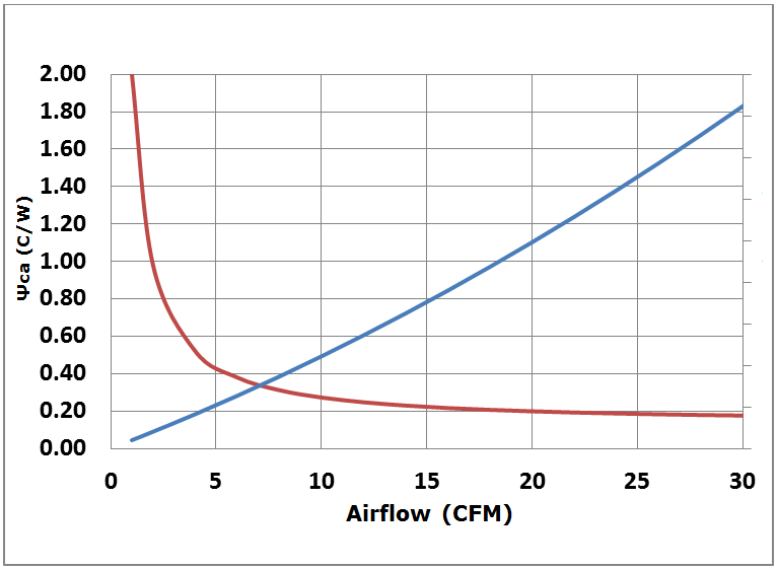
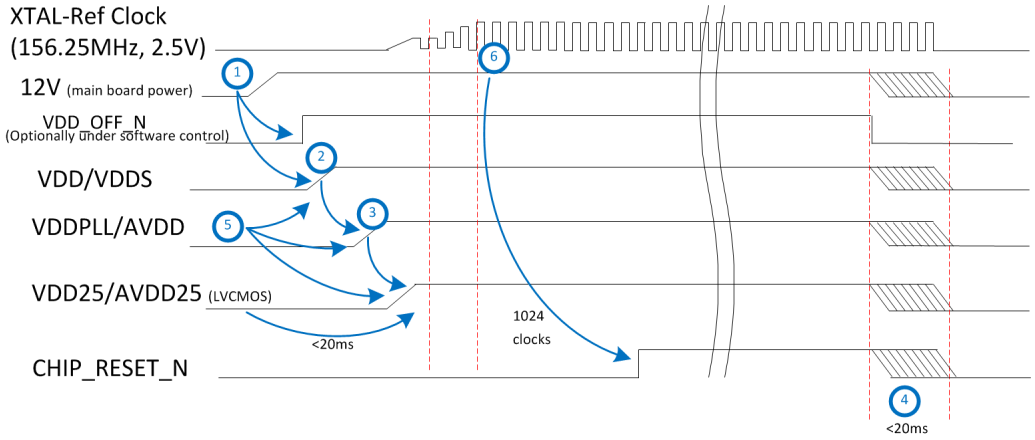


Figure 10-1 Thermal Characteristics

10.4 Power Supply Sequencing

Follow this order for power sequencing:

1. 12V
2. VDD/VDDS
3. VDDPLL/AVDD
4. VDD25/AVDD25



1. VDD/VDDS cannot start to ramp until the primary 12V rail is 90% of its final value. This is optionally under software control using the VDD_OFF_N signal driven by the FPGA.
2. AVDD & VDDPLL should not start to ramp until VDD/VDDS is 90% of its final value. This can also be gated by VDD_OFF_N.
3. Total Power-Up time, from when the VDD/VDDS rail starts rising until the VDD25 rail gets to its final level is < 20 ms
4. In case of Power-Off, it is recommend that all power-rails get to '0' level within 20 ms from the point the first power-rail starts powering off, although this risk is considered low.
5. Ramp time for each power-rail: 10 μs < Tramp < 1 ms.
6. Chip reset must be held asserted until power and clocks have been stable for 1024 clock cycles.

Figure 10-2 Power Supply Sequencing

10.5 REFCLK Specification

Table 10-9 156.25 MHz Clock

Symbol	Parameter	Min	Typ	Max	Units
VDD25	LVPECL voltage		2.5		V[1]
Vdiff	Differential voltage	0.4	0.7	1.0	V
Vcm	Common voltage	0.85	1.15	1.6	V
	Frequency		156.25		MHz
	Duty cycle	40	50	60	%
	Stability		30	50	ppm
	Skew between P & N			3	ps
Jrc	Input jitter RMS			0.24	ps

1. Input is 2.5V LVPECL. Internal termination is recommended.

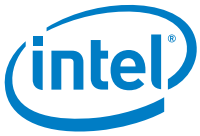


Table 10-10 125 MHz Clock

Symbol	Parameter	Min	Typ	Max	Units
VDD25	LVPECL voltage		2.5		V[1]
Vdiff	Differential voltage	0.4	0.7	1.0	V
Vcm	Common voltage	0.85	1.15	1.6	V
	Frequency		125		MHz[2]
	Duty cycle	40	50	60	%
Jrc	Input jitter RMS			0.24	ps

1. Input is 2.5V LVPECL. Internal termination is recommended.
2. Input clock can conform to the PCI Express Base Specification 3.0, Gen 2 requirements.

10.6 DC Characteristics of LVCMOS PADs

There are three types of LVCMOS PADs: 4 mA, 8 mA, and 12 mA. The characteristics are shown in Table 10-11.

The 4 mA LVCMOS are used for the following signals:

- LED_CLK,LED_EN,LED_DATA[2:0]
- I2C_SCL,I2C_SDA
- MDC,MDIO

The 8 mA LVCMOS are used for the following signals:

- DATA[31:0]
- PAR[3:0]
- DTACK_N,DERR_N,INTR_N
- GPIO[15:4], GPIO[2:0]
- PLL_CLKOUT
- TXRDY_N,RXRDY_N,RXEOT_N

The 12 mA LVCMOS are used for the following signals:

- GPIO[3]/SPI_SCK

Table 10-11 DC Characteristics of LVCMOS Pins

Parameter	Symbol	Test Conditions	Min	Typ	Max	Units
Output HIGH Current	IOH	VDD25=2.5V	-	4	-	mA
	IOH	VDD25=2.5V	-	8	-	mA
	IOH	VDD25=2.5V	-	12	-	mA
Output LOW Current	IOL	VDD25=2.5V	-	-4	-	mA
	IOL	VDD25=2.5V	-	-8	-	mA
	IOL	VDD25=2.5V	-	-12	-	mA
Output HIGH Voltage	VOH	VDD25=2.5V	VDD25 - 0.4	-	-	V

Table 10-11 DC Characteristics of LVCMOS Pins (Continued)

Parameter	Symbol	Test Conditions	Min	Typ	Max	Units
Output LOW Voltage	VOL	VDD25=2.5V	-	-	0.4	V
Input Current	IL	0V < V_{pad} < VDD25	-	-	± 5	µA
	IL	V_{pad} ≥ VDD25	-	-	± 500	µA
Input HIGH Level (Input and I/O pins)	VIH	Guaranteed Logic HIGH Level	1.7	-	VDD25 +1.2	V
Input LOW Level (Input and I/O pins)	VIL	Guaranteed Logic LOW Level	-0.3	-	0.7	V
Clamp Diode Voltage	VIK	VDD=Min, IIN=-18mA	-	-0.7	-1.2	V

10.7 Ethernet Output Specifications

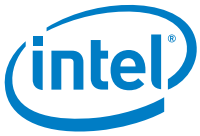
Table 10-12 Characteristics Ethernet Serdes Outputs

Symbol	Parameter	Min	Typ	Max	Units
VO-PP	Output voltage (peak-to-peak, differential)	400	-	1200	mV
VTCM	Transmit Common-mode Voltage	-	500	-	mV
JTT	Duty Cycle Distortion	-	-	50	mUI
	Random Jitter Component (RJ)	-	-	8.8	mUI
	Deterministic Jitter Component (DJ)	-	-	150	mUI
ZOD	Differential Output Impedance	80	100	120	ohms
TTR, TTF	Rise, Fall Times of Differential Outputs (controllable)	29	-	60	ps

10.8 Ethernet Input Specifications

Table 10-13 Characteristics Ethernet Serdes Inputs

Symbol	Parameter	Min	Typ	Max	Units
VI-PP	Input Voltage (peak-to-peak, differential)	100	-	-	mV
VICM	AC Input Common Mode Voltage	-	-	150	mV
JRT	Jitter Tolerance Mask at Baudrate/25000	-	-	1.5	UI
	Jitter Tolerance Mask at Baudrate/1667	-	-	0.1	UI
	Jitter Tolerance Mask at Baudrate/2	-	-	0.1	UI
ZIN	Differential Input Impedance	80	100	120	ohms
LDR	Differential Return Loss	8	-	-	dB
XDS	XAUI Differential Pair Skew	-	-	15	ps
KDS	10G Serial Differential Pair Skew	-	-	4.5	ps
XLS	XAUI Lane-to-Lane Skew	-	-	12.8	ns
KLS	40GBase-nn Lane-to-Lane Skew	-	-	TBD	ns



10.9 PCIe Output Specifications

Table 10-14 Characteristics PCIe SerDes Outputs

Symbol	Parameter	Min	Typ	Max	Units
VO-PP	Output voltage (peak-to-peak, differential)	800	-	1200	mV
VOCM	DC Output Common-mode Voltage	0	-	3.6	V
JTT	Duty Cycle Distortion	-	-	50	mUI
	Random Jitter Component (RJ)	-	-	8.8	mUI
	Deterministic Jitter Component (DJ)	-	-	150	mUI
ZOSE	Single Ended Output Impedance	-	50	-	ohms
ZOD	Differential Output Impedance	80	100	120	ohms
TTR, TTF	Rise, Fall Times of Differential Outputs (controllable)	29	-	60	ps

Note: FM5000/FM6000 PCIe meets the PCIe Specification rev 3.0

10.10 PCIe Input Specifications

Table 10-15 Characteristics PCIe SerDes Inputs

Symbol	Parameter	Min	Typ	Max	Units
VI-PP	Input Voltage (peak-to-peak, differential)	100	-	-	mV
VICM	AC Input Common Mode Voltage	-	-	150	mV
ZID	Differential Input Impedance	80	100	120	ohms
TRE	Received Eye Width	0.4	-	-	UI
DRL	Differential Return Loss	8	-	-	dB

Note: FM5000/FM6000 PCIe meets the PCIe Specification rev 3.0

10.11 EBI Interface, General Timing Requirements

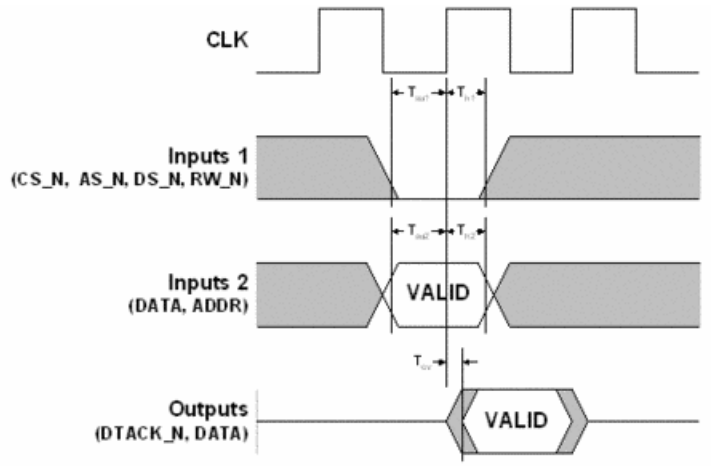
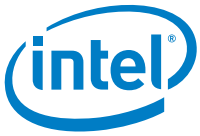


Figure 10-3 EBI Signal Timing

Table 10-16 EBI Interface Timing Constraints

Parameter	Symbol	Min	Typ	Max	Units	Test Conditions
Clock cycle	Tsu1	-	-	66	MHz	-
Input group 1 setup time to rising edge of clock	Tsu1	4.0	-	-	ns	-
Input group 1 hold time from rising edge of clock	Th1	0.5	-	-	ns	-
Input group 2 setup time to rising edge of clock	Tsu1	4.0	-	-	ns	-
Input group 2 hold time from rising edge of clock	Th1	0.5	-	-	ns	-
Rising edge of clock to output valid	Tov	0	-	6.5	ns	10 pf load
	Tov	0	-	7.5	ns	30 pf load

Notes: DTACK_INV, RW_N_INV are static signals. They must be stable before RESET_N is de-asserted. INTR is asynchronous signals.



10.12 JTAG Interface

The JTAG interface follows standard timing as defined in the IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture, 2001.

Note: When not using the JTAG interface, either drive the TCK pin with an external clock, or drive the TRST_N pin low. Conversely, when using the JTAG interface assert TRST_N along with chip reset to ensure proper reset of the JTAG interface prior to use.

NOTE: ***This page intentionally left blank.***

11.0 Mechanical Specification

11.1 1677-Ball Package Dimensions

This section contains the FM5000/FM6000 package dimensions. The 1677-ball FM5000/FM6000 package is lead-free. See Table 11-1 and Table 11-2 for complete pin lists.

Packaging, storage moisture and re-flow conditions can be referred to Intel Manufacturing Advance Service (MAS) report.

Note: The recommended contact force for a heat sink is 15 psi.

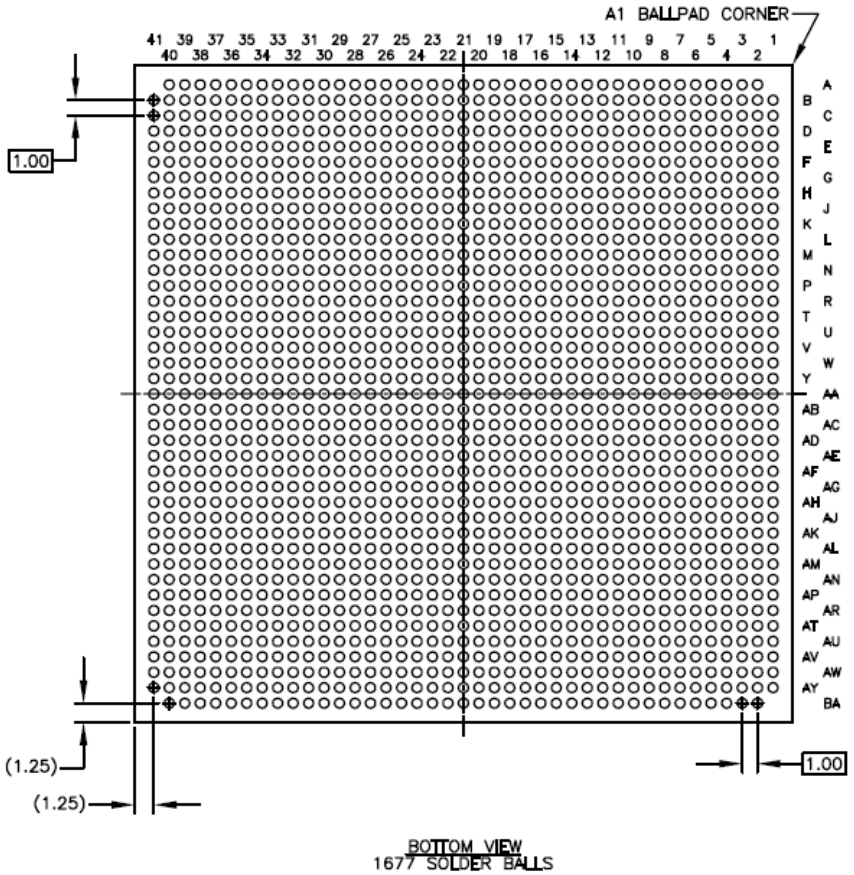
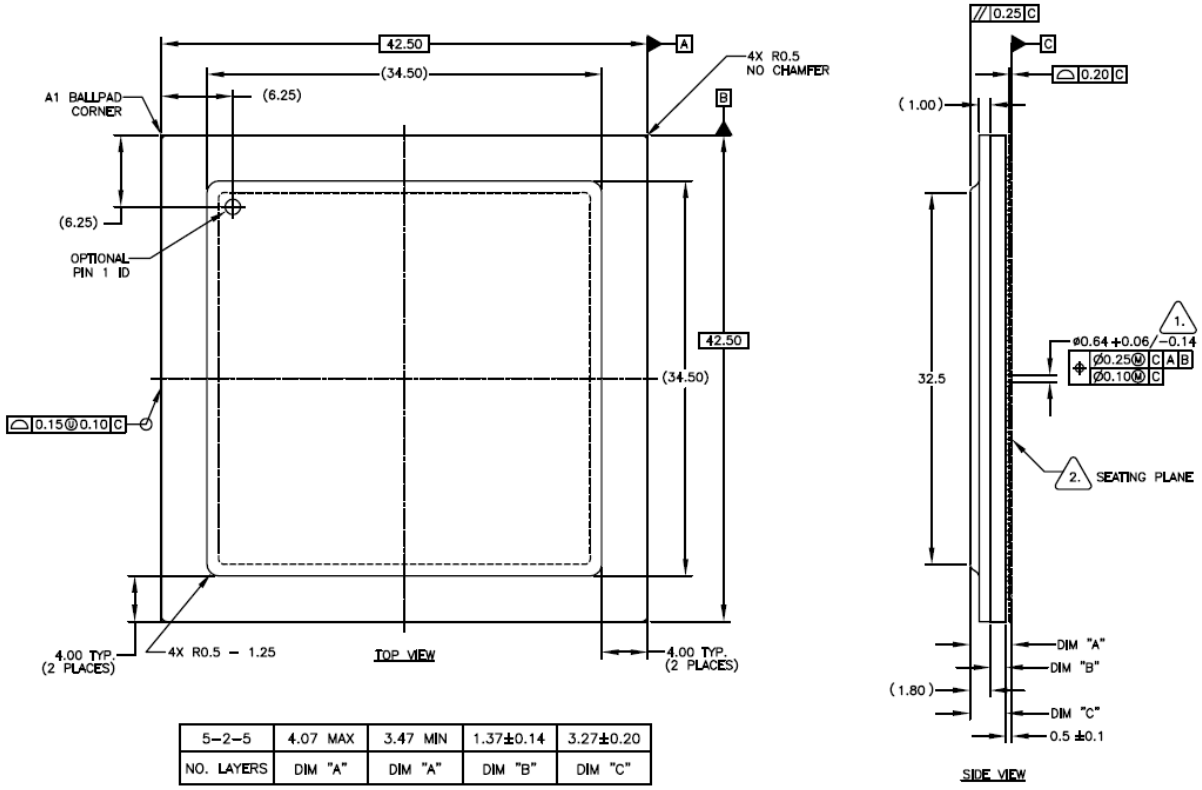


Figure 11-1 1677-Ball Package Bottom View



5–2–5	4.07 MAX	3.47 MIN	1.37±0.14	3.27±0.20
NO. LAYERS	DIM "A"	DIM "A"	DIM "B"	DIM "C"

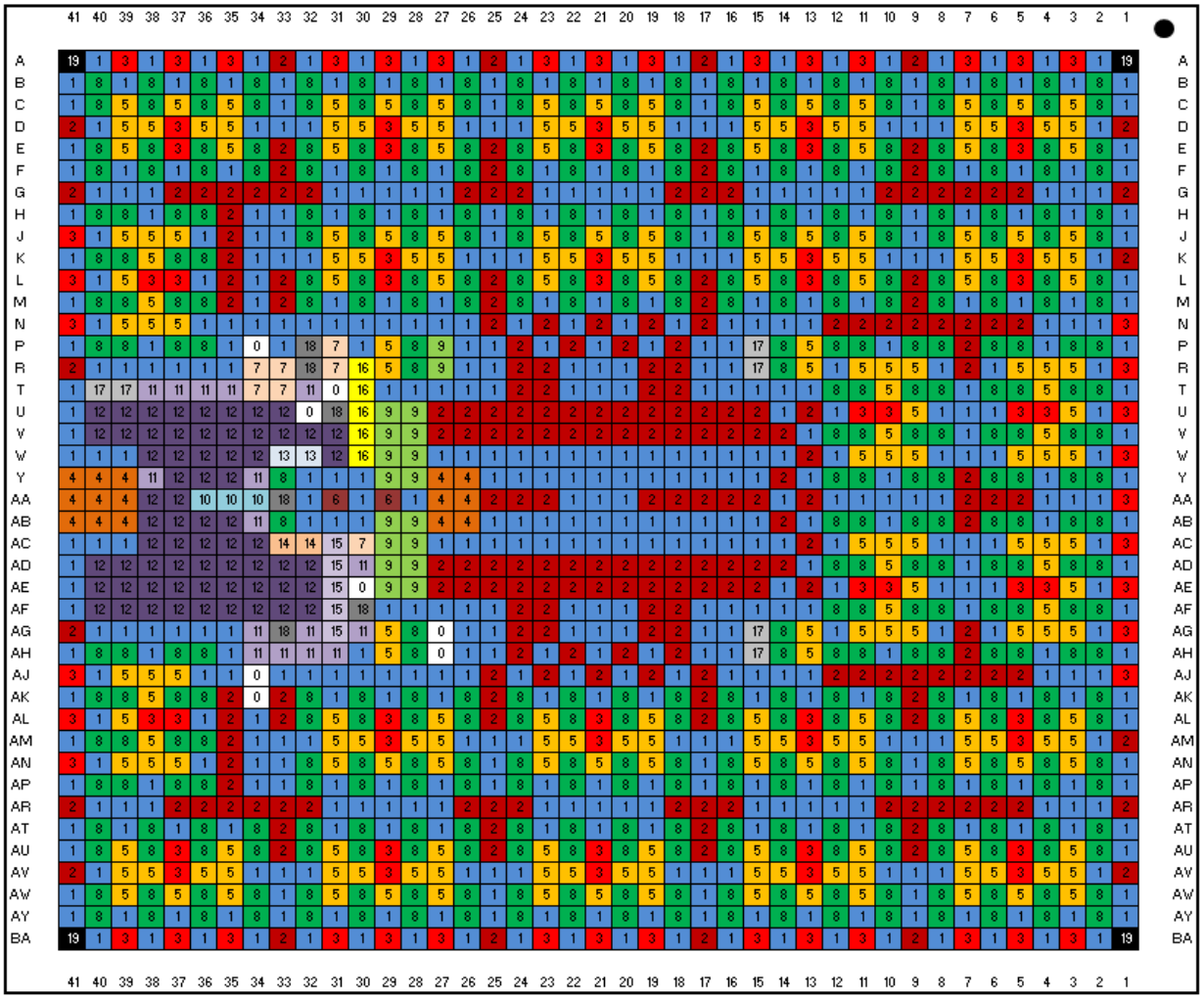
NOTES: UNLESS OTHERWISE SPECIFIED

1. DIMENSION IS MEASURED AT THE MAXIMUM SOLDER BALL DIAMETER, PARALLEL TO PRIMARY DATUM C.

2. PRIMARY DATUM C AND SEATING PLANE ARE DEFINED BY THE SPHERICAL CROWNS OF THE SOLDER BALLS.

Figure 11-2 1677-Ball Package Top/Side View

11.2 1677-Ball Package



Legend							
0	Misc. Signals	5	AVDD	10	DMA Interface	15	JTAG
1	VSS	6	AVDD25	11	GPIO and Strapping	16	LED
2	VDD	7	Misc. Power	12	External Bus Interface	17	Reserved
3	VDDS	8	Ethernet Port	13	MDIO	18	NC
4	VDD25	9	PCIe	14	I2C	19	No Ball

Figure 11-3 FM5000/FM6000 Ballout Diagram (Bottom View)

11.3 Pin List Ordered by Location

Table 11-1 Pin List Ordered by Location

Pin	Name	Pin	Name	Pin	Name	Pin	Name
A1	NoBall	A2	VSS	A3	VDDS	A4	VSS
A5	VDDS	A6	VSS	A7	VDDS	A8	VSS
A9	VDD	A10	VSS	A11	VDDS	A12	VSS
A13	VDDS	A14	VSS	A15	VDDS	A16	VSS
A17	VDD	A18	VSS	A19	VDDS	A20	VSS
A21	VDDS	A22	VSS	A23	VDDS	A24	VSS
A25	VDD	A26	VSS	A27	VDDS	A28	VSS
A29	VDDS	A30	VSS	A31	VDDS	A32	VSS
A33	VDD	A34	VSS	A35	VDDS	A36	VSS
A37	VDDS	A38	VSS	A39	VDDS	A40	VSS
A41	NoBall	B1	VSS	B2	P20_TDP	B3	VSS
B4	P20_TCP	B5	VSS	B6	P20_TBP	B7	VSS
B8	P20_TAP	B9	VSS	B10	P14_TDP	B11	VSS
B12	P14_TCP	B13	VSS	B14	P14_TBP	B15	VSS
B16	P14_TAP	B17	VSS	B18	P12_TDP	B19	VSS
B20	P12_TCP	B21	VSS	B22	P12_TBP	B23	VSS
B24	P12_TAP	B25	VSS	B26	P08_TDP	B27	VSS
B28	P08_TCP	B29	VSS	B30	P08_TBP	B31	VSS
B32	P08_TAP	B33	VSS	B34	P04_TDP	B35	VSS
B36	P04_TCP	B37	VSS	B38	P04_TBP	B39	VSS
B40	P04_TAP	B41	VSS	C1	VSS	C2	P20_TDN
C3	AVDD	C4	P20_TCN	C5	AVDD	C6	P20_TBN
C7	AVDD	C8	P20_TAN	C9	VSS	C10	P14_TDN
C11	AVDD	C12	P14_TCN	C13	AVDD	C14	P14_TBN
C15	AVDD	C16	P14_TAN	C17	VSS	C18	P12_TDN
C19	AVDD	C20	P12_TCN	C21	AVDD	C22	P12_TBN
C23	AVDD	C24	P12_TAN	C25	VSS	C26	P08_TDN
C27	AVDD	C28	P08_TCN	C29	AVDD	C30	P08_TBN
C31	AVDD	C32	P08_TAN	C33	VSS	C34	P04_TDN
C35	AVDD	C36	P04_TCN	C37	AVDD	C38	P04_TBN
C39	AVDD	C40	P04_TAN	C41	VSS	D1	VDD
D2	VSS	D3	AVDD	D4	AVDD	D5	VDDS
D6	AVDD	D7	AVDD	D8	VSS	D9	VSS
D10	VSS	D11	AVDD	D12	AVDD	D13	VDDS
D14	AVDD	D15	AVDD	D16	VSS	D17	VSS

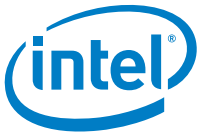


Table 11-1 Pin List Ordered by Location (Continued)

Pin	Name	Pin	Name	Pin	Name	Pin	Name
D18	VSS	D19	AVDD	D20	AVDD	D21	VDDS
D22	AVDD	D23	AVDD	D24	VSS	D25	VSS
D26	VSS	D27	AVDD	D28	AVDD	D29	VDDS
D30	AVDD	D31	AVDD	D32	VSS	D33	VSS
D34	VSS	D35	AVDD	D36	AVDD	D37	VDDS
D38	AVDD	D39	AVDD	D40	VSS	D41	VDD
E1	VSS	E2	P20_RDP	E3	AVDD	E4	P20_RCP
E5	VDDS	E6	P20_RBP	E7	AVDD	E8	P20_RAP
E9	VDD	E10	P14_RDP	E11	AVDD	E12	P14_RCP
E13	VDDS	E14	P14_RBP	E15	AVDD	E16	P14_RAP
E17	VDD	E18	P12_RDP	E19	AVDD	E20	P12_RCP
E21	VDDS	E22	P12_RBP	E23	AVDD	E24	P12_RAP
E25	VDD	E26	P08_RDP	E27	AVDD	E28	P08_RCP
E29	VDDS	E30	P08_RBP	E31	AVDD	E32	P08_RAP
E33	VDD	E34	P04_RDP	E35	AVDD	E36	P04_RCP
E37	VDDS	E38	P04_RBP	E39	AVDD	E40	P04_RAP
E41	VSS	F1	VSS	F2	P20_RDN	F3	VSS
F4	P20_RCN	F5	VSS	F6	P20_RBN	F7	VSS
F8	P20_RAN	F9	VDD	F10	P14_RDN	F11	VSS
F12	P14_RCN	F13	VSS	F14	P14_RBN	F15	VSS
F16	P14_RAN	F17	VDD	F18	P12_RDN	F19	VSS
F20	P12_RCN	F21	VSS	F22	P12_RBN	F23	VSS
F24	P12_RAN	F25	VDD	F26	P08_RDN	F27	VSS
F28	P08_RCN	F29	VSS	F30	P08_RBN	F31	VSS
F32	P08_RAN	F33	VDD	F34	P04_RDN	F35	VSS
F36	P04_RCN	F37	VSS	F38	P04_RBN	F39	VSS
F40	P04_RAN	F41	VSS	G1	VDD	G2	VSS
G3	VSS	G4	VSS	G5	VDD	G6	VDD
G7	VDD	G8	VDD	G9	VDD	G10	VDD
G11	VSS	G12	VSS	G13	VSS	G14	VSS
G15	VSS	G16	VDD	G17	VDD	G18	VDD
G19	VSS	G20	VSS	G21	VSS	G22	VSS
G23	VSS	G24	VDD	G25	VDD	G26	VDD
G27	VSS	G28	VSS	G29	VSS	G30	VSS
G31	VSS	G32	VDD	G33	VDD	G34	VDD
G35	VDD	G36	VDD	G37	VDD	G38	VSS
G39	VSS	G40	VSS	G41	VDD	H1	VSS
H2	P22_TDP	H3	VSS	H4	P22_TCP	H5	VSS

Table 11-1 Pin List Ordered by Location (Continued)

Pin	Name	Pin	Name	Pin	Name	Pin	Name
H6	P22_TBP	H7	VSS	H8	P22_TAP	H9	VSS
H10	P16_TDP	H11	VSS	H12	P16_TCP	H13	VSS
H14	P16_TBP	H15	VSS	H16	P16_TAP	H17	VSS
H18	P10_TDP	H19	VSS	H20	P10_TCP	H21	VSS
H22	P10_TBP	H23	VSS	H24	P10_TAP	H25	VSS
H26	P06_TDP	H27	VSS	H28	P06_TCP	H29	VSS
H30	P06_TBP	H31	VSS	H32	P06_TAP	H33	VSS
H34	VSS	H35	VDD	H36	P02_RDN	H37	P02_RDP
H38	VSS	H39	P02_TDN	H40	P02_TDP	H41	VSS
J1	VSS	J2	P22_TDN	J3	AVDD	J4	P22_TCN
J5	AVDD	J6	P22_TBN	J7	AVDD	J8	P22_TAN
J9	VSS	J10	P16_TDN	J11	AVDD	J12	P16_TCN
J13	AVDD	J14	P16_TBN	J15	AVDD	J16	P16_TAN
J17	VSS	J18	P10_TDN	J19	AVDD	J20	P10_TCN
J21	AVDD	J22	P10_TBN	J23	AVDD	J24	P10_TAN
J25	VSS	J26	P06_TDN	J27	AVDD	J28	P06_TCN
J29	AVDD	J30	P06_TBN	J31	AVDD	J32	P06_TAN
J33	VSS	J34	VSS	J35	VDD	J36	VSS
J37	AVDD	J38	AVDD	J39	AVDD	J40	VSS
J41	VDDS	K1	VDD	K2	VSS	K3	AVDD
K4	AVDD	K5	VDDS	K6	AVDD	K7	AVDD
K8	VSS	K9	VSS	K10	VSS	K11	AVDD
K12	AVDD	K13	VDDS	K14	AVDD	K15	AVDD
K16	VSS	K17	VSS	K18	VSS	K19	AVDD
K20	AVDD	K21	VDDS	K22	AVDD	K23	AVDD
K24	VSS	K25	VSS	K26	VSS	K27	AVDD
K28	AVDD	K29	VDDS	K30	AVDD	K31	AVDD
K32	VSS	K33	VSS	K34	VSS	K35	VDD
K36	P02_RCN	K37	P02_RCP	K38	AVDD	K39	P02_TCN
K40	P02_TCP	K41	VSS	L1	VSS	L2	P22_RDP
L3	AVDD	L4	P22_RCP	L5	VDDS	L6	P22_RBP
L7	AVDD	L8	P22_RAP	L9	VDD	L10	P16_RDP
L11	AVDD	L12	P16_RCP	L13	VDDS	L14	P16_RBP
L15	AVDD	L16	P16_RAP	L17	VDD	L18	P10_RDP
L19	AVDD	L20	P10_RCP	L21	VDDS	L22	P10_RBP
L23	AVDD	L24	P10_RAP	L25	VDD	L26	P06_RDP
L27	AVDD	L28	P06_RCP	L29	VDDS	L30	P06_RBP
L31	AVDD	L32	P06_RAP	L33	VDD	L34	VSS

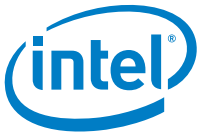


Table 11-1 Pin List Ordered by Location (Continued)

Pin	Name	Pin	Name	Pin	Name	Pin	Name
L35	VDD	L36	VSS	L37	VDDS	L38	VDDS
L39	AVDD	L40	VSS	L41	VDDS	M1	VSS
M2	P22_RDN	M3	VSS	M4	P22_RCN	M5	VSS
M6	P22_RBN	M7	VSS	M8	P22_RAN	M9	VDD
M10	P16_RDN	M11	VSS	M12	P16_RCN	M13	VSS
M14	P16_RBN	M15	VSS	M16	P16_RAN	M17	VDD
M18	P10_RDN	M19	VSS	M20	P10_RCN	M21	VSS
M22	P10_RBN	M23	VSS	M24	P10_RAN	M25	VDD
M26	P06_RDN	M27	VSS	M28	P06_RCN	M29	VSS
M30	P06_RBN	M31	VSS	M32	P06_RAN	M33	VDD
M34	VSS	M35	VDD	M36	P02_RBN	M37	P02_RBP
M38	AVDD	M39	P02_TBN	M40	P02_TBP	M41	VSS
N1	VDDS	N2	VSS	N3	VSS	N4	VSS
N5	VDD	N6	VDD	N7	VDD	N8	VDD
N9	VDD	N10	VDD	N11	VDD	N12	VDD
N13	VSS	N14	VSS	N15	VSS	N16	VSS
N17	VDD	N18	VSS	N19	VDD	N20	VSS
N21	VDD	N22	VSS	N23	VDD	N24	VSS
N25	VDD	N26	VSS	N27	VSS	N28	VSS
N29	VSS	N30	VSS	N31	VSS	N32	VSS
N33	VSS	N34	VSS	N35	VSS	N36	VSS
N37	AVDD	N38	AVDD	N39	AVDD	N40	VSS
N41	VDDS	P1	VSS	P2	P24_TAP	P3	P24_TAN
P4	VSS	P5	P24_RAP	P6	P24_RAN	P7	VDD
P8	P18_TAP	P9	P18_TAN	P10	VSS	P11	P18_RAP
P12	P18_RAN	P13	AVDD	P14	ETH_RCK4P	P15	Reserved
P16	VSS	P17	VSS	P18	VDD	P19	VSS
P20	VDD	P21	VSS	P22	VDD	P23	VSS
P24	VDD	P25	VSS	P26	VSS	P27	PX_RCK_P
P28	ETH_RCK2P	P29	AVDD	P30	VSS	P31	VFB1
P32	NC	P33	VSS	P34	CHIP_RESET_N	P35	VSS
P36	P02_RAN	P37	P02_RAP	P38	VSS	P39	P02_TAN
P40	P02_TAP	P41	VSS	R1	VDDS	R2	VSS
R3	AVDD	R4	AVDD	R5	AVDD	R6	VSS
R7	VDD	R8	VSS	R9	AVDD	R10	AVDD
R11	AVDD	R12	VSS	R13	AVDD	R14	ETH_RCK4N
R15	Reserved	R16	VSS	R17	VSS	R18	VDD
R19	VDD	R20	VSS	R21	VSS	R22	VSS

Table 11-1 Pin List Ordered by Location (Continued)

Pin	Name	Pin	Name	Pin	Name	Pin	Name
R23	VDD	R24	VDD	R25	VSS	R26	VSS
R27	PX_RCK_N	R28	ETH_RCK2N	R29	AVDD	R30	LED_CLK
R31	VFB2	R32	NC	R33	VDDK	R34	VSSK
R35	VSS	R36	VSS	R37	VSS	R38	VSS
R39	VSS	R40	VSS	R41	VDD	T1	VSS
T2	P24_TBP	T3	P24_TBN	T4	AVDD	T5	P24_RBP
T6	P24_RBN	T7	VSS	T8	P18_TBP	T9	P18_TBN
T10	AVDD	T11	P18_RBP	T12	P18_RBN	T13	VSS
T14	VSS	T15	VSS	T16	VSS	T17	VSS
T18	VDD	T19	VDD	T20	VSS	T21	VSS
T22	VSS	T23	VDD	T24	VDD	T25	VSS
T26	VSS	T27	VSS	T28	VSS	T29	VSS
T30	LED_DATA0	T31	PAD_TRI_N	T32	GPIO[7] BOOT_MODE[0]	T33	VDDSK
T34	VSSK2	T35	GPIO[5] SPI_MOSI	T36	GPIO[4] SPI_CS_N	T37	GPIO[3] SPI_SCK
T38	GPIO[6] SPI_MISO	T39	Reserved	T40	Reserved	T41	VSS
U1	VDDS	U2	VSS	U3	AVDD	U4	VDDS
U5	VDDS	U6	VSS	U7	VSS	U8	VSS
U9	AVDD	U10	VDDS	U11	VDDS	U12	VSS
U13	VDD	U14	VSS	U15	VDD	U16	VDD
U17	VDD	U18	VDD	U19	VDD	U20	VDD
U21	VDD	U22	VDD	U23	VDD	U24	VDD
U25	VDD	U26	VDD	U27	VDD	U28	PX_TAP
U29	PX_TAN	U30	LED_DATA1	U31	NC	U32	TESTMODE
U33	ADDR[2]	U34	ADDR[3]	U35	ADDR[4]	U36	ADDR[5]
U37	ADDR[6]	U38	ADDR[7]	U39	ADDR[8]	U40	ADDR[9]
U41	VSS	V1	VSS	V2	P24_TCP	V3	P24_TCN
V4	AVDD	V5	P24_RCP	V6	P24_RCN	V7	VSS
V8	P18_TCP	V9	P18_TCN	V10	AVDD	V11	P18_RCP
V12	P18_RCN	V13	VSS	V14	VDD	V15	VDD
V16	VDD	V17	VDD	V18	VDD	V19	VDD
V20	VDD	V21	VDD	V22	VDD	V23	VDD
V24	VDD	V25	VDD	V26	VDD	V27	VDD
V28	PX_TBP	V29	PX_TBN	V30	LED_DATA2	V31	DTACK_N
V32	ADDR[10]	V33	ADDR[11]	V34	ADDR[12]	V35	ADDR[13]
V36	ADDR[14]	V37	ADDR[15]	V38	ADDR[16]	V39	ADDR[17]
V40	ADDR[18]	V41	VSS	W1	VDDS	W2	VSS

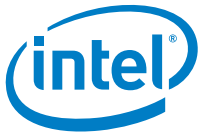


Table 11-1 Pin List Ordered by Location (Continued)

Pin	Name	Pin	Name	Pin	Name	Pin	Name
W3	AVDD	W4	AVDD	W5	AVDD	W6	VSS
W7	VSS	W8	VSS	W9	AVDD	W10	AVDD
W11	AVDD	W12	VSS	W13	VDD	W14	VSS
W15	VSS	W16	VSS	W17	VSS	W18	VSS
W19	VSS	W20	VSS	W21	VSS	W22	VSS
W23	VSS	W24	VSS	W25	VSS	W26	VSS
W27	VSS	W28	PX_TCP	W29	PX_TCN	W30	LED_EN
W31	DERR_N	W32	MDC	W33	MDIO	W34	ADDR[19]
W35	ADDR[20]	W36	ADDR[21]	W37	ADDR[22]	W38	ADDR[23]
W39	VSS	W40	VSS	W41	VSS	Y1	VSS
Y2	P24_TDP	Y3	P24_TDN	Y4	VSS	Y5	P24_RDP
Y6	P24_RDN	Y7	VDD	Y8	P18_TDP	Y9	P18_TDN
Y10	VSS	Y11	P18_RDP	Y12	P18_RDN	Y13	VSS
Y14	VDD	Y15	VSS	Y16	VSS	Y17	VSS
Y18	VSS	Y19	VSS	Y20	VSS	Y21	VSS
Y22	VSS	Y23	VSS	Y24	VSS	Y25	VSS
Y26	VDD25	Y27	VDD25	Y28	PX_TDP	Y29	PX_TDN
Y30	VSS	Y31	VSS	Y32	VSS	Y33	ETH_CLKOUT_A
Y34	GPIO[2] IGN_PAR	Y35	CS_N	Y36	AS_N	Y37	INTR_N
Y38	GPIO[9] BOOT_MODE[2]	Y39	VDD25	Y40	VDD25	Y41	VDD25
AA1	VDDS	AA2	VSS	AA3	VSS	AA4	VSS
AA5	VDD	AA6	VDD	AA7	VDD	AA8	VSS
AA9	VSS	AA10	VSS	AA11	VSS	AA12	VSS
AA13	VDD	AA14	VSS	AA15	VDD	AA16	VDD
AA17	VDD	AA18	VDD	AA19	VDD	AA20	VSS
AA21	VSS	AA22	VSS	AA23	VDD	AA24	VDD
AA25	VDD	AA26	VDD25	AA27	VDD25	AA28	VSS
AA29	AVDD25	AA30	VSS	AA31	AVDD25	AA32	VSS
AA33	NC	AA34	RXRDY_N	AA35	TXRDY_N	AA36	RXEOT_N
AA37	RW_N	AA38	CLK_EBI	AA39	VDD25	AA40	VDD25
AA41	VDD25	AB1	VSS	AB2	P23_TAP	AB3	P23_TAN
AB4	VSS	AB5	P23_RAP	AB6	P23_RAN	AB7	VDD
AB8	P17_TAP	AB9	P17_TAN	AB10	VSS	AB11	P17_RAP
AB12	P17_RAN	AB13	VSS	AB14	VDD	AB15	VSS
AB16	VSS	AB17	VSS	AB18	VSS	AB19	VSS
AB20	VSS	AB21	VSS	AB22	VSS	AB23	VSS
AB24	VSS	AB25	VSS	AB26	VDD25	AB27	VDD25

Table 11-1 Pin List Ordered by Location (Continued)

Pin	Name	Pin	Name	Pin	Name	Pin	Name
AB28	PX_RAP	AB29	PX_RAN	AB30	VSS	AB31	VSS
AB32	VSS	AB33	ETH_CLKOUT_B	AB34	GPIO[0] DTACK_INV	AB35	PAR[0]
AB36	PAR[1]	AB37	PAR[2]	AB38	PAR[3]	AB39	VDD25
AB40	VDD25	AB41	VDD25	AC1	VDDS	AC2	VSS
AC3	AVDD	AC4	AVDD	AC5	AVDD	AC6	VSS
AC7	VSS	AC8	VSS	AC9	AVDD	AC10	AVDD
AC11	AVDD	AC12	VSS	AC13	VDD	AC14	VSS
AC15	VSS	AC16	VSS	AC17	VSS	AC18	VSS
AC19	VSS	AC20	VSS	AC21	VSS	AC22	VSS
AC23	VSS	AC24	VSS	AC25	VSS	AC26	VSS
AC27	VSS	AC28	PX_RBP	AC29	PX_RBN	AC30	VDDPLL
AC31	TDI	AC32	I2C_SDA	AC33	I2C_SCL	AC34	DATA[0]
AC35	DATA[1]	AC36	DATA[2]	AC37	DATA[3]	AC38	DATA[4]
AC39	VSS	AC40	VSS	AC41	VSS	AD1	VSS
AD2	P23_TBP	AD3	P23_TBN	AD4	AVDD	AD5	P23_RBP
AD6	P23_RBN	AD7	VSS	AD8	P17_TBP	AD9	P17_TBN
AD10	AVDD	AD11	P17_RBP	AD12	P17_RBN	AD13	VSS
AD14	VDD	AD15	VDD	AD16	VDD	AD17	VDD
AD18	VDD	AD19	VDD	AD20	VDD	AD21	VDD
AD22	VDD	AD23	VDD	AD24	VDD	AD25	VDD
AD26	VDD	AD27	VDD	AD28	PX_RCP	AD29	PX_RCN
AD30	GPIO[1] RW_INV	AD31	TCK	AD32	DATA[5]	AD33	DATA[6]
AD34	DATA[7]	AD35	DATA[8]	AD36	DATA[9]	AD37	DATA[10]
AD38	DATA[11]	AD39	DATA[12]	AD40	DATA[13]	AD41	VSS
AE1	VDDS	AE2	VSS	AE3	AVDD	AE4	VDDS
AE5	VDDS	AE6	VSS	AE7	VSS	AE8	VSS
AE9	AVDD	AE10	VDDS	AE11	VDDS	AE12	VSS
AE13	VDD	AE14	VSS	AE15	VDD	AE16	VDD
AE17	VDD	AE18	VDD	AE19	VDD	AE20	VDD
AE21	VDD	AE22	VDD	AE23	VDD	AE24	VDD
AE25	VDD	AE26	VDD	AE27	VDD	AE28	PX_RDP
AE29	PX_RDN	AE30	PLL_CLKOUT	AE31	TRST_N	AE32	DATA[14]
AE33	DATA[15]	AE34	DATA[16]	AE35	DATA[17]	AE36	DATA[18]
AE37	DATA[19]	AE38	DATA[20]	AE39	DATA[21]	AE40	DATA[22]
AE41	VSS	AF1	VSS	AF2	P23_TCP	AF3	P23_TCN
AF4	AVDD	AF5	P23_RCP	AF6	P23_RCN	AF7	VSS
AF8	P17_TCP	AF9	P17_TCN	AF10	AVDD	AF11	P17_RCP

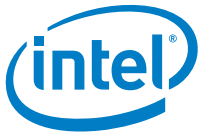


Table 11-1 Pin List Ordered by Location (Continued)

Pin	Name	Pin	Name	Pin	Name	Pin	Name
AF12	P17_RCN	AF13	VSS	AF14	VSS	AF15	VSS
AF16	VSS	AF17	VSS	AF18	VDD	AF19	VDD
AF20	VSS	AF21	VSS	AF22	VSS	AF23	VDD
AF24	VDD	AF25	VSS	AF26	VSS	AF27	VSS
AF28	VSS	AF29	VSS	AF30	NC	AF31	TMS
AF32	DATA[23]	AF33	DATA[24]	AF34	DATA[25]	AF35	DATA[26]
AF36	DATA[27]	AF37	DATA[28]	AF38	DATA[29]	AF39	DATA[30]
AF40	DATA[31]	AF41	VSS	AG1	VDDS	AG2	VSS
AG3	AVDD	AG4	AVDD	AG5	AVDD	AG6	VSS
AG7	VDD	AG8	VSS	AG9	AVDD	AG10	AVDD
AG11	AVDD	AG12	VSS	AG13	AVDD	AG14	ETH_RCK3P
AG15	Reserved	AG16	VSS	AG17	VSS	AG18	VDD
AG19	VDD	AG20	VSS	AG21	VSS	AG22	VSS
AG23	VDD	AG24	VDD	AG25	VSS	AG26	VSS
AG27	TEST_CLK_P	AG28	ETH_RCK1P	AG29	AVDD	AG30	GPIO[10] PARITY_EVEN
AG31	TDO	AG32	GPIO[8] BOOT_MODE[1]	AG33	NC	AG34	GPIO[11] I2C_ADDR[0]
AG35	VSS	AG36	VSS	AG37	VSS	AG38	VSS
AG39	VSS	AG40	VSS	AG41	VDD	AH1	VSS
AH2	P23_TDP	AH3	P23_TDN	AH4	VSS	AH5	P23_RDP
AH6	P23_RDN	AH7	VDD	AH8	P17_TDP	AH9	P17_TDN
AH10	VSS	AH11	P17_RDP	AH12	P17_RDN	AH13	AVDD
AH14	ETH_RCK3N	AH15	Reserved	AH16	VSS	AH17	VSS
AH18	VDD	AH19	VSS	AH20	VDD	AH21	VSS
AH22	VDD	AH23	VSS	AH24	VDD	AH25	VSS
AH26	VSS	AH27	TEST_CLK_N	AH28	ETH_RCK1N	AH29	AVDD
AH30	VSS	AH31	GPIO[12] I2C_ADDR[1]	AH32	GPIO[13] I2C_ADDR[2]	AH33	GPIO[14] DATA_HOLD SPI_IO3
AH34	GPIO[15] SPI_IO2	AH35	VSS	AH36	P01_RDN	AH37	P01_RDP
AH38	VSS	AH39	P01_TDN	AH40	P01_TDP	AH41	VSS
AJ1	VDDS	AJ2	VSS	AJ3	VSS	AJ4	VSS
AJ5	VDD	AJ6	VDD	AJ7	VDD	AJ8	VDD
AJ9	VDD	AJ10	VDD	AJ11	VDD	AJ12	VDD
AJ13	VSS	AJ14	VSS	AJ15	VSS	AJ16	VSS
AJ17	VDD	AJ18	VSS	AJ19	VDD	AJ20	VSS
AJ21	VDD	AJ22	VSS	AJ23	VDD	AJ24	VSS
AJ25	VDD	AJ26	VSS	AJ27	VSS	AJ28	VSS

Table 11-1 Pin List Ordered by Location (Continued)

Pin	Name	Pin	Name	Pin	Name	Pin	Name
AJ29	VSS	AJ30	VSS	AJ31	VSS	AJ32	VSS
AJ33	VSS	AJ34	DIODE_IN	AJ35	VSS	AJ36	VSS
AJ37	AVDD	AJ38	AVDD	AJ39	AVDD	AJ40	VSS
AJ41	VDDS	AK1	VSS	AK2	P21_RAN	AK3	VSS
AK4	P21_RBN	AK5	VSS	AK6	P21_RCN	AK7	VSS
AK8	P21_RDN	AK9	VDD	AK10	P15_RAN	AK11	VSS
AK12	P15_RBN	AK13	VSS	AK14	P15_RCN	AK15	VSS
AK16	P15_RDN	AK17	VDD	AK18	P09_RAN	AK19	VSS
AK20	P09_RBN	AK21	VSS	AK22	P09_RCN	AK23	VSS
AK24	P09_RDN	AK25	VDD	AK26	P05_RAN	AK27	VSS
AK28	P05_RBN	AK29	VSS	AK30	P05_RCN	AK31	VSS
AK32	P05_RDN	AK33	VDD	AK34	DIODE_OUT	AK35	VDD
AK36	P01_RCN	AK37	P01_RCP	AK38	AVDD	AK39	P01_TCN
AK40	P01_TCP	AK41	VSS	AL1	VSS	AL2	P21_RAP
AL3	AVDD	AL4	P21_RBP	AL5	VDDS	AL6	P21_RCP
AL7	AVDD	AL8	P21_RDP	AL9	VDD	AL10	P15_RAP
AL11	AVDD	AL12	P15_RBP	AL13	VDDS	AL14	P15_RCP
AL15	AVDD	AL16	P15_RDP	AL17	VDD	AL18	P09_RAP
AL19	AVDD	AL20	P09_RBP	AL21	VDDS	AL22	P09_RCP
AL23	AVDD	AL24	P09_RDP	AL25	VDD	AL26	P05_RAP
AL27	AVDD	AL28	P05_RBP	AL29	VDDS	AL30	P05_RCP
AL31	AVDD	AL32	P05_RDP	AL33	VDD	AL34	VSS
AL35	VDD	AL36	VSS	AL37	VDDS	AL38	VDDS
AL39	AVDD	AL40	VSS	AL41	VDDS	AM1	VDD
AM2	VSS	AM3	AVDD	AM4	AVDD	AM5	VDDS
AM6	AVDD	AM7	AVDD	AM8	VSS	AM9	VSS
AM10	VSS	AM11	AVDD	AM12	AVDD	AM13	VDDS
AM14	AVDD	AM15	AVDD	AM16	VSS	AM17	VSS
AM18	VSS	AM19	AVDD	AM20	AVDD	AM21	VDDS
AM22	AVDD	AM23	AVDD	AM24	VSS	AM25	VSS
AM26	VSS	AM27	AVDD	AM28	AVDD	AM29	VDDS
AM30	AVDD	AM31	AVDD	AM32	VSS	AM33	VSS
AM34	VSS	AM35	VDD	AM36	P01_RBN	AM37	P01_RBP
AM38	AVDD	AM39	P01_TBN	AM40	P01_TBP	AM41	VSS
AN1	VSS	AN2	P21_TAN	AN3	AVDD	AN4	P21_TBN
AN5	AVDD	AN6	P21_TCN	AN7	AVDD	AN8	P21_TDN
AN9	VSS	AN10	P15_TAN	AN11	AVDD	AN12	P15_TBN
AN13	AVDD	AN14	P15_TCN	AN15	AVDD	AN16	P15_TDN

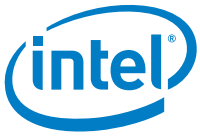


Table 11-1 Pin List Ordered by Location (Continued)

Pin	Name	Pin	Name	Pin	Name	Pin	Name
AN17	VSS	AN18	P09_TAN	AN19	AVDD	AN20	P09_TBN
AN21	AVDD	AN22	P09_TCN	AN23	AVDD	AN24	P09_TDN
AN25	VSS	AN26	P05_TAN	AN27	AVDD	AN28	P05_TBN
AN29	AVDD	AN30	P05_TCN	AN31	AVDD	AN32	P05_TDN
AN33	VSS	AN34	VSS	AN35	VDD	AN36	VSS
AN37	AVDD	AN38	AVDD	AN39	AVDD	AN40	VSS
AN41	VDDS	AP1	VSS	AP2	P21_TAP	AP3	VSS
AP4	P21_TBP	AP5	VSS	AP6	P21_TCP	AP7	VSS
AP8	P21_TDP	AP9	VSS	AP10	P15_TAP	AP11	VSS
AP12	P15_TBP	AP13	VSS	AP14	P15_TCP	AP15	VSS
AP16	P15_TDP	AP17	VSS	AP18	P09_TAP	AP19	VSS
AP20	P09_TBP	AP21	VSS	AP22	P09_TCP	AP23	VSS
AP24	P09_TDP	AP25	VSS	AP26	P05_TAP	AP27	VSS
AP28	P05_TBP	AP29	VSS	AP30	P05_TCP	AP31	VSS
AP32	P05_TDP	AP33	VSS	AP34	VSS	AP35	VDD
AP36	P01_RAN	AP37	P01_RAP	AP38	VSS	AP39	P01_TAN
AP40	P01_TAP	AP41	VSS	AR1	VDD	AR2	VSS
AR3	VSS	AR4	VSS	AR5	VDD	AR6	VDD
AR7	VDD	AR8	VDD	AR9	VDD	AR10	VDD
AR11	VSS	AR12	VSS	AR13	VSS	AR14	VSS
AR15	VSS	AR16	VDD	AR17	VDD	AR18	VDD
AR19	VSS	AR20	VSS	AR21	VSS	AR22	VSS
AR23	VSS	AR24	VDD	AR25	VDD	AR26	VDD
AR27	VSS	AR28	VSS	AR29	VSS	AR30	VSS
AR31	VSS	AR32	VDD	AR33	VDD	AR34	VDD
AR35	VDD	AR36	VDD	AR37	VDD	AR38	VSS
AR39	VSS	AR40	VSS	AR41	VDD	AT1	VSS
AT2	P19_RAN	AT3	VSS	AT4	P19_RBN	AT5	VSS
AT6	P19_RCN	AT7	VSS	AT8	P19_RDN	AT9	VDD
AT10	P13_RAN	AT11	VSS	AT12	P13_RBN	AT13	VSS
AT14	P13_RCN	AT15	VSS	AT16	P13_RDN	AT17	VDD
AT18	P11_RAN	AT19	VSS	AT20	P11_RBN	AT21	VSS
AT22	P11_RCN	AT23	VSS	AT24	P11_RDN	AT25	VDD
AT26	P07_RAN	AT27	VSS	AT28	P07_RBN	AT29	VSS
AT30	P07_RCN	AT31	VSS	AT32	P07_RDN	AT33	VDD
AT34	P03_RAN	AT35	VSS	AT36	P03_RBN	AT37	VSS
AT38	P03_RCN	AT39	VSS	AT40	P03_RDN	AT41	VSS
AU1	VSS	AU2	P19_RAP	AU3	AVDD	AU4	P19_RBP

Table 11-1 Pin List Ordered by Location (Continued)

Pin	Name	Pin	Name	Pin	Name	Pin	Name
AU5	VDDS	AU6	P19_RCP	AU7	AVDD	AU8	P19_RDP
AU9	VDD	AU10	P13_RAP	AU11	AVDD	AU12	P13_RBP
AU13	VDDS	AU14	P13_RCP	AU15	AVDD	AU16	P13_RDP
AU17	VDD	AU18	P11_RAP	AU19	AVDD	AU20	P11_RBP
AU21	VDDS	AU22	P11_RCP	AU23	AVDD	AU24	P11_RDP
AU25	VDD	AU26	P07_RAP	AU27	AVDD	AU28	P07_RBP
AU29	VDDS	AU30	P07_RCP	AU31	AVDD	AU32	P07_RDP
AU33	VDD	AU34	P03_RAP	AU35	AVDD	AU36	P03_RBP
AU37	VDDS	AU38	P03_RCP	AU39	AVDD	AU40	P03_RDP
AU41	VSS	AV1	VDD	AV2	VSS	AV3	AVDD
AV4	AVDD	AV5	VDDS	AV6	AVDD	AV7	AVDD
AV8	VSS	AV9	VSS	AV10	VSS	AV11	AVDD
AV12	AVDD	AV13	VDDS	AV14	AVDD	AV15	AVDD
AV16	VSS	AV17	VSS	AV18	VSS	AV19	AVDD
AV20	AVDD	AV21	VDDS	AV22	AVDD	AV23	AVDD
AV24	VSS	AV25	VSS	AV26	VSS	AV27	AVDD
AV28	AVDD	AV29	VDDS	AV30	AVDD	AV31	AVDD
AV32	VSS	AV33	VSS	AV34	VSS	AV35	AVDD
AV36	AVDD	AV37	VDDS	AV38	AVDD	AV39	AVDD
AV40	VSS	AV41	VDD	AW1	VSS	AW2	P19_TAN
AW3	AVDD	AW4	P19_TBN	AW5	AVDD	AW6	P19_TCN
AW7	AVDD	AW8	P19_TDN	AW9	VSS	AW10	P13_TAN
AW11	AVDD	AW12	P13_TBN	AW13	AVDD	AW14	P13_TCN
AW15	AVDD	AW16	P13_TDN	AW17	VSS	AW18	P11_TAN
AW19	AVDD	AW20	P11_TBN	AW21	AVDD	AW22	P11_TCN
AW23	AVDD	AW24	P11_TDN	AW25	VSS	AW26	P07_TAN
AW27	AVDD	AW28	P07_TBN	AW29	AVDD	AW30	P07_TCN
AW31	AVDD	AW32	P07_TDN	AW33	VSS	AW34	P03_TAN
AW35	AVDD	AW36	P03_TBN	AW37	AVDD	AW38	P03_TCN
AW39	AVDD	AW40	P03_TDN	AW41	VSS	AY1	VSS
AY2	P19_TAP	AY3	VSS	AY4	P19_TBP	AY5	VSS
AY6	P19_TCP	AY7	VSS	AY8	P19_TDP	AY9	VSS
AY10	P13_TAP	AY11	VSS	AY12	P13_TBP	AY13	VSS
AY14	P13_TCP	AY15	VSS	AY16	P13_TDP	AY17	VSS
AY18	P11_TAP	AY19	VSS	AY20	P11_TBP	AY21	VSS
AY22	P11_TCP	AY23	VSS	AY24	P11_TDP	AY25	VSS
AY26	P07_TAP	AY27	VSS	AY28	P07_TBP	AY29	VSS
AY30	P07_TCP	AY31	VSS	AY32	P07_TDP	AY33	VSS

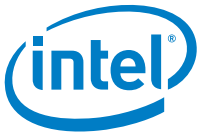


Table 11-1 Pin List Ordered by Location (Continued)

Pin	Name	Pin	Name	Pin	Name	Pin	Name
AY34	P03_TAP	AY35	VSS	AY36	P03_TBP	AY37	VSS
AY38	P03_TCP	AY39	VSS	AY40	P03_TDP	AY41	VSS
BA1	NoBall	BA2	VSS	BA3	VDDS	BA4	VSS
BA5	VDDS	BA6	VSS	BA7	VDDS	BA8	VSS
BA9	VDD	BA10	VSS	BA11	VDDS	BA12	VSS
BA13	VDDS	BA14	VSS	BA15	VDDS	BA16	VSS
BA17	VDD	BA18	VSS	BA19	VDDS	BA20	VSS
BA21	VDDS	BA22	VSS	BA23	VDDS	BA24	VSS
BA25	VDD	BA26	VSS	BA27	VDDS	BA28	VSS
BA29	VDDS	BA30	VSS	BA31	VDDS	BA32	VSS
BA33	VDD	BA34	VSS	BA35	VDDS	BA36	VSS
BA37	VDDS	BA38	VSS	BA39	VDDS	BA40	VSS
BA41	NoBall						

11.4 Pin List Ordered by Name

Table 11-2 Pin List Ordered by Name

Name	Pin	Name	Pin	Name	Pin	Name	Pin
ADDR[2]	U33	ADDR[3]	U34	ADDR[4]	U35	ADDR[5]	U36
ADDR[6]	U37	ADDR[7]	U38	ADDR[8]	U39	ADDR[9]	U40
ADDR[10]	V32	ADDR[11]	V33	ADDR[12]	V34	ADDR[13]	V35
ADDR[14]	V36	ADDR[15]	V37	ADDR[16]	V38	ADDR[17]	V39
ADDR[18]	V40	ADDR[19]	W34	ADDR[20]	W35	ADDR[21]	W36
ADDR[22]	W37	ADDR[23]	W38	AS_N	Y36	AVDD	C3
AVDD	C5	AVDD	C7	AVDD	C11	AVDD	C13
AVDD	C15	AVDD	C19	AVDD	C21	AVDD	C23
AVDD	C27	AVDD	C29	AVDD	C31	AVDD	C35
AVDD	C37	AVDD	C39	AVDD	D3	AVDD	D4
AVDD	D6	AVDD	D7	AVDD	D11	AVDD	D12
AVDD	D14	AVDD	D15	AVDD	D19	AVDD	D20
AVDD	D22	AVDD	D23	AVDD	D27	AVDD	D28
AVDD	D30	AVDD	D31	AVDD	D35	AVDD	D36
AVDD	D38	AVDD	D39	AVDD	E3	AVDD	E7
AVDD	E11	AVDD	E15	AVDD	E19	AVDD	E23
AVDD	E27	AVDD	E31	AVDD	E35	AVDD	E39
AVDD	J3	AVDD	J5	AVDD	J7	AVDD	J11

Table 11-2 Pin List Ordered by Name (Continued)

Name	Pin	Name	Pin	Name	Pin	Name	Pin
AVDD	J13	AVDD	J15	AVDD	J19	AVDD	J21
AVDD	J23	AVDD	J27	AVDD	J29	AVDD	J31
AVDD	J37	AVDD	J38	AVDD	J39	AVDD	K3
AVDD	K4	AVDD	K6	AVDD	K7	AVDD	K11
AVDD	K12	AVDD	K14	AVDD	K15	AVDD	K19
AVDD	K20	AVDD	K22	AVDD	K23	AVDD	K27
AVDD	K28	AVDD	K30	AVDD	K31	AVDD	K38
AVDD	L3	AVDD	L7	AVDD	L11	AVDD	L15
AVDD	L19	AVDD	L23	AVDD	L27	AVDD	L31
AVDD	L39	AVDD	M38	AVDD	N37	AVDD	N38
AVDD	N39	AVDD	P13	AVDD	P29	AVDD	R3
AVDD	R4	AVDD	R5	AVDD	R9	AVDD	R10
AVDD	R11	AVDD	R13	AVDD	R29	AVDD	T4
AVDD	T10	AVDD	U3	AVDD	U9	AVDD	V4
AVDD	V10	AVDD	W3	AVDD	W4	AVDD	W5
AVDD	W9	AVDD	W10	AVDD	W11	AVDD	AC3
AVDD	AC4	AVDD	AC5	AVDD	AC9	AVDD	AC10
AVDD	AC11	AVDD	AD4	AVDD	AD10	AVDD	AE3
AVDD	AE9	AVDD	AF4	AVDD	AF10	AVDD	AG3
AVDD	AG4	AVDD	AG5	AVDD	AG9	AVDD	AG10
AVDD	AG11	AVDD	AG13	AVDD	AG29	AVDD	AH13
AVDD	AH29	AVDD	AJ37	AVDD	AJ38	AVDD	AJ39
AVDD	AK38	AVDD	AL3	AVDD	AL7	AVDD	AL11
AVDD	AL15	AVDD	AL19	AVDD	AL23	AVDD	AL27
AVDD	AL31	AVDD	AL39	AVDD	AM3	AVDD	AM4
AVDD	AM6	AVDD	AM7	AVDD	AM11	AVDD	AM12
AVDD	AM14	AVDD	AM15	AVDD	AM19	AVDD	AM20
AVDD	AM22	AVDD	AM23	AVDD	AM27	AVDD	AM28
AVDD	AM30	AVDD	AM31	AVDD	AM38	AVDD	AN3
AVDD	AN5	AVDD	AN7	AVDD	AN11	AVDD	AN13
AVDD	AN15	AVDD	AN19	AVDD	AN21	AVDD	AN23
AVDD	AN27	AVDD	AN29	AVDD	AN31	AVDD	AN37
AVDD	AN38	AVDD	AN39	AVDD	AU3	AVDD	AU7
AVDD	AU11	AVDD	AU15	AVDD	AU19	AVDD	AU23
AVDD	AU27	AVDD	AU31	AVDD	AU35	AVDD	AU39
AVDD	AV3	AVDD	AV4	AVDD	AV6	AVDD	AV7
AVDD	AV11	AVDD	AV12	AVDD	AV14	AVDD	AV15
AVDD	AV19	AVDD	AV20	AVDD	AV22	AVDD	AV23

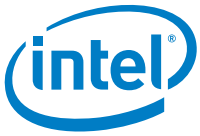


Table 11-2 Pin List Ordered by Name (Continued)

Name	Pin	Name	Pin	Name	Pin	Name	Pin
AVDD	AV27	AVDD	AV28	AVDD	AV30	AVDD	AV31
AVDD	AV35	AVDD	AV36	AVDD	AV38	AVDD	AV39
AVDD	AW3	AVDD	AW5	AVDD	AW7	AVDD	AW11
AVDD	AW13	AVDD	AW15	AVDD	AW19	AVDD	AW21
AVDD	AW23	AVDD	AW27	AVDD	AW29	AVDD	AW31
AVDD	AW35	AVDD	AW37	AVDD	AW39	AVDD25	AA29
AVDD25	AA31	CHIP_RESET_N	P34	CLK_EBI	AA38	CS_N	Y35
DATA[0]	AC34	DATA[1]	AC35	DATA[2]	AC36	DATA[3]	AC37
DATA[4]	AC38	DATA[5]	AD32	DATA[6]	AD33	DATA[7]	AD34
DATA[8]	AD35	DATA[9]	AD36	DATA[10]	AD37	DATA[11]	AD38
DATA[12]	AD39	DATA[13]	AD40	DATA[14]	AE32	DATA[15]	AE33
DATA[16]	AE34	DATA[17]	AE35	DATA[18]	AE36	DATA[19]	AE37
DATA[20]	AE38	DATA[21]	AE39	DATA[22]	AE40	DATA[23]	AF32
DATA[24]	AF33	DATA[25]	AF34	DATA[26]	AF35	DATA[27]	AF36
DATA[28]	AF37	DATA[29]	AF38	DATA[30]	AF39	DATA[31]	AF40
DERR_N	W31	DIODE2_IN	T39	DIODE2_OUT	T40	DIODE_IN	AJ34
DIODE_OUT	AK34	DTACK_N	V31	ETH_CLKOUT_A	Y33	ETH_CLKOUT_B	AB33
ETH_RCK1N	AH28	ETH_RCK1P	AG28	ETH_RCK2N	R28	ETH_RCK2P	P28
ETH_RCK3N	AH14	ETH_RCK3P	AG14	ETH_RCK4N	R14	ETH_RCK4P	P14
GPIO[0] DTACK_INV	AB34	GPIO[1] RW_INV	AD30	GPIO[2] IGN_PAR	Y34	GPIO[3] SPI_SCK	T37
GPIO[4] SPI_CS_N	T36	GPIO[5] SPI_MOSI	T35	GPIO[6] SPI_MISO	T38	GPIO[7] BOOT_MODE[0]	T32
GPIO[8] BOOT_MODE[1]	AG32	GPIO[9] BOOT_MODE[2]	Y38	GPIO[10] PARITY_EVEN	AG30	GPIO[11] I2C_ADDR[0]	AG34
GPIO[12] I2C_ADDR[1]	AH31	GPIO[13] I2C_ADDR[2]	AH32	GPIO[14] DATA_HOLD SPI_IO3	AH33	GPIO[15] SPI_IO2	AH34
I2C_SCL	AC33	I2C_SDA	AC32	INTR_N	Y37	LED_CLK	R30
LED_DATA0	T30	LED_DATA1	U30	LED_DATA2	V30	LED_EN	W30
MDC	W32	MDIO	W33	NC	P32	NC	R32
NC	U31	NC	AA33	NC	AF30	NC	AG33
NoBalL	A1	NoBall	A41	NoBall	BA1	NoBall	BA41
P01_RAN	AP36	P01_RAP	AP37	P01_RBN	AM36	P01_RBP	AM37
P01_RCN	AK36	P01_RCP	AK37	P01_RDN	AH36	P01_RDP	AH37
P01_TAN	AP39	P01_TAP	AP40	P01_TBN	AM39	P01_TBP	AM40
P01_TCN	AK39	P01_TCP	AK40	P01_TDN	AH39	P01_TDP	AH40
P02_RAN	P36	P02_RAP	P37	P02_RBN	M36	P02_RBP	M37
P02_RCN	K36	P02_RCP	K37	P02_RDN	H36	P02_RDP	H37
P02_TAN	P39	P02_TAP	P40	P02_TBN	M39	P02_TBP	M40

Table 11-2 Pin List Ordered by Name (Continued)

Name	Pin	Name	Pin	Name	Pin	Name	Pin
P02_TCN	K39	P02_TCP	K40	P02_TDN	H39	P02_TDP	H40
P03_RAN	AT34	P03_RAP	AU34	P03_RBN	AT36	P03_RBP	AU36
P03_RCN	AT38	P03_RCP	AU38	P03_RDN	AT40	P03_RDP	AU40
P03_TAN	AW34	P03_TAP	AY34	P03_TBN	AW36	P03_TBP	AY36
P03_TCN	AW38	P03_TCP	AY38	P03_TDN	AW40	P03_TDP	AY40
P04_RAN	F40	P04_RAP	E40	P04_RBN	F38	P04_RBP	E38
P04_RCN	F36	P04_RCP	E36	P04_RDN	F34	P04_RDP	E34
P04_TAN	C40	P04_TAP	B40	P04_TBN	C38	P04_TBP	B38
P04_TCN	C36	P04_TCP	B36	P04_TDN	C34	P04_TDP	B34
P05_RAN	AK26	P05_RAP	AL26	P05_RBN	AK28	P05_RBP	AL28
P05_RCN	AK30	P05_RCP	AL30	P05_RDN	AK32	P05_RDP	AL32
P05_TAN	AN26	P05_TAP	AP26	P05_TBN	AN28	P05_TBP	AP28
P05_TCN	AN30	P05_TCP	AP30	P05_TDN	AN32	P05_TDP	AP32
P06_RAN	M32	P06_RAP	L32	P06_RBN	M30	P06_RBP	L30
P06_RCN	M28	P06_RCP	L28	P06_RDN	M26	P06_RDP	L26
P06_TAN	J32	P06_TAP	H32	P06_TBN	J30	P06_TBP	H30
P06_TCN	J28	P06_TCP	H28	P06_TDN	J26	P06_TDP	H26
P07_RAN	AT26	P07_RAP	AU26	P07_RBN	AT28	P07_RBP	AU28
P07_RCN	AT30	P07_RCP	AU30	P07_RDN	AT32	P07_RDP	AU32
P07_TAN	AW26	P07_TAP	AY26	P07_TBN	AW28	P07_TBP	AY28
P07_TCN	AW30	P07_TCP	AY30	P07_TDN	AW32	P07_TDP	AY32
P08_RAN	F32	P08_RAP	E32	P08_RBN	F30	P08_RBP	E30
P08_RCN	F28	P08_RCP	E28	P08_RDN	F26	P08_RDP	E26
P08_TAN	C32	P08_TAP	B32	P08_TBN	C30	P08_TBP	B30
P08_TCN	C28	P08_TCP	B28	P08_TDN	C26	P08_TDP	B26
P09_RAN	AK18	P09_RAP	AL18	P09_RBN	AK20	P09_RBP	AL20
P09_RCN	AK22	P09_RCP	AL22	P09_RDN	AK24	P09_RDP	AL24
P09_TAN	AN18	P09_TAP	AP18	P09_TBN	AN20	P09_TBP	AP20
P09_TCN	AN22	P09_TCP	AP22	P09_TDN	AN24	P09_TDP	AP24
P10_RAN	M24	P10_RAP	L24	P10_RBN	M22	P10_RBP	L22
P10_RCN	M20	P10_RCP	L20	P10_RDN	M18	P10_RDP	L18
P10_TAN	J24	P10_TAP	H24	P10_TBN	J22	P10_TBP	H22
P10_TCN	J20	P10_TCP	H20	P10_TDN	J18	P10_TDP	H18
P11_RAN	AT18	P11_RAP	AU18	P11_RBN	AT20	P11_RBP	AU20
P11_RCN	AT22	P11_RCP	AU22	P11_RDN	AT24	P11_RDP	AU24
P11_TAN	AW18	P11_TAP	AY18	P11_TBN	AW20	P11_TBP	AY20
P11_TCN	AW22	P11_TCP	AY22	P11_TDN	AW24	P11_TDP	AY24
P12_RAN	F24	P12_RAP	E24	P12_RBN	F22	P12_RBP	E22

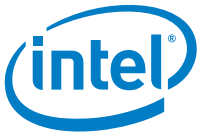


Table 11-2 Pin List Ordered by Name (Continued)

Name	Pin	Name	Pin	Name	Pin	Name	Pin
P12_RCN	F20	P12_RCP	E20	P12_RDN	F18	P12_RDP	E18
P12_TAN	C24	P12_TAP	B24	P12_TBN	C22	P12_TBP	B22
P12_TCN	C20	P12_TCP	B20	P12_TDN	C18	P12_TDP	B18
P13_RAN	AT10	P13_RAP	AU10	P13_RBN	AT12	P13_RBP	AU12
P13_RCN	AT14	P13_RCP	AU14	P13_RDN	AT16	P13_RDP	AU16
P13_TAN	AW10	P13_TAP	AY10	P13_TBN	AW12	P13_TBP	AY12
P13_TCN	AW14	P13_TCP	AY14	P13_TDN	AW16	P13_TDP	AY16
P14_RAN	F16	P14_RAP	E16	P14_RBN	F14	P14_RBP	E14
P14_RCN	F12	P14_RCP	E12	P14_RDN	F10	P14_RDP	E10
P14_TAN	C16	P14_TAP	B16	P14_TBN	C14	P14_TBP	B14
P14_TCN	C12	P14_TCP	B12	P14_TDN	C10	P14_TDP	B10
P15_RAN	AK10	P15_RAP	AL10	P15_RBN	AK12	P15_RBP	AL12
P15_RCN	AK14	P15_RCP	AL14	P15_RDN	AK16	P15_RDP	AL16
P15_TAN	AN10	P15_TAP	AP10	P15_TBN	AN12	P15_TBP	AP12
P15_TCN	AN14	P15_TCP	AP14	P15_TDN	AN16	P15_TDP	AP16
P16_RAN	M16	P16_RAP	L16	P16_RBN	M14	P16_RBP	L14
P16_RCN	M12	P16_RCP	L12	P16_RDN	M10	P16_RDP	L10
P16_TAN	J16	P16_TAP	H16	P16_TBN	J14	P16_TBP	H14
P16_TCN	J12	P16_TCP	H12	P16_TDN	J10	P16_TDP	H10
P17_RAN	AB12	P17_RAP	AB11	P17_RBN	AD12	P17_RBP	AD11
P17_RCN	AF12	P17_RCP	AF11	P17_RDN	AH12	P17_RDP	AH11
P17_TAN	AB9	P17_TAP	AB8	P17_TBN	AD9	P17_TBP	AD8
P17_TCN	AF9	P17_TCP	AF8	P17_TDN	AH9	P17_TDP	AH8
P18_RAN	P12	P18_RAP	P11	P18_RBN	T12	P18_RBP	T11
P18_RCN	V12	P18_RCP	V11	P18_RDN	Y12	P18_RDP	Y11
P18_TAN	P9	P18_TAP	P8	P18_TBN	T9	P18_TBP	T8
P18_TCN	V9	P18_TCP	V8	P18_TDN	Y9	P18_TDP	Y8
P19_RAN	AT2	P19_RAP	AU2	P19_RBN	AT4	P19_RBP	AU4
P19_RCN	AT6	P19_RCP	AU6	P19_RDN	AT8	P19_RDP	AU8
P19_TAN	AW2	P19_TAP	AY2	P19_TBN	AW4	P19_TBP	AY4
P19_TCN	AW6	P19_TCP	AY6	P19_TDN	AW8	P19_TDP	AY8
P20_RAN	F8	P20_RAP	E8	P20_RBN	F6	P20_RBP	E6
P20_RCN	F4	P20_RCP	E4	P20_RDN	F2	P20_RDP	E2
P20_TAN	C8	P20_TAP	B8	P20_TBN	C6	P20_TBP	B6
P20_TCN	C4	P20_TCP	B4	P20_TDN	C2	P20_TDP	B2
P21_RAN	AK2	P21_RAP	AL2	P21_RBN	AK4	P21_RBP	AL4
P21_RCN	AK6	P21_RCP	AL6	P21_RDN	AK8	P21_RDP	AL8
P21_TAN	AN2	P21_TAP	AP2	P21_TBN	AN4	P21_TBP	AP4

Table 11-2 Pin List Ordered by Name (Continued)

Name	Pin	Name	Pin	Name	Pin	Name	Pin
P21_TCN	AN6	P21_TCP	AP6	P21_TDN	AN8	P21_TDP	AP8
P22_RAN	M8	P22_RAP	L8	P22_RBN	M6	P22_RBP	L6
P22_RCN	M4	P22_RCP	L4	P22_RDN	M2	P22_RDP	L2
P22_TAN	J8	P22_TAP	H8	P22_TBN	J6	P22_TBP	H6
P22_TCN	J4	P22_TCP	H4	P22_TDN	J2	P22_TDP	H2
P23_RAN	AB6	P23_RAP	AB5	P23_RBN	AD6	P23_RBP	AD5
P23_RCN	AF6	P23_RCP	AF5	P23_RDN	AH6	P23_RDP	AH5
P23_TAN	AB3	P23_TAP	AB2	P23_TBN	AD3	P23_TBP	AD2
P23_TCN	AF3	P23_TCP	AF2	P23_TDN	AH3	P23_TDP	AH2
P24_RAN	P6	P24_RAP	P5	P24_RBN	T6	P24_RBP	T5
P24_RCN	V6	P24_RCP	V5	P24_RDN	Y6	P24_RDP	Y5
P24_TAN	P3	P24_TAP	P2	P24_TBN	T3	P24_TBP	T2
P24_TCN	V3	P24_TCP	V2	P24_TDN	Y3	P24_TDP	Y2
PAD_TRI_N	T31	PAR[0]	AB35	PAR[1]	AB36	PAR[2]	AB37
PAR[3]	AB38	PLL_CLKOUT	AE30	PX_RAN	AB29	PX_RAP	AB28
PX_RBN	AC29	PX_RBP	AC28	PX_RCK_N	R27	PX_RCK_P	P27
PX_RCN	AD29	PX_RCP	AD28	PX_RDN	AE29	PX_RDP	AE28
PX_TAN	U29	PX_TAP	U28	PX_TBN	V29	PX_TBP	V28
PX_TCN	W29	PX_TCP	W28	PX_TDN	Y29	PX_TDP	Y28
RW_N	AA37	RXEOT_N	AA36	RXRDY_N	AA34	Reserved	P15
Reserved	R15	Reserved	AG15	Reserved	AH15	TCK	AD31
TDI	AC31	TDO	AG31	TESTMODE	U32	TEST_CLK_N	AH27
TEST_CLK_P	AG27	TMS	AF31	TRST_N	AE31	TXRDY_N	AA35
VDD	A9	VDD	A17	VDD	A25	VDD	A33
VDD	D1	VDD	D41	VDD	E9	VDD	E17
VDD	E25	VDD	E33	VDD	F9	VDD	F17
VDD	F25	VDD	F33	VDD	G1	VDD	G5
VDD	G6	VDD	G7	VDD	G8	VDD	G9
VDD	G10	VDD	G16	VDD	G17	VDD	G18
VDD	G24	VDD	G25	VDD	G26	VDD	G32
VDD	G33	VDD	G34	VDD	G35	VDD	G36
VDD	G37	VDD	G41	VDD	H35	VDD	J35
VDD	K1	VDD	K35	VDD	L9	VDD	L17
VDD	L25	VDD	L33	VDD	L35	VDD	M9
VDD	M17	VDD	M25	VDD	M33	VDD	M35
VDD	N5	VDD	N6	VDD	N7	VDD	N8
VDD	N9	VDD	N10	VDD	N11	VDD	N12
VDD	N17	VDD	N19	VDD	N21	VDD	N23

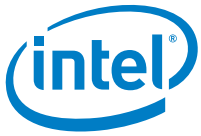


Table 11-2 Pin List Ordered by Name (Continued)

Name	Pin	Name	Pin	Name	Pin	Name	Pin
VDD	N25	VDD	P7	VDD	P18	VDD	P20
VDD	P22	VDD	P24	VDD	R7	VDD	R18
VDD	R19	VDD	R23	VDD	R24	VDD	R41
VDD	T18	VDD	T19	VDD	T23	VDD	T24
VDD	U13	VDD	U15	VDD	U16	VDD	U17
VDD	U18	VDD	U19	VDD	U20	VDD	U21
VDD	U22	VDD	U23	VDD	U24	VDD	U25
VDD	U26	VDD	U27	VDD	V14	VDD	V15
VDD	V16	VDD	V17	VDD	V18	VDD	V19
VDD	V20	VDD	V21	VDD	V22	VDD	V23
VDD	V24	VDD	V25	VDD	V26	VDD	V27
VDD	W13	VDD	Y7	VDD	Y14	VDD	AA5
VDD	AA6	VDD	AA7	VDD	AA13	VDD	AA15
VDD	AA16	VDD	AA17	VDD	AA18	VDD	AA19
VDD	AA23	VDD	AA24	VDD	AA25	VDD	AB7
VDD	AB14	VDD	AC13	VDD	AD14	VDD	AD15
VDD	AD16	VDD	AD17	VDD	AD18	VDD	AD19
VDD	AD20	VDD	AD21	VDD	AD22	VDD	AD23
VDD	AD24	VDD	AD25	VDD	AD26	VDD	AD27
VDD	AE13	VDD	AE15	VDD	AE16	VDD	AE17
VDD	AE18	VDD	AE19	VDD	AE20	VDD	AE21
VDD	AE22	VDD	AE23	VDD	AE24	VDD	AE25
VDD	AE26	VDD	AE27	VDD	AF18	VDD	AF19
VDD	AF23	VDD	AF24	VDD	AG7	VDD	AG18
VDD	AG19	VDD	AG23	VDD	AG24	VDD	AG41
VDD	AH7	VDD	AH18	VDD	AH20	VDD	AH22
VDD	AH24	VDD	AJ5	VDD	AJ6	VDD	AJ7
VDD	AJ8	VDD	AJ9	VDD	AJ10	VDD	AJ11
VDD	AJ12	VDD	AJ17	VDD	AJ19	VDD	AJ21
VDD	AJ23	VDD	AJ25	VDD	AK9	VDD	AK17
VDD	AK25	VDD	AK33	VDD	AK35	VDD	AL9
VDD	AL17	VDD	AL25	VDD	AL33	VDD	AL35
VDD	AM1	VDD	AM35	VDD	AN35	VDD	AP35
VDD	AR1	VDD	AR5	VDD	AR6	VDD	AR7
VDD	AR8	VDD	AR9	VDD	AR10	VDD	AR16
VDD	AR17	VDD	AR18	VDD	AR24	VDD	AR25
VDD	AR26	VDD	AR32	VDD	AR33	VDD	AR34
VDD	AR35	VDD	AR36	VDD	AR37	VDD	AR41

Table 11-2 Pin List Ordered by Name (Continued)

Name	Pin	Name	Pin	Name	Pin	Name	Pin
VDD	AT9	VDD	AT17	VDD	AT25	VDD	AT33
VDD	AU9	VDD	AU17	VDD	AU25	VDD	AU33
VDD	AV1	VDD	AV41	VDD	BA9	VDD	BA17
VDD	BA25	VDD	BA33	VDD25	Y26	VDD25	Y27
VDD25	Y39	VDD25	Y40	VDD25	Y41	VDD25	AA26
VDD25	AA27	VDD25	AA39	VDD25	AA40	VDD25	AA41
VDD25	AB26	VDD25	AB27	VDD25	AB39	VDD25	AB40
VDD25	AB41	VDDK	R33	VDDPLL	AC30	VDDS	A3
VDDS	A5	VDDS	A7	VDDS	A11	VDDS	A13
VDDS	A15	VDDS	A19	VDDS	A21	VDDS	A23
VDDS	A27	VDDS	A29	VDDS	A31	VDDS	A35
VDDS	A37	VDDS	A39	VDDS	D5	VDDS	D13
VDDS	D21	VDDS	D29	VDDS	D37	VDDS	E5
VDDS	E13	VDDS	E21	VDDS	E29	VDDS	E37
VDDS	J41	VDDS	K5	VDDS	K13	VDDS	K21
VDDS	K29	VDDS	L5	VDDS	L13	VDDS	L21
VDDS	L29	VDDS	L37	VDDS	L38	VDDS	L41
VDDS	N1	VDDS	N41	VDDS	R1	VDDS	U1
VDDS	U4	VDDS	U5	VDDS	U10	VDDS	U11
VDDS	W1	VDDS	AA1	VDDS	AC1	VDDS	AE1
VDDS	AE4	VDDS	AE5	VDDS	AE10	VDDS	AE11
VDDS	AG1	VDDS	AJ1	VDDS	AJ41	VDDS	AL5
VDDS	AL13	VDDS	AL21	VDDS	AL29	VDDS	AL37
VDDS	AL38	VDDS	AL41	VDDS	AM5	VDDS	AM13
VDDS	AM21	VDDS	AM29	VDDS	AN41	VDDS	AU5
VDDS	AU13	VDDS	AU21	VDDS	AU29	VDDS	AU37
VDDS	AV5	VDDS	AV13	VDDS	AV21	VDDS	AV29
VDDS	AV37	VDDS	BA3	VDDS	BA5	VDDS	BA7
VDDS	BA11	VDDS	BA13	VDDS	BA15	VDDS	BA19
VDDS	BA21	VDDS	BA23	VDDS	BA27	VDDS	BA29
VDDS	BA31	VDDS	BA35	VDDS	BA37	VDDS	BA39
VDDSK	T33	VFB1	P31	VFB2	R31	VSS	A2
VSS	A4	VSS	A6	VSS	A8	VSS	A10
VSS	A12	VSS	A14	VSS	A16	VSS	A18
VSS	A20	VSS	A22	VSS	A24	VSS	A26
VSS	A28	VSS	A30	VSS	A32	VSS	A34
VSS	A36	VSS	A38	VSS	A40	VSS	B1
VSS	B3	VSS	B5	VSS	B7	VSS	B9

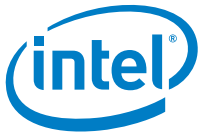


Table 11-2 Pin List Ordered by Name (Continued)

Name	Pin	Name	Pin	Name	Pin	Name	Pin
VSS	B11	VSS	B13	VSS	B15	VSS	B17
VSS	B19	VSS	B21	VSS	B23	VSS	B25
VSS	B27	VSS	B29	VSS	B31	VSS	B33
VSS	B35	VSS	B37	VSS	B39	VSS	B41
VSS	C1	VSS	C9	VSS	C17	VSS	C25
VSS	C33	VSS	C41	VSS	D2	VSS	D8
VSS	D9	VSS	D10	VSS	D16	VSS	D17
VSS	D18	VSS	D24	VSS	D25	VSS	D26
VSS	D32	VSS	D33	VSS	D34	VSS	D40
VSS	E1	VSS	E41	VSS	F1	VSS	F3
VSS	F5	VSS	F7	VSS	F11	VSS	F13
VSS	F15	VSS	F19	VSS	F21	VSS	F23
VSS	F27	VSS	F29	VSS	F31	VSS	F35
VSS	F37	VSS	F39	VSS	F41	VSS	G2
VSS	G3	VSS	G4	VSS	G11	VSS	G12
VSS	G13	VSS	G14	VSS	G15	VSS	G19
VSS	G20	VSS	G21	VSS	G22	VSS	G23
VSS	G27	VSS	G28	VSS	G29	VSS	G30
VSS	G31	VSS	G38	VSS	G39	VSS	G40
VSS	H1	VSS	H3	VSS	H5	VSS	H7
VSS	H9	VSS	H11	VSS	H13	VSS	H15
VSS	H17	VSS	H19	VSS	H21	VSS	H23
VSS	H25	VSS	H27	VSS	H29	VSS	H31
VSS	H33	VSS	H34	VSS	H38	VSS	H41
VSS	J1	VSS	J9	VSS	J17	VSS	J25
VSS	J33	VSS	J34	VSS	J36	VSS	J40
VSS	K2	VSS	K8	VSS	K9	VSS	K10
VSS	K16	VSS	K17	VSS	K18	VSS	K24
VSS	K25	VSS	K26	VSS	K32	VSS	K33
VSS	K34	VSS	K41	VSS	L1	VSS	L34
VSS	L36	VSS	L40	VSS	M1	VSS	M3
VSS	M5	VSS	M7	VSS	M11	VSS	M13
VSS	M15	VSS	M19	VSS	M21	VSS	M23
VSS	M27	VSS	M29	VSS	M31	VSS	M34
VSS	M41	VSS	N2	VSS	N3	VSS	N4
VSS	N13	VSS	N14	VSS	N15	VSS	N16
VSS	N18	VSS	N20	VSS	N22	VSS	N24
VSS	N26	VSS	N27	VSS	N28	VSS	N29

Table 11-2 Pin List Ordered by Name (Continued)

Name	Pin	Name	Pin	Name	Pin	Name	Pin
VSS	N30	VSS	N31	VSS	N32	VSS	N33
VSS	N34	VSS	N35	VSS	N36	VSS	N40
VSS	P1	VSS	P4	VSS	P10	VSS	P16
VSS	P17	VSS	P19	VSS	P21	VSS	P23
VSS	P25	VSS	P26	VSS	P30	VSS	P33
VSS	P35	VSS	P38	VSS	P41	VSS	R2
VSS	R6	VSS	R8	VSS	R12	VSS	R16
VSS	R17	VSS	R20	VSS	R21	VSS	R22
VSS	R25	VSS	R26	VSS	R35	VSS	R36
VSS	R37	VSS	R38	VSS	R39	VSS	R40
VSS	T1	VSS	T7	VSS	T13	VSS	T14
VSS	T15	VSS	T16	VSS	T17	VSS	T20
VSS	T21	VSS	T22	VSS	T25	VSS	T26
VSS	T27	VSS	T28	VSS	T29	VSS	T41
VSS	U2	VSS	U6	VSS	U7	VSS	U8
VSS	U12	VSS	U14	VSS	U41	VSS	V1
VSS	V7	VSS	V13	VSS	V41	VSS	W2
VSS	W6	VSS	W7	VSS	W8	VSS	W12
VSS	W14	VSS	W15	VSS	W16	VSS	W17
VSS	W18	VSS	W19	VSS	W20	VSS	W21
VSS	W22	VSS	W23	VSS	W24	VSS	W25
VSS	W26	VSS	W27	VSS	W39	VSS	W40
VSS	W41	VSS	Y1	VSS	Y4	VSS	Y10
VSS	Y13	VSS	Y15	VSS	Y16	VSS	Y17
VSS	Y18	VSS	Y19	VSS	Y20	VSS	Y21
VSS	Y22	VSS	Y23	VSS	Y24	VSS	Y25
VSS	Y30	VSS	Y31	VSS	Y32	VSS	AA2
VSS	AA3	VSS	AA4	VSS	AA8	VSS	AA9
VSS	AA10	VSS	AA11	VSS	AA12	VSS	AA14
VSS	AA20	VSS	AA21	VSS	AA22	VSS	AA28
VSS	AA30	VSS	AA32	VSS	AB1	VSS	AB4
VSS	AB10	VSS	AB13	VSS	AB15	VSS	AB16
VSS	AB17	VSS	AB18	VSS	AB19	VSS	AB20
VSS	AB21	VSS	AB22	VSS	AB23	VSS	AB24
VSS	AB25	VSS	AB30	VSS	AB31	VSS	AB32
VSS	AC2	VSS	AC6	VSS	AC7	VSS	AC8
VSS	AC12	VSS	AC14	VSS	AC15	VSS	AC16
VSS	AC17	VSS	AC18	VSS	AC19	VSS	AC20

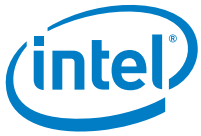


Table 11-2 Pin List Ordered by Name (Continued)

Name	Pin	Name	Pin	Name	Pin	Name	Pin
VSS	AC21	VSS	AC22	VSS	AC23	VSS	AC24
VSS	AC25	VSS	AC26	VSS	AC27	VSS	AC39
VSS	AC40	VSS	AC41	VSS	AD1	VSS	AD7
VSS	AD13	VSS	AD41	VSS	AE2	VSS	AE6
VSS	AE7	VSS	AE8	VSS	AE12	VSS	AE14
VSS	AE41	VSS	AF1	VSS	AF7	VSS	AF13
VSS	AF14	VSS	AF15	VSS	AF16	VSS	AF17
VSS	AF20	VSS	AF21	VSS	AF22	VSS	AF25
VSS	AF26	VSS	AF27	VSS	AF28	VSS	AF29
VSS	AF41	VSS	AG2	VSS	AG6	VSS	AG8
VSS	AG12	VSS	AG16	VSS	AG17	VSS	AG20
VSS	AG21	VSS	AG22	VSS	AG25	VSS	AG26
VSS	AG35	VSS	AG36	VSS	AG37	VSS	AG38
VSS	AG39	VSS	AG40	VSS	AH1	VSS	AH4
VSS	AH10	VSS	AH16	VSS	AH17	VSS	AH19
VSS	AH21	VSS	AH23	VSS	AH25	VSS	AH26
VSS	AH30	VSS	AH35	VSS	AH38	VSS	AH41
VSS	AJ2	VSS	AJ3	VSS	AJ4	VSS	AJ13
VSS	AJ14	VSS	AJ15	VSS	AJ16	VSS	AJ18
VSS	AJ20	VSS	AJ22	VSS	AJ24	VSS	AJ26
VSS	AJ27	VSS	AJ28	VSS	AJ29	VSS	AJ30
VSS	AJ31	VSS	AJ32	VSS	AJ33	VSS	AJ35
VSS	AJ36	VSS	AJ40	VSS	AK1	VSS	AK3
VSS	AK5	VSS	AK7	VSS	AK11	VSS	AK13
VSS	AK15	VSS	AK19	VSS	AK21	VSS	AK23
VSS	AK27	VSS	AK29	VSS	AK31	VSS	AK41
VSS	AL1	VSS	AL34	VSS	AL36	VSS	AL40
VSS	AM2	VSS	AM8	VSS	AM9	VSS	AM10
VSS	AM16	VSS	AM17	VSS	AM18	VSS	AM24
VSS	AM25	VSS	AM26	VSS	AM32	VSS	AM33
VSS	AM34	VSS	AM41	VSS	AN1	VSS	AN9
VSS	AN17	VSS	AN25	VSS	AN33	VSS	AN34
VSS	AN36	VSS	AN40	VSS	AP1	VSS	AP3
VSS	AP5	VSS	AP7	VSS	AP9	VSS	AP11
VSS	AP13	VSS	AP15	VSS	AP17	VSS	AP19
VSS	AP21	VSS	AP23	VSS	AP25	VSS	AP27
VSS	AP29	VSS	AP31	VSS	AP33	VSS	AP34
VSS	AP38	VSS	AP41	VSS	AR2	VSS	AR3

Table 11-2 Pin List Ordered by Name (Continued)

Name	Pin	Name	Pin	Name	Pin	Name	Pin
VSS	AR4	VSS	AR11	VSS	AR12	VSS	AR13
VSS	AR14	VSS	AR15	VSS	AR19	VSS	AR20
VSS	AR21	VSS	AR22	VSS	AR23	VSS	AR27
VSS	AR28	VSS	AR29	VSS	AR30	VSS	AR31
VSS	AR38	VSS	AR39	VSS	AR40	VSS	AT1
VSS	AT3	VSS	AT5	VSS	AT7	VSS	AT11
VSS	AT13	VSS	AT15	VSS	AT19	VSS	AT21
VSS	AT23	VSS	AT27	VSS	AT29	VSS	AT31
VSS	AT35	VSS	AT37	VSS	AT39	VSS	AT41
VSS	AU1	VSS	AU41	VSS	AV2	VSS	AV8
VSS	AV9	VSS	AV10	VSS	AV16	VSS	AV17
VSS	AV18	VSS	AV24	VSS	AV25	VSS	AV26
VSS	AV32	VSS	AV33	VSS	AV34	VSS	AV40
VSS	AW1	VSS	AW9	VSS	AW17	VSS	AW25
VSS	AW33	VSS	AW41	VSS	AY1	VSS	AY3
VSS	AY5	VSS	AY7	VSS	AY9	VSS	AY11
VSS	AY13	VSS	AY15	VSS	AY17	VSS	AY19
VSS	AY21	VSS	AY23	VSS	AY25	VSS	AY27
VSS	AY29	VSS	AY31	VSS	AY33	VSS	AY35
VSS	AY37	VSS	AY39	VSS	AY41	VSS	BA2
VSS	BA4	VSS	BA6	VSS	BA8	VSS	BA10
VSS	BA12	VSS	BA14	VSS	BA16	VSS	BA18
VSS	BA20	VSS	BA22	VSS	BA24	VSS	BA26
VSS	BA28	VSS	BA30	VSS	BA32	VSS	BA34
VSS	BA36	VSS	BA38	VSS	BA40	VSSK	R34
VSSK2	T34						

11.5 EPL Blocks

The FM5000/FM6000 are limited by port bandwidth, but the board designer has the freedom to choose which ports to use. For example, the FM6324 allows up to 240 GbE of core bandwidth. In this case the board designer can chose any 24 10 GbE or 6 40 GbE port locations from the pin list described in the sections that follow.

The FM5224 has fixed locations for the high speed 10 GbE /40 GbE interfaces. There are two choices available. EPL 23 and 24 (port 68..71 or port 72..76 in the port map) can used to provide up to eight 10 GbE SerDes output lanes or two XAUI or two 40 GbE interfaces. An alternate choice is to use EPL1/2 and EPL3/4 (ports 5..7 and ports 8::11 in the port map), which are paired EPLs that can be individually selected by software through an internal MUX. For example, EPL1 could be configured as four SFP+ ports and EPL2 could be configured as a QSFP+ port. In this case, a simple API command can select which port types are active and connected internally to the switch core. Keep in mind that the FM5224 is limited to a maximum of eight 10 GbE or two XAUI or two 40 GbE ports. For additional information about the paired EPL structure, see Section 6.0, "Ethernet Port Logic (EPL)".

11.5.1 FM5224 10 GbE EPL Location Flexibility

The FM5224 is 240 GbE sku of the FM5000/FM6000 series. When all EPLs remain fixed at 1/2.5 GbE operation, there's a maximum of x8 channels to the internal Ethernet ports that can be used at 10 GbE speed.

EPL1..4 (paired EPL1 and EPL2; EPL3 and EPL4) or EPL23/EPL24 can be selected as a dedicated 10 GbE or 40 GbE uplink for SFP+/QSFP dual-design purposes as long as any combination of these EPLs (1>>4, 23/24) don't exceed the x8 10 GbE channel links to the internal switch fabric.

Paired EPL (EPL8) offers more SerDes lane outputs than an independent EPL (EPL4) for SFP+/QSFP dual-design uses, as long as this muxing results without violating the maximum 8 x 10 GbE or 2 x 40 GbE bandwidth capability.

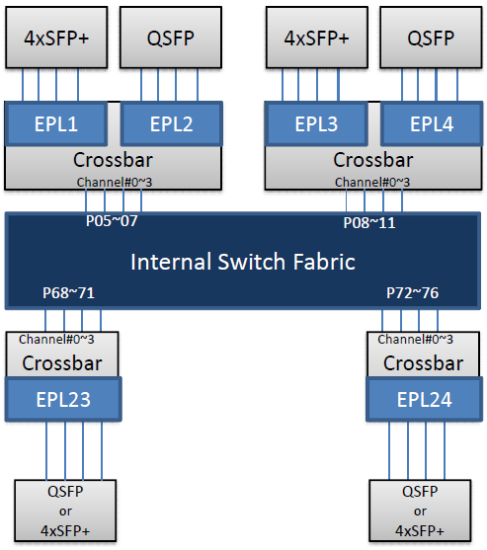
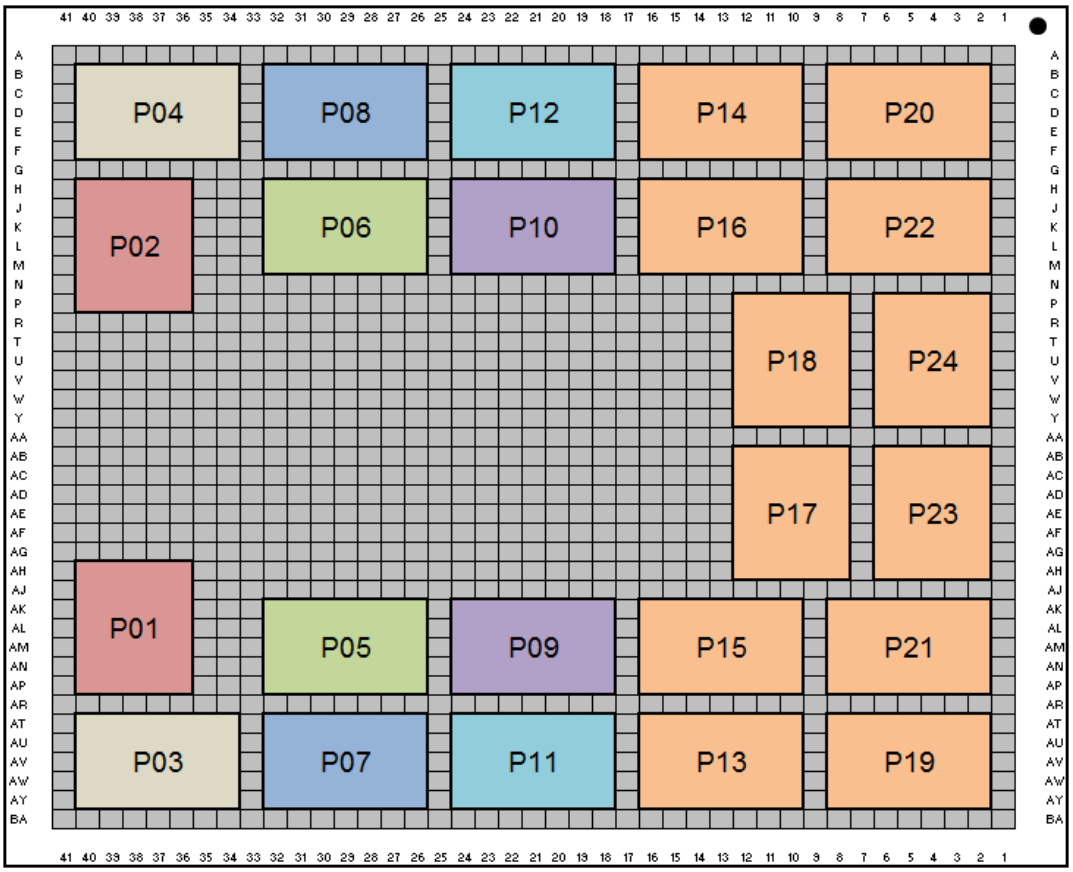


Figure 11-4 FM5224 EPL Blocks



Legend	
EPL4	EPL8
40 Gb BW Each	40 Gb BW Shared
	40 Gb BW Shared
	40 Gb BW Shared
	40 Gb BW Shared
	40 Gb BW Shared
	40 Gb BW Shared

Figure 11-5 FM5000/FM6000 1677 Package EPLs (Bottom View)