Architecture Guide

C-5e/C-3e NETWORK PROCESSOR

SILICON REVISION B0

C5EC3EARCH-RM
Rev 04 PRODUCTION

freescale™ semiconductor

Architecture Guide

C-5e/C-3e Network Processor
Silicon Revision B0

C5EC3EARCH-RM
Rev 04

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

freescale™
semiconductor

CONTENTS

CHAPTER 3 Executive Processor

CHAPTER 4 Fabric Processor

LIST OF FIGURES

LIST OF TABLES

ABOUT THIS GUIDE

Guide Overview

The C-5e/C-3e Network Processor Architecture Guide describes the full architecture of the C-5e Network Processor and describes the differences between the C-5e™ NP and the C-3e™ NP. Freescale reserves the right to change the detail specifications as may be required to permit improvements in the design of its products. It is intended for system architects and developers to enable you to fully understand how the C-5e network processor (C-5e NP) works and how the processor can be used to implement your networking applications. This guide is also useful as a reference during product design and development, and a Register Reference is provided for that purpose. This guide assumes a good familiarity with communications hardware design and implementation. This guide also assumes good working knowledge of the C-Ware Software Toolset.

This guide covers the following topics:

- Introduction describes the major components and functions of the C-5e NP, supported interfaces, addressing scheme, system configurations, cell/packet handling (!OC-48), and cell/packet handling for high speed (OC-48). Also describes the differences between the C-5e NP and the C-3e NP.
- Channel Processors describes the major components and functions of the CPs, processing of data streams, memory areas, interface transactions, configuration space, and using block moves.
- Executive Processor describes the major components and functions of the XP, memory areas, supported external interfaces, initialization options, and internal XP interfaces.
- Fabric Processor describes the major components and functions of the FP, flow sequence for both FPTx and FPRx, Fabric Interface modes supported and their configurations, debug and test features, and setup.
- Buffer Management Unit describes the major components and functions of the BMU, memory areas, types of transactions, configuration space, and setup.

- Table Lookup Unit describes the major components and functions of the TLU, flow process, supported table types, operation, software algorithms, formats, examples, commands, mapping, configuration and status registers, application considerations, and special applications.
- Queue Management Unit describes the major components and functions of the QMU, flow process, memory areas, queuing operations, types of transactions, configuration space, setup, multicast support, system level multicast operations performance, and external scheduler mode.
- Internal Buses describes the interconnect components of the C-5e NP including the Ring Bus, Payload Bus and Global Bus.
- C-5e NP Registers lists all the C-5e NP registers including their function, purpose, address, access, fields, bit positions, default values, and options.
- Using Aggregate Mode describes using the C-5e NP in this mode of operation to support Gigabit Ethernet, FibreChannel, OC-12 and OC-12c interfaces.
- SONET/SDH CP Support describes the mapping between the C-5e NP and SONET Byte Overhead definitions for OC-3c, OC-12, and OC-12c protocols, and configuration.
- RISC Core Custom Instructions describes sixteen (16) custom instructions used in the CPRC and XPRC. The name, format, description and operation for each instruction is provided.
- PCI Byte Swapping describes the C-5e NP feature that allows easy transition between the PCI Bus and C-5e NP environments.
- C-5e NP System Configuration describes C-5e NP system configurations using the M-5 companion device, as well as, the M-5's function, block diagram and specifications.

Information contained in this guide does not represent a commitment on the part of Freescale Corporation.

Architecture Guide Classifications

Table 1 describes the Architecture Guide classifications of Advance, Preliminary, and Production.

Table 1 Architecture Guide Classifications

CLASSIFICATION	DESCRIPTION
Advance Information	Used to advise customers of the proposed addition to the product line. This document will typically contain some useful information including interfacing with the user's system and some specifications. The goal of this document is to allow customers to begin designs but with expectation of changes. Specification details may be changed later without notice.
Preliminary Information	Describes pre-production or first production devices and is usually indicative of production stage performance. Minor changes should be expected as characteristic spreads become better controlled. Specification details may be changed slightly without notice, but the customer can design their product based on this architecture guide.
Production Data	Defines the long-term specified production limits based on fully characterized data. It includes a disclaimer to allow improvements in specifications and modifications that do not affect form, fit or function in original applications; if absolute maximum ratings are changed, they should improve rather than downgrade.

Using PDF Documents

Electronic documents are provided as PDF files. Open and view them using the Adobe® Acrobat® Reader application, version 3.0 or later. If necessary, download the Acrobat Reader from the Adobe Systems, Inc. web site:

http://www.adobe.com/prodindex/acrobat/readstep.html

PDF files offer several ways for moving among the document's pages, as follows:

- To move quickly from section to section within the document, use the *Acrobat bookmarks* that appear on the left side of the Acrobat Reader window. The bookmarks provide an expandable outline view of the document's contents. To display the document's Acrobat bookmarks, press the "Display both bookmarks and page" button on the Acrobat Reader tool bar.
- To move to the referenced page of an entry in the document's Contents or Index, click on the entry itself, each of which is hyperlinked.
- To follow a cross-reference to a heading, figure, or table, click the blue text.
- To move to the beginning or end of the document, to move page by page within the document, or to navigate among the pages you displayed by clicking on hyperlinks, use the Acrobat Reader navigation buttons shown in this figure:

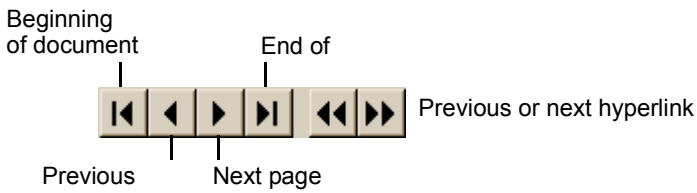


Table 2 summarizes how to navigate within an electronic document.

Table 2 Navigating Within a PDF Document

TO NAVIGATE THIS WAY	CLICK THIS
Move from section to section within the document.	A bookmark on the left side of the Acrobat Reader window
Move to an entry in the Table of Contents.	The entry itself
Move to an entry in the Index.	The page number

Table 2 Navigating Within a PDF Document

TO NAVIGATE THIS WAY	CLICK THIS
Move to an entry in the List of Figures or List of Tables.	The Figure or Table number
Follow a cross-reference (highlighted in blue text).	The cross-reference text
Move page by page.	The appropriate Acrobat Reader navigation buttons
Move to the beginning or end of the document.	The appropriate Acrobat Reader navigation buttons
Move backward or forward among a series of hyperlinks you have selected.	The appropriate Acrobat Reader navigation buttons

Guide Conventions

The following visual elements are used throughout this guide, where applicable:

This icon and text designates information of special note.

Warning: *This icon and text indicate a potentially dangerous procedure. Instructions contained in the warnings must be followed.*

Warning: *This icon and text indicate a procedure where the reader must take precautions regarding laser light.*

This icon and text indicate the possibility of electro-static discharge (ESD) in a procedure that requires the reader to take the proper ESD precautions.

Related Product Documentation

Table 3 lists the user and reference documentation for Freescale's C-Port silicon documentation set.

Table 3 C-Port Silicon Documentation Set

DOCUMENT NAME	PURPOSE	DOCUMENT ID
C-5 Network Processor Architecture Guide	Describes the full architecture of the C-5 network processor.	C5NPARCH-RM
C-5 Network Processor Data Sheet	Describes hardware design specifications for the C-5 network processor.	C5NPDATA-DS
C-5e/C-3e Network Processor Architecture Guide	Describes the full architecture of the C-5e and C-3e network processors.	C5EC3EARCH-RM
C-5e Network Processor Data Sheet	Describes hardware design specifications for the C-5e network processor.	C5ENPB0-DS
C-3e Network Processor Data Sheet	Describes hardware design specifications for the C-3e network processor.	C3ENPB0-DS
C-5 Network Processor to C-5e Network Processor Comparison Delta Document	Describes key architectural features of the C-5e, and highlights main differences between C-5 and C-5e.	C5C5EDELTA-RM
M-5 Channel Adapter Architecture Guide	Describes the full architecture of the M-5 channel adapter.	M5CAARCH-RM
M-5 Channel Adapter Data Sheet	Describes hardware design specifications for the M-5 channel adapter.	M5CA0-DS

Revision History

Table 4 provides details about changes made for each revision of this guide.

Table 4 C-5e/C-3e NP Architecture Guide Revision History

REVISION	CHANGES
04	• Chapter 1, added notes to applicable OC-48c configurations that clarify conditions pertaining to realizing full OC-48c line rate. • Chapter 1, Write Control Blocks (WrCB0_ , WrCB1_) section, changed WrCB0_DMA_Addr bite [13:0] LineAddr field to WrCB0_DMA_Addr bite [13:4] LineAddr field. • Chapter 1, Read Control Blocks (RdCB0_ , RdCB1_) section, changed RdCB0_DMA_Addr bits [15:4] LineAddr field to RdCB0_DMA_Addr bits [13:4] LineAddr field. • Chapter 1, removed all references to the Q-5, FPTx Sequence Numbers and Virtual Queueing functions, Virtual Queueing functions in relation to the Q-5. • Chapter 2, removed all references to Virtual Queueing and the Q-5. • Chapter 4, removed all references to the FPTx Sequence Numbers and Virtual Queueing functions. • Chapter 6, added information about what causes time out errors for Find, Findr and Findw commands to their respective command error type section. • Chapter 7, removed all references to Virtual Queueing and the Q-5. • Appendix A, added information to SDP_Mode3 Register, Manual_FEBE field about K2 operation. • Appendix A, RdCB0_DMA_Addr Register (CP Rd Control Block0 Function), added Reserved field [15:14] to table and bit position artwork. • Appendix F, changed to detail the C-5e NP and M-5 system configurations and the functionality of the M-5.

Table 4 C-5e/C-3e NP Architecture Guide Revision History (continued)

REVISION	CHANGES
03	• Chapter 1, C-3e NP Architecture Overview section, removed OC-3 from the C-5e NP Compared to the C-3e NP (Differences) table as a supported interface for both C-5e and C-3e. • Chapter 2, Configuration for Recirculation Operations Using RxSDP and TxSDP section, added a note that: Bit level recirculation for an SDP is not supported for CPs within the same cluster. • Chapter 2, SDP TxByte Processor Transmit Control Block (TxCB0_, TxCB1_) section, changed a reference that bit [22] Own0 field and bit [23] Own1 field are in TxCB0_Ctl registers rather than in TxCB0_Sys_Addr register. • Chapter 5, BMU Buffer Memory Organization section, changed the Number of Buffers per Pool Legal range= 0 to 65,528 (must be in multiples of 8), rather than 0 to 65,535 (must be in multiples of 8). • Chapter 5, BTag Deallocation Operation section, changed the implementation figure to show the BTag is held in the last (least significant) two bytes inside the last 32bit word of the 64Byte DMEM. • Chapter 5, MUC Allocation Operation section, changed the implementation figure to show the Reference Count (8bytes) is held in the last (least significant) byte inside the last 32bit word of the 64Byte DMEM. • Chapter 6, TLU Performance section, changed the SRAM Access for Find Command table to reflect the PFX assumed key size of 32bits, rather than 48bits. Also, enhanced the TLU Latency section for clarification. • Chapter 6, Add Command section, changed the note to indicate that the read occurs and then six (6) clocks later the value is written back to the SRAM, rather than four (4) clocks. • Chapter 6, Table Lookup Unit (TLU) Overview section, changed 34bit control bus to connect to ZBT SRAMs to 31 control signals to connect to ZBT SRAMs. • Chapter 6, TLU_Memory Register bit position [5:0] BankConfig, clarified the encoded values relation to memory capacity per bank. • Chapter 7, QMU Performance section, updated to reflect QMU performance formula and typical speeds. • Chapter 7, QMU Multicast Support (Non-System Level) section, changed the multicast vector going from 18 to 32bits and table mapping the levels to the correct queue was lengthened from 144 to 256 entries. Also, changed Multicast_Destination0 to Multicast_Destination255 Registers (QMU Configuration Function) in Appendix A.

Table 4 C-5e/C-3e NP Architecture Guide Revision History (continued)

REVISION	CHANGES
03 (Continued)	• Appendix A, Num_Descriptors Register (QMU Configuration Function), changed Legal range= 0 to 16,383, rather than 0 to 16,384. Also, changed applicable values in Chapter 7. • Appendix A, Dyn_Des_Usage_Lim_Pool0 Register (QMU Configuration Function), changed Legal range 0 to 16K -1, rather than 16K-1. Also, changed applicable values in Chapter 7. • Appendix A, Dyn_Descriptor_Pool0_Usage Register (QMU Status Function), changed Legal range= 0 to 16K-1, rather than 0=16K. Added an example for clarification. Also, changed applicable values in Chapter 7. • Appendix A, Num BTags0 to Num BTags29 Registers (Number of BTags in a Pool Function), clarified that the Default value of 0 = 0 BTags allocated. • Appendix A, PCI Device ID Register (XP PCI Configuration Function), added Device ID and Revision ID information for both C-5e B0 and C-3e B0. • Appendix A, PCI Vendor ID register description added. Bit Position 15:0 contains the read-only vendor ID, which is 0x150E. • Appendix A, TxMsg0_Ctl Register (CP Ring Bus Tx Message Control Function), changed Avial field (31bit position) to include SDP in addition to CPRC. • Appendix A, TxCB0_Ctl Register (XP DMEM#24 Transfer Tx Control Block0 Function), changed the Reset Value to 1x0xx00x000x. • Appendix A, TxCB0_Ctl Register (CP Tx Control Block0 Function), clarified the SDP state bit position [19:18] to indicate this field is for SDP engines for CPs or the PCI engine for the XP. • Appendix A, Event0 Register (CP Event and Interrupt Function), changed bit position [61] filed name from MCError to PayloadError. Because the error returns for a request sent to either a BMU or QMU error. • Appendix A, SDP_Mode5 Register (CP Mode Configuration Function), changed information about fields ForcePathAIS3 to ForcePathAIS0 (17:14bits). Specifically, the operation varies according to the SONET mode selected.

Table 4 C-5e/C-3e NP Architecture Guide Revision History (continued)

REVISION	CHANGES
03 (Continued)	• Appendix A, TxFI_Configuration Register (FP Tx Configuration Function), changed bit position [25] filed name from U2TriEnable to U2PHYTriEnable as well as its description. • Appendix B, Aggregate Mode Application Examples section, updated to reflect token passing in the RxByte processor rather than the RxBit processor. • Appendix B, Aggregate Mode Application Examples section, specifically pertaining to Non-blocking Operation, removed the statements about CP cluster mapping and restrictions related to the BMU memory bandwidth. These restrictions do not apply to the C-5e NP. • Appendix C, SONET/SDH Frame Format Overview section, added a figure to show the Receive SONET Pointer State Machine Operation and updated two applicable tables for the corresponding transport overhead byte H1, STS #1 for receive SONET/SDH OC-3c and OC-12/OC-12c. • Appendix C, Determining Signal Degrade/Signal Failure Conditions with C-5e NP section, changed the Detection Threshold formula's subtract sign to an addition sign. An addition sign was added to the example below which changed the resulting values of 581 errors to 663 errors. In addition, in the Possible Settings for OC-3c Detection Times table the threshold value of 581 was changed to 663, and the two 64 values were changed to 63.
02	• Chapter 2, added CRC types available per SDP blocks and their generator polynomial (formula) for each supported CRC type. • Chapter 6, added throughput and latency formula and applicable artwork to clarify TLU performance elements. • Chapter 8, updated internal bus characteristics for both C-5e/C-3e. Added general bandwidth formulas for both C-5e and C-3e. Updated the Ring Bus Latency section. • Appendix A, updated SONET_Event, SDP_Mode2, 3 and 4 registers. Updated XP PCI Device and PCI Revision ID registers. • Appendix C, updated SONET/SDH overhead writable byte s positions, definitions, and notes. Added SONET overhead readable bytes positions, definitions, and notes. Added SONET/SDH frame format overview, and receive OC-3c, OC-12, OC-12c statistics counters for both transport & path overhead. Added sections on: C-5e NP SONET Support, Automatic Protection Switch (APS) support, and Determining Signal Degrade/Failure Conditions. • Appendix D added. Describes sixteen (16) custom instructions used in the CPRC and XPRC. The name, format, description and operation for each instruction is provided. • Glossary, added new terms. Typographic corrections throughout.

Table 4 C-5e/C-3e NP Architecture Guide Revision History (continued)

REVISION	CHANGES
01	• Related User Documentation, changed part numbers. • Chapter 1, added information about the differences between the C-5e NP and the C-3e NP. • Chapter 4, restructured, enhanced, and updated. Added information about CSIX-L1, UTOPIA3 Like to M-5, new weighting algorithm, congestion handling, DBE inputs and outputs, Rx and Tx Byte Processor's mapping between global and byte addressing, Rx and Tx Byte memory space. • Chapter 6, restructured, enhanced, and updated. Added information about Hash-Trie-Key, Chained Hash, Chained Index, PFX (Longest-Prefix Match), Flat Data tables, as well as, TLU operation, software algorithms, and mapping. Changes to: commands, registers, table configuration, data formats and examples. • Chapter 7, added information about: external scheduler mode (Q-5), Payload Bus response format for the external mode, speculative unicast-enqueue operations, and internal SRAM data structures in external mode. Updated the Payload Bus response format for the internal mode. • Appendix A, added new registers and fields in the CP, XP, FP, and QMU configuration space.

Chapter 1

INTRODUCTION

Chapter Overview

This chapter covers the following topics:

- C-5e NP Architecture Overview
- C-5e NP Block Diagram and Flow Processes
- C-5e NP Address Mapping
- C-3e NP Architecture Overview

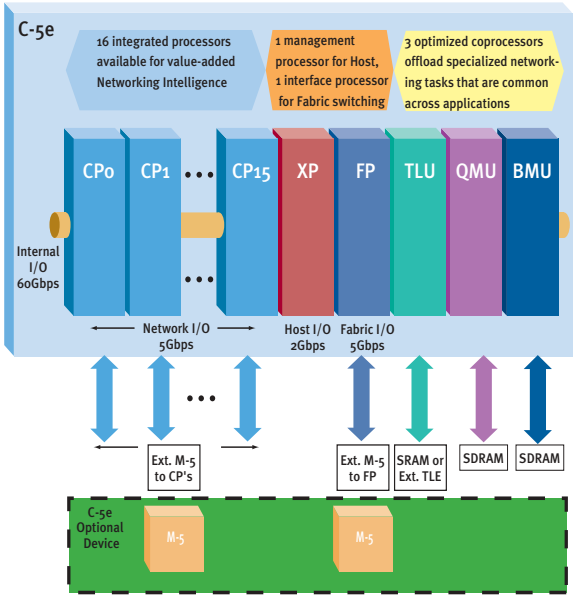
C-5e NP Architecture Overview

The C-5e™ NP implements an architecture specifically designed for communications applications. Cell and packet processing, table lookup processing, and queue management functions are all integrated into the C-5e NP architecture. With the addition of physical interface chips, memory chips (for payload, circuit/routing tables, and payload descriptor queues), and minimal support logic, a single C-5e NP can be used to implement highly-intelligent mixed media, multiport, multiprotocol switches, multiplexors, and concentrators. Multiple C-5e NPs can be used in conjunction with a switching fabric to implement larger scale switching systems.

Highly-Integrated Architecture

The C-5e NP's highly-integrated architecture employs dedicated processors for each networking channel and a series of coprocessors that offload many common networking-specific tasks. Refer to Figure 1 on page 58. This architecture allows the processors and coprocessors to support concurrent processing, which helps the C-5e NP to deliver software flexibility at hardware speeds. In addition, the C-5e NPs RISC instruction set is specially designed to handle communications functions efficiently, even further enhancing its performance. Also the C-5e can be used with the M-5, refer to "C-5e NP System Configuration and Overview" on page 834.

Figure 1 C-5e NP Processors and Coprocessors



C-5e NP Modes of Operation

The C-5e NP supports three (3) different modes of operation (Single Channel, Pipeline Channel, and Aggregate Channel) based upon your application needs, allowing you to increase processing power or increase bandwidth.

Single Channel Mode

CPs operate independently of each other at full duplex and can support, for example, OC-3.

Pipeline Channel Mode

To scale processing power for a particular application, the CPs can be linked for pipelined processing on a single data stream. This allows processing power to be applied independently of data rate. Using this mode, different CPs sequentially process cells/packets, achieving a high-level of processing. For example, AAL2.

Aggregate Channel Mode

To scale serial bandwidth capabilities, the CPs can be aggregated into parallel clusters for wider data streams. The C-5e NP's 16 CPs can be partitioned into four (4) groups of four (4) CPs called *clusters*. Clusters allow the CPs to share resources (IMEM and DMEM) and support aggregation. A cluster of CPs can be configured, for example, to work together to support one physical interface (such as OC-12), or either the receive or transmit portion of one physical interface (such as Gigabit Ethernet). For more information about Aggregation in the C-5e NP. Refer to Appendix B.

C-5e NP Supported Interfaces

The C-5e NP's architecture supports a variety of industry-standard serial and parallel protocols and individual port data rates ranging from DS1 (1.544Mbps) to Gigabit Ethernet (1000Mbps). The interfaces supported include:

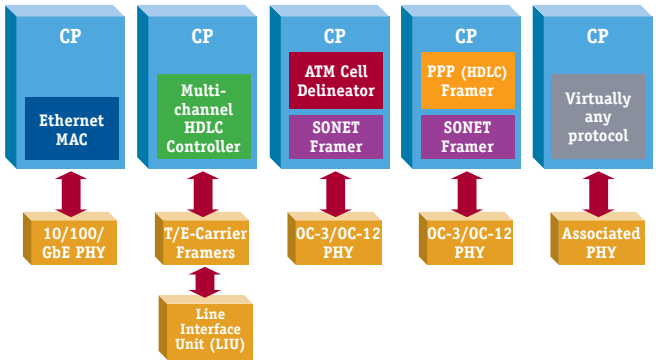
- 10/100Mb Ethernet (RMII)
- 1Gb Ethernet (GMII and TBI)
- OC-3c
- OC-12 (as an aggregation of four OC-3c data streams)/OC-12c
- OC-48c (using various configurations). Refer to "C-5e NP System Configuration and Overview" on page 834.

- OC-48 (using various configurations). Refer to "C-5e NP System Configuration and Overview" on page 834.
- 100Mbit FibreChannel

Each CP comprises of a set of microprogrammable, special-purpose processors, called Serial Data Processors (SDPs), that provide features such as Ethernet MAC and SONET framing, multichannel HDLC control, and ATM cell delineation. Figure 2 on page 60 shows the physical interfaces and examples of the processing provided by the CP's SDPs for the interface type.

The programmability of the C-5e NP can also support a variety of special interfaces, such as various xDSL encapsulations and proprietary protocols.

Figure 2 Examples of SDP Programmability



Major Components of the C-5e NP

The C-5e NP contains eighteen (18) processors (16CPs, XP, and FP) and three (3) coprocessors that operate as shared resources for the CPs and each other, and perform some networking-specific tasks. Refer to Table 5 on page 61.

Table 5 C-5e NP Major Components

ITEM	DEVICE TYPE	FUNCTION
Channel Processor (CP)	Programmable Processor	The programmable Channel Processors (CPs) are responsible for receiving, processing, and transmitting cells or packets. The CP's design and on-chip memory architecture incorporate a number of features that result in a uniquely capable engine for the execution of high-performance data communication tasks.
Executive Processor (XP)	Programmable Processor	Provides network control and management functions in user applications. The XP's Peripheral Component Interface (PCI) supplies an industry-standard 32bit 33/66MHz channel to attach additional processors and line interfaces. The XP also has a PROM and serial bus interface.
Fabric Processor (FP)	Programmable Processor	Manages the high-speed fabric interface. FP channels attach to a switch fabric or very high performance line interfaces. The FP supports the CSIX-L1, UTOPIA-1, -2, -3, Power X(CSIX-L0), RRIZMA, and UTOPIA3 Like to M-5 protocols.
Buffer Management Unit (BMU)	Programmable Coprocessor	Manages centralized payload storage during the forwarding process. An independent high-bandwidth memory interface connects to external memory for the actual storage of payload data.
Table Lookup Unit (TLU)	Programmable Coprocessor	Provides table search and associated data storage services to the CPs, XP, and FP. An independent high-bandwidth memory interface connects to external memory for storage of circuit and forwarding tables.
Queue Management Unit (QMU)	Programmable Coprocessor	Manages application-defined descriptor queues among the CPs, FP, and the XP. An independent high-bandwidth memory interface connects to external memory for storage of payload descriptor queues.

C-5e NP Interconnect Components

The C-5e NP also contains three (3) independent data buses that provide internal communication paths between the eighteen (18) processors (16CPs, XP, and FP) and three (3) coprocessors, supporting concurrent processing. Refer to Table 6 on page 62.

Table 6 C-5e NP Interconnect Components

ITEM	DEVICE TYPE	FUNCTION
Payload Bus	Slotted, multichannel, shared, arbitrated bus	Carries payload data and payload descriptors between the processors and the BMU and QMU.
RIng Bus	Slotted ring-topology bus	Provides bounded latency transactions between the processors and the TLU. It also supports inter-processor communication.
Global Bus	Slotted, multichannel, shared, arbitrated bus	Supports inter-processor communication via a conventional flat memory-mapped addressing scheme.

Other Supported Features

In addition, the C-5e NP provides these other features that provide better application integration. Refer to Table 7 on page 62.

Table 7 C-5e NP Other Supported Interfaces

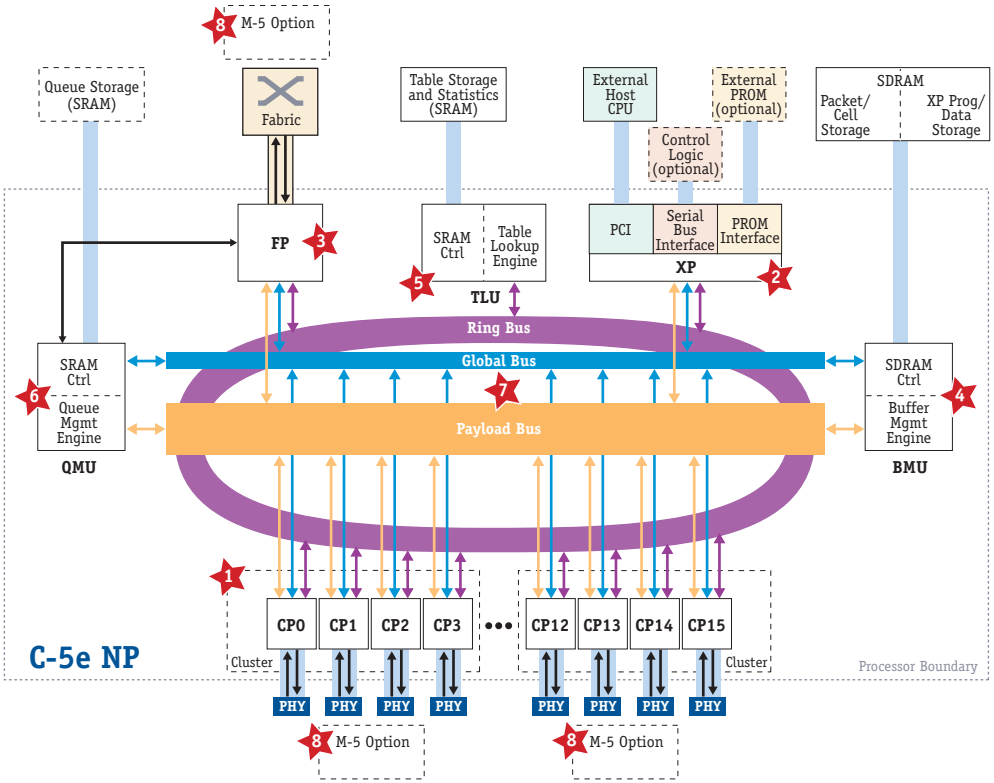
FEATURES	FUNCTION
Byte Swapping	Used to move data between the PCI Bus Little Endian environment and the C-5e NP Big Endian environment. Refer to Appendix E for details.
SONET/SDH Support	Provides hardware support to extract, insert and analyze SONET/SDH (Synchronous Digital Hierarchy). Refer to Appendix C for details.
Multicast Operations	Allows multicast elaboration using the BMU and QMU components of the C-5e NP. Refer to "Multicast Support (System Level)" on page 453 for details.

C-5e NP Block Diagram and Flow Processes

The full architecture of the C-5e NP is shown in Figure 3 on page 64. The major components of the C-5e NP are numbered on the diagram, as well as one (1) external companion device, and include:

1. Channel Processors (CPs)
2. Executive Processor (XP)
3. Fabric Processor (FP)
4. Buffer Management Unit (BMU)
5. Table Lookup Unit (TLU)
6. Queue Management Unit (QMU)
7. Internal Buses (Ring Bus, Global Bus and Payload Bus)
8. Channel Adapter (M-5), external companion device

Figure 3 C-5e NP Simplified Block Diagram



Cell and Packet Forwarding Overview (! OC-48)

Each CP within the C-5e NP has special-purpose components that aid in cell/packet parsing and verification. These components also aid in creating separate data and control paths for the cell/packet. Data is sent via the Payload Bus to the BMU for temporary storage in SDRAM. In parallel with the data being stored, application-specific control data is abstracted into a short descriptor that is used to make forwarding decisions. All interfaces use the Ring Bus to consult forwarding tables in the TLU. The interfaces access the QMU (to enqueue frame descriptors to another interface or processor) through the Payload Bus. Cells/packets that are terminated in the chip or require management processing (such as for routing updates) are enqueued for handling by the XP.

Note that the receive and transmit processes can occur on the same or on different CPs.

The CP handles typical packet forwarding as described in the following sections. The receive and transmit flows are described in detail and shown in Figure 4 on page 67.

Receiving Packets

1. On reception of a serial bit stream, the RxSDP (Receive Serial Data Processor) program detects the packet framing and organizes the bit stream into a byte stream. The SDP program also characterizes and parses the byte stream, performing pattern matching and checking validity criteria.
2. The RxSDP places application-defined fields in Extract Space for access by the CPRC (Channel Processor RISC Core).
3. The SDP launches table lookup requests on extracted data fields using the Ring Bus.
4. Concurrently, the byte stream is transported to a double 64-byte buffer in local DMEM (Data Memory), where it accumulates 64-byte segments (until reaching the end of the packet). The 64-byte segments are transported via the Payload Bus and the BMU to pre-allocated packet buffers in external SDRAM.
5. The CPRC program, upon receiving the extracted data fields and table lookup results, determines the destination queue and other forwarding parameters for the packet, and constructs a forwarding descriptor data structure. This data structure, at a minimum, includes the identity of the packet buffer in which the packet resides. The descriptor is transmitted by the CPRC receive program to the QMU via the Payload Bus. The QMU copies the descriptor into a descriptor buffer and chains that buffer onto the desired queue.

Transmitting Packets

6 The transmitting CPRC program discovers, via a background queue status distribution mechanism, that a queue it services contains a forwarding descriptor. The program reads the descriptor via the Payload Bus from the QMU.

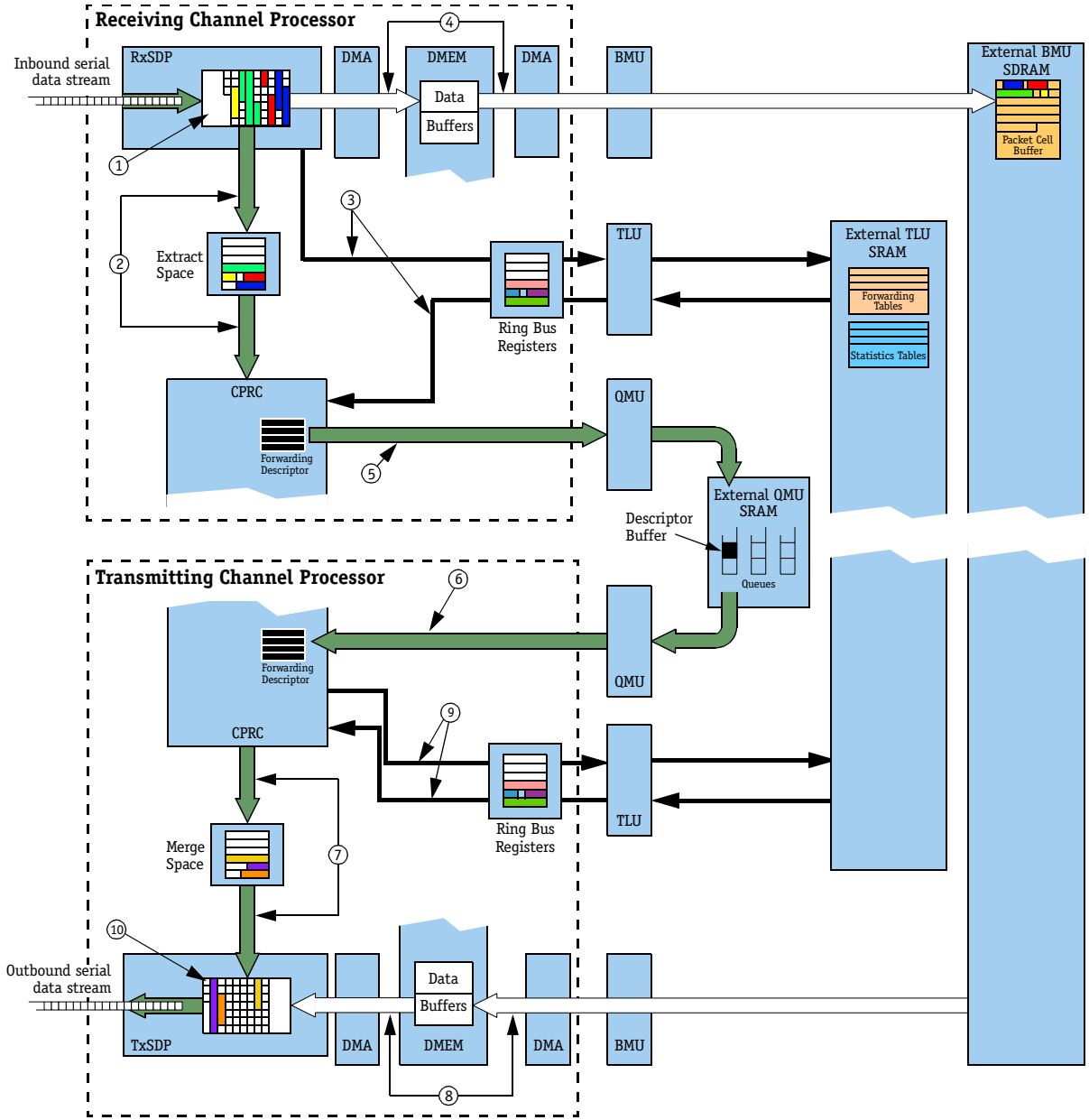
7 The transmit CPRC program inspects the descriptor and using the information it contains, parameterizes the TxSDP program by filling the Merge Space with the format information and packet data field contents necessary to perform the packet transformation. The CPRC sets up the payload transfer from SDRAM via the BMU to local DMEM.

8 The data stream is transported from the BMU via the Payload Bus in 64-byte segments to a double 64-byte buffer in DMEM. The segments are then passed as a byte stream to the TxSDP program, which transforms the packet, substituting data fields from the Merge Space.

9 As a part of either the receive or transmit process, the CPRC program can exercise other C-5e NP resources, such as performing additional table lookups or accessing packet data directly as it flows through the DMEM data buffers.

10 The TxSDP converts the byte stream to a serial bit stream, applies framing, and transmits the bit stream.

Figure 4 Typical Cell/Packet Forwarding Application Receive and Transmit Data Flow (! OC-48)



Cell and Packet Forwarding Overview (OC-48)

The C-5e NP handles OC-48 in a similar fashion as !OC-48 except, for the addition of the external Channel Adapter (M-5) companion device (optional) in front of the CPs for PDU ordering and bus translations.

The receive and transmit flows for !OC-48 are described in detail and shown in Figure 4 on page 67 and the receive and transmit flow for OC-48 is similar with the differences described here.

Receiving Packets

The Rx flow for OC-48 is similar to that for !OC-48 except for the following:

- Prior to the serial bit stream being received by the RxSDP (item1), the external M-5 companion device provides the required PDU ordering for both packet and cells for the CPs, as well as bus translations from multi-physical (MPHY), UTOPIA Level 3 (UTOPIA-L3), and Saturn POS-PHY Level 3 (POS-PHY-L3).

Transmitting Packets

The Tx flow for OC-48 is similar to that for !OC-48 except for the following:

- After the byte stream is converted to a serial bit stream by the TxSDP, framing applied and the bit stream is transmitted (item 10), the M-5 orders the packets /cells and provides the required bus translation.

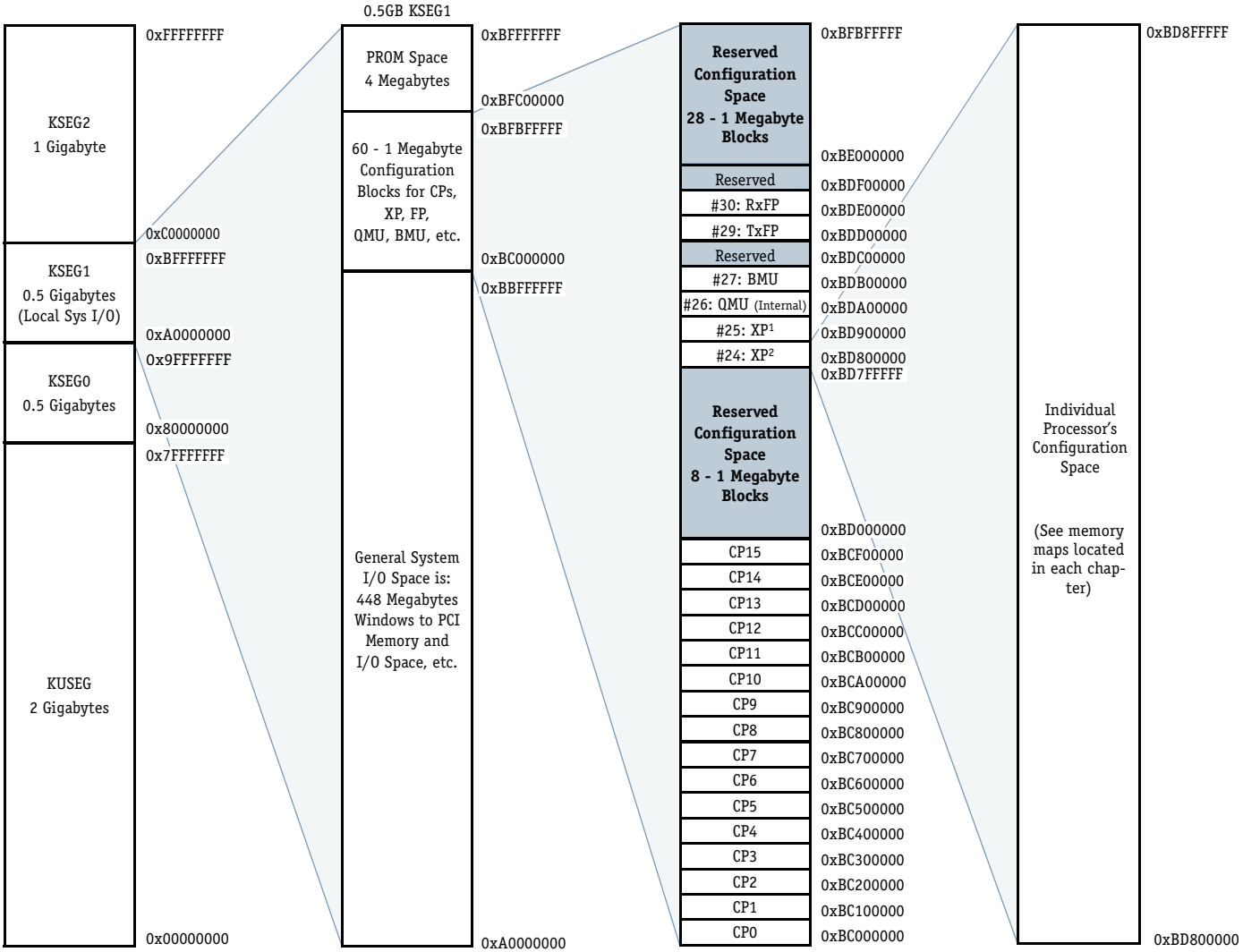
C-5e NP Address Mapping

The C-5e NP supports inter-processor communication using a single, flat memory model. This allows all C-5e NP processors to view all memory-mapped state and configuration registers.

The Channel Processors (CPs) cannot access certain registers reserved exclusively for the Executive Processor (XP).

Each memory resource in the C-5e NP is mapped within its own contiguous 1MByte block of memory. Thus, the specific location of any processor's resource block (local DMEM and registers) within the physical memory space can be mapped using multiple 1MByte offsets. Refer to Figure 5 on page 70.

Figure 5 C-5e NP Physical Address Memory Map



1: XP #25 can only be accessed by the XP, it is not visible to CPs.
2: The CPs can only access DMEM in XP #24.

Configuration Register Definitions

The CP configuration registers form the "base register set" for the C-5e NP. Each CP duplicates the base register set within its own configuration space.

The XP registers include a subset of the base register set, as well as the system interface registers. The XP's "CP-like" base registers are located at the same address offsets within the XP's configuration space as are the CP configuration registers. The FP has a subset of the "CP-like" base registers.

The configuration registers are listed in this chapter by incremental address (CPs, XP, FP, QMU, and so on).

Processor Base Address Offsets

Each C-5e NP processors/coprocessors have a unique 5bit processor ID value. The starting address for any CP (as well as the XP, FP, QMU, BMU, and TLU) can be determined by adding 0xBC000000 to the 5bit processor ID shifted left 20 bit positions, to provide the 1MByte of address space assigned to each processor. For example, the address space for CP5 is: 0xBC000000 + (0x05 << 20) = 0xBC500000. Refer to the memory map in Figure 5 on page 70. In addition, the Ring Bus node IDs for the CPs, XP, FP, and TLU are listed in Table 8 on page 71.

Table 8 Ring Bus Node IDs

UNIT	NODE ID	UNIT	NODE ID
CP0	0	CP10	10
CP1	1	CP11	11
CP2	2	CP12	12
CP3	3	CP13	13
CP4	4	CP14	14
CP5	5	CP15	15
CP6	6	XP	24
CP7	7	FP*	30
CP8	8	TLU	31
CP9	9		

* Transmit only. The FP cannot read messages on the Ring Bus. Thus any messages sent to the FP cannot be removed from the Ring Bus, eventually filling up the Ring Bus.

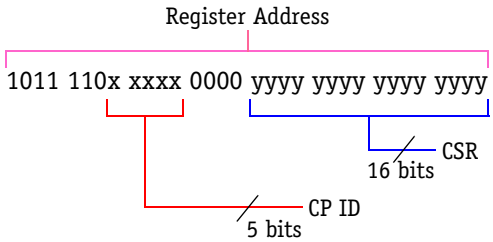
As shown in Figure 5 on page 70, within this space, CP0's 12kBytes of local DMEM begins at the base address (for CP0 this is 0xBC000000), the configuration register space begins at the base address plus 16kBytes (0xBC004000) (with a CP ID shifted of 20 bits). This mapping makes all of DMEM and configuration space available within the positive signed 16bit offset from the CP0 base address.

Configuration Register Address Offsets

Most configuration registers are described only once in this manual. However, some registers have unique variations for different processors. In this case, each register variation is defined.

The register addresses as listed substitute the letter *n* in the address for the processor's ID. By substituting the processor's ID for *n*, you can calculate the address for all individual registers in the C-5e NPs address space. Refer to Figure 6 on page 72 and Figure 5 on page 70.

Figure 6 Register Address Format (in bits)



Byte Ordering

The C-5e NP uses Big Endian byte ordering. However, the XP because of its Peripheral Component Interface (PCI), is capable of handling either Big or Little Endian format from the PCI.

It is recommended that developers use only Big Endian format when developing applications.

C-3e NP Architecture Overview

This section provides an overview of the C-3e NP. The C-3e NP implements an architecture similar to that of the C-5e NP except it targets applications with lower bandwidth and lower power requirements than those traditionally targeted by the C-5e NP. Figure 7 on page 73 shows the C-3e block diagram and Table 9 on page 74 lists the differences between the C-5e NP and C-3e NP.

Figure 7 C-3e NP Block Diagram

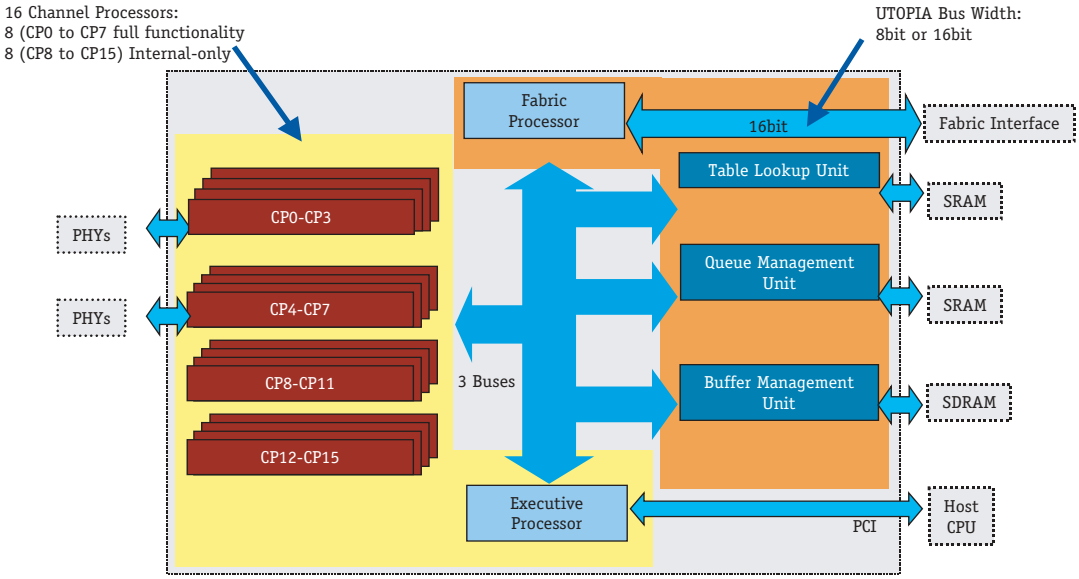


Table 9 C-5e NP Compared to the C-3e NP (Differences)

CHARACTERISTIC	ITEM	C-5E	C-3E
Electrical	Frequency	266MHz typical	180MHz typical
	Voltage	1.2V, +/- 0.1V	1.1V, +/- 0.1V
	Power Consumption	9W@266MHz typical	5W@180MHz nominal
Physical	Layout	840 pin HiTCE CBGA, 1mm	728 pin HiTCE CBGA, 1mm
Performance	Supported Interfaces	• 10/100Mb Ethernet (RMII) • 1Gb Ethernet (GMII and TBI) • OC-3c • OC-12 (as an aggregation of four OC-3c data streams)/OC-12c • OC-48c (using various configurations). Refer to "C-5e NP System Configuration and Overview" on page 834. • OC-48 (using various configurations). Refer to "C-5e NP System Configuration and Overview" on page 834. • 100Mbit FibreChannel	• 10/100Mb Ethernet (RMII) • 1Gb Ethernet (GMII and TBI) • OC-3c • OC-12 (as an aggregation of four OC-3c data streams)/OC-12c • 100Mbit FibreChannel Note: For Gbit Ethernet, full line rate is *only* achieved using dual-cluster, single-interface mode for minimum sized packets (64Bytes). Also, for both Gb Ethernet and OC-12 *full* speeds are not possible through a single queue.
	Supported Fabric Interfaces	• UTOPIA1,2, and 3, IBM PRIZMA, PowerX (CSIX-L0), and CSIX-L1	• UTOPIA1, UTOPIA2, and UTOPIA3 as detailed here:

UTOPIA LEVEL	BUS WIDTH (BITS)	MODE
-L1	8	ATM
	16	ATM/PHY
-L2	8	ATM
	16	ATM/PHY
-L3	8	ATM
	16	ATM/PHY

Table 9 C-5e NP Compared to the C-3e NP (Differences) (continued)

CHARACTERISTIC	ITEM	C-5E	C-3E
Functional	CPs Available	• 16 (CP0 to CP15) with full functionality, configurable for full I/O or recirculation	• 8 (CP0 to CP7) with full functionality, configurable for full I/O or recirculation. • 8 (CP8 to CP15) with limited functionality, configurable only for bit and Byte level recirculation.
	Bit Loopback Operation (*CP PIN_Mode* register)	No restriction.	Must set: • bits [20:18] *RxClkMUX* field to 6 for inverted transmit clock (for local loopback) and • bits [17:14] TxClkMUX field to 1 for T1 source. Refer to "PIN_Mode Register (CP Mode Configuration Function)" on page 546.
Logic Design	XP *PCI Device ID* register	• 0x2	• 0x3 Refer to "PCI Device ID Register (XP PCI Configuration Function)" on page 588.
	XP *PCI Revision ID* register	• 0x10	• 0x10 Refer to "PCI Revision ID Register (XP PCI Configuration Function)" on page 593.
FP	FP Pin Voltage	Supports 3.3V and 2.5V devices.	Supports 3.3V devices.
	FP Performance	• FPTx single queues reach OC-12 speeds.	• FPTx single queues do *not* reach OC-12 speeds. • The maximum achievable FPTx bandwidth is 1.66Gb/s (UTOPIA3 16bit), or 800Mb/s (UTOPIA2).
Pinouts	CP, FP, NC, and CLKS	Refer to *C-5e Network Processor Data Sheet*.	Refer to Table 10 on page 76.
Compatibility	Companion device	• M-5	

Table 10 on page 76 lists those pins that are *not* used for the C-3e NP compared to the C-5e NP.

Table 10 112 Pins Not Used for C-3e NP that are Used for C-5e NP

SECTION	SIGNAL NAME	TOTAL PINS	TYPE	I/O	SIGNAL DESCRIPTION
CP8	CP8_0 - CP8_6	7	LVTTL/ LVPECL	I/O	Refer to *C-5e Network Processor Data Sheet.*
CP9	CP9_0 - CP9_6	7		I/O	
CP10	CPA_0 - CPA_6	7		I/O	
CP11	CPB_0 - CPB_6	7		I/O	
CP12	CPC_0 - CPC_6	7		I/O	
CP13	CPD_0 - CPD_6	7		I/O	
CP14	CPE_0 - CPE_6	7		I/O	
CP15	CPF_0 - CPF_6	7		I/O	
FP	FIN16 - FIN31	16	LVTTL	I	Fabric Data Bus In
FP	FPRxCTL5 - FPRxCTL3	3		I/O	Receive Control Signals
FP	FPOUT16 - FOUT31	16		O	Fabric Data Bus Out
FP	FPTxCTL5 - FPTxCTL3	3		I/O	Transmit Control Signal
NC	NC0 - 9	10		I/O	Reserved for future functionality
CLKS	CCLK4 - 7	4		I	Clock
FP	VDDF	4	P	N/A	Fabric I/O Supply (3.3 or 2.5V)
Total		**112**			

Chapter 2

CHANNEL PROCESSORS

Chapter Overview

This chapter covers the following topics:

Channel Processors (CPs) Overview

The C-5e NP has a dedicated, programmable CP for each of its sixteen (16) line interfaces to handle cell and packet forwarding. The CPs are designated (CP0,CP1 ... CP14, CP15) and can be aggregated to handle higher-speed interfaces and share memory resources. Each CP consists of Serial Data Processors (SDP) and a Channel Processor RISC Core (CPRC), which together perform cell and packet processing via special-purpose memories, namely Instruction Memory (IMEM) and Data Memory (DMEM) that loosely couple these processors. Refer to for details about Aggregation.

CP Major Components

The major components of each CP are listed in Table 11 on page 78. In addition, Figure 8 on page 79 shows the CP Block Diagram.

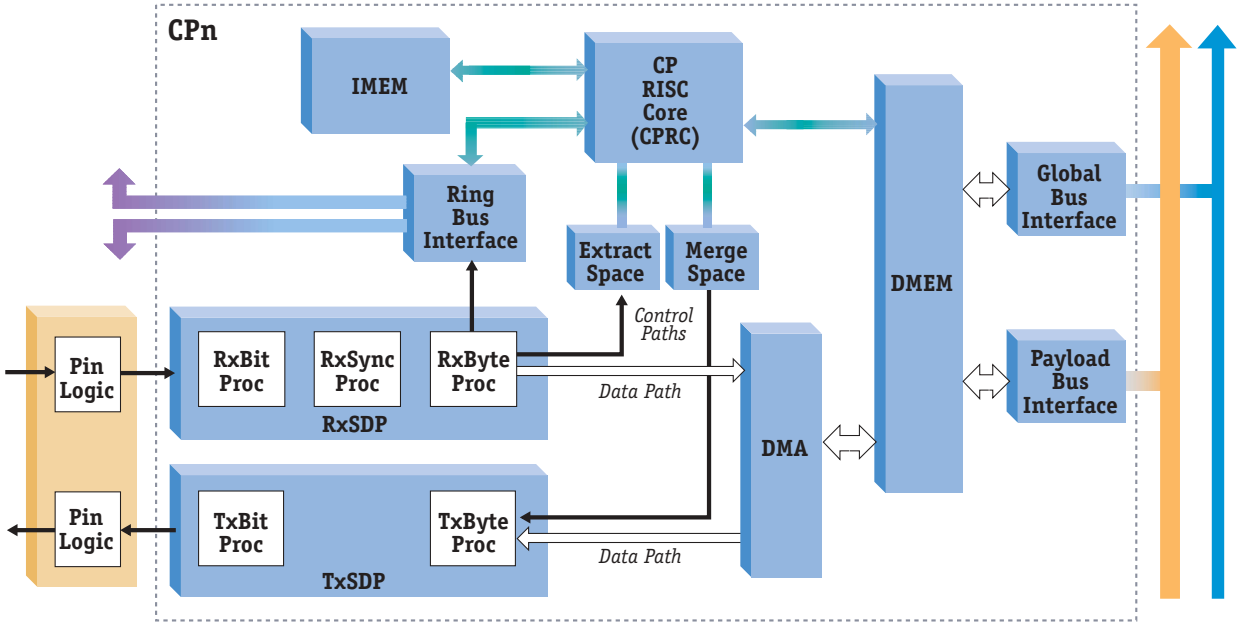
Table 11 Major Components of the CPs and Their Functions

ITEM	FUNCTION
SDPs	Provide microprogrammable interfaces for receive (Rx) and transmit (Tx) between external serial streams and the rest of the CP elements. The receive SDP (RxSDP) and the transmit SDP (TxSDP) can be programmed to process some of the most common types of networking traffic, such as SONET, Ethernet, and ATM. The RxByte programmable processor of the RxSDP and the TxByte programmable processor of the TxSDP can be further programmed for specialized applications. On receipt of a cell/packet, the RxSDP provides serial-to-parallel conversion, validates and interprets the header and payload, and initiates table lookups. On transmission of a cell/packet, the TxSDP applies the header and payload, and provides parallel-to-serial conversion.
CPRC	Programmed to support the following application functions: • Characterizing cells/packets and building descriptors • Initiating additional table lookups • Collecting all table lookup results • Making forwarding and filtering decisions based on the parsed header data and table lookup results (classifying cells/packets) • Making scheduling decisions (based on the characterization of cells/packets) The CPRC implements a subset of the MIPS™ 1 instruction set (excluding multiply, divide, floating point, unaligned loads and stores, move to hi and move to lo), eight (8) Branch Likely instructions from the standard MIPS™ 2 instruction set, and sixteen (16) custom instructions. In addition, the CPRCs support four-way fast context switching.

Table 11 Major Components of the CPs and Their Functions (continued)

ITEM	FUNCTION
Memory	Two (2) types of memory are available: IMEM and DMEM. • Each CP has 8kBytes IMEM that contain the RISC Instructions in RAM. In Cluster mode, four (4) adjacent CPs provide 32kBytes instruction memory (IMEM) that are shared among the CPs within that cluster. • Each CP has 12kBytes of local non-cached data memory (DMEM) for storage of data. Each CPRC can access the local DMEM of any other CPRC within that cluster within one to four additional cycles of latency (depending on CP contention for the DMEM) for a total of 48kBytes. In addition, the DMEM can also be accessed as remote memory by other CPs and the XP via the Global Bus.
Configuration Space	This area of the CP contains a number of registers used to communicate with the SDP and the bus controllers (Payload Bus and Global Bus). The CP's registers can also be accessed by other components of the C-5e NP, (XP and other CPs via the Global Bus).

Figure 8 Channel Processor Block Diagram



Serial Data Processors (SDPs) Overview

Each CP includes a Serial Data Processor (SDP) that contains microcode-programmable components for receive processing (RxSDP) and for transmit processing (TxSDP). Each SDP provides a programmable bit- and byte-level interface to the physical layer (PHY) and acts as the interface between external serial data streams and all other CP elements. The SDP can also launch table lookups to the Table Lookup Unit (TLU).

The SDPs and CPRC operate independently on their specific forwarding tasks and interact to forward a packet to its destination. For example, during receive operations, the RxSDP assembles cell/packet data into DMEM buffers that are written to SDRAM over the Payload Bus under CPRC control. In the process, the RxSDP extracts application-defined fields, placing them in shared registers for access by the CPRC.

Code running on the CPRC characterizes the incoming cell/packet, synthesizes a descriptor that directs the management and routing of the cell/packet, and classifies the cell/packet by enqueuing the descriptor to the appropriate QMU queue. Because the SDP and the CPRC are pipelined to forward cells and packets, many parts of the forwarding process can be performed concurrently.

Supported External Interfaces

Each CP currently supports (in hardware and C-Ware application microcode) the following external interfaces as shown in Table 12 on page 80.

The programmability feature of the SDPs enables support for many other physical interface types, including various xDSL encapsulations and proprietary protocols.

Each type of physical layer PHY has both a characteristic frequency and a particular configuration for the input and output clocks. To accommodate these needs, each CP has a transmit clock mux (txclk mux) that can select among eight (8) global clocks that are sourced externally, two (2) clocks that are sourced from CPs and driven globally, or a clock that is received locally (that is, for OC-12c).

Table 12 Supported Interfaces & Transmit Clock Mux Selects

SUPPORTED INTERFACES	SUGGESTED SOURCE	ENCODED VALUE	APPLICABLE NOTES
T1	CCLK0	1	N/A
E1	CCLK1	2	
E3	CCLK2	3	
T3	CCLK3	4	

Table 12 Supported Interfaces & Transmit Clock Mux Selects (continued)

SUPPORTED INTERFACES	SUGGESTED SOURCE	ENCODED VALUE	APPLICABLE NOTES
RMII	CCLK4	5	CCLK4-7 are internally tied to 0 in C-3e NP.
Fibre Channel	CCLK5	6	
MII	MII clk (CPn_1 in each cluster)	7	To be used by MII Mode for GMII autonegotiate down to 100/10BaseT.
GMII/Gigabit Ethernet	CCLK6	9	CCLK4-7 are internally tied to 0 in C-3e NP
OC3	CCLK7	A	
	internal0	B	Internal0 and internal1 are internally buffered versions of the receive clocks on CP4 and CP8, respectively.
	internal1	C	
	receive clock	D	Use the 'even' receive clock in an 'even-odd' pair, for example CP8 and CP9 will use receive clock from CP8.
	receive clock	E	N/A
0,8,F = transmit clock disabled (internally set to 0) for both C-5e and C-3e			

The SDP can be run synchronously with the C-5e NP clock by driving the C-5e NP's clock signal into the corresponding external clock port to which the SDP is configured.

Up to two (2) receive clock muxes (rxclk mux), *internal_0* and *internal_1*, can also be used as transmit clocks for other CPs. This is intended for Telephony applications where the received clock is considered to be more accurate than the local clock source. The CPs that drive these internal clocks are limited to two fixed locations, CP0 and CP8.

CP I/O requires configuration based on the configuration register that specifies the type of port. In addition, there are controls for the rclk mux and the tclk mux that are also based on the port type.

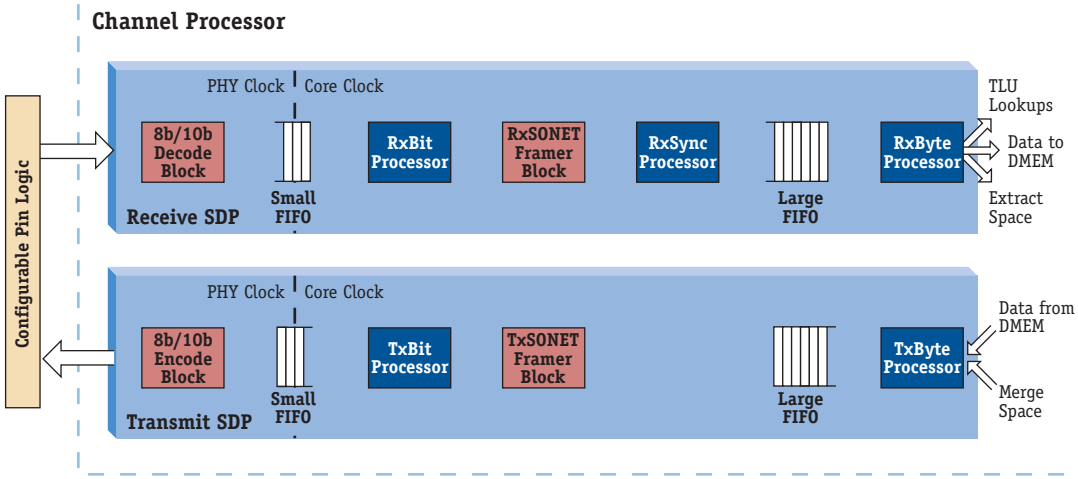
SDPs Functions

The SDP is a pipeline of serially-connected, microcode programmable processors and configurable logic blocks, as shown in Figure 9 on page 82. The data path among these processors and blocks is application configurable. These processors and blocks implement a programmable interface between a port's PHY, where data is serialized, encoded, and encapsulated in protocol-dependent ways, and the CPRC, which expects data to be byte-wide, decoded, delineated, and with header fields naturally aligned.

The receive SDP (RxSDP) accepts serial data from the C-5e NP's physical layer circuitry and performs serial-to-parallel conversion on the data. It delineates frames and cells from the protocol that carries them, performs error and data integrity checks, and compares and extracts fields in the data stream. It provides the contents of the extracted fields to the CPRC in the CPRC's Extract Space. The CPRC's local memory (DMEM) provides buffering for the payload data going to SDRAM.

During data transmission, the CPRC's DMEM buffers the payload data that originates from SDRAM. The transmit SDP (TxSDP) performs field insertion, deletion, and replacement in the outgoing data stream using fields from the CPRC's Merge Space. It performs checksum and CRC generation, frame encapsulation and encoding, and parallel-to-serial conversion on the data. The TxSDP forwards the data to the C-5e NP's external physical layer circuitry.

Figure 9 Rx and Tx SDP Programmable Processors and Configurable Logic Blocks



If a set of CPs is aggregated, those CPs' SDPs are also configured as aggregated. The behavior of aggregated SDPs is described in .

SDPs Major Components

The RxSDP and TxSDP features two (2) types of hardware. Refer to Table 13 on page 83.

Table 13 Types of Hardware Features in the RxSDP and TxSDP

TYPE OF HARDWARE FEATURES	ITEMS	FUNCTION
Programmable Processors	RxBit Processor, RxSync Processor, RxByte Processor, TxBit Processor and TxByte Processor. Refer to Figure 9 on page 82.	Each embedded processor has special-purpose hardware such as a dedicated instruction store, a bank of internal registers, ALU, CAM, CRC engines, and so on for performing a unique set of operations. See the *C-Ware Microcode Programming Guide (part number CSTMCPG-UG/D)* for more information.
Configurable Logic Blocks	8b/10b Decode Block, RxSONET Framer Block, 8b/10b Encode Block and TxSONET Framer Block. Refer to Figure 9 on page 82.	Each block is optimized to perform one function, and, while not fully programmable is configurable by the application. Examples of these configurable logic blocks are the SONET Framers and the 8b/10b Decode/Encode Blocks used for TBI protocols.

These programmable processors and configurable logic blocks are described in more detail in "RxSDP Detail Operations" on page 88 and "TxSDP Detail Operations" on page 93.

The SDP's processors *are microcode programmable, and SDP's* blocks *are configurable only.*

Each pipeline configuration uses FIFO blocks between certain pairs of processors. The FIFOs allow for some elasticity during relatively short periods of time when one SDP processor's throughput does not exactly match that of another that feeds it or is fed by it.

The run-time path for data traffic through this pipeline of SDP processors and blocks is configured by the application using higher-level calls to the C-Ware Communications Programming Interfaces (CPIs). This configuration takes place under application control as part of initializing the C-5e NP's ports. The available pipeline configurations are determined at the application level. Refer to the *C-Ware Reference Library* document in the *C-Ware Application Development Guide* for more information. These CPI calls configure the RxSDP and TxSDP by setting the appropriate bits in the *SDP_Mode3* (RxSDP) and *SDP_Mode5* (TxSDP) registers.

By distributing the SDP's tasks among a pipeline of embedded programmable processors and configurable logic blocks, the available processing achievable at wire speed can be quite high. For OC-12, Gigabit Ethernet, and FibreChannel, the C-5e NP's SDPs can be configured (via C-Ware CPIs) into aggregated channels that cooperatively achieve an even higher aggregate throughput.

A CP can also be configured to allow *recirculation* of data traffic from the TxSDP to the RxSDP. Refer to "Configuration for Recirculation Operations Using RxSDP and TxSDP" on page 98.

Common Components of the Programmable Processors

Refer to Table 14 on page 85 and Figure 10 on page 84 for the common components of the programmable processors (RxBit, RxSync, RxByte, TxBit and TxByte) inside the SDPs and their functions.

Figure 10 Common Components of Programmable Processors

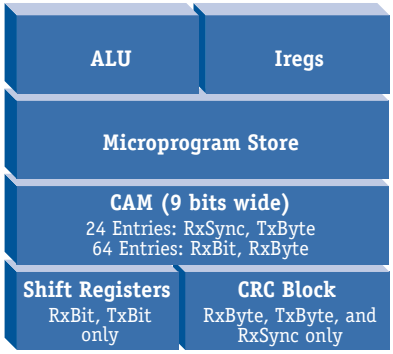


Table 14 Common Components of Programmable Processors and Their Functions

<table>
<tr><th>ITEM</th><th>FUNCTION</th></tr>
<tr><td>Arithmetic Logic Unit (ALU)</td><td>The eight-bit arithmetic and logic unit, where computations take place.</td></tr>
<tr><td>Internal Registers (Iregs)</td><td>Consists of sixteen (16) (Ireg0 to Ireg15) registers for the embedded processor's internal use:
Ireg0 to Ireg3 can serve as general-purpose registers but also can serve as counters (can be incremented directly by microcode). Ireg0 to Ireg3 can be incremented separately as eight-bit registers or as 16bit register pairs (Ireg0/Ireg1 and Ireg2/Ireg3). If Ireg0 to Ireg3 are used as pairs, Ireg1 is the least-significant half of its pair, as is Ireg3.
Ireg4 to Ireg9 are eight-bit, general-purpose registers.
Ireg10 to Ireg12 are currently not implemented.
Ireg13's two least significant bits (LSBs) provide access to the ninth bit of Ireg0 and of Ireg15.
Ireg14 is the Control/Status register.
Ireg15 is the microcode program counter.</td></tr>
<tr><td>Microprogram Store</td><td>Provides storage for the processor's microcode program. It comprises a series of 52bit microprogram words. Its characteristics are shown here:<table><tr><th>PROGRAMMABLE PROCESSORS</th><th>MICROPROGRAM STORE SIZE *</th></tr><tr><td>RxBit</td><td>64 words</td></tr><tr><td>RxSync</td><td>64 words</td></tr><tr><td>RxByte</td><td>512 words</td></tr><tr><td>TxBit</td><td>256 words</td></tr><tr><td>TxByte</td><td>384 words</td></tr></table>* 52bit word</td></tr>
</table>

Table 14 Common Components of Programmable Processors and Their Functions (continued)

<table>
<tr><th>ITEM</th><th>FUNCTION</th></tr>
<tr><td>Content Addressable Memory (CAM)</td><td>Provides a nine (9) bit wide lookup table. Its characteristics are shown here:
<table>
<tr><th>PROGRAMMABLE PROCESSORS</th><th>CAM</th></tr>
<tr><td>RxBit</td><td>64 entries x 9 bits wide</td></tr>
<tr><td>RxSync</td><td>24 entries x 9 bits wide</td></tr>
<tr><td>RxByte</td><td>64 entries x 9 bits wide</td></tr>
<tr><td>TxBit</td><td>None</td></tr>
<tr><td>TxByte</td><td>24 entries x 9 bits wide</td></tr>
</table>
</td></tr>
<tr><td>Shift Registers</td><td>Supports serial-to-parallel conversion.
Note: Only applies to RxBit and TxBit programmable processors.</td></tr>
</table>

Table 14 Common Components of Programmable Processors and Their Functions (continued)

<table>
<tr><th>ITEM</th><th>FUNCTION</th></tr>
<tr><td>Cyclic Redundancy Check (CRC)</td><td>Provides CRC checking, generation, and scrambling. Variations of CRC checking and generation are supported based on the SDP as detailed here:
<table>
<tr><th>PROGRAMMABLE PROCESSORS</th><th>CRC TYPES AVAILABLE</th></tr>
<tr><td>RxBit</td><td>None</td></tr>
<tr><td>RxSync</td><td>CRC-8 (ATM HEC), CRC-10 , CRC-16, CRC-32</td></tr>
<tr><td>RxByte</td><td>CRC-5, CRC-16, CRC-32</td></tr>
<tr><td>TxByte</td><td>CRC-5, CRC-8 (ATM HEC), CRC-10, CRC-16, CRC-32</td></tr>
<tr><td>TxBit</td><td>None</td></tr>
</table>
The generator polynomial (formula) for each supported CRC type is detailed here:
<table>
<tr><th>CRC TYPE</th><th>GENERATOR POLYNOMIAL (FORMULA)</th></tr>
<tr><td>CRC-5</td><td>$X^5 + X^2 + 1$</td></tr>
<tr><td>CRC-8 (ATM HEC)</td><td>$X^8 + X^2 + X + 1$</td></tr>
<tr><td>CRC-10</td><td>$X^{10} + X^9 + X^5 + X^4 + X + 1$</td></tr>
<tr><td>CRC-16</td><td>$X^{16} + X^{12} + X^5 + 1$</td></tr>
<tr><td>CRC-32</td><td>$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$</td></tr>
</table></td></tr>
</table>

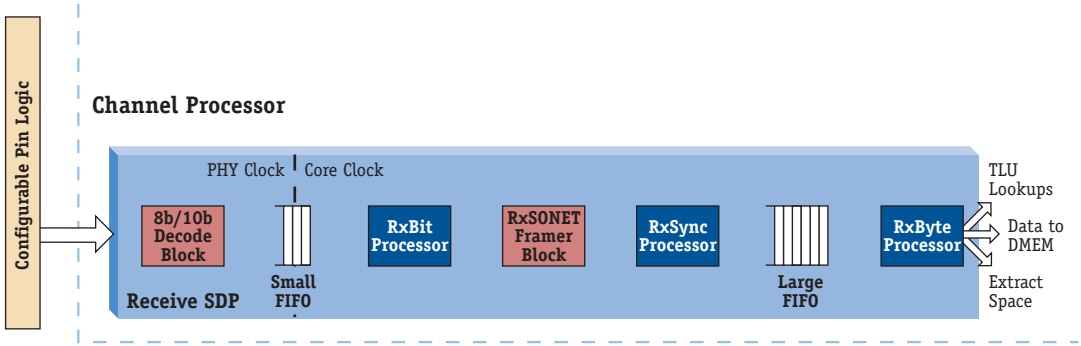
The CPRC restarts each SDP processor by toggling a reset signal. The microprogram stores and CAMs are visible to the CPRC and XP programs only for the purpose of loading these memories. The other registers and counters are not visible to CPRC and XP programs.

For information about the programmability of the SDP's embedded processors, see the *Microcode Programming Guide (part number CSTMCPG-UG/D)*.

RxSDP Detail Operations

This section provides more detail information regarding each RxSDP's programmable processors and configurable logic blocks. The individual items are shown in Figure 11 on page 88.

Figure 11 RxSDP Programmable Processors and Configurable Logic Blocks



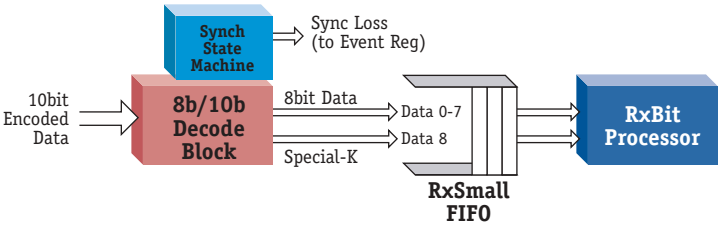
8b/10b Decode Configurable Logic Block

The 8b/10b Decode configurable logic block is located in the RxSDP.

This block contains hardware for encoding and decoding between 8-bit and 10-bit formats. This enables support for protocols that require 8b/10b encoding, such as FibreChannel or Gigabit Ethernet over the Ten Bit Interface (TBI).

As shown in Figure 12, in the RxSDP the 8b/10b Decode block decodes the 10-bit encoded data into its 8-bit equivalent, plus a Special-K signal, which is passed upstream as the ninth bit of the data stream. This bit is used during RxBit CAM lookups to differentiate between control and data characters. Using this decoded data stream, RxBit can be microprogrammed to perform frame delineation and to implement the Physical Coding Sublayer (PCS) state machine, defined in the IEEE 802.3 specification, section 36.

Figure 12 Operation of 8b/10b Decode Configurable Logic Block



Also in the 8b/10b Decode block, a special hardware component implements the synchronization state machine defined in the IEEE 802.3z specification, section 36.2.5.2.6 and in FibreChannel specifications. This component tracks whether comma code groups appear on odd or even positions in the incoming data stream to establish a state of synchronization, after which it tracks the occurrence of running disparity errors. After observing a defined density of disparity errors, the SyncLoss signal is asserted. This signal sets a bit in the *SONET_Event* register which then sets a bit in the *Event0* register that can be programmed to cause a CPRC interrupt, which allows the CPRC the opportunity to take corrective action, such as to initiate a re-establishment of the physical link.

RxSmallFIFO Configurable Logic Block

The RxSmallFIFO configurable logic block provides elasticity for the RxBit Processor's throughput requirements.

The RxSmallFIFO block also bridges the network clock domain with the C-5e NP's internal clock domain. The block has two (2) parts: the internal half's clock runs at the same frequency as the core clock, and the external half's clock runs at the same frequency (or a submultiple) as the external PHY's clock. This allows the input data stream to be converted to the core clock's higher frequency, thereby increasing all SDP throughput.

The RxSmallFIFO is sixteen (16) locations deep and ten (10) bits wide (that is, eight (8) data bits and two (2) control bits).

RxBit Programmable Processor

The RxBit programmable processor performs frame and cell delineation (including serial-to-parallel conversion), pattern matching, and field extraction on serial data up to nine (9) bits at a time. The extracted fields are written to the CP's Extract Space.

The payload output data is nine (9) bits wide.

The RxBit programmable processor can take specific actions in microcode in response to a particular pattern match. The RxBit programmable processor processes the data stream as a function of position. Pattern matches can include "don't cares." Extracted fields and status are written to programmable locations in the CP's Extract Space.

In a sense, the RxBit programmable processor is an intelligent serial-to-parallel converter, in that it is:

- Capable of detecting High-level Data Link Control (HDLC) frames and invalid sequences, as well as removing stuffed zeroes.
- Used to find the Synchronous Transport Signal (STS) frame in an OC-3c data stream.
- Capable of identifying and deleting the preamble of incoming Ethernet frames.
- Used for 1000BASE-X Gigabit Ethernet delimiter recognition.
- Used to implement the receive side of the 1000BASE-X physical convergence sub-layer of IEEE 802.3z.
- Used to delineate FibreChannel frames.

RxSONET Framer Configurable Logic Block

The RxSONET Framer configurable logic block obtains recovered receive clock frame sync (A1 and A2) and eight-bit (8) data from the physical layer interface chip or from the RxBit programmable processor. It descrambles the data, demultiplexes the transport overhead, checks (B1, B2) parity, and writes the overhead octets into the CP's register space. Refer to Figure 11 on page 88. Each STS frame has its own parity checker.

The RxSONET Framer also interprets the STS pointer in order to extract the Synchronous Payload Envelope (SPE). From the payload envelope, it demultiplexes the path overhead, checks (B3) parity, and writes the other overhead bytes to the CP's Extract Space. The remaining payload is passed downstream, which handles concatenated formats only.

The RxSONET Framer does no demultiplexing of the SPE payload. As such, it is only suitable for receiving frames or cells.

The RxSONET framer also provides support for detection and monitoring by software of various SONET/SDH defects such as Loss of Signal (LOS), Loss of Frame (LOF), Loss of Pointer (LOP), Path Remote Defect Indication (RDI-P), Line Alarm Indication Signal (AIS-L) to name just a few. Events of interest can be masked to enable either access via RC interrupt or polling.

RxSync Programmable Processor

RxSync is a general-purpose programmable processor that can be used to perform any generic processing to off-load either the RxBit or RxByte Processors. One example of this is synchronizing on ATM cells.

When configured for ATM, the RxSync programmable processor receives a byte data stream consisting of 53-byte ATM cells. It finds the cell boundaries by applying the Header Error Check (HEC) CRC sequentially to five-byte (5) segments, checking the first four (4) bytes against the fifth HEC byte. If the check fails, the cell delineator repeats the process. When it finds an application-defined number of successful HEC checks in a row, it enters the *in_sync* state. It remains in this state until there has been an application-defined number of consecutive HEC check failures, then it resumes the search.

After the data stream is synchronized, the RxSync programmable processor appends a status byte to the data stream so that the cell could be discarded if the HEC does not check.

The RxSync programmable processor can be programmed to parse the cell header, writing (Virtual Path Identifier (VPI), Virtual Channel Identifier (VCI)) payload_type and cell_loss_priority to locations in the CP's Extract Space.

When configured for Fast or Gigabit Ethernet, the RxSync programmable processor assists in 802.3x pause packet processing, passing the pause time up to the CPRC for processing.

The RxSync programmable processor is also capable of handling the HDLC byte escape sequence for Point-to-Point Protocol (PPP) over SONET.

RxLargeFIFO Configurable Logic Block

The RxLargeFIFO configurable logic block is located in front of the RxByte programmable processor and provides elasticity for the RxByte programmable processor, which typically has the greatest computational responsibility within the RxSDP. Therefore, this block can stage a cell while its VPI/VCI is being looked up by the C-5e NP's TLU.

RxByte Programmable Processor

The RxByte programmable processor performs pattern matching and field extraction. It also detects and parses the Ethernet and IP headers when acting as a pre-processor to the CPRC for switching and routing applications. It can extract fields and launch TLU lookups on fields, stream data to DMEM and to the CP's Extract Space, and has responsibility for interfacing with the CPRC.

The RxByte programmable processor can also perform CRC computations and checks and has the largest amounts of Content Addressable Memory (CAM) and microcode store space. It is expected that the RxByte microcode contains the majority of the application's RxSDP-resident custom functionality.

The RxByte programmable processor can take specific actions as a result of a particular pattern match on nine (9) bit words of data. The nine (9) bit match values are stored in a CAM, which is loaded along with the SDP's microcode. Refer to the *SDP Programming* document in the *C-Ware Application Development Guide* for details about CAMs.

The extracted fields are written to the CP's Extract Space, which is memory-mapped into the CP's Configuration Space. The RxByte programmable processor can also initiate lookups in the C-5e NP's TLU on the extracted fields via the Ring Bus interface.

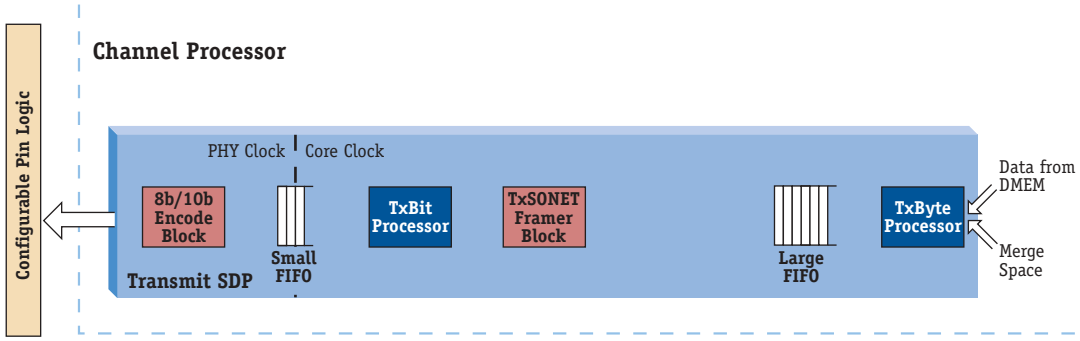
The receive data can pass through the RxByte programmable processor multiple times (via an idle port) by reading the processed data back from local memory and operating on it again via the SDP's recirculation path. Refer to "Configuration for Recirculation Operations Using RxSDP and TxSDP" on page 98 for details.

The RxByte programmable processor implements operations with CRC-16 and CRC-32 only. It can be programmed to check Ethernet and ATM Adaptation Layer 5 (AAL5) Frame Check Sequence (FCS). AAL5 CRC checking is performed with assistance from the TLU.

TxSDP Detail Operations

This section provides more detail information regarding each TxSDP's programmable processors and configurable logic blocks. The individual items are shown in Figure 13 on page 93.

Figure 13 TxSDP Programmable Processors and Configurable Logic Blocks



TxByte Programmable Processor

The TxByte programmable processor can be programmed to read data from DMEM, generate a valid CRC, and insert, delete, and replace fields in the egress traffic data stream. Like the RxByte programmable processor, it is expected that the TxByte microcode contains the majority of the application's TxSDP-resident custom functionality.

The transmit data can pass through the TxByte programmable processor multiple times (via an idle port) by writing the processed data back to local memory and operating on it again via the SDP's recirculation path. Refer to "Configuration for Recirculation Operations Using RxSDP and TxSDP" on page 98 for details.

TxLargeFIFO Configurable Logic Block and Options

The TxLargeFIFO configurable logic block is located after TxByte programmable processor and provides elasticity for the TxByte programmable processor, which typically has the greatest computational responsibility within the TxSDP. This block provides elasticity for field inserts and deletes by the TxByte programmable processor.

This FIFO is one-hundred-twenty-eight (128) locations deep and ten (10) bits wide (eight (8) bits of data and two (2) control bits). The second control bit is not accessible to the microcode and is used only for optional payload scrambling in OC-12 mode. The OC-12 data stream must consist of four (4) OC-3c streams.

In addition, the TxLargeFIFO allows two (2) selectable options: Automatic Idle Cell and PPP Flag Insertion, and Transmit FIFO High Water Mark as described here.

Automatic Idle Cell and PPP Flag Insertion Option

For the C-5e NP Version D0, the TxLargeFIFO supports automatically inserting ATM idle cells or PPP Flag characters into the output stream when needed.

For applications running on hardware prior to the C-5e NP Version D0, the TxByte microcode was required to perform this work. That microcode was conservative about when to send idle cells or PPP flags, to ensure that the TxLargeFIFO never emptied completely. This resulted in more idle cells or PPP flags being sent out than necessary, which in turn caused reduced bandwidth over the physical interface.

The new feature for the C-5e NP Version D0 sends idle cells or PPP flags only when the TxLargeFIFO becomes empty. Using this feature ensures that only the minimal number of idle cells or PPP flags are sent between packets or user data cells. It also allows TxLargeFIFO to be filled with real payload and not cluttered with idle characters.

To use the new feature, set the *Idle Insert Enable* bit in the *SDP_MODE4* register. For ATM applications, also set the *Idle Cell Mode* bit in the *SDP_MODE4* register. For PPP flag insertion, make sure that the *Idle Cell Mode* bit is clear. Refer to "SDP_Mode4 Register (CP Mode Configuration Function)" on page 536.

Transmit FIFO High Water Mark Option

Most applications process a protocol header in the SDP's TxByte serial processor prior to streaming payload data. The speed at which it processes a header is typically less than the transmit speed of the physical interface. If the header bytes were popped off the TxLargeFIFO at the rate of the physical interface while TxByte is sending them out at a rate less than that of the physical interface, an under-run condition can occur. An under-run causes corrupted data to be sent out on the interface when no payload data is available.

To work-around this problem prior to the C-5e NP Version C0, applications padded the TxLargeFIFO with enough flag bytes or idle cells to allow TxByte enough cycles to process a header. For all non-SONET applications, TxByte can signal TxBit when the data is ready to send at the physical interface speed. TxByte usually notifies TxBit at some point after it finishes processing the protocol header. In the meantime, TxBit sends out idle bytes to prevent under-running the physical interface.

In the C-5e NP Version D0 a *high water mark* feature has been added to eliminate the need to pad the TxLargeFIFO or coordinate with TxBit. The high water mark is a count of bytes that must be in the TxLargeFIFO for it to appear non-empty to the downstream SDP components that pop data out of it.

The high water mark value is set in the *SDP_MODE4* register. Setting this value to zero results in the same behavior as occurred in the C-5e NP Version C0 chip. Setting *SDP_MODE4* to a non-zero value causes the transmit FIFO to appear empty until the "high water mark" number of bytes are in the TxLargeFIFO. An application typically sets the high water mark value to the number of bytes in the protocol header that TxByte must process at less than the speed of the physical interface. Refer to "SDP_Mode4 Register (CP Mode Configuration Function)" on page 536.

The high water mark depth is tested at the end of each packet. The transmit FIFO uses the ninth bit as the end of packet or cell indication. If the TxLargeFIFO depth is less than the high water mark when the last byte of a packet or cell is unloaded "popped" from the TxLargeFIFO, the FIFO appears empty.

This feature can be used with the *automatic Idle Cell and PPP Flag insertion* logic. This logic inserts idle cells or PPP flags whenever a downstream SDP component unloads "pops" a byte from TxLargeFIFO and the FIFO appears empty.

TxSONET Framer Configurable Logic Block

The TxSONET Framer configurable logic block must receive data from TxLargeFIFO configurable logic block. It obtains most of the transport overhead and path overhead from the *TxSONETOH0* to *TxSONETOH31* registers. This allows the application developer to insert values into the transmit overhead. Refer to Figure 13 on page 93.

The transmit pointer value (bytes H1 and H2 in the SONET frame's Path Overhead section) is fixed.

The TxSONET Framer block generates B1, B2, and B3 bit-interleaved parity, then inserts the OC-12c overhead into the payload data and scrambles it to form either a SONET OC-3c, OC-12, or OC-12c format.

TxBit Programmable Processor

The TxBit programmable processor receives byte-wide data. Under microcode control it inserts, deletes, and replaces fields. Typically, the TxBit programmable processor provides the reverse functions of the RxBit programmable processor. TxBit contains a special-purpose shift register for performing parallel-to-serial conversion.

The TxBit programmable processor can impose minimum inter-frame gaps and monitor PHY status, for instance, in order to detect MAC collisions.

Input to the TxBit programmable processor is nine (9) bits wide and its output can be one, two, four, eight or ten (1, 2, 4, 8, or 10) bits at a time, depending on the type of physical interface.

The TxBit programmable processor can be viewed as an intelligent parallel-to-serial converter, because:

- The TxBit programmable processor modifies the data stream as a function of the data stream.
- The TxBit programmable processor is used for collision detection when configured for half-duplex 10Mbit/100Mbit Ethernet or Gigabit Ethernet.
- The TxBit programmable processor is used to implement the transmit side of the 1000BASE-X physical convergence sub-layer of IEEE 802.3z.

The TxBit programmable processor is capable of inserting the HDLC frame sequence, as well as stuffing zeros into the data stream as appropriate, under microcode control.

TxSmallFIFO Configurable Logic Block

The TxSmallFIFO configurable logic block provides elasticity for the TxBit programmable processor's throughput requirements.

The TxSmallFIFO configurable logic block also bridges the network clock domain with the C-5e NP's internal clock domain. The TxSmallFIFO configurable logic block has two (2) parts: the internal half's clock runs at the same frequency as the C-5e NP's core clock, and the external half's clock runs at the same frequency (or a submultiple) as the external physical layer's clock. This allows the outgoing data stream to be converted from the core clock's frequency to the PHY's clock frequency.

The TxSmallFIFO configurable logic block is sixteen (16) locations deep and ten (10) bits wide (that is, eight (8) data bits and two (2) control bits).

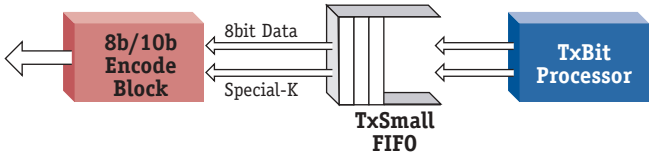
8b/10b Encode Configurable Logic Block

The 8b/10b Encode configurable logic block is located in the TxSDP.

This one (1) block contains hardware for encoding between 8bit and 10bit formats. This enables support for protocols that require 8b/10b encoding, such as FibreChannel or Gigabit Ethernet over the Ten Bit Interface (TBI).

As shown in Figure 14 on page 97, in the TxSDP the 8b/10b Encode block receives as input the 8bit data and Special-K bit (again, in the ninth bit position) and outputs the appropriate 10bit encoded value.

Figure 14 Operation of 8b/10b Encode Configurable Logic Block

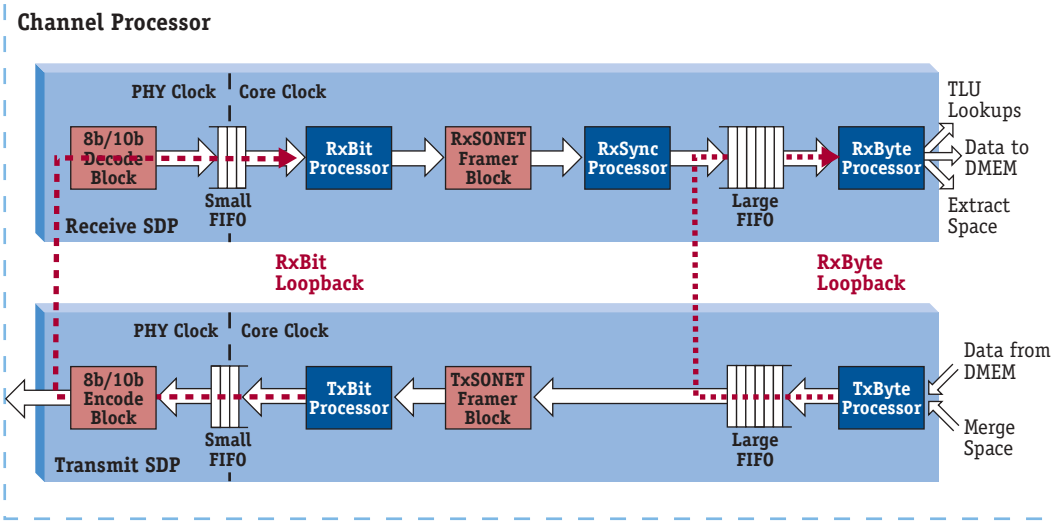


When transmitting idle code groups to the 8b/10b Encode block, the TxBit Processor should be programmed to transmit a series of (K28.5, D31.7). The D31.7 will be converted to the appropriate Dx.y value, either D5.6 or D16.2, depending on the block's internal odd/even state. This relieves TxBit from the burden of tracking the odd/even state of its output stream.

Configuration for Recirculation Operations Using RxSDP and TxSDP

Enabling *recirculation* for an SDP means to configure its RxSDP and TxSDP so that the output from the TxSDP processor is routed to the input of its corresponding RxSDP processor. This method is one way to pipeline your C-5e NP. The C-5e NP Pipeline Channel Mode, allows you to scale processing power for a particular application, the CPs can be linked for pipelined processing on a single data stream. Refer to Figure 15 on page 98.

Figure 15 SDP Recirculation Path Using Both RxBitLoopBack and RxByteLoopBack Bits



The SDP can be configured to permit recirculation of the data path in either of two (2) ways:

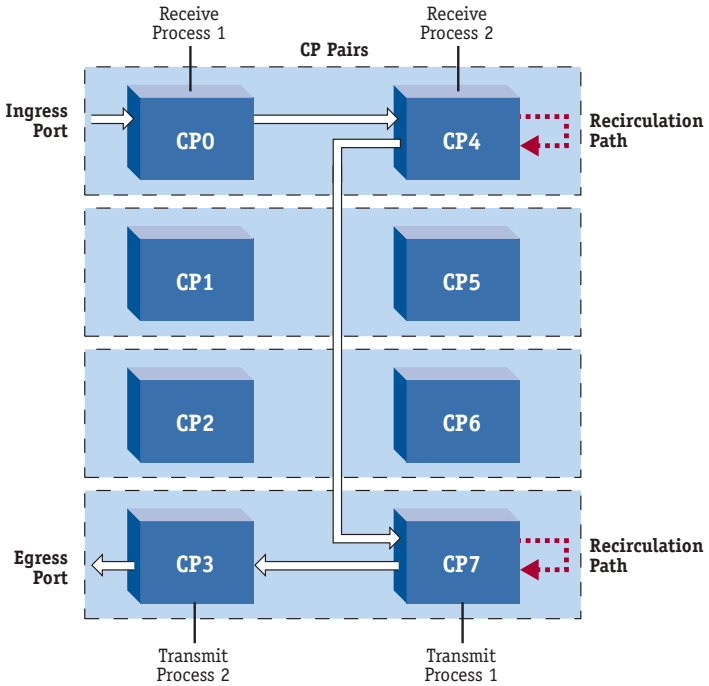
- From the TxByte programmable processor's output to the RxByte programmable processor's input, enabled by setting the *SDP_Mode3* register bit [25] *RxByteLoopBack* field.
- From the TxBit programmable processor's output to RxBit programmable processor's input, enabled by setting the *SDP_Mode3* register bit [24] *RxBitLoopBack* field.

Enabling recirculation for a CP's SDP can be useful in two (2) different ways: pipelining packet processing (during normal operations) and debugging and test.

- During normal operation, the ingress CP performs packet classification then enqueues the packet descriptor. If additional classification is desired, another CP with loopback set can dequeue the packet descriptor and process the packet again. The payload is moved from SDRAM through DMEM in the TxSDP. Using either the byte or bit

loopback, the data is brought back through the RxSDP for further classification. The second CP then enqueues the new packet descriptor for either actual transmit or for another stage of loopback processing for classification or forwarding decisions. Figure 16 on page 99 summarizes the data flow during normal operations. The recirculation operation can be used for any application that requires greater degrees of packet processing. For example, with applications such as multichannel HDLC and encryption that use T1 and T3 data rates.

Figure 16 Recirculation Shown for Normal Operations (for Cooperating CPs)



- SDP loopback can be used to debug the output of the TxSDP. Transmit data can be brought back on the chip for traffic analysis or chip test through the RxSDP without ever having to move data off chip.

Bit level recirculation for an SDP is not supported for CPs within the same cluster.

CP RISC (CPRC) Overview

Each of the sixteen (16) CPs has a Reduced Instruction Set Computer (RISC) Core. The dedicated CPRC in each channel orchestrates cell and packet processing. The CPRC operates at the C-5e NP's core clock rate.

The CPRC contains a 32bit data path and accesses memory using a 32bit physical address. Within the address space, a CPRC can reference its own local memory with zero-wait-state latency in the absence of remote contention from the Global Bus. Memory addresses outside of local memory range refer to remote memory space contained within other CPs and the Executive Processor (XP).

The CP contains memory-mapped, shared control registers used for CPRC-to-SDP communication. Refer to "CP Configuration Space" on page 112. The shared control registers also enable the CPRC to control the Payload and Ring Buses and enable the XP to configure the channel during initialization.

RISC Instruction Set Supported

The CPRC executes the following instruction set: a subset of MIPS™1 instruction set (excluding multiply, divide, floating point, unaligned loads and stores, move to hi and move to lo), eight (8) Branch Likely instructions from the standard MIPS™ 2 instruction set, and sixteen (16) custom instructions. Refer to "RISC Core Enhancements" on page 804. The CPRC supports the classes of instructions shown in Table 15 on page 101. The CPRC includes four (4) sets of 32 internal registers. Each set is associated with a CP's context. These internal registers are used to support *fast context switching*. These internal registers are defined in Table 16 on page 101.

The standard MIPS Coprocessor Zero (CP0) register are not supported. However, Freescale provides its own special purpose Coprocessor Zero registers. Refer to Table 17 on page 102.

It is highly recommended that you use the C-Ware Compiler when building your application code. Therefore, refer to the C-Ware Application Development Guide for information on using the Freescale compiler, which supports the CPRC.

Table 15 CPRC Supported Instruction Classes

CLASS OF INSTRUCTION	DESCRIPTION
Load and Store	Load immediate values and move data between memory and general registers.
Computational	Perform arithmetic and logical operations for values in registers.
Jump and Branch	Change program control flow.
Coprocessor Interface	Provide standard interfaces to the coprocessors.
Special	Perform miscellaneous tasks.

Table 16 CPRC (32) Internal Registers Definitions

CPRC INTERNAL REGISTER NAMES	SOFTWARE NAME	USE AND LINKAGE
$0	—	Always has the value of 0.
$at or $1	—	Reserved for the assembler.
$2:$3	v0 to v1	Used for expression evaluations and to hold integer function results. Also used to pass the static link when calling nested procedures.
$4:$7	a0 to a3	Used to pass the first four words of integer type actual arguments. Their values are not preserved across procedure calls.
$8:$15	t0 to t7	Temporary registers used for expression evaluations. Their values are not preserved across procedure calls.
$16:$23	s0 to s7	Saved registers. Their values must be preserved across procedure calls.
$24:$25	t8 to t9	Temporary registers used for expression evaluations. Their values are not preserved across procedure calls.
$26:$27 or $kt0:$kt1	k0 to k1	Used internally by the C-5e NP system services.
$28 or $gp	gp	Contains the global pointer.
$29 or $sp	sp	Contains the stack pointer.
$30	s8	A saved register (like s0 - s7).
$31	ra	Contains the return address used for expression evaluation.

See the *MIPSpro™ Assembly Language Programmer's Guide* for information about the standard MIPS™1 instruction set. It is available at http://www.mips.com/publications/index.html.

Table 17 Motorolas Coprocessor Zero Register Definitions

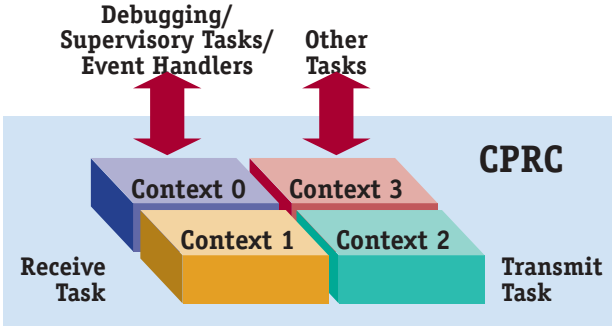
REGISTER	DEFINITION
R0	**Whoami Register** — Contains the DMEM base (hardcoded) for this CPRC.
R1	**Interrupt Table Register** — Contains the vector address for INT 0.
R2	**Break Table Register** — Contains the vector address for break 0.
R3	**Current Context Register** — The two LSBs are the current context register.
R4	**DMEM Comparison Address** — Contains the address at which debug pulse is generated.
R5	**DMEM Comparison Address Mask** — Contains the mask for the DMEM address.
R6	**DMEM Comparison Data** — Contains the data value for which debug pulse is generated.
R7	**DMEM Comparison Data Mask** — Contains the mask for the DMEM data.
R8	**Interrupt Flag** — The LSB in the Interrupt Flag.
R9	**Read/Write Mask** — The two LSBs are the Read mask and the Write mask for R4 to R7.

Fast Context Switching Configuration Using the CPRC

The CPRC and its memory architecture are optimized for the execution of real-time communication tasks, typically multiplexing processing between a number of different tasks (transmit and receive, at a minimum). The CPRC may be configured to incorporate a fast, four-way, context switching feature that replicates the entire CPRC register space four (4) times and can switch from one register set (one context) to another under software control or hardware interrupt.

Thus, actual processing (as opposed to manually saving the contents of one set of registers and then loading another) can begin on a different context in only two cycles. Therefore, these four (4) contexts can be used for debugging, supervisory tasks, event handlers, or other tasks.

Figure 17 CP Context Switching Feature Block Diagram



Fast Context Switching Detail Operations

Using the internal registers, context switching is accomplished two (2) ways:

1 Coprocessor instruction (software):

The software mechanism for executing a context switch is the Freescale Coprocessor Zero instructions. Refer to Table 17 on page 102.

- MTC0 $1 $3
- where $1 specifies the destination context, and where $3 is the source or current context. The contexts have no priority; how they are used is entirely designated by software.

2 Interrupt (hardware):

The hardware interrupt sequence is:

- All interrupts are disabled until an Restore From Exception (RFE) instruction is executed.
- The address of the next instruction to be executed in the interrupted context is saved in K1. Refer to "Interrupts" on page 104.
- Program execution continues with the instruction at the address specified in the interrupt vector.

Interrupts

The CPRC supports four (4) prioritized hardware interrupts, that can be triggered from any bits in the *Event0/Event1* Registers. There are four (4) MIPS-like register sets corresponding to each hardware context, one (1) register of which (K0) is shared by the other contexts.

K1 contains the program counter value and the context number of the interrupted context. These values are used in the execution of the RFE instruction to return to the previously interrupted context.

All interrupts and exceptions transfer control to a location found in the appropriate interrupt or break table. The base address of the interrupt table is specified by the contents of the interrupt table register ($1) in coprocessor zero. The base address of the exception table is specified by the contents of the break table register ($2) in coprocessor zero.

Interrupts are dispatched by a jump to the address equal to ((interrupt number * 8) + (interrupt table register)). Exceptions are dispatched by a jump to the address equal to ((break number * 8) + (break table register)). In addition to the jump, the register context is set to zero and interrupts are disabled. However, exceptions may still occur. Whether a hardware interrupt or an exception, the interrupted routine's register context and its next program counter are saved in K1 of context zero.

The K1 value points at the next instruction to be executed after the interrupt is serviced. RFE is normally used to: (1) resume the instruction flows at this point, (2) restore the proper register context, and (3) restore the Interrupt Enable Flag (IEF) to its value at the time of the interrupt or exception.

Interrupts are not recognized in a branch delay slot. Also, all exceptions fill the delay slot following a change of flow with a NOP instruction.

Interrupts are enabled by setting the IEF which is the LSB of coprocessor zero, Register 8, (Interrupt Flag). Refer to Table 17 on page 102. The IEF is preserved whenever an exception or an interrupt occurs and is restored by the RFE instruction.

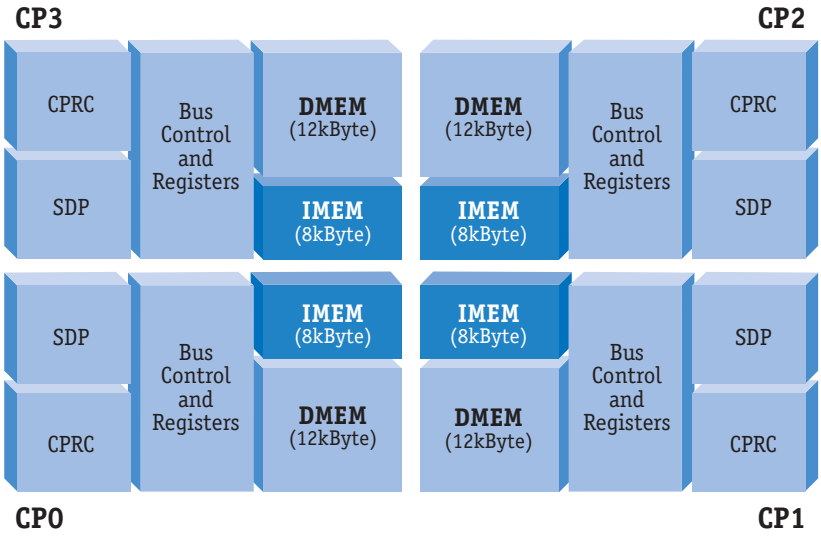
CP Memory (IMEM and DMEM)

The CP has local instruction memory (IMEM) and data memory (DMEM).

Instruction Memory (IMEM)

Each CP shares access to a 32kByte IMEM among a cluster of four (4) adjacent CPs, as shown in Figure 18 on page 105. The IMEM is configured as four (4) sub-arrays, with each CP in the cluster given access to the arrays, one per cycle, in fixed round-robin order. With this simple interleaved scheme, the four (4) adjacent CPRCs can access this memory at nearly full bandwidth.

Figure 18 Local and Shared Memory in a Channel Processor



When adjacent channels are configured to handle similar communication protocols, the large shared memory can contain both CP-specific code and cluster-shared code (such as exception routines).

At initialization time, the 32kByte array can be divided so that each CP gets a dedicated 8kByte sub-array. This array allocation removes all CP contention for IMEM (and also removes the ability to share code among CPs).

CPRC instruction execution outside of the shared local memory space is not supported.

Data Memory (DMEM)

Each CPRC in a cluster has a local 12kByte DMEM and can access the local DMEM of any other CPRC in the cluster with one additional cycle of latency.

The local 12kByte DMEM is organized as 16Byte lines providing 3.2GBps peak bandwidth through a single port. The memory resides in the global address space of the C-5e NP. Local CPRC and Global Bus references use a 4Byte (32bit) access path with zero-wait states in the absence of contention.

The SDP assembles payload data into 16Byte lines and writes it into local DMEM for receive cell/packet processing. For transmit, the SDP reads bytes of payload data from a 16Byte line buffer that is filled from DMEM using a single-cycle, 16Byte access. The Payload Bus controller moves buffers to and from SDRAM through this memory in 64Byte bursts comprised of four (4) consecutive 16Byte accesses.

The SDP and payload transactions have priority over CPRC transactions and use predetermined slots to access DMEM; this provides predictable bandwidth and latency and eliminates the need for extra data buffering. Global references and local CPRC references contend for unused DMEM access slots.

CP Memory Interface Transactions

The CP memory interface transactions are described in Table 18 on page 107.

Table 18 CP Memory Interface Transactions

TRANSACTION TYPE	MEMORY TYPE	DESCRIPTION
Payload Buffer Write	DMEM to SDRAM	The CPRC sets up a Write Control Block (*WrCB0* or *WrCB1*) or Receive Control Block (*RxCB0* or *RxCB1*) and clears the *Avail* bit to cause the bus controller to arbitrate for a payload write. When grant is acquired, the controller transfers 64Bytes in a four-cycle, 16Byte-per-cycle burst from local DMEM directly onto the Payload Bus. Transmission is successful if the receiver acknowledges (ACKs). Otherwise, the bus controller can retry or terminate depending on the programmable controller configuration.
Payload Buffer Read	SDRAM to DMEM	The CPRC sets up a Transmit Control Block (*TxCB0* or *TxCB1*) or Read Control Block (*RdCB0* or *RdCB1*) and clears the *Avail* bit to cause the bus controller to arbitrate for a payload read. When grant is acquired, the controller transfers the read address and makes the request. Transmission is successful if the receiver ACKs. Otherwise, the bus controller can retry or terminate depending on the programmable controller configuration. If the memory controller or queue controller accepts the request, it accesses SDRAM and returns the requested data. Access to DMEM is guaranteed; no acknowledgment is required. The bus controller bus moves 64Bytes in a four-cycle, 16Byte-per-cycle burst directly into the DMEM in consecutive cycles.
Rx SDP Byte Process	External to DMEM	The CPRC sets up a Receive Control Block (*RxCB0* or *RxCB1*) to control the SDP RxByte Processor. When the accumulation buffer fills with byte writes from the RxByte Processor, the 16Byte line is written into the DMEM at the address in the *RxCB0_SDP_Addr register bits [13:0] ByteAddr* field using the next guaranteed Receive access time to DMEM.
Tx SDP Byte Process	DMEM to External	The CPRC sets up a Transmit Control Block (*TxCB0* or *TxCB1*) to control the SDP TxByte Processor. When the TxByte Processor requests a byte read, a 16Byte line buffer is filled from DMEM at the address in the *RxCB0_SDP_Addr* register. Subsequent byte reads from the SDP are serviced from the line buffer. DMEM access uses the next guaranteed Tx access time to DMEM.

Table 18 CP Memory Interface Transactions (continued)

TRANSACTION TYPE	MEMORY TYPE	DESCRIPTION
CPRC Read/Write	DMEM	The CPRC uses word, half-word, and byte loads and stores to access the local DMEM cluster. Local DMEM access for transmit and receive Direct Memory Access (DMA) transactions is guaranteed and takes top priority. CPRC memory references falling within the local DMEM address space receive single-cycle access if memory is available. CPRC memory references falling outside the local DMEM but within the DMEM cluster, take a cycle to arbitrate for the desired DMEM array. When other CPRCs in the cluster have requested a DMEM array, the local CPRC participates in the arbitration. The arbitration scheme ensures that cluster accesses are serviced within the next four cycles that are free of local transmit and receive DMA.
CPRC Read/Write	Global Space	The CPRC uses 32bit word loads and stores to access global memory space. Load operations outside of the cluster DMEM space cause the bus controller to arbitrate for a global transaction. Upon acquisition of grant, the controller drives out the address and request. Transmission of the request is successful if the receiver ACKs. Otherwise, the controller can retry or terminate depending on the programmable controller configuration. Later, the receiver drives back the request data. The CPRC stalls until the load data arrives, so there can only be a single load to global space outstanding per CPRC. Store operations to global memory space dumps address and data into a write buffer in the bus controller. If the write buffer is full, the CPRC process stalls, otherwise the CPRC process continues. A valid write buffer entry causes the bus controller to arbitrate for the global bus. When grant is acquired, the controller drives out address and data. Transmission is successful if the receiver ACKs. Otherwise, the controller can retry or terminate depending on the programmable controller configuration. Since these transactions do not involve the local arrays, DMEM DMA can take place underneath.
CPRC Instruction Fetch	IMEM	Instruction fetch is always local to cluster IMEM. Note: Global memory addresses are not allowed.
Read	IMEM	The read uses the (lwc2) instruction.
Write	IMEM	The write uses the (swc2) instruction.

Table 18 CP Memory Interface Transactions (continued)

TRANSACTION TYPE	MEMORY TYPE	DESCRIPTION
Global Bus Read/Write	DMEM	Global Bus transactions are 32bit word length. From the point-of-view of the target receiving a CPRC Global Memory Space Read/Write, when the bus controller decodes a global read targeted at its local DMEM, it loads a read address latch and either sends back an ACK, if successful, or non-acknowledge (NACK) if the latch is full. The controller arbitrates for DMEM along with cluster DMEM requests. When granted, the controller reads the requested data of the DMEM array into a latch, then arbitrates for the bus. When the bus access is granted, the read data is returned to the requester which must ACK. When the bus control indicates a global write targeted at the local DMEM, the bus controller loads a write address and data latch and either sends back an ACK, if successful, or NACK if the latch is full. The controller arbitrates for DMEM taking the next available cycle to write the data into the array.
CPRC Read/Write	Configuration Space	Global configuration registers maintained on a per CP basis are addressed in global memory space. The CPRC reads and writes the registers over its 32bit data bus using word, half-word, and byte load and store operations. Access is guaranteed since these transactions do not involve the local arrays.
Global Read/Write	Configuration Space	Global access of configuration registers follows the same timing as global access of DMEM.

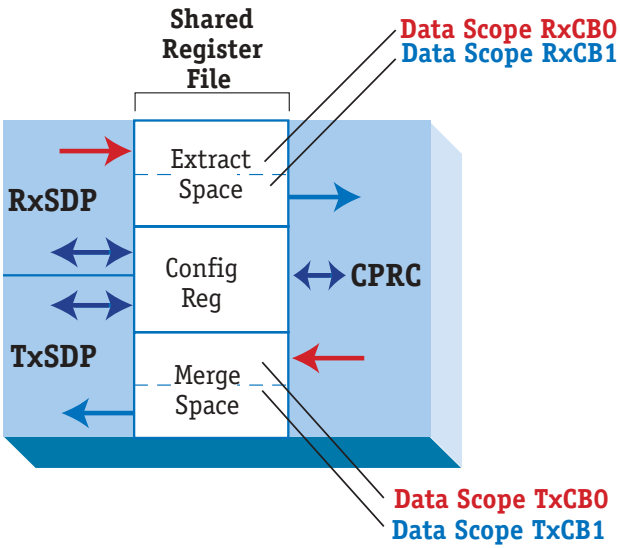
DataScope Purpose

The CP architecture provides access to two (2) sets of packet headers and data fields (*Datascope0* and *Datascope1*) to enable a unique feature called *data scoping*. Data scoping allows overlap of CPRC processing tasks while receiving and transmitting packets. Each data scope provides the CPRC with a coherent view of an individual packet on reception or transmission, including DMA parameters, *Extract* or *Merge* registers, and table lookup results.

The exact contents of the *Extract* and *Merge* registers are determined by microcode. Refer to the *C-Ware Microcode Programming Guide (part number CSTMCPG-UG/D) for details.*

Data scopes also eliminate the need for the CPRC program itself to manage the coherency of these disparate operations, allowing the construction of a simple, efficient two-stage software pipeline model. There are a total of four (4) data scopes available, two (2) for receive (Receive Control Blocks *RxCB0* and *RxCB1*) and two (2) for transmit (Transmit Control Blocks *TxCB0* and *TxCB1* registers). A diagram of a CP depicting the receive and transmit data scopes is shown in Figure 19 on page 110.

Figure 19 Four (4) Data Scopes Between the CPRC and SDPs



A hardware *receive* data scope is made up of Extract Space, an SDP Receive status register, and a Receive Control Block (*RxCB*). A hardware *transmit* data scope is made up of Merge Space, an SDP Transmit status register, and a Transmit Control Block (*TxCB*).

Data Scope Detail Operations

Initially, the RxSDP brings in payload, extracts fields and writes them to Extract Space, and launches table lookups on the Ring Bus in *Datascope0*. Hardware directs SDP DMEM writes to *RxCB0*, Extract Space writes to *RxSDP0_Ext0* to *RxSDP0_Ext31*, and status updates to *RxCtl0_Status*. Subsequently, the RxSDP signals that it has finished processing a cell/packet, triggering the hardware to switch to *Datascope1*.

Then the hardware directs SDP DMEM writes to *RxCB1*, Extract Space writes to *RxSDP1_Ext0* to *RxSDP1_Ext31*, and status updates to *RxCtl1_Status*. At the end of this cell/packet, hardware switches back to scope 0. The SDP is required to test the status bits in RxStatus to be sure the new scope is ready before processing the next cell/packet.

The CPRC must monitor both *RxCtln_Status* registers to track SDP processing. After the SDP finishes a scope, the CPRC must:

- Examine and possibly remove relevant data from the associated Extract Space.
- Examine the *RxCBn* to confirm that the payload DMA finished and reprogram the *RxCBn* if necessary.
- Examine and possibly remove relevant data from the Ring Bus response registers and reset the ownership bits.
- Update the *RxCtln_Status* to make the scope available to the SDP.

The TxSDP and CPRC operate in a similar manner to transmit the datascopes.

CP Configuration Space

Each CP has memory-mapped Configuration Space that contains a number of registers. Refer to Table 19 on page 113 for a list of CP registers by function. The CPRC uses these registers to communicate with the SDP, the bus controllers, and the XP.

Address Mapping of the CPs

Since the CP configuration space is duplicated for each CP (CP0 to CP15), the address listed in the memory maps and register descriptions begins with 0x*BC***n***0* where n should be replaced with the appropriate offset for the particular CP. Refer to Chapter 1 for addressing details.

The Configuration Space provides a 1MB block or segment of Configuration Space for each CP. Specific registers are located at offsets from each CP's Configuration Space base address.

Figure 20 CP Configuration Space Memory Map

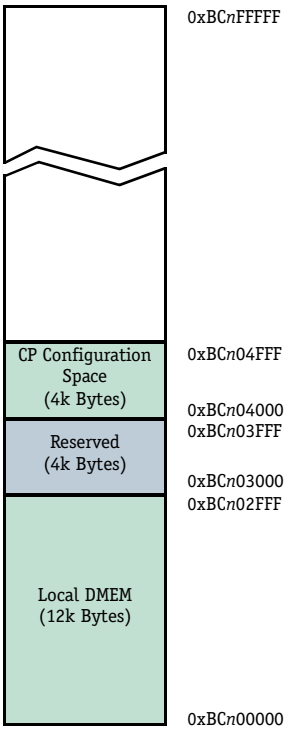


Table 19 CP Registers by Function

CP FUNCTION	SPECIFIC REGISTERS
DMEM	See "Data Memory (DMEM)" on page 106
Rx Extract	RxSDP0_Ext0 to RxSDP0_Ext15 RxSDP1_Ext0 to RxSDP1_Ext15
Rx Control Blocks	RxCB0_Sys_Addr, RxCB0_Ctl, RxCB0_DMA_Addr, RxCB0_SDP_Addr, RxCB0_Status, RxCB1_Sys_Addr, RxCB1_Ctl, RxCB1_DMA_Addr, RxCB1_SDP_Addr, RxCB1_Status
Tx Merge	TxSDP0_Merge0 to TxSDP0_Merge15 TxSDP1_Merge0 to TxSDP1_Merge15
Tx Control Blocks	TxCB0_Sys_Addr, TxCB0_Ctl, TxCB0_DMA_Addr, TxCB0_SDP_Addr, TxCB0_Status, TxCB1_Sys_Addr, TxCB1_Ctl, TxCB1_DMA_Addr, TxCB1_SDP_Addr, TxCB1_Status
Wr Control Blocks	WrCB0_Sys_Addr, WrCB0_Ctl, WrCB0_DMA_Addr, WrCB1_Sys_Addr, WrCB1_Ctl, WrCB1_DMA_Addr
Rd Control Blocks	RdCB0_Sys_Addr, RdCB0_Ctl, RdCB0_DMA_Addr, RdCB1_Sys_Addr, RdCB1_Ctl, RdCB1_DMA_Addr
Ring Bus Tx Message Control	TxMsg0_Ctl, TxMsg1_Ctl, TxMsg2_Ctl, TxMsg3_Ctl, TxMsg0_Data_H, TxMsg0_Data_L, TxMsg1_Data_H, TxMsg1_Data_L, TxMsg2_Data_H, TxMsg2_Data_L, TxMsg3_Data_H, TxMsg3_Data_L
Ring Bus Rx Response Control	RxResp0_Ctl, RxResp1_Ctl, RxResp2_Ctl, RxResp3_Ctl, RxResp4_Ctl, RxResp5_Ctl, RxResp6_Ctl, RxResp7_Ctl, RxResp0_DataH, RxResp0_DataL, RxResp1_DataH, RxResp1_DataL, RxResp2_DataH, RxResp2_DataL, RxResp3_DataH, RxResp3_DataL, RxResp4_DataH, RxResp4_DataL, RxResp5_DataH, RxResp5_DataL, RxResp6_DataH, RxResp6_DataL, RxResp7_DataH, RxResp7_DataL
Ring Bus Rx Message Control	RxMsg_Ctl, RxMsg_FIFO
SONET Rx Control	Rx_SONETH0 to Rx_SONETH31
SONET Tx Control	Tx_SONETH0 to Tx_SONETH31
SDP Rx Control	RxCtl_ByteSeq0, RxCtl_ByteSeq1, RxCtl_SyncSeq, RxCtl_BitSeq0, RxCtl_BitSeq1
SDP Tx Control	TxCtl_ByteSeq0, TxCtl_ByteSeq1, TxCtl_BitSeq0, TxCtl_BitSeq1

Table 19 CP Registers by Function (continued)

CP FUNCTION	SPECIFIC REGISTERS
CP Mode Configuration	CP_Mode0, CP_Mode1, SDP_Mode2, SDP_Mode3, SDP_Mode4, SDP_Mode5, Debug_Mode, PIN_Mode
Queue Status	Queue_Status0, Queue_Status1, Queue_Status2, Queue_Status3, Queue_Update0, Queue_Update1, Queue_Update2, Queue_Update3
Miscellaneous	Event_Timer, Cycle_Counter_H, Cycle_Counter_L
Event and Interrupt	Event0, Event0, Event_Mask1, Event_Mask1, Event_Access, Mask_Access, Interrupt_Mask0, Interrupt_Mask1, SONET_Event, SONET_Mask

For complete details about specific registers go to their reference. Refer to "CP Registers" on page 486.

Understanding CP Functions

The following is a discussion of the CP functions and the registers associated with each function.

Extract Space

Configuration Space contains 64Bytes of Extract Space per datascope (Datascope0/Datascope1) for passing fields extracted from the receive data stream (by the SDP RxByte programmable processor) to the CPRC. The RxByte programmable processor performs byte-wide write operations to the Extract Space by specifying the configuration register destination commands in microcode.

The RxByte programmable processor cannot read the Extract Space registers.

The CPRC accesses the memory-mapped Extract Space using load and store instructions. The CPRC can write to the Extract Space registers, but only during initialization and test periods when the *SDP_Mode3* register bit [30] *RxEnable* field is cleared.

The data format for the Extract Space is defined by agreement between the CPRC program and the RxByte programmable processor microcode. Refer to Table 20 on page 115 for Extract Space registers.

Table 20 Extract Space Registers

REGISTER NAME	PURPOSE	ADDRESS	DETAILS
RxSDP0_Ext0 to RxSDP0_Ext15	Used to pass fields extracted from the receive data stream by the RxSDP to the CPRC. These registers are used only for receive data scope0.	0xBC*n*04000 to 0xBC*n*0403C	See "RxSDP0_Ext0 to RxSDP0_Ext15 Registers (CP Rx Extract Space0 Function)" on page 492
RxSDP1_Ext0 to RxSDP1_Ext15	Same as registers *RxSDP0_Ext0* to *RxSDP0_Ext15*, except for data scope1.	0xBC*n*04200 to 0xBC*n*0423C	See "RxSDP1_Ext0 to RxSDP1_Ext15 Registers (for Datascope1)" on page 492

Merge Space

Configuration Space contains 64Bytes of Merge Space per datascope (Datascope0/Datascope1) for passing fields from the CPRC to the SDP TxByte programmable processor to merge in with the transmit data stream. The CPRC accesses the memory-mapped Merge Space using load and store instructions. The TxByte programmable processor performs byte-wide read operations from the Merge Space by specifying the configuration register source in microcode. The data format for the Merge Space registers is defined by the CPRC process and the SDP firmware. Refer to Table 21 on page 115 for Merge Space registers.

The TxByte programmable processor cannot write to the Merge Space registers.

Table 21 Merge Space Registers

REGISTER NAME	PURPOSE	ADDRESS	DETAILS
TxSDP0_Merge0 to TxSDP0_Merge15	Used to pass fields from the CPRC to the TxSDP to merge in with the transmit data stream. These registers are used only for transmit data scope0.	0xBC*n*04100 to 0xBC*n*0413C	See "TxSDP0_Merge0 to TxSDP0_Merge15 Registers (CP Tx Merge Space0 Function)" on page 492
TxSDP1_Merge0 to TxSDP1_Merge15	Same as registers *TxSDP0_Merge0* to *TxSDP0_Merge15*, except for data scope1.	0xBC*n*04300 to 0xBC*n*0433C	See "TxSDP1_Merge0 to TxSDP1_Merge15 Registers (for Datascope1)" on page 493

Control Block Registers

Configuration Space includes eight (8) sets of control registers called Control Blocks (CBs) that govern DMA operations to and from DMEM. The CPRC sets up the control registers to perform four (4) types of DMEM DMA operations:

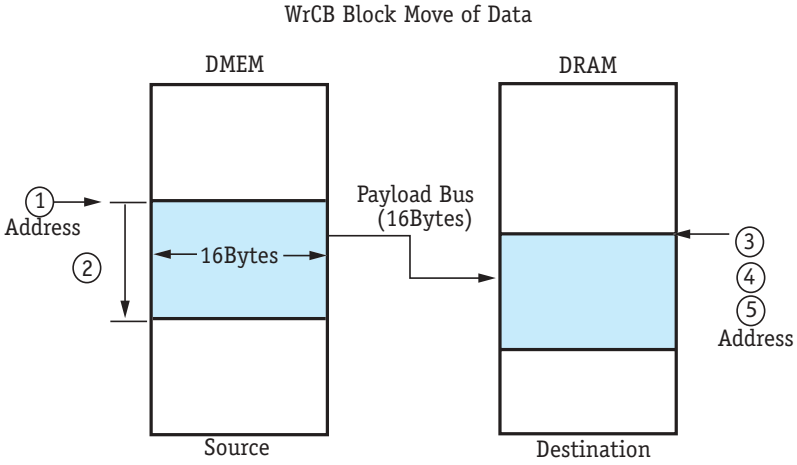
- Write Control Block (WrCB0_ , WrCB1_)
- Read Control Block (RdCB0_ , RdCB1_)
- SDP RxByte Processor Receive Control Block (RxCB0_ , RxCB1_)
- SDP TxByte Processor Transmit Control Block (TxCB0_ , TxCB1_)

Write Control Blocks (WrCB0_ , WrCB1_)

Two Write Control Blocks (WrCB0_ and WrCB1_) provide the capability for general purpose write tasks, such as Buffer Transfers, QMU enqueues and BTag writes. These tasks are DMA operations of programmable length that move data from DMEM over the Payload Bus in bursts of four (4) cycles with 16 bytes per burst.

Figure 21 on page 117 shows a Buffer Transfer. In general, data is moved from DMEM (the source) to SDRAM (the destination). Specifically, moving data starting at the *LineAddr* (1) location inside DMEM with a *Length* (2) to SDRAM beginning at the (*PoolID, BTag* and *Offset* (3,4,5)) location. These individual fields that are used to set up the details of the block move make up parts of these registers: WrCB*n*_Sys_Addr, WrCB*n*_Ctl and WrCB*n*_DMA_Addr.

Figure 21 DMA Operation (Buffer Transfer) Using WrCB*n*_ Registers



Register	Fields	
WrCBn_Sys_Addr	BTag ④	Offset ⑤
WrCBn_Ctl		Length ②
WrCBn_DMA_Addr	PoolID ③	LineAddr ①

Buffer Transfer Setup Using WrCBn_Sys_Addr, WrCBn_Ctl and WrCBn_DMA_Addr:
To set up this single contiguous data transfer, the CPRC writes a system address and a line address for DMEM to the WrCB. The length is written by the CPRC to the desired transfer length.

To perform Buffer Transfers involves setting the bits for WrCB0_Sys_Addr (0xBC*n*04400), WrCB0_Ctl (0xBC*n*04404) and WrCB0_DMA_Addr (0xBC*n*04408).

To set up a general contiguous data transfer, the CPRC process must do the following:

1 Ensure that the WrCB0 is available by testing that *WrCB0_Ctl* bit [31] *Avail* field=1.

2 Write *WrCB0_Sys_Addr* with the system address to be written in the form of:
WrCB0_Sys_Addr bits [31:16] *BTag* field = BTag, and
WrCB0_Sys_Addr bits [15:4] *Offset* field = Offset
(Offset is a 16Byte starting buffer offset, typically equal to 0, or aligned to a 64Byte boundary).

3 Write the *Pool ID* portion of the system address into *WrCB0_DMA_Addr* bits [20:16] *PoolID* field. Write *WrCB0_DMA_Addr* bits [13:4] *LineAddr* field with the location of a buffer in DMEM, typically aligned on a 64Byte boundary. This is the location that the DMA engine uses to begin transferring data out of DMEM to SDRAM.

4 Write *WrCB0_Ctl* with *WrCB0_Ctl* bits [13:4] *Length* field equal to the number of bytes to be transferred, and *WrCB0_Ctl* bit [31] *Avail* field equal to 0, and *WrCB0_Ctl* bit [29] *Modulo64* equal to 0 to cause the *WrCB0_Sys_Addr* bits [15:4] *Offset* field and *WrCB0_DMA_Addr* bits [13:4] *LineAddr* field to increment during the DMA for a contiguous block transfer.

For complete details about specific registers go to their reference. Refer to: "WrCB0_Sys_ Addr Register (CP Wr Control Block0 Function)" on page 499, "WrCB0_Ctl Register (CP Wr Control Block0 Function)" on page 500, and "WrCB0_DMA_Addr Register (CP Wr Control Block0 Function)" on page 501. You also have the following registers available for Control Block1: WrCB1_Sys_Addr, WrCB1_Ctl and WrCB1_DMA_Addr that function in the same manner.

Buffer Transfer Operations Using WrCBn_Sys_Addr, WrCBn_Ctl and WrCBn_DMA_Addr: An availability bit indicates DMA or CPRC control of the block of DMEM. When set, the CPRC controls the block; when clear, the DMA engine controls the block and is free to transfer out of it.

Clearing the availability bit, *WrCB0_Ctl* bit [31] *Avail* field, the CPRC process initiates a data transfer from DMEM beginning at the line addressed by *WrCB0_DMA_Addr* bits [13:4] *LineAddr* field to SDRAM beginning at the system address in *WrCB0_Sys_Addr*. The DMA engine transfers payload out of DMEM in bursts, decrementing *WrCB0_Ctl* bits [13:0] *Length* field by 64Bytes for each burst. Transfer continues until the *WrCB0_Ctl* bits [13:0] *Length* field equals 0 (at which time the DMA engine sets *WrCB_Ctl* bit [31] *Avail* field, thus returning control of the WrCB back to the CPRC process).

Initiating transfers with *WrCB0_Ctl* bits [13:0] *Length* field equal to 0 causes a single 64Byte transfer. If a 4-cycle, 64Byte transfer is started with *WrCB0_Ctl* bits [13:0] *Length* field less than 64 bytes, only the number of 16Byte lines needed to transmit the whole length actually get written into SDRAM and the *Length* field is set to 0 after the burst.

If the *WrCB0_Sys_Addr* bits [15:4] *Offset* field is aligned to a 64Byte boundary, a contiguous 64Byte block of DRAM is written. If the *WrCB0_Sys_Addr* bits [15:4] *Offset* is 64Byte unaligned, the DRAM block is written in a wrapped fashion which is typically not useful. Typically, contiguous transfers start with aligned offsets.

WrCB0_DMA_Addr bits [13:4] *LineAddr* field increments for each of the four 16Byte-per-cycle transfers to provide an address into DMEM. Typically, *WrCB0_DMA_Addr* bits [13:4] *LineAddr* field starts at a 64Byte aligned address. *WrCB0_DMA_Addr* bits [13:4] *LineAddr* field can start unaligned, but the resulting wrap behavior is not useful. Transfers with *Offset* unaligned make the most sense if the *Length* and *Offset* fields are set so that the resulting SDRAM completes a block of SDRAM, but does not wrap. For example, if *WrCB0_Sys_Addr* bits [15:4] *Offset* field= 0x0010 and *WrCB0_Ctl* bits [13:4] *Length* field= 0x0030, then the DMA moves three 16Byte lines from DMEM[1:3] to SDRAM[1:3]. The burst wraps to DMEM[0] and SDRAM[0], but the write is inhibited. If *WrCB0_Ctl* bit [29] *Modulo64* field equal to 0, *WrCB0_DMA_Addr* bits [13:4] *LineAddr field* and *WrCB0_Sys_Addr* bits [15:4] *Offset* field increment by 64 for each 64Byte burst.

Initiating transfers with the modulo64, W*rCB0_Ctl* bit [29] *Modulo64* field, equal to 1prevents an update of the *WrCB0_Sys_Addr* bits [15:4] *Offset* field and causes *WrCB0_DMA_Addr* bits [13:4] *LineAddr* field to increment modulo 64Bytes, effectively returning *WrCB0_DMA_Addr* bits [13:4] *LineAddr* field to wrap back to the starting value after a 4-cycle burst. This feature is useful for writes to the QMU or BMU when the system address contains a command, not an address. The WrCB can be used again without resetting the *WrCB0_Sys_Addr* bits [15:4] *Offset* field and *WrCB0_DMA_Addr* bits [13:4] *LineAddr* field.

The CPRC process can read the state of the DMA machine from *WrCB0_Ctl* bits [17:16] *State* field. The *WrCB0_Ctl* word generally should not be written by the CPRC process when hardware is operating (that is, *WrCB_Ctl* bit [31] *Avail* field= 0). On a write to *WrCB0_Ctl*, bits [17:16] *State* field is only updated if bit [31] *Avail* field =1.

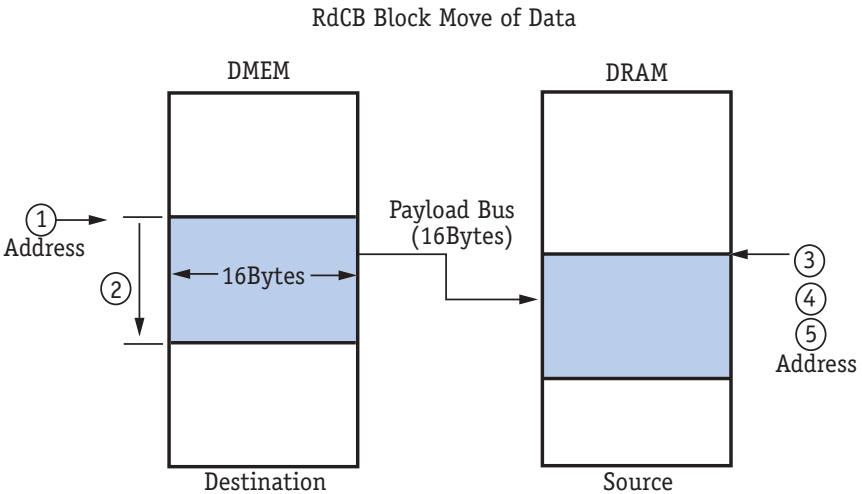
When DMA transaction requests receive a no-acknowledge (NACK) on the Payload Bus, the bus controller retries the request up to 16 (maximum) times before reporting a bus error. The bus error sets status in *WrCB0_Ctl* bits [27:24] *Error* field with an encoding, this generates an event for the *Event0* or *Event1* register, and immediately terminates the transfer by setting *WrCB0_Ctl* bit [31] *Avail* field. When the *WrCB0_Ctl* bit [30] *NoRetry* field is set, the bus controller does not retry, and reports the bus NACK immediately as a bus error.

Read Control Blocks (RdCB0_ , RdCB1_)

Two Read Control Blocks (RdCB0_ and RdCB1_) provide the capability for general purpose read tasks, such as Buffer Transfers, QMU dequeues and BTag allocates. These tasks move data across the Payload Bus into DMEM in bursts of four (4) cycles with 16Bytes per cycle.

Figure 22 on page 120 shows a Buffer Transfer. In general, you are moving data from SDRAM (the source) to DMEM (the destination). Specifically, you are moving data starting at the (*PoolID*, *BTag* and *Offset* (3,4,5)) location inside DRAM to DMEM at the *LineAddr* (1) location with a *Length* (2). These individual fields that are used to set up the details of the block move make up parts of these registers: RdCB*n*_Sys_Addr, RdCB*n*_Ctl and RdCB*n*_DMA_Addr.

Figure 22 DMA Operation (Buffer Transfer) Using RdCB*n*_ Registers



Register	Fields	
RdCBn_Sys_Addr	BTag (4)	Offset (5)
RdCBn_Ctl		Length (2)
RdCBn_DMA_Addr	PoolID (3)	LineAddr (1)

Buffer Transfer Setup Using RdCBn_Sys_Addr, RdCBn_Ctl and RdCBn_DMA_Addr:
To set up this single, contiguous data transfer, the CPRC writes a system address (consisting of a *PoolID*, *BTag*, and *Offset* for buffer memory in SDRAM), a line address for DMEM, to the RdCB. The length is written by the CPRC to be the desired transfer length.

To perform Buffer Transfers involves setting the bits for RdCB0_Sys_Addr (0xBC*n*04420), RdCB0_Ctl (0xBC*n*04424) and RdCB0_DMA_Addr (0xBC*n*04428).

To set up a general contiguous data transfer, the CPRC process must do the following:

1. Ensure that the RdCB0 is available by reading *RdCB0_Ctl* bit [31] *Avail* field =1.
2. Write *RdCB0_Sys_Addr* with the system address to be read, in the form of:
   RdCB0_Sys_Addr bits [31:6] BTag field = BTag, and
   RdCB0_Sys_Addr bits [15:4] Offset field = Offset
   (Offset is a 16Byte starting buffer offset, typically equal to 0, or aligned to a 64Byte boundary.
3. Write the *PoolID* portion of the system address to be read into *RdCB0_DMA_Addr* bits [20:16] *PoolID* field. Write *RdCB0_DMA_Addr* bits [13:4] *LineAddr* field with the location of a buffer in DMEM, typically aligned on a 64Byte boundary. This is the DMEM location that the DMA engine uses to begin writing data from SDRAM.
4. Write *RdCB0_Ctl* with *RdCB0_Ctl* bits [13:0] *Length* field equal to the number of bytes to be transferred, *RdCB0_Ctl* bit [31] *Avail* field equal to 0, and *RdCB0_Ctl.* bit [29] *Modulo64* equal to 0 to cause the *RdCB0_Sys_Addr* bits [15:4] *Offset* field and *RdCB0_DMA_Addr* bits [13:4] *LineAddr* field to increment during the DMA for a contiguous block register.

For complete details about specific registers go to their reference. Refer to: "RdCB0_Sys_Addr Register (CP Rd Control Block0 Function)" *on page 502*, "RdCB0_Ctl Register (CP Rd Control Block0 Function)" *on page 503*, and "RdCB0_DMA_Addr Register (CP Rd Control Block0 Function)" *on page 504*. *You also have the following registers available for Control Block: RdCB1_Sys_Addr, RdCB1_Ctl and RdCB1_DMA_Addr that function in the same manner.*

Buffer Transfer Operations Using RdCB*n*_Sys_Addr, RdCB*n*_Ctl and RdCB*n*_DMA_Addr: An availability bit indicates that the block of DMEM is controller by to either DMA or CPRC. When set, the CPRC controls the block; when clear, the DMA engine controls the block and is free to transfer into it.

Clearing the available bit, *RdCB0_Ctl* bit [31] *Avail* field, the CPRC process initiates a 64Byte data transfer from SDRAM beginning at the system address consisting of (*RdCB0_Sys_Addr* bits [31:16] *BTag* field, *RdCB0_Sys_Addr* bits [15:4] *Offset* field, *and RdCB0_DMA_Addr* bits [20:16] *PoolID* field to DMEM beginning at the 16Byte line addressed by *RdCB0_DMA_Addr* bits [13:4] *LineAddr* field. The SDRAM DMA engine transfers payload out of SDRAM in a 4-cycle, 16Byte-per-cycle burst, decrementing *RdCB0_Ctl* bits [13:0] *Length* field by 64 for each burst.

Transfer continues until *RdCB0_Ctl* bits [13:0] *Length* field equals 0 (at which time the DMA engine sets *RdCB0_Ctl* bit [31] *Avail* field, thus returning control of the RdCB back to the CPRC process).

Initiating transfers with *RdCB0_Ctl* bits [13:0] *Length* field equal to 0 causes a single 64Byte transfer. If a 4-cycle, 64Byte transfer is started with *RdCB0_Ctl* bits [13:4] *Length* field < 64Bytes, only the number of 16Byte lines needed to satisfy the whole length according to the *Length* field actually get read from SDRAM. Unpredictable data completes the full 64Bytes returned, and the Length field after the burst is set to 0.

If the *RdCB0_Sys_Addr* bits [13:4] *Offset* field is aligned to a 64Byte boundary, a contiguous 64Byte block of SDRAM is read. If the Offset is 64Byte unaligned, the SDRAM block is read in a wrapped fashion which is generally not useful. Typically, contiguous transfers start with aligned offsets.

RdCB0_DMA_Addr bits [13:4] *LineAddr* field increments for each of the four 16Byte-per-cycle transfers to provide an address into DMEM. Typically, R*dCB0_DMA_Addr* bits [13:4] *LineAddr* field starts at a 64Byte aligned address. *RdCB0_DMA_Addr* bits [13:4] *LineAddr* field can start unaligned, but the resulting wrap behavior is not useful. The burst read from SDRAM always returns 64Bytes. *RdCB0_Sys_Addr* bits [15:4] *Offset* field increments by 64 each 64Byte burst.

Initiating transfers with the modulo64, *RdCB0_Ctl* bit [29] *Modulo64,* equal to 1prevents updates of the *RdCB0_Sys_Addr* bits [15:4] *Offset* field and causes *RdCB0_DMA_Addr* bits [13:4] *LineAddr* field to increment modulo 64Bytes, effectively returning the *LineAddr* to wrap back to the starting value. This feature is useful for reads from the QMU or BMU where the system address contains a command, not an address. The RdCB can be used again without resetting the *Offset* and *RdCB0_DMA_Addr* bits [13:4] *LineAddr* field.

The CPRC process can read the state of the DMA machine at any time from *RdCB0_Ctl* bits [17:16] *State* field. The *RdCB0_Ctl* word generally should not be written by the CPRC process when hardware is operating (that is, *RdCB0_Ctl* bit [31] *Avail* field=0). On a write to *RdCB0_Ctl* bits [17:16] *State* field are only updated if bit [31] *Avail* field=1.

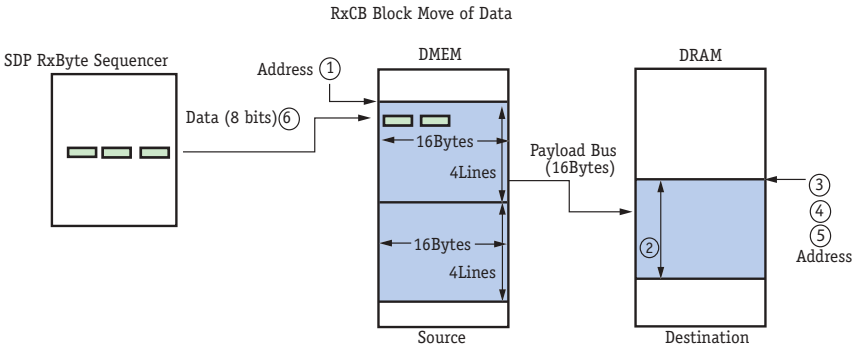
When DMA transaction requests receive a no-acknowledge (NACK) on the Payload Bus, the bus controller retries the request up to 16 times before reporting a bus error. The bus error sets status in *RdCB0_Ctl* bits [27:24] *Error* field, generates an event for the Event register, and immediately terminates the transfer by setting *RdCB0_Ctl* bit [31] *Avail* field. When the *RdCB0_Ctl bit [30] NoRetry* field is set, the bus controller does not retry, and reports the bus NACK immediately as a bus error.

SDP RxByte Processor Receive Control Blocks (RxCB0_, RxCB1_)

The CPRC controls receive operations using the two (2) Receive Control Blocks (RxCB0_ and RxCB1_) that handle the payload write operation much like the (WrCB0 and WrCB1), but add the capability to control SDP RxByte Processor writes to DMEM. The SDP RxByte Processor directs the incoming data stream into DMEM a byte at a time. In hardware, the byte stream is accumulated into 16Byte lines that are written to DMEM in a single cycle. Using a RxCB, the CPRC can set up a payload receive path from the RxByte Processor to DMEM to SDRAM. Payload data movement happens in hardware with no further CPRC control.

Figure 23 on page 124 shows a Buffer Transfer. In general, you are moving data from SDP (the source) to SDRAM (the destination). Specifically, you are moving data from the SDP RxByte Sequencer in 8bit units using *ByteAddr* (6) and counting the bytes moved with a *Length* (2) starting at the *LineAddr* (1) location inside DMEM to SDRAM at the (*PoolID, BTag* and *Offset* (3,4,5)) location. These individual fields are used to setup the details of the block move and make up parts of these registers: RxCB*n*_Sys_Addr, RxCB*n*_Ctl, RxCB*n*_DMA_Addr, and RxCB*n*_SDP_Addr.

Figure 23 DMA Operation (Buffer Transfer) Using RxCB*n* Registers



Register	Fields	
RxCBn_Sys_Addr	BTag 4	Offset 5
RxCBn_Ctl		Length 2 Counts the Bytes moved from the sequencer)
RxCBn_DMA_Addr	PoolID 3	LineAddr 1
RxCBn_SDP_Addr		ByteAddr 6

Buffer Transfer Setup Using RxCB*n*_Sys_Addr, RxCB*n*_Ctl RxCB*n*_DMA_Addr and RxCB*n*_SDP_Addr:
To perform Buffer Transfers involves setting the bits for RxCB0_Sys_Addr (0xBC*n*04080), RxCB0_Ctl (0xBC*n*4084), RxCB0_DMA_Addr (0xBC*n*04088), and RxCB0_SDP_Addr (0xBC*n*0408C).

To set up a typical single receive operation, the CPRC must do the following:

1 Ensure that the RxCB0 is available by testing that *RxCB0_Ctl* bit [31] *Avail* field = 1.

2 Write *RxCB0_Sys_Addr* with the system address to be written in the form of:
RxCB0_Sys_Addr bits [31:16] *BTag* field = *BTag*, and
RxCB0_Sy_Addr bits [15:4] *Offset* field = Offset
(Offset is a 16Byte starting buffer offset, typically aligned to a 64Byte boundary).

3 Write the Pool ID portion of the system address to be written into *RxCB0_DMA_Addr* bits [20:16] *PoolID* field. Write *RxCB0_DMA_Addr* bit [13:4] *LineAddr* field with a 16Byte address in DMEM, typically aligned on a 128Byte boundary. This is the location that the DMA engine uses to begin transferring data out of DMEM to SDRAM.

4 Write *RxCB0_SDP_Addr* bits [15:0] *ByteAddr* field with a 16Byte address in DMEM, typically the same value (buffer) as *RxCB0_DMA_Addr* bits [13:4] *LineAddr* field. This is the location that the SDP RxByte Processor uses to begin transferring bytes into DMEM.

5 Write the *RxCB0_Ctl* bits [15:0] *RxLength* field to zero, clear *RxCB0_Ctl* bit [23] *Own1* field and *RxCB0_Ctl* bit [22] *Own0* field to give ownership of the double buffer to the SDP DMA (rather than the SDRAM DMA) engine, and clear *RxCB0_Ctl* bit [31] *Avail* field that starts the SDRAM DMA engine.

For complete details about specific registers go to their reference. Refer to: "RxCB0_Sys_Addr Register (CP Rx Control Block0 Function)" on page 493, "RxCB0_Ctl Register (CP Rx Control Block0 Function)" on page 494, "RxCB0_DMA_Addr Register (CP Rx Control Block0 Function)" on page 497 and "RxCB0_SDP_Addr Register (CP Rx Control Block0 Function)" on page 498. *You also have the following registers available for Control Block1: RxCB1_Sys_Addr, RxCB1_Ctl, RxCB1_DMA_Addr and RxCB1_SDP_Addr that function in the same manner.*

Buffer Transfer Operations Using RxCB*n*_Sys_Addr, RxCB*n*_Ctl RxCB*n*_DMA_Addr and RxCB*n*_SDP_Addr:
The DMA engine always uses 128Bytes double buffering (that is, two (2) sequential 64Byte DMEM buffers (16bytes wide x 4lines high)) to handle payloads of arbitrary length. The transfer of these buffers can be individually controlled by the CPRC.

Typically, both buffers are enabled by the CPRC via the *RxCB0_Ctl* bit [23] *Own1* field and *RxCB0_Ctl* bit [22] *Own0* field and the DMA initiates transfers whenever the next 64Byte block within the buffer becomes available. Ownership bits track the status of the two contiguous 64Byte blocks in DMEM.

RxCB0_DMA_Addr bits [13:4] *LineAddr* field and *RxCB0_SDP_Addr* bits [15:0] *ByteAddr* field typically point to the 128Byte aligned buffer. Increments of *RxCB0_SDP_Addr* bits [15:0] *ByteAddr* field and *RxCB0_DMA_Addr* bits [13:4] *LineAddr* field are done modulo 128 so that writing or reading the last line in the buffer causes the pointers to wrap back to the start of the buffer.

RxCB0_Ctl bit [23] *Own1* field and *RxCB0_Ctl* bit [22] *Own0* field track the ownership of the two (2) 64Byte blocks in the 128Byte buffer. By clearing the ownership bits initially, the CPRC allows the SDP RxByte Processor to write into the DMEM buffers. When *RxCB0_SDP Addr* bits [15:0] *ByteAddr* field reaches a 64Byte boundary, the hardware sets the corresponding ownership bit to indicate that the SDRAM DMA engine now owns the block. It initiates a 64Byte transfer to SDRAM as soon as possible, incrementing

RxCB0_DMA_Addr bits [13:4] *LineAddr* field by 16 for each of the four (4), 16Byte-per-cycle transfers to provide an address into DMEM. It also adds 64 to *RxCB0_Sys_Addr* bits [15:4] *Offset* field to update the SDRAM address. When the DMA is complete, the *RxCB0_Ctl* bit [23] *Own1* field or *RxCB0_Ctl* bit [22] *Own0* field is cleared to allow the SDP to reuse that half of the double buffer.

Thus, *RxCB0_SDP_Addr* bits [15:0] *ByteAddr* field and *RxCB0_DMA_Addr* bits [13:4] *LineAddr field* act as a pair, following one another through a payload transfer. *RxCB0_SDP_Addr* bits [15:0] *ByteAddr* field leads as the SDP RxByte Processor fills DMEM with payload bytes. When a 64Byte buffer is full, an SDRAM DMA transaction uses the lagging *RxCB0_DMA_Addr* bits [13:4] *LineAddr* field to move the buffer to SDRAM. The SDP RxByte Processor forces a line write when signaling end-of-frame by setting *RxCtl0_Status* bit [31] *Avail* field, which must happen exclusive of an SDP byte write. Unwritten bytes at the end of the 16Byte line are undefined. This clears *RxCB0_SDP_Addr* bits [6:0] within the *ByteAddr* field, clears *RxCB0_DMA_Addr* bits [6:4] within the *LineAddr* field, and sets *RxCB0_Ctl* bit [31] *Avail* field to realign the double buffer and to return control of the RxCB to the CPRC.

In hardware, an accumulation buffer assembles sequential SDP RxByte Processor writes until a 16Byte DMEM line boundary is crossed. This triggers a line write to DMEM at the address in *RxCB0_SDP_Addr* bits [15:4] *ByteAddr* field if the associated ownership bit allows it. There are no hardware interlocks that assure the RxCB is configured before accepting SDP byte writes. The CPRC process must be sure to configure *RxCB0_SDP_Addr* bits [15:0] *ByteAddr* field and both *Own0* field bit [22] and *Own1* field bit [23] before passing the datascope to the SDP.

For some applications, the CPRC process can choose to write *RxCB0_DMA_Addr*, *RxCB0_Sys_Addr*, and *RxCB0_Ctl* bit [31] *Avail* field (using a byte operation later). For each byte transferred, hardware increments the *RxCB0_Ctl* bits [15:0] *RxLength* field to reflect the total number of bytes in the receive payload.

The RxCB can be used with *RxCB0_Sys_Addr* bits [15:4] *Offset* field and *RxCB0_DMA_Addr* bits [13:4] *LineAddr* field pointing to 16Byte addresses that are not 64Byte aligned. In this case, the SDRAM DMA inhibits any writes that wrap within the SDRAM block. This can be used to transfer partial blocks from DMEM to SDRAM in assembly operations. After a 4-cycle burst, the *RxCB0_Sys_Addr* bits [15:4] *Offset* field is always set to the next aligned 64Byte block.

There are eight (8) bits in each of the Out-Of-Band fields (OOB). Refer toTable 22 on page 127.They (OOB) are located in the *TxCB0_SDP_Addr* register, bits [31:24] are for OOB0 and bits [23:16] are for OOB1. Eight (8) Out-Of-Band bits are transferred to SDRAM along with every 64Byte payload transfer. The 7th Bit of OOB*n* indicates that the SDP encountered an error receiving this frame. The 6th Bit of OOB*n* indicates that this block of 64Bytes contains the End-of-Packet (EOP), and when the 6th Bit is set, the remaining six bits in the OOB*n* field indicate the position of the last byte. These (OOB*n*) bits get transferred to SDRAM automatically, based on SDP error signals and *RxCB0_Ctl* bits [15:0] *RxLength* field. During test, the RxCB can be used to write the OOB bits. Writing a buffer in DMEM and setting up *RxCB0_Sys_Addr*, *RxCB0_DMA_Addr* bits [13:4] *LineAddr* field, *RxCB0_Ctl* bit [29] *EOP* field, and *RxCB0_Ctl* bits [15:0] *RxLength* field determine what payload and OOB*n* bits get written to SDRAM.

Test software can force the transfer by clearing *RxCB0_Ctl* bit [31] *Avail* field and setting the appropriate *RxCB0_Ctl* bits [23:22] *Own1* or *Own0* bit.

Table 22 Out-of-Band Bits and Functions

SIDE-CAR BITS	FUNCTION
7	Packet Error
6	End of Packet (EOP)
5:0	Encoded Value (for valid Bytes) Legal Range= 0 to 63

The CPRC process can read the state of both DMA engines and the EOP status at any time from *RxCB0_Ctl* bit [19] *SDP State* field and the *RxCB0_Ctl* bits [17:16] *DMA State* field. The *RxCB0_Ctl* word generally should not be written by the CPRC process when hardware is operating (that is, when *RxCB0_Ctl* bit [31] *Avail* field=0).

On a write to *RxCB0_Ctl* bit [19] *SDP State* field, bits [17:16] *DMA State*, and bit [29] *EOP* fields are only updated if bit [31] Avail=1. Additionally, *RxCB0_Ctl* bit [29] *EOP* field is not updated if bit [28] *Protect_EOP*=1.

When DMA transaction requests receive no-acknowledge (NACK) on the Payload Bus, the bus controller retries the request up to 16 (maximum) times before reporting a bus error. The bus error sets status in *RxCB0_Ctl* bits [27:24] *Error* field, generates an event for the *Event 1*, and immediately terminates the SDRAM transfer by setting *RxCB0_Ctl* bit [31] *Avail* field. When the *RxCB0_Ctl* bit [30] *NoRetry* field is set, the bus controller does not retry, and reports the bus NACK immediately as a bus error.

Receive payload can be recycled back through the SDP RxByte Processor using a configuration option and explicit CPRC control. A path can be set up for payload bytes to travel from the SDP to DMEM, back to the SDP to DMEM and then to SDRAM.

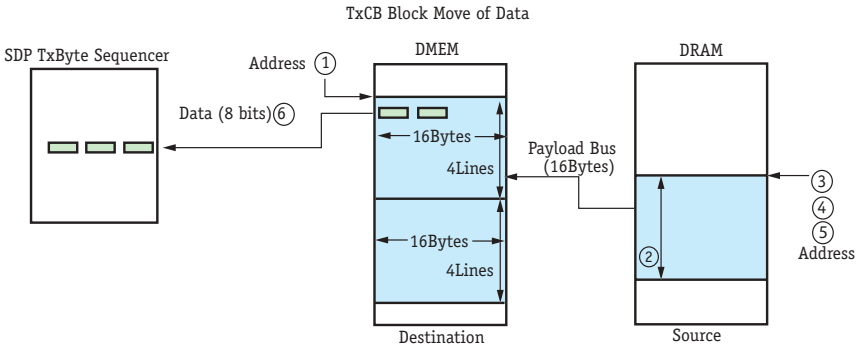
The NP supports two (2) near-end loopbacks for the purposes of recirculation. The first connects the output of the Large Transmit FIFO to the input of the Large Receive FIFO, the second connects the output of the Small Transmit FIFO to the input of the Small Receive FIFO. For more information about recirculation, see "Configuration for Recirculation Operations Using RxSDP and TxSDP" on page 98.

SDP TxByte Processor Transmit Control Block (TxCB0_, TxCB1_)

The CPRC controls transmit operations using the two (2) Transmit Control Blocks (TxCB0_ and TxCB1_). The TxCBs handle the payload read operation much like the (RdCB0 and RdCB1), but add the capability to control TxByte Processor reads from DMEM. Using a TxCB, the CPRC can set up a payload transmit path from SDRAM to DMEM, and from DMEM to the TxByte Processor. Payload data movement happens in hardware with no further CPRC control.

Figure 24 on page 129 shows a Buffer Transfer. In general, you are moving data from SDRAM (the source) to SDP (the destination). Specifically, you are moving data starting at the (*PoolID*, *BTag* and *Offset* (3,4,5)) location inside the SDRAM to DMEM at the *LineAddr* (1) location and moving 8bit units of data from the DMEM into the RxByte Sequencer using *ByteAddr* (6) and counting the bytes moved with a *Length* (2). These individual fields that are used to set up the details of the block move make up parts of these registers: TxCB*n*_Sys_Addr, TxCB*n*_Ctl, TxCB*n*_DMA_Addr and TxCB*n*_SDP_Addr.

Figure 24 DMA Operation (Buffer Transfer) Using TxCB*n*_ Registers



Register	Fields	
TxCBn_Sys_Addr	BTag 4	Offset 5
TxCBn_Ctl		Length 2 Counts the Bytes moved from the sequencer)
TxCBn_DMA_Addr	PoolID 3	LineAddr 1
TxCBn_SDP_Addr		ByteAddr 6

Buffer Transfer Setup Using TxCB*n*_Sys_Addr, TxCB*n*_Ctl TxCB*n*_DMA_Addr and TxCB*n*_SDP_Addr:
To perform Buffer Transfers involves setting the bits for TxCB0_Sys_Addr (0xBC*n*04180), TxCB0_Ctl (0xBC*n*4184), TxCB0_DMA_Addr (0xBC*n*04188), and TxCB0_SDP_Addr (0xBC*n*0418C).

To set up a typical single transmit operation for > 64Bytes of data, the CPRC must do the following:

1 Ensure that the TxCB0 is available by reading *TxCB0_Ctl* and testing that *TxCB0_Ctl* bit [31] *Avail* field=1.

2 Write *TxCB0_Sys* with the system address to be written in the form of:
TxCB0_Sys_Addr bits [31:16] *BTag* field = *BTag*, and
TxCB0_Sys_Addr bits [15:4] *Offset* field = Offset
(Offset is a 16Byte starting buffer offset, typically aligned to a 64Byte boundary).

3 Write the Pool ID portion of the system address to be written into *TxCB0_DMA_Addr* bits [20:16] *PoolID* field. *TxCB0_DMA_Addr* bits [13:4] *LineAddr* field with the location of a 64Byte buffer in DMEM, typically aligned on a 128Byte boundary. This is the DMEM location that the DMA engine uses to begin reading data from SDRAM.

4 Write *TxCB0_SDP_Addr* bits [15:0] *ByteAddr* field with a 16Byte address in DMEM, typically the same value (buffer) as *TxCB0_DMA_Addr* bits [13:4] *LineAddr* field. This is the location that the SDP TxByte Processor uses to begin transferring bytes out of DMEM.

5 Write the *TxCB0_Ctl* register, to initialize the *TxCB0_Ctl* bits [15:0] *TxLength* field with the number of bytes to transfer, clear *TxCB0_Ctl* bit [23] *Own1* field and *TxCB0_Ctl* bit [22] *Own0* to give ownership of the buffer to the SDRAM (rather than the SDP) DMA engine, enabling the prefetch of 128Bytes of payload, clear TxCB0_Ctl bit [31] *Avail* field to start the SDRAM DMA engine.

For complete details about specific registers go to their reference. Refer to: "TxCB0_Sys _Addr Register (CP Tx Control Block0 Function)" on page 505, "TxCB0_Ctl Register (CP Tx Control Block0 Function)" on page 506, "TxCB0_DMA_Addr Register (CP Tx Control Block0 Function)" on page 507 and "TxCB0_SDP_Addr Register (CP Tx Control Block0 Function)" on page 508. *You also have the following registers available for Control Block1: TxCB1_Sys_Addr, TxCB1_Ctl, TxCB1_DMA_Addr and TxCB1_SDP_Addr that function in the same manner.*

Buffer Transfer Operations Using TxCB*n*_Sys_Addr, TxCB*n*_Ctl TxCB*n*_DMA_Addr and TxCB*n*_SDP_Addr:
The CPRC can set up a payload transfer of arbitrary length using double buffering. *TxCB0_DMA_Addr* bits [13:4] *LineAddr* field and *TxCB0_SDP_Addr* bits [15:0] *ByteAddr* field typically point to a 128Byte aligned buffer. Increments of *TxCB_SDP* bits [15:6] *ByteAddr* field and *TxCB0_DMA_Addr* bits [13:4] *LineAddr* field are done to modulo 128 so that writing or reading the last line in the buffer causes the pointers to wrap back to the start of the buffer.

TxCB0_Ctl bit [22] *Own0* field and *TxCB0_Ctl* bit [23] *Own1* field track the ownership of the two 64Byte blocks in the 128Byte double buffer. By clearing the ownership bits initially, the CPRC allows the SDRAM DMA engine to prefetch payload into the DMEM buffers. *TxCB0_DMA_Addr* bits [13:4] *LineAddr* field increments by 16 for each of the four 16Byte-per-cycle transfers to provide an address into DMEM. It also adds 64 to *TxCB0_Sys_Addr* bits [15:4] *Offset* field to update the SDRAM address.

If the payload length is ≤64 bytes, only *TxCB0_Ctl* bit [22] *Own0* field should be cleared and *TxCB0_Ctl* bit [23] *Own1* field set to keep the DMA engine from wasting bandwidth by prefetching an extra block. If the payload length is >64 bytes, *TxCB0_Sys_Addr* bit [22] *Own0* field and *TxCB0_Sys_Addr* bit [23] *Own1* field must be cleared to prefetch the first two blocks of payload. When the blocks of payload arrive from SDRAM, the DMA engine sets the corresponding ownership bit to indicate that the SDP DMA engine now owns the block.

In hardware, setting *Own* causes the SDP TxByte Processor to read and buffer a 16Byte line of DMEM pointed to by *TxCB0_SDP_Addr* bits [15:6] *ByteAddr* field. The SDP TxByte Processor byte reads are serviced from this read buffer. For each byte transferred, the address in *TxCB0_SDP_Addr bits* [15:0] *ByteAddr* field is incremented. Crossing a 16Byte DMEM line triggers another line read from DMEM from the address in *TxCB0_SDP_Addr* bits [15:6] *ByteAddr* field. When the last line of a 64Byte block of payload has been read out of DMEM, the SDP DMA engine clears the corresponding *Own* bit to allow the SDRAM DMA engine to reuse that half of the buffer.

Thus, *TxCB0_DMA_Addr* bits [13:4] *LineAddr* field and *TxCB0_SDP_Addr* bits [15:0] *ByteAddr* field act as a pair, following one another through a payload transfer. *TxCB0_DMA_Addr* bits [13:4] *LineAddr* field leads as the DMA engine fills DMEM with payload from SDRAM. When a 64Byte buffer is full, the SDP TxByte Processor uses the lagging *TxCB0_SDP_Addr* bits [15:0] *ByteAddr* field to read bytes of payload from DMEM. When the TxCB0_Ctl bits [15:0] *TxLength* field equals 0, the hardware clears the *TxCB0_Sys_Addr* bit [22] *Own0* field and the *TxCB0_Sys_Addr* bit [23] *Own1* field and signals the SDP that the last byte was transmitted. The SDP TxByte Processor signals end-of-frame by setting *TxCtl_Status* bit [31] *Avail* field. This clears *TxCB0_SDP_Addr* bits [6:0] *within the ByteAddr* field, clears *TxCB0_DMA_Addr* bits [6:4] with the *LineAddr* field and sets *TxCB0_Ctl* bit [31] *Avail* field to realign to the double buffer and to return control of the TxCB to the CPRC.

There are eight (8) bits in each of the Out-Of-Band field (OOB). The OOB are located in the *TxCB0_SDP_Addr* register. Bits [31:24] are for OOB0 and bits [23:16] are for OOB1. Eight (8) OOB bits are transferred to SDRAM along with every 64Byte payload transfer. The 7th Bit of the OOB*n* indicates that the SDP encountered an error receiving this frame.

The 6th Bit of the OOB*n* indicates that this block of 64Bytes contains the last byte of the payload, and when the 6th Bit of the (OOB*n*) is set, the remaining six bits indicate the position of the last byte. These (OOB*n*) bits get transferred to *TxCB0_SDP_Addr* bits [31:24] *OutOfBand0* and *TxCB0_SDP_Addr* bits [23:16] *OutOfBand1* field for every payload read. Based on the *TxCB0_Ctl* bit [28] *OOB* field, the hardware uses either the *TxCB0_Ctl* bits [15:0] *TxLength* field or the *TxCB0_SDP_Addr OutOfBand* field to determine the last payload byte. Hardware decrements the *TxCB0_Ctl* bits [15:0] *TxLength* field and the appropriate *TxCB0_SDP_Addr OutOfBandn* field for each byte transferred. When *TxCB0_Ctl* bit [28] *OOB* field is clear, the *TxLength* equals 0 indicates the payload transfer is finished. When the *TxCB0_Ctl* bit [28] *OOB* is set, and the appropriate *TxCB0_SDP_Addr OutOfBandn* field indicates last byte and the position equals 0, the payload transfer is finished.

During test, the TxCB can be used to read the SDRAM data and OOB bits. Setting up *TxCB0_Sys_Addr* and *TxCB0_DMA _Addr* bits [13:4] *LineAddr* field determines where payload gets read into DMEM. Software can force the transfer by clearing *TxCB0_Ctl* bit [31] *Avail* field and setting the appropriate *TxCB0_Ctl* bit [23 or 22] *Ownn* bit. When the transfer finishes, the OOB bits can be read from TxCB0_SDP_Addr OutOfBandn field.

The CPRC process can read the state of both DMA engines and the EOP status at any time from *TxCB0_Ctl* bits [19:18] *SDP State* field and *TxCB0_Ctl* bits [17:16] *DMA State* field. The *TxCB0_Ctl* word generally should not be written by the CPRC process when hardware is operating (that is, when *TxCB0_Ctl* bit [31] *Avail=0*). On writes to *TxCB0_Ctl* bits [17:16] *DMA State* field, bits [19:18] *SDP State* field, and bit [29] *EOP* field are only updated if bit [31] *Avail*=1.

When DMA transaction requests receive no-acknowledge (NACK) on the payload bus, the bus controller retries the request up to 16 times before reporting a bus error. The bus error sets status in *TxCB0_Ctl* bits [27:24] *Error* field, generates an event for the *Event1* register, and immediately terminates the SDRAM transfer by setting *TxCB0_Ctl* bit [31] *Avail* field. When the *TxCB_Ctl* bit [30] *NoRetry* field is set, the bus controller does not retry, and reports the bus NACK immediately as a bus error.

Transmit payload can be recycled back through DMEM and retransmitted using a configuration option and explicit CPRC control. A path can be set up for the payload to travel from the SDRAM to DMEM to the SDP to DMEM to the SDP (a process call recirculation). For more information about recirculation, see "Configuration for Recirculation Operations Using RxSDP and TxSDP" on page 98.

Ring Bus Registers

Configuration Space contains registers to control the Ring Bus, including transmitting messages, receiving messages, and receiving responses.

Ring Bus Transmit (Tx) Messages Registers

Configuration Space includes four (4) sets of registers used to transmit messages on the Ring Bus. The four (4) consist of:
TxMsg0_Ctl, *TxMsg0_Data_H*, and *TxMsg0_Data_L;* T*xMsg1_Ctl*, *TxMsg1_Data_H*, and *TxMsg1_Data_L*; *TxMsg2_C*tl,*TxMsg2_Data_H*, and *TxMsg2_Data_L*; *TxMsg3_Ctl, TxMsg3_Data_H*, and *TxMsg3_Data_L*. The CPRC has access to all four (4) sets. The SDP RxByte and TxByte Processors have access to only sets zero and one, (*TxMsg0_Ctl, TxMsg0_Data_H*, and *TxMsg0_Data_L;* T*xMsg1_Ctl, TxMsg1_Data_H*, and *TxMsg1_Data_L)*.

Refer to the SDP Programming document in the *C-Ware Application Development Guide* for SDP addressing of these registers.

When programming, mutual exclusivity among users of each TxMsgn_Ctl must be maintained.

The 8Byte data portion of a single-slot Ring Bus message is written into two (2) 4Byte *TxMsg0_Data_H* and *TxMsg0_Data_L* registers. The control portion of a Ring Bus message is written into the *TxMsg0_Ctl* register bits [23, 19:0] in the exact format to be sent directly out on the Ring Bus control wires. Clearing the *TxMsg0_Ctl* bit [31] *Avail* transfers ownership of the transmit message registers to the Ring Bus control, effectively giving the send command. The Ring Bus controller then puts the 21bits of control from the *TxMsg0_Ctl* register and the 8Bytes of data from the *TxMsg0_Data_L* registers out on the Ring Bus. The Ring Bus controller sets the *TxMsg0_Ctl* bit [31] *Avail* when the message has gone out, indicating to the CPRC that the transmit message register set is available to send subsequent messages. Four (4) messages of 8Byte data length can be sent independently using the four (4) sets of transmit message registers. Transmit message register sets can also be combined to send messages of 16Bytes and 32Bytes length (two and four Ring Bus slots). Multiple slot messages may begin with any of the transmit message register sets. The additional data is placed in sequential, wrapped *TxMsg0_Data* registers. The beginning *TxMsg0_Ctl* register must contain the appropriate slot length. The sequential *TxMsg0_Ctl* registers that match participating sequential *TxMsg0_Data* registers must have the *Avail* bit [31] set, that is, must not be in use for another transmit, but otherwise have no effect on the transaction.

Ring Bus (Rx) Receive Message Registers

Configuration Space includes a set of registers used to receive unsolicited messages consisting of *RxMsg_Ctl*, and *RxMsg_FIFO*.

Unsolicited messages are of type: indication, confirmation, or request. These incoming messages enter a 4-entry x 8-Byte FIFO in the Ring Bus controller. The CPRC process uses load instructions to read the head of the FIFO from the *RxMsg_Ctl* and *RxMsg_FIFO* registers. When set, *RxMsg_Ctl* bit [31] *State* indicates that a complete, valid message resides in the FIFO. *RxMsg_Ctl* bits [23,14:10,4:0] contain the control portion of the message as received off the Ring Bus. The *Dst* field [9:5] which must be this channel's Ring Bus ID is not reported.

When the FIFO contains a valid message, the CPRC reads *RxMsg_FIFO* to obtain the first 4Bytes of the data portion of the Ring Bus message. The CPRC continues to read from the *RxMsg_FIFO* register to empty the complete message out of the FIFO. The CPRC process must track the message length given by initial *RxMsg_Ctl* bits [17:15] *Len* to know how many times to read the *RxMsg_FIFO* to obtain the complete message. When the *RxMsg_Ctl* indicates a message is valid, the entire data portion of the message is available through *RxMsg_FIFO*; there is no need for the CPRC process to check intermediate data status.

Ring Bus Receive (Rx) Response Registers

Messages initiated by the CPRC as a request type expect to receive a subsequent response type message, for example TLU requests. Configuration space includes eight (8) sets of registers used to receive responses. The eight (8) consist of:
RxResp0_Ctl, *RxResp0_Data_H*, *RxResp0_Data_L*; *RxResp1_Ctl*, *RxResp1_Data_H*, *RxResp1_Data_L*; *RxResp2_Ctl*, *RxResp2_Data_H*, *RxResp2_Data_L*; *RxResp3_Ctl*, *RxResp3_Data_H*, *RxResp3_Data_L*; *RxResp4_Ctl*, *RxResp4_Data_H*, *RxResp4_Data_L*; *RxResp5_Ctl*, *RxResp5_Data_H*, *RxResp5_Data_L*; *RxResp6_Ctl*, *RxResp6_Data_H*, *RxResp6_Data_L*; *RxResp7_Ctl*, *RxResp7_Data_H*, *RxResp7_Data_L*.

The control field of a Ring Bus response is moved into a *RxRespn_Ctl* register and the data field of a Ring Bus response slot is moved into a *RxRespn_Data_H*/*RxRespn_Data_L* register pair. Responses are directed to the specific one of eight register sets based on the *sequence* bits [12:10] of the incoming Ring Bus control field. The sequence field is merely an echo of the sequence field that was sent in the control field of the request message that triggered this response. *Sequence* field bits [14:13] have no effect on hardware and can be used by software for additional ordering information.

When set, *RxResp0_Ctl* bit [31] *Avail* indicates that a complete, valid response has been received. The *Dst* field [9:5] which must be this channel's Ring Bus ID, the *Type* field bits [19:18] which must be type response, and the *Length* field bits [17:15] which must be known by requesting software, are not reported.

Eight (8) responses of 8Byte data length can be received independently using the eight (8) sets of receive response registers. Receive response register sets can also be combined to receive responses of 16Byte and 32Byte length (two or four Ring Bus slots). Multiple slot responses begin with the receive response register set specified by the *sequence* bits. The additional data is placed in sequential, wrapped *RxResp0_Data* registers. The beginning *RxResp0_Ctl* register contains the Ring Bus control field. The sequential *RxResp0_Ctl* registers that match participating sequential *RxResp0_Data* registers are not updated.

While receiving cells/packets, the SDPRxByte Processor uses its access to the Ring Bus transmit message registers to initiate lookup requests for the TLU based on various fields (such as the destination address) extracted from the incoming header. The TLU responses to the lookup requests are received and interpreted by the CPRC.

For complete details about specific registers go to their reference. Refer to: "TxMsg0_Ctl Register (CP Ring Bus Tx Message Control Function)" on page 510, "TxMsg0_Data_H Register (CP Ring Bus Tx Message Control Function)" on page 512, "TxMsg0_Data_L Register (CP Ring Bus Tx Message Control Function)" on page 512, "RxMsg_Ctl Register (CP Ring Bus Rx Message Control Function)" on page 515, "RxMsg_FIFO Register (CP Ring Bus Rx Message Control Function)" on page 517, "RxResp0_Ctl Register (CP Ring Bus Rx Response Control Function)" on page 513, "RxResp0_Data_H Register (CP Ring Bus Rx Response Control Function)" on page 514, and "RxResp0_Data_L Register (CP Ring Bus Rx Response Control Function)" on page 515. *You also have the other registers available that comprise the sets, and function in the same manner.*

SDP Control and Status Registers

Configuration Space includes a number of general purpose registers for passing control and status information between the CPRC and the SDP Processors.

Five (5) control registers (*RxCtl_ByteSeq0*, *RxCtl_ByteSeq1*, *RxCtl_SyncSeq*, *RxCtl_BitSeq0* and *RxCtl_BitSeq1*) are allocated to communicating with the RxByte, RxBit, and RxSync Processors. The Rx processors perform byte-wide read and write operations from and to these registers under microcode control.

For complete details about specific registers go to their reference. Refer to: "RxCtl_ByteSeq0 Register (CP SDP Rx Control Function)" on page 518, Table 170 on page 518, "RxCtl_SyncSeq Register (CP SDP Rx Control Function)" on page 518, "RxCtl_BitSeq0 Register (CP SDP Rx Control Function)" on page 518, and Table 171 on page 519.

Four (4) control registers (*TxCtl_ByteSeq0*, *TxCtl_ByteSeq1*, *TxCtl_BitSeq0* and *TxCtl_BitSeq1)* are allocated to communicating with the TxByte and TxBit Processors. The Tx processors perform byte-wide read and write operations from and to these registers under microcode control.

For complete details about specific registers go to their reference. Refer to: "TxCtl_ByteSeq0 Register (CP SDP Tx Control Function)" on page 519, Table 172 on page 519, "TxCtl_BitSeq0 Register (CP SDP Tx Control Function)" on page 519, and Table 173 on page 519.

In addition, four (4) status registers (*RxCtl0_Status*, *RxCtl1_Status*,*TxCtl0_Status, and TxCtl1_Status),* two (2) each for the RxSDP and the TxSDP, contain predefined status bits used by the CPRC to track the SDP progress through cell/packet processing and vice-versa.

The CPRC process accesses all of these memory-mapped SDP control registers using load and store instructions. These registers have two (2) read and two (2) write ports, allowing CPRC and SDP access at all times. The CPRC process and SDP firmware must cooperate to ensure data integrity. For both receive and transmit, the SDP microcode sets the *Avail* bit [31] to signal end-of-frame, and thereby switch data scopes. For more information about data scopes. Refer to "Data Scope Detail Operations" on page 111.

For complete details about specific registers go to their reference. Refer to: "RxCtl0_Status Register (XP DMEM#24 Transfer Rx Control Block0 Function)" on page 621, Table 162 on page 509, "TxCtl0_Status Register (CP Tx Control Block0 Function)" on page 509, and Table 162 on page 509.

Miscellaneous Control Registers

Configuration Space includes miscellaneous control registers.

Event Registers

There are a number of events that can occur in a C-5e NP that are asynchronous, and that the CPRC must be able to process. These events must be recognized either by polling for them, or via interrupt notification. Refer to "Interrupt Access" on page 139 for more information. The C-5e NP has the capabilities to reduce the processing time required to respond to an asynchronous event. This event handling mechanism in the CPRC has the following properties:

- Software can identify events and dispatch them to their corresponding processing routines very quickly.
- Software can dynamically prioritize events.
- Software can choose which events will generate interrupts (if any), and which events it will poll.

Each of sixty-four (64) events in the CP is assigned an event number, and a corresponding bit in one of the two (2) 32bit event registers (*Event0* and *Event1*). When an event occurs in the CP (that is, the signal transitions from 0 to 1), it sets the corresponding bit in event registers.

*Most of the bits in the event registers can be interrogated and cleared independently of other state in Configuration Space. However, Event0 register bit [50] and Event1 register bit [21] are exceptions; these bits are not edge sensitive and cannot be cleared directly. Bit [21] represents the logical OR of the current bits in all of the four Queue Status registers (*Queue_Status0 *to* Queue_Status3*). Clearing the Queue Status registers clears Event1 register bit [21]. Similarly, bit [50] SONET event represents the logical OR of the masked bits in the SONET_Event register. Clearing or masking off all SONET events clears Event0 register bit [50].*

The Event Registers comprise two (2) words in the CP (*Event0* and *Event1*), and those words can be read by the CPRC and written with value 1 to clear bits. The normal mechanism for accessing the event status uses the "Event_Access Register (CP Event and Interrupt Function)" on page 557.

For complete details about specific registers go to their reference. Refer to: "Event0 Register (CP Event and Interrupt Function)" on page 552, *and* "Event1 Register (CP Event and Interrupt Function)" on page 555.

Event Access registers are a set of four (4) registers used to provide access to the *Event0* and *Event1* registers. The Event Access registers consist of: *Event_Mask0, Event_Mask1*, *Event_Access*, and *Mask_Access* register.

The *Event_Mask* defines which events the *Event_Access* responds to. It comprises two (2) 32bit registers in the CP. The event number in *Event0* and *Event1* registers is active if its corresponding bit is set in the *Event_Mask0* or *Event_Mask1* registers. This can be done at initialization time or dynamically. Individual bits can be set or cleared in *Event_Mask* by using the *Mask_Access*.

The *Event_Access* returns the logical AND and the logical NOR of the bits from *Event0/Event1* that are active. When the CP reads the value of *Event_Access*, it gets a value of 1 in bit [31] *All* field if all of the bits in the *Event0/Event1* that are set in *Event_Mask* are on. If any of the bits in the *Event0/Event1* that are active in the Event Access registers are reading, *Event_Access* returns 0 in bit [31] *All* field. This allows a program to check whether all interesting events have occurred. If no events are active in the Event Access registers, that is, the *Event_Mask*=0, reading *Event_Access* returns 1 in bit [31] *All* field. When the CP reads the value of *Event_Access*, it gets a value of 0 in bit [15] *None* field if any of the bits in the *Event0/Event1* that are set in *Event_Mask* are on. If all of these bits are 0, reading *Event_Access* returns 1 in bit [15] *None* field. This allows a program to check whether any interesting events have occurred.

The *Event_Access* also provides access to the events in the *Event_Mask0/Event_Mask1* registers, one at a time in a highest-to-lowest event number order. When a program reads *Event_Access*, bits [7:2] *EventNumber* field denotes the "highest numbered active bit", which is set in *Event0/Event1*. The *Event_Access*, bits [7:2] *EventNumber* field is positioned to allow software to use the read value directly as a word index. If *Event_Mask* & *Event0/Event1*= 0, indicating that none of the events active has occurred, reading *Event_Access* returns 0x8000 in bits [15:0] field.

While many of the bits in the *Event0/Event1* correspond to other bits in the CP, they are not directly linked to those bits. When an asynchronous event occurs in the CP, such as the *Avail* bit [31] being set in Receive Control Block 0 (RxCB0_), the corresponding bit gets set in the *Event0/Event1*. Clearing the bit in the *Event0/Event1* does not clear the *Avail* bit [31] in the *RxCB0_Ctl* register.

To clear a particular bit in *Event0/Event1*, a program writes the particular bit number into *Event_Access bits [7:2] ClearBit* field. This lets a program clear an event bit (after processing the event) by writing the same value to *Event_Access*.

To set a particular bit in *Event0/Event1*, a program writes the particular bit number into *Event_Access bits* [23:18] field. This provides a mechanism for setting a software event and having it recognized later in the event polling loop.

Another way to clear one or more bits in *Event0/Event1* is to write a mask value containing the bits to be cleared into the appropriate words of *Event0/Event1* directly. Bits in the *Event0/Event1* are "write 1 to clear".

Single bits in the *Event_Mask* can be set and cleared using the *Mask_Access*, which provides a decode mechanism similar to the one for the *Event0/Event1*. This allows an event dispatcher to dynamically change the events that are interesting to a program as the program modules progress from one stage to the next.

To clear a particular bit in *Event_Mask*, a program writes the particular bit number into *Mask_Access bits [7:2] ClearBit field*. To set a particular bit in *Event_Mask*, a program writes the particular bit number into *Mask_Access bits [23:18] SetBit* field. *Event_Mask0* and *Event_Mask1* registers are also directly writable.

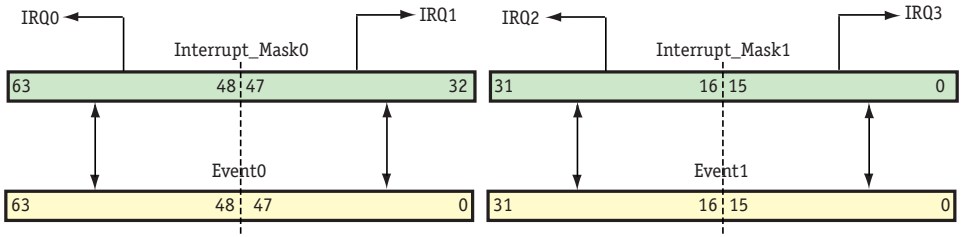
For complete details about specific registers go to their reference. Refer to: "Event_Mask0 Register (CP Event and Interrupt Function)" on page 557, "Event_Mask1 Register (for Mask1)" on page 557, "Event_Access Register (CP Event and Interrupt Function)" on page 557 and "Mask_Access Register (CP Event and Interrupt Function)" on page 559.

Interrupt Access

The CPRC implements four prioritized hardware interrupts, IRQ0-IRQ3. *Interrupt_Mask0* and *Interrupt_Mask1* registers provide a means for software to configure which events in the event register cause interrupts.

An interrupt is requested whenever a bit in the *Event0* or *Event1* register is set and its corresponding bit in the *Interrupt_Maskn* is set. Bits [63:48] in *Event0* register correspond to bits [63:48] in the *Interrupt_Mask0* register that also corresponds to IRQ0. This same type of relationship applies for IRQ1, IRQ2 and IRQ3. Refer to Figure 25 on page 140.

Figure 25 Relationship Between Interruput_Mask0, IRQ0 and Event0 Registers



For complete details about specific registers go to their reference. Refer to: "Interrupt_Mask0 Register (CP Event and Interrupt Function)" on page 559, and Table 180 on page 560.

Queue Status Registers

Queue status from the Queue Management Unit (QMU) is regularly broadcast on a side band of the C-5e NP buses. This status is automatically loaded into the four (4) queue status registers (*Queue_Status0*, *Queue_Status1*, *Queue_Status2 and Queue_Status3)* where it can be read by the CPRC. The CPRC can set bits in the queue status registers by accessing them through the update addresses. The logical OR of the bits in the status registers, provides a level-sensitive event for input to *Event1* register bit [21].

For complete details about specific registers go to their reference. Refer to: "Queue_Status0 Register (CP Queue Status Function)" on page 549 and "Queue_Statusn Registers (for Queue Status 1, 2 and 3)" on page 549.

Cycle Counter

A 64bit Cycle counter is provided in Configuration Space (*Cycle_Count_H* and *Cycle_Count_L*). The cycle counter initializes to 0 during reset and runs freely when reset is released. Thus, the cycle counters in each of the channels are synchronized. The full counter value is readable atomically by the CPRC reading two (2) registers. A copy of the top word is updated whenever the bottom word is read. Only the frozen copy of the top word can be read. For atomic access to the 64bit value, the bottom 32bit word should be read first, then the frozen top 32bit word.

For complete details about specific registers go to their reference. Refer to: "Cycle_Count_H Register (CP Miscellaneous Control Function)" on page 551, and "Cycle_Count_L Register (CP Miscellaneous Control Function)" on page 551.

Event Timer

One event timer register is provided in the Configuration Space (*Event_Timer*). The timer initializes to 0 during reset. After reset, the value in the timer always decrements once per core clock cycle. During the cycle that the timer decrements through 0, a timer event is recorded in the *Event0* register bit [52] *Time-out* field. The timer value can also be read by the CPRC. Applications can write a value into this register that decrements in the same fashion.

For complete details about specific registers go to their reference. Refer to:
"Event_Timer Register (CP Miscellaneous Control Function)" on page 550.

Understanding Block Moves of Data

Block moves are used to move data from/to the CPs to/from the BMU, or from/to the CPs to/from the QMU across the Payload Bus.This is done by using Wr, Rx, Rd and Tx features to achieve many different functions. Therefore, to use this feature you should have a basic understanding of the Payload process as described in the following sections.

Payload handling is divided into two (2) types:

- External, a data stream that is received from outside the C-5e NP and transmitted outside the C-5e NP using the Rx and Tx functions, and
- Internal, a movement of data within the C-5e NP using the Wr and Rd functions.

External Handling Overview

This is a general overview of the data movement coming into the C-5e NP. Refer to: "SDP RxByte Processor Receive Control Blocks (RxCB0_, RxCB1_)" on page 123 for more details of the Rx side, and "SDP TxByte Processor Transmit Control Block (TxCB0_, TxCB1_)" on page 128 for details of the Tx side.

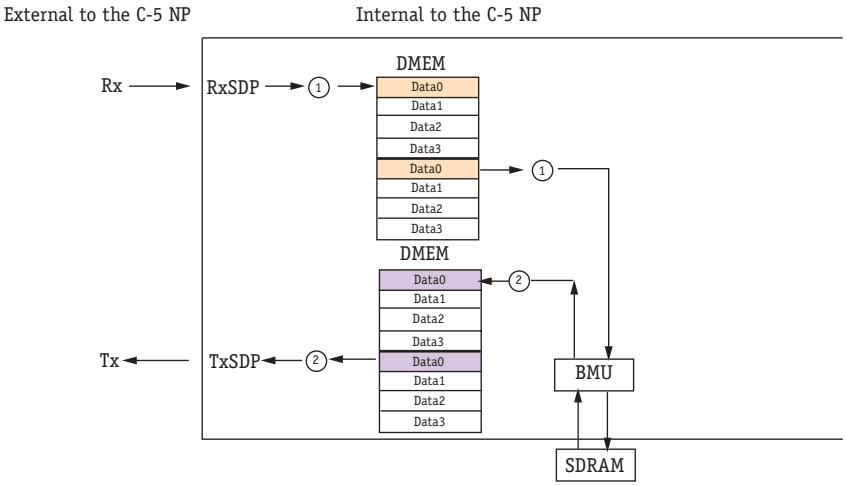
1 Payload handling process, the Rx side:

The flow of the payload handling process starts with a Rx of data from outside the C-5e NP that is processed by the RxSDP, then using the following registers (RxCB*n*_Sys_Addr, RxCB*n*_Ctl, RxCB*n*_DMA_Addr, and RxCB*n*_SDP_Addr) places data (data0) into a location inside the DMEM. The data inside DMEM is then written through the BMU, into the SDRAM for storage. This process explains why an external Rx is associated with an internal Wr. The Rx Control Blocks are used to provide block data moves across the Payload Bus from the CPs to SDRAM. Refer to Figure 26 on page 143.

2 Payload handling process, the Tx side:

The flow of the payload starts with a Tx of data from inside the C-5e NP using certain registers (TxCB*n*_Sys_Addr, TxCB*n*_Ctl, TxCB*n*_DMA_Addr and TxCB*n*_SDP_Addr) that reads the data stored in the SDRAM, through the BMU, then places the data (data0) into the DMEM that is then processed by the TxSDP to outside the C-5e NP. This process explains why an external Tx is associated with an internal Rd. The Tx Control Blocks are used to provide block data moves across the Payload Bus from the SDRAM to the CPs. Refer to Figure 26 on page 143.

Figure 26 Rx and TxCB*n*_ Handling Process Overview (for External Flow)



Rx (1)	Tx (2)
RxCBn_Sys_Addr	TxCBn_Sys_Addr
RxCBn_Ctl	TxCBn_Ctl
RxCBn_DMA_Addr	TxCBn_DMA_Addr
RxCBn_SDP_Addr	TxCBn_SDP_Addr

Internal Handling Overview

This is a general overview of the data movement inside the C-5e NP. Refer to: "Write Control Blocks (WrCB0_ , WrCB1_)" on page 116 for more details of the Wr side, and "Read Control Blocks (RdCB0_ , RdCB1_)" on page 120 for more details of the Rd side.

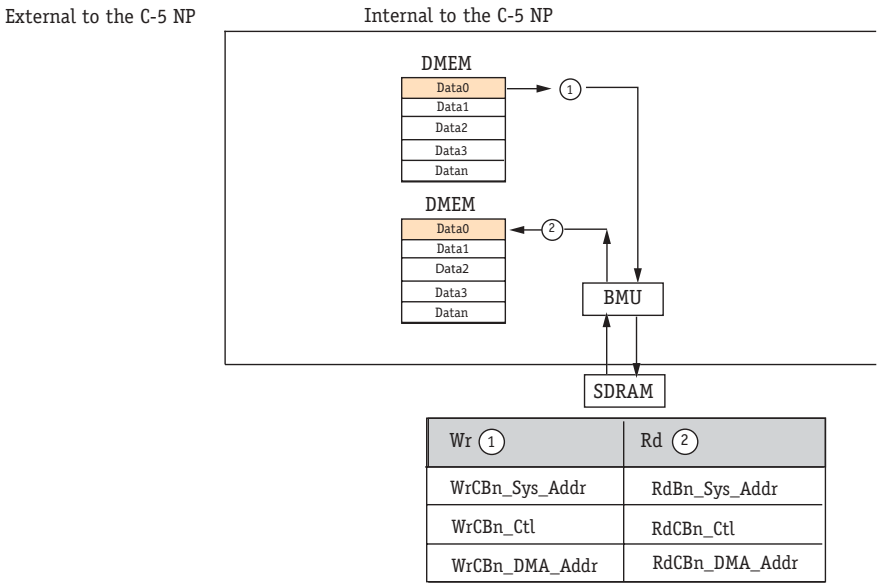
1 Payload handling process, the Wr side:

The flow of the payload handling process starts with a Wr of data from inside the C-5e NP using (WrCB*n*_Sys_Addr, WrCB*n*_Ctl, and WrCB*n*_DMA_Addr) that takes data (data0) from the DMEM and then writes it through the BMU into the SDRAM for storage. The Wr Control Blocks are used to provide block data moves across the Payload Bus from the SDRAM, QMU or BMU to the CPs. Refer to Figure 27 on page 144.

2 Payload handling process, the Rd side:

The flow of the payload handling process starts with a Rd of data from inside the C-5e NP using (RdCB*n*_Sys_Addr, RdCB*n*_Ctl, and RdCB*n*_DMA_Addr) that reads the data stored in the SDRAM, through the BMU then places the data (data0) into the DMEM. The Rd Control Blocks are used to provide block data moves across the Payload Bus from SDRAM to the CPs. Refer to Figure 27 on page 144.

Figure 27 Wr and RdCB*n*_ Handling Process Overview (for Internal Flow)



Wr (1)	Rd (2)
WrCBn_Sys_Addr	RdBn_Sys_Addr
WrCBn_Ctl	RdCBn_Ctl
WrCBn_DMA_Addr	RdCBn_DMA_Addr

Using Multi-Use Control Blocks to Achieve Different Functions

The Multi-Use Control Blocks (WrCB0_Sys_Addr, WrCB0_Ctl, WrCB0_DMA_Addr; RxCB0_Sys_Addr, RxCB0_Ctl, RxCB0_DMA_Addr, RxCB0_SDP_Addr; RdCB0_Sys_Addr, RdCB0_Ctl, RdCB0_DMA_Addr; and TxCB0_Sys_Addr, TxCB0_Ctl, TxCB0_DMA_Addr, TxCB0_SDP_Addr) can be programmed to make data moves to/from SDRAM, the BMU, or the QMU. All of these registers physically reside in the CP memory map at their respective addresses.

The individual fields of these registers are used to perform different functions. Refer to Table 23 on page 145. Detail examples of each, including actual field bit values, are shown in other locations of this manual as noted in this table.

In addition to the Mutli-Use Control Blocks (for Wr, Rx, Rd and Tx), eight (8) Fixed-Use Control Blocks (for Wr and Rd) are provided as programming short-cuts for various common operations. Refer to Table 24 on page 147.

Table 23 Multi-Use Control Blocks (for Wr, Rx, Rd and Tx)

MODE	CATEGORY	FUNCTION	FIELDS USED	DETAILS
CP to/from BMU	Memory Transactions	Buffer Memory Transfer Operation	PoolID, BTag, Offset	See "Write Control Blocks (WrCB0_ , WrCB1_)" on page 116 See "SDP RxByte Processor Receive Control Blocks (RxCB0_, RxCB1_)" on page 123 See "Read Control Blocks (RdCB0_ , RdCB1_)" on page 120 See "SDP TxByte Processor Transmit Control Block (TxCB0_, TxCB1_)" on page 128
CP to/from BMU	BTag Management Transactions	Initializing BTags	PoolID, BTag, Command, Pool	See "BTag Initialization Operation" on page 292.
		Allocating BTags		See "BTag Allocation Operation" on page 295.
		Deallocating BTags		See "BTag Deallocation Operation" on page 297.
	Multi-Use Management Transactions	Allocating (Multi-Use Counter)		See "MUC Allocation Operation" on page 300.
		Decrementing (Multi-Use Counter)		See "MUC Decrement Operation" on page 303.
		Reading (Multi-Use Counter)		See "MUC Read Operation" on page 305.

Table 23 Multi-Use Control Blocks (for Wr, Rx, Rd and Tx) (continued)

MODE	CATEGORY	FUNCTION	FIELDS USED	DETAILS
CP to/from QMU	Queue Management Transactions	Configure Queue	Mail Box#, Queue#, Command, PoolID	See "Configure Queue Operation" on page 431.
		Queue Status		See "Queue Status Operation" on page 433.
		Unicast Enqueue		See "Unicast Enqueue Operation" on page 435.
		Multicast Enqueue	Mail Box#, QueueLevel#, Command, PoolID	See "Multicast Enqueue Operation" on page 439.
		Dequeue	Mail Box#, Queue#, Command, PoolID	See "Dequeue Operation" on page 441.

Table 24 Fixed-Use Control Blocks (for Wr and Rd)

MODE	CATEGORY	FUNCTION	RD/WR CB USED	DETAILS
CP to/from BMU	BTag Management Transactions	Allocating BTags	Rd	See "RdCB0_BTag_Alloc (CP Rd Control Block0 Fixed Function)" on page 569.
		Deallocating BTags	Wr	See "WrCB0_BTag_Deallocate (CP Wr Control Block0 Fixed Function)" on page 571.
	Multi-Use Management Transactions	Allocating (Multi-Use Counter)		See "WrCB0_MUC_Allocate (CP Wr Control Block0 Fixed Function)" on page 571.
		Decrementing (Multi-Use Counter		See "WrCB0_MUC_Decrement (CP Wr Control Block0 Fixed Function)" on page 572.
CP to/from QMU	Queue Management Transactions	Unicast Enqueue		See "WrCB0_Uni_Enq (CP Wr Control Block0 Fixed Function)" on page 573.
		Speculative Unicast Enqueue		See "WrCB0_Spec_Uni_Enq (CP Wr Control Block0 Fixed Function)" on page 575.
		Multicast Enqueue		See "WrCB0_Multi_Enq (CP Wr Control Block0 Fixed Function)" on page 574.
		Dequeue	Rd	See "RdCB0_Dequeue (CP Rd Control Block0 Fixed Function)" on page 570.

The details column in Table 24 on page 147 goes directly to the individual register parameters, not to the BMU or QMU chapter like in Table 23 on page 145 since these are merely programming short-cuts.

C-5e Methods for Handling High Speed (OC-48) PDUs

This section pertains to the three (3) C-5e methods used to implement higher line speeds that apply to the *CPs only*. Some are used in conjunction with an external devices (M-5). Table 25 on page 148 lists the methods and some related aspects as an overview. Each of these methods are described in the following section.

Table 25 C-5e Methods and Purpose in Relation to Components, Operation, and External Companion Devices for CPs Only

C-5E METHODS	PURPOSE	APPLICABLE C-5E COMPONENT	APPLICABLE OPERATION	USED WITH
Sequence Numbers	Provides ordering inside the C-5e that enables extracting and merging of the associated descriptors with their payloads.	CPs	Enqueue and Dequeue	M-5
Aggregated Queueing	Provides two benefits: allows clustered CPs to share queues, and allows all 16 CPs to share a single queue in order to transmit a single concatenated stream. Both are used in order to reduce latency when operating at higher line speeds.	CPs	Dequeue Only	N/A
Speculative Enqueue	Allows a fixed latency for PDU streams, from the start to enqueue. It prevents uneven PDU flows that can cause overruns of PDUs, and large gaps in the streams. This provides a more efficient bandwidth of PDU flow. The enqueues are speculative because at the time they are enqueued, the CRC is *not* calculated.	CPs/QMU	Enqueue Only	M-5 Optional

Sequence Numbers for CPs

PDUs are received and transmitted into the M-5 out of order. Therefore, sequence numbers are used to provide ordering inside the C-5e that enables extracting and merging of the associated descriptors with their payloads. Sequence numbers are needed for the CPs for both enqueue and dequeue operations. Sequence numbers are 13bits long and are included in enqueue and dequeue operations once they are configured.

For CP enqueue and dequeue operations, the *SeqNum* bit [7], of the Payload address [27:0], indicates if sequence numbers are to be used in the request or not (1=use sequence numbers, 0=do not use sequence numbers).

Enqueue Operations Using Sequence Numbers

When sequence numbers are used for the CPs, the QMU serves its request in the following manner. The QMU alternates its preference between serving those in the normal FIFO order, and those with sequence numbers.

Occasionally, PDUs are dropped that create gaps in the sequence numbering. This causes an error that is taken into account in the new algorithm. Therefore, a saturating count of the missing sequence numbers that should have been enqueued to the QMU from the CPs are collected in the new *Missing_Front_Seq_Num_Cnt* register bits [31:0].

The new enqueue algorithm is used with the CPs. The descriptor is enqueued in the C-5e. Using the new *Front_Seq_Num* register bits [28:16] *front ingress sequence number* field, the QMU looks here to obtain the next sequence number for enqueuing descriptors from the front ports (CPs).

Error Handling and Error Conditions

Error handling is provided for occasional missing sequence numbers and is handled as follows:

When all mailboxes are full, but the expected sequence number does not appear, then the expected sequence number is incremented. The new *Missing_Front_Seq_Num_Cnt* register bits [31:0] count these events.

Three (3) error conditions can cause out-of-sync count between QMU and M-5. Under these conditions, while the QMU counts through sequence numbers to regain synchronization, large numbers of PDUs could be lost. Such events rarely occur. These conditions are as follows:

- After the C-5e NP is reset, the first sequence number seen by the QMU is far from zero (0). This would cause acceptable loss at startup.
- There is a missing sequence number, and shortly thereafter there is a very large PDU followed by a long stream of very small PDUs. While the large PDU is arriving, other write mailboxes in the same cluster cannot be filled by the CP. Due to the empty mailboxes, the QMU's expected sequence number does not increment past the missing number. Enqueues stop while the many small PDUs arrive, causing overrun of the CP FIFOs.
- The QMU receives a corrupted sequence number.

The enqueue algorithm assumes that each CP must eventually enqueue a descriptor. When any CP goes off-line, all enqueueing stops.

Dequeues Operations Using Sequence Numbers

Both the front ports (CPs) and the back port (FPTx) have separate sequence number spaces.

- Using the new *Front_Seq_Num* register bits [12:0] *front egress sequence number* field, the QMU looks here to obtain the next sequence number to supply with a descriptor to send to the front ports (CPs).
- Using the new *Back_Seq_Num* register bits [12:0] *back egress sequence number* field, the QMU looks here to obtain the next sequence number to supply with a descriptor to send to the back port (FP). New enqueue/dequeues formats exist using the Unicast Enqueue Operation (WrCB).

Aggregated Queueing for CPs

The purpose of aggregated queuing is two fold: to allow clustered CPs to share queues, and to allow all 16 CPs to share a single queue in order to transmit a single concatenated stream. Both are used in order to reduce latency when operating at higher line speeds.

Each CP has three (3) hardware operational modes:

- Single CP Queuing Mode (00), (1 CP)
- Cluster Aggregation Mode (01), (4CPs clustered) (128 Queues are available)
- Full-Chip Aggregation Mode (10), (all 16 CPs clustered) (128 Queues are available)

Except for full-chip aggregation, other possible mixes are allowed. For example, 1 CP in cluster aggregation, while another is in single CP queuing mode. However, all CPs in a cluster must be in the same mode.

In all three (3) modes the queue status register is now a true reflection of the QMUs queue empty/not-empty status. For all three (3) modes empty to not-empty (0 to 1) transactions are distributed in the same manner, via QMU broadcasts on the Global Bus. During the aggregation queuing the CPs masks the appropriate lower bits of the CPId to make the distribution across four (4) or sixteen (16) CPs. The Global Bus and Payload Bus use the same CPId bit masking. For all three (3) modes a not-empty to empty (1 to 0) transactions are detected over the Payload Bus. These transactions are watched for all dequeues and note queue lengths of zero (0). A queue length of zero (0) causes the associated queue status bit to clear to empty (zero).

While the QMU issues special broadcasts for the 0 to 1 transactions, the 1 to 0 transactions are communicated indirectly via the queue length that is embedded in a dequeue response.

- In single CP Queuing Mode, the operation is essentially the same as described above, (empty to not-empty and not-empty to empty) except there is no bit masking. Each CP acts independently.
- In Cluster Aggregation Mode, 4 CPs are aggregated and each cluster uses tokens for pipelining the transmit operation. The tokens are *not* used for ordering, this is done by the M-5. A total of 128 queues can be used by the aggregated CPs.
- In Full-Chip Aggregation Mode, the 16 CPs are aggregated for pipelining the transmit operation. Ordering, is done by the M-5 in cooperation with the QMU. A total of 128 queues can be used by the aggregated CPs.

Queue Length and Queue Status Trade-Offs

The use of queue lengths for the purpose of determining whether or not a dequeue should be issued are optional based upon the specifics of the application. Bear in mind, that this is an explicit trade-off between performance and functionality.

Changes in the Dequeue Paradigm

When the queue status indicates the queue is non-empty (>0) then a dequeue is issued. This is speculative since the software is not sure if the queue actually has a descriptor or not. This is because other CPs in other clusters may be looking at the same queue.

When there is *no* descriptor for the requesting CP, the QMU handles the dequeue request by setting the *dry bit*. A dry bit occurs when the requesting CP tried to dequeue an empty queue. A *dry dequeue* does not create a Payload Bus error and the mailbox status bits from the QMU indicates idle (00). Therefore, the dry dequeue no longer results in an error. The CP should either re-try or try to dequeue form another queue. The result is that an empty to not-empty transition, a transient number of CPs might see the queue status bit and dequeue simultaneously even though there may only be one (1) PDU to transmit. The number of transient CPs should be four (4) due to the dequeue token in the clusters. In this case, one (1) CP gets a descriptor and three CPs (3) get a dry queue message.

Implementation of Aggregated Queueing for CPs

The queue aggregation mode is selected using the new *Queue_Ctl* register bits [1:0] *QueueAggrMode* field. This register is located in the CP configuration space. Select the desired mode using the supported values. They are:

Table 26 Legal Values for Queue Aggregation for CPs

ENCODED VALUE	FUNCTION
00	Single CP Queuing (x1) (1 CP)
01	Cluster Aggregation Mode (x4) (4CPs clustered)
10	Full-Chip Aggregation Mode (x16) (all 16 CPs clustered)
11	Reserved

These bits exist in all CPs. Both the Payload Bus and Global Bus logic use these modes to select a wildcard on bits of the nodeID during a QMU queue status broadcast message and during QMU dequeue payload reads.

The QMU returns the encoded queue status number, (dequeue queue number - base queue number), in bits [20:14] of the first 16Bytes (128bits) of the dequeue read payload. Each Payload Bus detects dequeues aimed with the mode mask at its CP. When a match occurs, the Payload Bus creates a write to the new *Queue_Empty* register containing bits [20:14] *encoded queue status number* field and bits [13:0] *queue length* field. This is then used by the CP logic to clear a bit in the *Queue_Status* registers when the queue length== 0.

Hardware updates (that is, setting the bits) of the Queue_Status register can not occur during the hardware clear mechanism because of the bus timing.

Speculative Enqueues for CPs

Speculative Enqueues allow a fixed latency for PDU streams, from the start to enqueue. It prevents uneven PDU flows that can cause overruns of PDUs, and large gaps in the streams. Thus, providing a more efficient bandwidth of PDU flow. The enqueues are speculative because at the time they are enqueued, the CRC is *not* calculated. Therefore, the PDUs are accepted into the CPs faster because the PDUs are treated as valid upon reception. Erroneous PDUs are detected and dropped on transmission by the output port.

Speculative enqueues do not support multicast PDUs. Therefore, only non-speculative enqueues should be used for multicast PDUs. Furthermore, the XP does not support speculative enqueues. When attempted the CPI returns an error.

Operation of Speculative Enqueues

At a fixed time after processing starts, the Rx CP builds a descriptor that includes the PDU length that is calculated by the M-5 and then sends it (in a speculatively state) to the QMU. Later when the entire PDU has been received, a 2bit commit message, in *CP_Mode0* register bit [25] field *SendSpeculCommit* and bit [24] field *Valid/Invalid, are* sent to the QMU that indicate whether the PDU is valid. If it indicated an error then it would normally cause the PDU to be dropped. When an error is detected, the *Invalid* bit [24] is then used in the transmit path. Refer to Table 27 on page 153.

Table 27 Commit Message Format for the Commit Serial Line

FIELD NAME	BIT POSITION	DESCRIPTION
SendSpeculCommit	25	1= Initialize transfer 0= No action
Valid/invalid	24	0= Commit, valid descriptor 1= Commit, invalid descriptor

At commit time, the descriptor and BTag have already been stored in internal structures and cannot be deleted. Consequently it is up to the transmit port to drop the descriptor and deallocate or reuse the BTag.

When the head of a queue is not committed then the queue appears to be empty. The following provides more detail on this:

- When an uncommitted descriptor is enqueued in an otherwise empty queue then the queue ready (empty to non-empty) message is *not* issued.
- When a descriptor is dequeued and the next descriptor in that queue is uncommitted, then the queue length supplied with the dequeued descriptor is zero (0).

- When an uncommitted descriptor is at the head of the queue, then attempted dequeues result in a “dry queue” response.
- When a descriptor at the head of a queue is committed, a queue ready message is broadcast.

When dequeue occurs the transmit CP must examine the invalid bit and if the PDU is invalid then the PDU *must* be dropped. Since a commit message is under software control the QMU must be able to tolerate spurious commits in a graceful manner. Spurious commits mean commits that have no corresponding speculative enqueue. When a commit arrives at a mailbox and is *not* expected then the commit is simply ignored. This refers to a mailbox not containing an uncommitted speculative enqueue.

The speculative enqueue process is achieved using dedicated serial lines between each of the CPs and QMU. Serial status bits from the QMU mailboxes to the CPs function as described in the QMU section. This means that a mailbox status may be idle even though it is waiting for a commit.

The CP must send the commit message *after* the associated speculative enqueue, but *before* sending the next enqueue to the mailbox. If a mailbox receives another command while it is still waiting for a commit, the latter command is NACKed.

When using speculative enqueues, there must be a one-to-one association between CPs and write mailboxes. Specifically, CPn must only send enqueues to mailboxn.

Dequeue messages from the QMU include the invalid bit on the first beat of the Payload Bus.

Implementation of Speculative Enqueues for CPs

- A new QMU function called Speculative Unicast-Enqueue is available using a WrCB0_ (0x3) to provide CP to/from QMU payload transactions.
- Also, the two dedicated serial lines between each of the CPs and the QMU is used in the following manner: one is used for both read and write mailbox status from the QMU, while the other is used for speculative enqueue commit messages to the QMU.

Software launches the speculative commit transmission by writing the commit message, a 2bit commit message, in *CP_Mode0* register bit [25] field *FrameBit* and bit [24] field *Valid/Invalid*. The actual writing of these 2bits causes the CP to send the commit message to the QMU.

Chapter 3

EXECUTIVE PROCESSOR

Chapter Overview

This chapter covers the following topics:

Executive Processor (XP) Overview

The XP serves as a centralized computing resource for the C-5e NP and manages the system interfaces. One of the system interfaces it manages is the PCI bus, which is generally used for communication to an external *host processor*. If present, a host processor can provide device-wide coordination (for example, between multiple C-5e NPs), network management, signaling, and could possibly build all routing tables for the device of which the C-5e NP is a part. The XP can also perform many of these functions by itself. The XP has access to the internal Global, Ring, and Payload buses.

Typical XP functions include:

- Chip initialization and code download
- Routing/Switching table maintenance (either building tables or importing updates from the host)
- Statistics harvesting from CP DMEM and the TLU
- Fault detection/recovery
- Non-critical-path forwarding functions

XP Major Components

The major components of the XP are listed in Table 28 on page 156. In addition, Figure 28 on page 158 shows the XP Block Diagram.

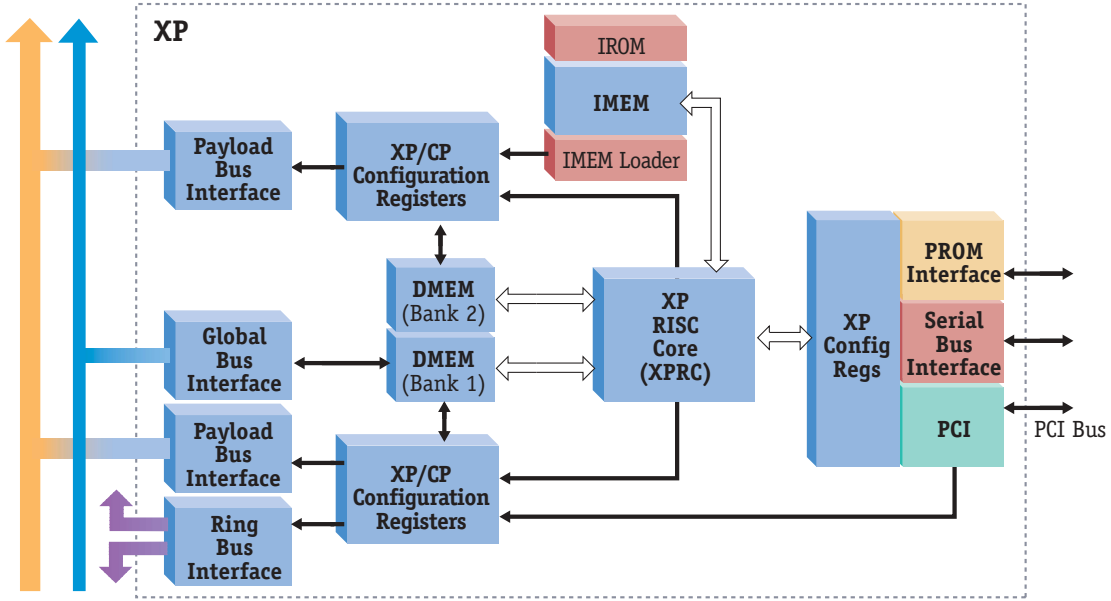
Table 28 Major Components of the XP and Their Function

ITEM	FUNCTION
XP RISC Core (XPRC)	Performs conventional supervisory tasks in the C-5e NP, including: • Reset and initialization of the C-5e NP • Program loading and control of CPs • Centralized exception handling • Management of a host interface through the PCI • Management of system interfaces (PCI, PROM, Serial Bus) This general purpose CPU implements a subset of the MIPS 1 instruction set (multiply, divide, floating point, and CPO instructions are not supported) with its own dedicated code and data store. The XPRC has Global Bus access to all CP configuration registers and DMEMs. In addition, the XPRC has Ring Bus access for table lookup operations. A 16-word Instruction ROM (IROM) is dedicated to the XPRC. Refer to "XP RISC (XPRC) Overview" on page 159.

Table 28 Major Components of the XP and Their Function (continued)

ITEM	FUNCTION
Memory	Two (2) types of memory are available: IMEM and DMEM. • The XP has 48kBytes of IMEM that contains the RISC Instructions in RAM. It is organized as two (2) 24kBytes banks for sharing within the XP. • The XP has 32kBytes of local non-cached data memory (DMEM) for storage of data. It is organized as two (2) 16kBytes banks. In addition, the DMEM can also be accessed as remote memory by CPs via the Global Bus. Refer to "XP Memory (IMEM and DMEM)" on page 165.
PCI	Provides an industry standard 32bit 33/66MHz PCI channel used for chip-level shared resources. The PCI has both *initiator* and *target* capabilities. A host is optional, but when present, it is capable of: • Requesting the Global Bus (which provides access to all CP configuration registers and DMEMs) • Requesting the Ring Bus (which provides access to table lookup operations) • Requesting XP processing and communicating with the XP for additional services • Supporting C-5e NP initialization Refer to "PCI Bus Interface" on page 167.
PROM Interface	Allows the XP to boot from an external PROM. The PROM interface is a low-speed, serial I/O interface that requires external glue logic to interface to an external PROM up to 4MBytes in size. Refer to "PROM Interface" on page 169.
Serial Bus Interface	Consists of a general purpose bi-directional, two-wire serial bus and I/O port. It allows the C-5e NP to control external logic with either of two (2) standard protocols. • The high-speed protocol uses a 16bit data format with 10bits of addressing, and supports transfers up to 25MHz. • The low-speed protocol uses an 8bit data format followed by an acknowledge bit and supports transfers at up to 400kbps. The bus supports a single master hierarchy that can operate as either a receiver or a transmitter. The bus also supports an integrated addressing and data-transfer protocol. Refer to "Serial Bus Interface" on page 171.
Configuration Space	This area of the XP contains a number of registers used to communicate with the SDP and the bus controllers (Payload Bus and Global Bus). The XP's registers can also be accessed by other components of the C-5e NP.(CPs via the Global Bus). Refer to "XP Configuration Space" on page 175.

Figure 28 Executive Processor Block Diagram



XP RISC (XPRC) Overview

The XPRC is a general purpose Central Processing Unit (CPU) founded on the same RISC Core used for the CP. Operating at the C-5e NP's core clock rate, the XPRC provides about.85 instructions per cycle (IPC) when executing out of local memory. The IPC and frequency targets offer about 190MIPS per channel on non-blocking code.

The XPRC contains a 32bit data path and accesses memory using a 32bit physical address. It has two (2) banks of local data memory (DMEM); references to memory within Bank 2 (also referred to as DMEM 25) occur with zero wait states; accesses to Bank 1 (also referred to as DMEM 24) incur one core clock cycle latency. Memory addresses outside of local memory range refer to remote memory (that is, the memory contained within the CPs, SDRAM, or I/O devices).

The XP contains memory-mapped control registers (blocks) used for DMA between DMEM and SDRAM, between PCI and SDRAM (via DMEM 24), as well as between SDRAM and IMEM (via DMEM 25). In addition, Configuration Registers enable the XPRC (and PCI interface) access to Payload, Global, and Ring Buses.

XPRC Instruction Set

The XPRC executes the following instruction set: a subset of MIPS™1 instruction set (excluding multiply, divide, floating point, unaligned loads and stores, move to hi and move to lo), eight (8) Branch Likely instructions from the standard MIPS™ 2 instruction set, and sixteen (16) custom instructions. Refer to "RISC Core Enhancements" on page 804. Its four (4) sets of 32 registers each support fast context switching. See the *MIPSpro™ Assembly Language Programmer's Guide* (available over the Internet at http://www.mips.com/publications/index.html) for information about the standard MIPS1 instruction set.

The standard MIPS Coprocessor Zero (CP0) registers are not supported. However, Freescale provides its own special purpose Coprocessor Zero registers.

It is highly recommended that you use the C-Ware Compiler when building your application code. Therefore, refer to the C-Ware Application Development Guide for information on using the Freescale compiler, which supports the CPRC.

XPRC Registers

The set of internal XPRC registers is defined in Table 29.

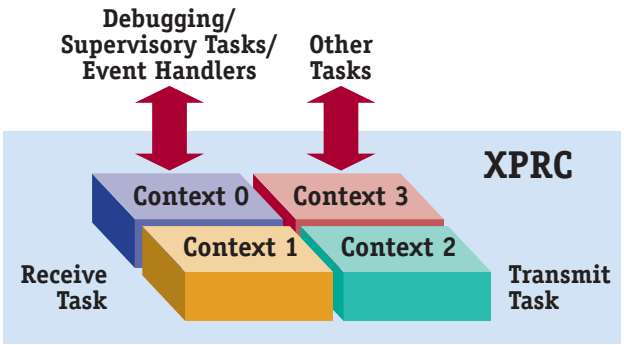
Table 29 Internal XPRC Register Definitions

REGISTER NAME	SOFTWARE NAME	USE AND LINKAGE
$0	—	Always has the value of 0.
$at or $1	—	Reserved for the assembler.
$2:$3	v0 to v1	Used for expression evaluations and for hold integer function results. Also used to pass the static link when calling nested procedures.
$4:$7	a0 to a3	Used to pass the first four words of integer type actual arguments. Their values are not preserved across procedure calls.
$8:$15	t0 to t7	Temporary registers used for expression evaluations, Their values are not preserved across procedure calls.
$16:$23	s0 to s7	Saved registers. Their values must be preserved across procedure calls.
$24:$25	t8 to t9	Temporary registers used for expression evaluations. Their values are not preserved across procedure calls.
$26:$27 or $kt0:$kt1	k0 to k1	Used internally by the C-5e NP system services.
$28 or $gp	gp	Contains the global pointer.
$29 or $sp	sp	Contains the stack pointer.
$30	s8	A saved register (like s0 - s7).
$31	ra	Contains the return address used for expression evaluation.

Context Switching

The XPRC incorporates a fast, four-way, context switching facility that replicates the entire XPRC register space four times and can switch from one register set (one context) to another under software control or hardware interrupt. Thus, actual processing (as opposed to manually saving the contents of one set of registers and then loading another) can begin on a different context in only two cycles. Therefore, you can use these four contexts for debugging, supervisory tasks, event handlers, or other tasks.

Figure 29 Executive Processor Context Switching



The XPRC includes four sets of 32 internal registers. Each register set is associated with a processor context. The set of registers are defined in Table 29.

Context switching is accomplished two (2) ways:

- Coprocessor instruction (software)
- Interrupt (hardware)

The software mechanism for executing a context switch is the MIPS MTC0 instruction:

```
MTC0 $1 $3
```

where $1 specifies the destination context. The contexts have no priority; how they are used is entirely designated by software.

The hardware interrupt sequence is:

- All interrupts are disabled until an RFE instruction is executed.
- The address of the next instruction to be executed in the interrupted context is saved in K1 (see "Interrupts").
- Program execution continues with the instruction at the address specified in the interrupt vector.

Interrupts

The XPRC supports four prioritized hardware interrupts, that can be triggered from any bits in the Event Register. There are four MIPS-like register sets corresponding to each hardware context, one register of which (K0) is shared between the other contexts.

K1 contains the program counter value and the context number of the interrupted context. These values are used in the execution of the RFE instruction to return to the previously interrupted context.

All interrupts and exceptions transfer control to a location found in the appropriate interrupt or break table. The base address of the interrupt table is specified by the contents of the interrupt table register ($1) in coprocessor zero. The base address of the exception table is specified by the contents of the break table register ($2) in coprocessor zero.

Interrupts are dispatched by a jump to the address equal to ((interrupt number * 8) + (interrupt table register)). Exceptions are dispatched by a jump to the address equal to ((break number * 8) + (break table register)). In addition to the jump, the register context is set to zero and interrupts are disabled. However, exceptions may still occur. Whether a hardware interrupt or an exception, the interrupted routine's register context and its next program counter are saved in K1 of context zero.

The K1 value points at the next instruction to be executed after the interrupt is serviced. RFE is normally used to: (1) resume the instruction flows at this point, (2) restore the proper register context, and (3) restore the Interrupt Enable Flag to its value at the time of the interrupt or exception.

Note that interrupts are not recognized in a branch delay slot. Also note that all exceptions fill the delay slot following a change of flow with a NOP instruction.

Interrupts are enable by setting the Interrupt Enable Flag (IEF) which is the LSB of coprocessor zero, Register 8 (see Table 30). The IEF is preserved whenever an exception or an interrupt occurs and is restored by the RFE instruction.

Table 30 Coprocessor Zero Register Definitions

REGISTER	DEFINITION
R0	**Whoami Register** — Contains the DMEM base (hardcoded) for this XPRC.
R1	**Interrupt Table Register** — Contains the vector address for INT 0.
R2	**Break Table Register** — Contains the vector address for break 0.
R3	**Current Context Register** — The two LSBs are the current context register. Set by setting ictxt in decoder.v.
R4	**DMEM Comparison Address** — Contains the address at which debug pulse is generated.
R5	**DMEM Comparison Address Mask** — Contains the mask for the DMEM address.
R6	**DMEM Comparison Data** — Contains the data value for which debug pulse is generated.
R7	**DMEM Comparison Data Mask** — Contains the mask for the DMEM data.
R8	**Interrupt Flag** — The LSB in the Interrupt Flag.
R9	**Read/Write Mask** — The two LSBs are the Read mask and the Write mask for R4 to R7.

Hardware Programming Resources

In addition to fast Context Switching, the XPRC contains resources to aid in efficient program design. These include:

- **Event Registers** — Each bit indicates that the corresponding event has or has not occurred since last set. Centralizing status monitoring into a single register allows for efficient event-driven software design. Many bits are pre-defined, providing high-speed reporting of events between on-chip subsystems (for example, data available on QMU queue). Other bits are software programmable.
- **Cycle Counter** — This 64bit counter is set to 0 when the chip is reset, and increments every core clock cycle thereafter. at overflow, the counter wraps to 0.
- **Countdown Timer** — Applications can set this timer to a value that decrements. When the timer reaches 0, it generates an event for the application.

Event Registers

There are a number of events that can occur in a C-5e NP that are asynchronous, and that the XPRC must be able to recognize and process. These events must be recognized either by polling for them, or via interrupt notification. To reduce the processing time required to respond to an asynchronous event (and hence to improve latency and reduce the chance of losing an event), this event handling mechanism in the XPRC has the following properties:

- Software can identify events and dispatch to their corresponding processing routines very quickly, on the order of 5 to 10 cycles.
- Software can dynamically prioritize events.
- Software can choose which events will generate interrupts (if any), and which it will process via polling.

Each of 64 events in the XP is assigned an event number, and a corresponding bit in one of the two (2) 32bit event registers (*Event0* and *Event1*). When an event occurs in the XP (that is, the signal transitions from 0 to 1), it sets the corresponding bit in event registers. The normal mechanism for accessing the event status uses the Event Access Control Block.

Most of the bits in the event registers can be interrogated and cleared independently of other state in Configuration Space. However, Event0 *register bits [55:52] are an exception; these bits are not edge sensitive and cannot be cleared directly. They represent the logical OR of the current bits in each of the Queue Status registers (*Queue_Status0 to Queue_Status3*). Clearing the Queue Status registers clears these* Event0 *register bits [55:52]. Refer to "*Executive Processor (XP) Configuration Registers*" on page 576*.

XP Memory (IMEM and DMEM)

The XP has both local instruction memory (IMEM) and local data memory (DMEM). These are local memory, not a level 1 cache. In addition, it has the capability to bring in overlays from SDRAM to either IMEM or DMEM, using DMA under program control. The XP also has a local Instruction ROM (IROM).

Instruction Memory

The XP has 48kByte IMEM, configured as two (2) sub-arrays. This memory is shared two (2) ways between the XP and the IMEM loader. The IMEM loader is a logical block that moves code overlays from SDRAM to IMEM. Using the Code Overlay Transfer Control Block, the IMEM loader can DMA code from SDRAM into IMEM via an intermediate buffer in DMEM bank 2 (DMEM #25).

XPRC instruction references outside of the local memory space are not supported. Similarly, the XP IMEM is not visible to any other processors on the chip or to the PCI interface.

Data Memory

The XP has a local 32kByte DMEM. This is organized into two 16kByte banks; bank 2 (DMEM #25) is accessed with zero latency; bank 1 (DMEM #24) is accessed with one additional cycle of latency.

DMEM is organized as 16Byte lines providing 3.2GBps peak bandwidth through a single port. It is accessed via a 4Byte (32bit) access path. The memory resides in the global address space of the C-5e NP; however, only Bank 1 (DMEM24) is accessible by CPs; DMEM Bank 2 (DMEM #25) is not visible to processors outside of the XP. Bank 2 does, however, interface to the Payload bus for data and code transfers, as well as the PCI Bus. Refer to "Executive Processor (XP) Configuration Registers" on page 576.

SDRAM

The XP has DMA access to SDRAM to support data transfers to/from the PCI, IMEM code overlays, and DMEM data overlays. All DMA is controlled using Control Blocks (WrCB0_, RdCB0_, RxCB0_, TxCB0_). SDRAM is not addressable in the global address space. The XP's control blocks provide the following types of SDRAM access:

- DMA to/from DMEM Bank 1 and SDRAM for data overlays.
  [control block RdCB/WrCB #24]
- DMA to/from DMEM Bank 2 and SDRAM for data overlays.
  [control block RdCB/WrCB #25]
- DMA to/from the PCI bus and SDRAM (via buffer in DMEM bank 1).
  [control block TxCB/RxCB #24]

The transfer control block presents transaction requests to the XP Outbound Transaction State Machine, which competes for access to the PCI Master in the XP Outbound Transaction Arbiter. The PCI address and transfer count information for the DMA transfer are provided via additional configuration registers in the XP Configuration Register Block.

- DMA from SDRAM to IMEM (via buffer in DMEM bank 2) for code overlays. [control block TxCB #25]
- DMA from a constant zero data to SDRAM. This is used to initialize SDRAM. [control block RxCB #25]

The transfer control block presents transactions to the IMEM Loader that interfaces directly into the IMEM. IMEM target address information for the DMA transfer is provided via additional configuration registers in the XP Configuration Register Block.

IROM The IROM provides the first instructions when the chip is initialized. It is only accessible by the XPRC. See "C-5e NP Interface Options for Initialization" on page 172 for more information.

XP Supported Interfaces

The XP manages the supervisory controls for the network interfaces as well as the set of pins that provide interfaces to other components in the system that are not memories or network interfaces. The XP supports three (3) system interfaces:

- 32bit PCI Interface (33MHz or 66MHz)
- PROM Interface
- Serial Bus Interface

PCI Bus Interface

Host communication to the C-5e NP is provided through the PCI interface. A host is optional, but when present, it is capable of requesting the Global Bus through the PCI interface. Using the PCI interface, a host can request XP processing through the PCI mailbox registers and communicate with the XP for additional services. A host is capable of supporting C-5e NP initialization without a ROM.

The XP can be configured to support a 32bit PCI interface capable of operating at either 33MHz or 66MHz. The PCI interface on the C-5e NP is fully compliant with the PCI Specification Revision 2.1. The C-5e NP PCI interface includes the following functions:

- Initiation of PCI transactions as a PCI Bus Initiator including:
  - Memory Reads and Writes
  - Internal DMA engines capable of transferring blocks of data between the C-5e NP's SDRAM and the PCI Bus under XP control
- Processing PCI transactions as a PCI Bus Target including:
  - Memory Writes
  - Memory Reads, Memory Read Line, and Memory Read Multiple
  - Configuration Read and Write
  - Single Delayed Transaction
  - Medium DEVSEL timing
  - Configurable via the PCI Interface and/or internal bus accesses from the XP
  - 32bit Addressing
  - 32bit Transfers

- 33MHz or 66MHz operation
- Support for a single PCI interrupt line (*PCI_INTA*)

The PCI Bus interface does *not* include support for the following functions:

- Exclusive accesses controlled by the *PCI_LOCK_N* signal as either an Initiator or a Target (all requirements for access exclusion to memory space within the C-5e NP are assumed to be handled through software semaphores)
- Special cycles
- PCI cache support (all memory space within the C-5e NP is NOT cacheable to an external processor)
- JTAG (IEEE 1149.1)
- Power management
- Bus arbitration logic (an external PCI Central Resource is required to support this function)

PCI Access to C-5e NP Physical Address Space

An external PCI Initiator can access C-5e NP physical address space through six (6) 1MByte windows in PCI address space. The System Interface Configuration Space contains six (6) standard PCI Base Address Registers (BARs) each defining a 1MB prefetchable memory region.

While the regions are defined as prefetchable, software is responsible for properly handling any read side effects that may occur within the C-5e NP.

For each of these BARs, there is a corresponding address translation register indicating the 1MByte page in C-5e NP physical address space that is to be accessed.

C-5e NP Access to PCI Address Space

The XP can access PCI address space through eight (8) programmable windows in the C-5e NP's physical address space. Each window is controlled by an XP BAR and a PCI address translation register. The BAR controls the location and size of the window in C-5e NP physical address space. The programmable window sizes are: 16kBytes, 32kBytes, 64kBytes, 128kBytes, 256kBytes, 512kBytes, 1MByte, or 2MByte. Each window can be up to 2MByte in size, but the windows can be programmed as any combination of the specified sizes (for example, there could be eight 2MByte windows, or four 128kByte, three 1MByte, and one 256kByte windows). The *PCI Address Translation* register controls the window's view into the PCI address space.

The C-5e NP provides an optional byte swapping mode for moving data between the PCI Bus Little Endian environment and the C-5e NP Big Endian environment. Refer to "PCI Byte Swapping Overview" on page 822.

PCI Registers

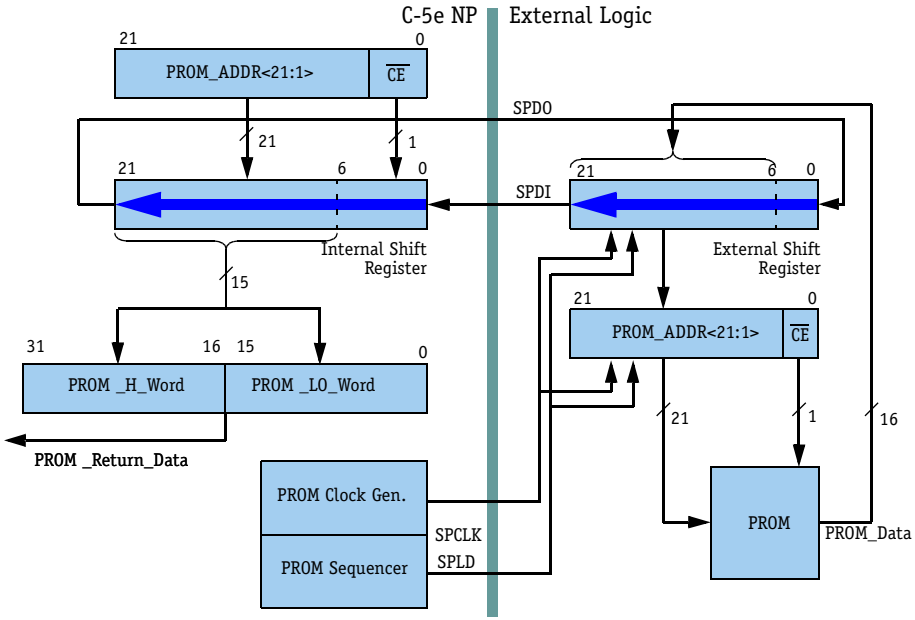
Table 190 on page 576 shows all the PCI Configuration registers. Refer to "Executive Processor (XP) Configuration Registers" on page 576.

PROM Interface

The PROM interface is a low-speed serial I/O interface that allows the C-5e NP to read from an external PROM. The PROM interface clock is created internally in the C-5e NP by dividing the core clock. The clock divider is programmable via the *XP Miscellaneous Control* register and can be set to values ranging from 2 to 16. The maximum PROM size addressable is 4MBytes, and must use a "by 16" part.

The external glue logic (which must be provided by the application) is illustrated in Figure 30 along with the internal mechanisms of the C-5e NP PROM interface. The glue logic consists of an external 22bit shift register with parallel-in and parallel-out capabilities, and a 22bit parallel-in/parallel-out register. Both registers must be positive edge triggered by the PROM interface clock, and perform a synchronous parallel load whenever SPLD is asserted high. For all other cycles, SPLD is asserted low, and the shift register should shift and the parallel register should hold.

Figure 30 PROM Interface



The PROM interface operates in the following manner. Whenever the XPRC, or an inbound transaction being serviced by the PCI target, requests a read from address 0xBFC00000 through 0xBFFFFFFC, the PROM interface initiates the following sequence (which accesses the 16bit wide external PROM to return a 32bit result). Note that two accesses are pipelined together to execute one 32bit fetch:

1 The PROM_ADDR is loaded into the C-5e NP internal shift register.

2 The PROM_ADDR is shifted into the external shift register for 22 SPCLK cycles.

3 SPLD is asserted for one SPCLK cycle, loading the PROM_ADDR into the external presentation register.

4 SPLD is deasserted for 22 SPCLK cycles. The PROM presents the first 16bit PROM_DATA. At the same time, the next PROM_ADDR is shifted into the external shift register.

5 SPLD is asserted for one SPCLK cycle, loading the PROM_ADDR into the external presentation register and the first PROM_DATA into the external shift register.

6 SPLD is deasserted for 22 SPCLK cycles, shifting the first PROM_DATA into the C-5e NP internal shift register.

7 SPLD is asserted for one SPCLK cycle, loading the first PROM_DATA into the C-5e NP PROM_RETURN_DATA register and the second PROM_DATA into the external shift register.

8 SPLD is deasserted for 22 SPCLK cycles, shifting the second PROM_DATA into the C-5e NP internal shift register.

9 SPLD is asserted for one SPCLK cycle, loading the second PROM_DATA into the C-5e NP PROM_RETURN_DATA register.

Serial Bus Interface

The Serial Bus interface is a general purpose bi-directional, two-wire serial bus and I/O port. It allows the C-5e NP to control external logic with either of two standard protocols. The high-speed protocol (MDIO) uses a 16bit data format with 10bits of addressing, and supports transfers up to 25MHz. The low-speed protocol uses an 8bit data format followed by an acknowledge bit and supports transfers at up to 400kbps. Software can select which protocol to use by setting the appropriate bits in the Serial Bus Configuration Register. When a serial bus transfer is active, an external pin is driven by the C-5e NP to indicate which protocol is being used (SPLD=0 indicates high-speed protocol, SPLD=1 indicates low-speed protocol).

The bus only supports a single master hierarchy that can operate as either a receiver or a transmitter. The bus also supports collision detection and arbitration, and an integrated addressing and data-transfer protocol.

Both SIDA and SICL are bi-directional lines that are connected (via a pull-up resistor) to the positive supply voltage. When the bus is free, both lines are HIGH. The output stages of the devices connected to the bus must have either an open-drain or open-collector in order to perform the wired-AND function required for its arbitration mechanism. Refer to "Serial Bus Configuration Register (XP Miscellaneous Control Function)" on page 602, and "Serial Bus Data Register (XP Miscellaneous Control Function)" on page 603.

C-5e NP Interface Options for Initialization

Typically, you use either the PCI or PROM interface to initialize the C-5e NP. Upon initialization, the XP begins executing at the first word of the 16-word IROM. The IROM uses the contents of location BD808300h as a pointer to a formatted boot image, copies the code from that image to the XP IMEM, and begins execution at the code's start address. Unless modified by an external system, the reset value in the boot image pointer is 0xBFC000000, which is the standard address for the boot PROM.

Using the PCI Interface Initialization Option

If you use the PCI to initialize the C-5e NP, you would normally use the C-5e NP as an intelligent peripheral to a host processor. Upon deassertion of the C-5e NP reset, all of the internal CPs and the XP continue to be held in a reset state and the external host processor is responsible for initialization.

The external system contains a C-5e NP boot image that is accessible to the XP via the PCI Bus. This image can be in a boot ROM or in any other memory region accessible via the PCI bus. The external host processor sets up the configuration registers in the System PCI to give the XP access to the boot image, sets the boot pointer to the address of the image in C-5e NP address space, and then releases the XP to begin fetching code over the PCI bus.

Using the PROM Interface Initialization Option

You would use the PROM interface to initialize the C-5e NP if the C-5e NP is used as a stand-alone processor in a single-C-5e NP system. Upon deassertion of reset, the XP immediately begins to fetch code from the PROM.

Other XP Interfaces

In addition, the XP has access to:

- PCI interface with both Initiator and Target capabilities.
- Global Bus access to all CP configuration registers and DMEMs from both the PCI target and the XPRC.
- Ring Bus access from both the PCI target and the XPRC.
- Payload Bus access from both the PCI target and the XPRC via Control Blocks.

All CP configuration registers and DMEMs are accessible via the Global Bus from the XPRC and PCI. However, CPs cannot access XP configuration registers or the PCI bus. Also, CPs can only access one bank of XP DMEM (Bank 1) via the Global Bus; Bank 2 is not visible. In addition, the PCI and XPRC have the same access to all resources with the exception of the IMEM and IROM.

Table 31 lists the accessibility of XP initiated data transactions to various C-5e NP Resources.

Table 31 Accessibility of XP Initiated Data Transactions to C-5e NP Resources

		TRANSACTION INITIATOR*						
		XPRC	PCI TARGET	CPS VIA GLOBAL BUS	TXCB/ RXCB #24	TXCB/ RXCB #25	RDCB/ WRCB #24	RDCB/ WRCB #25
RESOURCES	IROM	W	none	none	none	none	none	none
	XP SPECIFIC CONFIGURATION REGISTERS	W, H, B	W, H, B	none	none	none	none	none
	XP CP-LIKE CONFIGURATION REGISTERS†	W, H, B	W, H, B	none	none	none	none	none
	EXTERNAL SERIAL BUS	H, B	H, B	none	none	none	none	none
	PROM	W	W	none	none	none	none	none
	RING BUS	Yes	Yes	none	none	none	none	none
	CPS VIA GLOBAL BUS	W	W	none	none	none	none	none
	SDRAM (PAYLOAD ONLY)	none	none	none	16Bytes	16Bytes	16Bytes	16Bytes
	IMEM	W	none	none	none	16Bytes	none	none
	DMEM #24	W, H, B (1 stall)	W, H, B	W	16Bytes	none	16Bytes	none
	DMEM #25	W, H, B (no stall)	W, H, B	none	none	16Bytes	none	16Bytes
	PCI	W, H, B	W, H, B	none	16Bytes	none	none	none

* The table entries indicate accessibility in terms of byte (B), half-word (H), 32bit word (W), and larger transfer operations.

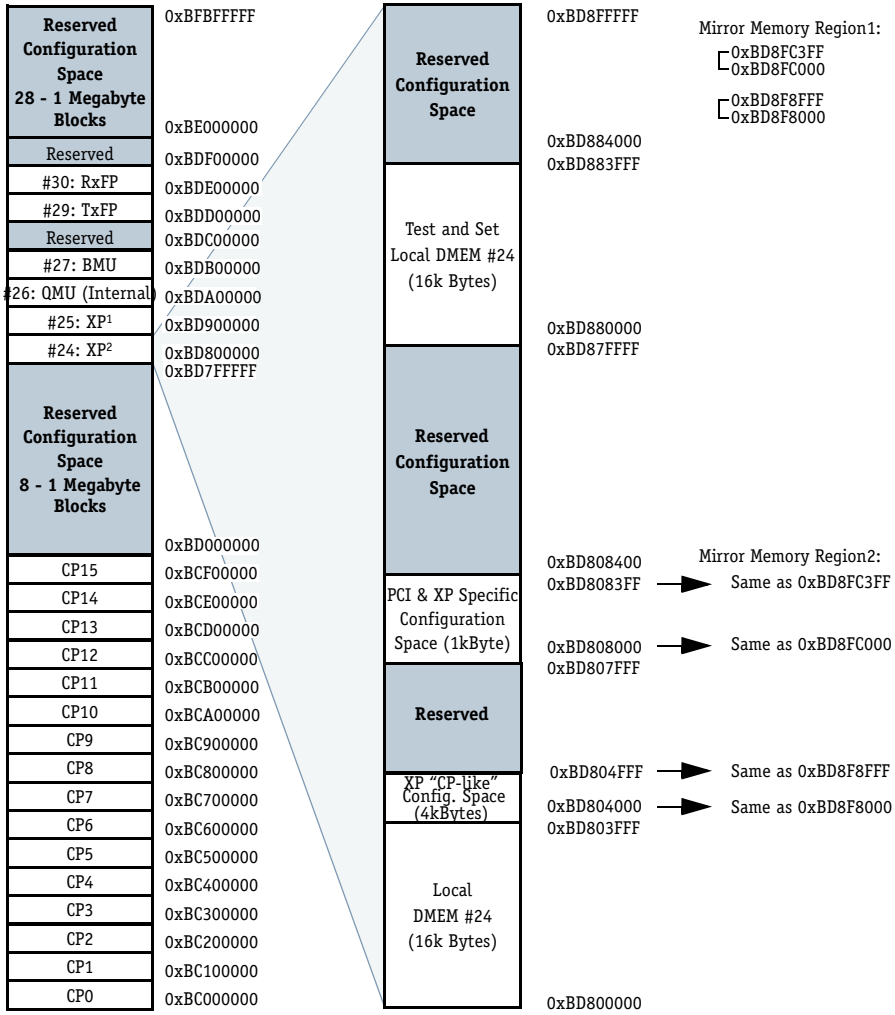
† All control blocks

XP Configuration Space

The Executive Processor (XP) has two areas of memory, referred to as XPSlot 24 and XPSlot 25. Both XPSlot 24 and XPSlot 25 has 1MByte of memory allocated to each for its use. Only the DMEM part of XPSlot 24 can be accessed by all Channel Processor (CPs). In contrast, no CP can access the XPSlot 25 area, however, the XP has full access to the XPSlot 25 area. The memory maps for the XPSlot 24, XP Slot 25, and PCI, XP and other miscellaneous registers are shown in Figure 31 on page 176, Figure 32 on page 177, and Figure 33 on page 178.

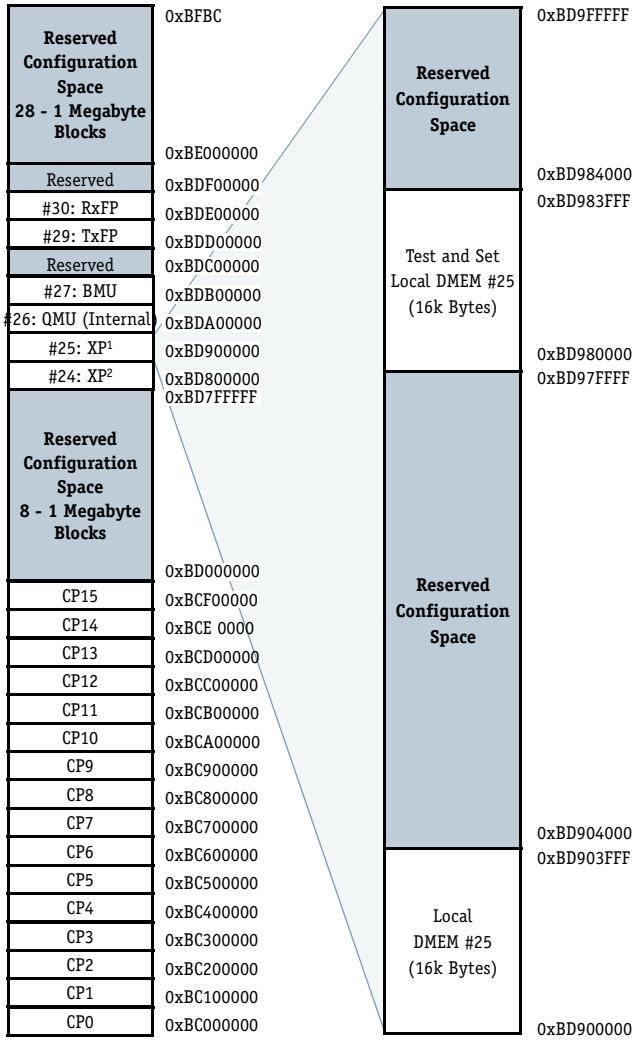
Although specific ranges of memory are allocated to specific functions, the entire area may not be used.

Figure 31 XP Configuration Space (Slot #24)



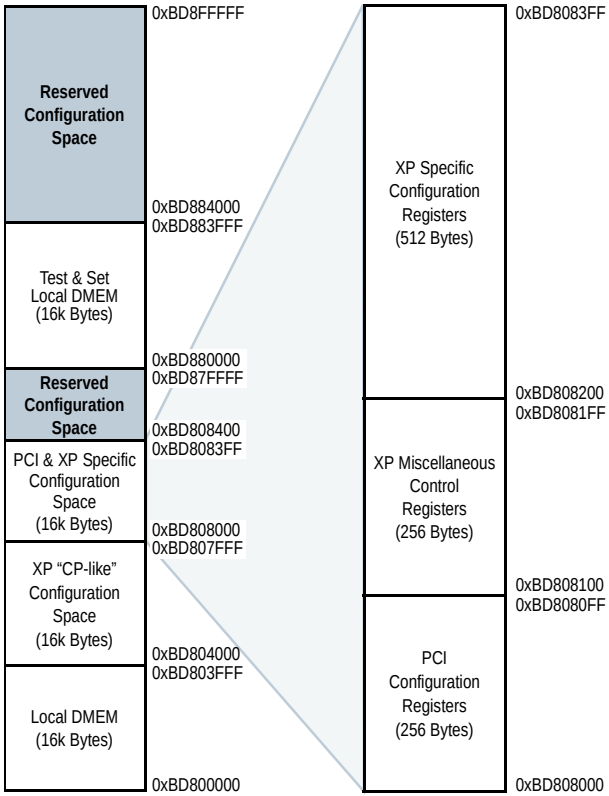
1: XP #25 can only be accessed by the XP, it is not visible to CPs.
2: The CPs can only access DMEM in XP #24.
3: Access to the XP Configuration Register's physical memory locations can be done using either one of two different Mirrored Memory Regions (Blocks).

Figure 32 XP Configuration Space (Slot #25)



1: XP #25 can only be accessed by the XP, it is not visible to CPs.
2: The CPs can only access DMEM in XP #24.

Figure 33 XP Slot #24 Configuration Space for PCI, XP and Miscellaneous Registers



*For complete details about specific registers go to their reference. Refer to "*Executive Processor (XP) Configuration Registers" on page 576.

Chapter 4

FABRIC PROCESSOR

Chapter Overview

This chapter covers the following topics:

Fabric Processor (FP) Overview

The FP provides a high-bandwidth port for the segmentation and reassembly of PDUs at up to OC-48 speeds. It behaves like a high-speed network interface port (up to 125MHz for two 32bit data paths) with advanced functionality that allows the C-5e NP to interface to an application-specific switching solution or a switching fabric. The FP can be configured to conform to seven (7) different fabric interfaces that include: CSIX-L1, UTOPIA-1, -2, -3, PRIZMA, Power X(CSIX-L0), and UTOPIA3 like to M-5. The programming flexibility of the FP allows it to support standards-based and customer proprietary switch fabric cell formats.

The FP performs flow mapping and management to and from the switching fabric. It can receive up to 159 flows concurrently, and supports transmission of up to 128 prioritized, simultaneous flows configured as either: a 32-port matrix with four (4) priority levels, or a 16-port matrix with eight (8) priority levels. These flows support:

- Unicast and multicast topologies
- C-5e NP-to-fabric link-level flow control
- End-to-end congestion management and flow control
- Segmentation and reassembly (SAR) of Protocol Data Units (PDUs) to and from configurable uniform fabric cells for the purposes of higher fabric utilization and Quality of Service (QoS) based arbitration

Think of the FP as a very high performance Channel Processor (CP) without a RISC Core (RC). It uses the same bus interfaces and data path constructs as a CP. The receive (FPRx) and transmit (FPTx) parts of the FP can operate both autonomously and asynchronously. The FPRx and FPTx each contain two (2) microcode programmable Byte Processors that use the same instruction architecture as the (CPs) SDP Byte Processors. Thereby, enabling the FP to adapt to customer proprietary fabric header formats.

Terminology

For definitions of terminology commonly associated with the FP, refer to the "Glossary" on page 847, at the end of this book. The word "segment" used in this chapter corresponds to the following:

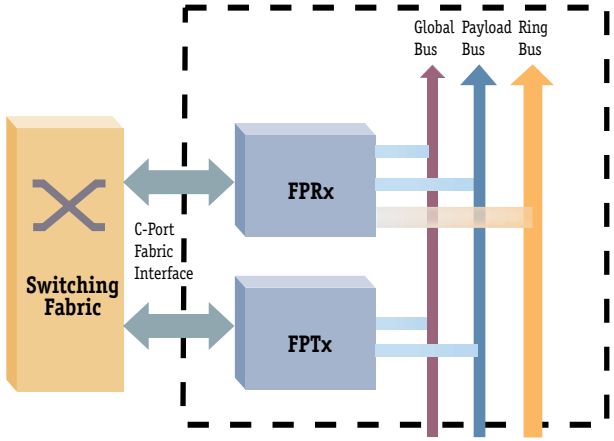
Table 36 Protocol-Specific Nomenclature

PROTOCOL	NOMENCLATURE
UTOPIA	Cell
PRIZMA	Packet
CSIX-L1	Frame
PowerX (CSIX-L0)	Frame

FP Block Diagram

Figure 40 shows a high-level diagram of the FP.

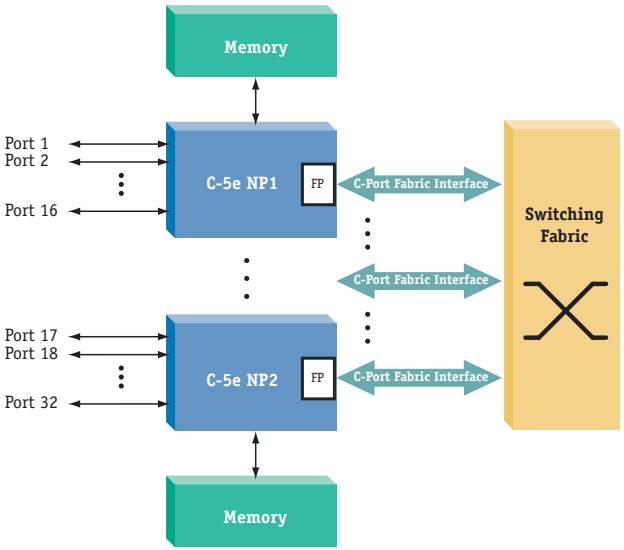
Figure 40 Fabric Processor Block Diagram



Multiple C-5e NP Configurations

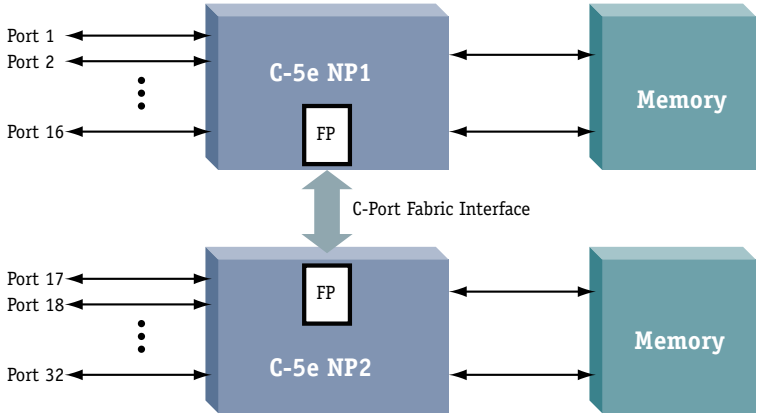
A switching fabric is used when more than two (2) C-5e NPs are required in a system. The switching solution has two (2) or more FP-type ports and provides a mechanism for switching cell or packet-based data from one C-5e NP to another. A homogenous, multi-C-5e NP application is shown in Figure 41.

Figure 41 Multiple C-5e NPs with Switching Fabric



The FP is designed with symmetric receive (Rx) and transmit (Tx) interfaces. Therefore, the C-5e NP can use the FP to provide a glueless two (2) chip solution, as shown in Figure 42.

Figure 42 Two C-5e NP Application



General FP Specifications

Table 37 on page 183 lists selected general FP specifications.

Table 37 FP General Specifications

ITEM	SPECIFICATION
Fabric interface frequency	Up to 125MHz
Separate Rx and Tx Data Buses	8bits, 16bits, or 32bits
Protocols supported	CSIX-L1, PowerX(CSIX-L0), PRIZMA, UTOPIA 1, 2, 3 ATM and PHY, (except 8bitPHY)
Segment size	8Bytes minimum, 204Bytes maximum (payload per segment: 4Bytes minimum, 196Bytes maximum) Note: Size must be a multiple of 4.
PDU size	5Bytes minimum, 64K -1Bytes maximum
Full Duplex Bandwidth	Up to 4000Mbps

FPTx Overview

The FPTx transmits segmented PDUs onto the external fabric interface. It services up to 128 Queue Management Unit (QMU) queues by dequeuing descriptors, reading payload out of Buffer Management Unit (BMU) buffers, breaking up the payload into segments and placing a header on each segment. The basic sequence of a PDU being transmitted from the FPTx consists of six (6) steps. Refer to "FP Transmit (FPTx) Sequence" on page 185.

FPRx Overview

The FPRx receives segments from the external fabric interface and writes them to BMU buffers, reassembling them into PDUs (up to 159 concurrently), while building and enqueuing the associated descriptors. The basic sequence of a PDU being received by the FPRx consists of six (6) steps. Refer to "FP Receive (FPRx) Sequence" on page 203.

FP Transmit (FPTx) Sequence

The FPTx performs essentially the same transmit function as a Channel Processor (CP), but with a high performance and mostly hard-wired implementation. The FPTx can actively transmit segments in a round-robin fashion from up to 8 PDUs. As many as 128 queues can be serviced.

In general, the FPTx services a number of QMU queues, using descriptors to identify the PDUs in the BMU buffer, and segments the PDUs and places a header at the beginning of each segment before transmitting onto the external fabric interface.

The basic sequence of a PDU being transmitted from the FPTx consists of six (6) steps. The six (6) steps include: FPTx Dequeuing PDUs, FPTx Decoding Descriptors, FPTx Reading Payload, FPTx Byte Processors Microcoding, FPTx Header and Payload Merging, and FPTx Fabric Interface Transmit Operation.

The basic FPTx PDU sequence is indicated in Table 38 on page 185, and Figure 43 on page 187 illustrates both the sequence and main components of the FPTx. In addition, Figure 44 on page 188 shows the Global Address Memory Map of the FPTx.

Table 38 FPTx PDU Sequence and Reference to Details

FPTX SEQUENCE	SEQUENCE NUMBER		DETAILS
FPTx Dequeuing PDUs	1	Whenever there is something to transmit (as indicated by the QMU), the FPTx makes a dequeue request to the QMU via a dedicated FP-to-QMU interface.	Refer to "FPTx Dequeuing PDUs" on page 189.
	1a	The descriptor returns from QMU via Payload Bus.	
FPTx Decoding Descriptors	2	The BTag, Pool, PDU length, and multicast flag are extracted from descriptor.	Refer to "FPTx Decoding Descriptors" on page 190.
	2a	The current queue length, which was returned along with the descriptor in step 1a, is saved.	
FPTx Reading Payload	3	The Payload is read from the BMU buffer pointed to by BTag/Pool and placed in DMEM awaiting transmission to the fabric interface.	Refer to "FPTx Reading Payload" on page 190.
FPTx Byte Processors Microcoding	4	The microcode programmable TxByte Processors generate headers for segments.	Refer to "FP TxByte Processors Microcoding" on page 191.

Table 38 FPTx PDU Sequence and Reference to Details (continued)

FPTX SEQUENCE	SEQUENCE NUMBER		DETAILS
FPTx Header and Payload Merging	5	Payload and Header are merged to form segments.	Refer to "FPTx Header and Payload Merging" on page 199.
Fabric Interface Transmit Operation	6	Segments are transmitted via the external Fabric interface until entire PDU (as indicated by PDU length) has been transmitted. The segments are interleaved with segments of other PDUs in a round-robin fashion.	Refer to "FPTx Fabric Interface Transmit Operation" on page 199.
	6a	The segment CRC is generated (optional).	
	6b	Segment data is adjusted according to endianness.	
	6c	If a multicast flag was set, the multicast counter associated with pool/BTag is decremented, otherwise the BTag is deallocated back to the pool.	

In addition, there are two related FPTx topics that are outside the basic sequence. Refer to "FPTx Advanced Features" on page 199, and "FPTx Error Reporting and Interrupts" on page 201.

Furthermore, for a description of functions that span both the FPRx and FPTx such as: FP Flow Control (Link-Level and Per-Queue), FP Descriptor Size, FP CRC, FP Endianness, and FP Payload Bus Bandwidth, please refer to "FPTx and FPRx General Considerations" on page 237.

Figure 43 FPTx Sequence and Block Diagram

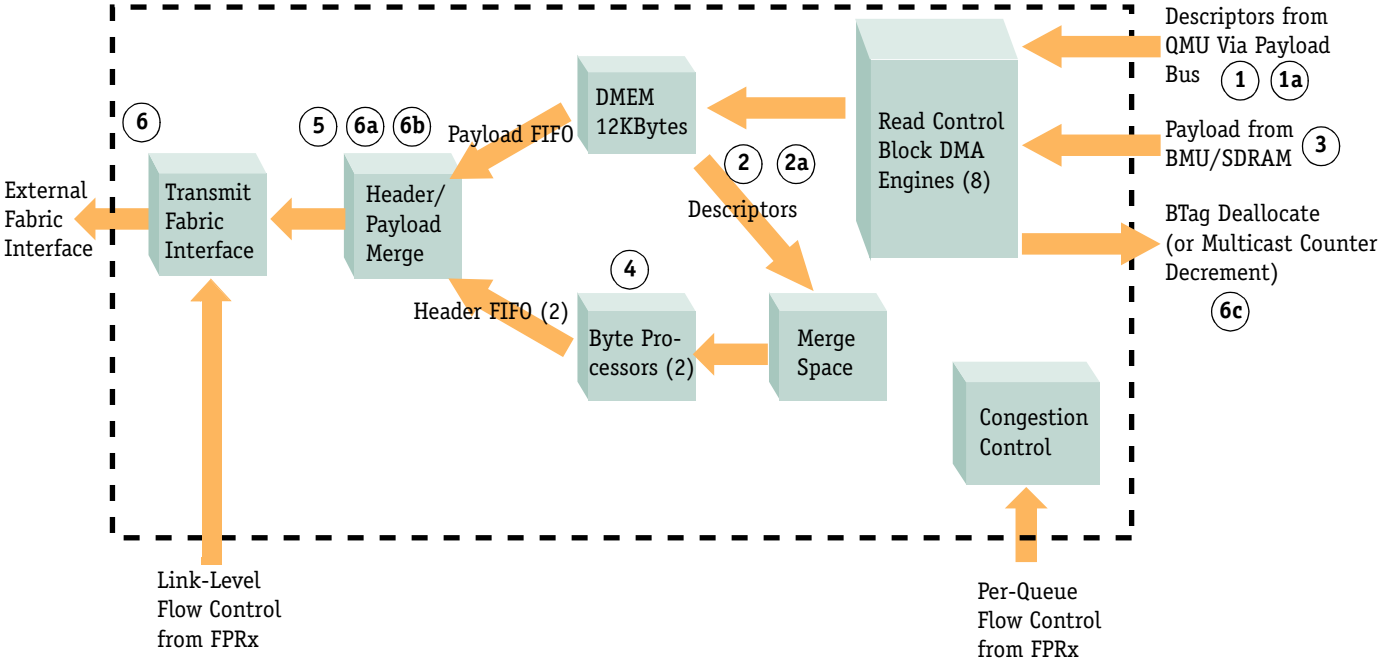
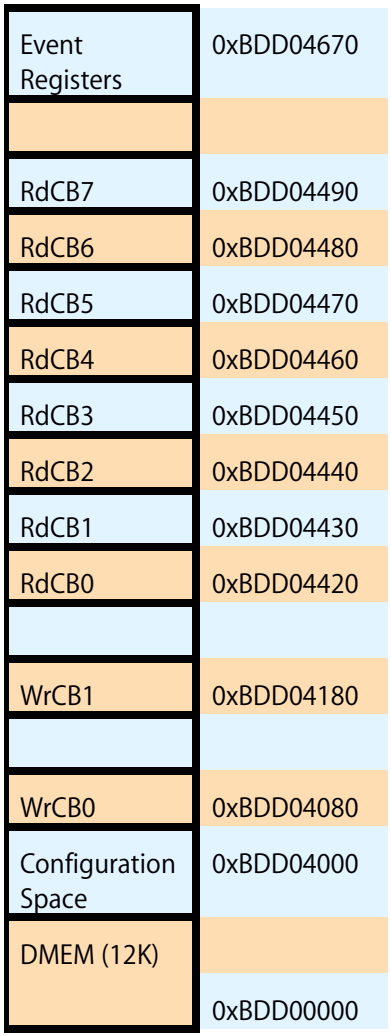


Figure 44 FPTx Global Address Memory Map



FPTx Dequeuing PDUs

The FPTx only begins dequeuing descriptors and transmitting segments after it has received a queue ready notification from the QMU via the global bus. After that, it continues transmitting PDUs from that queue until the queue length returned with a descriptor is zero, indicating that the queue is empty.

The FPTx's base queue is configured via the *TxSysConfig* register bits [24:16] *QueueOffset* field and must be configured to be the same queue that the QMU configuration specifies. The FPTx has no knowledge of the number of queues assigned to it. It services any queue for which it receives a queue ready notification from the QMU. Refer to "TxSysConfig Register (FP Tx Configuration Function)" on page 678.

The FPTx can service up to 128 queues. The FPTx can be configured so that its 128 queues can be organized as either: 32 ports with four (4) priorities per port, or 16 ports with eight (8) priorities per port. This organization is selected in *TxFCE_Configuration* register bit [18] *QueueDepth* field.

Even if fewer than 128 FPTx queues are used, the port organization is the same. The FPTx does not know how many queues are actually used.

Ports are contiguous sets of queues with the highest priority queue being the lowest numbered queue. For example, with a 32x4 organization, queues 0 to 3 are part of port 0, with queue 0 being the highest priority queue within that port and queue 3 being the lowest priority queue within that port.

If more than one (1) queue is non-empty, then FPTx transmits segments in an interleaved fashion, with up to eight (8) queues being transmitted simultaneously. The FPTx always completes transmission of one (1) PDU from a queue before beginning another PDU from that queue, thus ensuring in-order transmission from queues. When multiple queues are non-empty, the FPTx uses the algorithm described in "Weighting Algorithm" on page 199 to select the next queue to transmit.

FPTx Decoding Descriptors

Every descriptor that the FPTx dequeues must contain:

- A 5bit pool and 16bit BTag that point to the buffer to be transmitted.
- A 16bit PDU length indicating the amount of data in that buffer which should be transmitted.
- A 1bit multicast flag.

The FPTx can be configured to extract these parameters from different locations within the descriptor with a few constraints. This configuration is done using the *TxDescInfo* register. For multi-bit fields (length, pool, and BTag), the "position" indicates the position within the descriptor of the least significant bit of the field.

Bit positions within the descriptor are numbered so that bit position 0 is the least significant bit of the first 32bit word of the descriptor, bit position 32 is the least significant bit of the second 32bit word of the descriptor, etc.

*Multi-bit fields must be positioned so that they do not cross 32bit boundaries. For example, the 5bit pool field cannot be positioned at bit 30 of the descriptor because it would require some bits in the first 32bit word of the descriptor and some bits in the second 32bit word. Refer to "*TxDescInfo Register (FP Tx Configuration Function)*" on page 675*.

FPTx Reading Payload

Each PDU's payload is read from the Buffer Management Unit (BMU) into FPTx data memory (DMEM), from where it gets segmented and eventually transmitted onto the fabric interface. When the FPTx begins transmitting a PDU from a buffer, it always transmits from the beginning (offset 0) of that buffer. It is *not* possible for the FPTx to begin transmission from any offset inside the buffer.

The FPTx ignores the out-of-band bits returned from the BMU and relies on the length passed to it in the descriptor to determine what portion of the buffer to transmit.

FPTx Data Memory (DMEM)

The FPTx hardware uses 12KBytes of data memory for storage. It uses 2KBytes for storing payload for its 8 active flows, 8KBytes for up to 128 descriptors, and the remainder for BTags to be deallocated.

FP TxByte Processors Microcoding

The FPTx includes two (2) microcode programmable FPTx Byte Processors, similar to those in the Channel Processors (CPs). These TxByte Processors are used to generate the header for each segment, because header formats vary from fabric protocol to fabric protocol and application to application. Headers for consecutive segments are generated alternately by one (1) of the two (2) FPTx Byte Processors.

Typically, more information needs to be included in the header of the first segment of a PDU than is required in subsequent segments. For example, the header of the first segment typically must include information about the length of the PDU and the destination queue on the receiving Network Processor. This information needn't be conveyed in subsequent segments for that PDU.

For this reason, the FPTx supports two (2) different header sizes: one that is used for first or only segments, and another that is used for middle or last segments. Because the FPTx always appends as much payload as possible after the header, this allows more payload to be transmitted with each middle and last segment, thus increasing the efficiency of the transmission.

The header sizes are configured in *TxDM_Header/Payload Delimiter* register bits [7:0] *HeaderLen1* field for the first or only segments, and bits [15:8] *HeaderLen2* field for the middle or last segments. Headers for all cells can be configured to be the same size. FP TxByte Processor microcode must generate headers of exactly the sizes configured. Header sizes must be non-zero and multiples of 4. Refer to "TxDM_Header/Payload Delimiter Register (FP Tx Configuration Function)" on page 675.

FP TxByte Processors

In general, FP TxByte Processor microcoding is done much like microcoding for a SDP Byte Processor. The FPTx Byte Processor is capable of the same sequencing and ALU operations as the SDP Byte Processor, with 64 control store entries and 24 CAM entries.

There are four (4) unique aspects of the FP TxByte Processor, compared to a SDP. They include: different external test conditions, different inputs (header inputs), *no* input FIFO, and the Tx is *not* allowed to write a Creg (that is, the TxByte Processor's *TxStatus* register) on two (2) consecutive microinstructions.

External Test Conditions

There are eight (8) external test conditions available to the FPTx Byte Processor as described in the *C-Ware Microcode Programming Guide (part number CSTMCPG-UG/D)*. The only one that is used by the FPTx Byte Processor is bit1= Header FIFO empty. Therefore, bit [0] and bits [2:7] are *not* used by the FPTx Byte Processor. The Header FIFO empty test condition is true when the Header FIFO for the TxByte Processor is empty.

Header Inputs

The TxByte Processor can read seven (7) different inputs to construct a header. Refer to Table 39 on page 192.

Table 39 TxByte Processor Header Inputs and Their Descriptions

INPUTS	DESCRIPTION
Current payload length	This value represents the number of Bytes of payload appended to the header to form the segment. Typically this is only useful for applications such as PowerX (CSIX-L0) where the segment size is included in the fabric header. Refer to "Pay_Len [7:0]" on page 198. Note: If the CRC is enabled, the payload length includes an additional 4Bytes of CRC.
Current segment type	This value represents the segment type. Refer to "TxStatus [7:0]" on page 198.
FP queue	This value represents the FP queue number (offset from the FP base queue) from which the current segment's PDU came. Typically, this would be used to form the PDU ID or fabric address. Refer to "Src_Queue [6:0]" on page 198.
General Purpose Configuration Registers (TxByte_Ctl0 and TxByte_Ctl1)	There are 8Bytes of general purpose registers that can be initialized with global writes and read by either TxByte Processor. For example, one of these Bytes might be initialized to contain a unique Network Processor ID (for a multiple Network Processor system) that could then be incorporated into a PDU ID in the header used by the FPRx for reassembly. Both Byte Processors read the same value from these registers. There are no restrictions on when the two (2) TxByte Processors can read these registers; that is, they can both read any Byte any time, including different Bytes at the same time. Refer to "TxByte_Ctl0 & TxByte_Ctl1" on page 197. Note: The TxByte Processors cannot write to these registers.

Table 39 TxByte Processor Header Inputs and Their Descriptions (continued)

INPUTS	DESCRIPTION
Descriptor contents	Typically, headers contain at least some portion of the descriptor that the FPTx dequeued. All Bytes of the current descriptor are made available to the TxByte Processor through an internal memory known as *Merge Spac*e. Refer to "Merge0 - Merge63" on page 197.
Information from RxByte Processors	There are 17bits of information that the FPRx sends to the FPTx hardware for per-queue flow control (Refer to "Fabric to C-5e NP Per-Queue Flow Control" on page 238). These bits can also be read by the TxByte Processors as a general purpose communication mechanism from the RxByte Processors. If used for this purpose, disable flow control. Refer to "TxFI_Configuration Register (FP Tx Configuration Function)" on page 673. Also refer to the registers beginning "Pool0_CFG0 Register (FP Rx Pool Configuration Function)" on page 710.
Literals	

TxByte Processors Microcoding Performance Considerations

A segment does *not* begin transmission until both its payload and header are ready. For optimal performance, since their are two (2) TxByte Processors, the microcode should be constructed to complete header building faster than the time it takes to transmit two (2) segments on the interface, otherwise, bandwidth is wasted.

TxByte Processor Microcoding Minimum Requirements

For the TxByte Processor's microcode to operate properly, these minimum requirements must be followed:

- Wait for datascope ownership. Specifically, because the FPTx can transmit up to eight (8) queues at a time, it provides context or "datascope" for one of eight PDUs at a time to the TxByte Processor. However, there are times when the FPTx hardware is updating this context and therefore the datascope is *not* ready for processing. In this case, the TxByte Processor's microcode *must* wait until it is granted ownership of a datascope before it begins constructing a header. The microcode tests for datascope availability using bit [7] *Ownership* field of the *TxStatus* register of the TxByte Processor. Refer to Table 41 on page 197.
- Wait for the header FIFO to be empty using the header FIFO empty test condition.
- Build the header by writing out the header Bytes in sequence.

- Indicate end of header by using Merge9 (refer to *C-Ware Microcode Programming Guide (part number CSTMCPG-UG/D*). This indicates to the FPTx hardware that the full header is constructed and ready to be merged with payload to form a segment.
- Flip ownership for that datascope. After the TxByte Processor's microcode has finished generating a header, it passes ownership for the datascope to hardware by setting bit [7] *Ownership* field of the *TxStatus* register of the FPTx Byte Processor.

TxByte Processor's Memory Space and Registers

The TxByte Processor contains three (3) types of memory space, they include: Merge Space, Control Space, and TxByte General Purpose Space. Each memory space provides a different function and each space consists of a number of registers specific to that function. Both the Merge Space and Control Space are used to pass information between the TxByte Processor and associated FP hardware. In contrast, the TxByte General Purpose Space does *not* pass information between the TxByte Processor and associated FP hardware.

Table 40 on page 195 lists the three (3) Memory Spaces of a TxByte Processor and descriptions, Figure 45 on page 196 shows the TxByte Processor's Memory Map with specific registers, and Table 41 on page 197 provides a mapping of the Global Addresses (32bits) to the TxByte Processor's Byte Address (8bits) with descriptions, and access information for Global Bus, TxByte Processor and Hardware. The information in Table 40 on page 195, Figure 45 on page 196, and Table 41 on page 197 pertain to a *single* TxByte Processor.

Table 40 TxByte Processor Memory Space and Descriptions

MEMORY SPACE	DESCRIPTION
Merge Space	Sixty-four (64), 32bit Merge registers (256Bytes) are used for descriptor data, that are organized as eight (8) groups of 32Bytes. The datascope number selects the group and the TxByte Processor Address offset selects the Byte. The Merge space is globally accessed via the *TxMergeAddr* and *TxMergeData* registers. The *TxMergeAddr* register is used to index into the Merge Block and read/write data via the *TxMergeData* Register. Merge space registers are used for passing fields to be inserted as part of the Segment Header by the TxByte Processor. During normal operation the TxByte Processor performs Byte-width reads and the FPTx hardware writes the Merge registers with the entire Internal Descriptor. Byte 0 of the Descriptor is written to Merge[0], Byte *n* of the descriptor is written to Merge[n], and so on. The TxByte Processor cannot write these registers and is restricted to one descriptor (up to 32Bytes at a time dependent upon datascope). The Merged information is prepended as part of the Segment Header and formatted to the fabric destination descriptor format. There are eight descriptors of either 32 or 16Bytes in length as configured by the "TxFCE_Configuration Register (FP Tx Configuration Function)" on page 679. Refer to "TxMergeAddr Register (FP Tx Debug Function)" on page 687, and "TxMergeData Register (FP Tx Debug Function)" on page 687.
Control Space	Control information unique to the TxByte Processor. The TxByte Processor control space consists of six (6) registers. Refer to "TxStatus [7:0]" on page 198, "Src_Queue [6:0]" on page 198, "Pay_Len [7:0]" on page 198, "TxCG PR [0]" on page 198, "TxCG ID H" on page 198, and "TxCG ID L" on page 198.
TxByte General Purpose Configuration Space	4Byte General Purpose Configuration registers that are shared by both TxByte Processors. Refer to "TxByte_Ctl0 & TxByte_Ctl1" on page 197.

Refer to Appendix A for details about global accesses to Merge Space or TxByte General Purpose Configuration Space. The TxByte Control Space is not globally accessible and is therefore not found in Appendix A.

Figure 45 TxByte Processor Memory Map

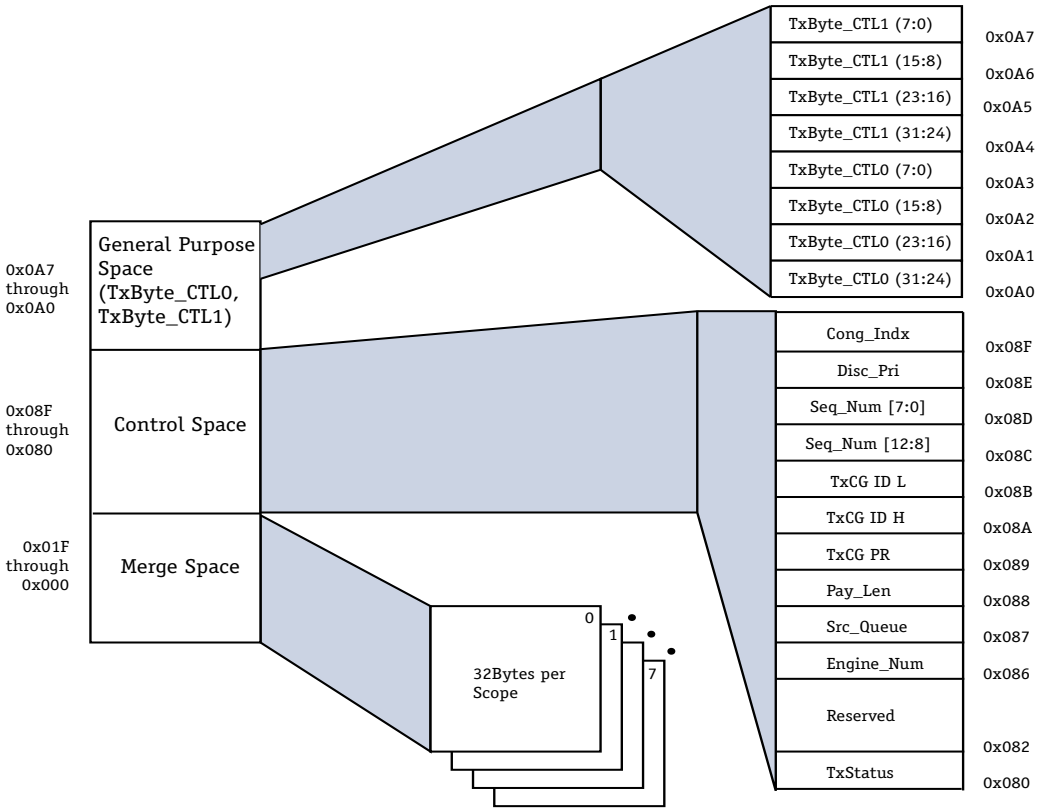


Table 41 TxByte Processor Mapping and Details

GLOBAL ADDRESS (TXBYTE0/TXBYTE1)	TXBYTE PROCESSOR ADDRESS (8BITS)	NAME	DESCRIPTION	ACCESS VIA GLOBAL BUS	ACCESS VIA TXBYTE PROCESSOR	ACCESS VIA HARDWARE
Indirectly accessible for debug purposes using 0xBDD0403C and 0xBDD04040 1	0x000 - 0x01F	Merge0 - Merge63	Sixty-four (64) 32bit merge registers (256Bytes) used for descriptor data, organized as eight(8) groups of 32Bytes. The datascope number selects the group and the TxByte offset selects the Byte.	R/W	R	W
0xBDD04048, 0xBDD0404C	0x0A0 - 0x0A7	TxByte_Ctl0 & TxByte_Ctl1	4Byte TxByte General Purpose Configuration registers (4Bytes for each TxByte Processor).	R/W	R	N/A

Table 41 TxByte Processor Mapping and Details (continued)

<table>
<tr><th>GLOBAL ADDRESS (TXBYTE0/TXBYTE1)</th><th>TXBYTE PROCESSOR ADDRESS (8BITS)</th><th>NAME</th><th>DESCRIPTION</th><th>ACCESS VIA GLOBAL BUS</th><th>ACCESS VIA TXBYTE PROCESSOR</th><th>ACCESS VIA HARDWARE</th></tr>
<tr><td rowspan="8">Note: These registers are not accessible via the Global Bus.</td><td>0x080</td><td>TxStatus [7:0]</td><td>Status register: bit [7] is the ownership bit. Bits [6:2] are not used. Bits [1:0] represent the Segment type. Legal ranges are detailed here:
<table><tr><th>ENCODED VALUES FOR [1:0]</th><th>SEGMENT TYPE</th></tr><tr><td>00</td><td>Continuation of Message (COM), (middle) segment</td></tr><tr><td>01</td><td>End of Message (EOM), (last) segment</td></tr><tr><td>10</td><td>Beginning of Message (BOM), (first) segment</td></tr><tr><td>11</td><td>First and only Message (FOM), (first and last) segment</td></tr></table></td><td>N/A</td><td>Bit [7]=W2
Bits [1:0]=R</td><td>Bit [7]=W
Bits [1:0]=W</td></tr>
<tr><td>0x087</td><td>Src_Queue [6:0]</td><td>Cell Source Queue # (0 - 127)</td><td>N/A</td><td>R</td><td>W</td></tr>
<tr><td>0x086</td><td>Engine_Num</td><td>Indicates which of the 8 FPTx flow engines is assigned to the segment. Legal range= 0 to 7.</td><td>N/A</td><td>R</td><td>W</td></tr>
<tr><td>0x088</td><td>Pay_Len [7:0]</td><td>Payload Length within cell</td><td>N/A</td><td>R</td><td>W</td></tr>
<tr><td>0x089</td><td>TxCG PR [0]</td><td>Congestion register Pause/Resume bit (bit 0)</td><td>N/A</td><td>R</td><td>W</td></tr>
<tr><td>0x08A</td><td>TxCG ID H</td><td>Congestion Flow ID High [15:8]</td><td>N/A</td><td>R</td><td>W</td></tr>
<tr><td>0x08B</td><td>TxCG ID L</td><td>Congestion Flow ID Low [7:0]</td><td>N/A</td><td>R</td><td>W</td></tr>
</table>

[1] Accessed via *TxMergeAddr* and *TxMergeData* registers (0xBDD0403C and 0xBDD04040).

[2] Once the header has been built, the TxByte Processor should set the ownership bit by performing a Creg write. The address and data of the write do not matter; the act of doing any Creg write sets the ownership bit. This characteristic is unique to the TxByte Processor and does not apply to the other byte processors.

FPTx Header and Payload Merging

The FPTx hardware merges the headers built by the two (2) TxByte Processors with the associated Payload, to form segments for transmission.

FPTx Fabric Interface Transmit Operation

The FPTx transmits segments on the fabric interface in compliance with the configured protocol (for example, CSIX-L1). Refer to "Fabric Interface Modes and Configurations" on page 243 for specific details about each of the seven (7) supported fabric interfaces.

FPTx Advanced Features

In addition to the normal transmission sequence, the FPTx provides features for certain advanced applications. The features include: weighting algorithm, and virtual queuing for FPTx.

Weighting Algorithm

There are eight (8) FPTx flow engines, and thus up to eight (8) queues can be serviced simultaneously. As long as there are eight (8) or fewer queues with something to transmit, each queue is assigned to a flow engine, and segments are transmitted in a round-robin fashion. In this case no weighting algorithm is used.

However, as soon as there are more than eight (8) queues with something to transmit, the FPTx must decide how to map the queues to the eight (8) flow engines. It does this by means of a configurable weighting algorithm.

The weighting algorithm can be configured by the user to accomplish the following:

- Allocate bandwidth among the queues within a port
- If desired, provide guaranteed service for a single Absolute Priority Queue within each port
- If desired, guarantee that queues are allowed to transmit a minimum number of bytes (Quantum Size) while being serviced

Allocating Bandwidth Among Queues

Bandwidth can be allocated among the queues within a port by configuring the Weight Counters for each of the 128 FPTx queues (either 32 ports by 4 queues or 16 ports by 8 queues). The *Weight Counter* represents the number of Payload Bytes currently allocated to the queue. The initial weight can be configured to be anywhere from 16KBytes to 240KBytes in 16KByte increments (15 possible values), using the *TxQueueWeight_Configuration* register bits [10:7] *WeightCtrValue* field. If not explicitly configured, the initial weight defaults to 16KBytes. The range of 15 possible initial values implies that the maximum bandwidth ratio of two queues within a port is 15:1. As each PDU is serviced, the hardware subtracts the number of payload bytes in that PDU from the associated Weight Counter. Once a queue has used all of its bandwidth, at the appropriate time the queue "refreshes" by having its initial weight added back to its Weight Counter (possibly multiple times).

An initial value of zero should not be used because it results in undefined behavior. A zero value in the weight counter represents no bandwidth left and a negative value represents a borrow towards future bandwidth.

As an example, to allocate bandwidth equally among the queues in port0 (assume 32x4 port depth), configure the initial Weight Counter values to be equal for q0 through q3. If the desired bandwidth allocation is q0=50%, q1=25%, q2=12.5%, q3=12.5%, set the initial values using the appropriate ratios, for instance q0=64KB, q1=32KB, q2=16KB, q3=16KB.

Absolute Priority Queues

The lowest queue in each port can be configured as an *absolute priority queue*. Absolute priority queues are serviced *before* any other queues, provided the following three (3) conditions are true: the queue is non-empty, the queue is *not* paused, and the queue is *not* already transmitting a flow. Absolute priority queues are serviced in a round-robin fashion starting from queue0. Once an absolute priority is serviced by a flow, it continues to be serviced from that flow until the queue is empty. Absolute priority is enabled using the *AbsolutePriority_Configuration* register.

Minimum Quantum Size

Queues normally transmit a single PDU while being serviced (except for Absolute Priority queues), after which the assigned flow engine is released and assigned to the next queue needing to be serviced. Some applications may desire for the queue to transmit a guaranteed minimum number of bytes while being serviced. This can be accomplished by configuring the *Minimum Quantum (MQ) Counter f*or each queue. The initial value of each MQ counter is a function of the initial value of the associated Weight Counter. The default

values for each are 0Bytes (no minimum quantum) and 16KBytes respectively. To set a minimum, configure the T*xQueueWeight_Configuration* register bits [14:11] *MQShiftVal* field. While a queue is being serviced, the length of each transmitted PDU gets subtracted from the queue's MQ Counter. Once the counter reaches <= 0, the queue is done being serviced. The flow engine is released and the MQ Counter resets to its initial value.

For complete details about registers related to the weighting algorithm, refer to: "TxQueueWeight_Configuration Register (FP Tx Configuration Function)" on page 676, and "Absolute Priority_Configuration Register (FP Tx Configuration Function)" on page 690.

FPTx Error Reporting and Interrupts

Four (4) error types are detected by the FPTx and logged in bits [31:28] of the *TxFCE_Configuration* register. In addition to being logged, these errors cause an interrupt to be sent to the XP (if bit [26] *IntEnable* field in the *TxFCE_Configuration* register is asserted). These error bits remain asserted until bit [27] *IntAck* field in the *TxFCE_Configuration* register is written to a 1. Bit [27] *IntAck* field must then be set to a 0 again. When writing the register, be careful *not* to change the other configuration fields, such as bits [25:24] *DescSize*. Refer to Table 42 on page 201 and "TxFCE_Configuration Register (FP Tx Configuration Function)" on page 679.

Table 42 FPTx Four (4) Error Types and Descriptions

ERROR TYPES	BIT	DESCRIPTION
Descriptor (QMU) Parity Error	31	Indicates that a parity error occurred when the QMU sent a descriptor. This error is only logged if bit [21] *QMUParityErrorEnable* field is set. In this case, the error has a separate enable in addition to the interrupt enable. If the error occurs (and is enabled), no additional PDUs are transmitted for the queue in question. Further, no PDUs are transmitted for the lower priority queues in the same port.
Buffer (BMU) Read Error	30	A buffer read error may occur for a number of different reasons. • **ECC error** - BMU detects an ECC error when it reads the buffer from SDRAM. • **Retry time-out** - BMU is unable to satisfy the buffer read request because it is too busy. • **Non-existent memory error** - This only occurs if the BMU were misconfigured to be storing buffers in a non-existent memory location. Note: Whatever data is transferred from the BMU is transmitted.

Table 42 FPTx Four (4) Error Types and Descriptions (continued)

ERROR TYPES	BIT	DESCRIPTION
Write (BMU) Error	29	A buffer write error may occur for a number of different reasons. • **Retry Time-out** - A BTag deallocate operation or a multi-use counter (MUC) operation could fail because the BMU is too busy to service the request or because the pool ID used for the operation is invalid. The invalid pool ID errors only occurs if the pool value in a descriptor passed to the FPTx is invalid or the FPRx was illegally configured to use an invalid pool. The BMU should never be too busy to service these operations. • **Multi-use Counter Decrement Error** - Multiuse counter decrement operations can also fail if the multi-use counter does not exist in the BMU. This could only occur in the event of a CP/XP software error; for example, if it failed to set up the counter, initialized it to a value which was too small, or decremented a counter more than once after transmitting. Note: If one of these write errors occurs, the FPTx will *not* retry the operation and the buffer is effectively leaked because its BTag is never freed.
Dequeue (QMU) Failure	28	A dequeue operation fails if the queue is empty when the request is made. A dequeue from an empty queue should only occur in the event of a CP/XP programming error where a CP or XP removes something from a queue belonging to the FPTx.

FP Receive (FPRx) Sequence

The FPRx performs essentially the same receive function that a Channel Processor (CP) does, but with a high-performance and mostly hard-wired implementation.

In general, the FPRx receives segments and writes them to BMU buffers, reassembling them into PDUs (up to 159 concurrently), while building and enqueuing the associated descriptors.

The basic sequence of a PDU being received by the FPRx consists of six (6) steps. The six (6) steps include: Fabric Interfaces Receive Operation, FPRx Header and Payload Splitting, FPRx Byte Processors Microcoding, FPRx Writing Payload, FPRx Building Descriptors, and FPRx Enqueuing PDUs.

The basic FPRx PDU sequence is indicated in Table 43 on page 203, and Figure 46 on page 205 illustrates both the sequence and main components of the FPRx. In addition, Figure 47 on page 206 shows the Global Address Memory Map of the FPRx.

Table 43 FPRx PDU Sequence and Reference to Details

FPRX SEQUENCE	SEQUENCE NUMBER		DETAILS
Fabric Interface Receive Operation	1	A segment arrives at fabric interface.	Refer to "Fabric Interface Receive Operation" on page 207.
	1a	The segment data is adjusted according to endianness. Refer to "FP Endianness (Byte and Bit Ordering)" on page 241.	
	1b	The segment CRC is checked.	
	1c	The in-band link-level flow control information extracted from the header (CSIX-L1 mode and PRIZMA mode).	
FPRx Header and Payload Splitting	2	The segment header is directed to a RxByte Processor's Header FIFO (subsequent segments alternate between the two (2) RxByte Processors); the payload is sent to the single Payload FIFO.	Refer to "FPRx Header and Payload Splitting" on page 207.

Table 43 FPRx PDU Sequence and Reference to Details (continued)

FPRX SEQUENCE	SEQUENCE NUMBER		DETAILS
FPRx Byte Processors Microcoding	3	The microcode programmable RxByte Processor processes the header:	"FP RxByte Processors Microcoding" on page 208.
	3a	It extracts PDU ID and segment type.	
	3b	It extracts PDU length (not necessary if using the default PDU length option).	
	3c	It processes in-band per-queue flow control (optional).	
	3d	It launches TLU lookup (optional).	
	3e	It saves header content for descriptor building (optional).	
FPRx Writing Payload	4	If this is the first segment of a PDU, then a buffer is selected for payload, based on the PDU length.	Refer to "FPRx Writing Payload" on page 223.
	4a	The payload is written to the BMU buffer from DMEM.	
FPRx Building Descriptors	5	The TLU response is returned (optional).	Refer to "FPRx Building Descriptors" on page 225.
	5a	If this is the first segment of a PDU, the descriptor is built.	
FPRx Enqueuing PDUs	6	If this is the last segment of a PDU, the descriptor is enqueued.	Refer to "FPRx Enqueuing PDUs" on page 233.

In addition, there is one related FPRx topic that is outside the basic sequence. Refer to "FPRx Interrupts" on page 236.

Furthermore, for a description of functions that span both the FPRx and FPTx such as: FP Flow Control (Link-Level and Per-Queue), FP Descriptor Size, FP CRC, FP Endianness, and FP Payload Bus Bandwidth, please refer to "FPTx and FPRx General Considerations" on page 237.

Figure 46 FPRx Sequence and Block Diagram

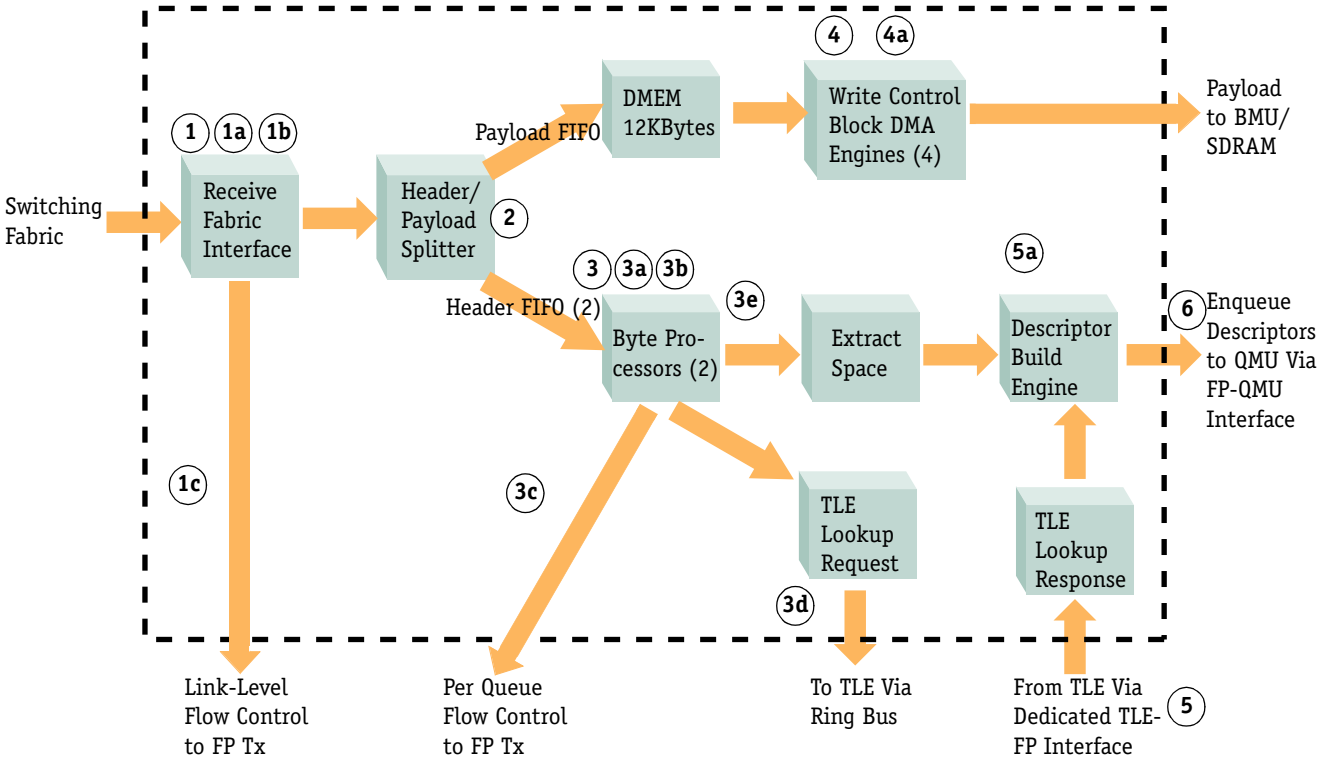


Figure 47 FPRx Global Address Memory Map

Debug State	0xBDE04700
Configuration Space & Statistics	0xBDE04600
TLU Response 256Bytes	0xBDE04500
Ring Bus	0xBDE04440
RdCB1	0xBDE04430
RdCB0	0xBDE04420
WrCB3	0xBDE04380
RxByte Processor1	oxBDE04290
WrCB2	0xBDE04280
Extract 1 (128Bytes)	0xBDE04200
WrCB1	0xBDE04180
RxByte Processor0	0xBDE04090
WrCB0	0xBDE04080
Extract0 (128Bytes)	0xBDE04000
	0xBDE03000
DMEM (12K)	0xBDE00000

Fabric Interface Receive Operation

The fabric interface represents the front-end of the FPRx pipeline. It is here that segments are received and link-level flow control performed, in compliance with the configured protocol (for example, CSIX-L1). The front-end hardware also performs initial error checking (parity, CRC, segment length) and makes adjustments for endianness. All received segments are then passed to the Header/Payload Splitter.

FPRx Header and Payload Splitting

Segments are split into headers and payload, which are in turn forwarded through separate pipelines. To accommodate different segments types, up to three (3) different splits between the header size and the payload size are allowed per application. The segment type must be identifiable by checking the masked value of some single Byte of the segment. Which Byte to check, the mask, and the comparison value are all configurable using the *RxDS_Header_Change1* and *RxDS_Header_Change2* registers. Segment type checking is done in a prioritized order with the first match specifying the type. If neither of the change registers matches, a configurable default splitting is done. Refer to "RxDS_Header_Change1 Register (FP Rx Configuration Function)" on page 699 and "RxDS_Header/Payload_Delimiter0 Register (FP Rx Configuration Function)" on page 700.

In general, as segments arrive, their headers are directed alternately to the Header FIFOs belonging to the two (2) RxByte Processors. All payload is directed into a single Payload FIFO.

Specifically, for each segment type, the number of segment bytes to be directed to the header FIFO of a RxByte Processor for header processing is configurable. Headers are always presumed to begin with the first byte of the segment (byte offset is zero). The portion of the segment which will be directed to the payload FIFO, ultimately to be stored in a BMU buffer, is also configurable. Header/payload regions are configured using *RxDS_Header/Payload_Delimiter0*, *RxDS_Header/Payload_Delimiter1*, and *RxDS_Header/Payload_Delimiter2* registers. These three (3) registers allow you to specify a header and payload which overlap, allowing as much or all of the header to be included in the payload or leaving a gap between header and payload. Refer to "RxDS_Header/Payload_Delimiter0 Register (FP Rx Configuration Function)" on page 700, and Table 227 on page 700.

RxDS_Header/Payload_Delimiter0 register *must* always be used and properly configured. *RxDS_Header/Payload_Delimiter1* and *RxDS_Header/Payload_Delimiter2* are optional. Freescale recommends that if using more than one (1) delimiter (*RxDS_Header/Payload_Delimiter0)*, use *RxDS_Header/Payload_Delimiter1* before using *RxDS_Header/Payload_Delimiter2*.

```
If (change reg 2 enabled AND (segment[change 2 reg index] &
   change reg 2 mask) ==
   change reg 2 value))
   Use delimiter reg 2
Else If (change reg 1 enabled AND (segment[change 1 reg
   index] & change reg 1 mask) == change reg 1 value))
   Use delimiter reg 1
Else
   Use delimiter reg 0
```

In the equation above, "&" means "bitwise".

The Payload Last FPRx index must have the lowest two bits (LSBs) set (that is, 0xmmmmmm11) as C-5e NP only supports 4Byte multiples for both payload and header. Correspondingly, the Payload First Index must have the two LSBs cleared (that is, 0xmmmmmm00).

The payload cannot end on byte 3 of the cell, so the Payload Last index must be greater than 3.

FP RxByte Processors Microcoding

The FPRx includes two (2) microcode programmable FP RxByte Processors, similar to those in the Channel Processors (CPs). These RxByte Processors are used to process the header for each segment; header formats vary by fabric protocol and application. Headers for consecutive segments are processed alternately by one (1) of the two (2) RxByte Processors. Only a single microcode program can be loaded for both RxByte Processors, however there are ways microcode execution can be made unique for each RxByte Processor.

FP RxByte Processors

In general, FP RxByte Processor microcoding is done much like microcoding for a SDP Byte Processor. The FP RxByte Processor is capable of the same sequencing and ALU operations as the SDP Byte Processor. The RxByte Processor has 24 entries in its CAM, just like a SDP. There are two (2) unique aspects of the RxByte Processor, compared to a SDP: different number of control store entries, and different external test conditions.

Control Store Entries

The RxByte Processor has 96 control store entries as opposed to a CP SDP Byte Processor that has 64.

External Test Conditions

There are eight (8) external test conditions available to the RxByte Processor as described in the *C-Ware Microcode Programming Guide (part number CSTMCPG-UG/D)*. Bits [0, 1, 4, 5, 6, 7] are used by the RxByte Processor as detailed in Table 44 on page 209. Therefore, bits [2:3] are *not* used.

Table 44 RxByte Processor External Test Conditions

BIT	ITEM	DESCRIPTION
0	Input data valid	This test condition indicates that the byte feeding the Processor is valid.
1	Token	This test condition can be used to test for ownership of a token passed between the two (2) Byte Processors.
2	Not Used	
3	Not Used	
4	Header FIFO Overflow	This test condition is used to detect overflows. Overflows are typically prevented using the *RxDS_Configuration* register bits [26:23] *HdrFIFOXOFF* field (backpressure mechanism). Certain applications can overwhelm the backpressure mechanism. For example, when the headers are large, and the link-level response time is too long. For these types of applications, this condition should be checked at the start of each header (within the first 4Byte word). When a Header FIFO Overflow condition is detected, it means the header was cutoff so the microcode must drop the segment and not use the header data. Note: The microcode can still use the First/Last Header Byte test condition in this case, even though the end of the header was cutoff. The last byte of the header that was *not* cutoff is flagged as Last Header Byte.
5	First/Last Header Byte	This test condition can be used to test if the byte currently being unloaded from the header FIFO is the first or last byte of the header. It is configurable whether this test condition indicates the first or last byte of the header via the RxByte Processor End Of Header field of the *RxDS_Configuration* register. Refer to "RxDS_Configuration Register (FP Rx Configuration Function)" on page 701.
6	Drop Mode	This test condition indicates that a segment needs to be dropped because the payload FIFO is full (512Bytes). Should only be applicable to PRIZMA applications. Refer to "RxByte Processor's Drop Mode" on page 260.

Table 44 RxByte Processor External Test Conditions (continued)

BIT	ITEM	DESCRIPTION
7	Control Word	This test condition indicates that the byte currently being unloaded is coming from the control FIFO, *not* the header FIFO. The control FIFO is *only* used in PowerX(CSIX-L0) mode so details of its usage are provided in the PowerX(CSIX-L0) section. Refer to "PowerX(CSIX-L0) Interface Mode" on page 263.

RxByte Processors Microcoding Minimum Requirements

For the RxByte Processor's microcode to operate properly, these minimum requirements must be followed:

- Wait for datascope ownership.
- Set up a PDU ID and segment type for the segment.
- Remove all bytes for that header from the Header FIFO.
- Flip ownership for that datascope.

Other functions the RxByte Processor may perform include:

- Setting up a PDU length for first or only segments, if a default PDU length isn't used.
- Launching a TLU lookup for use in descriptor building and enqueuing.
- Processing per-queue flow control information, and sending flow control messages to the FPTx.
- Storing header data in extract space for descriptor building.

RxByte Processor Memory Space and Registers

The RxByte Processor contains five (5) types of memory space, they include: Extract Space, Control Space, RxByte General Purpose Configuration Space, RxByte Shared Space, and Ring Bus Space. Each memory space provides a different function and each space consists of a number of registers specific to that function. Both the Extract Space and Control Space are used to pass information between the RxByte Processor and associated FP hardware. In contrast, the RxByte General Purpose Configuration Space does *not* pass information between the RxByte Processor and associated FP hardware. All of the registers within all five (5) of these spaces are mapped to Global Address Space for debug purposes.

Table 45 on page 211 lists the five (5) Memory Spaces of a RxByte Processor and descriptions, Figure 48 on page 213 shows the RxByte Processor's Memory Map with specific registers, and Table 46 on page 214 provides a mapping of the Global Addresses (32bits) to the RxByte Processor's Byte Address (8bits) with descriptions, and access information for Global Bus, RxByte Processor and Hardware. The information in Table 45 on page 211, Figure 48 on page 213, and Table 46 on page 214 pertain to only a *single* RxByte Processor.

Table 45 RxByte Processor Memory Space and Descriptions

MEMORY SPACE	DESCRIPTION
Extract Space	Each RxByte Processor has 128Bytes of Extract Space, divided across the number of Scopes (32Bytes descriptors implies 32Bytes of Extract Space for each of four (4) Scopes per processor). Refer to "RxExtractSpace0/" on page 214, and "RxExtractSpace1" on page 214.
Control Space	Control information unique to the RxByte Processor. The RxByte Processor control space consists of: RxStatus, RxFlowSegment, RxFlowSize, and RxTxCongestion. • For RxStatus refer to: "RxStatus0/" on page 214, and "RxStatus1" on page 214. • For RxFlowSegment refer to: "RxFlowSeg0/" on page 214, and "RxFlowSeg1" on page 214. • For RxFlowSize refer to: "RxFlowSize0/" on page 214, and "RxFlowSize1" on page 214 • For RxTxCongestion refer to: "RxTxcgs0/" on page 215, and "RxTxcgs1" on page 215.
RxByte General Purpose Configuration Space	4Byte RxByte General Purpose Configuration registers (4Bytes for each RxByte Processor). Refer to "RxByte0 General Purpose Configuration/" on page 215, and "RxByte1 General Purpose Configuration" on page 215.
RxByte Shared Space	Two (2) 4Byte RxByte Shared registers (8Bytes total accessible by either RxByte Processor0 or RxByte Processor1). Refer to "RxByte_Shared_Low/" on page 215, and "RxByte_Shared_High" on page 215.

Table 45 RxByte Processor Memory Space and Descriptions (continued)

MEMORY SPACE	DESCRIPTION
Ring Bus Space	Contains four (4) sets of Ring Bus registers. • For Ring Bus0 refer to: "Ring Bus TxMsg0_CTL" on page 216, "Ring Bus TxMsg0_D0H/L" on page 217, and "Ring Bus TxMsg0_D1H/L" on page 217 • For Ring Bus1 refer to: "Ring Bus TxMsg1_CTL" on page 217, "Ring Bus TxMsg1_D0H/L" on page 217, and "Ring Bus TxMsg1_D1H/L" on page 217 • For Ring Bus2 refer to: "Ring Bus TxMsg2_CTL" on page 217, "Ring Bus TxMsg2_D0H/L" on page 217, and "Ring Bus TxMsg2_D1H/L" on page 217 • For Ring Bus3 refer to: "Ring Bus TxMsg3_CTL" on page 217, "Ring Bus TxMsg3_D0H/L" on page 217, and "Ring Bus TxMsg3_D1H/L" on page 217

All of the individual Byte registers for the RxByte Processor's (RxByte Processor0 and RxByte Processor1) are detailed in Appendix A since they are accessible via the Global Bus. Refer to Table 46 on page 214 where they are listed by their Global Address and Appendix A where their individual fields are detailed using their Global addresses.

The RxByte Processor is analogous to the RxSync Processor in the CP's SDP.

Refer to the *C-Ware Microcode Programming Guide (part number CSTMCPG-UG/D).*

Figure 48 RxByte Processor Memory Map

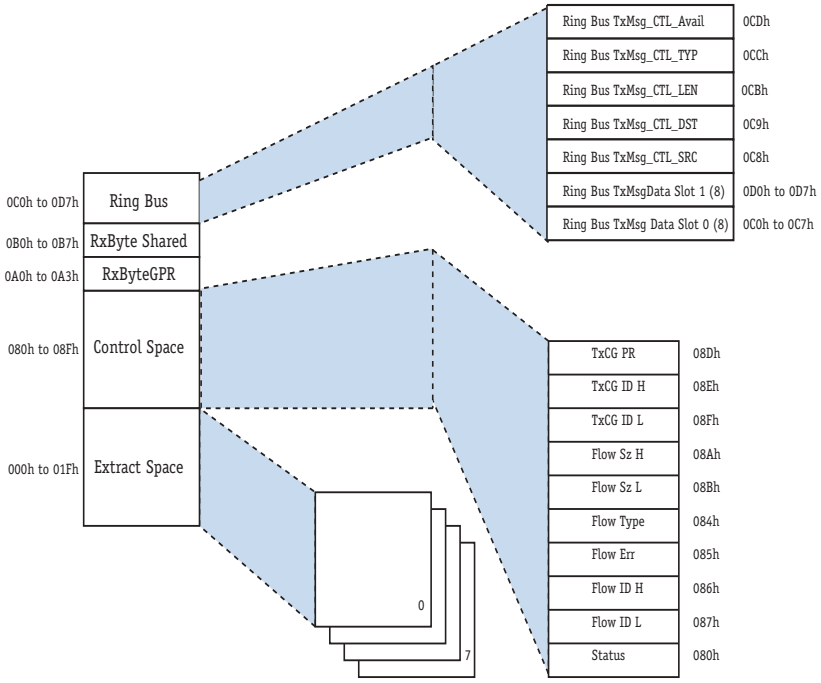


Table 46 RxByte Processor Mapping and Details

GLOBAL ADDRESS OFFSET FROM 0XBDE04000 (32BITS) (RXBYTE0/RXBYTE1)	RXBYTE PROCESSOR ADDRESS (8BITS)	NAME	DESCRIPTION	ACCESS VIA GLOBAL BUS	ACCESS VIA RXBYTE PROCESSOR	ACCESS VIA HARDWARE
000h-07Fh/ 200h-27Fh	000-01Fh 1 (Range/Field)	RxExtractSpace0/ RxExtractSpace1	A block of 128Bytes of Extract space divided by the number of Scopes per RxByte Processor.	R/W	W	R
0090h/ 0290h	080h-083h (Range)	RxStatus0/ RxStatus1	RxStatus Register2. This register is used to pass ownership of a scope between the RxByte Processor and the hardware.	R/W	R/W	R/W
	• 080h (Field)	• RxStatus	RxStatus Byte [2]. Bit[7] is the Own bit (0=RxByte has ownership, 1=hardware has ownership). Bits [6:0] are reserved. Bit [0] *must* be set to 0.			
	• 081h-083h (Field)	• Unused	Unused			
094h-097h/ 0294h-0297h	084h- 87h (Range)	RxFlowSeg0/ RxFlowSeg1	RxFlow Segment Register [2, 3]. This register is used to associate a segment with a PDU.	R/W	W	R
	• 086h–087h (Field)	• RxFlow_Seg[15:0]	PDU ID			
	• 085h (Field)	• RxFlow_Seg[17:16]	Flow Discard bit [17], Flow Error bit [16]			
	• 084h (Field)	• RxFlow_Seg[25:24]	Segment Type			
098h-09Bh/ 0298h-029Bh	088-08Bh (Range)	RxFlowSize0/ RxFlowSize1	RxFlow Size Register [2, 3]. This register specifies the PDU length.	R/W	W	R
	• 088-089h (Field)	• Unused	Unused			
	• 08A-08Bh (Field)	• RxFlow Size[15:0]	RxFlow Size [2, 3]			

Table 46 RxByte Processor Mapping and Details (continued)

GLOBAL ADDRESS OFFSET FROM 0XBDE04000 (32BITS) (RXBYTE0/RXBYTE1)	RXBYTE PROCESSOR ADDRESS (8BITS)	NAME	DESCRIPTION	ACCESS VIA GLOBAL BUS	ACCESS VIA RXBYTE PROCESSOR	ACCESS VIA HARDWARE
09Ch-09Fh/ 029Ch-029Fh	08Ch-08Fh (Range)	RxTxcgs0/ RxTxcgs1	RxTxCongestion Register[2, 3]. This register is used to send a per-queue flow control message to the FPTx.	R/W	W	R
	• 08Ch (Field)	• Unused	Unused			
	• 08Dh (Field)	• RxTxcgs[21]	Pause / Resume (1 = pause). A write to this bit triggers the message to be sent to the FPTx.			
	• 08Eh-08Fh (Field)	• RxTxcgs[15:0]	Flow ID			
0628h-062Bh/ 062Ch-062Fh	0A0h-0A3h (Range/Field)	RxByte0 General Purpose Configuration/ RxByte1 General Purpose Configuration	4Byte RxByte General Purpose Configuration register (4Bytes for each RxByte Processor) 3	R/W	R	N/A
0660h-0667h	0B0h-0B7h (Range/Field)	RxByte_Shared_Low/ RxByte_Shared_High	Two (2) 4Byte RxByte Shared registers (8Bytes total accessible by either RxByte0 or RxByte1) [3]	R	R/W	N/A

Table 46 RxByte Processor Mapping and Details (continued)

<table>
<tr><th>GLOBAL ADDRESS OFFSET FROM 0XBDE04000 (32BITS) (RXBYTE0/RXBYTE1)</th><th>RXBYTE PROCESSOR ADDRESS (8BITS)</th><th>NAME</th><th>DESCRIPTION</th><th>ACCESS VIA GLOBAL BUS</th><th>ACCESS VIA RXBYTE PROCESSOR</th><th>ACCESS VIA HARDWARE</th></tr>
<tr><td rowspan="8">440h</td><td>0C8h–0CDh (Range)</td><td>Ring Bus TxMsg0_CTL</td><td>Ring Bus0 (TxMsg) control 4,5</td><td rowspan="8">R/W</td><td rowspan="8">W</td><td rowspan="8">N/A</td></tr>
<tr><td>0CDh (Field)</td><td>Ring Bus TxMsg0_CTL[31]</td><td>Ring Bus0 Avail [4]</td></tr>
<tr><td>0CCh (Field)</td><td>Ring Bus TxMsg0_CTL[19:18]</td><td>Ring Bus0 Type</td></tr>
<tr><td>0CBh (Field)</td><td>Ring Bus TxMsg0_CTL[17:15]</td><td>Ring Bus0 Length – Bit 17 hardwired to 0. Legal values are detailed here:
<table><tr><th>ENCODED VALUE</th><th>LENGTH OF REQUEST</th></tr><tr><td>001</td><td>8Bytes (1Slot)</td></tr><tr><td>010</td><td>16Bytes (2Slots)</td></tr></table></td></tr>
<tr><td>0CAh</td><td>—</td><td>Ring Bus0 Sequence number is set by hardware based on the Scope, and not writable by the RxByte Processor or the Global Bus.</td></tr>
<tr><td>0C9h (Field)</td><td>Ring Bus TxMsg0_CTL[9:5]</td><td>Ring Bus0 Destination</td></tr>
<tr><td>0C8h (Field)</td><td>Ring Bus TxMsg0_CTL[4:0]</td><td>Ring Bus0 Source</td></tr>
</table>

Table 46 RxByte Processor Mapping and Details (continued)

GLOBAL ADDRESS OFFSET FROM 0XBDE04000 (32BITS) (RXBYTE0/RXBYTE1)	RXBYTE PROCESSOR ADDRESS (8BITS)	NAME	DESCRIPTION	ACCESS VIA GLOBAL BUS	ACCESS VIA RXBYTE PROCESSOR	ACCESS VIA HARDWARE
460h / 464h	C0 – C7h (Range/Field)	Ring Bus TxMsg0_D0H/L	High and Low Data for Slot0 (See WARNING below) 6, [3]	R/W	W	N/A
480h / 484h	D0 – D7h (Range/Field)	Ring Bus TxMsg0_D1H/L	High and Low Data for Slot17	R/W	W	
448h	C8h-CDh (Range/Field)	Ring Bus TxMsg1_CTL	Ring Bus1 (TxMsg) control	R/W	W	
468h / 46Ch	C0 – C7h (Range/Field)	Ring Bus TxMsg1_D0H/L		R/W	W	
488h / 48Ch	D0 – D7h (Range/Field)	Ring Bus TxMsg1_D1H/L	Note: Maps to RB1 D1 H/L	R/W	W	
450h	C8h-CDh (Range/Field)	Ring Bus TxMsg2_CTL	Ring Bus2 (TxMsg) control	R/W	W	
470h / 474h	C0 – C7h (Range/Field)	Ring Bus TxMsg2_D0H/L		R/W	W	
490h / 494h	D0 – D7h (Range/Field)	Ring Bus TxMsg2_D1H/L	Note: Maps to Ring Bus2 D1 H/L	R/W	W	
458h	C8h-CDh (Range/Field)	Ring Bus TxMsg3_CTL	Ring Bus3 (TxMsg) control	R/W	W	N/A
478h / 47Ch	C0 – C7h (Range/Field)	Ring Bus TxMsg3_D0H/L		R/W	W	
498h / 49Ch	D0 – D7h (Range/Field)	Ring Bus TxMsg3_D1H/L	Note: Maps to Ring Bus3 D1 H/L	R/W	W	

[1] 16 or 32 bytes mapped to RxByte Processor by datascope index: 0x00 to 0x1F (for 4 scopes per RxByte Processor, or 0x00 to 0x0F (for 8 scopes per RxByte Processor).

[2] 0 / 1 refers to RxByte Processor register set 0 and 1 respectively.

[3] The RxByte Processor uses little endian addressing for the bytes within this register (least significant byte is associated with the highest address).

[4] The TxMsg_CTL register fields are spread out over a range of six (6) 1Byte registers (address 0CAh is not accessible) from the RxByte Processor's perspective (as listed). However, global accesses to the register can be done with one 32bit access.

[5] Since the RxByte Processor accesses one Byte at a time, and the 5 Ring Bus Control fields (Avail, Type, Length, Destination, and Source) are less than a full Byte, keep in mind that the fields are right-justified (that is, the Avail bit is at position 0 within the Avail Register (Ring Bus TxMsg0_CTL[31]).

[6] Segments are automatically assigned to one of the 4 sets of Ring Bus registers as indicated here:

RING BUS REGISTER SET	AVAILABLE FOR USE BY:
0	RxByte0 segments with an even-numbered scope.
1	RxByte1 segments with an even-numbered scope.
2	RxByte0 segments with an odd-numbered scope.
3	RxByte1 segments with an odd-numbered scope.

[7] For requests with a length of 2Slots (16Bytes), the second slot of data comes from the Ring Bus *Tx Msg_D1* register. This is different from multi-slot requests made from the CP (in the CPs, there is no *TxMsg_D1* register; instead slots of data from multiple *TxMsg_D0* registers are strung together).

Warning: *Since there are two (2) sets of Ring Bus registers per RxByte Processor, yet only one (1) set is visible for a given segment, the RxByte Processor's microcode cannot perform a single initialization of the Ring Bus fields. It must initialize both sets of Ring Bus registers, or allow global initialization using the XP.*

RxByte Processors Datascopes

The FPRx provides each of the two (2) RxByte Processors with a number of datascopes or contexts into which it can store data. The number of datascopes per RxByte Processor depends on the descriptor size:

- When 12Byte or 16Byte descriptors are being used, then each RxByte Processor has eight (8) datascopes available to it.
- When 24Byte or 32Byte descriptors are being used, each RxByte Processor has four (4) datascopes available to it.

There are times when datascopes are *not* available to a RxByte Processor because FPRx has not finished consuming its contents. Because of this, microcode must wait until it is granted ownership of a datascope before it begins processing a header. Microcode tests for datascope availability using the *RxStatus1* register bit [7] *Ownership* field. Refer to Table 222 on page 692.

When RxByte Processor's microcode has finished processing a header, it passes ownership for the datascope to hardware by setting the *RxStatus1* register bit [7] *Ownership* field. This indicates that all of the data has been set up and the hardware can begin processing the segment. The hardware uses the datascope number as an index into extract space and TLU response space. The datascope also serves to keep the PDUs in order. Once the hardware is done handling the payload and enqueuing the PDU, the datascope is freed up.

RxByte Processors Set Up Control Information

The RxByte Processor's microcode must set up certain control information for each segment. At a minimum it must set up two (2) items: a PDU ID, and a segment type. In addition, the microcode must set up the PDU length unless, the default PDU length option is enabled.

- The PDU ID is a 16bit value that associates the payload within the segment with the payload from other segments from the same PDU. There is a configurable mask that the FPRx applies to the PDU ID before this association is done. The PDU ID mask is configured using the *RxFCE_Configuration0* register bits [15:0] *PDU IDMask* field. Typically the PDU ID is extracted from the header. The 16bit PDU ID value is written to the *RxFlowSeg1* register bits [15:0] *PDUID* field of the RxByte Processor. Refer to "RxFCE_Configuration0 Register (FP Rx Configuration Function)" on page 706 and Table 223 on page 694.
- The segment type indicates whether the segment is a first, middle, last, or only segment for the PDU. Typically the segment type is extracted from the header.
- The PDU length indicates the number of bytes of payload in the PDU and is used to find an appropriately sized buffer. Typically, the PDU length is extracted from the header of a first or only segment and written to the *RxFlowSz1* register. Refer to Table 224 on page 694. When the PDU length is *not* known, the FPRx can optionally use a default length value for all PDUs and thus, the PDU length need not be written by microcode. In this case, the same PDU length is presumed for all PDUs and the default length should be set up to be greater than or equal to the maximum PDU size. This option is configured using the *RxFCE_Configuration1* register. Refer to "RxFCE_Configuration1 Register (FP Rx Configuration Function)" on page 707.

When the default PDU length option is used, the FPRx error check which detects the premature arrival of a last segment must be disabled; last segments may appear to be premature because the PDU length is assumed to be a maximum length. Also, when this mode is used, all PDUs are delivered to the same BMU pool because the length being used to select a pool will always be the same.

RxByte Processors Writing to Extract Space

The RxByte Processors can write bytes to extract space, in order for those bytes to be made available for descriptor building. The number of bytes available per datascope depends on the descriptor size. When the FP is configured to use 12Byte or 16Byte descriptors, 16Bytes of extract space are available. Otherwise, 32Bytes of extract space are available.

Because the Descriptor Build Engine (DBE) may start building a descriptor as soon as it receives ownership, microcode must finish writing all bytes to extract space before passing ownership of the first segment to the hardware. Descriptors are *not* built for either middle or last segments. Therefore, the microcode for those segment types need not write any data to the extract space.

RxByte Processors Performing TLU Lookups

The RxByte Processor can launch requests on the Ring Bus to perform TLU (Table Lookup Unit) lookups. In general, the requests are made using four (4) sets of Ring Bus *TxMsgn_Ctl* and *TxMsgn_Data0H/L* registers (plus T*xMsgn_Data1* in the case of 16Byte requests). These registers are very similar to the Ring Bus Transmit registers in the CP; the exceptions are detailed in Table 44 on page 209. Many of the register fields can be statically configured via global writes: Type, Length of request (8Bytes or 16Bytes), Source Ring Bus Node, and Destination Ring Bus Node. Fields which are unique to each segment can be written by the RxByte Processor. Examples of these are the key and index, which are a part of the Ring Bus data (that is,*TxMsgn_Data0_H/L*). After filling out any necessary request fields and checking the Ring Bus *TxMsgn_Ctl* register bit [31] *Avail* field, the microcode can clear the *Avail* bit to launch the request on to the Ring Bus.

The responses to the TLU lookups come back via a dedicated TLU->FP interface, as opposed to via the Ring Bus. The data is placed in response memory (256Bytes) where it can be accessed by the Descriptor Build Engine (DBE).

RxByte Processor Microcode Programming Guidelines for the TLU Interface

The following guidelines should be used when microcode programming the RxByte Processor to interface with the TLU.

- Since the FP does not use the Ring Bus for responses (only requests), do *not* send any Ring Bus messages to the FP. A message targeted for the FP would circulate the Ring indefinitely, and enough of these would degrade Ring Bus performance to zero. For this reason, the only node that the FP should send messages to is the TLU, which has a dedicated response interface to the FP. The exception is proxy requests; the FP can safely send these to any node on the Ring Bus because there is no response.
- Only one (1) TLU operation can be performed per-segment (that is, the sequence number of the request and the index into response memory are determined by the datascope, and there is only one (1) datascope per segment).
- TLU operations can *only* be launched on first/only segments, because those are the only segment types for which a descriptor is built.
- TLU operations, if used, must always be launched on every first/only segment. If the microcode wants to discard a segment via the Discard or Error indicators, it should set the segment type to middle and not launch a TLU operation. If the segment type remains first/only, a TLU operation would have to be launched.
- The response size can be 8Bytes, 16Bytes, or 32Bytes. This response size is a function of fields within the lookup request data (that is, TxMsg*n*_Datan_H/L). The size of the request and size of the response are independent, but the response size cannot be greater than the size of the response memory (256Bytes) divided by the number of datascopes. For example, with a 16-datascope configuration (that is, 16Byte descriptors), the response size can be 8Bytes or 16Bytes. For 8-datascope configurations (that is, 24Bytes or 32Bytes descriptors), the response size can be 8, 16, or 32.

Do not use 32Byte responses with 16Byte descriptors.

RxByte Shared Registers

There are a total of 8Bytes of shared registers (*RxByte_Shared_Low* and *RxByte_Shared_High*) which are accessible by either RxByte Processor0 or RxByte Processor1. These registers can be initialized via global writes. Both RxByte Processors have read and write access to the 8Bytes of shared space. The RxByte Processors can read any byte at any time. The microcode should prevent the RxByte Processors from writing the same byte at the same time, or else the resulting value will be undefined. All other combination of writes are allowed, including simultaneous writes to different bytes. Refer to "RxByte_Shared_Low/" on page 215 in Table 46 on page 214.

RxByte General Purpose Configuration Registers

There are two (2) 4Byte general purpose configuration registers (*RxByte 0 General Purpose Configuration* and *RxByte1 General Purpose Configuration*), one (1) for each RxByte Processor. These registers can be initialized via global writes. The 4Byte configuration registers act as separate, private storage for each RxByte Processor. Typically, they are used to customize the execution behavior of one (1) RxByte Processor from the other. They are configured via global writes, and can be read (not written) by the associated Byte Processor. Refer to "RxByte0 General Purpose Configuration/" on page 215 in Table 46 on page 214.

RxByte Processors Discarding Segments

Sometimes the RxByte Processor may want to discard a segment. A common use for discarding segments would be to discard "idle" segments which only convey flow control information.

To discard a segment, RxByte Processor's microcode must:

- Set the segment type to be a middle segment
- Set the *RxFlowSeg0* register bit [17] *FlowDisc* field or bit [16] *FlowError* field. Refer to "RxFlowSeg0 Register (FP RxByte Processor Function)" on page 693. Either of these bits causes the segment to be dropped, and increments a separate counter in the FPRx statistics registers.
- Pass ownership of the segment to hardware

The result of discarding a segment is that any payload associated with the segment in the payload FIFO is dropped and not written to any BMU buffer.

RxByte Processors Token Passing

If needed for synchronization purposes, a token can be passed between the two (2) RxByte Processors. After reset, the token is owned by RxByte Processor0. To pass the token to the other RxByte Processor, a RxByte Processor should set *Token Out*, which is bit [2] in the Processor's internal control register. Refer to the *C-Ware Microcode Programming Guide (part number CSTMCPG-UG/D)* for specific details about the *Token Out* bit. The *Token Out* bit must have been previously cleared, so that there is a 0 ->1 transition. Ownership of the token bit can be tested by using the RxByte Processors external test condition bit 1.

A common reason for using a token is to implement a semaphore for writing the 8Bytes of RxByte Shared Registers. For example, only the RxByte Processor that owns the token is allowed to write to a certain location in the RxByte Shared Registers. This prevents data corruption that could occur if both RxByte Processors wrote to the same location at the same time.

FPRx Writing Payload

As the FPRx receives PDUs it must determine which BMU buffer they will be stored in. The FPRx can be configured to use buffers in up to four (4) of the BMU's thirty (30) pools. Which of the thirty (30) BMU pools the FPRx uses is configurable in the BMU. Refer to "BMU Configuration Space" on page 307.

To prepare for incoming PDUs, the FPRx keeps a store of BTags for each pool that it is using. It can store a maximum of 256 BTags (8 blocks of 32 BTags) per pool. When the FPRx is enabled, it requests BTags from the BMU to fill its store for each pool up to a configured maximum for each pool. As PDUs are received and BTags used, the FPRx BTag stores are depleted. When the number of BTags in a store drops below a configurable threshold, the FPRx begins requesting more BTags from the BMU and continues to do so until it has filled to its configurable maximum.

If the FPRx requests BTags from the BMU and the BMU cannot satisfy that request, a statistics register is incremented and the "No BTags available from BMU" interrupt (if enabled) is sent to the XP. The FP Rx continues to request until the request is satisfied. If the request cannot be completed after 16 attempts, a "BTag allocation timeout" interrupt (if enabled) is generated.

FPRx buffer pool configuration is done using two (2) configuration registers; *PooIn_CFG0* and *PooIn_CFG1*, where n=0,1,2,3. These two (2) registers are associated with each of the four (4) buffer pools. Refer to"Pool0_CFG0 Register (FP Rx Pool Configuration Function)" on page 710, and "Pool0_CFG1 Register (FP Rx Pool Configuration Function)" on page 711.

Storing the Payload to the BMU Process

When the first segment of a PDU arrives, the RxByte Processor extracts the PDU length from the header. This length is used by the FPRx to select a BMU buffer large enough for the PDU. The FPRx compares this length to the buffer sizes for the four (4) pools that it has been configured to use. It checks the pools in order (from FPRx pool0 to FPRx pool3) and when the first match is found, it tries to use a buffer from that pool. If there are no BTags available for that pool, that PDU is dropped.

Since the pools are checked in sequential order and the PDU assigned based on the first fit, the FPRx should be configured to use buffer pools with monotonically increasing sized buffers. For example, FPRx pool 0-3 might be assigned to buffer pools with buffers of size 64B, 256KB, 2KB, and 64KB respectively.

The BMU does not support a buffer size of 128Bytes.

If the length of the PDU *cannot* be extracted from the first segment, the FPRx can be configured to use a default PDU length. If this default length is used, all PDUs are assumed to be this length for purposes of buffer selection. As PDU payload is received, it is stored in a buffer until a last segment for the PDU is received.

The entire content of that last segment is stored in the buffer because the FPRx has no way of knowing how much of the segment's payload is valid. The use of default PDU length is configured via the RxFCE_Configuration1 register. Also, the PDU length check must also be disabled if this feature is used. Refer to "RxFCE_Configuration1 Register (FP Rx Configuration Function)" on page 707.

It is required that PDU segments be delivered to the FPRx interface in order. Payload Bytes for a PDU are always stored in a BMU buffer in the order that they are received on the FP interface beginning at offset 0 in the buffer. When writing payload to a BMU buffer, the FP does not set the out-of-band bits of the BMU buffer to meaningful values, so they cannot be used by any transmitting CP.

FPRx Data Memory (DMEM)

The FPRx hardware uses 12KBytes of data memory for storage. It uses 10KBytes for storing payload in 64Byte increments (one for each of 159 concurrent flows), and 2KBytes for storing BTags.

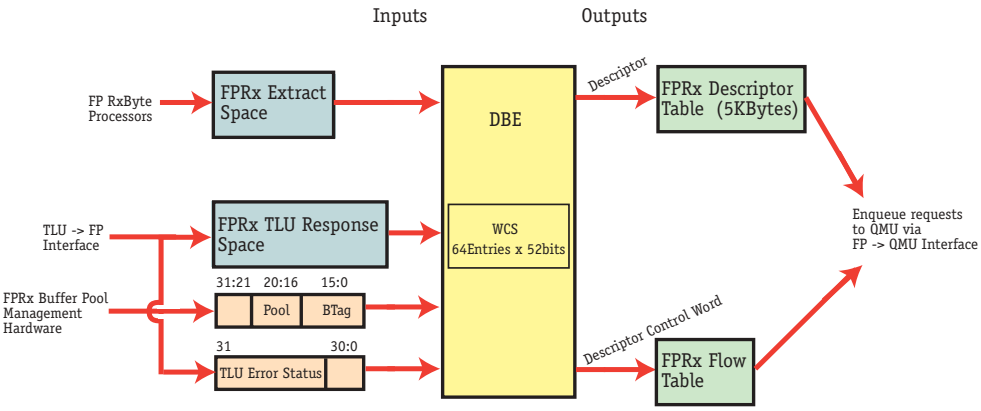
FPRx Building Descriptors

As the FPRx receives PDUs it must build descriptors to be sent to the QMU. This is done using the FPRx Descriptor Build Engine (DBE).

Descriptor Build Engine's (DBE) Microcode Programming to Build the Descriptor

The microcode programmable Descriptor Build Engine (DBE) composes two (2) items: descriptors, and the associated FP -> QMU descriptor control word. Figure 48 on page 213 shows the various inputs and outputs of the DBE.

Figure 49 Descriptor Build Engine (DBE) Inputs and Outputs



The descriptor build sequence is programmed using the Descriptor Build Engine's (DBE) microcode. The microcode control store has room for 64 52bit microinstructions. The instruction set is limited to variations of Move Data (from Source to Destination) with bit manipulation capabilities (mask, shift). Operations may be performed on words, half words or bytes. A literal field can be used for mask operations, and/or to write absolute data of up to 16bits. No jump, branch, or loop control exists, so all descriptors are built with the same straight-line path through the microcode.

The DBE begins constructing a descriptor each time ownership of a first or only segment is passed to hardware. The DBE has the option to use data from four (4) external sources, plus "literals" in the DBE commands. The four (4) external sources include: Extract Space Data, TLU Response Space Data, BTag and Pool Data, and TLU Error Status Data.

To complete the build process, the microcode must write to the Valid bit in the descriptor control word. This write operation signals to the hardware that the descriptor is ready to be enqueued. The last micro-instruction should assert the Restart bit. The descriptors are saved in a 5kBytes descriptor table, that can hold one (1) descriptor for each of 159 concurrent PDUs. When the descriptor has been built, and the entire PDU has been received without an error, the descriptor and its control word are enqueued to the QMU.

FPRx Extract Space Data and TLU Response Space Data

Each of the extract and TLU response areas are divided into multiple datascopes to allow pipelining. The Extract space is either 16Bytes or 32Bytes per datascope depending on the configured descriptor size, and the responses are either 8Bytes, 16Bytes or 32Bytes depending upon the response size configuration.The Extract Space Data is for the current datascope. This is typically header data, which was copied to extract space by a RxByte Processor. The TLU Response Data, (if any) is for the current datascope.

A descriptor build operation begins once the associated Byte Processor has passed extract ownership to hardware (that is, written a 1 to the MSB of the status byte). Additionally, if configured to do so, the DBE waits until the TLU has provided a response for the given datascope. 32Bytes of descriptor table memory are always available for writing by the DBE, however only the amount of the descriptor specified by the size, starting at relative offset “0 “(in the datascope), is transferred to the QMU.

BTag/Pool Data

The FPRx buffer pool management hardware presents the BTag and Pool information for each PDU to the DBE. Typically, the DBE is programmed to read this data information and place it in the descriptor.

TLU Error Handling

If the DBE algorithm uses TLU Response Space Data, and the TLU fails, the FPRx DBE can handle it in two (2) ways as described here:

- If desired, PDUs can be dropped when a TLU lookup fails. To implement this option, the DBE should move the value of the TLU Error Status indication (bit31 of the source data, where source index = 100) over to the Drop bit in the control word. The effect of this instruction is to set the Drop bit when there is a TLU error. This instruction can occur anywhere within the DBE sequence. When the DBE finishes building the descriptor and its control word, the state of the Drop bit determines whether the descriptor gets dropped or enqueued. Refer to Table 48 on page 227 and Table 49 on page 228.

A failed TLU lookup has no valid response data associated with it.

- Enqueue the PDU using a configurable Default Queue Number when the TLU fails. For applications that derive the queue number from TLU response data, this option allows a configurable default queue number to be used when a TLU lookup fails. To enable this option, configure the *RxCFE_Configuration*2 register bits [17:0] *DefaultQNum* field and assert bit [18] *DefaultQEn* field. Refer to "RxFCE_Configuration2 Register (FP Rx Configuration Function)" on page 709.

Descriptor Build Engine's (DBE) Descriptor Control Word

In addition to building the descriptor, the DBE also builds the Descriptor Control Word that gets sent to the QMU. The byte addressing is just like that of the descriptor; bits [31:24] represent Byte0; bits [7:0] represent Byte3. Individual bits can be set using bit mask operations, just as with a descriptor. Setting the Drop bit causes the PDU to be dropped in the FPRx, without being sent to the QMU. To indicate to the hardware that the descriptor and control word are complete, the BDE microcode should assert the Valid bit in the control word.

Any write to the Valid bit causes it to be asserted, regardless of the value (0 or1) of the source data. To write to the control word without setting the Valid bit, simply mask the Valid bit off, or perform an operation that does not update byte 0.

Table 47 Control Word Format

Bit Position	31	30	29 … 9	8 … 0
Field Name	**Drop**	**Valid**	**Rsvd**	**Queue Number**

Table 48 Descriptor Build Engine (DBE) Command Format

Bit Position	51	50	49 … 48	47 … 46	44	43 … 39	38 … 34	33 … 30	29 … 28	16	15 … 0
Field Name	**Restart**	**Op-Code**	**Desc Size**	**Src**		**Src Indx**	**Dest Indx**	**Src Shift**	**Dest**	**Rsvd**	**Literal/Mask**

Table 49 Descriptor Build Engine (DBE) Command Format Fields

FIELD NAME	BIT POSITION	DESCRIPTION
Restart	51	**Restart —** A micro-instruction with Restart=1 causes the micro-instruction pointer to restart the algorithm for the next descriptor. Restart=0 has *no* effect on the descriptor build process. Note: Since the Restart effectively terminates the instruction flow for a given descriptor, this bit should *only* be used on the last micro-instruction.
Op-Code	50:49	**Operation —** Operations to be performed on Descriptor. Legal values are detailed here:

ENCODED VALUE	OPERATION
00	Move/Write from Src -> Dest
01	Move Mask/Read Modify Write • Uses the literal/mask field to determine which bits of the destination data should be updated. This operation can *only* be used with byte and half-word operand sizes. • For each bit in the mask that is a "1", the corresponding bit is updated in the destination data. For each "0", the bit in the in the desalination data retains its previous value. • For byte operations, bits [15:8] of the literal/mask field act as the mask; bits [7:0] can be used as a literal, if desired. • For half-word operations, *all* 16bits of the literal/mask field are used as the mask.
10	NOP — Reserved
11	NOP —Reserved

Table 49 Descriptor Build Engine (DBE) Command Format Fields (continued)

<table>
<tr><td>Operand Size</td><td>48:47</td><td>Operand Size — Legal values are detailed here:

<table><tr><th>ENCODED VALUE</th><th>OPERAND SIZE</th></tr><tr><td>00</td><td>Word</td></tr><tr><td>01</td><td>Byte</td></tr><tr><td>10</td><td>Half Word</td></tr><tr><td>11</td><td>Reserved</td></tr></table></td></tr>
<tr><td>Data</td><td>46:44</td><td>Data — The source of the data. Legal values are detailed here:

<table><tr><th>ENCODED VALUE</th><th>DATA SOURCE</th></tr><tr><td>000</td><td>Extract Space</td></tr><tr><td>001</td><td>TLU Response (The response comes back via the TLU-> FP interface)</td></tr><tr><td>010</td><td>Buffer Memory Information
• Source data= {11’b0, PoolID[4:0], BTag[15:0]}</td></tr><tr><td>011</td><td>Literal
• The location of the literal data within the 32bit source data is dependent on the operand size.
Note: Literals can only be used for byte and half-word operations.
• For Byte operations, source data= {Literal[7:0], 0x000000}
• For Half-word operations, source data= {Literal[15:0], 0x0000}</td></tr><tr><td>100</td><td>TLU Error indication
• When this source is selected, bit31 of the source data indicates whether the TLU lookup encountered an error (1) or not (0). This bit can be written to bit31 of the control word, which is the Drop PDU bit.</td></tr><tr><td>101</td><td>Reserved</td></tr><tr><td>110</td><td>Reserved</td></tr><tr><td>111</td><td>Reserved</td></tr></table></td></tr>
<tr><td>Src Indx</td><td>43:39</td><td>Source Index — The byte offset into source (e.g. extract space offset, buffer offset).</td></tr>
</table>

Table 49 Descriptor Build Engine (DBE) Command Format Fields (continued)

<table>
<tr><td>Dest Indx</td><td>38:34</td><td>Destination Index — The byte offset into the destination descriptor data. Bits [38:36] select which word of the descriptor to update. When the destination is the control word instead of the descriptor then, bits [38:36] are a don't care. In either case, bits [35:34] select the offset within the 4Byte word.</td></tr>
<tr><td>Src Shift</td><td>33:30</td><td>Source Shift — Bit [33] of this field, indicates the direction of the shift. 1= indicates shift left, 0= indicates shift right.
Bits [32:30] of this field, indicates the number of bits to shift. Legal range is 0 to 7.
Note: This is a shift and not a rotate operation; bits do not wrap. For Shift Right, 0's are shifted into the MSB. Likewise for Shift Left, 0's are shifted into the LSB. Also, shift operations occur prior to mask operations. The bits which get shifted outside of the operand source field get dropped. The shift operation has no effect on which destination bits get updated (the destination data is a function of the operand size, the destination index, and optionally a bit mask).</td></tr>
<tr><td>Dest</td><td>29</td><td>Destination — This field determines whether the command's output should be written to the descriptor (0) or the control word (1):
<table><tr><th>ENCODED VALUE</th><th>DESTINATION</th></tr><tr><td>0</td><td>Descriptor</td></tr><tr><td>1</td><td>Control Word</td></tr></table></td></tr>
<tr><td>Reserved</td><td>28:16</td><td>Reserved</td></tr>
<tr><td>Literal/ Mask</td><td>15:0</td><td>Literal/Mask Field — This field can be used as either: a source data (when Source=011), or as a mask for a Move Mask operation.
Note: In either case, this field can only be used with byte or Half-word operand sizes.
• When used as literal source data, the location of the literal data within the 32bit source data is dependent on the operand size.
For byte operations: source data = {Literal[7:0], 0x000000}
For Half-word operations: source data = {Literal[15:0], 0x0000}
• When used as a mask, this field determines which of the bits in the destination data should be updated and which should retain their previous value. Only destination bits which have the corresponding mask bit set to "1"are updated.
For Move Mask operations with a size of 1Byte, bits [15:8] of this field act as the mask. Thus, bits [7:0] can be used as a literal in this case.
For Move Mask operations with a size of a Half-word, bits [15:0] of this field act as the mask.</td></tr>
</table>

Alignment
The source and destination indices *must* be consistent with the operand size. Refer to Table 50 on page 231.

Table 50 Source and Destination Alignments based on Operation

OPERAND SIZE	ALLOWABLE OFFSETS (BITS [1:0] OF THE SOURCE & DESTINATION INDICES)
Word	00
Half-word	00, 10
Byte	00, 01, 10, 11

For example, assume a write operation is performed with source data = AABBCCDDh, using the alignment rules in Table 50 on page 231. Refer to Table 51 on page 232 to see what the destination data would look like after the write operation based on the alignment and operand size. Certain destination bytes are updated, and others retain their previous value.

Table 51 DBE Operand Alignment Examples

OPERAND SIZE	SRC IDX[1:0]	DST IDX[1:0]	DESTINATION DATA (HEX)1
Byte	00	00	AA------
		01	--AA----
		10	----AA--
		11	------AA
	01	00	BB------
		01	--BB----
		10	----BB--
		11	------BB
	10	00	CC------
		01	--CC----
		10	----CC--
		11	------CC
	11	00	DD------
		01	--DD----
		10	----DD--
		11	------DD
Half Word	00	00	AABB----
		10	----AABB
	10	00	CCDD----
		10	----CCDD
Word	00	00	AABBCCDD

1 A dash in the Destination Data field means no change to existing destination data.

FPRx Enqueuing PDUs

After the last, or only segment of a PDU is received and processed by a RxByte Processors and a Descriptor built by the DBE, the FPRx enqueues the descriptor. The FPRx only does unicast enqueues; it *never* does multicast enqueues. However, PDUs that need to be multi-casted can be enqueued to the XP for further processing. The QMU assumes a descriptor weight of one (1) for all enqueues from the FPRx.

There are three (3) FPRx enqueue configuration options and considerations: TLU Error Handling, Enqueue Race Condition Handling, and Failed Enqueued Operation Handling.

The FPRx does not enqueue a descriptor if the DBE indicates that the PDU should be dropped or, if some other error has been detected with the PDU in such a way that the PDU will be dropped.

TLU Error Handling

The FPRx can be configured to enqueue descriptors to a default queue if a TLU error occurs. This default queue is used instead of the queue indicated by the DBE. This is done by setting up the default queue number and default queue enable fields in the Rx*FCE_Configuration2* register. Refer to "RxFCE_Configuration2 Register (FP Rx Configuration Function)" on page 709, as well as, "TLU Error Handling" on page 226.

Enqueue Race Condition Handling

Normally, a descriptor is enqueued to the QMU as soon as the PDU has been fully received by the FPRx, even though the final Payload writes to the BMU may *not* have completed. Though difficult, an application could theoretically create a race condition by dequeuing the PDU and reading its Payload before the Payload has been written. Such theoretical applications should avoid the race by simply asserting the *RxFCE_Configuration2* register bit [20] *EnqWaitWrCB* field which stalls the enqueue operations until the relevant Payload writes are complete. Typical applications need not add this extra stall latency. Refer to"RxFCE_Configuration2 Register (FP Rx Configuration Function)" on page 709.

Failed Enqueue Operation Handling

When the FPRx sends an enqueue request, the QMU may sometimes respond with a failure indication, due to a full queue. The FPRx handles these failed enqueues in two (2) ways: either drop the PDU, or retry the enqueue request.

- The default is to simply drop the PDU and increment the Enqueue Error Counter. When the PDU is dropped, the BTag is deallocated back to the BMU.
- The FPRx can be configured to retry the enqueue, using the *RxFCE_Configuration2* register bit [22] *RetryEnq* field. In this mode, the enqueue operation continues to be tried indefinitely until it finally succeeds. While an enqueue is being retried, the FP can continue to make dequeue requests on the FP to QMU interface; however, no other enqueue operation can start. If the QMU remains full for an extended period, the FPRx may become congested, after which it applies backpressure to the external Fabric interface so that no segments are lost. Refer to "RxFCE_Configuration2 Register (FP Rx Configuration Function)" on page 709.

Congestion Handling

The FPRx may encounter three (3) types of congestion that include: FPRx Payload FIFO, FPRx Header FIFO, and FPRx Scope. Each of these congestion conditions can create backpressure.

FPRx Payload FIFO Backpressure

The FPRx payload FIFO can hold up to 512Bytes of payload from segments. When the number of bytes available in the payload FIFO becomes lower than a configurable XOFF threshold, the FPRx requests link-level flow control. As the payload FIFO then drains and the number of bytes available reaches a separate, configurable XON threshold, the FPRx allows transmission to continue. Refer to "RxDS_Configuration Register (FP Rx Configuration Function)" on page 701.

FPRx Header FIFO Backpressure

The two (2) FPRx header FIFO's have a capacity of 64Bytes (16words) each. When a header FIFO fills up beyond a configurable threshold, the FPRx applies link-level flow control to the fabric. Since there is just a single threshold value (no hysteresis), the FPRx may cross the threshold multiple times in quick succession, sending multiple link-level pause and resume commands to the fabric.

Those applications with large headers and a long flow control response time may overwhelm the backpressure mechanism and cause header FIFO overflows. Therefore, the microcode for such applications *must* check the header FIFO overflow test condition and cleanly drop the offending segments. The header FIFO threshold that applies to both FIFOs is configured using the *RxDS_Configuration* register bits [26:23] *HdrFIFOXOFF* field. Refer to "RxDS_Configuration Register (FP Rx Configuration Function)" on page 701.

FPRx Scope Backpressure

In addition to payload and header FIFO congestion, a third type of congestion can occur: lack of FPRx scopes. As segments are received, each is assigned to a particular scope (set of hardware resources). The number of scopes available is dependent on the descriptor size. For 32Byte descriptors there are eight (8) scopes, meaning the FPRx can actively process eight (8) segments at a time. When all eight (8) scopes are in use, subsequent segments are stalled because the RxByte Processor's microcode must wait for a scope to become available. Eventually the payload or header FIFO gets congested and applies backpressure to the fabric.

Some applications may benefit from detecting the congestion earlier than at the payload or header FIFOs, using the XOFF-No-Scope mechanism. When this mechanism is enabled, link-level flow control asserts to the fabric as soon as a RxByte Processor encounters a "no scope available" condition, and deasserts as soon as a scope becomes available, which could happen immediately. Scopes are most likely to become a bottleneck with applications using variable-length segments, since a string of short segments can be received quickly. The XOFF-No-Scope mechanism is configured using the *RxFCE_Configuration2* register bit [21] *XOFFNoScopeEn* field. Refer to "RxFCE_Configuration2 Register (FP Rx Configuration Function)" on page 709.

FPRx Interrupts

Six (6) events in the FPRx are logged in the *RxFP_Interrupt_Event* register. Each bit within this register remains asserted until written with a '1' ("write 1 to clear"), for instance by an interrupt handler running on the XP. The *RxFP_Interrupt_Enable_Mask* register determines which events cause an interrupt are sent to the XP. Refer to Table 52 on page 236.

Table 52 FPRx Interrupts

ITEM	DESCRIPTION
Parity Error	A parity error was detected on the receive interface.
No BTag Available on Allocate	A BTag allocation request failed because none were available from the BMU.
Buffer Write Errors	The only condition which can cause a buffer write error would be the BMU being unable to satisfy repeated buffer write attempts because it is too busy.
BTag Programming Error	Non-existent memory error - this would only occur if the BMU were misconfigured to be storing BTags in a non-existent memory location.
BTag ECC Error	ECC error - BMU detects an ECC error when it reads a block of BTags out of SDRAM.
BTag Allocation Retry Time-out	Retry time-out - BMU is unable to satisfy the allocate request because it is too busy.

FPTx and FPRx General Considerations

There are five (5) functions that span both the FPRx and FPTx and need to be considered based upon individual application requirements. They include: FP Flow Control (Link-Level and Per-Queue), FP Descriptor Size, FP CRC, FP Endianness, and FP Payload Bus Bandwidth. In the discussions that follow, the "direction" of the flow control is defined as the direction of the request, *not* the direction of the traffic.

Link-Level Flow Control

Link-Level flow control is available for both: Fabric to C-5e NP direction, and C-5e NP to Fabric direction.

Fabric to C-5e NP Link-Level Flow Control

Fabric to C-5e NP link-level flow control occurs when the fabric asks the FPTx to stop transmission for the entire link; that is, all queues. In UTOPIA modes, this is done using the UTOPIA protocol flow control signaling. For non-UTOPIA modes, the fabric makes a flow control request to the FPRx which gets passed to the FPTx. PowerX (CSIX-L0) fabrics send out-of-band link-level requests using the control pins. CSIX-L1 and PRIZMA fabrics send in-band messages, embedded in the segment header.

C-5e NP to Fabric Link-Level Flow Control

The C-5e NP applies link-level flow control to the fabric when the FPRx becomes congested. The flow control request is either made using the UTOPIA protocols, or (in non-UTOPIA modes) the FPRx asks the FPTx to transmit a flow control request to the fabric on its behalf. Non-UTOPIA link-level flow control are covered in more detail in the mode-specific sections. Refer to "CSIX-L1 Interface Mode" on page 243, "UTOPIA Interface Modes" on page 250, "PRIZMA Interface Mode" on page 258, "PowerX(CSIX-L0) Interface Mode" on page 263, and "UTOPIA3 Like to M-5 Interface Mode" on page 267.

Latency Considerations of Flow Control

Flow control operations have an inherent latency due to pipelining of data feeding the FPRx header and payload FIFOs, delays in issuing a pause request to the fabric, and latencies in a fabric's ability to pause. The XOFF thresholds should be set high enough so that even with such latency, the incoming data does not overrun the payload or header FIFOs. Refer to "RxDS_Configuration Register (FP Rx Configuration Function)" on page 701.

There is some latency between the time that a FIFO threshold is hit and the time a pause is requested of the fabric. For example, with the various UTOPIA protocols, the flow enable signal can *only* be deasserted at certain times during cell transmission. If the XOFF condition is reached after that time has passed, the FPRx must be capable of receiving another full cell after the one currently arriving.

The XOFF thresholds must be set with this in mind. Other protocols which request link-level flow control via an FPTx transmission have analogous latencies that must be considered when setting the thresholds.

The Payload FIFO XON threshold should always be set to a value greater than the XOFF threshold.Typically it is set to a value high enough that the XOFF condition won't immediately recur.

Per-Queue Flow Control

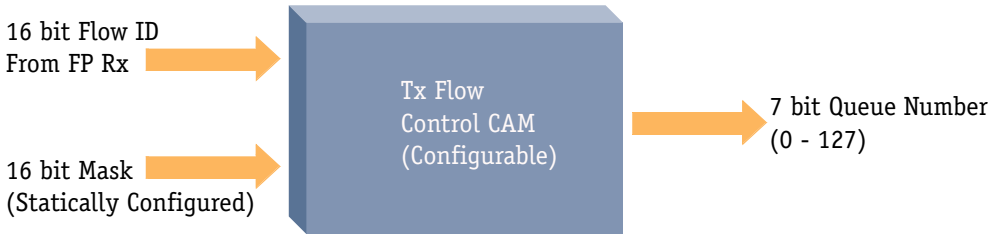
Per-Queue flow control is *only* available for the Fabric to C-5e NP direction. The C-5e NP does *not* apply per-queue flow control to the fabric. It relies only on link-level flow control in that direction.

Fabric to C-5e NP Per-Queue Flow Control

A fabric can request that the FPTx pause or resume transmission from a particular queue.

The flow control request is handled by the RxByte Processor, or in the case of CSIX-L1, dedicated hardware. In either case, the FPRx passes a 16bit flow ID and a pause/resume bit to the FPTx (this could be different from a PDU ID). This flow ID is qualified with a 16bit mask (refer to "TxFCE_Configuration Register (FP Tx Configuration Function)" on page 679) and then used for a lookup in the TxFlow Control CAM (content addressable memory). The output of the CAM is a 7bit queue number which corresponds to one of the 128 FPTx queues; this is how the FPTx knows which queue to pause or resume. Refer to Figure 50 on page 239.

Figure 50 Mapping Per-Queue Flow Control Requests to FPTx Queues



If the lookup matches a CAM entry, the corresponding queue is paused or resumed. If the lookup does *not* match any CAM entry, no queue is paused or resumed. The default mapping preloaded into the CAM by hardware is simply a one-to-one mapping, such that match values of 0 to 127 correspond to queue numbers 0 to 127. The mask and CAM are completely configurable to any mapping scheme. To enable flow control, assert the *TxFCE_Configuration* register bit [19] *FCEnable* field. Refer to "TxFCE_Configuration Register (FP Tx Configuration Function)" on page 679.

Except for CSIX-L1 mode, per-queue flow control requests are processed by the RxByte Processors. To process these, the RxByte Processor must write 2Bytes to form a 16bit flow ID and then write to the pause/resume register. If one of the flow ID Bytes is *not* being used, (that is, it is masked off), then the RxByte Processor need not write to that byte. It is the act of writing to the pause/resume register which sends the request to the FPTx, so that *must* be done after the flow ID has been set up.

Per-queue flow control requests are buffered into a 4-deep FIFO for each RxByte Processor. A single, 16-entry FIFO in the FPTx is fed alternately from the two (2) RxByte Processor's FIFOs. The FPTx can drain a flow control request from its FIFO every core clock cycle.

If a queue is paused while actively transmitting, no further segments are transmitted for that PDU except segments already in the output FIFOs. When that queue is resumed, the FPTx immediately restarts the PDU from the point in which it left off.

If a queue is paused while *not* actively transmitting, the FPTx will not begin transmission from that queue even if it is non-empty. In this situation, once a resume occurs, the FPTx transmits from that queue when that queue's normal turn comes about.

In non-CSIX-L1 modes. Because the FPTx FIFO is fed alternately from the two (2) RxByte Processor's FIFOs, there is no inherent ordering between flow control requests from the two (2) RxByte Processors. This could be accomplished through microcode synchronization if required.

There is no harm in sending a pause request for a queue that is already paused or a resume request for a queue that has already been resumed. This has no effect on the FPTx's per-queue flow control.

TxFlow CAM Configuration Procedure

To configure the CAM, follow this procedure:

1. Set the *TxFCE_Configuration* register bit [15:0] *FlowMask* field to 0xffff.
2. Then delete the preloaded CAM entries by performing 128 writes to the *TxFlowCAM* register; on each write set the DEL (delete) bit and the next 16bit Match value.

   Remember that the preloaded match values are 0 through 127.
3. Next, write the desired entries into the CAM by writing to the *TxFlowCAM* register, with bit [26] *WT* field asserted. The 16bit match value and 8bit write data will thus be placed into the CAM.

Bit7 of the write data must be zero; bits [6:0] represent the 7bit queue number.

FP Descriptor Size

The FP supports descriptor sizes of 12Bytes, 16Bytes, 24Bytes, and 32Bytes. The descriptor size is configured using the *TxFCE_Configuration* register bits [25:24] *DescSize* field on the FPTx and the *RxFCE_Configuration0* register bits [25:24] *DescSize* field on the FPRx. The encoded values for these two (2) fields are different; however, the important point is that the descriptors sizes must be the same for both the FPRx and FPTx. Refer to "TxFCE_Configuration Register (FP Tx Configuration Function)" on page 679 and "RxFCE_Configuration0 Register (FP Rx Configuration Function)" on page 706.

FP CRC

A 32bit CRC can optionally be generated and included in each segment. The region of the segment for which CRC is calculated can be specified using the *TxFI_CRC* register bits [23:16] *FirstIndex* field and bits [15:8] *LastIndex* field; and *RxF_CRC* register bits [23:16] *FirstIndex* field and bits [15:8] *LastIndex* field. The start of the region (*FirstIndex*) is configurable, but must be a multiple of 4. The end of the region (*LastIndex*) must extend to end of the segment; the CRC value resides in the last 4Bytes of the segment.

The values in these 2 fields will *not* be exactly equal. With CRC checking enabled, the CRC of each segment is checked for all segments that are received, and failing segments are dropped. Refer to "TxFI_CRC Register (FP Tx Configuration)" on page 678, and "RxFI_CRC Register (FP Rx Configuration Function)" on page 703.

The inclusion of CRC in each segment decreases the amount of payload included in each segment.

The CRC cannot be used by the FPRx in conjunction with variable length cells.

FP Endianness (Byte and Bit Ordering)

The FP can handle either Big Endian or Little Endian byte ordering. This is configurable using the *TxFI_Configuration* register bit [4] *BigEnd* field and *RxFI_Configuration* register bit [3] *BigEnd* field. Refer to "TxFI_Configuration Register (FP Tx Configuration Function)" on page 673, and "RxFI_Configuration Register (FP Rx Configuration Function)" on page 696.

Refer toTable 53 on page 241 and Table 54 on page 241. The Byte Processors in both the FPTx and FPRx processes the most significant byte first. Bit ordering is always fixed at [7:0] within a byte.

Table 53 Big Endian Byte Ordering on Data Pins 31:0

Most Significant Byte			Least Significant Byte
31:24	23:16	15:8	7:0

Table 54 Little Endian Byte Ordering on Data Pins 31:0

Least Significant Byte			Most Significant Byte
31:24	23:16	15:8	7:0

Byte Order Requirements per Fabric Interface Mode

Specific byte order is required per specific Fabric Interface Modes as listed in Table 55 on page 242.

Table 55 Byte Order Requirements per Fabric Interface Mode

FABRIC INTERFACE MODE	BYTE ORDER REQUIRED
CSIX-L1	Big Endian
PowerX (CSIX-L0)	Little Endian
PRIZMA	Big Endian
UTOPIA	Either Big or Little Endian
UTOPIA3 Like to M-5	Big Endian

FP Payload Bus Bandwidth

Depending on the bandwidth needed by an FP application, the Payload Bus should be configured to allocate a higher proportion of its bandwidth to the FP. This is done using the *XP Miscellaneous Control* register bit [9] *ZBFP* field. Refer to "XP Miscellaneous Control Register (XP Configuration Function)" on page 611.

Fabric Interface Modes and Configurations

The FP supports seven (7) different fabric interfaces that include:

- CSIX-L1
- UTOPIA3, UTOPIA2, and UTOPIA1
- PRIZMA
- PowerX (CSIX-L0)
- UTOPIA3 like to M-5

Details pertaining to each of the seven (7) fabric interfaces are provided in their applicable sections. Topics include: supported and unsupported protocol items, unique protocol items, configuration settings, and pin mapping.

Each data bus can be configured for widths of 8 (data bits 7:0 are used) , 16 (bits 15:0), or 32 (bits 31:0). In 8bit mode, data bits 31:8 are unused. In 16bit mode, data bits 31:16 are unused.

For NP-to-NP operations (Back-to-back) where two (2) C-5e's can be directly connected via the Fabric Port (FP), the NPs should be configured in UTOPIA3 mode, with the FPTx's configured as an ATM device and the FPRx's configured to be PHY devices.

CSIX-L1 Interface Mode

The FP can be configured to operate using the Common Switch Interface level 1 (CSIX-L1) protocol. CSIX-L1 is an industry standard interface between a network processor and a switching fabric as specified by the Network Processing Forum, formerly CSIX/CPIX. For specific CSIX-L1 specifications, refer to *The Network Processing Forum's CSIX-L1 Specification* located on their web site: http://www.npforum.org

Table 56 on page 243 lists the supported items, Table 57 on page 245 lists Freescale optional extensions to CSIX-L1, and Table 58 on page 246 lists the unsupported items.

Table 56 CSIX-L1 Supported Items and Descriptions

ITEM	DESCRIPTION
CSIX-L1 Compliant protocol	C-5e supports.
LVCMOS I/O Buffer support for either 2.5V or 3.3V.	C-5e supports.
32bit Data bus	C-5e supports.
125MHz Fabric clock	C-5e supports.

Table 56 CSIX-L1 Supported Items and Descriptions (continued)

ITEM	DESCRIPTION
Maximum frame size up to 180Bytes	C-5e supports.
Variable -size frames	In general, there is no limitation on how small the frames can be for typical applications. For example, the FPRx can handle a 12Byte data frame or an 8Byte flow control frame. If the FPRx receives a string of short frames and the microcode can not keep up, the header FIFO back-pressure mechanism kicks in and the fabric is flow controlled without any overflow or lost frames. The *only* limitation regarding minimum frame size occurs with applications with large headers (for example, 32Byte descriptors). In this case, with large headers and short frames, the header FIFO could overflow before the header FIFO back-pressure mechanism has a chance to pause the fabric. This happens because the CSIX-L1 link-level flow control is in-band and has a long latency. Applications with large headers can prevent header FIFO overflow by enforcing a minimum frame size in the FPTx. This allows all data frames to have enough padding which permits the FPRx microcode to keep up. A drawback is that the extra overhead of pad bytes are carried throughout the system.
Horizontal parity	C-5e supports.
Frame types: Transmission	Idle, unicast, multicast and broadcast.
Frame types: Reception	Idle, unicast, multicast, broadcast and flow control. For other types, see below.
Fabric-Generated Command & Status/CSIX Reserved Frames	Although these frames are not fully defined in the CSIX-L1 specification, the C-5e provides some level of support for receiving these on egress. They are treated just like a data frame. The FPRx microcode can examine the frame type and enqueue the messages to the XP for further handling.
Link-level flow control of the “data queue” in both directions (C-5e -> fabric and fabric -> C-5e)	The FP pauses the fabric when the FPRx runs out of a resource (payload FIFO, header FIFO, scopes).

Table 56 CSIX-L1 Supported Items and Descriptions (continued)

ITEM	DESCRIPTION
Ready bit for "control queue" is ignored	• For fabric -> C-5e direction, the control ready is irrelevant because C-5e does not transmit any flow control frames. For the initialization sequence, C-5e only looks at the incoming data ready. The CSIX-L1 specification indicates the control ready and data ready bits need to be asserted at the same time during initialization. Therefore, there is no need to look at the incoming control ready. • For C-5e -> fabric direction, the C-5e always asserts control ready (after the initialization sequence). Thus, C-5e is always ready to receive flow control frames.
Fabric flow control (per-queue)	The incoming 4bit Speed Variable is decoded as: • 0000= pause • Everything else= resume
Fabric flow control with port and class wildcarding	Handled by hardware.
Fabric flow control messages of any type	C-5e supports.

Table 57 Freescale Optional Extensions to CSIX-L1

ITEM	DESCRIPTION
Number of programmable dead cycles	Although the CSIX-L1 specifies a single dead cycle, the C-5e can handle anywhere from 0 to 8 dead cycles on egress (FPRx).
CSIX Turbo Mode	CSIX Turbo mode is an extension to the CSIX-L1 frame format, which allows 4Bytes of overhead to be eliminated per frame. Normally, the FPTx appends a dummy word to each CSIX data frame to act as a placeholder for the Vertical Parity field. In Turbo mode, there is *no* Vertical Parity field and the last word of each frame can be used for valid payload. To implement Turbo mode, CRC *must* be disabled in the FPTx, and the FPTx microcode must set the lowest 2bits of the payload length to 00 or 11. For a fabric to support this mode, it *must* forward the entire frame from the ingress C-5e to the egress C-5e without modifying the field normally used for Vertical Parity.

Table 58 CSIX-L1 Unsupported Items and Descriptions

ITEM	DESCRIPTION
CSIX RxClk	The C-5e does not generate RxClk. An external device must supply RxClk to the C-5e (RxClk connects to the FPTx) as well as to the CSIX fabric.
Separate wildcarding of unicast vs. multicast vs. broadcast	All wildcarding is applied across the C-5e's 128 queues, regardless of type. Essentially the type is assumed to be "All" for wildcards.
Vertical parity checking and generation	C-5e does not supports.
Pad Bytes =0	C-5e sometimes transmits pad bytes (extra Bytes between the end of the Payload and the beginning of the Vertical Parity field) whose value is *not* =0. The CSIX-L1 specification indicates that these Bytes should be=0 for vertical parity checking. C-5e does *not* support vertical parity checking.
C-5e-Generated Command & Status /CSIX Reserved frames	C-5e does *not* generate frames of these types.
"Commence processing" of new frame upon Unexpected SOF	Upon detecting an Unexpected SOF, the FP smoothly drops the new frame as well as the previous one.
Ready bit for "control queue" is ignored	• For fabric -> C-5e direction, control ready is irrelevant because the C-5e does not transmit any flow control frames. For initialization sequence, C-5e only looks at the incoming data ready. The CSIX-L1 specifications indicate the control ready and data ready bits need to be asserted at the same time during initialization. Therefore, there is no need to look at the incoming control ready. • For C-5e -> fabric direction, C-5e always asserts control ready (after the initialization sequence). Thus, C-5e is always ready to receive flow control frames.
64bit, 96bit and 128bit data buses	C-5e does not supports.
HSTL I/O buffers	C-5e does not supports.
Fabric clock frequencies up to 250MHz	C-5e does not supports.
Interoperate with different network processors	C-5e does *not* support interoperability with different network processors.

CSIX-L1 Flow Control

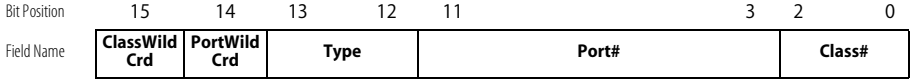
The C-5e can handle any combination of CSIX-L1 flow control messages including: back-to-back flow control frames, and multiple messages per frame (limited only by the configured maximum payload size).

If a horizontal parity error is detected on a 4Byte word within a flow control frame, the flow control messages which straddle that word are ignored, as are all subsequent messages in that frame.

The FPRx extracts key bits from incoming 32bit flow control messages and sends them to the FPTx flow control CAM, using the format in Table 59 on page 247. The CAM and its mask value are highly configurable. Refer to "FPTx and FPRx General Considerations" on page 237.

The FPRx indicates a pause or resume to the FPTx based upon the CSIX-L1 4bit *Speed Variable*. A value of 0000 is treated as a pause; all others are treated as a resume. Based on how the FPTx CAM is configured, flow control messages with unsupported types or non-existent queues simply miss the CAM and are ignored. Messages that hit in the CAM are mapped to any of 128 queues. The CAM actually has 160 entries; the extra 32 entries can be used to provide some added flexibility.

Table 59 FPRx to FPTx Flow Control Format for CSIX-L1



FIELD NAME	BIT POSITION	DESCRIPTION
ClassWildCrd	15	**Class Wildcard —** Refer to *The Network Processing Forum's CSIX-L1 Specification*.
PortWildCrd	14	**Port Wildcard —** Refer to *The Network Processing Forum's CSIX-L1 Specification*.

<table>
<tr><th>FIELD NAME</th><th>BIT POSITION</th><th>DESCRIPTION</th></tr>
<tr><td>Type</td><td>13:12</td><td>Type — Defines the type of message. Legal types are detailed here:

<table>
<tr><th>ENCODED VALUE</th><th>TYPE</th></tr>
<tr><td>00</td><td>Unicast</td></tr>
<tr><td>01</td><td>Multicast</td></tr>
<tr><td>10</td><td>Broadcast</td></tr>
<tr><td>11</td><td>All</td></tr>
</table>
</td></tr>
<tr><td>Port#</td><td>11:3</td><td>Port Number— Defines the port. The port is extracted from the lower 9bits of the 12bit destination address in the flow control message. Typical applications have 16 or 32 ports; extra bits can be masked off using the FPTx flow control mask.</td></tr>
<tr><td>Class#</td><td>2:0</td><td>Class Number — Defines the class. These bits are taken from the upper bits [7:5] of the class in the base header, as mandated per CSIX-L1. Typical applications have 4 or 8 priorities.</td></tr>
</table>

The C-5e does not support the separate notion of unicast wildcarding vs. multicast wildcarding vs. broadcast wildcarding. Thus, wildcarding applies to all 128 queues regardless of "Type " bits [13:12] in Table 59 *on page 247*.

CSIX-L1 Configuration

The FP can be configured to operate with CSIX-L1 interface as follows. Refer toTable 60 on page 249.

Table 60 CSIX-L1 Configuration Settings

SETTING	DETAILS
Set the *RxFI_Configuration* register bits [30:27] *Interface* field to 0x2.	Refer to "RxFI_Configuration Register (FP Rx Configuration Function)" on page 696.
Set the *TxFI_Configuration* register bit [7] *Interface* field to 1.	Refer to "TxFI_Configuration Register (FP Tx Configuration Function)" on page 673.
Set the *TxFI_Configuration* register bit [31] *CFIEnable* field to 1. before setting the *RxFI_Configuration* register bits [31] *CFIEnable* to 1.	Refer to "TxFI_Configuration Register (FP Tx Configuration Function)" on page 673, and to "RxFI_Configuration Register (FP Rx Configuration Function)" on page 696.
Optionally, variable length frames can be set. Set the *TxFI_Configuration* register bit [23] *VariableCells* field, and deselect the *RxFI_Configuration* register bit [23] *FixedSizeSegment* field.	

CSIX-L1 Pin Mapping

Refer to Table 61 on page 249.

For the CSIX-L1 Mode, VDDF= 2.5V.

Table 61 C-5e NP to Fabric Interface Pin Mapping for CSIX-L1 Mode

FPRX SIGNALS				FPTX SIGNALS			
C-5E NP	I/O	CSIX-L1	NOTE	C-5E NP	I/O	CSIX-L1	NOTE
FRxCTL0	Input	n/a		FTxCTL0	Input	n/a	
FRxCTL1	Input	n/a		FTxCTL1	Input	n/a	
FRxCTL2	Input	TxSOF		FTxCTL2	Output	RxSOF	
FRxCTL3	Input	n/a		FTxCTL3	Input	n/a	
FRxCTL4	Input	n/a		FTxCTL4	Input	n/a	
FRxCTL5	Input	n/a		FTxCTL5	Input	n/a	
FRxCTL6	Input	TxPrty		FTxCTL6	Output	RxPrty	

UTOPIA Interface Modes

The FPTx can be configured to operate using either the UTOPIA1/2 or UTOPIA 3 protocol. For either protocol, it can be programmed to operate as either: the ATM (master) device or the PHY (slave) device. In ATM mode, the interface can operate with an 8bit, 16bit, or 32bit bus width. In PHY mode, it can *only* operate with a 16bit or 32bit bus width. Refer to Table 62 on page 250.

Table 62 Freescale Supported UTOPIA Protocols, Modes and Their Bus Widths

BUS WIDTH (BITS)	UTOPIA1 PHY	UTOPIA1 ATM	UTOPIA2 PHY	UTOPIA2 ATM	UTOPIA3 PHY	UTOPIA3 ATM
32	n/a	n/a	n/a	n/a	Yes	Yes
16	n/a	n/a	Yes	Yes	Yes	Yes
8	No	Yes	No	Yes	No	Yes

UTOPIA Interpretation and C-5e Implementation

The UTOPIA specifications are ambiguous and subject to interpretation. Below is Motorola's interpretation of the specifications, as well as, information about the C-5e NP's implementation of the protocols. In addition, whether and/or how optional UTOPIA specifications are handled by C-5e NP are described. Specific references to the UTOPIA specifications from the ATM Forum Technical committee are noted where applicable. The following UTOPIA documents were used:

- UTOPIA Specification Level 1, Version 2.01, af-phy-0017.000, March 21, 1994
- UTOPIA Level 2, Version 1.0, af-phy-0039.000, June 1995
- UTOPIA 3 Physical Layer Interface, af-phy-0136.000, November, 1999

Differences between the two (2) protocols (UTOPIA3, UTOPIA2) include: supported items, the meaning of "cell transfer period", and asserting and deasserting specifications pertaining to control signals. There are a total of six (6) control signals used for Rx and Tx purposes in both UTOPIA3 and UTOPIA2. Each control signal has unique specifications. Each specification and its implementation is described.

UTOPIA3 Implementation

Table 63 on page 251 lists the supported and unsupported items of the C-5e NP in relation to the UTOPIA3 interface.

Table 63 UTOPIA3 Supported and Unsupported Items

ITEM	DESCRIPTION
Single PHY operation	C-5e supports.
Multi-PHY operation	Unsupported by C-5e directly, but can be supported with the M-5.
8bit, 16bit or 32bit interface	C-5e supports all three
52Byte cells	C-5e supports.
Parity pin	C-5e supports.
Full transfer (cell-level handshaking)	C-5e supports. Once cell transmission starts, the cell is transferred, uninterrupted. (Section 2.2.5)
RxClk and TxClk	Are inputs, thus are never driven by the C-5eNP.

Table 64 on page 252 lists the six (6) control signals for UTOPIA3 along with their specifications and implementation using the following definition of “cell transfer period” and the following definition of “asserted” and “deasserted”:

- The "cell transfer period" referred to in Table 64 on page 252, refers to the consecutive bus cycles, starting with the cycle in which Start of Cell (SOC) is asserted and lasting the number of bus cycles required to transfer the fixed cell size. The first cycle of a cell transfer is the cycle in which SOC is asserted. All cycles during the cell transfer are valid data cycles.
- The terms “asserted" and "deasserted" are used as logical terms, and correspond to different logic values depending on the active state of the signal. For example, RxEnb and TxEnb are active low, therefore asserted is a logic value of 0 and deasserted is a logic value of 1.

Table 64 UTOPIA3 Control Signal Specifications and Implementation

SIGNAL	DESCRIPTION
TxClav	• Asserted by PHY to indicate readiness to accept a cell. • Can change from deasserted to asserted at any time. • Can only change from asserted to deasserted two (2)cycles after the cycle in which TxSOC is asserted (Section 3.2.1). • Once asserted (indicating readiness to accept a cell), it must stay asserted until associated "cell transfer" begins (Section 3.2.1).
TxEnb	• Asserted during "cell transfer." • Can only change from deasserted to asserted if TxClav was asserted two (2) cycles previously (Section 3.1.1). • Can only change from deasserted to asserted with the assertion of TxSOC. • Can only change from asserted to deasserted at end of "cell transfer." • Must be deasserted at end of cell if another cell is not starting immediately after the current one. Note: In UTOPIA3 mode, C-5e NP ignores the TxEnb signal and considers there to be valid cell data on the data lines for consecutive cycles starting with an SOC cycle, until the fixed cell size number of bytes have been received.
TxSOC	• Asserted for one cycle to indicate first cycle of cell. • Can only be asserted for a single cycle. • Can only be asserted when TxEnb is asserted (Section 3.1.1). • Can only be asserted when TxClav was asserted for the two (2) previous cycles. • Cannot be asserted in the middle of "cell transfer."
RxClav	• Asserted by PHY whenever it has a cell available to transfer. • Can only change from asserted to deasserted at the same time RxSOC changes from deasserted to asserted (Section 3.2.2); that is, If the PHY does not have a subsequent cell to transmit, it must indicate so at the beginning of the current cell. • Once asserted, it must stay asserted until the cycle after the next RxSOC assertion (Section 3.2.2). • Can change from deasserted to asserted at any time.

Table 64 UTOPIA3 Control Signal Specifications and Implementation (continued)

SIGNAL	DESCRIPTION
RxEnb	• Asserted "in response" (Section 3.2.2) to `RxClav` assertion to "initiate" (Section 3.1.2) a "cell transfer." • Can only change from deasserted to asserted when RxClav was asserted two (2) cycles before. • Must remain asserted during the "cell transfer," at least until two (2) cycles before end of cell (Section 3.2). • Must be deasserted two (2) cycles before end of cell if: Another cell cannot be received or `RxClav` has been deasserted during the "cell transfer." • Once asserted (indicating readiness to accept a cell), it must stay asserted until an associated "cell transfer" begins.
RxSOC	• Asserted for one (1) cycle to indicate first cycle of cell. • Can only be asserted for a single cycle. • Can only be asserted when RxEnb was asserted for the two (2) previous cycles. • Cannot be asserted in the middle of "cell transfer."

UTOPIA2 Implementation

Table 63 on page 251 lists the supported and unsupported items of the C-5e NP in relation to the UTOPIA2 interface.

Table 65 UTOPIA2 Supported and Unsupported Items

ITEM	DESCRIPTION
Handshaking response time	Handshaking response time is expected to be one (1) cycle, *not* two (2) like UTOPIA3. For example, if RxEnb is deasserted during the middle of a cell transfer for one cycle, the UTOPIA PHY is expected to insert one (1) invalid data cycle on the very next cycle.
Handshaking	C-5e NP *only* supports cell-level handshaking.
Clocks	Clocks are expected to be provided by (outputs from) the ATM device. The C-5e NP does *not* drive any clocks. It requires them to be inputs.
54Byte segments	Unsupported. C-5e segments must be 4Byte aligned.

Freescale recommends that a UTOPIA1 or 2 PHY device tristate the RxSOC, RxData (and we presume RxPrty) lines during cycles following cycles where RxEnb is not asserted. C-5e NP implements this recommendation.

Table 66 on page 254 lists the six (6) control signals for UTOPIA2 along with their specifications and implementation using the following definition of “cell transfer period” and the following definition of “asserted” and “deasserted”:

- The "cell transfer period" referred to in Table 66 on page 254, refers to the consecutive bus cycles, starting with the cycle in which SOC is asserted and ending with the valid bus cycle which transfers the last byte of the fixed cell size. During the cell transfer there can be any number of invalid data cycles as indicated by deassertion of the Enb signal. The first cycle of a cell transfer is the cycle in which SOC is asserted.
- The terms “asserted" and "deasserted" are used as logical terms, and correspond to different logic values depending on the active state of the signal. For example, `RxEnb` and `TxEnb` are active low, therefore asserted is a logic value of 0 and deasserted is a logic value of 1.

Table 66 UTOPIA2 Control Signal Specifications and Implementation

SIGNAL	DESCRIPTION
TxClav	• Asserted by PHY to indicate readiness to accept a cell. • Can change from deasserted to asserted at any time. • Can change from asserted to deasserted any time. • Once asserted (indicating readiness to accept a cell), it must stay asserted until an associated “cell transfer” begins. • Must deassert four (4) cycles before the end of a “cell transfer” to avoid transfer of a subsequent cell. If asserted four (4) cycles before the end of a cell transfer, it must stay asserted at least until a subsequent “cell transfer” begins. Note: As recommended by the UTOPIA specification, C-5e NP keeps `TxClav` asserted through the cell transfer until at least the fourth cycle before the end of the cell.
TxEnb	• Asserted during valid cycles of a “cell transfer.” • Must be asserted with `TxSOC`. • Can be deasserted during a “cell transfer” to indicate invalid data cycles. When TxEnb is deasserted, data on TxData is invalid. • Cannot be asserted when a “cell transfer” is not in progress. Note: The C-5e NP will not deassert TxEnb during a “cell transfer.”

Table 66 UTOPIA2 Control Signal Specifications and Implementation (continued)

SIGNAL	DESCRIPTION
TxSOC	• Asserted for one (1) cycle to indicate first cycle of cell. • Can only be asserted for a single cycle. • Can only be asserted when TxEnb is asserted. • Can only be asserted when TxClav was asserted in the previous cycles. Note: TxSOC cannot be asserted in the middle of "cell transfer."
RxClav	• Asserted by PHY whenever it has a cell available to transfer. • Can change from deasserted to asserted at any time. • Once asserted, it must stay asserted until the cycle after the next "cell transfer" begins. • Must remain asserted throughout a "cell transfer." • Must be asserted to coincide with the cycle following the last cycle of a "cell transfer" to allow back-to-back "cell transfer."
RxEnb	• Asserted in response to RxClav assertion to a "cell transfer." • Must be asserted with RxSOC. • Can only change from deasserted to asserted when RxClav was asserted in the previous cycle. • Must be deasserted one (1) cycle before end of cell if a subsequent cell cannot be received. • Can be deasserted during a "ell transfer" to indicate invalid data cycles. When RxEnb is deasserted, data on RxData in the following cycle is invalid. Note: C-5e NP will not deassert RxEnb during a "cell transfer."
RxSOC	• Asserted for one (1) cycle to indicate first cycle of cell. • Can only be asserted for a single cycle. • Can only be asserted when RxEnb was asserted in the previous cycle. • Cannot be asserted in the middle of "cell transfer." Note: C-5e NP tristates RxSOC in cycles following cycles where RxEnb is deasserted.

UTOPIA Configuration

The FP can be configured to operate with UTOPIA2 or 3 interface as follows. Refer toTable 67 on page 256.

Table 67 UTOPIA Configuration Settings

SETTING	DETAILS
For UTOPIA2, set the *TxFI_Configuration* register bit [24] *U2Mode* field and bit [25] *U2TriEnable* field to 1.	Refer to "TxFI_Configuration Register (FP Tx Configuration Function)" on page 673.
For UTOPIA2 or 3, set the *RxFI_Configuration* register bits [30:27] *Interface* field appropriately (to 0x0=UTOPIA3, to 0x5=UTOPIA1 and 2). Note: The FPTx defaults to UTOPIA3 mode unless one of the UTOPIA 2, PRIZMA, PowerX (CSIX-L0), or CSIX-L1 mode bits is asserted in the *TxFI_Configuration* register.	Refer to "RxFI_Configuration Register (FP Rx Configuration Function)" on page 696.

UTOPIA Pin Mapping

Refer to Table 68 on page 256 for ATM mode, and Table 69 on page 257 for PHY mode.

Table 68 C-5e NP to Fabric Interface Pin Mapping for UTOPIA1, 2, 3 ATM Mode

FPRX SIGNALS				FPTX SIGNALS			
C-5E NP	I/O	UTOPIA	NOTE	C-5E NP	I/O	UTOPIA	NOTE
FRxCTL0	Output	RxEnb1	Pullup or No Connection	FTxCTL0	Output	TxEnb[1]	Pullup or No Connection
FRxCTL1	Input	RxClav		FTxCTL1	Input	TxClav	
FRxCTL2	Input	RxSOC		FTxCTL2	Output	TxSOC	
FRxCTL3	Input	n/a		FTxCTL3	Input	n/a	
FRxCTL4	Input	n/a		FTxCTL4	Input	n/a	
FRxCTL5	Input	n/a		FTxCTL5	Input	n/a	
FRxCTL6	Input	RxPrty		FTxCTL6	Output	TxPrty	

[1] Both RxEnb and TxEnb are Active Low.

Table 69 C-5e NP to Fabric Interface Pin Mapping for UTOPIA1, 2, 3 PHY Mode

FPRX SIGNALS				FPTX SIGNALS			
C-5E NP	**I/O**	**UTOPIA**	**NOTE**	**C-5E NP**	**I/O**	**UTOPIA**	**NOTE**
FRxCTL0	Input	TxEnb1	Pullup	FTxCTL0	Input	RxEnb[1]	Pullup
FRxCTL1	Output	TxClav	No Connection	FTxCTL1	Output	RxClav	No Connection
FRxCTL2	Input	TxSOC		FTxCTL2	Output	RxSOC	
FRxCTL3	Input	n/a		FTxCTL3	Input	n/a	
FRxCTL4	Input	n/a		FTxCTL4	Input	n/a	
FRxCTL5	Input	n/a		FTxCTL5	Input	n/a	
FRxCTL6	Input	TxPrty		FTxCTL6	Output	RxPrty	

[1] Both TxEnb and RxEnb are Active Low.

PRIZMA Interface Mode

The FP should be configured for PRIZMA mode when interfacing to the UDASL chip, which in turn interfaces to the IBM PRIZMA-E or PRIZMA-EP switch fabric.

In PRIZMA fabric terminology, the segment is called a "packet". Therefore, the term "packet" is used here.

There are five (5) PRIZMA items that need some additional discussion about FP handling. They include: packet sizes, in-band flow control, link-level flow control, idle packets, queue grants, and RxByte Processor's drop mode.

Packet Sizes

The PRIZMA fabric supports packet sizes between 64 and 80Bytes. The C-5e NP supports this range of packet sizes, *but* packet sizes must be a multiple of 4Bytes. The PRIZMA fabric must be configured to place the packet qualifier byte as the first byte of the header. Typically, the destination bitmap (on ingress to the fabric) or queue grants (on egress from the fabric) would be the next Bytes of the header. Because microcode generates the PRIZMA address bitmap and processes, the queue grant bits in the PRIZMA header, the PRIZMA fabric can be configured to have 16 or 32 queues per priority.

In-Band Flow Control

When operating with a PRIZMA fabric, the fabric is configured to use in-band flow control and the UTOPIA protocol Enb and Clav signals are *not* used. Therefore, Enb inputs to the C-5e NP should be pulled down and the Clav inputs to the UDASL chip should be pulled up. The SOC pins of the C-5e NP and the UDASL should be connected.

When using in-band flow control, the FPTx generates PRIZMA-format idle packets whenever it has no segments to transmit or whenever it has been paused by the PRIZMA fabric. Packets (data or idle) are transmitted from the C-5e NP in an absolutely back-to-back fashion on the interface; that is, there will be no unused cycles on the bus. The C-5e NP must be configured to generate idle packets using the *TxFI_Configuration* register to enable idle packet generation and the *TxDM_Header/Payload Delimiter* register to specify the length of idle packets so they are the same length as data packets. The *TxIdleData* register must be programmed to a value of 0xCCCCCCCC so that the content of idle packets is correct.

When idle packets are generated, no microcode generated header is used, so PRIZMA microcode needs no support for idle packet header generation.

Refer to "TxFI_Configuration Register (FP Tx Configuration Function)" on page 673, "TxDM_Header/Payload Delimiter Register (FP Tx Configuration Function)" on page 675, and "TxIdleData Register (FP Tx Configuration Function)" on page 688.

Link-Level Flow Control

If congestion occurs in the FPRx, link-level flow control is sent to the fabric via the FPTx. The FPTx asserts the TxPause bits in the PRIZMA packet qualifier byte of the packets that it transmits. The FPTx always asserts or deasserts all four (4) of the TxPause bits together; it does *not* perform link-level flow control on a per-priority basis.

The C-5e NP presumes that the UDASL chip is configured to use full inband flow control in such a way that, when the UDASL's input FIFO fills and it requires link-level flow control, it is expected to deassert the shared memory grant bit, for at least one priority, in the packet qualifier of packets that it transmits to the C-5e NP. The FPRx extracts PRIZMA shared memory grant information from the packet qualifier byte of each packet; microcode does not have to do this.

If a shared memory grant is lost for any priority, the C-5e NP pauses all data packet transmission and transmits only idle packets to the fabric. Again, link-level flow control is implemented as all-or-nothing, *not* per-priority. After the C-5e NP powers up, it does *not* allow the FPTx to transmit until it has received a PRIZMA packet with a shared memory grant, for each of the enabled priorities.

The number of desired priorities is set using the *RxDS_Configuration* register bits [29:28] *NumPri* field. If the FPRx receives a shared memory grant for a priority that is *not* enabled, that shared memory grant is ignored and has no effect on of link-level flow control.

Idle Packets

The FPRx expects that idle packets are received whenever the fabric has no data packets to transmit or whenever the C-5e NP has paused the fabric. FPRx data splitting registers *must* be configured to appropriately identify and split idle packets. Microcode must recognize and discard idle packets. Refer to "RxByte Processors Discarding Segments" on page 222.

When the PRIZMA fabric is paused, it stops transmitting data packets but continues to transmit idle packets into the C-5e NP. Because some portion of these packets must be directed into the payload FIFO, the payload FIFO may eventually overflow and lose some of this idle packet data. This is *not* a problem because this payload would be discarded when it is popped from the payload FIFO anyway. Refer to "RxByte Processor's Drop Mode" on page 260.

Queue Grants

Queue grants in PRIZMA headers sent into the C-5e NP can be handled by FPRx microcode so they generate per-queue flow control messages to the FPTx. The microcode may only be able to process a subset of the Queue grants for a given packet; this performance limitation is dependent on the number of Queue grants per packet, the packet size, the core clock frequency, and the fabric interface clock frequency. As an example, assume that microcode can only process eight (8) Queue grants per packet, in an application with 128 queues (32 ports x 4 priorities). Therefore, it would take 16 packets before all Queues grants were processed. The processing task is split between the two (2) Byte Processors, with the shared registers used as a scratch-pad area to keep track of which Queue grants to process next. A token passing scheme could be implemented so that the byte processors would *not* collide while accessing the shared registers. Refer to "RxByte Processors Token Passing" on page 223.

RxByte Processor's Drop Mode

When the amount of data in the payload FIFO passes the configurable XOFF threshold, link-level flow control can be applied by the FPRx back to the fabric (refer to "FPRx Payload FIFO Backpressure" on page 234). Despite this, PRIZMA applications continue to transmit idle packets and those packets are forwarded to the header and payload FIFOs for in-band per-queue flow control processing. Therefore, it is possible for the payload FIFO to continue filling up well past its threshold. When it fills up to the maximum of 512Bytes, Drop Mode is activated, causing subsequent payloads to be dropped.

CSIX-L1 also has idle packets, but they require no processing and therefore do not get sent to the header/payload FIFOs. Thus, the Drop Mode only occurs with the PRIZMA mode.

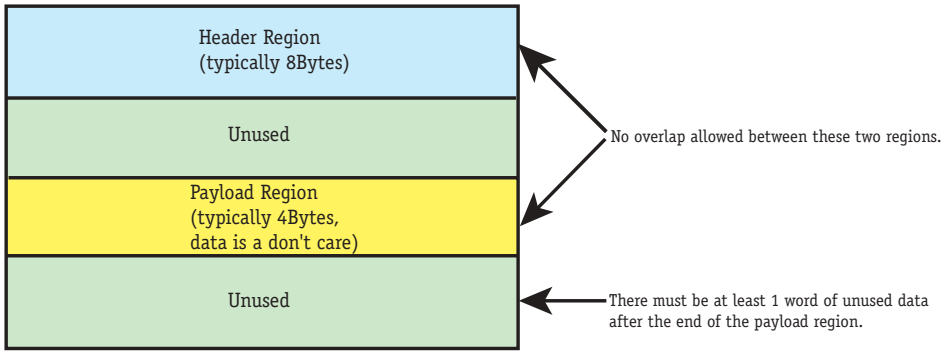
In Drop Mode, the RxByte Processors continue to process incoming headers. However, there is no payload associated with these headers. Therefore, the RxByte Processor's microcode must *not* advance the datascope (which would pass the ownership of the segment to the remainder of the hardware pipeline). Thus, the RxByte Processor's microcode needs to test for Drop Mode via external test condition6 (refer to"External Test Conditions" on page 209), before deciding whether to advance the datascope.

To prevent a race between a Drop Mode assertion and the point at which the microcode samples the Drop Mode external test condition, ensure the idle packet header and payload regions are configured as described here:

- Header and payload regions for idle packets *must not* overlap, and
- Payload region *must end before* the end of the packet.

Refer to Figure 51 on page 261.

Figure 51 Idle Packet Configuration Requirements for FPRx to Prevent a Race Condition



PRIZMA Configuration

The FP can be configured to operate with the PRIZMA interface as follows. The configuration is similar to the 32bit UTOPIA3 PHY operation with the differences listed in Table 70 on page 261.

Table 70 PRIZMA Delta Configuration Settings

SETTING	DETAILS
Set the *TxFI_Configuration* register bit [21] *PRIZMA* field to 1.	Refer to "TxFI_Configuration Register (FP Tx Configuration Function)" on page 673.
Set the *TxFI_Configuration* register bit [18] *IdleCell* field to 1.	
Set the *RxFI_Configuration* register bits [30:27] *Interface* field to 0x4.	Refer to "RxFI_Configuration Register (FP Rx Configuration Function)" on page 696.
Program the *TxIdleData* register bits [31:0] to contain a value of 0xCCCCCCCC.	Refer to "TxIdleData Register (FP Tx Configuration Function)" on page 688.

Table 70 PRIZMA Delta Configuration Settings (continued)

SETTING	DETAILS
Program *TxDM_Header/Payload Delimiter* bits [23:16] *IdleCellLen* field with a value which is 1 less than the number of fabric interface cycles required to transmit a packet.	Ref to "TxDM_Header/Payload Delimiter Register (FP Tx Configuration Function)" on page 675.

While the C-5e NP supports the basic protocol of the PRIZMA fabric, a particular application may not be possible due to limitations of microcode space and cycle time. Consult the PRIZMA application note to understand some of the trade-offs involved in developing a PRIZMA application.

PRIZMA Pin Mapping

Refer to Table 71 on page 262.

Table 71 C-5e NP to Fabric Interface Pin Mapping for PRIZMA Mode

FPRX SIGNALS				FPTX SIGNALS			
C-5E NP	I/O	PRIZMA	NOTE	C-5E NP	I/O	PRIZMA	NOTE
FRxCTL0	Input	TxEnb1	Not connected to fabric.	FTxCTL0	Input	RxEnb[1]	Not connected to fabric.
FRxCTL1	Output	TxClav	No Connection	FTxCTL1	Output	RxClav	No Connection
FRxCTL2	Input	TxSOP		FTxCTL2	Output	RxSOP	
FRxCTL3	Input	n/a		FTxCTL3	Input	n/a	
FRxCTL4	Input	n/a		FTxCTL4	Input	n/a	
FRxCTL5	Input	n/a		FTxCTL5	Input	n/a	
FRxCTL6	Input	TxPrty	Optional	FTxCTL6	Output	RxPrty	Optional

[1] Both TxEnb and RxEnb are Active Low.

PowerX(CSIX-L0) Interface Mode

The FP can be configured to operate with the PowerX (CSIX-L0) TeraChannel® Switch Fabric. Table 72 on page 263 lists both the supported, as well as, unsupported items of the C-5e in relation to the PowerX (CSIX-L0) interface as follows.

Table 72 PowerX (CSIX-L0) Supported and Unsupported Items

ITEM TYPE	SUPPORTED ITEM	UNSUPPORTED ITEM
Interface	32bit PowerX (CSIX-L0)	16bit PowerX (CSIX-L0)
Transmits	Invalid data, start of frame data, valid frame data, pause, and resume PowerX (CSIX-L0) bus cycles	Abort or flow control cycles
Receives	Invalid data, start of frame data, valid frame data, pause, and resume and flow control bus cycles	Abort or reserved bus cycles

PowerX(CSIX-L0) Constraints

The C-5e NP can be programmed to support any of the PowerX (CSIX-L0) frame types, however it may *not* be possible to support all types in a single application given microcode and configuration constraints.

The use of the service channel and urgency fields of the PowerX (CSIX-L0) header is completely a function of the application microcode.

PowerX(CSIX-L0) Requirements

The payload length field of the PowerX (CSIX-L0) header must be generated by FPTx microcode. FPTx hardware makes the current segment length available to microcode for this purpose. FPRx hardware extracts the payload length field of the PowerX (CSIX-L0) header on frames that it receives to determine when the frame has been completely delivered. There is no need for FPRx microcode to process the payload length field.

The FPTx can optionally generate variable-length frames which the PowerX (CSIX-L0) fabric supports. When configured to do so, the FPTx generates fixed, maximum-sized frames for all frames of a PDU except the last one. For the last frame of the PDU, the FPTx transmits the shortest possible frame (which is a multiple of 4Bytes).

For short frames, dead cycles are inserted as necessary to meet the minimum SOF-to-SOF spacing. This is done using the *TxDM_Header/Payload Delimiter* register bits [28:24] *MinSOF-SOFSpacing* field. The PowerX (CSIX-L0) fabric must be guaranteed a minimum number of clock cycles between consecutive SOF cycles.

Setting the *MinSOF-SOFSpacing* filed guarantees a minimum SOF-to-SOF gap by having a minimum size variable frame length. Refer to "TxDM_Header/Payload Delimiter Register (FP Tx Configuration Function)" on page 675.

If variable-length frames are not enabled, the FPTx always transmit maximum-sized frames for all frames of a PDU. The use of variable-length frames can be enabled using the *TxFI_Configuration* register bit [23] *VariableCellSize* field.

The use of variable-length frames is incompatible with the use of CRC in those frames. If CRC is to be used, fixed frames must be used.

As PowerX (CSIX-L0) flow control bus cycles are received by the C-5e NP, the FPRx directs the flow control messages to a control FIFO in one of the Byte Processors. The control FIFOs parallel the header FIFOs in the two (2) Byte Processors and control messages are delivered alternately to each of the Byte Processors. The C-5e NP should be configured to direct *only* the 2 meaningful bytes of PowerX (CSIX-L0) flow control bus cycles to a control FIFO. This is done using the *RxDS_Configuration* register bits [22:20] *CtlWordSize* field. Refer to "RxDS_Configuration Register (FP Rx Configuration Function)" on page 701.

PowerX(CSIX-L0) Byte Processor Unloading

In PowerX (CSIX-L0) mode, when a Byte Processor unloads the header FIFO, it is really unloading the control or header FIFO. Control FIFO contents take precedence over header FIFO contents so that if a Byte Processor does a FIFO unload it gets a control FIFO byte if there are any present. The Byte Processor can test whether the byte it has unloaded came from the control FIFO instead of the header FIFO using the "control word" test condition. If it is true, the byte came from the control FIFO and is part of a flow control message.

For PowerX (CSIX-L0) flow control messages (16bits), just as with headers, the byte Processors process the most significant byte first. Since PowerX (CSIX-L0) uses Little Endian byte ordering, the Byte Processor first sees the byte which was received on pins 7:0, and next sees the byte received on pins 15:8. Also, upon receiving a flow control message, an FPRx Byte Processor can pause or resume the corresponding FPTx queue by writing a flow control message to the FPTx.

PowerX(CSIX-L0) Configuration

The following configuration settings must be implemented for the FP to operate with a PowerX (CSIX-L0) interface. Table 73 on page 265 lists the applicable configuration settings.

Table 73 PowerX (CSIX-L0) Configuration Settings

SETTING	DETAILS
Set the *TxFI_Configuration* register bit [19] *PowerX (CSIX-L0)* field to 1.	Refer to "TxFI_Configuration Register (FP Tx Configuration Function)" on page 673.
If variable size frames are desired, then set the *TxFI_Configuration* register bit [23] *VariableCellSize* to 1, set the *TxFI_CRC* register bit [31] *Enable* field to 0, and set *RxFI_Configuration* register bit [23] *FixedSizeSegments* to 0.	Refer to "TxFI_Configuration Register (FP Tx Configuration Function)" on page 673), and "RxFI_Configuration Register (FP Rx Configuration Function)" on page 696.
Set the *RxFI_Configuration* register bits [30:27] *Interface* field to 0x3 for PowerX (CISX-L0).	Refer to "RxFI_Configuration Register (FP Rx Configuration Function)" on page 696.
Set the *RxFI_Configuration* register bit [26] *ByteParity* field to 1 to select parity on each byte lane.	
Set the *RxFI_Configuration* register bits [22:16] *ParityCtlMask* field to 0000111binary (7dec.) so that control pins 2:0 are included in parity calculations.	
Set both the *RxFI_Configuration* register bit [3] *BigEnd* field and the *TxFI_Configuration* register bit [4] *BigEnd* field to 0 for little endianness.	Refer to "RxFI_Configuration Register (FP Rx Configuration Function)" on page 696, and ("TxFI_Configuration Register (FP Tx Configuration Function)" on page 673.
Set both the *RxFI_Configuration* register bit [0] *RegInput* field and the *TxFI_Configuration* bit [0] *RegInput* field to 1. Note: The *TxFI_Configuration* register bit [0] *RegInput* field is a "don't care" since in the PowerX (CSIX-L0) mode there are no inputs to the FPTx. However, we recommend setting this filed to 1.	
Set the *RxDS_Configuration* register bit [19] *CtlWordDisable* field to 0 for PowerX (CISX-L0) mode.	Refer to "RxDS_Configuration Register (FP Rx Configuration Function)" on page 701.
Set the *RxDS_Configuration* register bits [22:20] *CtlWordSize* field to 2 for PowerX (CISX-L0).	

Table 73 PowerX (CSIX-L0) Configuration Settings (continued)

SETTING	DETAILS
Set the *TxDM_Header/Payload Delimiter* register bits [28:24] *MinSOF-SOFSpacing* field to the minimum SOF to SOF timing.	Refer to "TxDM_Header/Payload Delimiter Register (FP Tx Configuration Function)" on page 675.

While the C-5e NP supports the basic protocol of the PowerX (CSIX-L0) fabric, a particular application may not be possible due to limitations of microcode space and time. You should consult the PowerX (CSIX-L0) Application Note to understand some of the trade-offs involved in developing a PowerX (CSIX-L0) application.

PowerX(CSIX-L0) Pin Mapping

Refer to Table 71 on page 262.

Table 74 C-5e NP to Fabric Interface Pin Mapping for PowerX (CSIX-L0)

FPRX SIGNALS				FPTX SIGNALS			
C-5E NP	I/O	POWERX (CSIX-L0)	NOTE	C-5E NP	I/O	POWERX (CSIX-L0)	NOTE
FRXCTL0	Input	RxCtrl[0]		FTXCTL0	Output	TxCtrl[0]	
FRXCTL1	Input	RxCtrl[1]		FTXCTL1	Output	TxCtrl[1]	
FRXCTL2	Input	RxCtrl[2]		FTXCTL2	Output	TxCtrl[2]	
FRXCTL3	Input	RxPrty[3]		FTXCTL3	Output	TxPrty[3]	
FRXCTL4	Input	RxPrty[2]		FTXCTL4	Output	TxPrty[2]	
FRXCTL5	Input	RxPrty[1]		FTXCTL5	Output	TxPrty[1]	
FRXCTL6	Input	RxPrty[0]		FTXCTL6	Output	TxPrty[0]	

UTOPIA3 Like to M-5 Interface Mode

In general, the configuration for this mode is identical to UTOPIA3 PHY mode with two (2) exceptions noted here:

- Variable length cells *must be* enabled. The FPRx hardware extracts the cell length from the M-5 header.
- The RxClav signal (output of FPTx) is ignored by the M-5. Therefore, the FPTx to M-5 protocol is slightly different than UTOPIA3.

The protocol between the FPRx and M-5 is compliant with UTOPIA3.

Refer to *M-5 Channel Adapter Architecture Guide (part number M5CAARCH-RM/D)* for additional information.

FP Debug and Test

The FP provides four (4) types of debug and test features. These include: FP Debug Mux, FPRx Statistics Registers, FP Internal Debug State Registers, and Debug and Test of Selected FP Internal Memories.

FP Debug Mux

Through use of the configurable Debug MUX, certain FP events can be logged in event counters located in the Executive Processor (XP). For instance, the number of PDUs that are transmitted can be logged. For details about the specific FP events that can be monitored, refer to "RxFP_Debug_Mux_Control Register (FP Rx Debug Function)" on page 714, and "TxFP_Debug_Mux_Control Register (FP Tx Debug Function)" on page 681. For information on how to use the XP event counters, refer to Chapter 3.

FPRx Statistics Registers

The FPRx contains its own statistics registers for logging the number of received segments, PDUs, errors, etc. These statistics are available through nineteen (19) registers. These registers are automatically updated by hardware, and can be initialized to a value of 0 at any time by simply performing a global write to the particular statistics register. The statistics can be read with global reads. Refer to "RxFP_Statistics Registers (FP Rx Statistics Function)" on page 719.

FP Internal Debug State Registers

Some internal FP state points are made visible through two (2) 32bit debug state registers, one (1) in the FPTx and one (1) in the FPRx. These registers can be accessed with global reads, and contain the current status of internal state machines and other key state points. Refer to "TxDebug_Internal_State Register (FP Tx Debug Function)" on page 689 and "RxDebug_Internal_State Register (FP Rx Statistics Function)" on page 722.

Debug and Test of Selected FP Internal Memories

It is possible to write to and read from eight (8) of the internal memories in the FP for debug and test purposes. They include: Rx PDU ID CAM, Rx Flow Table, Descriptor Table, Tx Flow Table, Merge Space, DMEMs, TLU Response Space, FP Read and Write Control Blocks (RdCBs and WrCBs). In addition, the WCSs and most CAMs are loaded by software during initialization. Refer to "Initialization of Selected FP Internal Memories" on page 273.

While accessing the following memories, the FP should be disabled.

Rx PDU ID CAM Access

The Rx PDU ID CAM is updated and accessed by FPRx hardware to keep track of active PDU IDs. For debug and test purposes, it can be read and written using the "RxPDU_ID_CAM Register (FP Rx Debug Function)" on page 718. To write to the CAM, set up the 16-match value and the corresponding 8bit data value, and assert the Write bit. To delete an entry, write the match value to the register and assert the Delete bit.

To read an entry, set up the match value, set the Search bit, and then read out the 8bit result by reading the register.

Rx Flow Table and Descriptor Table Access

The internal memory that the FPRx uses to save the current state of active PDUs, as well as, the memory in which the Descriptor Build Engine (DBE) stores descriptors can be read and written. These two (2) spaces are accessed via the "RxMemory_Data Register (FP Rx Debug Function)" on page 717 and "RxMemory_Address Register (FP Rx Debug Function)" on page 717, using the addresses listed below. To write a location in one of these memories, set up the memory address register and then write the data to the memory data register. The act of writing to the data register triggers the hardware to perform the memory write. To read a location: simply set up the address register, and then collect the result by reading the data register.

Bit [13] of the address register selects between the Descriptor Table (0) and RxFlow Table (1).

- The Descriptor Table is organized as 1280 entries of 32bits. Bits [12:2] in the address register select the entry. Address bits [1:0] are irrelevant because accesses are performed in 32bit quantities.
- The RxFlow Table is organized as 160 entries of 72 bits. Bits [11:4] in the address register select the entry. Bits [3:2] select between fields within the 72bit word. Refer to Table 75 on page 270.

Refer to Figure 52 on page 270 Address Map for Descriptor Table and RxFlow Table.

Figure 52 Address Map for Both Descriptor Table and RxFlow Table Memories for Debug Purposes

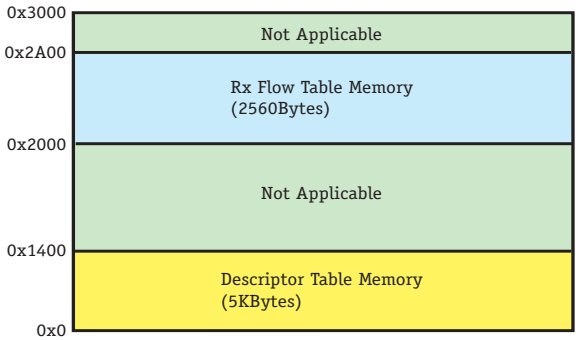


Table 75 RxFlow Table Memory, Field Selection

ADDRESS [3:2]	DATA[31:0]
00	btag[15:0], offset[15:2], 00
01	000, pool[4:0], buffer[7:0], length[15:0]
10	00000000, Drop, Queue Valid, 13bits of don't care, Queue[8:0]
11	Reserved

Tx Flow Table Access

The internal memory that the FPTx uses to save the current state of active PDUs can be read and written. To read an entry, simply write the 7bit index (Tx queue number) to *TxFlowTbl* register, and then read the resulting 60bit data from the high (upper 28 bits) and low (lower 32 bits) data registers. To write an entry, set up the data registers first, then write the 7bit address to *TxFlowTbl* register while asserting bit [16] *WT* field. Refer to the descriptions of "TxFlowTbl Register (FP Tx Debug Function)" on page 684, "TxFlowTbl_Data_Low Register (FP Tx Debug Function)" on page 684, and "TxFlowTbl_Data_High Register (FP Tx Debug Function)" on page 685.

Merge Space Access

Merge Space is an internal memory where the FPTx copies descriptors, to make their content available to the Byte Processors for header construction. This space can be read and written via the "TxMergeAddr Register (FP Tx Debug Function)" on page 687 and "TxMergeData Register (FP Tx Debug Function)" on page 687.

DMEMs Access

The FPTx and FPRx each have a 12KBytes Data Memory (DMEM), for a total of 24KBytes, that can be accessed with global reads and writes, using the global addresses from the "TxByte Processor Memory Map" on page 196 and "RxByte Processor Memory Map" on page 213.

- The FPTx uses 2KBytes for storing payload for its 8 active flows, 8KBytes for up to 128 descriptors, and the remainder for BTags to be deallocated.
- The FPRx uses 10KBytes for storing payload in 64B buffers (one for each of 159 concurrent flows), and 2KBytes for storing BTags.

TLU Response Space Access

The TLU response space in the FPRx (256Bytes) can be globally read for debug purposes only, at the locations specified in the "RxByte Processor Memory Map" on page 213.

FP Read and Write Control Blocks (RdCBs and WrCBs) Access

The control blocks in the FP are controlled by hardware and are *not* software programmable. This is quite different from how control blocks are used in the Channel Processors (CPs). Refer to Table 76 on page 271.

Table 76 Global Access for FP Control Blocks

PURPOSE	CONTROL BLOCK	GLOBAL RD ACCESS	GLOBAL WR ACCESS
For internal manufacturing testing (Only)	FPTx	Allowed	Disallowed
	FPRx	Allowed	Allowed

Refer to "Using Multi-Use Control Blocks to Achieve Different Functions" on page 144 for details regarding specific registers used to compose the control block. In addition, the base global address for the FP control blocks are shown in Figure 44 on page 188 and Figure 47 on page 206.

FP Setup

The FP must be initialized as described here. Using Global Bus operations, various registers and internal memories must be written with appropriate values to allow the FP to function. Initialization must be completed before the FP can be put online.

FP Initialization Steps

The following steps should be performed in sequence to initialize the FP.

1. While keeping the FPRx and FPTx disabled, set up the configuration registers. Load all FP control stores (microcode). Configure and enable all of the resources used by the FP (BMU, QMU, TLU).
2. Enable the FPRx using the *RxFP_Enable* register bit [31] *Enable* field and the FPTx using the *TxFP_Enable* register bit [31] *Enable* field.
3. After the FPRx has had time to acquire BTags for all of its pools, set the *RxFI_Configuration* register bit [31] *CFIEnable* field.
4. Set the *TxFI_Configuration* register bit [31] *CFIEnable* field.

In CSIX-L1 mode, swap step 3 and step 4. This allows the FPTx to cleanly transmit idle frames by the time the FPRx begins the CSIX-L1 initialization sequence.

Once initialized, the FP cannot be dynamically reconfigured and reinitialized. There must be a full C-5e NP reset to reinitialize the FP.

Initialization Options for SDRAM

Another initialization issue has to do with SDRAM. When the FPRx receives payload and writes it into SDRAM, the write is done with 16Byte granularity. This means the PDU payload could end at address 0, 16, 32 or 48 within the last 64Byte block. If the PDU is later transmitted by a CP's SDP, the payload is read in 64Byte blocks, so there may be some uninitialized data in the last block. This uninitialized data can cause an ECC error. There are three (3) ways to handle this case:

- Initialize SDRAM at startup or,
- Disable ECC checking or,
- Configure the FPRx to always write in 64Byte blocks, using the *RxFCE_Configuration2* register bit [10] *Force64ByteWrCBTransfers* field.

Initialization of Selected FP Internal Memories

There are six (6) internal memories that are loaded by software during initialization and are accessible, they include: FPTx Flow Control CAM, TxByte Processor's WCS, TxByte Processor's CAM, RxByte Processor's WCS, RxByte Processor's CAM, and RxDescriptor Build Engine's WCS.

FPTx Flow Control CAM

The *FPTx Flow Control CAM*, which the FPTx uses to map a per-queue flow control request to a queue number, can be read and written, using the "TxFlowCAM Register (FP Tx Debug Function)" on page 685. The default mapping initialized into the CAM by hardware is simply a one-to-one mapping, such that the 7bit queue number equals the 16bit match value, for the range of 0 to 127. The procedure to reconfigure the CAM is done using software. Refer to "Fabric to C-5e NP Per-Queue Flow Control" on page 238.

TxByte Processor's WCSs/CAMs Access

The two (2) TxByte Processors each contain a WCS and a CAM (not to be confused with the FPTx Flow Control CAM). Each of these WCSs and CAMs must be loaded by *Internal Scan Chains* accessible using the *TxWCS_CAM* register. As with the CPs SDPs these memories must be loaded using an internal scan chain accessible in Global Address Space. Refer to "TxWCS_CAM (Tx WCS CAM Function)" on page 683.

The two (2) FPTx Byte Processors are loaded in parallel. That is, the two TxByte Processors run identical microcode.

RxByte Processor's WCSs/CAMs and the RxDescriptor Build Engines's WCS Access

The two (2) RxByte Processors each contain a WCS and a CAM. In addition, the RxDescriptor Build Engine (DBE) contains a WCS. Of these three (3) WCSs and two (2) CAMs, all five (5) may be read using the internal scan access, but *only* the two (2) CAMs may be written using the internal scan access. The three (3) WCSs, may be written to using a *Special Byte Access*. Both the scan access and special byte write access is provided using the "RxWCS_CAM Register (FP RxWCS CAM Function)" on page 704. Refer to Table 77 on page 274.

Table 77 RxByte Processor's WCSs/CAMs and RxDBE's WCS Access

STORE TYPE	DESCRIPTION	SCAN ACCESS RD	SCAN ACCESS WR	SPECIAL BYTE ACCESS (WR ONLY)
RxByte Processor0 WCS	96-word Writable Control Store, 52bits each	Y	N	Y
RxByte Processor1 WCS	96-word Writable Control Store, 52bits each	Y	N	Y
RxDescriptor Build Engine WCS	64 -word Writable Control Store, 52bits each	Y	N	Y
RxByte Processors0 CAM	24-entry Content Addressable Memory, 52bits each	Y	Y	N
RxByte Processors1 CAM	24-entry Content Addressable Memory	Y	Y	N

The CAMs referred to are the RxByte Processor's CAMs (not be confused with the FPRx PDU ID CAM, which does not need to be loaded by software).

Warning: *The mechanisms described below can only be used when the FPRx is held in reset.*

There are two (2) RxByte Processors in the FPRx. When writing or reading from the associated stores, both are written and read at the same time. There is no mechanism to separately load RxByte Processor0's WCS/CAM and RxByte Processors1's WCS/CAM. This enforces RxByte Processor0 to run the same processor code/CAM data as RxByte Processor1. The CAM must be written and read using the internal scan access as described below. Refer to "Using the Internal Scan Access (Wr) for RxByte Processor's CAMs" on page 276, and "Using the Internal Scan Access (Rd) for RxByte Processor's WCS/CAM" on page 277.

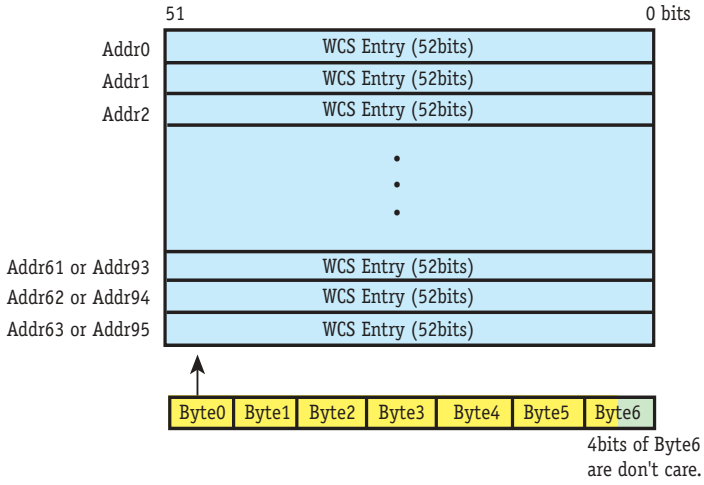
The WCSs must be written via the byte write mechanism described below. Data may only be read back from both the WCSs and CAMs via scan. This is intended for diagnostics use only (i.e. memory validation), as such the procedure is rather complex and not optimized for operational use. When the data is read back, via scan, the data from each of the two RxByte Processors is streamed back to bits *Scan_Out0* and *Scan_Out1* respectively. Data from the DBE is streamed to the DBE Scan Out bit.

The two (2) FPRx Byte Processors are loaded in parallel. That is, the two RxByte Processors run identical microcode.

Using the Special Byte Access (Wr only) for the RxByte Processor's WCSs and RxDBE's WCS

To byte write either the RxByte Processors or the RxDBE WCS's, the FP must be placed in reset. This ensures the byte 'Address' counter is pointing to Byte 0. There is no direct ability to specify the addressing in RxByte Processors or RxDBE WCS's. The WCSs are loaded left to right, with the most significant byte first. The most significant byte of each word is only half used, with the most significant 4bits of this byte as "don't care" values. Thus, each WCS "word" is 7Bytes long as shown in Figure 53 on page 275.

Figure 53 Byte Load Sequence Mapping to DBE's WCS and RxByte Processor's WCS



Note: DBE's WCS is 64Words (Addr0 to Addr63) and RxByte Processor's WCS are 96Words (Addr0 to Addr95).

Each of the three (3) WCS's (of the RxByte Processors and RxDBE) has a byte pointer that advances sequentially. The pointer starts with the most significant byte (nibble) of the WCS, and increments through to the least significant byte of the WCS moving from WCS location0 through to WCS location last. For RxDBE WCS the last location is 63, that is, left (MSB) to right (LSB), bottom (Addr0) to top (Addr 63). For RxByte Processors WCS's the last location is 95, that is, left (MSB) to right (LSB), bottom (Addr0) to top (Addr 95).

The byte write hardware along with the scan chain organization is designed and optimized for byte write loading of the WCS (vs. scan loading). Specifically, the CAM is first in the chain, followed by the WCS. This allows the CAM to be loaded first via scan with out the need to shift bits down the entire chain for the WCS. When the CAM is loaded in this fashion, the WCS will have undetermined data written to it. In this case, it is expected that the WCS are loaded after the CAMs have been initialized.

Using the Internal Scan Access (Wr) for RxByte Processor's CAMs

The CAM/WCS Scan Chain is 94bits long comprised of the following fields left to right where the right most bit of the right most field is shifted in first: CAM Addr, CAM Group, CAM Pattern, CAM Tag, WCS Data. The CAMs are at the near end of the chain (i.e. all WCS fields are at the end of the chain). Refer to Table 78 on page 277.

The CAMs, if used, should be written prior to the WCS Byte loading. It is only required to scan the 42bits of CAM fields and then perform an update operation using *FPRx WCS_CAM* register bit [2] *WCS/CAMScanUpdate* field. Scanning to the CAM may be done on a Random Access basis since the CAM Addr field selects the specific CAM location during the update operation. Both RxByte Processor0 and RxByte Processor1 CAMs are updated simultaneously. Refer to Figure 54 on page 276.

Figure 54 RxByte Processors Scan Chain

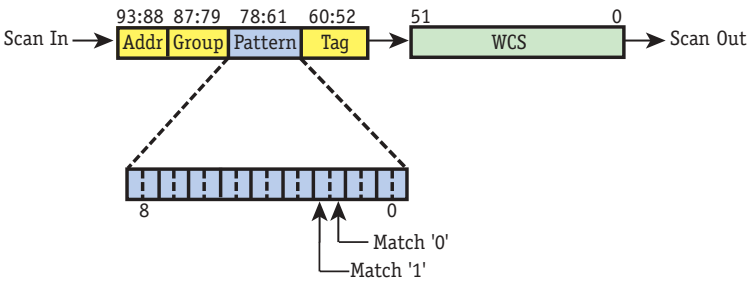


Table 78 RxByte Processors Scan Chain Fields

FIELD NAME	BIT POSITION	DESCRIPTION
Addr	93:88	**CAM Address —** Six bits. Used during Writes only to select a CAM location for initialization. The FPRx CAM is 24 entries long (i.e. Addr 0 to 23 valid).
Group	87:79	**CAM Group** — A nine bit index created by the RxByte Program Counter used during reads to qualify the pattern match.
Pattern	78:61	**CAM Match Pattern —** A nine – two bit pattern used to qualify a match on '1', '0', 'X'. The Pattern may be driven from either RxByte Processor IREG3 from the Payload bus. Each two sequential bits of this field select: 00 – Match 'X', 01 – Match '1', 10 – Match '0', 11 – Invalid entry (no match possible). Note: A 'X' is a don't care.
Tag	60:52	**CAM Tag —** A nine bit value associated with a match. When a Match on a pattern is made the Tag Value is available for use on the RxByte Processor B-Bus.
WCS	51:0	**Writable Control Store Data** —

WARNING: *Writing the CAMs can invalidate WCS entries. As such, the CAMs, if used, should be written first followed by WCS Byte write operations to load the WCSs.*

Using the Internal Scan Access (Rd) for RxByte Processor's WCS/CAM

The CAM/WCS Scan Access (Rd) Chain is the same 94bit chain used in the Scan Access (Wr) operation described above and is not optimized for operational reading (i.e. it is intended for diagnostic manufacturing pass/fail screening). The address of the CAM during reads is selected by the CAM Group and Match values, that is, the CAM addr field is not used. The address of the WCS is selected by the byte write counter, requiring a destructive write to the WCS prior to the SCAN CAPTURE selection via the FP WCSs register. To further complicate matters, and to avoid writing the word or WCS intended to be read, bit 2 of the byte address selection is inverted during the read so that you first write to location 0xABCD XOR 0x0004, to read 0xABCD. After you have captured the scan and scanned out the results, you select the next WCS address by performing seven (7) additional byte writes, thereby incrementing the WCS Byte address to the next word, and again performing a SCAN CAPTURE. In reading the WCS in this destructive fashion, you must be careful to either byte write original data back to 0xABCD XOR 0x0004 or to rewrite the entire WCS between reads.

In all cases where the WCS/CAM has been written, since the RxByte Processors0 and RxByte Processor1 CAMs and WCSs are written at the same time, Scan Out0 should always equal Scan Out1. Refer to *C-Ware Microcode Programming Guide (part number CSTMCPG-UG/D)* for details regarding Scan Out.

Using the Internal Scan Access (Rd) for DBE's WCS

The DBE Scan Chain is a separate 52bit Scan Chain but operates, using the appropriate *FPRx WCS_CAM* register bits, precisely as described in "Using the Internal Scan Access (Rd) for RxByte Processor's WCS/CAM" on page 277.

Chapter 5

BUFFER MANAGEMENT UNIT

Chapter Overview

This chapter covers the following topics:

Buffer Management Unit (BMU) Overview

The Buffer Management Unit (BMU) provides the interface to external SDRAM for the C-5e NP. The BMU partitions the SDRAM into buffers accessible to the Channel Processors (CPs), the Executive Processor (XP), and the Fabric Processor (FP) for payload storage. The BMU also provides services for managing the buffer handles called Buffer Tags (BTags) and services for maintaining BTag reference count tables called the Multi-Use Counters (MUC), which are used for forwarding payload to multiple targets.

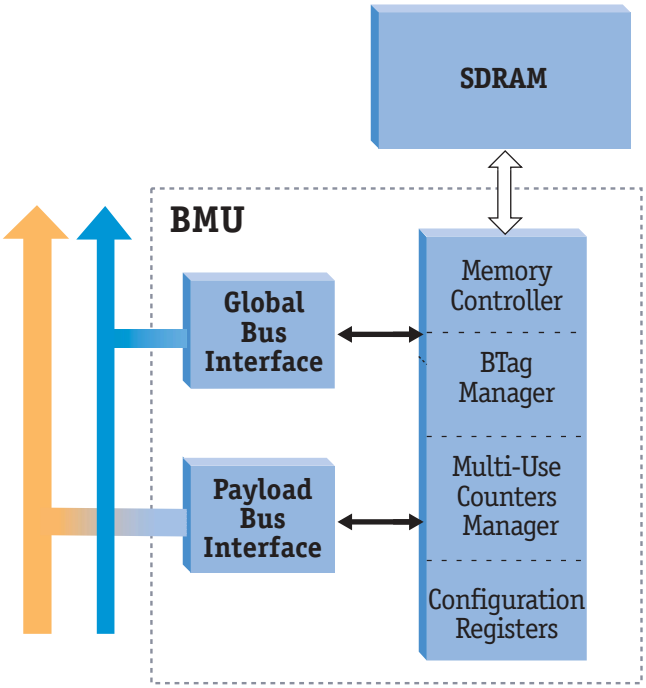
BMU Major Components

The major components of the BMU are listed inTable 79 on page 280. In addition, Figure 55 on page 281 shows the BMU Block Diagram.

Table 79 Major Components of the BMU and Their Functions

ITEM	FUNCTION
Memory Controller	Processes all requests for SDRAM transactions, primarily buffer memory requests for Payload storage. Buffer access is made from CP or XP application software, or the FP hardware using a Payload transaction Control Block (WrCB0, RdCB0,RxCB0,TxCB0).
BTag Manager	Handles BTag allocation and deallocation. BTag operations are made from CP or XP application software, or the FP hardware using a Payload transaction Control Block (WrCB0, RdCB0,RxCB0,TxCB0).
Multi-Use Counter Manager	Handles Multi-Use Counter (MUC) allocation, decrement and automatic BTag deallocation. MUC operations are made from CP or XP application software using a Payload transaction Control Block (WrCB0, RdCB0, RxCB0, TxCB0).
Configuration Registers	Used for setting up physical and buffer memory configuration, and for debug and test. Configuration operations are made from CP or XP application software using loads/stores from/to Global memory space.

Figure 55 BMU Block Diagram



BMU Physical Memory Organization

The SDRAM memory array is organized as 128bit words operating in accordance with Joint Electronic Device Engineering Council (JEDEC) specifications at 100MHz and 133MHz (depending on the SDRAM used). This provides a maximum bandwidth of 12.8Gbps or 16Gbps respectively. The BMU supports four-beat bursts of 16 Bytes each, optimized for 64-Mbyte parts, and for similar parts with four (4) internal banks. The C-5e NP supports one (1) physical bank of SDRAM, but a number of parts and arrays are supported. Registered DIMMs can be supported by adjusting timing parameters.

In addition to the 128bit words of user data, the BMU can be configured to handle an additional eleven (11) bits of data, two (2) Out-of-Band (OOB) bits and nine (9) ECC (Error Correction Code) bits when ECC is enabled.

When the Out-of-Band (OOB) (2bits) and ECC (9bits) are used the total bits stored is increased from 128bit words to 139bit words. Therefore, the number of parts increase to accommodate the additional 11bits. Refer to Table 80 on page 282.

Table 80 Supported SDRAM Configurations

PARTS*	NUMBER OF PARTS	CAPACITY OF SDRAM CARD	ADDRESS BITS
64Mbx8	18	128MB	27
64Mbx16	9	64MB	26
64Mbx32	5	32MB	25
128Mbx8	18	256MB	28
128Mbx16	9	128MB	27
128Mbx32	5	64MB	26
256Mbx16	9	256MB	28
256Mbx32	5	128MB	27

* The C-5e NP only supports 12bit row addressing SDRAM components. The row address must be exactly 12bits. The column addressing can be any number of bits, from 4 up to 12.

All transactions with the SDRAMs are 4 beat bursts=64Bytes of data. Writes of quantities < 16Bytes are not supported due to the addition of SECDED (Single Error Correcting, Double Error Detecting) ECC (Error Correction Code) support. Such writes would require read-modify-write transactions using up twice the write bandwidth.

Out-of-Band Bits

The Out-of-Band (OOB) bits hold control information that travels with payload data. The bits are organized as 8bits per 64Bytes of data and stored as 2bits for each 16Bytes. Therefore, to move all Out-of-Band (OOB) bits [7:0] with 64Bytes of user data the sequence is: 2bits are stored with the first 16Bytes of data, then the next 2bits are stored with the second 16Bytes of data, then the next 2bits are stored with the third 16Bytes of data, then the next 2bits are stored with the fourth and final 16Bytes of user data. Refer toTable 22 on page 127.

SECDED ECC Support

Data stored in SDRAM can be protected by a *Single Error Correcting, Double Error Detecting* (SECDED*) Error Correction Code* (ECC) if ECC is enabled and extra memory is included in the system. Nine (9) ECC bits can correct all single bit errors and detect all double bit errors across 130bits (128bits of data and 2bits for OOB) of data read/written per SDRAM clock cycle. For each 130bit write, nine (9) ECC *check bits* are generated and stored along with the user data (typically 128bits). When the data is read back from SDRAM, the nine (9) check bits are re-generated and checked against the check bits that were stored. If they are the same, then there is no error. If there is an odd number of bits that differ, then there is a single bit error. If there is an even number of bits that differ then there is a double bit error. This is implemented using the *ECC Enable* single bit register and the *Single ECC Error* register, that counts the number of (ECC) errors that have occurred. Refer to Table 91 on page 307.

BMU Buffer Memory Organization

The Buffer Memory is organized as described in the following sections.

Buffer Pools

The BMU divides SDRAM into sections called *buffer pools*. Pools are intended to provide protection among the many users of buffer memory, and to allow the applications (via chip configuration) to carve memory into different size buffers. Up to 30 buffer pools can be configured. Each Pool Area= (Buffer Size * Number of Buffers). The *Pool0 Base* to *Pool29 Base* registers are used to define the base address in SDRAM for a pool (Buffer Memory). Refer to Figure 56 on page 286.

Configuration software must ensure that pools do not overlap and that there is enough physical memory to hold all the pools.

Buffers

Each buffer pool contains up to 65,528 fixed size buffers. The number of buffers and size of the buffers can be different for each buffer pool. The number of buffers in a pool must be a multiple of eight (8) and the size of each buffer must be a power of two (2) between 64kBytes and 64Bytes, excluding 128Byte buffers. The Buffer Size is user selectable using the *Pool0 BTag Shift* to *Pool29 BTag Shift* registers. Refer to Table 81 on page 285 and Figure 56 on page 286.

Pools are generally configured during system initialization. Unpredictable behavior results when a pool is accessed prior to initialization. Refer to "BMU Setup" on page 310.

Buffer Tags (BTags)

Each buffer in a pool has a handle defined that identifies its location in the pool. These handles are called *Buffer Tags* (BTags). There is a one to one relationship between Buffers and BTags (1Buffer to 1BTag). Each BTag is 2Bytes. The BTags themselves are stored in SDRAM and inside the BMU. To allocate (assign) a buffer to a CP or XP, software must issue a BTag read (RdCB) request. Buffers are allocated in multiples of eight (8). The *Num BTag0* to *Num BTag29* registers are used to set the number of BTags in a Pool. The *BTag FIFO Base0* to *BTag Base29* registers are used to define the base address in SDRAM for the Pool BTag FIFO. Each Pool BTag FIFO Area= (2Bytes * Number of Buffers).

Storage Space (SDRAM Partitions)

The SDRAM space is partitioned using the variables shown in Table 81 on page 285. In addition, refer to Figure 56 on page 286.

Table 81 Legal Ranges for SDRAM Partition Variables

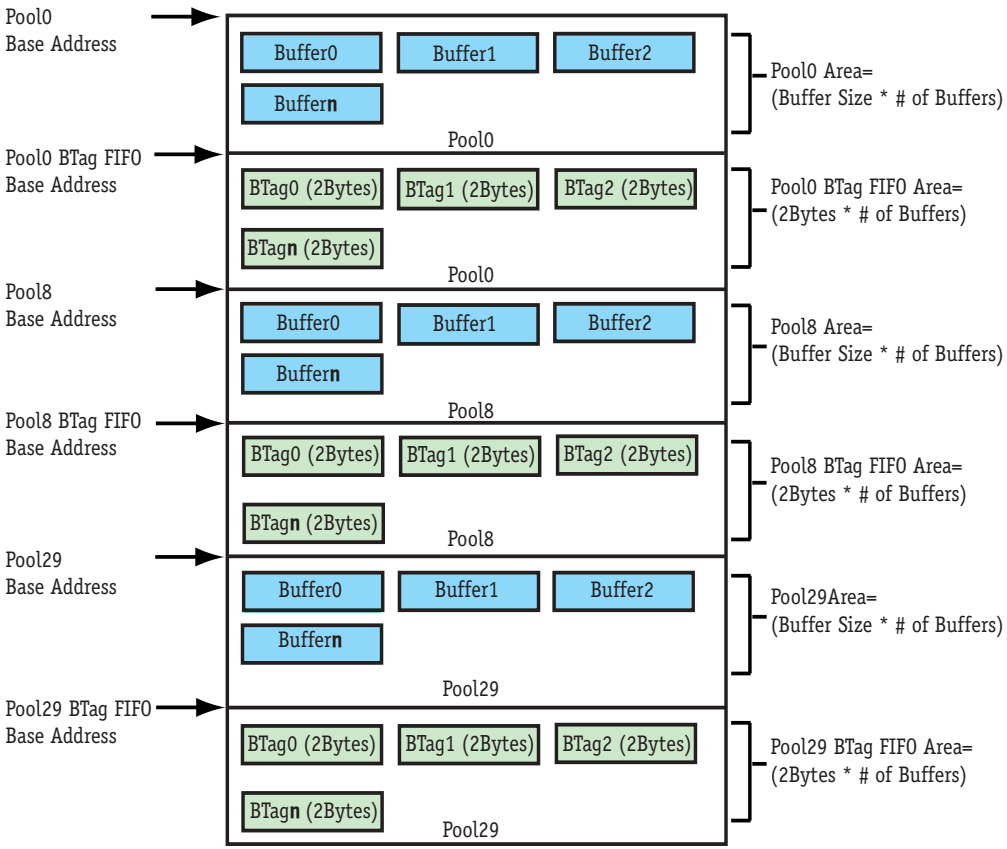
ITEM	RANGE
Number of Pools	0 to 29
Number of Buffers per Pool	0 to 65,528 (must be in multiples of 8)
Individual Buffer Size	SIZE / ENCODED VALUE 64kB / 0 32kB / 1 16kB / 2 8kb / 3 4kB / 4 2kB / 5 1kB / 6 512B / 7 256B / 8 128B / (Not Supported) 64B / 10
Number of BTags per Pool	0 to 65,528 (must be in multiples of 8)

Buffer Access

All transactions with the SDRAM are 64Bytes in length. Access to buffers<64Bytes in length still requires 64Byte transactions. Operations of <16Bytes of data are not supported in the BMU. All buffer accesses must be aligned to 16Byte boundaries. The minimum size of an internal data transfer is 64Bytes, taking four (4) 16Byte slots on the Payload Bus. Buffer transfers of <64Bytes result in empty slots on the Payload Bus. Buffer writes of<64Bytes use data masking to suppress the undesired writes to SDRAM.

The BMU is optimized for 64Byte aligned access to buffers. Unaligned transfers are possible, but require special handling. Refer to “Unaligned Buffers” on page 290.

Figure 56 SDRAM Storage Space for User Data Example



Types of Transactions

The BMU supports seven (7) functions divided into three (3) categories. The different functions are initiated by CPs, XP or the FP using the Multi-Use Control Blocks by just changing the fields. Multi-Use Control Blocks use the following registers: WrCB0_Sys_Addr, WrCB0_Ctl, WrCB0_DMA_Addr, WrCB0_SDP_Addr; RxCB0_Sys_Addr, RxCB0_Ctl, RxCB0_DMA_Addr, RxCB0_SDP_Addr; RdCB0_Sys_Addr, RdCB0_Ctl, RdCB0_DMA_Addr, RdCB0_SDP_Addr; and TxCB0_Sys_Addr, TxCB0_Ctl, TxCB0_DMA_Addr, TxCB0_SDP_Addr. Refer to Table 82 on page 287, Table 83 on page 288, and Table 84 on page 289.

Table 82 Multi-Use Control Blocks (for Wr, Rx, Rd and Tx)

MODE	CATEGORY	FUNCTION	FIELDS USED	DETAILS
• CP to/from BMU • XP to/from BMU • FP to/from BMU	Memory Transactions	Buffer Memory Transfer Operation	PoolID, BTag, Offset	See "Using Wr/Rd Control Blocks for Payload Transactions" on page 290 and "Using Rx/Tx Control Blocks for Payload Transactions" on page 290.
	BTag Management Transactions	Initializing BTags	PoolID, BTag, Command, Pool	See"BTag Initialization Operation" on page 292.
		Allocating BTags		See "BTag Allocation Operation" on page 295.
		Deallocating BTags		See "BTag Deallocation Operation" on page 297.
	Multi-Use Counter Management Transaction	Allocating Multi-Use Counters		See "MUC Allocation Operation" on page 300.
		Decrementing Multi-Use Counters		See "MUC Decrement Operation" on page 303.
		Reading Multi-Use Counters		See "MUC Read Operation" on page 305.

Table 83 WrCB0_ Variables per Field for BMU

<table>
<tr><th>REGISTER</th><th>FIELD NAME</th><th>BIT POSITION</th><th>DESCRIPTION</th></tr>
<tr><td>WrCB0_DMA_ Addr</td><td>PoolID</td><td>20:16</td><td>PoolID —
<table>
<tr><th>OPERATION TYPE</th><th>VALUE</th></tr>
<tr><td>Buffer Memory Transfer</td><td>0 to 29</td></tr>
<tr><td>BTag</td><td>30</td></tr>
<tr><td>Multi-Use Counter</td><td>30</td></tr>
</table></td></tr>
<tr><td rowspan="4">WrCB0_Sys_ Addr</td><td>BTag</td><td>31:16</td><td>Buffer Tag —
<table>
<tr><th>OPERATION TYPE</th><th>VALUE</th></tr>
<tr><td>Buffer Memory Transfer</td><td>Enter the BTag associated with the Buffer. Legal Range= 0 to 65528.</td></tr>
<tr><td>BTag</td><td>0</td></tr>
<tr><td>Multi-Use Counter</td><td>Enter the BTag associated with the counter. Legal Range= 0 to 65528.</td></tr>
</table></td></tr>
<tr><td>Offset</td><td>15:4</td><td>Offset —
<table>
<tr><th>OPERATION TYPE</th><th>VALUE</th></tr>
<tr><td>Buffer Memory Transfer</td><td>Enter the Offset within a Buffer.</td></tr>
</table></td></tr>
<tr><td>CMD</td><td>15:9</td><td>Command —
<table>
<tr><th>OPERATION TYPE</th><th>VALUE</th></tr>
<tr><td>BTag Initialization</td><td>0</td></tr>
<tr><td>BTag Deallocate</td><td>1</td></tr>
<tr><td>Multi-Use Counter Allocation</td><td>2</td></tr>
<tr><td>Multi-Use Counter Decrement</td><td>3</td></tr>
</table></td></tr>
<tr><td>Pool</td><td>8:4</td><td>Buffer Pool —
<table>
<tr><th>OPERATION TYPE</th><th>FUNCTION</th><th>VALUE</th></tr>
<tr><td>BTag Initialization</td><td>Enter the Pool to write to.</td><td rowspan="4">0 to 29</td></tr>
<tr><td>BTag Deallocation</td><td>Enter the Pool of the Buffer being deallocated.</td></tr>
<tr><td>Multi-Use Counter Allocation</td><td>Enter the Pool associated with the counter.</td></tr>
<tr><td>Multi-Use Counter Decrement</td><td>Enter the Pool associated with the counter.</td></tr>
</table></td></tr>
</table>

Table 84 RdCB0_ Variables per Field for BMU

<table>
<tr><th>REGISTER</th><th>FIELD NAME</th><th>BIT POSITION</th><th>DESCRIPTION</th></tr>
<tr><td>RdCB0_DMA_ Addr</td><td>PoolID</td><td>20:16</td><td>PoolID —
<table><tr><th>OPERATION TYPE</th><th>VALUE</th></tr><tr><td>Buffer Memory Transfer</td><td>0 to 29</td></tr><tr><td>BTag</td><td>30</td></tr><tr><td>Multi-Use Counter</td><td>30</td></tr></table></td></tr>
<tr><td rowspan="4">RdCB0_Sys_ Addr</td><td>BTag</td><td>31:16</td><td>Buffer Tag —
<table><tr><th>OPERATION TYPE</th><th>VALUE</th></tr><tr><td>Buffer Memory Transfer</td><td>Enter the BTag associated with the Buffer. Legal Range= 0 to 65528.</td></tr><tr><td>BTag</td><td>0</td></tr><tr><td>Multi-Use Counter</td><td>Enter the BTag associated with the counter. Legal Range= 0 to 65528.</td></tr></table></td></tr>
<tr><td>Offset</td><td>15:4</td><td>Offset —
<table><tr><th>OPERATION TYPE</th><th>VALUE</th></tr><tr><td>Buffer Memory Transfer</td><td>Enter the Offset within a Buffer.</td></tr></table></td></tr>
<tr><td>CMD</td><td>15:9</td><td>Command —
<table><tr><th>OPERATION TYPE</th><th>VALUE</th></tr><tr><td>BTag Allocation</td><td>0</td></tr><tr><td>Multi-Use Counter Read</td><td>1</td></tr></table></td></tr>
<tr><td>Pool</td><td>8:4</td><td>Buffer Pool —
<table><tr><th>OPERATION TYPE</th><th>FUNCTION</th><th>VALUE</th></tr><tr><td>BTag Allocation</td><td>Enter the Pool from which to allocate the Buffers.</td><td rowspan="2">0 to 29</td></tr><tr><td>Multi-Use Counter Read</td><td>Enter the Pool associated with the counter.</td></tr></table></td></tr>
</table>

Buffer Memory Transactions

Buffer Memory Transactions are Payload Data Block Moves using Control Blocks (WrCB0, RdCB0, RxCB0 and TxCB0). Each is described here.

Using Wr/Rd Control Blocks for Payload Transactions

Writes to SDRAM and reads from SDRAM use: WrCB0_Sys_Addr, WrCB0_Ctl, WrCB0_DMA_Addr registers and RdCB0_Sys_Addr, RdCB0_Ctl, RdCB0_DMA_Addr registers. Refer to "Write Control Blocks (WrCB0_ , WrCB1_)" on page 116 and "Read Control Blocks (RdCB0_ , RdCB1_)" on page 120.

These registers (WrCB0_Sys_Addr, WrCB0_Ctl, WrCB0_DMA_Addr and RdCB0_Sys_Addr, RdCB0_Ctl, RdCB0_DMA_Addr) are physically located in the respective CPs, XP, and FP and not in the BMU Configuration Space.

Using Rx/Tx Control Blocks for Payload Transactions

Receiving payload to SDRAM and transmitting payload from SDRAM use: RxCB0_Sys_Addr, RxCB0_Ctl, RxCB0_DMA_Addr, RxCB0_SDP_Addr registers and TxCB0_Sys_Addr,TxCB0_Ctl, TxCB0_DMA_Addr, TxCB0_SDP_Addr registers. Refer to "SDP RxByte Processor Receive Control Blocks (RxCB0_, RxCB1_)" on page 123 and "SDP TxByte Processor Transmit Control Block (TxCB0_, TxCB1_)" on page 128.

These registers (RxCB0_Sys_Addr,RxCB0_Ctl, RxCB0_DMA_Addr, RxCB0_SDP_Addr and TxCB0_Sys_Addr, TxCB0_Ctl, TxCB0_DMA_Addr, TxCB0_SDP_Addr) are physically located in the respective CPs, and XP and not in the BMU Configuration Space.

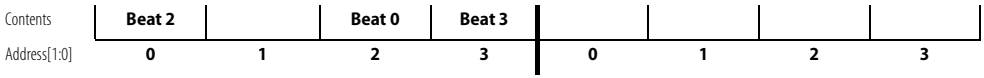
Read/Write Ordering

Since SDRAM is four-way bank interleaved, the BMU uses a round-robin algorithm to choose requests for each bank. This can result in a read response returning in an order other than the order they were issued or acknowledged on the buses.

Unaligned Buffers

The memory controller reads and writes to the SDRAM in naturally aligned 64Byte quantities. Any portion of a naturally aligned 64Bytes block can be read or written; however, special attention must be given to algorithms that require the crossing of a 64Byte boundary. Any transaction that attempts to read or write a data length from an address that causes the least significant two (2) bits of offset to increment from 0x3 to 0x0 will wrap. That is, the other bits of the address are not affected. For example, a write of length 48Bytes to an address with offset bit [1:0]== 0x2 will write memory as shown in Figure 57 on page 291.

Figure 57 Buffer Wrapping



If the intent is to write across the 64Byte boundary then two (2) writes are required. For the same alignment as above, the first write is length 32Bytes at offset bits [1:0] == 0x2 and the second write is length 16Byte at the address of the next contiguous block. Refer to Figure 58 on page 291.

Figure 58 Unaligned Buffer Access

BTag Management Transactions

The BMU maintains a BTag FIFO for BTag allocation and deallocation.

Space for the entire BTag FIFO for each pool is located in the SDRAM, the location defined in the *BTag FIFO Base0* to *BTag FIFO Base29* registers. BTags are allocated from the FIFO and deallocated to the FIFO. The BMU reads BTags in groups of eight (8) and collects eight (8) BTags before writing them back to the FIFO. The BMU maintains an on-chip, hardware-managed cache that can temporarily store BTags from the various pools. The BTag cache typically provides quicker access for allocation and deallocation of BTags and reduces the use of SDRAM bandwidth for BTag management. When the BTag FIFO Cache is empty, operations bypass directly to SDRAM FIFO space. The BTag FIFO Cache space and BTag FIFO SDRAM space are extensions of each other rather than a subset/superset relationship.

BTag Transaction Functions (Operation and Examples)

BTag transactions consist of three (3) different functions: Initialization, Allocation, and Deallocation. BTag transactions are invoked using Control Blocks (WrCB0, and RdCB0). Each is described here along with examples.

BTag Initialization Operation

Warning: *All BTags must be initialized by software before allocating them to access buffer memory.*

BTag Initialization uses a control block (WrCB0) to write starting values from the DMEM of either the requesting CP or XP to initialize BTag FIFO Space.

The base address of the BTag FIFO SDRAM Space is specified in the BTag FIFO Base address register for each pool (*BTag FIFO Base0* to *BTag FIFO Base29*). The size of the Pool*n* BTag FIFO Area= (2Bytes * Number of Buffers). A BTag must be written for each buffer in the pool. The number of BTags per pool must be a multiple of eight (8) because all BTags are written, stored, and read in groups of eight (8).

BTags are 16bit values written to the BTag FIFO Space in groups of 8, 16, 24, or 32. The write control blocks (WrCB) are used for this purpose. The software must generate a buffer in local DMEM containing the 16bit BTags. The BTag numbers themselves can be in any order but they must use all integral values for a given pool from 0 to the number of BTags minus 1. Internally, BMU hardware takes care of allocating the BTags between BTag FIFO Cache Space and BTag FIFO SDRAM Space.

BTag Initialization Example

Buffer Initialization uses a control block (WrCB0) to write the BTags to the BMU from the DMEM of either the requesting CP or XP.

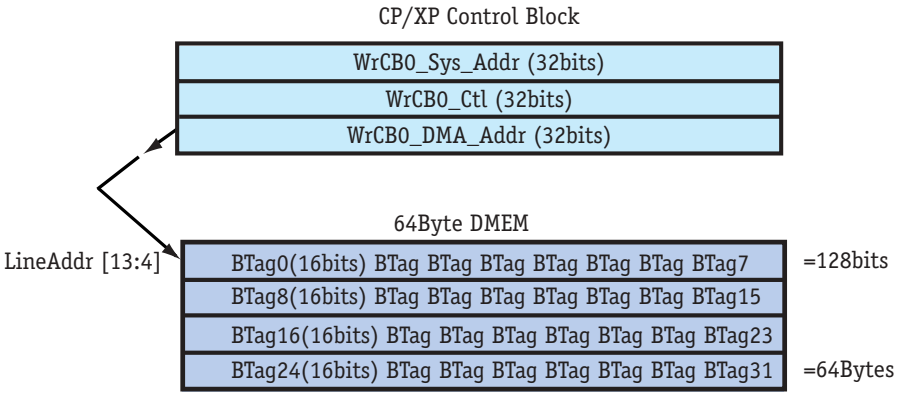
The bits for WrCB0_Sys_Addr, WrCB0_Ctl, and WrCB0_DMA_Addr are set as shown in Table 85 on page 293.

Table 85 WrCB0_ Settings for BTag Initialization

REGISTER	FIELD NAME	BIT POSITION	DESCRIPTION
WrCB0_Ctl	Length	13:0	**Length** — Length of DMA transfer in Bytes. Legal range =16 to 64Bytes. LENGTH (BYTES) / NUMBER OF BTAGS 16 / 8 32 / 16 48 / 24 64 / 32
WrCB0_Sys_Addr	BTag	31:16	**Buffer Tag** — Enter 0 for BTag Operation.
	CMD	15:9	**Command** — Enter 0 for BTag Initialization.
	Pool	8:4	**Buffer Pool** — Pool to write to. Legal range= 0 to 29
WrCB0_DMA_Addr	PoolID	20:16	**PoolID** — Enter 30 for BTag Operation.
	LineAddr	13:4	**DMEM Line Address** — DMEM 16Byte line address for DMA.

The WrCB0_DMA_Addr bits [13:0] *LineAddr* field refers to the Local DMEM buffer address that is generally 64Byte aligned. All 32BTags, (16bit BTags, 128bits) are located inside the 64Byte DMEM as shown in Figure 59 on page 294.

Figure 59 BTag Initialization Implementation



BTag Allocation Operation

Buffer Allocation uses a control block (RdCB0) to read BTags from the BMU into the DMEM of the requesting CP, XP, or FP. Allocation assigns a particular BTag (from the BMU) to be used by a particular processor.

BTag Allocation Example

The bits for RdCB0_Sys_Addr, RdCB0_Ctl and RdCB0_DMA_Addr are set as shown in Table 86 on page 295.

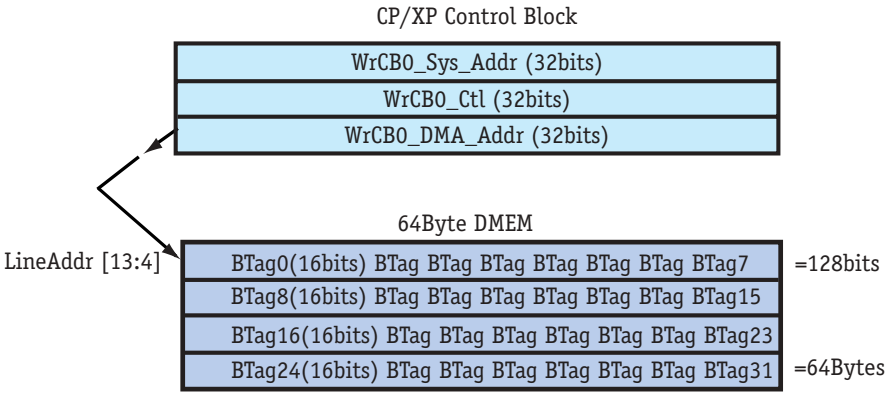
Table 86 RdCB0_ Settings for BTag Allocation

REGISTER	FIELD NAME	BIT POSITION	DESCRIPTION
RdCB0_Ctl	Length	13:4	**Length** — Length of DMA transfer in Bytes. Legal range=16 to 64Bytes.
RdCB0_Sys_Addr	BTag	31:16	**Buffer Tag** — Enter 0 for BTag Operation.
	CMD	15:9	**Command** — Enter 0 to BTag Allocation.
	Pool	8:4	**Buffer Pool** — Pool from which to allocate the Buffers. Legal range= 0 to 29
RdCB0_DMA_Addr	PoolID	20:16	**PoolID** — Enter 30 for BTag Operation.
	LineAddr	13:4	**DMEM Line Address** — DMEM 16Byte line address for DMA.

LENGTH (BYTES)	NUMBER OF BTAGS
16	8
32	16
48	24
64	32

The RdCB0_DMA_Addr bits [13:4] *LineAddr* field refers to the Local DMEM buffer address that is generally 64Byte aligned. All 32BTags, (16bit BTags, 128bits) are located inside the 64Byte DMEM as shown in Figure 60 on page 296.

Figure 60 BTag Allocation Implementation



BTag Deallocation Operation

Buffer Deallocation uses a control block (WrCB0) to write BTags back to the BMU from DMEM by either the requesting CP, XP, or FP. Deallocation returns a particular BTag (from a particular CP or XP) back to a pool in the BMU.

BTag Deallocation Example

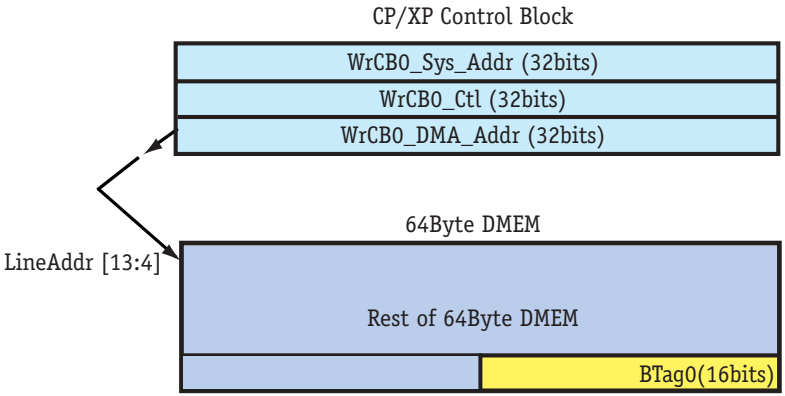
The bits for WrCB0_Sys_Addr, WrCB0_Ctl and WrCB0_DMA_Addr are set as shown in Table 87 on page 297.

Table 87 WrCB0_ Settings for BTag Deallocation

REGISTER	FIELD NAME	BIT POSITION	DESCRIPTION
WrCB0_Ctl	Length	13:0	**Length** — Length of DMA transfer in Bytes. Legal value=16Bytes, required to access 1 line of DMEM. LENGTH (BYTES) / NUMBER OF BTAGS 16 / 1
WrCB0_Sys_Addr	BTag	31:16	**Buffer Tag** — Enter 0 for BTag Operation
	CMD	15:9	**Command** — Enter 1 for BTag Deallocation.
	Pool	8:4	**Buffer Pool** — Pool of Buffers being Deallocated. Legal range= 0 to 29
WrCB0_DMA_Addr	PoolID	20:16	**PoolID** — Enter 30 for BTag Operation.
	LineAddr	13:4	**DMEM Line Address** — DMEM 16Byte line address for DMA.

The WrCB0_DMA_Addr bits [13:4] *LineAddr* field refers to the Local DMEM buffer address that is 16Byte aligned. The first two Bytes inside the first 32bit word of the 64Byte DMEM holds the BTag (the first two Bytes) as shown in Figure 61 on page 298.

Figure 61 BTag Deallocation Implementation



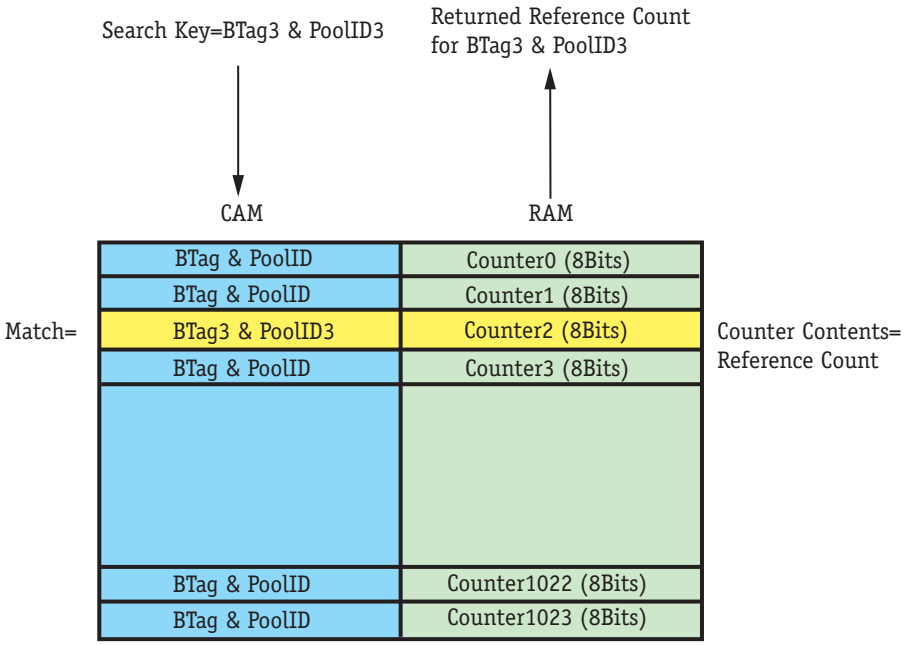
Multi-Use Counter (MUC) Management Transactions

Multi-Use Counters (MUC) are used to track buffer accesses when a buffer has multiple targets (CPs, XP, or FP), such as, a multicast packet. Each time an application running on a particular CP accesses the Multi-Use buffer, its Multi-Use Counter is decremented. When a counter reaches zero, all users have accessed the buffer and the BTag is deallocated.

Typically, the software prefetches a number of BTags without knowing whether or not they are going to be used as single BTags or Multi-Use BTags. That fact only becomes apparent later during processing, *after* the buffer has been written to memory. At that point the software tries to allocate a counter for the Multi-Use BTag from a particular Pool.

There are 1024 8bit counters available. One counter is associated with one (1) BTag at any one time using a *Content Addressable Memory* (CAM) array. BTag & PoolID are stored in the CAM to form the association. An initial Reference Count is stored in the counter. When the software must associate a counter with a buffer, it sends a command to the BMU to allocate a MUC. Refer to Figure 62 on page 300.

Figure 62 Multi-Use Counter Table



MUC Transaction Functions (Operation and Examples)

MUC transactions consist of three (3) different functions: Allocation, Decrement, and Read. MUC transactions are invoked using Control Blocks (WrCB0, and RdCB0). Each is described here along with examples.

MUC Allocation Operation

MUC Allocation uses a control block (WrCB0) to write an initial reference count from DMEM of the requesting CP or XP. MUC Allocation assigns the (BTag & Pool) to a MUC (from the BMU) with the initial reference count.

MUC Allocation Example

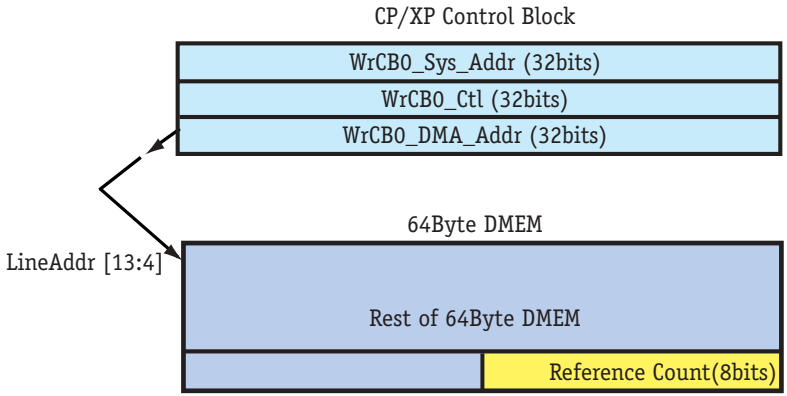
The bits for WrCB0_Sys_Addr, WrCB0_Ctl, and WrCB0_DMA_Addr are set as shown in Table 88 on page 301.

Table 88 WrCB0_ Settings for Multi-Use Counter Allocation

REGISTER	FIELD NAME	BIT POSITION	DESCRIPTION
WrCB0_Ctl	Length	13:0	**Length** — Length of DMA transfer in Bytes. Legal value=16Bytes, required to access 1 line of DMEM. LENGTH (BYTES) / NUMBER OF BTAGS 16 / 1
WrCB0_Sys_Addr	BTag	31:16	**Buffer Tag** — The BTag associated with the counter. Legal range= 0 to 65528.
	CMD	15:9	**Command** — Enter 2 for Multi-use Counter Allocation.
	Pool	8:4	**Buffer Pool** — The Pool associated with the counter. Legal range= 0 to 29.
WrCB0_DMA_Addr	PoolID	20:16	**Pool ID** — Enter 30 for Multi-Use Counter Operation.
	LineAddr	13:4	**DMEM Line Address** — DMEM 16Byte line address for DMA.

The WrCB0_DMA_Addr bits [13:4] *LineAddr* field refers to the Local DMEM buffer address that is 16Byte aligned. The first Byte inside the first 32bit word of the 64Byte DMEM holds the reference count as shown in Figure 63 on page 302.

Figure 63 Multi-Use Counter Allocation Implementation



MUC Decrement Operation

MUC Decrement uses a control block (WrCB0) to identify a MUC in the BMU and decrement the associated reference count. Only one (1) counter can be decremented per operation. When the MUC decrements to zero, the BMU hardware automatically deallocates the counter and the associated BTag.

MUC Decrement Example

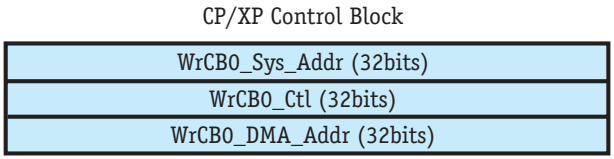
The write control block (WrCB) is used to send the BMU MUC decrement command. The bits for WrCB0_Sys_Addr, WRCB0_Ctl, and WrCB0_DMA_Addr are set as shown in Table 89 on page 303.

Table 89 WrCB0_ Settings for Multi-Use Counter Decrement

REGISTER	FIELD NAME	BIT POSITION	DESCRIPTION
WrCB0_Ctl	Length	13:0	**Length** — Length of DMA transfer in Bytes. Legal value=16Bytes, required to generate a single line operation. LENGTH (BYTES) | NUMBER OF BTAGS 16 | 1
WrCB0_Sys_Addr	BTag	31:16	**Buffer Tag** — The BTag associated with the counter. Legal Range= 0 to 65528.
	CMD	15:9	**Command** — Enter 3 for Multi-Use Counter Decrement.
	Pool	8:4	**Buffer Pool** — The Pool associated with the counter. Legal Range= 0 to 29.
WrCB0_DMA_Addr	PoolID	20:16	**PoolID** — Enter 30 for Multi-Use Counter Operation.
	LineAddr	13:4	**DMEM Line Address** — Not used.

The WrCB0_DMA_Addr bits [13:4] *LineAddr* field is not used as shown in Figure 64 on page 304.

Figure 64 Multi-Use Counter Decrement Implementation



MUC Read Operation

A control block (RdCB0) is used to read specific MUC contents from the BMU into DMEM by either the requesting CP or XP. This function is intended for debug and test purposes. A MUC read request can read one (1) counter per operation.

MUC Read Example

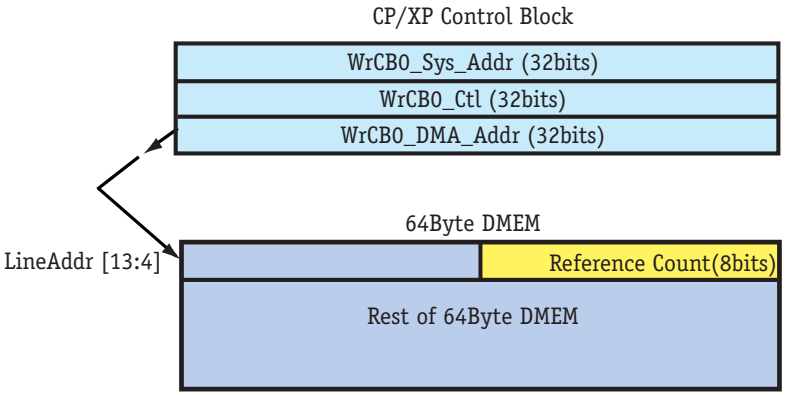
The bits for RdCB0_Sys_Addr, RdCB0_Ctl and RdCB0_DMA_Addr are set as shown in Table 90 on page 305.

Table 90 RdCB0_ Settings for Multi-Use Counter Read

REGISTER	FIELD NAME	BIT POSITION	DESCRIPTION
RdCB0_Ctl	Length	13:4	**Length** — Length of DMA transfer in Bytes. Legal value=16Bytes, required to access 1 line of DMEM. LENGTH (BYTES) / NUMBER OF BTAGS 16 / 1
RdCB0_Sys_Addr	BTag	31:16	**Buffer Tag** — The BTag associated with the counter. Legal Range= 0 to 65528.
	CMD	15:9	**Command** — Enter 1 for Multi-Use Counter Read.
	Pool	8:4	**Buffer Pool** — The Pool associated with the counter. Legal Range= 0 to 29.
RdCB0_DMA_Addr	PoolID	20:16	**PoolID** — Enter 30 for Multi-Use Counter Operation.
	LineAddr	13:4	**DMEM Line Address** — DMEM 16Byte line address for DMA.

The RdCB0_DMA_Addr bits [13:4] *LineAddr* field refers to the Local DMEM buffer address that is 16Byte aligned. The first Byte inside the first 32bit word of the 64Byte DMEM holds the reference count as shown in Figure 65 on page 306.

Figure 65 Multi-Use Counter Read Implementation



BMU Configuration Space

The BMU has memory-mapped Configuration Space that contains a number of registers. The registers are used for three (3) purposes: physical memory configuration, buffer memory configuration, and test and debug. Refer to Table 91 on page 307 for a list of BMU registers by function.

Table 91 BMU Registers

BMU REGISTER TYPES	REGISTER FUNCTION	SPECIFIC REGISTER DETAILS
Physical Memory Configuration	Physical memory size in Bytes. This configuration register is written with a value representing the amount of physical memory that software had determined was present in the system.	See "Memory Size Register (Miscellaneous Function)" on page 663.
	SDRAM controller configuration register. A write to this register tells the SDRAM controller the timing properties of the SDRAM and also initiates the SDRAM configuration process.	See "SDRAM Config Register (Miscellaneous Function)" on page 664.
	A single bit in a register that enables the Single Error Correction/Double Error Detecting (SECDED) error code if set to 1. ECC is disabled if the register bit is set to 0.	See "ECC Enable and Test Enable Register (Miscellaneous Function)" on page 665.
Buffer Memory Configuration	Starting physical address in SDRAM for each Pool.	See "Pool0 Base to Pool29 Base Registers (Buffer Pool Base Address Function)" on page 660.
	BTag shift amount is an encoded version of buffer size for each Pool telling hardware how much to shift the BTag during address calculations.	See "Pool0 BTag Shift to Pool29 BTag Shift Registers (Buffer Size for a Pool Function)" on page 661.
	Starting physical address in SDRAM for the BTag FIFO for each Pool.	See "BTag FIFO Base0 to BTag FIFO Base29 Registers (BTag FIFO Base Address Function)" on page 662.
	The number of BTags in each Pool.	See "Num BTags0 to Num BTags29 Registers (Number of BTags in a Pool Function)" on page 662.

Table 91 BMU Registers (continued)

BMU REGISTER TYPES	REGISTER FUNCTION	SPECIFIC REGISTER DETAILS
Test and Debug	This read-only register counts the number of single Error Correction Code (ECC) errors that have occurred.	See "Single ECC Errors Register (Miscellaneous Function)" on page 665.
	Control for ECC read and write test modes.	See "ECC Enable and Test Enable Register (Miscellaneous Function)" on page 665.
	BMU C-5e NP debug register in canonical format.	See "Debug Config Register (Miscellaneous Function)" on page 666.
	These two (2) registers capture the write address of a transaction that led to a write memory violation.	See "Wr_Mem_Violation_Hi Register (Miscellaneous Function)" on page 667 and "Wr_Mem_Violation_Lo Register (Miscellaneous Function)" on page 667.

Test and Debug Registers

The BMU contains various registers to provide test and debug access to internal state.

Memory Error Reporting

The *Single ECC Error* register is reset to 0 by hardware. After reset, the register counts the number of corrected single-bit ECC errors encountered during SDRAM access. Single-bit, corrected errors are not reported anywhere else.

Error conditions detected by the BMU are generally reported back to the requester except for single-bit ECC corrected errors and some violations on write transactions. The *Wr_Mem_Violation_Hi* and *Wr_Mem_Violation_Lo* registers capture the global or payload address of transactions that cause write violations.

ECC Test Modes

The *ECC_Enable and Test _Enable* register controls error checking during normal operation. In addition, the ECC Enable and Test Enable register provides ECC write and read test modes for testing ECC RAMs and portions of the chip data path. When bit [1] *ECC Write Test* field is enabled, the *ECC WriteTest Bits* field bits [10:2] provide the test ECC write data directly rather than the normal ECC generation logic. When this mode is enabled, all four (4) 16Byte beats of a payload write transaction write the same test ECC write data. When *ECC Read Test Enable* field bit [11] is enabled, rather than checking the ECC, the ECC bits are returned directly from SDRAM in the least significant 9bits of the data on a payload read transaction. All four (4) 16Byte beats of the payload read return the associated ECC data for the beat.

Debug Register

The BMU has a tap for the global debug logic. The *Debug Config* register controls multiplexors that allow selection of various BMU events or transactions for routing to the global debug counters located in the XP. Refer to "XP Debug Mode Register (XP Mode Configuration Function)" on page 625, and "Debug Counter0 Control Register (XP Configuration Function)" on page 618.

For complete details about specific registers go to their reference. Refer to "Buffer Management Unit (BMU) Configuration Registers" on page 654.

BMU Setup

Prior to the CP, XP, or FP accessing the SDRAM, the BMU must be set up properly, configured, and initialized as described in the following steps and using the applicable registers listed in Table 91 on page 307.

Warning: *Attempting to access a buffer pool before it is set up results in unpredictable behavior.*

1 Configure physical memory:

   a Write encoded physical memory size (either 64, 128, or 256MBytes) to the *Memory Size* register.

   b Write memory timing parameters to the *SDRAM Config* register.

   c Hardware disables ECC error correction and detection on reset. Write a 1 to bit [0] of the *ECC Enable* register to enable checking.

2 Configure buffer memory:

   a Write the physical address for the starting location in SDRAM for each Pool used into the *Pool0 Base* to *Pool29 Base* registers. Pool Base registers for unused Pools need not be configured.

   b Write the BTag shift amount to set buffer size for each Pool used into the *Pool0 BTag Shift* to *Pool29 BTag Shift* registers. Pool BTag Shift registers for unused Pool need not be configured.

   c Write the physical address for the starting location in SDRAM for the BTag FIFO for each Pool used into the *BTag FIFO Base0* to *BTag FIFO Base29* registers. BTag FIFO Base registers for unused Pools need not be configured.

   d Write number of BTags for each Pool used into the *Num BTag0* to *Num BTag29* registers. The number of BTags per pool must be a multiple of 8. Num BTag registers for unused Pools need not be configured.

3 Initialize BTags:

   a All BTags must be initialized before they can be allocated. Initialization software must write the BTag values to the BTag FIFO for each Pool used using the BTag initialization payload transaction. Refer to "BTag Initialization Example" on page 293. BTags can be written in groups of 8, 16, 24, or 32 at a time.

Specific values can be written in any order, but when completely initialized each FIFO must use all the BTags numbered from 0 to Pool size minus 1.

If memory size is not known at configuration time, software must auto size physical memory. Hardware sets Memory Size to the maximum value at reset. Software configures a temporary pool or fabricates a BTag directly (in this case 0) without allocating and writes to location 0 of physical memory. Then software writes to location 64M and reads back location 0. If location 0 is overwritten, the physical memory limit has been reached and the address wrapped. If location is not overwritten, software tests the next physical memory boundary, that is, 128M, and so on until the physical memory limit is discovered. Then the Memory Size register can be written and buffer memory configured for normal operation.

Chapter 6

TABLE LOOKUP UNIT

Chapter Overview

This chapter covers the following topics:

Table Lookup Unit (TLU) Overview

The Table Lookup Unit (TLU) provides access to application-defined routing topology, control, and statistics tables in external SRAM. It accesses an external SRAM array operating at up to 133MHz. Communication between the processors (CPs, XP, and FP) and the TLU is carried out via messages passed on the Ring Bus. Each processor (16CPs, XP, and FP), as well as the TLU is a *node* on the Ring Bus. The Ring Bus uses a 64bit wide data path. Refer to "Ring Bus Overview" on page 477.

The internal architecture of the TLU is extensively pipelined. This allows the TLU to service a number of outstanding requests simultaneously to ensure the most efficient use of the available external SRAM cycles.

The TLU supports several types of table lookup algorithms and provides resources for efficient generation of table entry addresses in SRAM, "hash" generation of addresses, and binary table searching algorithms for both exact-match and longest-prefix match strategies.

The TLU also provides resources for efficiently managing and manipulating table keys and associated data. Table entry insertion and deletion can be performed by the XPRC, CPRCs, or by memory mapped access to the Ring Bus registers on the XP from an external host processor.

The TLU allows: the mapping of thirty-two (32) lookup tables, supports four (4) different table types in hardware that can be used to support six (6) algorithms in software, and configurable table sizes.

The associated data maintenance facilities of the TLU also serve as a high-performance statistics accumulation resource and as an intermediate storage medium for segmentation and reassembly (SAR) operations.

The C-5e NP uses external 68bit data bus, and a 31bit control bus to connect to ZBT SRAMs (at frequencies up to 133MHz) for storage of its tables. The C-5e allows implementation of tables up to 2^{24}x 64bit entries using up to 64Mbit SRAM technology.

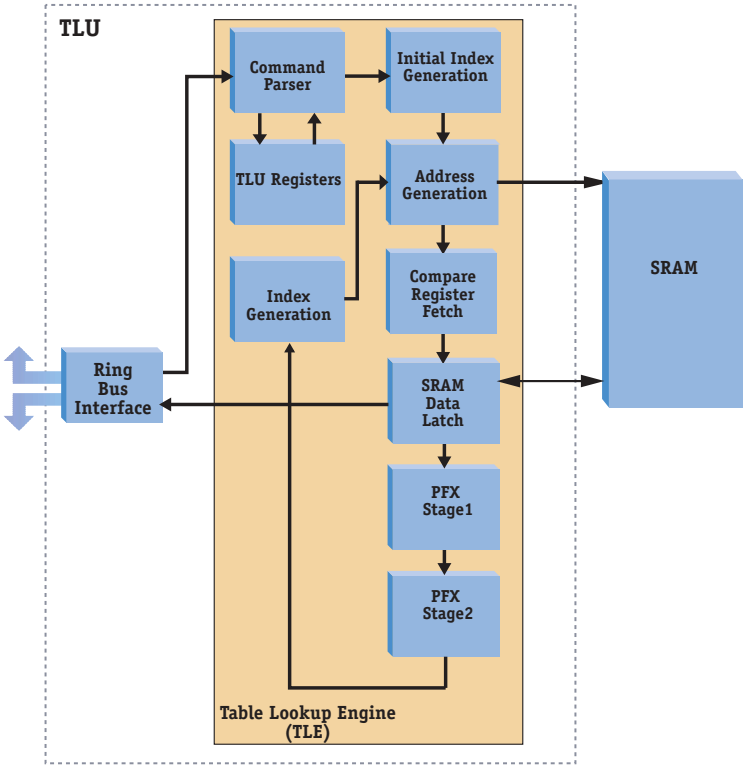
TLU Major Components

The major components of the TLU are listed in Table 92 on page 315. In addition, Figure 66 on page 316 shows the TLU Block Diagram.

Table 92 Major Components of the TLU and Their Functions

ITEM	FUNCTION
Table Lookup Engine (TLE)	Performs table lookups. The TLE comprises nine (9) blocks: command parser, TLU registers, initial index generation, address generation, compare register fetch, SRAM data latch, PFX stage1, PFX stage2 and index generation. In addition to these nine (9) blocks that comprise the TLE there are two (2) blocks that comprise the TLU: Ring Bus interface and TLU SRAM.
Ring Bus Interface	Connects the Table Lookup Engine (TLE) to the C-5e's internal Ring Bus.
Command Parser	Decodes the Table Lookup Engine (TLE) commands and reads/writes internal control registers.
TLU Registers	Contains all the internal TLU register values.
Initial Index Generation	Calculates *initial Index* based upon *table type* and lookup *key*.
Address Generation	Translates current index and table into an SRAM register.
Compare Register Fetch	Compares previous SRAM key with lookup key. Fetches data from register array.
SRAM Data Latch	Latches incoming data from SRAM read.
PFX Stage1	Calculates the next index based on encoded data from the SRAM fetch.
PFX Stage2	Calculates the next index based on encoded data from the SRAM fetch.
Index Generation	Generates the next index.
TLU SRAM	The TLU's SRAM internal Memory Controller is designed to maximize the bandwidth utilization of the SRAMs. The SRAM Memory Controller supports SRAM frequencies to 133MHz using 3.3V LVTTL., and physical interface SRAM technologies up to 64Mbits.

Figure 66 TLU Block Diagram



TLU Flow Process

The nine (9) blocks of the TLU allows the implementation of a variety of table lookup algorithms to meet different application needs. In general, its functional blocks are organized in a basic loop that performs the following functions:

1. Parses a Ring Bus command.
2. Calculates the initial index based on a Key (for example, the head of a Trie or an initial hash value).
3. Evaluates the current table node.
   a. Fetches SRAM data based on initial index.
   b. Fetches a portion of the Key
   c. Uses SRAM data, table format and key information to either go to step 4 or step 5.
4. Calculates a new index (and then go back to step 3).
5. Fetches the data at the current index.
6. Returns the data to the CPs or XP via the Ring Bus, or to the FP via a dedicated path between the TLU and FP.

Each block has several programmable, pipelined stages. Each stage, passes data to the next downstream stage. At any given time, every stage can have valid data, allowing many TLU operations to occur simultaneously.

The preceding TLU flow process applies specifically to Find, Findr and Findw operations. The flow process is different for the other operations. Some commands, such as read and some write commands only pass through the loop once.

TLU Flow Process Details

This section describes the transactional flow through the entire TLU in more detail that comprises all eleven (11) blocks. The blocks include: Ring Bus Interface, Command Parser, TLU Registers, Initial Index Generation, Address Generation, Compare Register Fetch, SRAM Data Latch, PFX Stage1, PFX Stage2, Index Generation and TLU SRAM.

Ring Bus Interface and Command Parser

The Ring Bus Interface block of the TLU is the only interface between the TLU and the rest of the C-5e NP. The Ring Bus comprises a receive (Rx) and a transmit (Tx) side.

- The Receive (Rx) section of the Ring Bus Interface monitors the Ring Bus for commands destined for the TLU. When a command is received (Rx), the command is removed from the Ring Bus and sent to the TLU's Command Parser block. TLU commands can be either an *indication* or a *request* message type.
  - If an indication is received, the TLU immediately places a confirmation message on the Ring Bus back to originating ring node. The purpose of the confirmation is to reserve a slot on the Ring Bus so that the originating node can place a new message on the Ring Bus. This indication-confirmation reservation technique is useful for guaranteeing bandwidth on the Ring Bus under high-load conditions. The Fabric Processor should always send indications to the TLU rather than requests, as it requires dedicated Ring Bus bandwidth. Refer to Table 103 on page 357 for list of the TLU commands, parameters, command ID's returned data and functions.
  - If a request is received, the TLU responds in a similar nature as an indication message type, except *no* confirmation is sent.
  - During normal operations the TLU should never be busy; however, if it is busy, then the TLU returns an error to the calling CP and terminates the request.
- The Transmit (Tx) side of the Ring Bus Interface is responsible for returning data to the requesting CPs or XP. Each Ring Bus slot message returns at most 32Bytes. If more than 32Bytes are requested, then multiple messages are transmitted. Since all Ring Bus responses are initiated by a specific request, the only command information passed back to the requesting node (CPs, or XP) is the sequence number. Refer to"Ring Bus Registers" on page 133, and "Ring Bus Overview" on page 477.

Responses sent to the TLU are discarded.

TLU Registers

Contains all the internal TLU register values.

Initial Index Generation

The Initial Index Generator block calculates the initial index into a table.

Address Generation

The Address Generation block calculates the SRAM address for the next SRAM access. An address is calculated as follows:

*SRAM address = (base_address * 256) + (index << entry size) + offset*

The TLU does not support the burst access feature of the SRAMs. Instead, it has an internal burst counter that automatically increments the address for consecutive reads and writes. For non-data tables, the entry size should be set to 0 (8Bytes).

Compare Register Fetch

The Compare Register Fetch block is responsible for comparing the last SRAM read data with the current key. Compares occur whenever a find type command (*Find*, *Findw*, or *Findr*) accesses a Key table. Successful compares allow *Findw* commands to execute and F*indr* commands to return TLU response data. PFX tables return the correct answer by design, therefore, the compare function is skipped for these table types.

Each node compare of a Trie or Hash table takes a SRAM cycle.

SRAM Data Latch

The SRAM Data Latch block latches the data from an SRAM read so that it can be processed by the next stage.

PFX Stage1 and PFX Stage2

The PFX algorithms requires two (2) pipe stages to decompress the code value from the previous read.

Index Generation

The Index Generator block incrementally calculates the next index into a table. Branch decisions are calculated by examining a number of different operands. These include: the current Key, fields from the last entry fetch operation, the current index, and a variety of internal registers. These internal registers are used to keep track of the current bit position for tries, pointers for most significant prefix matching, and other general record keeping functions. After the first calculation is performed, the Index Generation block calculates the next index for iterative functions.

The Index Generation block can be programmed on a per table basis using one of two (2) methods for generating a new index. The two (2) methods include:

- Incrementing of the previous index. This method is used *only* for SRAM burst access.
- Using a entry fetch operation. This method is used for all tries and hashes. For lists with multiple pointers (that is, tries), the previous stage passes control information to this block describing the correct pointer to choose.

TLU SRAM

The TLU's SRAM internal Memory Controller is designed to maximize the bandwidth utilization of the SRAMs. The SRAM Memory Controller supports SRAM frequencies to 133MHz using 3.3V LVTTL. The maximum amount of memory supported by the TLU is 128MBytes in four banks, when SRAM technology supports 4M x 18pins parts. The SRAM physical interface supports SRAM technologies up to 64Mbits. Refer to Table 93 on page 320.

Table 93 TLU SRAM Configurations

SRAM TECHNOLOGY	MINIMUM TABLE SIZE, 1 BANK	MAXIMUM TABLE SIZE, 4 BANKS
1Mb (32k x 32pins)	256kBytes	1MBytes
2Mb (64k x 32pins)	512KBytes	2MBytes
4Mb (256k x 18pins)	2MBytes	8MBytes
8Mb (512k x 18pins)	4MBytes	16MBytes
16Mbit (1M x 18pins)	8MBytes	32MBytes
32Mbit (2M x 18pins)	16MBytes	64MBytes
64Mb (4M x 18pins)	32MBytes	128MBytes

The TLU can perform a read or write operation every cycle using ZBT SRAMs. The physical interface provides four (4) CEx (chip enable) and WEx (write enable). Each bank (up to 4) gets its own CEx (CE0x, CE1x, CE2x or CE3x) and WEx (WE0x, WE1x, WE2x or WE3x).

The SRAM controller does not support bursting. Sequential accesses are generated using an internal address incrementer.

TLU Supported Table Types

Networking systems use synchronized tables containing topology and control information to make forwarding and characterization decisions. Such tables are accessed in the forwarding path for lookup resolution and forwarding decisions, and are typically managed by an agent that runs on an application processor that adds, removes, and modifies entries in these tables. Examples of such databases would be an IP Routing Table or an ATM Virtual Connection (VC) table.

The TLU hardware resolves table lookups by understanding four (4) different table types in hardware. With each table type the TLU understands a variety of possible table format transitions that can occur based on data software enters into the TLU SRAM. During a table lookup operation: the TLU receives a *Table Id* value that is used to read data from the TLU configuration register that tell the TLU what the hardware T*able Type* is, the expected *Initial Data Format*, and how to use the *Key* to get to the first SRAM entry. The API software currently uses the available data format transitions to create six (6) software algorithms.

Software algorithms fall into two (2) main categories: exact-match, and longest-prefix match. Exact-match algorithms perform some sort of memory node structure traversal and then uses data stored in memory to check against the key. In an exact-match algorithm, a lookup returns data successfully *only* if the key matches. In contrast, in a longest-prefix match algorithm, the TLU compares parts of the key to values stored in memory and when it can no longer match the key, it returns data. Table 94 on page 321 lists the software algorithms, hardware table types and their functions.

The TLU supports four (4) different Key sizes: 32, 48, 96 and 112bit. In addition, intermediate key sizes are supported by masking unused bits to zero.

Warning: *Ensure all table memory areas are initialized before performing table inserts or lookups. Possible unreported errors providing erroneous data could occur.*

Table 94 Supported Table Types (Software Algorithms and Hardware)

SOFTWARE ALGORITHMS	HARDWARE TABLE TYPES (INITIAL FORMAT)	FUNCTION
Hash-Trie-Key	Hash	Used for *exact match* algorithms. That is, algorithms that require a lookup key to be matched exactly.
Chained Hash	PFX (Longest-Prefix Match)	A special case of a PFX table where a portion of the key is used for an initial index lookup. The results of the indexed lookup is combined along with the hashed initial key to create a second hashed lookup.

Table 94 Supported Table Types (Software Algorithms and Hardware) (continued)

SOFTWARE ALGORITHMS	HARDWARE TABLE TYPES (INITIAL FORMAT)	FUNCTION
Chained Index	PFX (Longest-Prefix Match)	A portion of the key is used for an index lookup. The results of the indexed lookup is combined with part of the initial key to create a second index lookup.
PFX (Longest-Prefix Match)	PFX (Longest-Prefix Match)	Used for Longest-Prefix Matching searches for IPv4 and TCP/IP routing.
Flat Data Table	Data	Contain the data associated with an entry. Data tables can be used as a stand-alone table or as the last table (that contains the associated data for a previously evaluated key) in a set of tables.
External	External	Used to interface with external lookup engines when it is swapped with a SRAM bank.

For every table Id there is an associated configuration register. The TLU supports thirty-two (32) tables. Tables are numbered from (0 to 31). There are thirty-two (32) configuration registers (numbered from (0 to31)) and thirty-two (32) virtual table registers (also numbered from (0 to31)). Refer to "TLU Configuration and Status Registers" on page 383.

By programming the virtual table registers, a received table Id value is converted from the virtual table Id to the physical table Id which maps directly to the configuration register number to use. Within the configuration register for each physical table Id value is a base table address configured by software. Also, in the configuration registers is the initial table format to expect, the method of generating the initial index, and how an index value maps to a physical memory address. This means all table entries can be accessed by an index value, and individual bytes by an offset value. The entries can be 8Bytes to 1024Bytes in length.

Implementation Considerations

Prior to implementation consider the following:

- The XOR command can only be used in tables 0 to 7.
- Only Data tables should use more than 8Bytes (1 slot) per entry. Refer to bits [23:21] in the "Table_Configuration1 Register" on page 389.

TLU Operation Overview

In general, the TLU performs a lookup using the nine (9) steps listed in Table 95 on page 323.

Table 95 TLU General Operation Step/Action Table

STEP	ACTION
1	TLU parses the Ring Bus message. • Extracts all of the information it needs to perform it's functions from the Ring Bus data: • Type of command (FIND, FINDR, FINDW). • Table ID and key value. What is expected back and how: FIND returns an address like token to where the data is stored; FINDR returns data of a volume and offset into the data entry specified in the Ring Bus message; FINDW uses the volume and offset information to do a write to the data entry instead of a read. • Ring Bus information contains which node to return the data to, and the response slot to use.
2	TLU maps Virtual Tables to Physical Tables. The TLU uses the Table ID number from the table lookup Ring Bus message as a virtual table ID number and checks the virtual table ID registers to find the physical table ID that maps to the virtual table ID.
3	TLU reads the configuration registers. The TLU uses the physical table ID and it's associated configuration registers to determine: • Initial data format to expect. • Base address of the table. • How table addressing works for this table.
4	TLU uses the current format as a state and computes: • Next index to read= function of (key value, key size, data format, configuration registers)
5	TLU converts the index to an address using: • Address= (base address memory page number * 256) + next index * size of entry programmed in configuration register). NOTE: This occurs while format/state is *not* data.
6	TLU reads the SRAM data.
7	TLU computes the next index to read using: • Next index to read= function of (key value, SRAM data, current format).

Table 95 TLU General Operation Step/Action Table (continued)

STEP	ACTION
8	TLU computes next format based on: key value, key size, current format, and SRAM data.
9	TLU performs the required FIND response: • FIND, returns current index value. • FINDR, read from address + offset in Ring Bus message and returns that data. • FINDW, write the data sent in the Ring Bus message. NOTE: This occurs when format/state *is* data.

TLU Operation Details

At the core of the TLU's operation is a state machine. The state machine uses a key, and the data contents of the initial node to determine which state to expect. It can determine which data format to expect next; therefore enabling it to generate the next node address. The TLU operates in the following manner: reads the initial data format, checks SRAM for the format data to act upon, generates the next address, traverses the table to get to the associated data for a given key.

The seven (7) allowed TLU state transitions are listed in Table 96 on page 324.

Table 96 TLU Allowed State Transitions

TRANSITION STATES
Hash -> Trie
Hash -> Key
Trie -> Trie
Trie -> Key
PFX -> PFX
PFX -> Hash
PFX -> Data

Table 97 on page 325 lists the applicable TLU states as they relate to the six (6) algorithms in software, lists the applicable hardware table types (initial format) as they relate to the software algorithms, and lists the encoded values as they relate to the applicable hardware table types (initial format).

The hardware table type is selected using the Table_Configuration1 register bits [31:28] Type field which uses encoded values.

Table 97 Relationships of TLU States, Software Algorithms, Hardware Table Types, and Encoded Values

TRANSACTION STATES	SOFTWARE ALGORITHMS	HARDWARE TABLE TYPES (INITIAL FORMAT)	ENCODED VALUES (TABLE TYPE)
• Hash -> Trie • Trie -> Trie • Trie -> Key	Hash-Trie-Key	Hash	3
• PFX -> PFX • PFX -> Data • PFX -> Hash • Hash -> Trie • Trie -> Trie • Trie -> Key	Chained Hash	PFX	1
• PFX -> PFX • PFX -> Data • PFX -> Hash • Hash -> Trie • Trie -> Trie • Trie -> Key	Chained Index	PFX	1
• PFX -> PFX • PFX -> Data • PFX -> Hash • Hash -> Trie • Trie -> Trie • Trie -> Key	PFX	PFX	1
N/A	Flat Data Table	Data	0
N/A	External	External	6

TLU Operation Example

As an example, if you program the configuration registers using the following parameters: table type=3 (Hash table type, initial format of Hash), bit depth for the hash =5, and a base address for the table. Then the TLU would take these inputs and then read in data from SRAM, interpret the data and transverse the table.

When the initial format is hash, the TLU performs a hash function over the entire key bits, takes the lowest few bits (in this particular example the least significant 5) and finds the first node (a node is a set of data stored in a table).

By reading the node, and knowing it's format the TLU knows if the next node is a key leaf node *or* a trie node. When it is a trie node, the TLU understands the trie node's format and traverses the table knowing when to transition from trie format to key format (Trie -> Key) automatically.

Once arriving at a key node, the TLU reads the key stored in SRAM and compares it to the key being looked up, and upon a match returns the requested data. The TLU reacts to what ever you stored in SRAM to determine it's ultimate algorithm and behavior. The API provides an implementation of a very common software algorithm called Hash-Trie-Key that takes a key and payload data as insert/delete information and manages a table for you.

Software Algorithms

This section describes each of the six (6) software algorithms: Hash-Trie-Key, Chained Hash, Chained Index, PFX (Longest-Prefix Match), Flat Data Table, and External. Where applicable, the transition states, recommended memory organization (conceptually), block diagram, data format, and example is provided for each of the software algorithms.

Hash-Trie-Key

Figure 67 on page 327 shows the Hash->Trie->Key state transitions and Table 98 on page 327 lists their details.

Figure 67 Hash -> Trie -> Key State Transition Diagram

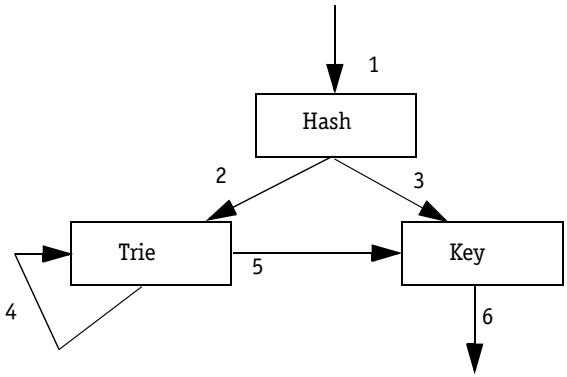


Table 98 Hash -> Trie -> Key State Transition Details

STEP	ACTION
1	Initial state is set to Hash based on the configuration registers. In the hash state, the TLU performs a memory read, then transitions to either the Trie *or* Key format states. Specifically, the TLU: • Computes the hash value based on the hash bit depth in the configuration register, and the key. C-5e hash entries are stored two to a 64bit memory slot. The low order bit of the hash value determines: if the upper 32bits (lowest bit=0) *or* if the lower 32bits (lowest bit=1) is to be used. The remaining bits are added to the Base Address of the table to compute which SRAM 64bit memory address to read from. • Parses the hash formatted data. Bit Position 31 30 24 23 0 Field Name Flag Count Link

Table 98 Hash -> Trie -> Key State Transition Details (continued)

STEP	ACTION
2	When the Flag value is set to zero (0), the next state is a Trie. The TLU reads in the 64bit data slot associated with the Link value, and uses the bit in the key associated with the Count value to determine: if the left (bit=0) high order 32bits of the Trie data is to be used *or* if the right (bit=1) low order 32bits of the Trie data is to be used. Trie entries are formatted as shown here: Upper 32=left=testbit=0 Lower 32=right=testbit=1 Bit Position 63 62 56 55 32 31 30 24 23 0 Field Name: **Flag** | **Count** | **Link** | **Flag** | **Count** | **Link** By going left or right, either the left Flag bit [63] or the right Flag bit [31] is examined to determine the next state. If the bit is a one (1), the TLU transitions to the Key state. Otherwise it remains in it's current state.
3	The TLU can transition from the Hash or Trie states to Key, if the Flag bit is set to one (1) in the Hash or Trie formatted data.
4	Once it enters the Trie state, it remains in the Trie state until the TLU detects that the Flag bit is set to one (1).
5	The TLU can transition from the Hash or Trie states to Key, if the Flag bit is set to one (1) in the Hash or Trie formatted data.
6	Once the TLU enters the Key state, the TLU treats the Key and Data states almost as equivalent. The TLU reads the Key data stored in the entry and matches the Key in the Ring Bus message against the Key in the entry. When they match the lookup is successful, and when they do *not* match the lookup fails.

Software uses TLU write commands to initialize the SRAM values. Then these SRAM values are read to determine the Hash, Trie, and Key state transitions. Software can set flag, counts, and links to make the table perform a variety of algorithms. The hardware is designed and tested against the API table services Hash-Trie-Key algorithm. Furthermore, the same hardware is also used in part for the Chained Hash software algorithm by setting the flags, counts, and links differently. Therefore, software uses hardware to implement two (2) different algorithms.

Refer to Figure 68 on page 329 for the Hash-Trie-Key recommended memory organization.

Figure 68 on page 329 shows how memory is laid out conceptually by the API. This is not hardware imposed, but a recommended map that can be useful to explain how the hardware works conceptually.

Figure 68 Hash-Trie-Key Recommended Memory Organization (Conceptually)

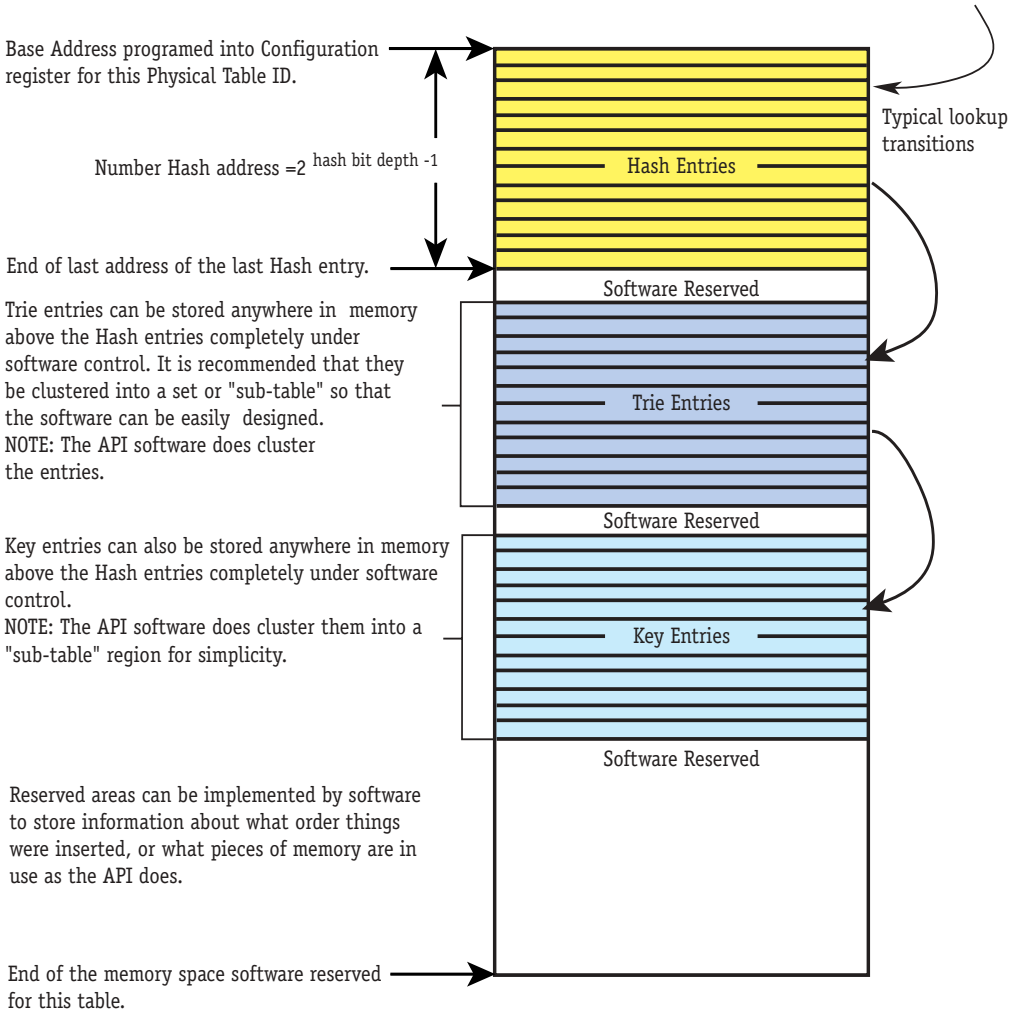


Figure 68 on page 329 shows the recommended memory organization for Hash-Trie-Key software algorithm. It basically uses a clustered set or "sub-table". These three (3) sub-tables as detailed here.

Hash Sub-Table

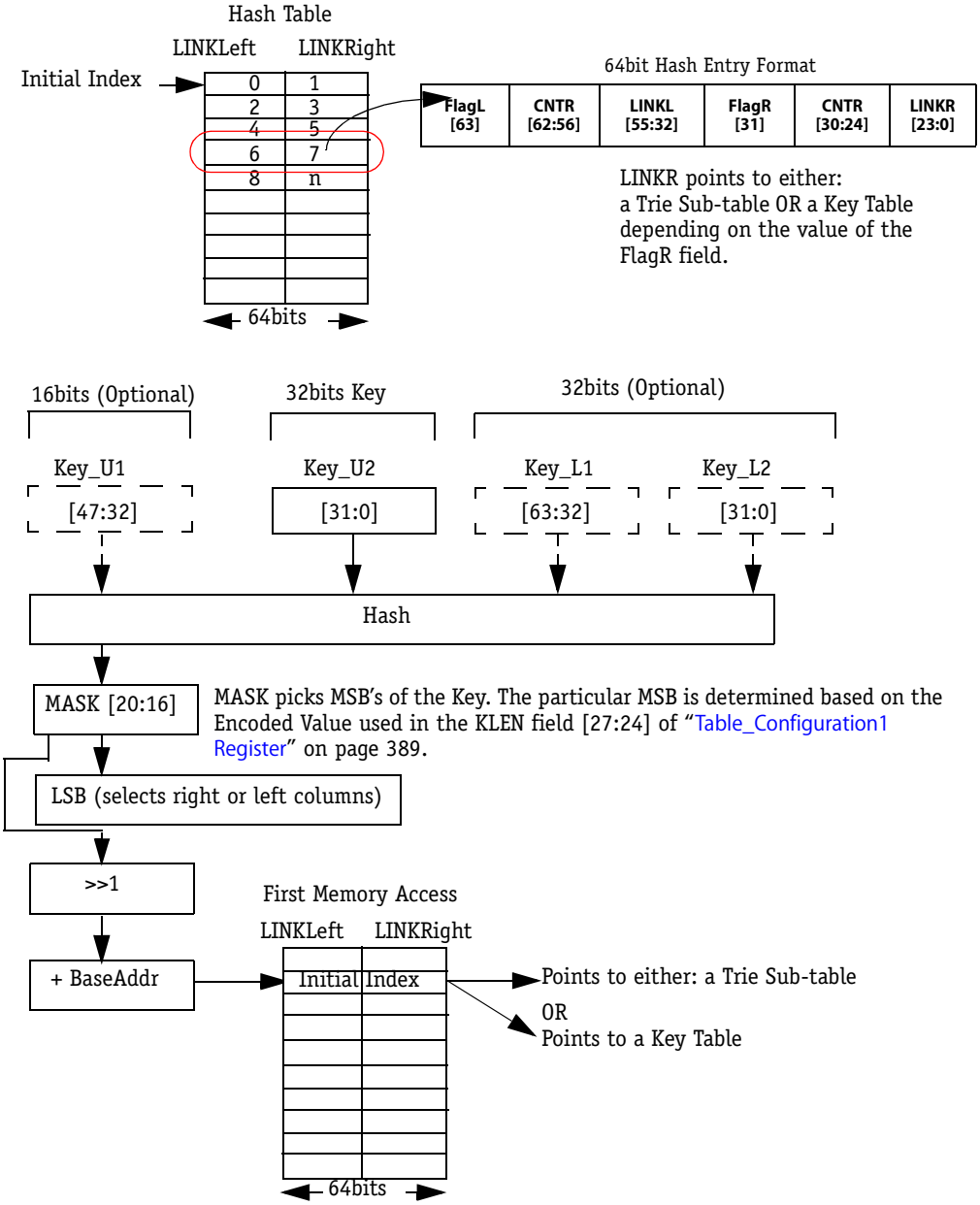
The hash function is a fixed function that produces a hash index of the lookup key. Typically, the hash function exhibits good randomness such that changing one (1) bit in a key causes approximately half of the bits (in the hash index) to change as a result.

Hash tables are used for *exact match* algorithms. That is, algorithms that require a lookup key to be matched exactly. Hash tables evaluate the lookup key via a *hash function*. The hash function returns an index into the Hash table that contains an entry that points to another entry in another table. The table entry *pointed to* by the Hash table can be either:

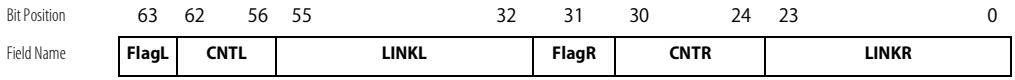
- An entry in a Trie table (used for collision resolution).
- A key to be used for comparison in a key table.

You can calculate the number of collisions using a Chi^2 distribution. Thus, the probability of a collision based on random data is: # real entries/ # buckets. For example, if you have 64k real 32bit keys and hash to a 18bit index, your hash table would have 256k buckets. Dividing the 64k entries by 256k indicates that there would be an average of one (1) collision for every four (4) entries. At most one (1) bucket would have eight (8) entries, and all of the other buckets would have something less than eight (8). Refer to Figure 69 on page 331.

Figure 69 Hash Sub-Table Block Diagram



Hash Sub-Table Data Entry Format



FIELD NAME	BIT POSITION	DESCRIPTION
FlagL	63	**Flag Left** — A 1= points to the key entry, a 0= points to a trie entry.
CNTL	62:56	**Count Left** — The total number of bits that match traversing down the branch.
LINKL	55:32	**Link Left** — If Flag [63]= 0 index of associated data, or if Flag [63]= 1 index to an element in the next table in the algorithm.
FlagR	31	**Flag Right** — A 1= points to the key entry, a 0= points to a trie entry.
CNTR	30:24	**Count Right** — The total number of bits that match traversing down the branch.
LINKR	23:0	**Link Right** — If Flag [31]= 0 index of associated data, or if Flag [31]= 1 index to an element in the next table in the algorithm.

The TLU format uses two (2) hash entries per 64bits. Therefore, a 64bit word # = hash value/2. Consequently, a hash value/2 = 1 slot and the upper 32bits are used if it is a *even* (left) hash value or the lower 32bits are used if it is a *odd* (right) hash value.

Hash Sub-Table Example

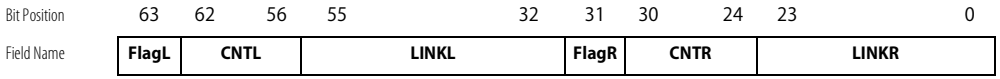
An example of a Hash table is an 802.1D Bridge Forwarding Table. Collisions can be resolved by having entries that collide point to a Trie table. Ultimately, the entry key and associated data are stored in a data table.

Trie Sub-Table

Trie tables are binary skip trees that provide an efficient means for resolving Hash table collisions. A benefit of using Trie tables for collision resolution in conjunction with a Hash table is that for *n* colliding keys that collide in the table, usually only $\log_2(n)$ entries need to be checked to resolve the collisions. For collision resolution, only the bits that differ in the collided keys are checked. Leaf nodes point to a Key table entry that does a complete key match of the input key. The size of each node of this table structure is 8Bytes or one (1) SRAM location.

Trie table entry resolves 1bit of collision at a time. As the table is traversed, the count fields in the previous node are used to specify which bit of the key is to be evaluated in the current node. The value (0 or 1) of the bit being evaluated indicates whether to take the left branch or the right branch (0=left branch, 1=right branch) of the tree.

Trie Sub-Table Data Entry Format



FIELD NAME	BIT POSITION	DESCRIPTION
FlagL	63	**Chain Flag Left** — A 1= next node is a key, a 0= next node is a trie.
CNTL	62:56	**Count Left** — Specifies the position of the bit in the key that is used to determine whether to take the left or right branch in the next left branch node. If the CNTL value is a non-zero, then the CNTL value refers to an individual bit in the Key, and if the corresponding bit value is 0 take left branch and if the bit is 1 take right branch.
LINKL	55:32	**Link Left** — If FlagL [63]= 0 index to the left branch of the Trie, or if FlagL [63]= 1 to the next table.
FlagR	31	**Chain Flag Right** — A 1= next node is a key, a 0= next node is a trie.
CNTR	30:24	**Count Right** — Specifies the position of the bit in the key that is used to determine whether to take the left or right branch in the next right branch node. If the CNTR value is a non-zero, then the CNTR value refers to an individual bit in the Key, and if the corresponding bit value is 0 take left branch and if the bit is 1 take right branch.
LINKR	23:0	**Link Right** — If FlagR [31]= 0 index to the right branch of the Trie, or If FlagR [31]= 1 the next table.

Trie Sub-Table Example

The use of the *count* (CNT) field is shown in Figure 70 on page 334. The three (3) in the Hash table's CNT field means that the third MSB of the three (3) collision entries is to be used when evaluating the lookup key in the next "node". The *index* value of the entry in the Hash table points the next entry.

Specifically in this example, the index value points to the "root" node in a new table (a Trie table) to be used for collision resolution.

Examining the third MSB of the collided values shows that one entry has a bit value of zero (0) and the other two entries have bit values of one (1). In other words, the value of 0, 1, and 1. This value (0, 1) has the following meaning:

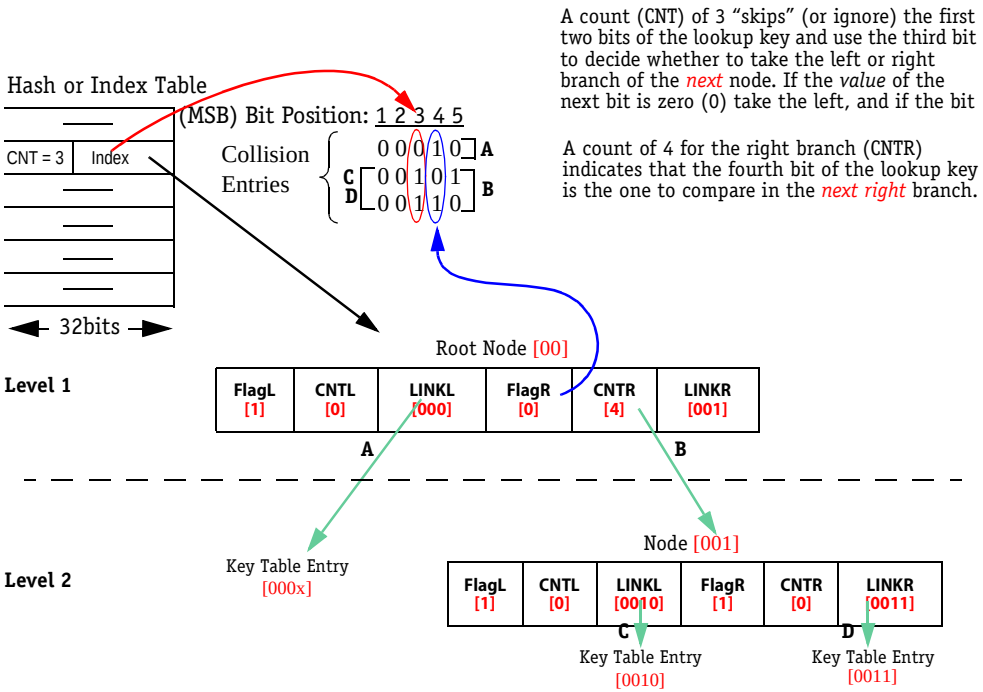
- The zero (0) value indicates to take the *left* branch from the root node.
- The one (1) value indicates to take the *right* branch from the root node.

Since there is only one entry with the third MSB equal to zero (0), the next left branch node (level 2) in the Trie table can point to the Key table entry that contains the data.

In general, the trie rules are:

- A Flag of 1 means that the next link is a key.
- A Flag of 0 and CNT != 0 means that the next link is a trie.

Figure 70 Trie Sub-Table Showing Skip Function



However, there are two (2) “collision” entries with a value of (1) for their third bit (00101 and 00110). Thus, the program must move to a lower level in the tree to obtain the correct indexes (to Key table entries) for each lookup key. Notice that the root node’s *CNTR* (count right) field is set to four (4). This means that in the next right node (level 2), the fourth bit of the lookup key should be evaluated. Again, a bit value of (0) indicates branch left and a value of (1) indicates branch right, (0=left branch, 1=right branch).

Since there are only two (2) remaining collision entries and their fourth bits are different (one is a 0 and the other is a 1), both the left and right nodes will match the first four bits of one of the two unresolved entries and since the CNT is zero (0), each link points to the correct index into the associated Key table.

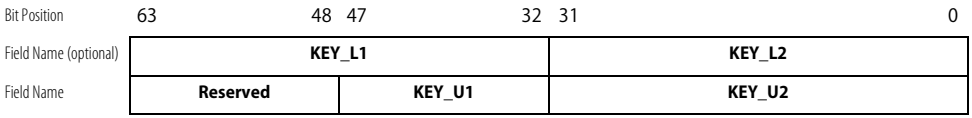
The exact match on the lookup key takes place in the Key table. The Trie is used to resolve search key values that collided in the Hash table. Thus, the depth of the Trie table needs to be only deep enough to distinguish two (2) or more keys (depending on the number of collisions) from each other.

The *CNTL* (count left) and *CNTR* (count right) fields also allow you to “ignore or skip” one (1) or more bits in a lookup key and thus omit what would be the corresponding nodes on a branch of the tree. Thus, if the values for a number of bits in the Trie entries are the same across a number of entries, the TLU can perform an “exact” match without traversing a level of the Trie for every bit in the lookup key.

Key Sub-Table

Key table are used in *exact match* algorithm, and contain both the key of an entry and the associated data. The last step of an exact match algorithm compares the lookup key with the key from the entry in the Key sub-table. The TLU supports key sizes of 32, 48, 96, and 112bits. The associated data portion of the entry in a Key sub-table is typically returned to the requesting node (CP or XP) via the Ring Bus.

Key Sub-Table Data Entry Format



FIELD NAME	BIT POSITION	DESCRIPTION
KEY_L1*	63:32	**Key Lower 1** — Contains the lower middle portion for 112bit and 96bit keys
KEY_L2*	31:0	**Key Lower 2** — Contains the LSB for 112bit and 96bit keys.
Reserved	63:48	Read as zero.
KEY_U1*	47:32	**Key Upper 1** — The MSB of 48bit and 112bit keys.
KEY_U2*	31:0	**Key Upper 2** — The MSB of 32bit and 96bit keys. Used also for the upper middle portion of 112bit and 48bit keys.

[1] For key sizes up to 48 bits, only one entry is used (KEY_U1 & KEY_U2). For larger key sizes, both fields are used.

A Key Data structure occupies the first one (1) or two (2) entries of a data entry. It is used by the TLU to verify that a node's data matches the search key. It is typically the termination of Trie and Hash tables. For key sizes up to 48bits, only one (1) entry is used. For larger key sizes, both fields are used.

Key Sub-Table Example

An example of an exact match algorithm that would be implemented using the TLU is an 802.1D forwarding table for layer 2 bridging applications. In this application, you would use a Hash table for the initial index evaluation, a Trie table to resolve collisions (if necessary), and a Key table to store both the key and associated data.

Chained Hash

Figure 71 on page 337shows the Chained Hash Key state transitions and Table 99 on page 337 lists their details.

Figure 71 Chained Hash State Transition Diagram

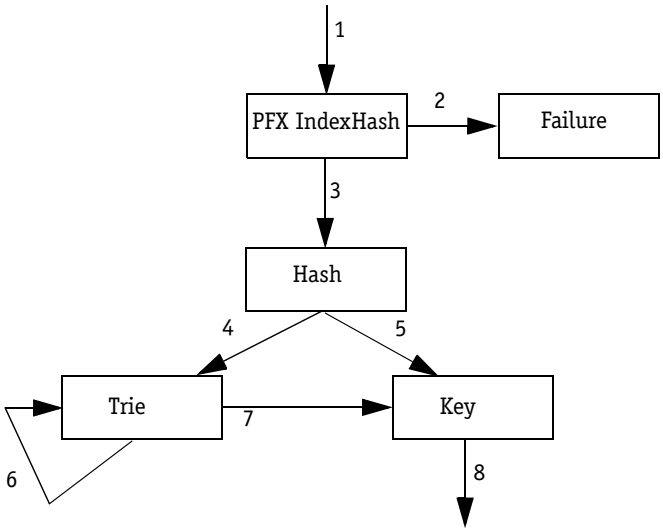
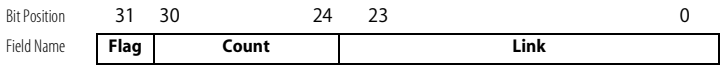
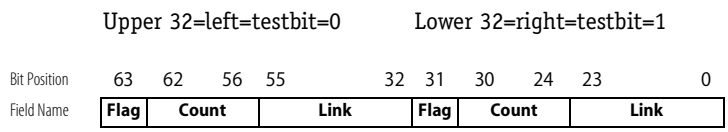


Table 99 Chained Hash State Transition Details

STEP	ACTION
1	Initial state is set to PFX based on the configuration registers. In the PFX state, the TLU performs a memory read, then transitions to either the Hash *or* Failure format states. Specifically, the TLU: • Computes the first address to read. The first address is the first N bits plus the Base Address of the table where N is the bit depth programed into the configuration register. hash value based on the hash bit depth in the configuration register, and the key. C-5e hash entries are stored two to a 64bit memory slot. The low order bit of the hash value determines: if the upper 32bits (lowest bit=0) *or* if the lower 32bits (lowest bit=1) is to be used. The remaining bits are added to the Base Address of the table to compute which SRAM 64bit memory address to read from. • Parses the PFX index formatted data. On reading a PFX entry, there are a few bits that determine the type and the link address. If the type= 0x2 the lookup has no corresponding hash table to transition to, so the TLU lookup fails. If the type=3 then the link points to the start of a hash table, and the TLU continues as if beginning a Hash-Trie-Key lookup and the hash size, and start of the hash table are programmed into the PFX entry.

Table 99 Chained Hash State Transition Details (continued)

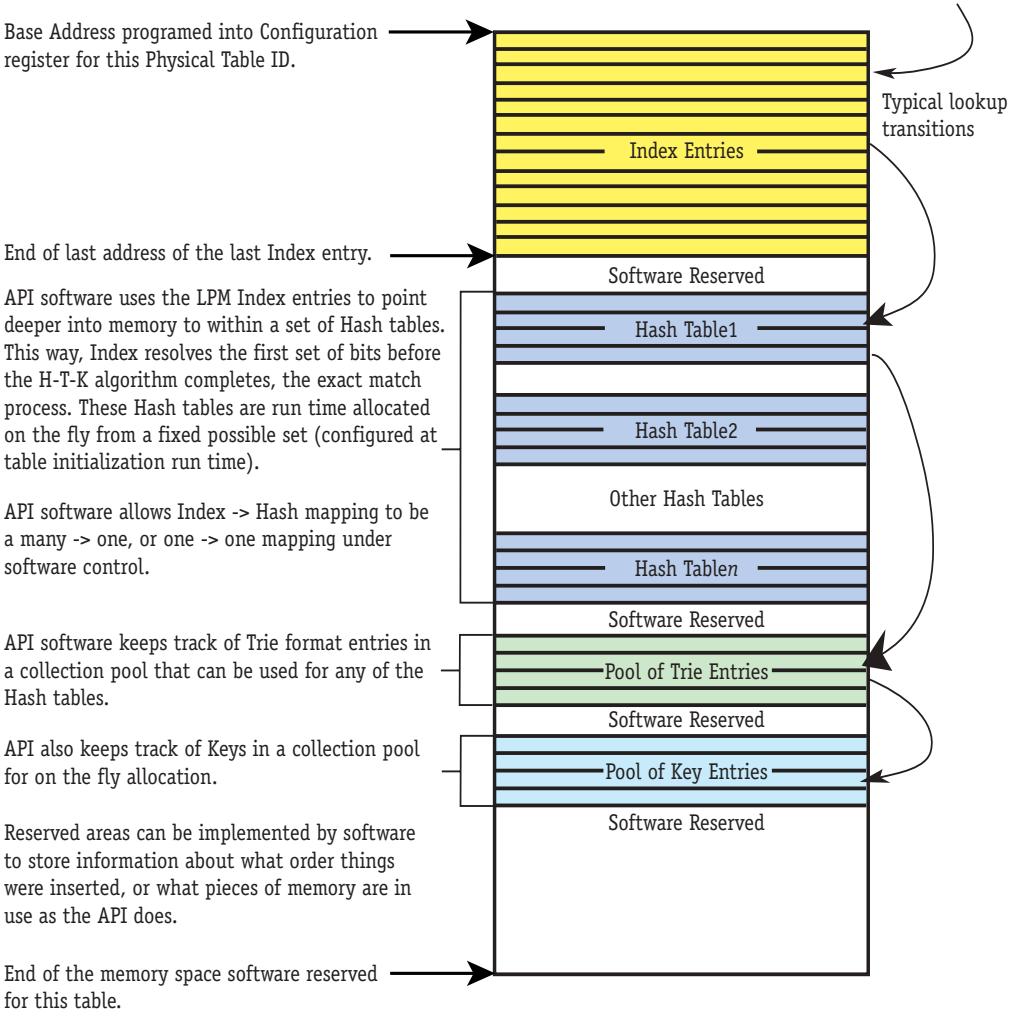
STEP	ACTION
2	If the Type bits in the PFX entry = 0x2, then the index tells the TLU to return a lookup miss condition.
3	Once the TLU enters the Hash state, the TLU performs a memory read, then transitions to either the Trie or Key format states. Specifically, the TLU: • Computes the Hash value based on hash bit depth contained in the LPPM index entry and the Key. C-5e hash entries are stored two to a 64bit memory slot. The low order bit of the hash value determines: if the upper 32bits (lowest bit=0) *or* if the lower 32bits (lowest bit=1) is to be used. The remaining bits are added to the Base Address of the table to compute which SRAM 64bit memory address to read from. • Parses the hash formatted data. 
4	When the Flag value is set to zero (0), the next state is a Trie. The TLU reads in the 64bit data slot associated with the Link value, and uses the bit in the key associated with the Count value to determine: if the left (bit=0) high order 32bits of the Trie data is to be used *or* if the right (bit=1) low order 32bits of the Trie data is to be used. Trie entries are formatted as shown here:  By going left or right, either the left Flag bit [63] or the right Flag bit [31] is examined to determine the next state. If the bit is a one (1), the TLU transitions to the Key state. Otherwise it remains in it's current state.
5	The TLU can transition from the Hash or Trie states to Key, if the Flag bit is set to one (1) in the Hash or Trie formatted data.
6	Once it enters the Trie state, it remains in the Trie state until the TLU detects that the Flag bit is set to one (1).
7	The TLU can transition from the Hash or Trie states to Key, if the Flag bit is set to one (1) in the Hash or Trie formatted data.
8	Once the TLU enters the Key state, the TLU treats the Key and Data states almost as equivalent. The TLU reads the Key data stored in the entry and matches the Key in the Ring Bus message against the Key in the entry. When they match the lookup is successful, and when they do *not* match the lookup fails.

Software uses TLU write commands to set the SRAM values. Then these SRAM values are read to determine the Hash, Trie, and Key state transitions. Software can set flag, counts, and links to make the table perform a variety of algorithms. The hardware is designed and tested against the API table services Hash-Trie-Key algorithm. Furthermore, the same hardware is also used in part for the Chained Hash software algorithm by setting the flags, counts, and links differently. Therefore, software uses hardware to implement two (2) different algorithms.

Refer to Figure 72 on page 340 for the Chained Hash recommended memory organization.

Figure 72 on page 340 *shows how memory is laid out conceptually by the API. This is not hardware imposed, but a recommended map that can be useful to explain how the hardware works conceptually.*

Figure 72 Chained Hash Recommended Memory Organization (Conceptually)



Chained hash table types supports the table topology imposed by IEEE 802.1Q VLANs (Virtual Bridged LAN, a subset of Ethernet protocols). Refer to Figure 73 on page 341.

The Ethernet frame, (Ethernet MAC), shown here, contains a 12bit identifier called a Virtual Local Area Network (VLAN) that maps in a many to one fashion to a Forwarding Identifier (FID). Therefore, the FID should be a part of the bit pattern used to perform an exact lookup along with other routing data in the Ethernet frame's header.

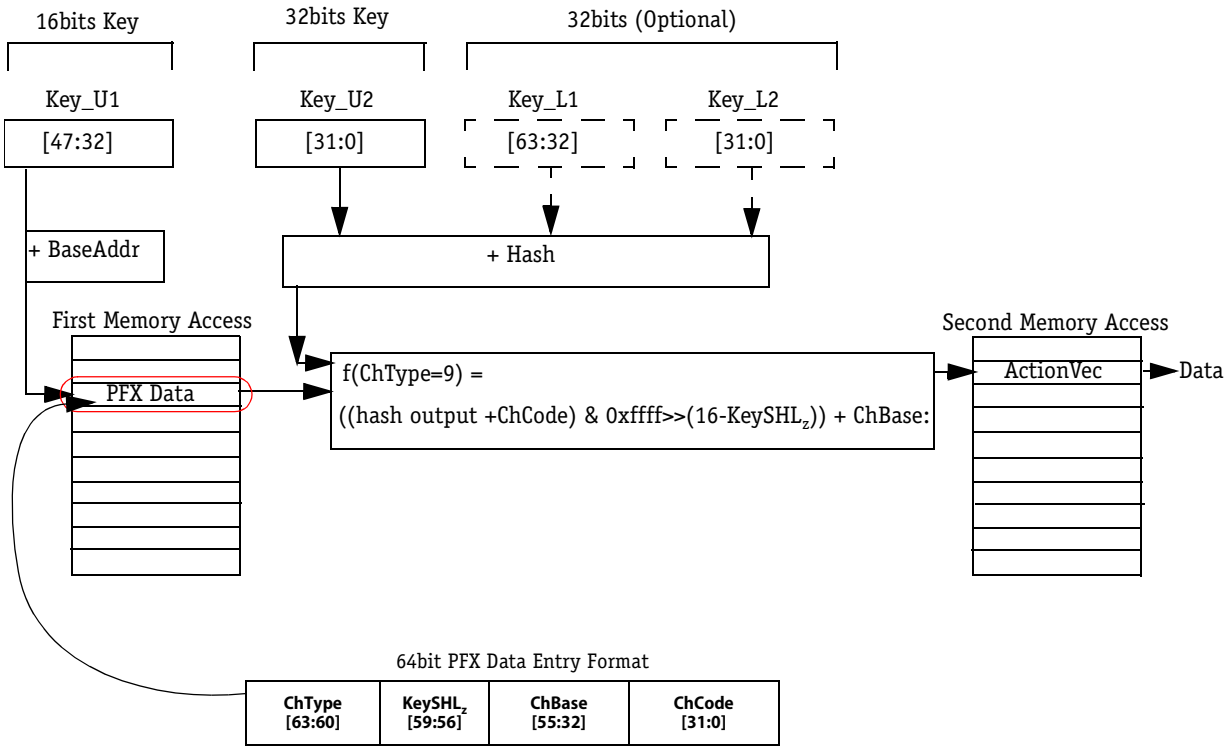
The Ethernet Media Access Control (MAC) frame has the following format:

Number of Bits	48	48	16	1	3	12	=>44
Field Name	MAC DA	MAC SA	Ethernet Type	CFI	Priority	VLAN	Payload

Using the chained hash key format allows one lookup to find any required data within the stored entry data. The entry data could consist of the FID, the egress port, which port the address was learned from and other data. Refer to Figure 74 on page 342.

Chained Hash tables only support 48bit or 112bit keys.

Figure 73 Chained Hash Block Diagram

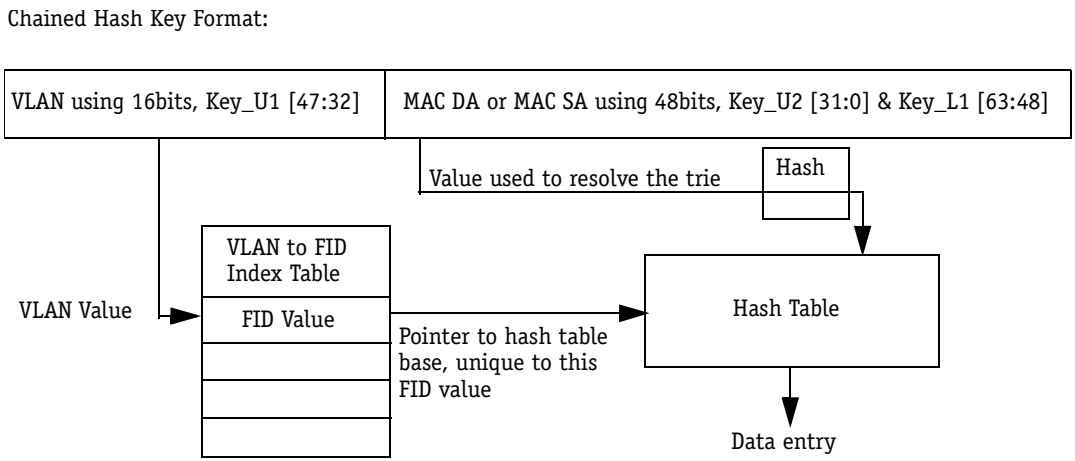


Chained Hash Data Entry Format

The chain hash data entry format uses the same format as PFX (Longest-Prefix Match). Refer to "PFX (Longest-Prefix Match) Data Entry Format" on page 352.

Chained Hash Example

Figure 74 Chained Hash Ethernet Application Example



NOTE: Key_U1 [47:32] and Key_U2 [31:0] should be set to 0.

The application should be configured in the following manner to support IEEE 802.1Q VLANs (Virtual Bridged LAN):

The key can consist of the VLAN and either MAC DA or MAC SA and other data. The application uses a 96bit or greater key. This application performs separate lookups for learning and egress port retrieval, but each lookup would be fast and not require VLAN to FID conversion. This particular application is best for applications that require egress data regularly, but use address port learning asynchronously.

Chained Index

Figure 75 on page 343 shows the Chained Index state transitions and Table 100 on page 343lists their details.

Figure 75 Chained Index Transition States

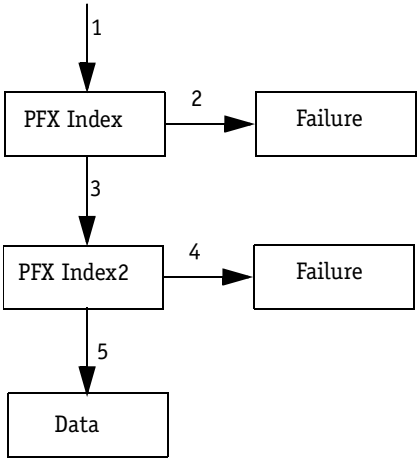


Table 100 Chained Index State Transition Details

STEP	ACTION
1	Initial state is set to PFX based on the configuration registers. The configuration registers hold a number that tells the TLU how many of the first few bits of the Key passed to it (up to 16) to use to form the First Index Value. This value is then added to the Base Table Address by the TLU, and the TLU reads the data slot (a 64bit word) from that location. The data it reads is in the following format: Bit Position 63 60 59 56 31 0 Field Name: **Type** | **BaseAddress** | **Unused** This is the format for a PFX Index entry. The TLU reads this information (previously initialized and/or written to by software. The Type is either Index, or Fail. If the type is Fail it transitions to the Fail state, otherwise it transitions to the Index state. If the type is not set to Index, or Fail, the TLU reacts in a way inconsistent with this algorithm. The TLU interprets it as a PFX type, and continues to perform a TLU lookup based on PFX information.
2	If the Type is Fail, then *no* data is returned by the TLU.

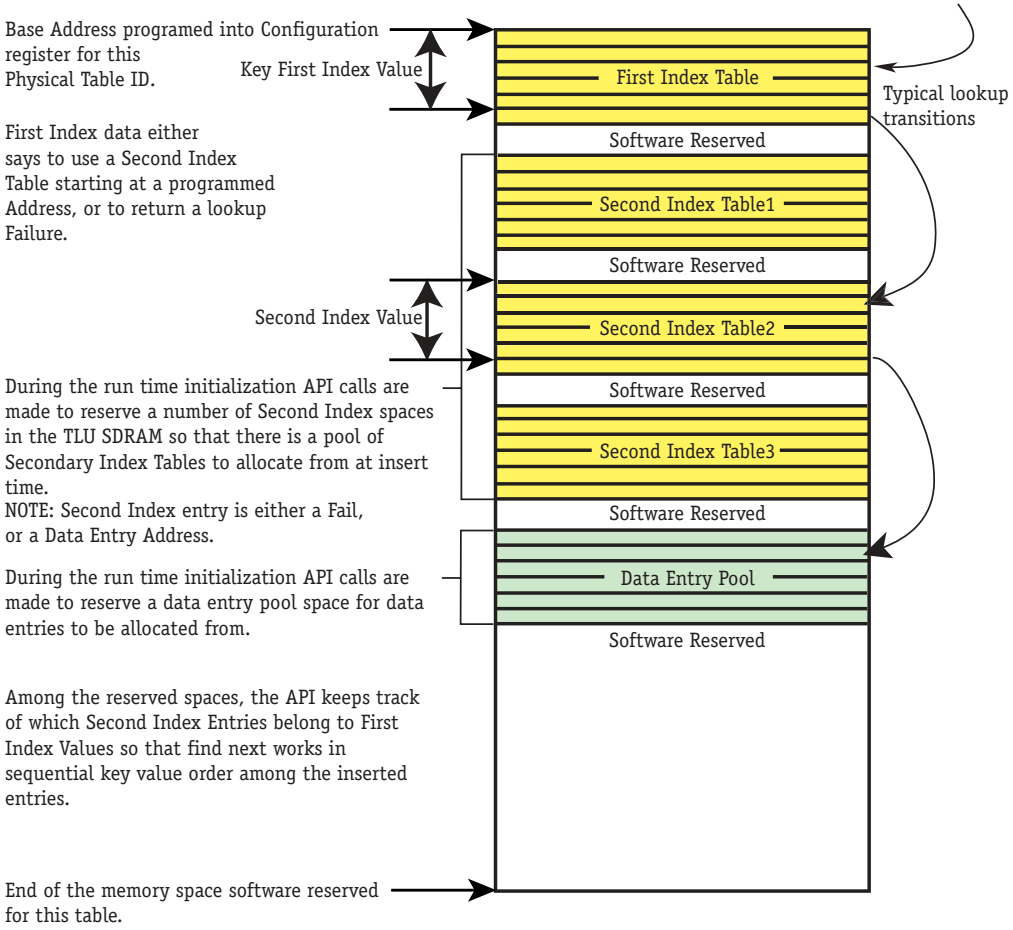
Table 100 Chained Index State Transition Details (continued)

STEP	ACTION
3	If the Type is Index, then the TLU uses the value of Size that it just read from the Index TLU entry, and uses the next size+1 bits to form a Second Index Value. It then takes this Second Index Value, and adds it to the Base Address stored in the entry it just read to compute a new memory address, and then reads that entry. The TLU hardware then reads this new PFX and interprets it the same way as in step 1, but software should configure it to have a Type of either: a pointer to Data, or Fail. If the Type is a pointer to Data, then the Base Address value is interpreted as a data pointer.
4	If the Type is Fail, then *no* data is returned by the TLU.
5	If the Type is Data, then the data is returned by the TLU.

Refer to Figure 76 on page 345 for the Chained Index recommended memory organization.

Figure 76 on page 345 *shows how memory is laid out conceptually by the API. This is not hardware imposed, but a recommended map that can be useful to explain how the hardware works conceptually.*

Figure 76 Chained Index Recommended Memory Organization (Conceptually)



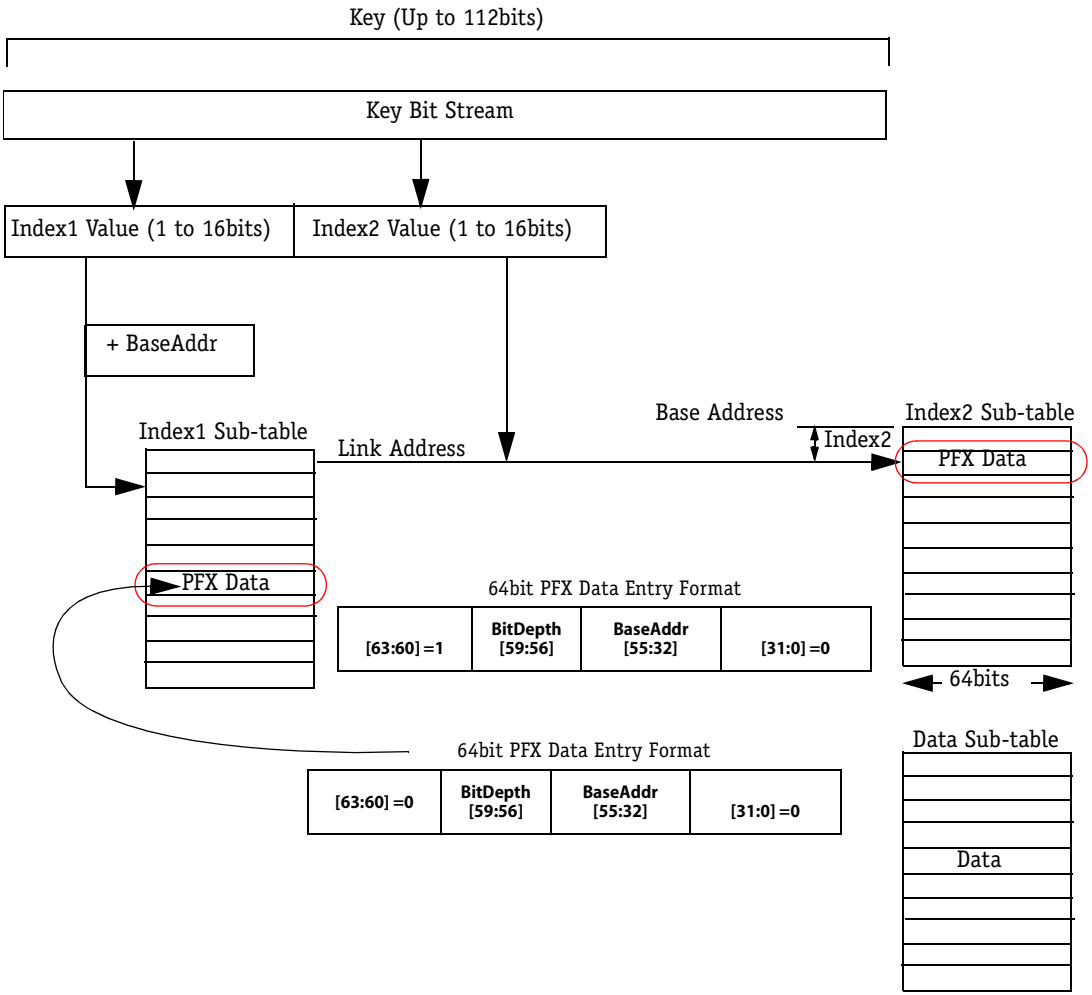
The Chained Index is another method for ATM circuit resolution, like a regular Index. Refer to Figure 77 on page 346.

In ATM, a connection is uniquely identified by a combination of a Virtual Path Identifier (VPI) + a Virtual Connection Identifier (VCI). This is a hierarchy where there can be a number of VCIs that belong to a single VPI. For ATM, the Chained Index provides two (2) functions:

- A way to perform lookups on a combined VPI and VCI route for forwarding.
- A way to access each VCI within a given VPI group.

Therefore, Chained Index provides a means from the host side to send VPI + VCI to the TLU and have that data record returned. This prevents having multiple tables.

Figure 77 Chained Index Block Diagram



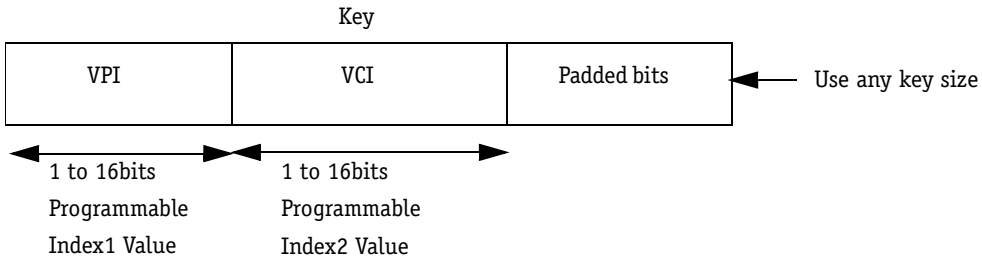
Chained Index Data Entry Format

The chain index data entry format uses the same format as PFX (Longest-Prefix Match). Refer to "PFX (Longest-Prefix Match) Data Entry Format" on page 352.

Chained Index Example

For example, for a ATM application load VPI and the VCI. Up to 112bits could be used for some other type of application. Refer to Figure 78 on page 347.

Figure 78 Chained Index ATM Application Example



The number of bits used for the VPI field or for the VCI field are not constants and can vary from application to application.

Chained Index vs. Chained Hash

Either the Chained Index table type or a Chained Hash table type could be used to perform a VPI/VCI lookup. In either case the data format would be the same (PFX-Data), the only difference is the selected table type. The distinction between the two (2) types is the relationship between cycle time versus memory resources. The two (2) cases are as follows:

- Case 1: Using the Chained Index table type, takes one cycle to access 8bytes but requires significantly more TLU memory to implement.
- Case 2: Using the Chained Hash table type, takes three cycles to access, but requires less memory.

PFX (Longest-Prefix Match)

PFX (Longest-Prefix Match) tables are used for *most-significant prefix match* algorithms. For example, in the case of searching for a 32bit IP address on a network, the PFX sub-table would search using the specific 32bit address. However, if a match was *not* found on the 32bit search then the PFX sub-table would return a close match of 24bits (not necessarily 24bits). Therefore, the PFX compresses the routing tables down so you do not need to know every host address on the network.

For example, consider a table with the following entries and depending on the lookup key what is returned:

Table entries: (IPS / number of bits in the mask)

- A= 214.21.128.4/32
- B= 214.21.128/17
- C= 214.21/16
- D= 214/8

Lookups and returns:

- If the lookup key is 214.21.128.4, then key A is returned.
- If the lookup key is 214.21.130.5, then key B is returned.
- If the lookup key is 214.21.55.5, then key C is returned.
- If the lookup key is 214.44.55.5, then key D is returned.
- If the lookup key is 215.44.55.5, then a miss is returned.

The PFX (Longest-Prefix Match) data structure is a compressed trie with several compression methods supported. The compression format used at each level of the trie is encoded within the trie itself. The hardware engine (TLU) can decompress any of the supported formats. Also, the build software (host via PCI or XP) determines dynamically which compression method to use at each level of the trie. The PFX search procedure always terminates with a match value for the search key. When a search key does not match any prefix in the table, its search terminates with an associated-data that points to a default route or no-match entry.

Figure 79 on page 349 shows the PFX (Longest-Prefix Match) state transitions and Table 101 on page 349lists their details.

Figure 79 PFX Transition States

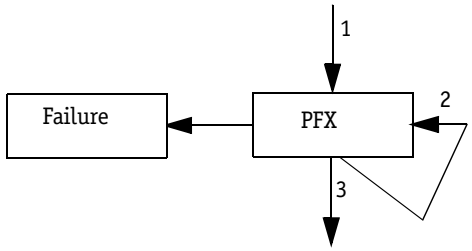


Table 101 PFX State Transition Details

STEP	ACTION
1	Accesses into indexed sub-table, using *MASK* MSB's of the key from the *Table_Configuration1* register. The results of this access is either: a pointer to associated data, or a PFX table entry. The PFX table entry consists of: • The base address of the next chunk (ChBase) that holds all the prefixes that are extensions of the prefix from step #1. • The number of bits of the key decoded to select an entry from the next chunk (that is, log2 of the uncompressed chunk size), ($KeySHL_z$). • The compression format of the chunk (ChType). • The summary code (ChCode) used to compute a random-access into the compressed chunk.
2	• If the result of step #1 is ChType 1 (ActionVec), then ChBase points to the keys associated data. Go to step #3. • If the result of step #1 is ChType 2 then there is no match and a miss is returned. For all other ChTypes, decode the next address, retrieve a new encoded entry and repeat step #2.
3	Finally, dereference the associated data pointer. NOTE: There is no compare step in the PFX trie search.

Refer to Figure 80 on page 350 for the PFX recommended memory organization.

Figure 80 on page 350 *shows how memory is laid out conceptually by the API. This is not hardware imposed, but a recommended map that can be useful to explain how the hardware works conceptually.*

Figure 80 PFX Recommended Memory Organization (Conceptually)

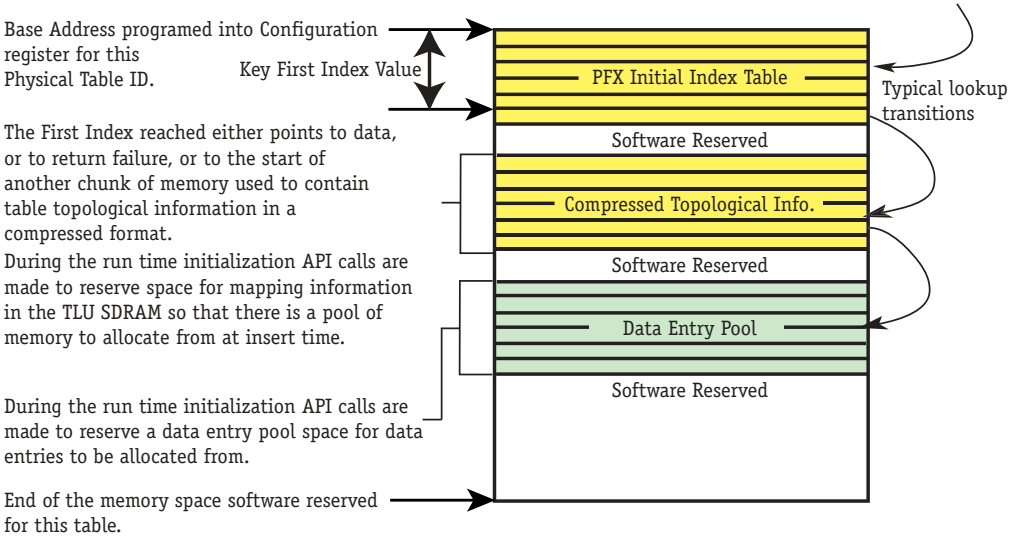
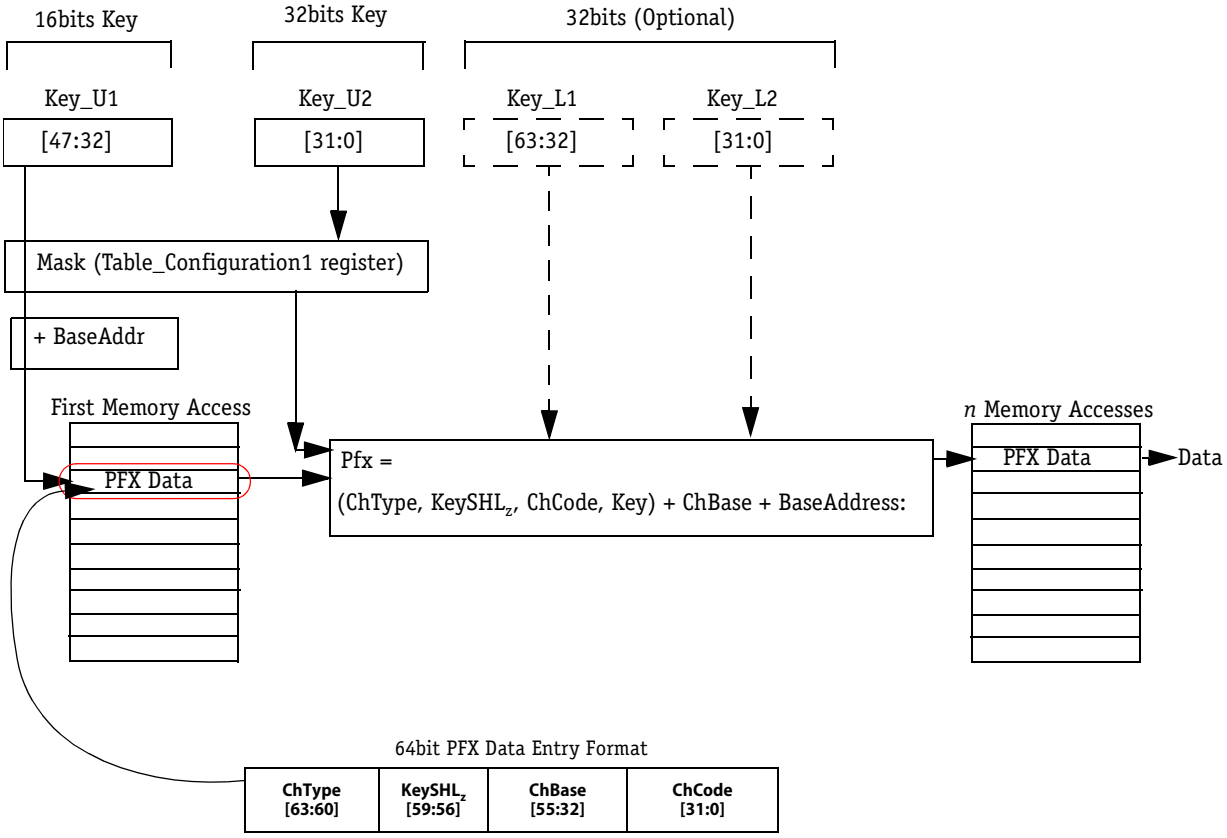
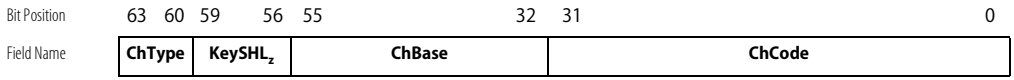


Figure 81 PFX Block Diagram



PFX (Longest-Prefix Match) Data Entry Format

The data entry format for PFX (Longest-Prefix Match) is described here:



FIELD NAME	BIT POSITION	DESCRIPTION
ChType	63:60	**Chunk Type** — Defines the type of compression used to encode the chunk. Legal values are provided here: ENCODED VALUE / COMPRESSION TYPE / DETAILS 0 / Initial / See "Initial Chunk Type" on page 353. 1 / ActionVec / See "ActionVec Chunk Type" on page 353. 2 / Fail (search terminates) / See "Fail Chunk Type" on page 353. 3 to 8 / Proprietary / N/A 9 / Hash / See "Hash Chunk Type" on page 353. 10 to 15 / Reserved / N/A
KeySHL$_z$	59:56	**Key Shift Left** — This is the number of key bits "consumed" by the instruction minus one. Thus, the encoded value ranges from 0 to 15. This indicates how many positions to shift-left the key *after* this stage. This is the log2 of the uncompressed size of the chunk. Note: KeySHL= KeySHL$_z$+1.
ChBase	55:32	**Chunk Base Address** — This is the beginning address where the chunk is stored. Chunks are stored in contiguous locations starting here.
ChCode	31:0	**Chunk Summary Code** — This allows the compressed chunk contents to be decompressed without loss. It also allows the search procedure to convert any desired offset in the uncompressed chunk, into the appropriate offset in the compressed chunk without having to read any of the compressed chunk contents. Note: The interpretation of this field is dependent on the *ChType* [63:60].

PFX (Longest-Prefix Match) Chunk Types Details

Each of the four (4) chunk types are described here:

Initial Chunk Type

The Initial chunk type indicates an uncompressed chunk. Typically, INIT is used for the root node of a PFX trie. For example, the first SRAM access may be to an index sub-table of 64k entries that fully decodes the high-order 16bits of the search key. The INIT chunk type would be used, with the $KeySHL_z$=15 and ChBase=0. The ChCode field is ignored.

ActionVec Chunk Type

The ActionVec chunk type indicates that a valid leaf (best matching prefix) has been reached. The data word holds the 60bit associated data record to be fetched from the table. This 60bit result may be the actual TLU return value, or a pointer that needs to be dereferenced to yield a wider TLU return value.

Fail Chunk Type

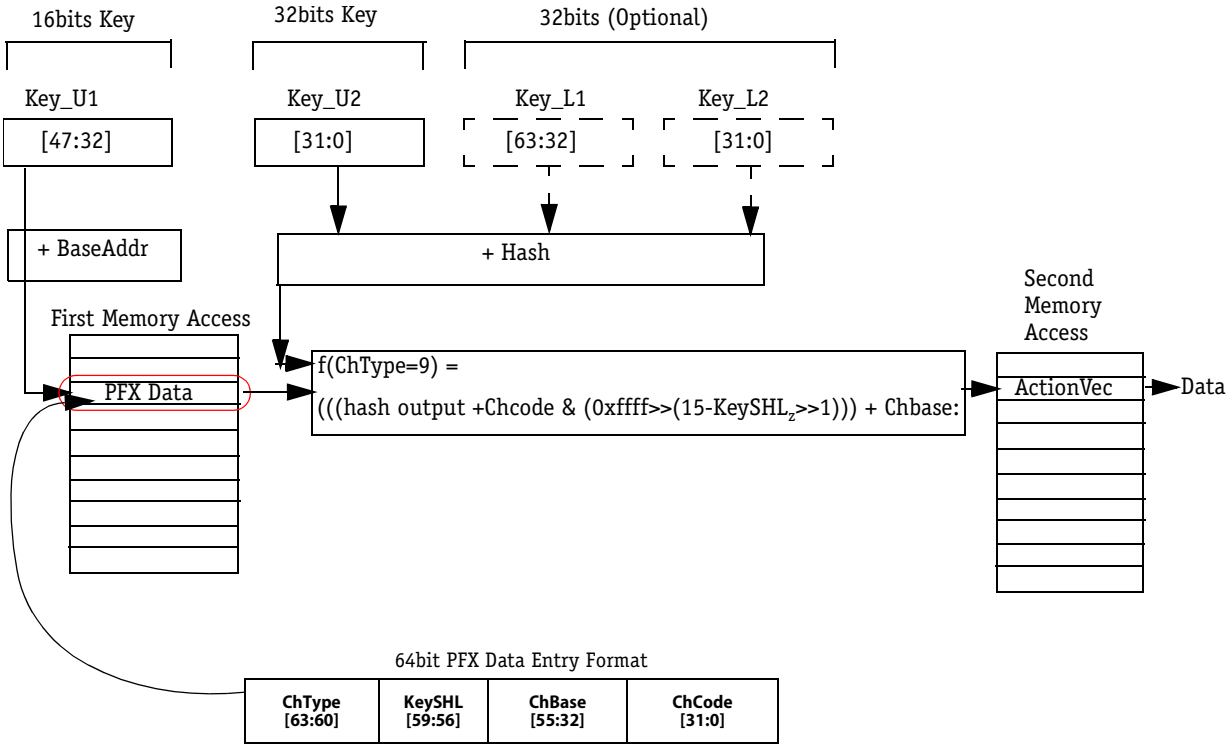
The Fail chunk type indicates that a null leaf (no matching prefix) has been reached. The data word is ignored and the TLU always returns a miss.

Hash Chunk Type

The Hash Chunk Type is used for Chained Hash table type operations.

To implement the chained hash, set the table type to PFX. The first memory access uses the Key_U1 as an index into a table. A value is returned from the table lookup that has base address, a pointer to a second table with a ChType=9, and a mask field ($KeySHL_z$). This second table is accessed using the base address specified + the hashed value of keys Key_U2, (Key_L1 & Key_L2) masked with BitDepth.

Figure 82 Hash Chunk Type Block Diagram

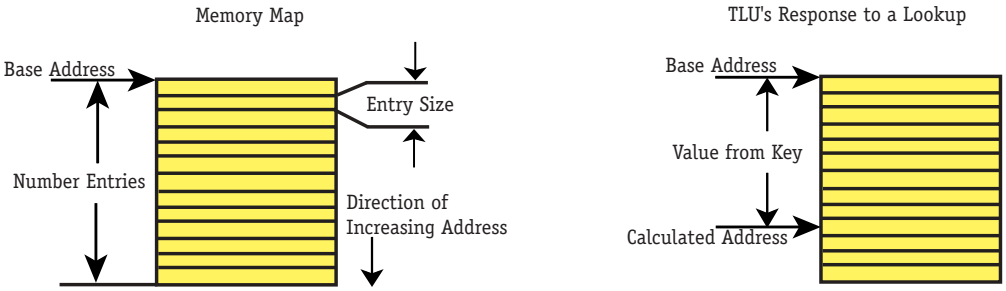


Flat Data Data tables contain the data associated with an entry. Data tables can be used as a stand-alone table.

Address= BaseAddress + Index <<Log_2 (Entry Size Slots)

A simple flat data table uses four (4) parameters that must be passed to the TLU. These include: TableId, Base Address, Number of Bits from Key (Keybits) to use in address generation (log_2 of the number of entries), and Entry Size. The API is used to keep track of four (4) items: which tables are in which TableId, which TableId's are available, which memory locations are already in use by other tables, and how much memory is needed by the current table. Refer to Figure 83 on page 355.

Figure 83 Flat Data Recommended Memory Organizational (Conceptually)



The API must mirror the lookup functionality of the hardware to perform inserts.

As this software algorithm provides a simple array to store data in and retrieve, it has no concept of delete. Therefore, a insert call is needed to overwrite data using a delete sential value. When the TLU is told to perform a lookup on a given key, regardless of the key size, it only uses the key value stored in the Key_U2 portion of the key. Refer to Table 102 on page 355.

Table 102 Key Format

Bit Position	0 16	17 531	32 63	64 95	96 127
Field Name	**Padded**	**Key_U1**	**Key_U2**	**Key_L1**	**Key_L2**

Within the Key_U2 portion of the word, the hardware uses the least significant Keybits of the word. Keybits are then shifted by the log of the Entry Size and added to the Base Address to determine which address to read. The TLU reads the data and returns the data. Because the hardware has no concept of memory maps, the TLU performs the calculation and returns the data with no capacity for error checking or knowledge of what is uninitialized data entries. It is a simple memory read. The API on the XP and host processors provide bounds checking in software to help programers identify software application flaws easier.

Flat Data Example

A common example of a stand-alone Data table would be the ATM VC table. Data can be read and written to this table by an index. Typically the concatenated VPI/VCI makes up the index. Another more specific example would be a implementing Partial CRC-32's for

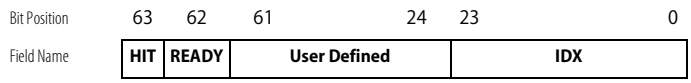
ATM Adaptation Layer-type 5 (AAL-5) reassembly that are stored in a Data table type structure. Refer to "Partial CRC-32 Support" on page 400.

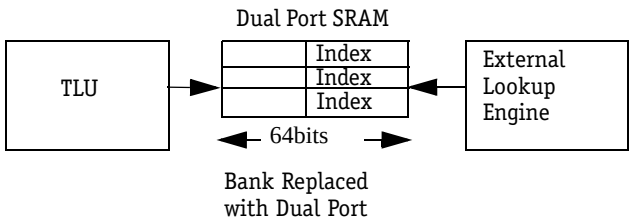
External The SRAM interface may alternatively be used to communicate with third party lookup engines. One or more SRAM banks are replaced by dual ported memories accessible to both the external lookup engine and the TLU. This memory serves as go-between for the two (2) devices. An external lookup table is constructed in at least one (1) of the shared memories. The table contains one (1) or more entries. Each entry must comply with the "external table interface format". These entries serve as mailboxes to synchronize data transfer between the two (2) devices.

When a *Find* or *Findr* lookup command is used to access the external table, the least significant 24 bits of the Key (32 or 48 bit keys only) are used to index the appropriate entry. The TLU then proceeds to continuously poll the entry until the *READY* field bit [62] is set by the external lookup engine. The TLU then checks the contents of the *HIT* field bit [63], if set to one (1), the TLU returns the appropriate lookup data, if set to zero (0), a lookup miss error (type 1) is returned.

The TLU polls for the *READY* field bit [62] assertion for up to 255 times before quitting and returning a watchdog time (WDT) error (type 2).

Figure 84 External Table Interface Format





TLU Commands Overview

The following section describes the eleven (11) commands used to control the TLU. These commands are sent to the TLU via the Ring Bus. Refer to Table 103 on page 357 for a list of the TLU commands with their parameters, command ID, returned data, and function.

Table 103 TLU Commands

COMMAND (PARAMETERS)	COMMAND ID	RETURNED DATA	FUNCTION
Write (VTBL#, IDX, MSK, DATA, OFF, LEN)	0x5	None	Write data into a virtual table at *index* plus *offset*.
Read (VTB#, IDX, OFF, LEN	0x4	Data	Reads data from a virtual table at *index* plus *offset*. Sets Ring Bus Error Flag if *key* is not found.
Find (VTBL#, KEY)	0x9	Physical Table Value, or Error with Index	Finds a *key* using *VTBL#*. Returns *index* of data. Sets Ring Bus Error Flag if *key* is *not* found.
Findw (VTBL#, KEY, DATA, OFF, LEN)	0x8	Physical Table Value, or Error with Index	Writes *data* into a table using a *key*. Sets Ring Bus Error Flag if the key is not found. Returns Index if found.
Findr (VTBL#, KEY, DATA, OFF, LEN)	0xc to 0xf	Data, or Error with Index	Reads *length* double words of *data* from a *vtable#* using a *key* at *offset* double words. Sets Ring Bus Error Flag if the key is not found.
XOR (VTBL#, IDX, DATA/PCRC, OFF, MSK, LAST)	0x7	None, or CRC in CRC mode.	XORs up to a 32bit value to *index* plus *offset*. Note: A special CRC mode exists for CRC calculations.
Add (VTBL#, IDX, DATA, OFF, MSK)	0x6	None	Adds up to a 32bit value to *index* plus *offset*.
WriteReg (ADDR, DATA)	0x2	None	Write *data* to TLU register at *ADDR*.
ReadReg (ADDR, DATA)	0x3	Data	Read *data* from TLU register at *ADDR*.
Echo (DATA)	0x1	Data	Return *data* from TLU. For test purposes.
NOP ( )	0x0	None	Inserts a NOP into the TLU pipe. Used to skip an SRAM access during that cycle.

A Ring Bus overflow error message is provided to indicate when a Ring Bus input FIFO is full. When a destination node (CPs, XP, or FP) is busy, the Ring Bus node sends an error message in the message FIFO back to the source node (CPs, XP or FP). The source node can then decide whether to resend the message or generate an error condition.

Only errors for messages are generated. No errors are generated for responses. Responses stay on the Ring Bus.

TLU Command Parameters

The TLU command parameters along with their functions are listed in Table 104 on page 358.

Table 104 TLU Command Parameters

TLU COMMAND PARAMETER FIELD	TLU COMMAND PARAMETER NAME	FUNCTION
VTBL#	Virtual table number	Virtual tables are mapped to a physical tables using the *Virtural_Table_Configuration* register. The actual physical table (*TBL#*) accessed is translated using the *Virtual_Table_Configuration* register.
IDX	Table index number	The index number points to a specific entire in a table. The index is used by the *Read*, *Write*, *Add*, and *XOR* commands. It is a 24bit value. An SRAM address is generated by multiplying the index with the size field programmed in the *Table_Configuration1* register.
OFF	Offset	The *offset* in 8Byte increments into the table entry. The legal range= 0 to 127. The actual SRAM address is given by: (base_address [table# [vtable#]] * 256) + (index << size [table# [vtable#]]) + offset
LEN	Length	The number of 8Byte words to read. Valid values are: • 0 for 8Bytes • 1 for 16Bytes • 2 for 32Bytes
MSK	Mask	Byte *mask* for *Writes* and Arithmetic Logic Unit (ALU) operations. Each bit corresponds to one (1) Byte.
KEY	Key	The *key* is used for all find commands (*Find*, *Findw*, *Findr*), to generate an index using the *VTBL#*. The TLU supports four (4) different Key sizes: 32, 48, 96 and 112bit. In addition, intermediate key sizes are supported by masking unused bits to zero.

Table 104 TLU Command Parameters (continued)

TLU COMMAND PARAMETER FIELD	TLU COMMAND PARAMETER NAME	FUNCTION
ADDR	Address	*Address* of a register to write or read.
DATA	Data	*Data* refers to the *DATA* field in each individual TLU command format. The purpose of the data field varies based on the TLU command. Therefore, for the specific definition of the *DATA* field refer to the particular TLU command format.
CRC	CRC enable	This enables the CRC mode.

CRC parameter is only used in the CRC Mode associated with the XOR command. Refer to "XOR Command" on page 372.

Detail TLU Commands

Each of the eleven (11) TLU commands are described in the following section along with its purpose, command ID, fields, bit positions. Also provided, where applicable are the command's data alignment rules, returned data and error types.

Write Command

The *Write* command is used to write data to the TLU's SRAM. Two (2) types of writes are available:

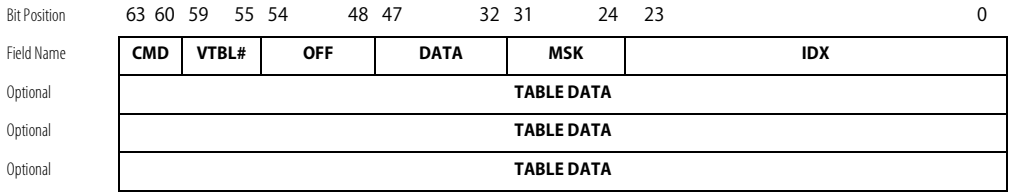
- The first type, is a masked write from one (1) to two (2) bytes in length. The bytes are selected using the *MSK* field bits [31:24]. Bytes must be contiguous and aligned on word boundaries (*that is,* a mask of 0x06 is illegal, while 0x03 and 0xc0 are both legal mask values). The write data is contained in the *DATA* field bits [47:32] in the first control word.
- The second type, uses consecutive Ring Bus slots to write to consecutive SRAM locations. Since the write command occupies the first Ring Bus slot, up to three (3) SRAM locations may be written consecutively. The TLU uses the value in the Ring Bus length field to determine the actual number of SRAM location to write. For two-slot writes, the mask can be set at either 0x0f or 0xf0 to write 32bits to the SRAM; or set to 0xff to write 64bits. For four-slot writes the mask field should be set to 0xff.

Data written with masks set to anything other than 0xff generate a read-modify write cycle. This means that a read occurs and then six clocks later the value is written back to the SRAM. Since this operation is not locked, another process could be executing a read-modify write on the same address resulting in corrupted data.

Write Command Format

Purpose Writes data to the TLU's SRAM.

Command ID 0x5



FIELD NAME	BIT POSITION	DESCRIPTION
CMD	63:60	**Command** — Set to 0x5 for Write.
VTBL#	59:55	**Virtual Table Number** — Identifies a Virtual table to use for the lookup. Legal range= 0 to 31.
OFF	54:48	**Offset** — Offset (in 8Byte increments) into table for read/write. Legal range= 0 to 127.
DATA	47:32	**Data** — Data field for 8 or 16bit (single bus slot) writes. *MSK* field determines data alignment in SRAM.
MSK	31:24	**Write Mask** — Byte mask for single slot writes (8Bytes). The mask is also used for two-slot and four-slot writes.
IDX	23:0	**Index** — Designates a table entry in a given TBL#.
TABLE DATA	63:0	**Table Data** — Data to write to SRAM. If these fields are present, then the *Data* field bits [47: 32] in the first slot is ignored. Set *MSK* field bits [31:24] to 0xff.

Write Command Data Alignment Rules

- 8bit writes are placed in the *DATA* field of the first slot and aligned to a byte boundary.
  - For masks of 0x01, 0x04, 0x10, and 0x40 data is stuffed into *DATA* field bits [39:32] of slot1.
  - For masks of 0x02, 0x08, 0x20, 0x80 data is stuffed into *DATA* field bits [47:40].
- 16bit writes are also placed in the *DATA* field bits [47:32] of the first slot.
- 32bit write data is stuffed into *TABLE DATA* field bits [31:0] of the second slot.
- 64bit write data is stuffed into *TABLE DATA* field bits [63:0] of the second slot.
- 192bit write data is stuffed into *TABLE DATA* field bits [63:0] of the second, third, and fourth slot.

Write Command Returned Data

The *Write* command does *not* return any data.

Write Command Error Types

The *Write* command does *not* return any errors.

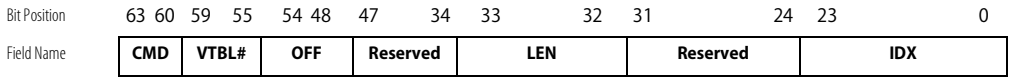
Read Command

The R*ead* command is used to read data from the TLU SRAM.

Read Command Format

Purpose Read data from the TLU's SRAM.

Command ID 0x4



FIELD NAME	BIT POSITION	DESCRIPTION
CMD	63:60	**Command** — Set to 0x4 for Read.
VTBL#	59:55	**Virtual Table Number** — Identifies a Virtual table to use for the lookup. Legal range= 0 to 31.
OFF	54:48	**Offset** — Offset (in 8Byte increments) into table for write. Legal range= 0 to 127.
Reserved	47:34	Read as zero.
LEN	33:32	**Length** — Tells the TLU how many SRAM locations to access. All SRAM reads are in multiples of 8Bytes. Number of 8Byte double words to read - 1. Legal ranges are detailed here:
Reserved	31:24	Read as zero.
IDX	23:0	**Index** — Designates a table entry in a given TBL#.

ENCODED VALUE	ACTUAL LENGTH
0	1
1	2
2	Illegal (Not supported)
3	4

Read Command Data Alignment Rules
The Read command does *not* have these rules.

Read Command Returned Data
The *Read* command returns the requested data to the calling function (CP or XP). If *index* or *offset* is out of range, the returned data is undefined.

Read Command Error Types
If a *parity mismatch* is found, then the Ring Bus error bit is set to one (1), the data field bits [15:0] are set to 0x3, and bits [55:32] are set with the address of the parity error.

Find Command

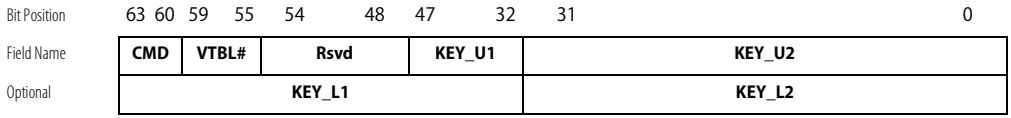
The *Find* command attempts to locate a key in a table using a preprogrammed table as specified by *VTBL#*.

Prior to executing this command, ensure the Table_Configuration1 register is properly setup. If not the Find command returns a indeterminate value.

Find Command Format

Purpose Find an index and a table, given a key.

Command ID 0x9



FIELD NAME	BIT POSITION	DESCRIPTION
CMD	63:60	**Command** — Set to 0x9 for Find.
VTBL#	59:55	**Virtual Table Number** — Identifies a Virtual table to use for the lookup. Legal range= 0 to 31.
Reserved	54:48	Read as zero.
KEY_U1	47:32	**Key Upper 1** — Upper 16bits of a 48bit or 112bit key.
KEY_U2	31:0	**Key Upper 2** — All 32bits of a 32bit key; lower 32bits of 48bit key; upper 32bits of 96bit key; upper-middle 32bits of 112bit key.
KEY_L1	63:32	**Key Lower 1** — Middle 32bits of 96bit key; lower-middle 32bits of 112bit key.
KEY_L2	31:0	**Key Lower 2** — Lower 32bits of 96bit key or 112bit key.

Find Command Data Alignment Rules
The *Find* command does *not* have these rules.

Find Command Returned Data
- The *Find* command returns a *table* value and an *index*. The data is formatted with the *index* in the *KEY_U2* field bits [31:0] and *table* in *KEY_U1* field bits [35:32].
- If the *key* is *not* found, then undetermined data is returned to the calling function (CP or XP) and the Ring Bus error bit is set to one (1).

Find Command Error Types
- If a *Find* command takes more than 255 SRAM accesses, the *Find* command times out, the Ring Bus error bit is set to one (1), and the data field bits [31:0] is set to 0x2.
  - In general, a time out error occurs while traversing a tree. Typically, this error occurs because the table data is corrupted as a result of improper implementation of the application software. For more details, refer to the *C-Ware API User Guide*.
- If the *key* is not found, then the Ring Bus error bit is set to one (1), the data field bits [31:0] are set to 0x1, and bits [55:32] are set with the initial index.

The returned initial index is useful in hash tables since the initial index is simply a hash of the key that has been masked off according to the settings in the Table_Configuration1 register.

- If a *parity mismatch* is found, then the Ring Bus error bit is set to one (1), the data field bits [15:0] are set to 0x3, and bits [55:32] are set with the address of the parity error.

Findw Command

The *Findw* command performs a *Find* followed by *Write*. As with a *Find* command, it locates a *key* in a table using a preprogrammed table as specified by *VTBL#*.

Prior to executing this command, ensure the Table_Configuration1 register is properly setup. If not the Findw command returns a indeterminate value.

Findw Command Format

The Findw has a similar format to Write, except it can only write 8Bytes of data, and does not support write masks.

Purpose — Find an index and a table, given a key and write data to the TLU's SRAM. Note: The Findw accommodates both a 2-slot and 4-slot format, as shown here.

Command ID — 0x8

Findw 2-slot format:

Bit Position	63 60	59 55	54 48	47 32	31 0
Field Name	CMD	VTBL#	OFF	KEY_U1	KEY_U2
Field Name	TABLE DATA				

Findw 4-slot format:

Bit Position	63 60	59 55	54 48	47 32	31 0
Field Name	CMD	VTBL#	OFF	KEY_U1	KEY_U2
Field Name	KEY_L1				KEY_L2
Field Name	TABLE DATA				
Optional	DUMMY DATA				

FIELD NAME	BIT POSITION	DESCRIPTION
CMD	63:60	**Command** — Set to 0x8 for Findw.
VTBL#	59:55	**Virtual Table Number** — Identifies a Virtual table to use for the lookup. Legal range= 0 to 31.
OFF	54:48	**Offset** — Offset (in 8Byte increments) into table for write. Legal range= 0 to 127.
KEY_U1	47:32	**Key Upper 1** — Upper 16bits of a 48bit or 112bit key.

FIELD NAME	BIT POSITION	DESCRIPTION
KEY_U2	31:0	**Key Upper 2** — All 32bits of a 32bit key; lower 32bits of 48bit key; upper 32bits of 96bit key; upper-middle 32bits of 112bit key.
KEY_L1	63:32	**Key Lower 1** — Middle 32bits of 96bit key; lower-middle 32bits of 112bit key.
KEY_L2	31:0	**Key Lower 2** — Lower 32bits of 96bit key or 112bits key.
TABLE DATA	63:0	**Table Data** — Data to write to SRAM.
DUMMY DATA	63:0	**Dummy Data** — Dummy field sent when long keys (>48bits) are used.

Findw Command Data Alignment Rules

The *Findw* command does *not* have these rules.

Findw Command Returned Data

- The *Findw* command returns a *table* value and an *index*. The data is formatted with the *index* in the *KEY_U2* field bits [31:0] and *table* in *KEY_U1* field bits [35:32].
- If the *key* is *not* found, then undetermined data is returned to the calling function (CP or XP) and the Ring Bus error bit is set to one (1).

Findw Command Error Types

- If the *Findw* command takes more than 255 SRAM accesses, the *Findw* command times out, the Ring Bus error bit is set to one (1), and the data field bits [31:0] is set to 0x2.
  - In general, a time out error occurs while traversing a tree. Typically, this error occurs because the table data is corrupted as a result of improper implementation of the application software. For more details, refer to the *C-Ware API User Guide*.
- If the *key* is not found, then the Ring Bus error bit is set to one (1), the data field bits [31:0] are set to 0x1, and bits [55:32] are set with the initial index.

The returned initial index is useful in hash tables since the initial index is simply a hash of the key that has been masked off according to the settings in the Table_Configuration1 register.

If a *parity mismatch* is found, then the Ring Bus error bit is set to one (1), the data field bits [15:0] are set to 0x3, and bits [55:32] are set with the address of the parity error.

Findr Command

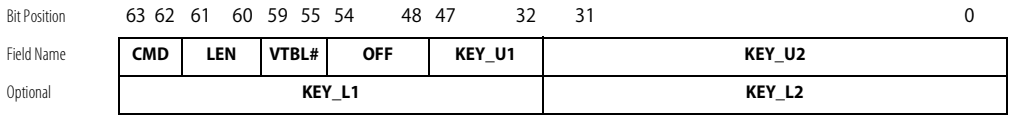
The *Findr* command performs a *Find* on a *key* and then a *Read*. As with a the *Find* command, it locates a *key* in a table using a preprogrammed table as specified by *VTBL#*.

Prior to executing this command, ensure the Table_Configuration1 register is properly setup. If not the Findr command returns a indeterminate value.

Findr Command Format

Purpose Find an index and a table, given a key and read data from the TLU's SRAM.

Command ID 0xc - 0xf



FIELD NAME	BIT POSITION	DESCRIPTION
CMD	63:62	**Command** — Set to 0xc - 0xf for Findr. Note: Command field is only 2bits.
LEN	61:60	**Length** — Tells the TLU how many SRAM locations to access. All SRAM reads are in multiples of 8Bytes. Number of 8Byte double words to read - 1. Legal ranges are detailed here: **ENCODED VALUE** / **ACTUAL LENGTH** 0 / 1 1 / 2 2 / Illegal (Not supported) 3 / 4
VTBL#	59:55	**Virtual Table Number** — Identifies a Virtual table to use for the lookup. Legal range= 0 to 31.
OFF	54:48	**Offset** — Offset (in 8Byte increments) into table entry for read. Legal range= 0 to 127.
KEY_U1	47:32	**Key Upper 1** — Upper 16bits of a 48bit or 112bit key.

FIELD NAME	BIT POSITION	DESCRIPTION
KEY_U2	31:0	**Key Upper 2** — All 32bits of a 32bit key; lower 32bits of 48bit key; upper 32bits of 96bit key; upper-middle 32bits of 112bit key.
KEY_L1	63:32	**Key Lower 1** — Middle 32bits of 96bit key; lower-middle 32bits of 112bit key.
KEY_L2	31:0	**Key Lower 2** — Lower 32bits of 96bit key or 112bit key.

Findr Command Data Alignment Rules
The *Findr* command does *not* have these rules.

Findr Command Returned Data
The *Findr* command returns the requested data to the calling function (CP or XP). If offset is out-of-range, the returned data is undefined.

Findr Command Error Types
- If a *Findr* command takes more than 255 SRAM accesses, the command times out, the Ring Bus error bit is set to one (1), and the data field bits [31:0] is set to 0x2.
  - In general, a time out error occurs while traversing a tree. Typically, this error occurs because the table data is corrupted as a result of improper implementation of the application software. For more details, refer to the *C-Ware API User Guide.*
- If the *key* is not found, then the Ring Bus error bit is set to one (1), the data field bits [31:0] are set to 0x1, and bits [55:32] are set with the initial index.

The returned initial index is useful in hash tables since the initial index is simply a hash of the key that has been masked off according to the settings in the Table_Configuration1 register.

- If a *parity mismatch* is found, then the Ring Bus error bit is set to one (1), the data field bits [15:0] are set to 0x3, and bits [55:32] are set with the address of the parity error.

Add Command

The *Add* command behaves similarly to the *Write* command, except that data is added to the existing data in the table. *Add* supports 8, 16, and 32bit add-ends.

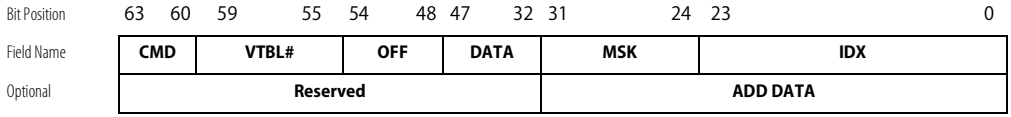
- For 8 and 16bit add-ends, the data is packed in the *DATA* field bits [47:32] of the first register. The *MSK* field bits [31:24] is used to identify the correct byte lane of the target add.
- For 32bit add-ends, the data is located in the lower 32bits of the *ADD DATA* field bits [31:0] in the optional register. The *MSK* field [31:24] is used to indicate if the target is aligned in the upper half of the SRAM (0xf0) or the lower half (0x0f) of the SRAM. To read the result of the *Add*, issue a *Read* at least four (4) clocks after the *Add* has been issued.

The Add command generates a read-modify-write cycle. This means that a read occurs and then six (6) clocks later the value is written back to the SRAM. This operation currently is not locked. Another process could be executing a read-modify-write on the same address resulting in corrupted data.

Add Command Format

Purpose Adds data to a Table Entry.

Command ID 0x6



FIELD NAME	BIT POSITION	DESCRIPTION
CMD	63:60	**Command** — Set to 0x6 for Add.
VTBL#	59:55	**Virtual Table Number** — Identifies a Virtual table to use for the lookup. Legal range= 0 to 31.
OFF	54:48	**Offset** — Offset (in 8Byte increments) into table entry for write. Legal range= 0 to 127.
DATA	47:32	**Data** — Data field for 8 or 16bit (single bus slot) writes. *MSK* field determines data alignment in SRAM.
MSK	31:24	**Byte Mask** — Byte mask for single slot writes (8Bytes).
IDX	23:0	**Index** — Designates a table entry in a given TBL#.

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	63:32	Read as zero.
ADD DATA	31:0	**Additional Data** — Optional Data field for 32bit adds. *MSK* field bits [31:24] determines data alignment in SRAM.

Add Command Data Alignment Rules

- 8bit writes are placed in the *DATA* field bits [47:32] of the first slot and aligned to a byte boundary.
  - For masks of 0x01, 0x04, 0x10, and 0x40 data is stuffed into *DATA* field bits [39:32] of slot1.
  - For masks of 0x02, 0x08, 0x20, and 0x80 data is stuffed into *DATA* field bits [47:40].
- 16bit writes are placed in the *DATA* field bits [47:32] of the first slot.
- 32bit write data is stuffed into the *ADD DATA* field bits [31:0] of the optional register.

Add Command Returned Data

The *Add* command does *not* return any data.

Add Command Error Types

The *Add* command does *not* return any errors.

XOR Command

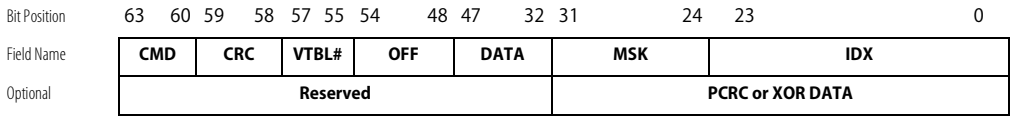
The *XOR* command behaves similarly to the *Add* command when *CRC* field bits [56:55] is set to 0x0. *XOR* supports 8, 16, and 32bit operands.

If the CRC field bits [56:55] are set to non-zero, 0x1, then the XOR command functions differently. Refer to "CRC Mode (Using the Non-zero XOR Command Options)" on page 374.

XOR Command Format

Purpose — Performs partial XOR operation on table data.
Note: The XOR command provides an alternative CRC Mode function using the available CRC field non-zero options.

Command ID — 0x7



FIELD NAME	BIT POSITION	DESCRIPTION
CMD	63:60	**Command** — Set to 0x7 for XOR operation.
CRC	59:58	**CRC** — This entry is the CRC as detailed here: **ENCODED VALUE** / **FUNCTION** 00 — XOR 01 — CRC (non-last) 10 — CRC Tx Last Note: This ensures that the Ring Bus error bit is not set. 11 — CRC Rx Last
VTBL#	57:55	**Virtual Table Number** — Identifies a Virtual table to use for the lookup. Legal range= 0 to 8. Note: XOR command is restricted to the first 8 tables.
OFF	54:48	**Offset** — Offset (in 8Byte increments) into table for write. Legal range= 0 to 127.

FIELD NAME	BIT POSITION	DESCRIPTION
DATA	47:32	**Data** — The data to be XORed when using 1-Slot. Mask determines destination. The TLU always initializes this field to all zeros at the end of the CRC calculation.
MSK	31:24	**Byte Mask** — Byte mask for single slot writes (8Bytes)
IDX	23:0	**Index** — Designates a table entry in a given TBL#.
Reserved	63:32	Read as zero.
PCRC	31:0	**Partial CRC** — Generated by the SDP. This value is XORed with the value at *VTBL#, IDX,* and *OFF.*
XOR DATA		**XOR Data** — Optional data for 32bit XORs.

XOR Command Data Alignment Rules

- 32bit write data is stuffed into bits [31:0] in the optional register, *PCRC* or *XOR DATA* field bits [31:0].
- 16bit writes are placed in the *DATA* field bits [47:32] of the first slot.
- 8bit writes are placed in the *DATA* field bits [47:32] of the first slot and aligned to a byte boundary.
  - So for masks of 0x01, 0x04, 0x10, and 0x40 data is stuffed into *DATA* field bits [39:32] of slot1.
  - For masks of 0x02, 0x08, 0x20, and 0x80 data is stuffed into *DATA* field bits [47:40].

XOR Command Returned Data

The XOR command does *not* return any data.

XOR Command Error Types

XOR command does *not* return any errors.

CRC Mode (Using the Non-zero XOR Command Options)

The CRC Mode also uses the same *XOR* command format. However, if the *CRC* field bits [56:55] are set to non-zero, 0x1, then the *XOR* command functions differently. When a non-zero value is selected three (3) CRC modes are available. Refer to Table 105 on page 374.

Table 105 Non-zero CRC Modes, Their Names and Parity Error Status

ENCODED VALUE	FUNCTION NAME	PARITY ERROR STATUS
00	XOR	No parity error
01	CRC (non-last)	
10	CRC TX Last	Potential for parity error
11	CRC Rx Last	

The TLU expects the target data to be in the format as described in "Partial CRC-32 Support" on page 400. Refer to "Partial CRC-32 Support" on page 400 and this section for a better understanding of the *XOR* command CRC Mode functions.

Table 106 Non-zero CRC Modes and Their Functions

ENCODED VALUE/ NAME	FUNCTION DETAILS
01/CRC (Non-last)	If set to 01,then the *PCRC* optional field bits [31:0] is XORed with the data at *index* and *offset*. *CRC_Len* field bits [47:32] of the Partial CRC-32 Data Entry Format is incremented by one (1). Refer to "Partial CRC-32 Data Entry Format" on page 401.
10/CRC Tx Last	If set to 10, then *PCRC* field bits [31:0] is XORed as above except, the data is *not* written back to SRAM. Instead, a 12bit index is generated from the upper 12bits of *CRC_Len* field bits [47:32].
11/CRC Rx Last	If set to 11, then *PCRC* field bits [31:0] is XORed as above except, the data is *not* written back to SRAM. Instead, a 12bit index is generated from the upper 12bits of *CRC_Len* field bits [47:32].

After a CRC Mode has been executed and completed, either a CRC Tx Last (10) or CRC Rx Last (11). Their data is located in the DATA field bits [47:32] in the XOR command format, and are transferred to the CRC_Len field bits [47:32] in the Partial CRC-32 Data Entry Format. The DATA field [47:32] (16bits) should hold a zero (0) for CRC Tx Last (10), whereas, it should hold a one (1) for CRC Rx Last (11). The DATA field bits [47:32] value is transferred to the CRC_Len field bits [47:32] which is used to reset the cell counter, that is, to either a zero (0) or a one (1). Refer to "Partial CRC-32 Support" on page 400.

CRC Mode Flow

The generated 12bit index, resulting from either a CRC Tx Last (10) or CRC Rx Last (11), is used to access the *Partial CRC table*. The Partial CRC table is a 4K table that is used to convert from CRC-32 to FCS or from FCS to CRC-32, starting at the *CRC-32_FCS_Correction_Table_Base_Address* register. The data from this table is rotated and XORed up to 16 times depending on the bottom four (4) bits of the *CRC_Len* field bits [47:32] of the Partial CRC-32 Data Entry Format and finally XORed with the *PCRC* field bits [31:0].

For CRC Rx Last (11) only, t he final value is compared with *CRC 32_Checkvalue* register bits [31:0]. Next, the TLU returns the data in the CRC data structure with the Ring Bus error flag set to one (1) to indicate the status of the compare.

For both the CRC Tx Last (10) and CRC Rx Last (11), the TLU resets the *PCRC* field bits [31:0] to zero (0).

The *XOR DATA* field bits [47:32] are copied to the *CRC_Len* field bits [47:32] in the SRAM entry.

If CRC field bits [56:55] are set to non-zero, then MSK field bits [31:24] must be set to 0x0f.

CRC Mode Data Alignment Rules

The alternative XOR function does *not* have these rules.

CRC Mode Returned Data

- For CRC (non-last) (01), does *not* return any data.
- For CRC Tx Last (10), the returned value is the actual (Full) CRC calculation. The CRC is loaded in bits [31:0] and the length is in bits [47:32].
- For CRC Rx Last (11), returns: the calculated CRC is in bits [47:16], the cell length is in bits [63:48] and a pass code of zero (0) in bits [15:0].

CRC Mode Error Types

- For CRC Rx Last (11) only:
  - If the calculated CRC does *not* equal the *CRC32_Checkvalue register*, then the Ring Bus error bit is set to one (1) and bits [15:0] are set to one (1).

CRC Mode Parity Error

- For both CRC Tx Last (10) and CRC Rx Last (11):
  - Reads to the TLU that generate a parity error return with the error bit set to one (1) and the error code bits [15:0] set to three (3). Parity checking is disabled for read modify write operations. Parity errors indicate an error in the FCS Correction table.

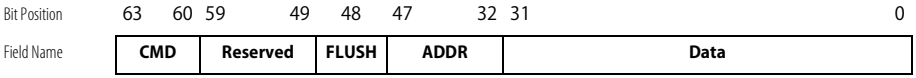
Write Register Command

The *WriteReg* command writes *data* to the register at *index*.

WriteReg Command Format

Purpose Write data to a Register.

Command ID 0x2



FIELD NAME	BIT POSITION	DESCRIPTION
CMD	63:60	**Command** — Set to 0x2 for Write Register command.
Reserved	59:49	Read as zero.
FLUSH	48	**Flush** — If set during a register write, then the TLU stalls until the pipe is empty before updating the register. This bit MAY need to be set if switching virtual tables on the fly. Note: Using this bit significantly slows down the TLU.
ADDR	47:32	**Address** — Address of register to write.
Data	31:0	Write data.

WriteReg Command Data Alignment Rules

All registers are 32 bits.

WriteReg Command Returned Data

The WriteReg command does *not* return any data.

WriteReg Command Error Types

The WriteReg command does *not* return any errors.

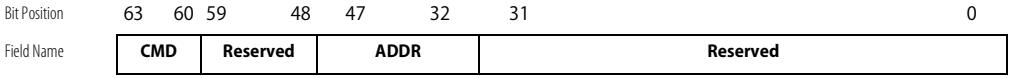
Read Register Command

The *ReadReg* command reads a register at an *address*.

ReadReg Command Format

Purpose Reads data from a Register.

Command ID 0x3



FIELD NAME	BIT POSITION	DESCRIPTION
CMD	63:60	**Command** — Set to 0x3 for a Read Register command.
Reserved	59:48	Read as zero.
ADDR	47:32	**Address** — Address of register to read.
Reserved	31:0	Read as zero.

ReadReg Command Data Alignment Rules

All registers are 32 bits.

ReadReg Command Returned Data

The ReadReg command returns *data* and returns the *value* of the register in the lower 32bits of the Ring Bus returned data, while the upper 32bits are undefined.

ReadReg Command Error Types

The ReadReg command does *not* return any errors.

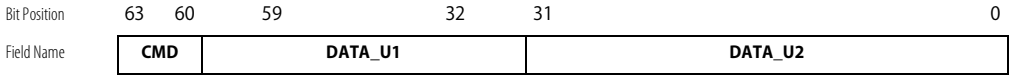
Echo Command

The *Echo* command "echoes" the input command and returns the input data to the output. The length of the returned data is always 8Bytes.

Echo Command Format

Purpose Copy the input command to the output.

Command ID 0x1



FIELD NAME	BIT POSITION	DESCRIPTION
CMD	63:60	**Command** — Set to 0x1 for the Echo command.
DATA_U1	59:32	**Data Upper 1** — Data to echo.
DATA_U2	31:0	**Data Upper 2** — Data to echo.

Echo Command Data Alignment Rules

The Echo command does *not* have these rules.

Echo Command Returned Data

The Echo command does *not* return any data.

Echo Command Error Types

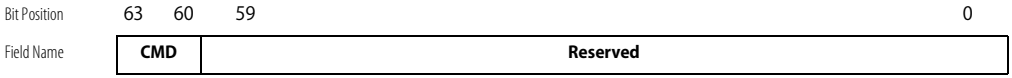
The Echo command does *not* return any errors.

No-Operation (NOP) Command

The *NOP* command inserts an empty slot into the TLU control pipe, causing the TLU to skip an SRAM access during that cycle.

Purpose Insert an empty slot into the TLU pipeline.

Command ID 0x0



FIELD NAME	BIT POSITION	DESCRIPTION
CMD	63:61	**Command** — Set to 0x0 for NOP.
Reserved	59:0	Read as zero.

Data Alignment Rules for NOP Commands

The NOP command does *not* have these rules.

Returned Data for NOP Commands

The NOP command does *not* return any data.

Error Types for NOP Commands

The NOP command does *not* return any errors.

TLU Table Mapping

The TLU uses *virtual* table numbers (*VTBL#*) to access physical tables (TBL#). The TLU has thirty-two (32) virtual tables. These virtual tables are mapped to physical tables using the *Virtual_Table_Configuration* register. Each virtual table can be mapped to anyone of the thirty-two (32) physical tables. This provides the ability to change the virtual table to physical table mapping on the fly.

In addition, virtual tables allow you to build and update one (1) copy of a topology table in the TLU while another table is being used by the forwarding path for lookups. The TLU can switch between these two (2) tables simply by adjusting the value of the *Virtual_Table_Configuration* register to point to the new table. This technique is called hot swapping of tables.

Hot swapping can also be used in two (2) other ways:

- To swap a new table in for an old table if the update would create a temporary corrupted table.
- To swap a new table for other reasons. For example, seemlessly switching to a newer, larger table when one grows too big.

Mapping Virtual Tables to Physical Tables

The *Virtual_Table_Configuration* register controls the mapping of virtual tables to the physical tables. By changing the value of the *TBL#* field in the *Virtual_Table_Configuration* register, the application can start performing its lookups in the new table as soon as the new value is written. The *Virtual_Table_Configuration* register is written to using the TLU *WriteReg* (0x2) command.

Ensure the WriteReg command's FLUSH field bit [48] is set. This guarantees that all table lookups in progress are completed before changing the virtual to physical table mapping.

Figure 85 on page 382 shows an example where there are two (2) copies of the Key table:

- One that is actively being used in the forwarding process
- The other is being populated by an application running on the XP or an external host

Figure 85 Example of Two Copies of a Table

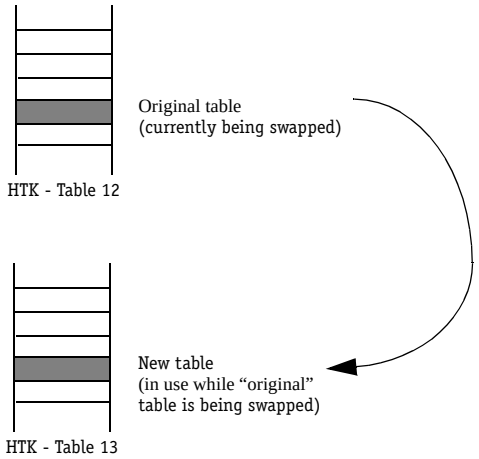


Table Number	
Virtual	**Actual**
5	5
7	6
12	12
12	13

TLU Configuration and Status Registers

TLU registers are 32bits wide, are accessed through the Ring Bus, and are addressed with a 16bit address. The TLU supports thirty-two (32) tables. Tables are numbered from (0 to 31).

For every table Id there is an associated configuration register. There are thirty-two (32) configuration registers (numbered from (0 to31)) and thirty-two (32) virtual table registers (also numbered from (0 to31)).

For TLU registers ≥ 0x100, the least significant byte defines the table number (*TBL#*) for the associated register. For example, to write the *Table _Configuration1* register for TBL#6, the register address would be 0x206.

TLU Registers

In general, the TLU configuration registers:

- Set the base address.
- Select the initial format.
- Calculates the initial address from a key.

Specifically, seven (7) registers are used to set up the TLU's virtual tables. The registers are used for three (3) purposes: CRC-32 mode operation, collecting TLU statistics, and configuration of the tables.

Table 107 TLU Registers

TLU REGISTER TYPE	REGISTER FUNCTION	SPECIFIC REGISTER DETAILS
CRC-32 Mode	Compares the final CRC-32 checksums.	See "CRC-32_Checkvalue Register" on page 384
	Contains base address of 2k Entry Correction Table.	See "CRC-32_FCS_Correction_Table_Base_Address Register" on page 385
Statistics	Records the minimum number of TLU FIFO slots after register reset.	See "TLU_Statistics Register" on page 386

Table 107 TLU Registers (continued)

TLU REGISTER TYPE	REGISTER FUNCTION	SPECIFIC REGISTER DETAILS
Table Configuration	Configure the TLU SRAM interface.	See "TLU_Memory Register" on page 387.
	Maps data tables to external tables.	See "External_Data_Table Register" on page 388.
	Defines table type's, key length, table entry size, mask and base address.	See "Table_Configuration1 Register" on page 389.
	Maps a virtual table to a physical table.	See "Virtual_Table_Configuration Register" on page 391.

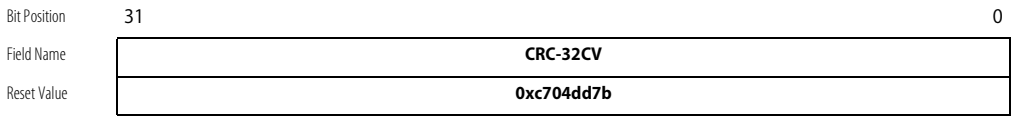
Each register is listed here along with its purpose, applicable fields, and parameters:

CRC-32_Checkvalue Register

This register compares the final CRC-32 checksums. It is used with the *XOR* command, CRC Mode function.

Purpose Used to compare the final Cyclic Redundancy Check (CRC) 32 checksums.

Address 0x0



FIELD NAME	BIT POSITION	DESCRIPTION
CRC-32CV	31:0	**CRC-32 Check Value** — Used to compare the final CRC-32 checksums.

CRC-32_FCS_Correction_Table_Base_Address Register

This register contains the base address of the 2K Entry Correction Table used to convert from CRC-32 to FCS or from FCS to CRC-32. The 2K Entry Correction Table is used by the *XOR* command, CRC Mode function, to calculate a final CRC given a sequence of partial CRCs.

Purpose Base address for the 2K Correction Factor table used to convert between FCS and CRC-32.

Address 0x1

Bit Position	31 … 16	15 … 0
Field Name	**Reserved**	**CRC/FCS Base**
Reset Value	**raz**	**0x0**

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:16	Read as zero.
CRC/FCS Base	15:0	**CRC/FCS Base Address** — This is the base address of the 2k Entry Correction Table used to convert from FCS to CRC or from CRC to FCS. The base address format is the same as for *Table_Configuration1* register.

TLU_Statistics Register

This register records the minimum number of TLU input FIFO slots after the register was reset. The input TLU FIFO is 80 (0x50) slots deep. The register is reset to 0x50 at power up, and can be reset again by writing any value to the register.

Purpose Records the minimum number of TLU input FIFO slots after the register was reset.

Address 0x2

Bit Position	31 … 8	7 … 0
Field Name	**Rsvd**	**MINFIFO**
Reset Value	**0x0**	**0x50**

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:8	Read as zero.
MINFIFO	7:0	**Minimum FIFO Slots** — Records the minimum number of TLU input FIFO slots after the register was reset.

TLU_Memory Register

Purpose Configure the TLU SRAM interface.

Address 0x3

Bit Position	31	30	6	5	0
Field Name	ParityEnb	Rsvd			BankConfig
Reset Value	0	0			0x1

FIELD NAME	BIT POSITION	DESCRIPTION
ParityEnb	31	**Parity Enable** — 1= enables parity checking, 0= disables parity checking. The TLU always writes the parity bit regardless of the state of the parity bit. When the parity bit is set, then any TLU read that occurs is checked against the eight (8) parity lines on the TLU SRAM. A mismatch causes the TLU to return with the error flag set and an error code of 0x3 in bits [31:0]. Also, the TLU returns the address of the parity error in bits [55:32].
Reserved	30:6	Read as zero.
BankConfig	5:0	**Bank Configuration** — Determines at what address the TLU starts addressing a new bank. The TLU provides a separate CEx line for each of the four (4) possible banks of the TLU Engine. (TCE0x, TCE1x, TCE2x, and TCE3x). The *BankConfig* field [5:0] determines at what address range each of the CEx lines becomes active. Refer to description for detail settings. Settings detailed here:

ENCODED VALUE	NUMBER OF ADDRESS BITS	MEMORY CAPACITY PER BANK
000001	17 (default)	1MBytes
000010	18	2MBytes
000100	19	4MBytes
001000	20	8MBytes
010000	21	16MBytes
100000	22	32MBytes

External_Data_Table Register

Purpose Maps data tables to external tables.

Address 0x4

Bit Position	31 ... 5	4 ... 0
Field Name	**Rsvd**	**TBL#**
Reset Value	**0**	**0x0**

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:5	Read as zero.
TBL#	4:0	**Table Number** — Physical table number.

Table_Configuration1 Register

This register defines the table's type, key length, table entry size, mask and its base address. The TLU SRAM is 64bits wide and all addressing is on an 8Byte boundary. Therefore, a TLU SRAM address of 1 refers to byte 8, and an address of 8 refers to byte 64, and so on. The base address in *Table_Configuration1* register is the TLU SRAM address divided by 256. Thus, the next base address is: current base address + (table entry size/8) x number of entries) with the result rounded up to the next 2KByte boundary.

Purpose Defines the table type, key length, table entry size, mask and base address.

Address 0x100 to 0x11f

Bit Position	31 28	27 24	23 21	20 16	15 0
Field Name	TYPE	KLEN	SIZE	MASK	BADDR
Reset Value	0	0	0	0	0

FIELD NAME	BIT POSITION	DESCRIPTION
TYPE	31:28	**Table Type** — Defines table type. Legal ranges are detailed here:

ENCODED VALUE	TABLE TYPE
0x0	Data
0x1	PFX (Longest-Prefix Match)
0x2	Reserved
0x3	Hash
0x4	Reserved
0x5	Reserved
0x6	External
0x7 to 0xF	Reserved

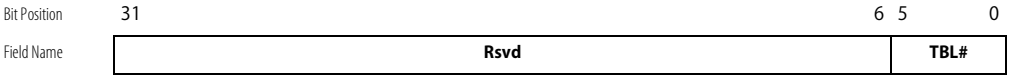
<table>
<tr><th>FIELD NAME</th><th>BIT POSITION</th><th>DESCRIPTION</th></tr>
<tr><td>KLEN</td><td>27:24</td><td>Key Length — Defines the length of the key. Legal ranges are detailed here:
Note: Intermediate key sizes are supported by masking unused bits to zero.
<table><tr><th>ENCODED VALUE</th><th>KEY SIZE (BITS)</th></tr><tr><td>0x4</td><td>32</td></tr><tr><td>0xc</td><td>48</td></tr><tr><td>0x7</td><td>96</td></tr><tr><td>0xf</td><td>112</td></tr></table></td></tr>
<tr><td>SIZE</td><td>23:21</td><td>Table Entry Size — This field defines the size of an individual table entry. The entry size is $2^{(Table\_Entry\_Size+3)}$ + Size bytes.
Note: The minimum entry size is 8Bytes and the maximum is 1024Bytes.</td></tr>
<tr><td>MASK</td><td>20:16</td><td>Mask — Generally, it defines a mask to apply to the key for an indexed lookup.This field is used by each table type in a slightly different way as detailed here:
<table><tr><th>TABLE TYPE</th><th>MASK USAGE</th></tr><tr><td>Data</td><td>Not used.</td></tr><tr><td>PFX</td><td>Associates to $KeySHL_z$ field [59:56] in PFX data entry format. Legal range= 0 to 15.</td></tr><tr><td>Hash</td><td>The size of the hash table is defined as 2^Mask.</td></tr></table></td></tr>
<tr><td>BADDR</td><td>15:0</td><td>Base Address — Defines the base SRAM offset (index) of a table. The base SRAM offset is defined as (BADDR x 256). The upper bounds of the table are not defined.</td></tr>
</table>

Virtual_Table_Configuration Register

This register maps a virtual table (*VTBL#*) to a physical table (*TBL#)*. All TLU commands use a virtual table number (*VTBL#*) as an argument. The default reset value is the least significant three (3) bits of the table address, so the default value of the register at address 0x306 is 0x6. All table references are through a virtual table (*VTBL#*).

Purpose Maps a virtual table to a physical table.

Address 0x300 to 0x31f



FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:6	Read as zero.
TBL#	5:0	**Table Number** — Physical table number.

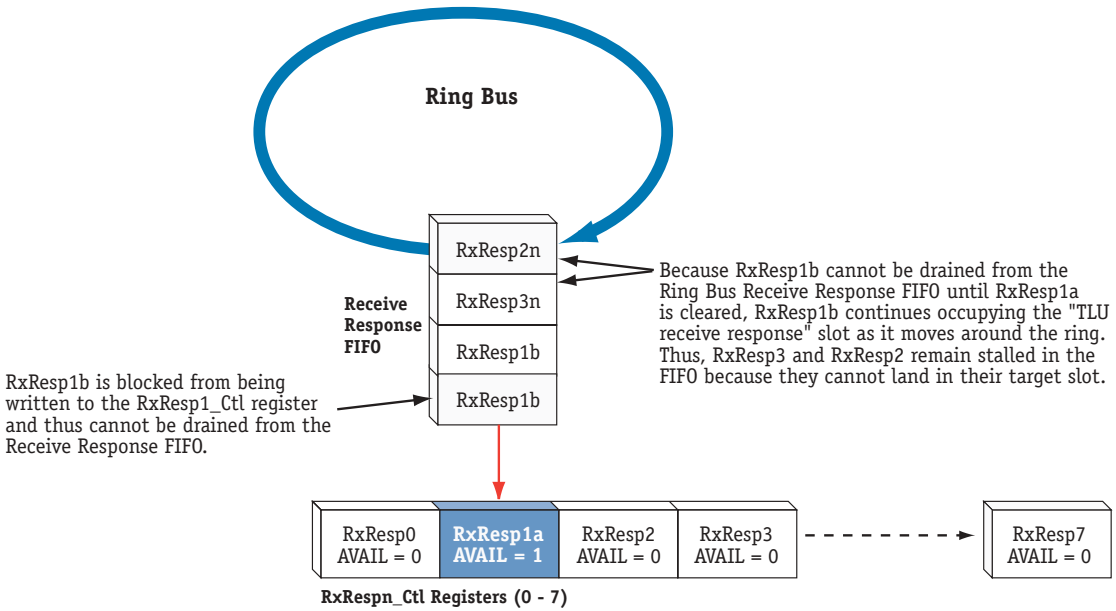
TLU Application Considerations

The following section covers issues that are important to application design. For more information about how to implement tables in applications see the *C-Ware Reference Library* document in the *C-Ware Application Development Guide*.

TLU/Ring Bus Control Register Response Slot Usage

TLU lookup results are returned via the Ring Bus and put into the *RxRespn_Ctl* register that was specified during the Ring Bus launch. If a TLU response slot is currently "occupied" with a previous response (Resp1) and a second lookup result (Resp2) destined for the same slot is ready, the second response (Resp2) is placed in the eight (8) slot receive response FIFO and remains in the FIFO until the first response is released and response slot is cleared. A response slot is released by deasserting the *AVAIL* bit [31] in the Ring Bus' *RxRespn_Ctl* register. Refer to Figure 86 on page 392.

Figure 86 TLU/Ring Bus Control Register Response Slot Usage



Responses move into the receive response FIFO on the destined processor (CP or XP). If the response at the head of the FIFO can't get into a response register slot because it's occupied by another response, then it is "trapped" at the head of the FIFO, causing head-of-line blocking. When the entire response FIFO is full, then other responses destined for this CP start circling the Ring Bus. Refer to "Ring Bus Overview" on page 477.

TLU Performance

The following sections detail two elements of TLU performance: throughput and latency.

TLU Throughput

Refer to Figure 87 on page 393 and Table 108 on page 393.

Figure 87 Throughput Formula

```
Throughput Formula:
        Throughput (Commands/Sec)= SRAM Clock Frequency/Average # of SRAM Accesses per Command

            For Example:

                = 133MHz/2 Accesses/PFX Find Command= 66M PFX Find Commands/Second
```

To determine the average number of SRAM access per TLU command for an application, first characterize the percentage of each command type. Then, refer to Table 108 on page 393 and Table 109 on page 394 for the number of accesses for each type. Next, calculate the average.

For instance, assume an application that uses 50% PFX FIND commands and 50% XOR commands. The FINDs require 3 SRAM accesses and the XORs require 2 SRAM accesses. The average is (.5(3) + .5(2)) = 2.5 SRAM accesses/command or (SRAM AVG.)

Table 108 SRAM Accesses per TLU Command

TLU COMMAND TYPES	SRAM ACCESSES
Write()	Roundup (# bytes written/8)
Read()	Roundup (# bytes read/8)
Add() or XOR()	2
Find()	Refer to Table 109 on page 394.
Findw()	Find + Roundup (# bytes written/8 +1)
Findr()	Find + Roundup (# bytes read/8 +1)

The number of SRAM accesses to issue a *Find* type command is highly dependent on the algorithm being used and upon the table entries themselves. Estimated values are given in Table 109 on page 394. For the Trie structures, assume an equally balanced tree with 64k nodes. If there are *n* keys and 2 x *n* hash buckets, the number of collisions is almost always less than eight (8).

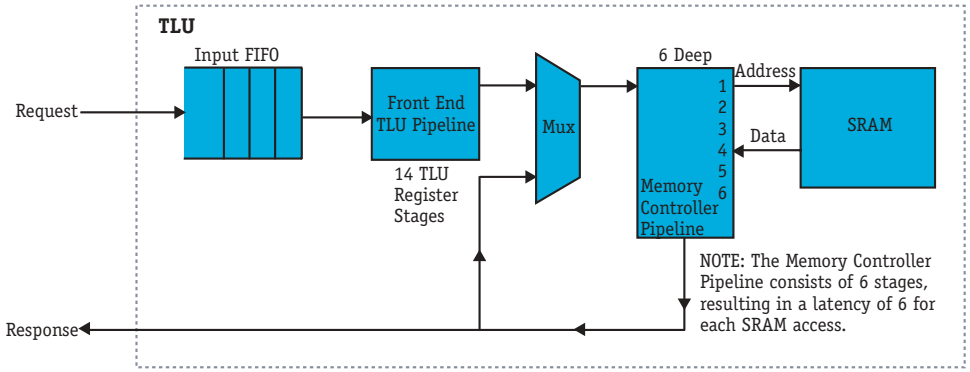
Table 109 SRAM Access for Find Command

	SRAM ACCESSES		
TABLE TYPE	**MIN.**	**TYPICAL**	**MAX.**
Hash Trie Key	2	$2 + \log_2 n$	2 + n
PFX (Longest-Prefix Match) Assumes a key size of = 32bits.	1	2	3
Chained Index	2	2	3
Chained Hash	3	$3 + \log_2 n$	3 + n
External	Vendor dependent		

TLU Latency

Use the following as a general guideline for estimating the TLU latency. Performance estimates are based on a ZBT SRAM. The TLU operation frequency is assumed to be equal to and synchronous with the SRAM clock frequency. Synchronization to the C-5e core clock occurs using the input and output FIFOs. Refer to Figure 88 on page 395.

Figure 88 TLU Pipeline Elements Affecting Latency Formula



Latency Formula:
Latency (ns)= 1000/Clock Speed (MHz) * (Round-Trip Ring Bus Delay + Input FIFO Delay + Front-End TLU Pipeline Delay + Memory Controller Delay)

Best Case TLU Latency	Worst Case TLU Latency
Item 1: 18 core clocks	90 core clocks (Note: Assumes Ring Bus request is accepted by the TLU and not recirculated or returned to sender).
Item 2: 1 TLU clock	Number of request that are backlogged * SRAM AVG.
Item 3: 14 TLU clocks	14 * SRAM AVG. in TLU clocks
Item 4: 6 TLU clocks	6 * SRAM AVG. in TLU clocks

Note for Worst Case, Items 2,3 & 4: SRAM AVG. refers to the average number of SRAM accesses, for more details refer to the 'TLU Throughput" section.

Note for Best Case, Item 4: Best occurs for access to Index tables. Access to Complex table types require more than 6 TLU clocks.

For the worst case TLU latency, the rate at which the TLU pipeline advances is limited by the rate at which commands complete all of their SRAM accesses. This rate can be specified in terms of SRAM AVG. as described in "TLU Throughput" on page 393.

The Input FIFO delay is dependent on the FIFO backlog. The backlog is a function of the input command rate versus the SRAM access rate. Fourteen (14) is the number of register stages within the Front End TLU Pipeline. The Memory Controller depth is a fixed value of 6. It represents the number of register stages in the memory controller's pipeline.

Table Sizing Examples

To aid in sizing tables, examples using two (2) typical applications are described in this section. Both examples are for a Layer 2/3 switch and list the required tables types, number of entries, sizes of entry, and sizes for the tables.

As the numbers indicate in the sizing examples shown in this section, an application implementing both a Bridge Address table and an IP Routing table uses about 8.5M of SRAM space. Refer to Table 110 on page 396, and Table 111 on page 396.

These examples do not include any other types of statistics or QoS tables, nor do they count an external TLU device out one bank.

Bridge Address Table Sizing Example

The Bridge Address table consists of one (1) Index table that is used to map VIDs to FIDs for 802.1Q, one (1) Hash table consisting of three (3) sub-tables used for the MAC address lookup. The sub-tables consists of: Hash, Trie, and Key for the 24Bytes of associated data. A typical size ranges from 10k to 128k entries. This example uses a 128k entry table.

Table 110 Bridge Address Table Sizing Example

TABLE TYPE	SUB-TABLE TYPES	NUMBER OF ENTRIES	ENTRY SIZE (BYTES)	TABLE SIZE (BYTES)
Hash Trie Key	Hash	256k	8	1M
	Trie	32k	8	262k
	Key	128k	32	4.1M

IP Routing Table Sizing Example

The IP Routing table consists of a PFX table, and two entry types including: Table, and Data. A typical size for an IP routing table ranges from 5k and 64k. This example uses 38k entries with 16Bytes of associated data.

Table 111 IP Routing Table Sizing Example

TABLE TYPE	ENTRY TYPE	NUMBER OF ENTRIES	ENTRY SIZE (BYTES)	TABLE SIZE (BYTES)
PFX	Table Entry	98k	8	800k
	Data Entry	38k	16	608k

Table 111 *on page 396 consists of preliminary numbers.*

TLU Special Applications

This section describes two (2) specific applications that are supported: long lookups and partial CRC-32 for ATM Adaptation Layer-type 5 (AAL-5).

Using the RxByte Processor for Long Lookups

Some protocols that are implemented in the RxByte processor require that sophisticated, long lookups be launched from the SDP to hide latency.

Since the RxByte processor only has access to two (2) of the *TxMsg* registers to launch lookups from, the SDP can only launch lookups of keys that are up to 112bits in length. The two (2) lookup slots are referred to as *TxMsg0* and *TxMsg1*. The size of these registers are listed in Table 112 on page 397.

Table 112 TxMsg*n* Registers and Their Size

REGISTER	SIZE (BITS)
TxMsg*n*_Ctl	32
TxMsg*n*_Data_H	32
TxMsg*n*_Data_L	32

When launching a lookup from the SDP, the *TxMsgn_Ctl* register contains control information for putting the request onto the Ring Bus. Such information would contain the source Ring Bus node, destination Ring Bus node, message length, return message slot, and so on.

To use multi-slots, you must launch the lookup using the *TxMsg0_Ctl* register in addition to the *TxMsgn_Data_n* registers. The keys are contained in some or all of *TxMsg0_Data_H*, *TxMsg0_Data_L*, *TxMsg1_Data_H*, and *TxMsg1_Data_L* registers.

The *TxMsgn_Data_n* registers contain the actual data that the destination node (in this case the TLU) processes. Information contained in the *TxMsgn_Data_n* registers would include: the lookup table in the TLU, the command (*Read*, *Findr*, *XOR*, etc.) and most importantly the lookup key.

For keys that are 48bits, 96bits, and 112bits, two (2) sets of the *TxMsgn_Data_n* registers must be used. The formats of these registers are shown in Table 113 on page 398.

Table 113 Large Key Data Format, >48bits

	MSB			LSB
FIELD	**BYTE 0**	**BYTE 1**	**BYTE 2**	**BYTE 3**
TxMsg0_Data_H	XXX	XXX	KU1	KU1
TxMsg0_Data_L	KU2	KU2	KU2	KU2
TxMsg1_Data_H	KL1	KL1	KL1	KL1
TxMsg1_Data_L	KL2	KL2	KL2	KL2

XXX = Reserved by TLU.
KU1 = Key upper 1 (upper 16b of key 1)
KU2 = Key upper 2 (lower 32b of key 1)
KL1 = Key lower 1 (upper 32b of key 2)
KL2 = Key lower 2 (lower 32b of key 2)

Depending on the configuration of the TLU table's key size, the information looked up as the key from the set of registers varies as listed in Table 114 on page 398.

Table 114 Key Size versus Key Match

KEY SIZE (BITS)	KEY MATCH
32	Key match is done on contents of KU2 fields
48	Key match is done on contents of KU1 + KU2 fields
96	Key match is done on contents of KU2 + KL1 + KL2 fields
112	Key match is done on contents of KU1 + KU2 + KL1 + KL2 fields

Long Lookup Example for an Ethernet Application

For IEEE 802.1Q tagged frames, the Ethernet application must lookup the MAC address and the VLAN ID of the frame which is: 48bits + 12bits = 60bits. Since there is no 60bit key size available, use the next larger key size available which is (96bit). The format for the lookup is listed in Table 115 on page 399.

Table 115 Ethernet Application Lookup Format

	MSB			LSB
FIELD	**BYTE 0**	**BYTE 1**	**BYTE 2**	**BYTE 3**
TxMsg0_Ctl	control information + length, indicating lookup uses both slots			
TxMsg0_Data_H	XXX	XXX	UUU	UUU
TxMsg0_Data_L	VVV	VVV	MM0	MM1
TxMsg1_Data_H	MM2	MM3	MM4	MM5
TxMsg1_Data_L	PPP	PPP	PPP	PPP

XXX = Reserved by the TLU
UUU = Unused
VVV = VLAN ID
MM*n* = MAC address
PPP = Not used by application (padded). These bits should be cleared by the application when doing lookups and when installing entries into the TLU.

From the SDP's perspective, the *TxMsgn_Data* words are mapped as shown in Table 116 on page 399.

Table 116 TxMsgn_Ctl Mapping

BIT	MAPPING
TxMsg0_Data7	XXX
TxMsg0_Data6	XXX
TxMsg0_Data5	UUU
TxMsg0_Data4	UUU
TxMsg0_Data3	VVV
TxMsg0_Data2	VVV
TxMsg0_Data1	MM0
TxMsg0_Data0	MM1

Table 116 TxMsgn_Ctl Mapping (continued)

BIT	MAPPING
TxMsg1_Data7	MM2
TxMsg1_Data6	MM3
TxMsg1_Data5	MM4
TxMsg1_Data4	MM5
TxMsg1_Data3	PPP
TxMsg1_Data2	PPP
TxMsg1_Data1	PPP
TxMsg1_Data0	PPP

In this case, the fields that are matched (given that this is a 96bit key) are the VVV + MM*n* fields, resulting in the lookup of the MAC address + VLAN ID according to the 802.1Q specification.

Ethernet Application Example Implementation Notes

In the Ethernet application example, the SDP lookups just the VID + MAC DA (destination address) in the RxByte processor if the frame is VLAN tagged. The CPRC gets the result of that look up that provides the forwarding route.

If the frame is untagged, then the MAC DA address is used with the Port VLAN ID (PVID) of the port from the SDP.

Only the TxMsg0_Ctl register can be used for multi-slot lookups.

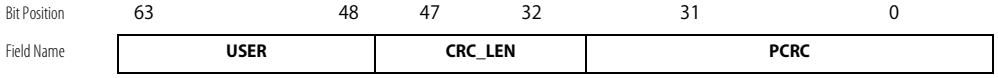
Partial CRC-32 Support

Partial CRC-32's for ATM Adaptation Layer-type 5 (AAL-5) reassembly can be stored and accumulated in a Data table type structure. The format and implementation steps are described in this section.

This is achieved using the SDP in conjunction with the TLU to accumulate the partial CRC's for many Virtual Connections (VC). Specifically, this is supported using the TLU's *XOR* command, a VC data table, a CRC-32 Correction Table, the *CRC-32_Correction_Table_Base_Address* register, and the *CRC-32_Checkvalue* register. The *XOR* command automatically increments the *CRC_Len* field bits [47:32] after adding a new partial CRC. Refer to "XOR Command" on page 372.

Partial CRC-32 Data Entry Format

The Cyclic Redundancy Check (CRC) entry is used for CRC error checking as part of a Virtual Connection (VC) data table. The entry must conform to this format:



FIELD NAME	BIT POSITION	DESCRIPTION
USER	63:48	**User Defined Data** — The user can insert anything here.
CRC_LEN	47:32	**CRC Length** — Number of cells holding current packet.
PCRC	31:0	**Partial CRC** — The current partial CRC.

Partial CRC-32 General Setup

Follows these steps to implement the Partial CRC-32 Operation:

1. Create and initialize CRC-32 Correction Table. (Each entry contains a CRC calculated for 0xff plus 48 times the number of cells; that is, 4Bytes of zeros).
2. Set up *CRC-32_Correction_Table_Base_Address* register to the start address of the newly created table (CRC-32 Correction Table).
3. In the *SDPMode3* and *SDPMode5* registers, set both the *RxByteCRCinit* field and *TxByteCRCinit* field= 0 in order to initialize the CRC-32 block to zero when the *CRCinit* command is executed by the RxByte and TxByte programmable processor's microcode.

Partial CRC-32 Rx Setup and Operation

Follow this step to implement the Rx side of the Partial CRC-32 Operation:

1. Set up Rx CRC data entry in the receive VC table as follows: *CRC_Len*= 1, *PCRC*= 0.

All cells except for the last end of message (EOM) cell are handled via the RxByte programmable processor, as follows:

1. RxByte programmable processor calculates *PCRC* on current cell.
2. RxByte programmable processor sends TLU XOR command with current PCRC.

TLU's *XOR* command settings are indicated here:

Bit Position	63 60	59 58	57 55	54 48	47 32	31 24	23 0
Field Name/Setting	**CMD=7**	**CRC= 0x01**	**VTBL#= User Defined**	**OFF=User Defined Offset**	**N/A**	**MSK=0x0f**	**IDX=User Defined Index**
Optional	**Reserved**					**PCRC=From RxByte Calculation**	

3 TLU *XORs* PCRC with (accumulated) PCRC and obtains a (new accumulated) PCRC. Then this value is shifted by 48Bytes and stored as the new PCRC in the VC table's CRC-32 Entry.

Last cell end of message (EOM) is handled via the RxByte programmable processor, as follows:

1 RxByte programmable processor calculates PCRC on last cell.

2 RxByte programmable processor sends TLU *XOR* command with PCRC and sets the *XOR* command bits [59:58] *CRC* field= 11 (Rxlast).

TLU's *XOR* command settings are indicated here:

Bit Position	63 60	59 58	57 55	54 48	47 32	31 24	23 0
Field Name/Setting	**CMD=7**	**CRC= 0x11**	**VTBL#= User Defined**	**OFF=User Defined Offset**	**N/A**	**MSK=0x0f**	**IDX=User Defined Index**
Optional	**Reserved**					**PCRC=From RxByte Calculation**	

3 TLU receives request and *XORs* PCRC with (accumulated PCRC); TLU does a table lookup based on PDU length to PCRC and adds the value looked up with the (accumulated PCRC). TLU compares (accumulated PCRC) with *CRC-32_Checkvalue* register. If the *CRC_Checkvalue* register does not match, the TLU sets a Ring Bus error indication for the PCRC. TLU re-sets accumulated *PCRC*=0, and *CRC_Len*= 1.

Partial CRC-32 Tx Setup and Operation

Follow this step to implement the Tx side of the Partial CRC-32 Operation:

1 Initialize Tx CRC data entry in the transmit VC Table as follows: *CRC_Len*=0, and *PCRC*=0.

All cells except for the last end of message (EOM) are implemented via the SDP, as follows:

1 CPRC make the cell available to SDP for transmission.

2 TxByte programmable processor calculates PCRC as it transmits the cell.

3 TxByte programmable processor sends TLU *XOR* command request containing PCRC; TLU *XORs* with the PCRC, performs 48Byte shift and sends command to store this in the table as the (new accumulated PCRC).

TLU's *XOR* command settings are indicated here:

Bit Position	63 60	59 58	57 55	54 48	47 32	31 24	23 0
Field Name/Setting	CMD=7	CRC= 0x01	VTBL#= User Defined	OFF=User Defined Offset	N/A	MSK=0x0f	IDX=User Defined Index
Optional	Reserved					PCRC=From TxByte Calculation	

Last cell end of message (EOM) is implemented, as follows:

1 CPRC sends TLU *XOR* command (prior to giving last cell to SDP). The TLU does a lookup into the CRC-32 Correction Table indexed by the *CRC_Len* field (which is equivalent to the number of cells in the PDU -1). The TLU XORs this with the (accumulated) PCRC and returns the PCRC (accumulated for all cells except the last) to the CPRC via the Ring Bus. The TLU sets *PCRC*= 0, *CRC_Len*= 0 (based upon data field) after returning the (accumulated PCRC) in the Partial CRC-32 Data Entry Format. The TLU sets the error bit in the Ring Bus register.

TLU's *XOR* command settings are indicated here:

Bit Position	63 60	59 58	57 55	54 48	47 32	31 24	23 0
Field Name/Setting	CMD=7	CRC= 0x10	VTBL#= User Defined	OFF=User Defined Offset	N/A	MSK=0x0f	IDX=User Defined Index
Optional	Reserved					PCRC=0	

2 CPRC writes (accumulated) PCRC to its merge register space.

3 The TxByte programmable processor accumulates a PCRC for the last cell. It then *XORs* this value with the (accumulated PCRC) from its CPRC's merge space, performs the 1's complement and appends to the frame. SDP transmits cell with correct CRC.

Single cell packets are implemented, as follows:

1 CPRC writes PCRC Accumulated= 0xffffffff to merge register space.

Chapter 7

QUEUE MANAGEMENT UNIT

Chapter Overview

This chapter covers the following topics:

Queue Management Unit (QMU) Overview

The Queue Management Unit (QMU) provides queueing service to all the processors (CPs, XP and FP) on the C-5e NP. The QMU queues are used by the processors to switch *payload descriptors* from input processors (CPs, XP and FP) to output processors (CPs, XP and FP) using Control Blocks (WrCB0_ or RdCB0_) via the Payload Bus.

The C-5e NP processors generate the descriptor data in their respective (DMEM), then write the data into a queue stored in the SRAM. The configurable QMU performs queue management while simply passing the descriptor data through without modification; it does not parse the data records that it enqueues.

The QMU provides up to five-hundred-twelve (512) queues using an on-chip memory (internal SRAM) for control structures and off-chip memory (external SRAM) for descriptor storage.

The C-5e provides two modes for managing queues. They consist of:

- Internal Mode (using the internal QMU only).
- External Mode. Refer to "External Scheduler Mode" on page 457.

Warning: *Although the C-5e NP provides an External QMU Mode, it does not support an external traffic manager device. In addition, the associated registers and fields should not be implemented.*

Queues can be allocated asymmetrically to processors, such that one (1) CP could have zero (0) to one-hundred-twenty-eight (128) queues. Up to 16,384 descriptor buffers can be enqueued simultaneously across all queues.

Generally, the data types enqueued in the QMU are either:

- A payload descriptor including a payload buffer tag (BTag), or
- A user-defined inter-processor message.

Payload Descriptors Enqueued to the QMU

Payload Descriptors are small fixed-size (12, 16, 24, or 32Bytes) data structures that contain all the information required to complete the forwarding of a received payload data unit from the ingress processor, for example, the information required to build a header at the output interface. Payload descriptors are created by the application program running on the ingress processor, generally a Channel Processor RISC Core (CPRC).

Typically, the QMU queues are used as egress queues associated with specific data for egress processors, (CPs, the XP, or the FP). Only one (1) egress processor can drain each queue, but any of the ingress processors can write into any queue.

User-Defined Inter-processor Messages Enqueued to the QMU

Inter-processor messages are also small fixed-sized (12, 16, 24, or 32Bytes) data structures that contain user defined information. Generally, inter-processor messages are used to orchestrate control plane activities such as flow control, statistics gathering, or table maintenance. For example, an ingress processor could build a message then enqueue it to a queue serviced by the XP for off-line processing.

QMU Major Components

The major components of the QMU are listed inTable 117 on page 407. In addition, Figure 89 on page 409 shows the QMU Block Diagram.

Table 117 Major Components of the QMU and Their Functions

ITEM	FUNCTION
Queue Management Engine	The Queue Management Engine (QME) manages all descriptor queues in SRAM and maintains per-queue status information (descriptor weight, queue weight, and queue length) that is delivered with each descriptor to the dequeueing processor (CPs, XP, or FP). The QME provides multicast elaboration for descriptors that must be replicated to more than one queue. A single buffer descriptor can be enqueued to any number of output processors (CPs, XP, or FP) with a single command (multicast enqueue) from the descriptor generator (CPs, or XP). The descriptor is enqueued at a specified queue level at each listed output port so that each port receives a copy of the descriptor. The QME implements and operates on a link-list of descriptors. Every descriptor buffer in the QMU starts out as an entry on a free-descriptor buffer list. Then during an enqueue operation, a descriptor buffer is removed from the free-descriptor buffer list and the payload descriptor is copied into a queue. The reverse process happens for dequeue operations. The QME must perform (2) two functions to respond to a processor's (CPs, XP, or FP) enqueue or dequeue request. First, the descriptor must be stored in or retrieved from the external QMU SRAM array. Second, the internal queue controls inside the internal SRAM must be updated to add the entry onto or remove the entry from the desired queue.

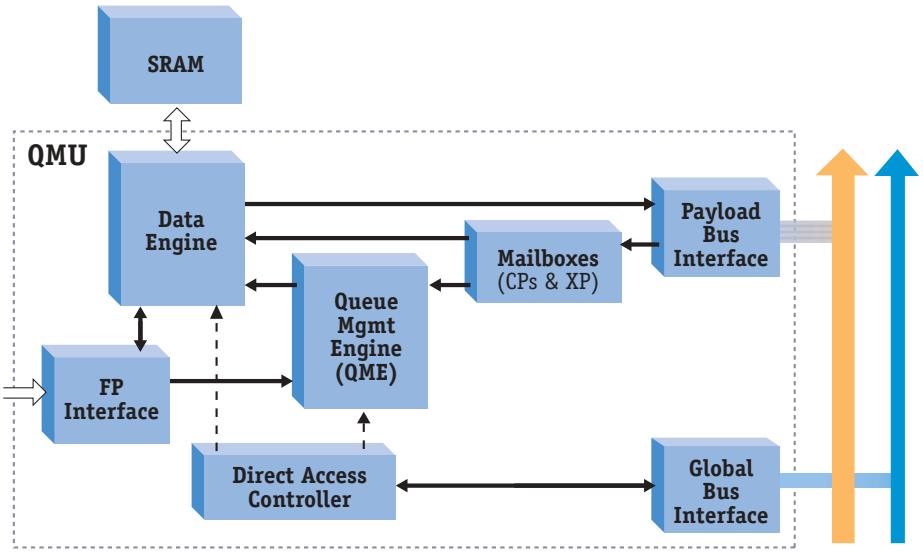
Table 117 Major Components of the QMU and Their Functions (continued)

ITEM	FUNCTION
Direct Access Controller	Initialization, queue configuration changes, and read/write of the QMU internal registers and QMU external memory are performed using loads and stores on the Global Bus. If the transaction is a read, the Direct Access Controller (DAC) gathers the value from the addressed location in the QMU and returns it to the requesting processor on the Global Bus. When one of the QMU's queues state changes from empty to non-empty as the result of an enqueue operation, the Queue Ready Generator (QRG) takes the queue number, determines the processor to notify, then generates the appropriate message.
Mailboxes	Multi-Use Control Block (WrCB0_, RdCB0_) transfers from CPs and the XP arrive at the QMU over the Payload Bus and are held in a Mailbox. Each processor (CPs, XP) has its own Write Mailbox (CP*n* Wr MailBox) and its own Read Mailbox (CP*n* Rd Mailbox). Each mailbox holds a single command used to perform specific operations. Queue status and dequeue operations are performed using (RdCB0_). Unicast enqueue, multicast enqueue, and configure queue operations are performed using (WrCB0_). Each of the two (2) mailboxes for each CP or the XP operate independently of the other. They each have their own available/busy status and success/fail status information reported to the CP or XP independently.
Fabric Port Interface	The Fabric Port has a separate interface to the QMU because of its very high throughput requirement. When the FP has queueing operations pending, it receives no less than half the QMU's descriptor throughput. A dedicated path is used to write FP queueing operations from the FP to the QMU's FP command FIFO. The QMU returns descriptors to the FP over the Payload Bus. Queue Ready Generator (QRG) reports from the QMU are sent to the FP, just like those for any of the CPs or the XP. The QMU buffers up to eight (8) enqueue operations and eight (8) dequeue operations for the FP.
Data Engine	The Data Engine moves payload descriptors or user-defined inter-processor messages on and off the queues stored in external SRAM. This interface adjusts the timing of the internal information (address and data) so that the setup and hold times on the external memory interface are met for both memory writes and reads.
External SRAM	The QMU uses external ZBT SRAM to hold the data enqueued in the QMU queues. The interface is 32bits wide. The memory is organized in power-of-2 sized blocks big enough to hold the configured descriptors.
Internal SRAM	Contains queue configuration information that includes descriptor link-list, descriptor weight, descriptor dynamic memory, queue head-tail, queue length, queue weight, queue parameters, and dynamic descriptor Pool*n* usage.

Table 117 Major Components of the QMU and Their Functions (continued)

ITEM	FUNCTION
Configuration Registers	Used for mapping of queues to CPs, XP and FP, configuration of the QMU, QMU debugging, and collecting QMU statistics.

Figure 89 QMU Block Diagram



QMU Flow Process

The QMU flow is described in this section.

Flow Details for CPs/XP Inputs and FP Inputs

The front part of the input flow is a little different between the CPs/XP and the FP. However, the activities occurring from the Command Processor on are the same for both the CPs/XP and FP. The entire CPs/XP input flow is detailed first, then the front part of the FP later.

CPs and XP Input Flow

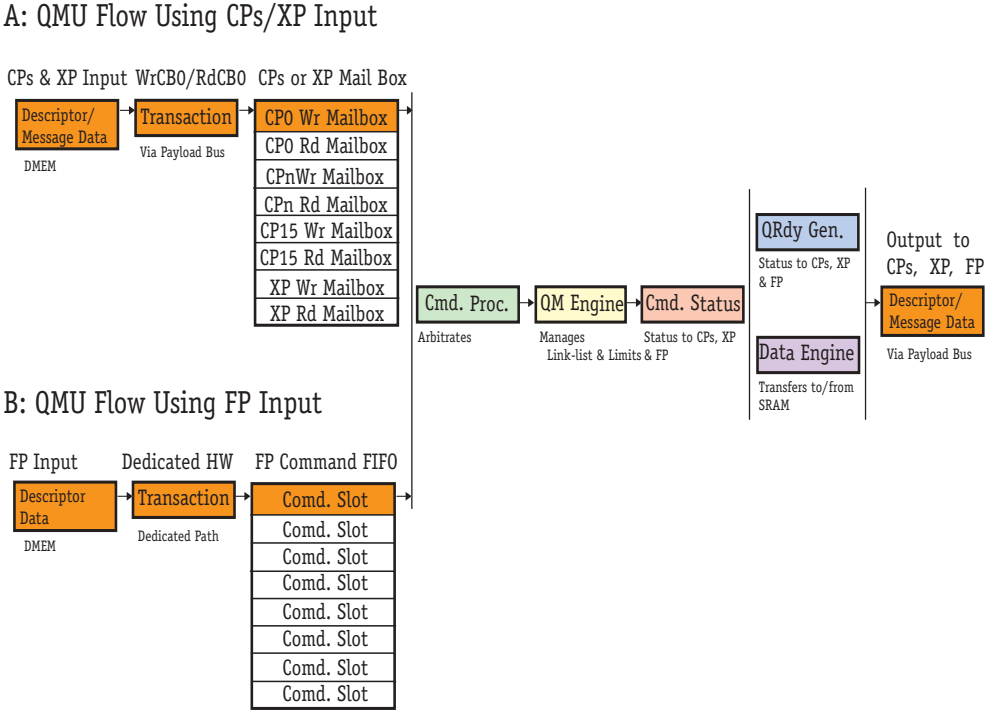
Each processor (CPs, XP) writes the descriptor or message data (user-defined inter-processor message) into its DMEM, then writes the data and a command using a Control Block (WrCB0_) to the processors' dedicated mailbox (CP*n* Wr Mailbox) via the Payload Bus. This data remains in this holding register until the command processor (Cmd. Proc.) arbitrates to determine which mailbox to execute next. The command status generator (CSG) monitors the status of each mailbox throughout the process and reports that status back to the processor. The mailbox empty or non-empty state is used to determine when the applicable mailbox is available for a new operation.

The queueing management engine (QME), upon receipt of the command, manages the free descriptor buffer list, queueing, and storage of queue data in the SRAM using a link-list. A link-list tracks the free descriptor buffers, used descriptor buffers for queueing, and the location of the data (descriptor data) in queues in the SRAM. The queue ready generator (QRG), upon receipt of the data, notifies the processor that the queue is non-empty. It determines the correct processor using the queue number and sends the message over the Payload Bus. The data engine (DE), upon direction of the queue engine (QE), physically transfers the data to/from the external SRAM. During a write (enqueue operation), the data engine (DE) reads the content of the (CP*n* Wr Mailbox) that holds the data, then writes that data into SRAM per the queueing engine (QE) and its link-list. During a read (dequeue operation), the data engine (DE) reads the data (descriptor data) from the SRAM per the queueing engine (QE) link-list, then FIFOs them to transmit back to the CPs/XP using the (RdCB0_) via Payload Bus.

FP Input Flow

The FP writes the descriptor data and a command into its (DMEM), then writes the data and a command using a dedicated interface to the FP command FIFO in the QMU. This data is held in a FP command FIFO (FPCFIFO) until the command processor (Cmd. Proc.) arbitrates to determine which holding register to execute next. At this point, the rest of the FP input flow is identical to the CPs/XP flow starting with the command status generator (CSG), as described in "CPs and XP Input Flow" on page 410.

Figure 90 QMU Flow Diagram



Queue Organization

The queue memory consists of both external and internal SRAM. Each is described here.

External SRAM

The queue memory's external SRAM is organized in the following manner:

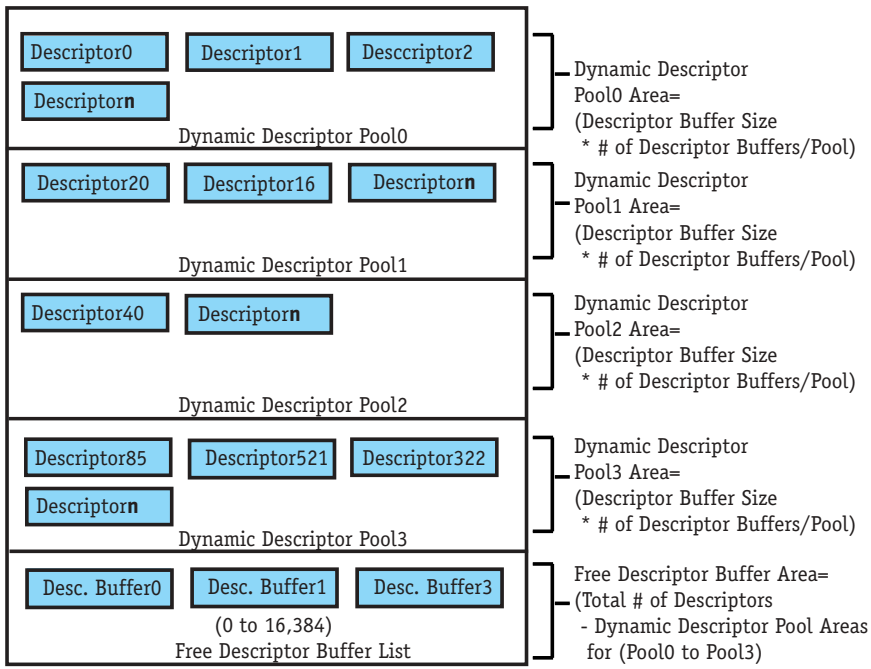
Descriptor Buffer

Descriptor buffers are fixed-sized data structures that are written with payload descriptors and linked to a queue during an enqueue operation. Descriptors are stored in the descriptor buffers located in the QMU's external (SRAM). The QMU allows (1 to 16,384) total descriptor buffers using the *Num_Descriptors* register. The Descriptor buffer sizes supported are (12, 16, 24, or 32Bytes), using the *Descriptor_Size* register. Refer to Figure 91 on page 413, and Table 119 on page 416.

Dynamic Descriptor Pools

The QMU maintains sections of SRAM called *dynamic descriptor pools* containing descriptors. Pools are intended to provide protection among the many users of the queues. Up to four (4) dynamic descriptor pools can be configured. Each Dynamic Descriptor Pool Area= (Descriptor Buffer Size * Number of Descriptors per Pool). Each Dynamic Descriptor Pool*n* can grow to contain a number of descriptors, the legal range is (0 to 16K). A limit is used to guarantee a maximum number of descriptor buffers a queue may hold within the (0 to 16K) range. This limit is implemented using a Dynamic Descriptor Usage Limit Pool. There are four (4) of these registers: *Dyn_Des_Usage_Lim_Pool0* to *Dyn_Des_Usage_Lim_Pool3*. Each Dynamic Descriptor Pool has an associated Dynamic Descriptor Usage Limit Pool. Refer to Figure 91 on page 413 and Table 119 on page 416.

Figure 91 External SRAM Storage Space for Descriptor Buffer Data



Dynamic Descriptor Usage Limit PooIn

During a unicast enqueue to a queue, permission to take the next free descriptor buffer and enqueue it on the queue is based on the state of that queue's Dynamic Descriptor Usage Limit. Each queue is assigned to one (1) of four (4) registers: *Dyn_Des_Usage_Lim_Pool0* to *Dyn_Des_Usage_Lim_Pool3*. Each pool has a usage counter and a usage limit parameter. The usage count is not allowed to exceed the usage limit during unicast enqueues.

The idea behind the Dynamic Descriptor Buffer Usage Limit pools is to provide separation between service classes that need to use dynamic buffering. For example, the use of dynamic buffers by ATM's Variable Bit Rate (VBR) service should not impact the availability of dynamic buffers used by the Constant Bit Rate (CBR) service. The limit on each dynamic pool's descriptor usage is individually configurable because different services require different degrees of traffic variability (burstyness).

When an enqueue is requested to a queue that is in its dynamic range, the queue's pool descriptor usage is compared to that pool's usage limit. If the current pool usage is below the pool's limit, the enqueue is done and the pool usage count is incremented. When that descriptor is later dequeued, the pool usage count is decremented. Refer to "Queue Length Allowance and Length Limit Parameters" on page 420.

When the QMU is initialized, the pool usage limits, the total number of descriptors, and the allowances of all the initialized queues must be configured to work together. The total number of Descriptors allocated among all four (4) pools of the *Dyn_Des_Usage_Lim_Pool0* to *Dyn_Des_Usage_Lim_Pool3* registers should be < the number of dynamically enqueued descriptors. This way, there will be no depletions of the supply of free descriptor buffers.

All queues that are members of a single multicast group should share the same pool. See "Multicast Operation Throughput Considerations" on page 445 for more information.

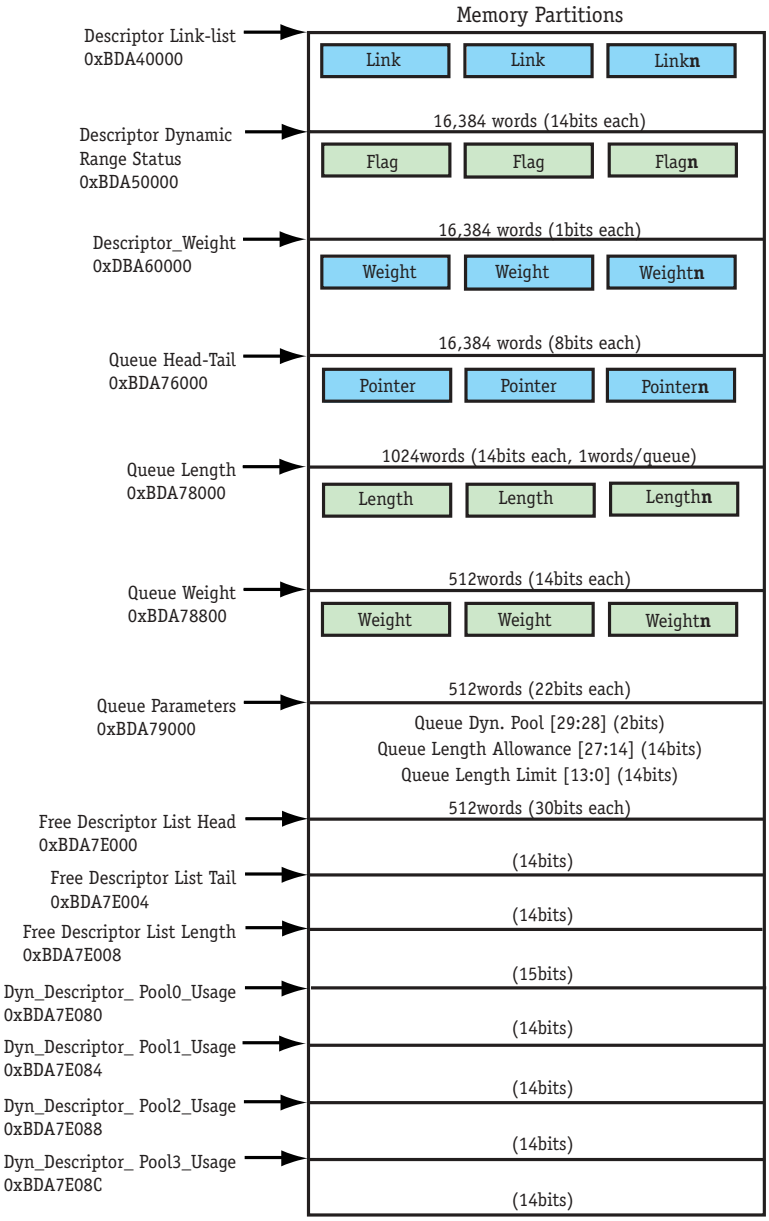
Internal SRAM

The queue memory's internal SRAM contains all the data structures required to build the queues, maintain queue status, and provide descriptor buffer management. The internal memory is divided into the following sub-sections. Refer to Figure 92 on page 415, and Table 118 on page 414.

Table 118 QMU Internal SRAM Sub-Sections and Their Functions

MEMORY SUB-SECTIONS	FUNCTION
Descriptor Link-list	Descriptor links that form the queues.
Descriptor Dynamic	One (1) bit per descriptor that flags wether the descriptor was allocated in a queue's dynamic range.
Descriptor Weight	The weight of each descriptor.
Queue Head-Tail	Pointers to each queue's head and tail.
Queue Length	The length of each queue.
Queue Weight	The accumulated weight of each queue.
Queue Parameters	Configuration for each queue.
Free Descriptor List Head	Pointer to free descriptor list head.
Free Descriptor List Tail	Pointer to free descriptor list tail.
Free Descriptor List Length	The length of free descriptor list.
Dynamic Descriptor Pool0 Usage to Dynamic Descriptor Pool3	The number of dynamically enqueued descriptors in each pool.

Figure 92 Internal SRAM Space



QMU Variables

The QMU uses the following variables shown in Table 119 on page 416.

Table 119 Legal Ranges for SRAM Variables

<table>
<tr><th>ITEM</th><th>RANGE</th></tr>
<tr><td>Number of Dynamic Descriptor Buffer Pools</td><td>Up to 4</td></tr>
<tr><td>Number of Descriptor Buffers per Pool</td><td>0 to 16K</td></tr>
<tr><td>Number of Descriptors allowed in the QMU</td><td>1 to 16,384 as detailed here:
<table><tr><th>PROGRAMMED VALUE</th><th>NUMBER OF DESCRIPTORS</th></tr><tr><td>0
•
•
•
16,383</td><td>1
•
•
•
16,384</td></tr></table></td></tr>
<tr><td>Individual Descriptor Size</td><td><table><tr><th>SIZE (BYTES)</th><th>ENCODED VALUE</th></tr><tr><td>12</td><td>0</td></tr><tr><td>16</td><td>1</td></tr><tr><td>24</td><td>2</td></tr><tr><td>32</td><td>3</td></tr></table></td></tr>
<tr><td>Number of Descriptors Enqueued Dynamically to the Queues Associated with a Pool</td><td>0 to 16K</td></tr>
</table>

Table 119 Legal Ranges for SRAM Variables (continued)

<table>
<tr><th>ITEM</th><th>RANGE</th></tr>
<tr><td>Number of Queues Allowed in the QMU</td><td>0 to 511 as detailed here:
<table><tr><th>PROGRAMMED VALUE</th><th>NUMBER OF QUEUES</th></tr><tr><td>0
•
•
•
511</td><td>1
•
•
•
512</td></tr></table></td></tr>
<tr><td>Number of Queues Mapped per Processor</td><td>Up to 128</td></tr>
<tr><td>Queue Level Number (for Multicast Enqueue Operations Only)</td><td>0 to 7</td></tr>
</table>

Queue Mapping and Parameter Characteristics

The basic queue characteristics are described in this section.

Queue to Processor Mapping

A queue is a FIFO that contains descriptor data.The QMU supports (0 to 511) queues available to the entire C-5e NP. Up to 128 (of those 512) queues can be mapped to a given processor, (CPs, XP, or FP).

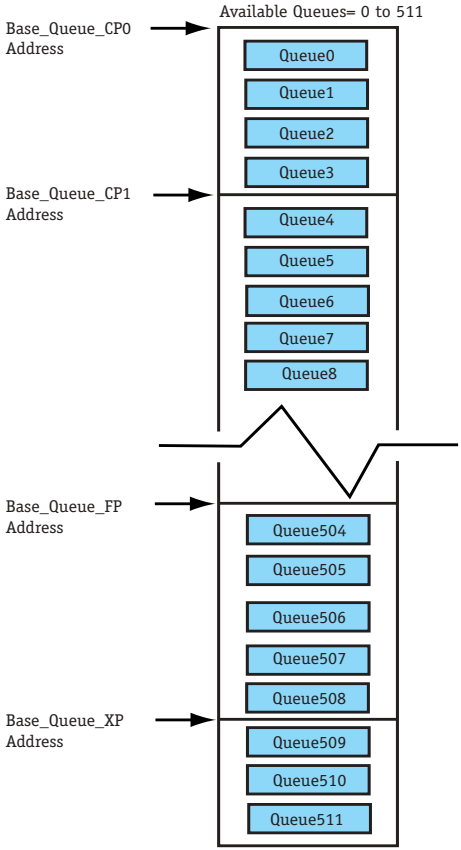
The QMU uses a simple mapping scheme to establish a flexible association between processors and their queues for both unicast enqueue and multicast enqueue operations.

The QMU maps individual queue numbers (0 to 511) to their respective processors (CPs, XP, or FP) using the *Base_Queue_CP0* to *Base_Queue_CP15*, *Base_Queue_FP*, and *Base_Queue_XP* registers. Specifically, using the *Base_Queue_Number* bits [8:0] field contained in the *Base_Queue_CP0* to *Base_Queue_CP15*, *Base_Queue_FP*, and *Base_Queue_XP* registers.

The *Base_Queue_nnn* is the lowest numbered queue owned by that processor. The rest of a processor's queues are located at offsets above the *Base_Queue_nnn*. The offset range cannot be larger than one-hundred-twenty-eight (128). Each processor is assigned its queues sequentially within the (0 to 511) queue number range.

These queues are the ones targeted by unicast and multicast enqueues to that processor, and are the queues that stimulate queue non-empty transition notifications to the processor. This scheme allows an arbitrary number of queues to be assigned to each processor (the *Base_Queue_ Number* bits [8:0] fields are unconstrained). Refer to Figure 93 on page 419.

Figure 93 Mapping Queues to Processors for Unicast/ Multicast Enqueue Operations Example



Queue to Processor Mapping Rules

The assignment of queues to processors must follow these rules:

- The ranges of queues starts with queue 0 for CP0.
- The range for CP1 follows immediately after the range for CP0 with no gap. This pattern continues through the remaining CPs in increasing CP number, and then to the FP and the XP.
- The last XP queue must not exceed the total number queues initialized.
- There can be no more than one-hundred-twenty-eight (128) queues per processor.

When there are no queues assigned to a processor, its *Base_Queue_nnn* register must still be programmed. Its value should be the offset for the "next available queue" following the last queue of the next lower numbered processor.

If a processor is assigned zero (0) queues, the next higher processor (based on Processor ID) is assigned the same Base_Queue_Number bits [8:0] as the lower processor. Therefore, two (or more) processors can have the same Base_Queue_Number bits [8:0] value if the lower processor(s) has been assigned zero (0) queues.

If any descriptors are enqueued for a processor with no queues, those descriptors are enqueued in a queue of the higher numbered processor. When one of the next-higher numbered processor's queues is non-empty, the notification is sent to the highest numbered processor sharing the *Base_Queue_Number* bits [8:0] value, but the higher numbered processor becomes the owner of the queue.

Queue Length Allowance and Length Limit Parameters

The supply of Descriptor Buffers within the QMU must be properly managed to prevent the depletion of Descriptor Buffers for queues not at the "traffic hot-spots". To avoid this situation, a minimum number of Descriptor Buffers always needs to be available to each queue regardless of the use of Descriptor Buffers by other queues. Two (2) parameters are used to set the minimum and maximum amount of Descriptor Buffers used to provide a free flow of Descriptor Buffers to the queues that need them at a given point in time. They are: *Queue Length Allowance* and the *Queue Length Limit*. Allowance is the minimum amount of the (0 to 16K) range, whereas, Limit is the maximum of the (0 to 16K) range. Thus, each queue is initialized with these two (2) parameters:

- **Queue Length Allowance** — Is the guaranteed minimum allocation of Descriptor Buffers of (0 to 16K) range. Buffers are implicitly reserved so that a queue can always fill to its allowance. The sum of the allowances of all the queues is always smaller than the total supply of Descriptor Buffers, typically much smaller.
- **Queue Length Limit** — Is the guaranteed maximum number of unicast Descriptor Buffers a queue is allowed to hold in the range of (0 to 16K). The sum of the limits of all the queues is typically much greater than the supply of Descriptor Buffers.

These parameters (Allowance and Limit) refer to the number *of Descriptors enqueued in a queue; they do not specify which Descriptor Buffers are enqueued in any given queue.*

When a queue has filled to an amount between its Allowance and its Limit, it is in its *dynamic range*. In this *dynamic range*, Descriptor Buffers may or may not be available to the queue for enqueueing depending on the use of Descriptor Buffers by other queues.

Whether or not a Descriptor Buffer is available in this situation depends on the number of Descriptor Buffers currently enqueued in other queues.

Descriptor Buffer availability is dynamic in that it depends on the current state of the QMU that results from the dynamic traffic pattern. If other queues are not using many Descriptor Buffers at a given moment in time, a queue at the “traffic hot-spot” can get more than its share. If there are a lot of enqueued Descriptor Buffers, each queue gets at least its Allowance, (that is, its fair share under congestion). This gives the QMU the appearance of many more Descriptor Buffers than it actually has if they had been dedicated to specific queues.

Queueing Operations

The QMU receives queue operation requests via the Payload Bus. Enqueue Operations use a control block (WrCB0) to write the descriptor data into a queue in the QMU's external SRAM from the DMEM of either the requesting CP, or XP via the Payload Bus. In addition, Dequeue Operations use a control block (RdCB0) to read descriptor data from a queue in the QMU's SRAM into the DMEM of the requesting CP, or XP via the Payload Bus. Descriptor data is transferred to/from the QMU in 16Byte units; up to two (2) units (32Bytes) can be transferred in one (1) Payload operation along with other information contained in the payload transaction.

QMU Run Enable

The QMU must be enabled to process queueing operation requests from the processors (CPs, XP, or FP). The *QMU_Run_Enable* register has one (1) bit that enables the QMU to process Payload Bus operations. This bit must be set after initialization to allow the QMU to go online.

This bit can be cleared at any time. Once cleared, this bit disables the QMU from starting the processing of any subsequent queue operations. A queue operation currently being executed is not interrupted by the clearing of this bit, but no subsequent queue operations are started. This allows a clean shutdown of the QMU so that its internal state is not corrupted. Once the QMU is shut down, a processor (CPs, XP, or FP) can access all of the internal state of the QMU (both registers and memory). Any queue commands left in the mailboxes (Wr Mailbox/Rd Mailbox) are executed in normal order when the QMU is re-enabled, just as if the QMU had never been disabled. Refer to "QMU_Run_Enable Register (QMU Enable Queue Function)" on page 637.

Enqueue Operation

Processors (CPs, XP, or FP) enqueue payload descriptors to designate to the output processor how to forward packets/cells. Upon a successful enqueue, the requesting processor is freed from having to keep track of the packet/cell buffer. It has effectively handed this packet/cell buffer off to the output function by way of the QMU.

Payload (Wr/Rd) Servicing Order During Enqueue Operation

Queue operations arriving from the CPs and the XP are serviced from the mailboxes by type (Wr/Rd) in the order of their arrival at the QMU over the Payload Bus. Payload write (Wr) operations from any CP or the XP are executed in arrival order before any waiting payload read (Rd) operations are executed in their arrival order. In order to facilitate the FP operating at full capacity, the QME gives FP commands at least one half the queueing bandwidth in proportion to its traffic bandwidth. The CP/XP operations, either read or write operations, are interleaved with FP operations.

Since only one (1) command of each type (Wr/Rd) can be buffered in the QMU from each of the CPs and the XP, a mailbox status for each ensures that commands are not lost. Direct wiring of mailbox status and command execution status information from the QMU to the *CP_Mode0* register ensures that the CPs do not need to spend system-level resources to determine if the previous command completed successfully or not.

An attempt to over-write a busy mailbox causes a Payload Bus NACK. An attempt does not disturb the previously written command.

Causes of Enqueue Failure

When an enqueue operation failure occurs, the requesting processor (CPs, XP, or FP) must either retry the enqueue at a later time or drop the packet/cell and reuse the Buffer Tag (BTag). An enqueue operation can fail for the following reasons:

- A *Dyn_Descriptor_Pooln_Usage* shows that the pool is empty when a queue is in its dynamic range.
- The *Free_Descriptor_Buffer_List* has no descriptor buffers left.
- The unicast target queue has exceeded its buffer-use limit.
- The target queue does not exist (that is, it was never configured).

Dequeue Operation

Because the QMU queues are output queues, there must be a single processor that owns each active queue. That owner is responsible for draining the queue. A successful dequeue operation triggers the entire forwarding process. Therefore, when a dequeue failure occurs, there is no descriptor to drive a packet/cell transmission.

Queue Servicing Policy During Dequeueing Operation

The dequeueing processors (CPs, XP, or FP) determine the queue service policy, not the QMU. In addition, several queues can be assigned to each CP for Quality of Service (QoS) purposes. The dequeueing processor is free to implement any priority, service weighting, or scheduling policy.

Dequeueing processors are responsible for scheduling their own workloads. To aid in this, the QMU provides *weights* for each queue. Each queue entry has a weighting factor associated with it. In typical frame-switching applications, each descriptor is assigned a weight which is related to the length of the frame stored in the associated BMU buffer. The sum of the enqueued descriptor weights is kept for each QMU queue.

Each enqueue and dequeue operation updates the value for its queue. For multicast enqueue operations, the queue weight is adjusted for all the specified queues.

When a descriptor is dequeued from a queue, the descriptor's weight, the queue's new weight, and the queue's length are returned to the processor's DMEM along with the descriptor itself. Refer to "Dequeue Operation" on page 441.

Causes of Dequeue Failures

A dequeue operation can fail for these reasons:

- The target queue is empty.
- The target queue does not exist (that is, it was never configured).

Status Reporting

Status is reported on Mailboxes and Queues. Queue status information is made up of three (3) types: Empty to Non-Empty State Notification, Dequeue Operation, and Buffer Management.

Mailbox Availability and Status Reporting of Mailboxes

The QMU maintains a write (CP*n* Wr Mailbox) and a read (CP*n* Rd Mailbox) mailbox for each CP and the XP (XP Wr Mailbox, and XP Rd Mailbox). These mailboxes are invisible to application developers except that the state of each is reported via the *CP_Mode0* register for the processor. Specifically, the *CP_Mode0* register, bits [23:22] *QMURdMbxStatus* field and bits [21:20] *QMUWrMbxStatus* field are visible. Using these two (2), two (2) bit fields, status states reported include: operation success, operation error, busy-wait, or busy-executing. The mailbox scheme provides a single holding register per source (that is, only one (1) command is outstanding per mailbox at a time).

This allows success/fail operation information to be conveyed back to the requesting processor (CPs or XP) in an orderly fashion. Errors reported (set to 1) in the *CP_Mode0* register occur when the previous queue operation encountered an error and should be repeated. Generally, it means the queue operation failed, because of a resource error such as queue non-empty or descriptor buffer pool limit was exceeded.

Mailbox status is also reported through both *Event0* and *Event1* registers. Specifically, *Event0* register bit [60] *QMUError* field, as well as the *Event1* register bit [31] *QRdMbxAvail* field, bit [26] *QRdMbxBusy* field, bit [15] *QWrMbxAvail* field, and bit [10] *QWrMbxBusy* field are used to determine when the applicable mailbox is available for a new queue. The avail bits indicate the mailbox made a state transition from busy to empty.

The busy bits indicate a transition from empty to busy. The error bit indicates a transition from busy to error.

Queue Status Information

There are several types of queue status information that can be accessed to inform the processors (CP, XP, or FP) in their forwarding tasks:

- Queue non-empty transition notifications presented via *Queue_Status0, Queue_Status1, Queue_Status2* and *Queue_Status3* registers, as well as, the *Event0* and *Event1* registers.
- Dequeue status information (the Queue's weight and length) that is included with a descriptor when it is dequeued. Refer to "Dequeue Operation" on page 441.
- Buffer management status that is included in the *Free_Descriptor_Buffer_List* and *Dyn_Descriptor_Pooln_Usage* registers.

Queue Empty to Non-empty State Notification Process Information

Queue status is made visible to the processors (CPs, XP, or FP) via the *Event1* register and the *Queue_Status0*, *Queue_Status1*, *Queue_Status2*, *Queue_Status3* registers in each processor's configuration space. Payload operations (RdCB0) can be used to read the complete queue status from the QMU when the normal updates that come with a dequeue are insufficient. Refer to "Queue Status Operation" on page 433. Autonomous announcements of queue status by the QMU are made on the Global Bus and are handled by CP and XP hardware.

When a queue becomes non-empty, its dequeueing processor (CP, XP, or FP) can begin to drain it. The QMU announces this change of state (empty to non-empty) to the queue's dequeueing processor indicating the queue is now ready to be serviced. The dequeueing processor's software is then responsible to keep track of the ready status of the queue until the queue is completely drained.

Queue change of state (empty to non-empty) notifications are issued whether the enqueued descriptor came from a unicast or a multicast enqueue. Each multicast target queue is treated individually in the generation of these notifications.

Queue status non-empty transitions are automatically loaded into the four (4) queue status registers, where they can be read by the CPRC. *Queue_Status0*, *Queue_Status1, Queue_Status2, Queue_Status3* bits are set to one (1) when a queue state changes from empty to non-empty.

The dequeueing processor is responsible for clearing these bits when it handles the status change (that is, before it begins to read descriptors from the queue). The logical OR of the bits in these status registers also provides a level-sensitive event for input to the *Event1* register. Refer to "Queue_Status0 Register (CP Queue Status Function)" on page 549, Table 177 on page 549.

Dequeue Status Information

The dequeueing processor (CPs, XP, or FP) of the descriptors is given a queue and descriptor status with each dequeue operation. The QMU sends the following three (3) types of status information to DMEM along with the descriptor being delivered:

- Descriptor Weight, Queue Weight, and Queue Length
  - The Descriptor Weight — The weight of that descriptor.
  - The Queue Weight — Accumulated weight of the queue after the dequeue of this descriptor.
  - Queue Length — The queue length after the dequeue of this descriptor.

The queue status information can also be obtained using the Queue Status Operation. Refer to"Queue Status Operation" on page 433.

Buffer Management Information

The processor receives an update of the queue's length with each descriptor read (RdCB0 for Dequeue Operation and Queue Status Operations). When the reported queue length drops to zero (0), the processor has just received the last enqueued descriptor and should stop issuing descriptor reads until it receives a queue non-empty transition notification for that queue.

Additional QMU status information can be read during on-line operation via Global Bus transactions using Load operations from an egress processor. These Global reads deliver the values of these registers instantaneously. Their values typically vary (very rapidly and widely) when the QMU is active. Some form of sampling and time-averaging is necessary to get an accurate sense of the congestion level.

The QMU makes available status information about the processors level of congestion. That status information is contained in the *Free_Descriptor_Buffer_List* and *Dyn_Descriptor_Pooln_Usage* registers. These two (2) registers can be read using Global reads when the QMU is on-line to access the status information.

Refer to "Free_Descriptor_Buffer_List Register (QMU Control Function)" on page 652, "Dyn_Descriptor_Pool0_Usage Register (QMU Status Function)" on page 653, and Table 215 on page 653.

Processors can also read the instantaneous contents of the various statistics counters using Loads over the Global Bus. There are sixteen (16) QMU Statistics registers that are available. Refer to "QMU Registers" on page 447.

Types of Transactions

The QMU supports six (6) Queueing functions. The different functions are initiated by CPs or the XP using the Multi-Use Control Blocks by just changing the fields. Multi-Use Control Blocks use the following registers: WrCB0_Sys_Addr, WrCB0_Ctl, WrCB0_DMA_Addr, WrCB0_SDP_Addr; and RdCB0_Sys_Addr, RdCB0_Ctl, RdCB0_DMA_Addr, RdCB0_SDP_Addr. Refer to Table 120 on page 428, Table 121 on page 429, and Table 122 on page 430.

Table 120 Multi-Use Control Blocks (for Wr and Rd)

MODE	CATEGORY	FUNCTION	FIELDS USED	DETAILS
• CP to/from QMU • XP to/from QMU	Queue Management Transactions	Configure Queue	Mail Box#, Queue#, Cmd, PoolID	See "Configure Queue Operation" on page 431.
		Queue Status		See "Queue Status Operation" on page 433.
		Unicast Enqueue		See "Unicast Enqueue Operation" on page 435.
		Speculative Unicast Enqueue		See "Speculative Unicast Enqueue Operation" on page 437.
		Multicast Enqueue	Mail Box#, QueueLevel#, Cmd, PoolID	See "Multicast Enqueue Operation" on page 439.
		Dequeue	Mail Box#, Queue#, Cmd, PoolID	See "Dequeue Operation" on page 441.

Table 121 WrCB0_ Variables per Field for QMU

<table>
<tr><th>REGISTER</th><th>FIELD NAME</th><th>BIT POSITION</th><th>DESCRIPTION</th></tr>
<tr><td>WrCB0_DMA_Addr</td><td>PoolID</td><td>20:16</td><td>PoolID —
<table>
<tr><th>OPERATION TYPE</th><th>VALUE</th></tr>
<tr><td>Configure Queue</td><td>31</td></tr>
<tr><td>Unicast Enqueue</td><td></td></tr>
<tr><td>Multicast Enqueue</td><td></td></tr>
</table></td></tr>
<tr><td rowspan="4">WrCB0_Sys_Addr</td><td>Mail Box#</td><td>28:24</td><td>Mail Box Number —
<table>
<tr><th>OPERATION TYPE</th><th>VALUE</th></tr>
<tr><td>Configure Queue</td><td rowspan="3">Enter the number of the processor’s mailbox. Legal range=0 to 16. Generally, the initiating processors ID.</td></tr>
<tr><td>Unicast Enqueue</td></tr>
<tr><td>Multicast Enqueue</td></tr>
</table></td></tr>
<tr><td>Queue#</td><td>20:12</td><td>Queue Number —
<table>
<tr><th>OPERATION TYPE</th><th>VALUE</th></tr>
<tr><td>Configure Queue</td><td>Queue to configure. Legal range=0 to 511.</td></tr>
<tr><td>Unicast Enqueue</td><td>Queue to write to. Legal range=0 to 511.</td></tr>
</table></td></tr>
<tr><td>QLevel#</td><td>18:16</td><td>Queue Level Number —
<table>
<tr><th>OPERATION TYPE</th><th>VALUE</th></tr>
<tr><td>Multicast Enqueue</td><td>Index to the Queue Number to write to. Legal range=0 to 7.</td></tr>
</table></td></tr>
<tr><td>CMD</td><td>10:8</td><td>Command —
<table>
<tr><th>OPERATION TYPE</th><th>VALUE</th></tr>
<tr><td>Configure Queue</td><td>1</td></tr>
<tr><td>Speculative Unicast Enqueue</td><td>3</td></tr>
<tr><td>Unicast Enqueue</td><td>4</td></tr>
<tr><td>Multicast Enqueue</td><td>6</td></tr>
</table></td></tr>
</table>

Table 122 RdCB0_ Variables per Field for QMU

<table>
<tr><th>REGISTER</th><th>FIELD NAME</th><th>BIT POSITION</th><th>DESCRIPTION</th></tr>
<tr><td>RdCB0_DMA_Addr</td><td>PoolID</td><td>20:16</td><td>PoolID —
<table>
<tr><th>OPERATION TYPE</th><th>VALUE</th></tr>
<tr><td>Queue Status</td><td rowspan="2">31</td></tr>
<tr><td>Dequeue</td></tr>
</table></td></tr>
<tr><td rowspan="3">RdCB0_Sys_Addr</td><td>Mail Box#</td><td>28:24</td><td>Mail Box Number —
<table>
<tr><th>OPERATION TYPE</th><th>VALUE</th></tr>
<tr><td>Queue Status</td><td rowspan="2">Enter the number of the processor’s mailbox. Legal range=0 to 16. Generally, the initiating processors ID.</td></tr>
<tr><td>Dequeue</td></tr>
</table></td></tr>
<tr><td>Queue#</td><td>20:12</td><td>Queue Number—
<table>
<tr><th>OPERATION TYPE</th><th>VALUE</th></tr>
<tr><td>Queue Status</td><td>Queue Status being read. Legal range=0 to 511.</td></tr>
<tr><td>Dequeue</td><td>Queue being Dequeued. Legal range=0 to 511.</td></tr>
</table></td></tr>
<tr><td>CMD</td><td>10:8</td><td>Command —
<table>
<tr><th>OPERATION TYPE</th><th>VALUE</th></tr>
<tr><td>Queue Status</td><td>2</td></tr>
<tr><td>Dequeue</td><td>5</td></tr>
</table></td></tr>
</table>

Queue Management Transactions

Queue transactions consist of six (6) different functions (Configure Queue, Queue Status, Unicast Enqueue, Speculative Unicast Enqueue, Multicast Enqueue, and Dequeue). Queue transactions are invoked using Control Blocks (WrCB0, and RdCB0).

Queue Transaction Functions (Operation and Examples)

Each is described here along with examples.

Configure Queue Operation

Configure Queue uses a control block (WrCB0) to send configuration information from the (DMEM) of either the requesting CP or XP to the QMU's Queue Management Engine (QME).

Configure Queue Example

The bits for WrCB0_Sys_Addr, WrCB0_Ctl, and WrCB0_DMA_Addr are set as shown in Table 123 on page 431.

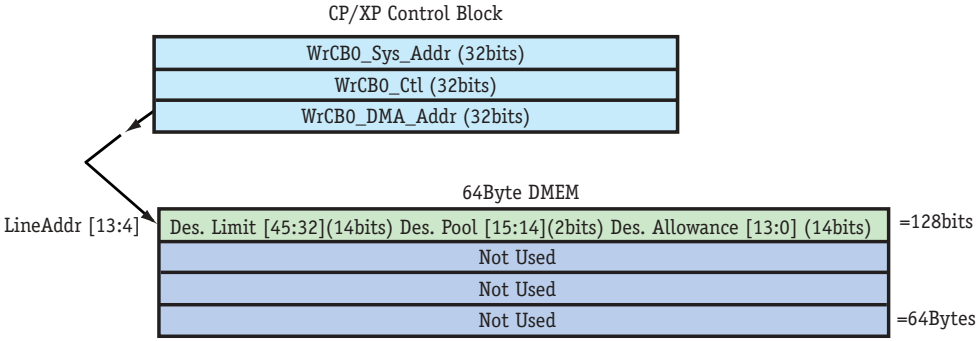
Table 123 WrCB0_ Settings for Configure Queue

REGISTER	FIELD NAME	BIT POSITION	DESCRIPTION
WrCB0_Ctl	Length	13:0	**Length** — Length of DMA transfer in Bytes. LENGTH (BYTES): 64
WrCB0_Sys_Addr	Mail Box#	28:24	**Mail Box Number** — Enter the number of the processor's mailbox. Legal range=0 to 16.
	Queue#	20:12	**Queue Number** — Queue to write to configure. Legal range=0 to 511.
	Cmd	10:8	**Command** — Enter 1 for Configure Queue Operation.
WrCB0_DMA_Addr	PoolID	20:16	**PoolID** — Enter 31 for Queue Operation.
	LineAddr	13:4	**DMEM Line Address** — DMEM 64Byte line address for DMA.

The WrCB0_DMA_Addr bits [13:0] *LineAddr* field refers to the Local DMEM buffer address that is generally 64Byte aligned. The Descriptor Limit [45:32] (14bits), Descriptor Pool [15:14] (2bits) and Descriptor Allowance [13:0] (14bits) are located in the first 128bit line

inside the 64Byte DMEM as shown in Figure 94 on page 432. The second, third and fourth 128bit lines are not used.

Figure 94 Configure Queue Implementation



Refer Table 132 on page 468 *for detail descriptions of these fields.*

Queue Status Operation

Queue Status uses a control block (RdCB0) to read a single queue's length and weight from the QMU into the (DMEM) of the requesting CP, or XP.

Queue Status Example

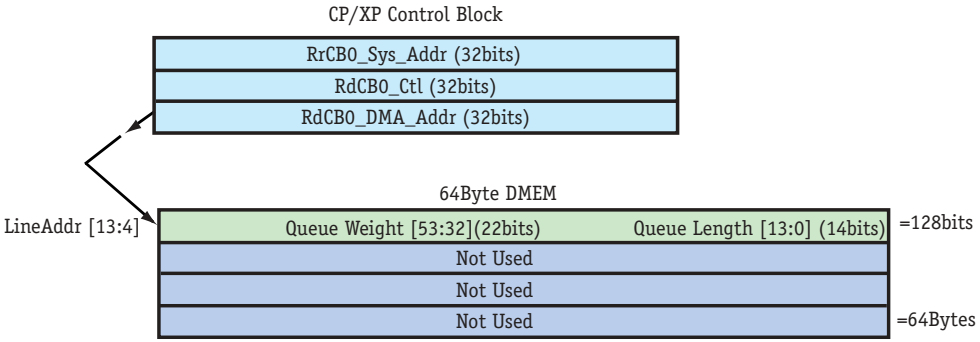
The bits for RdCB0_Sys_Addr, RdCB0_Ctl and RdCB0_DMA_Addr are set as shown in Table 124 on page 433.

Table 124 RdCB0_ Settings for Queue Status

REGISTER	FIELD NAME	BIT POSITION	DESCRIPTION
RdCB0_Ctl	Length	13:4	**Length** — Length of DMA transfer in Bytes. LENGTH (BYTES) 64
RdCB0_Sys_Addr	MailBox#	28:24	**Mail Box Number** — Enter the number of the processor's mailbox. Legal range=0 to 16.
	Queue#	20:12	**Queue Number** — Queue Status being read. Legal range=0 to 511.
	Cmd	10:8	**Command** — Enter 2 for Queue Status Operation.
RdCB0_DMA_Addr	PoolID	20:16	**PoolID** — Enter 31 for Queue Operation.
	LineAddr	13:4	**DMEM Line Address** — DMEM 64Byte line address for DMA.

The RdCB0_DMA_Addr bits [13:4] *LineAddr* field refers to the Local DMEM buffer address that is generally 64Byte aligned. The Queue's Weight [53:32] (22bits) and its Length [13:0] (14bits) are located in the first 128bit line inside the 64Byte DMEM as shown in Figure 95 on page 434. The second, third and fourth 128bit lines are not used.

Figure 95 Queue Status Implementation



Refer Table 132 on page 468 *for detail descriptions of these fields.*

Unicast Enqueue Operation

Unicast Enqueue uses a control block (WrCB0) to write the Descriptor data into a queue in the QMU's (SRAM) from the (DMEM) of either the requesting CP or XP.

Unicast Enqueue Example

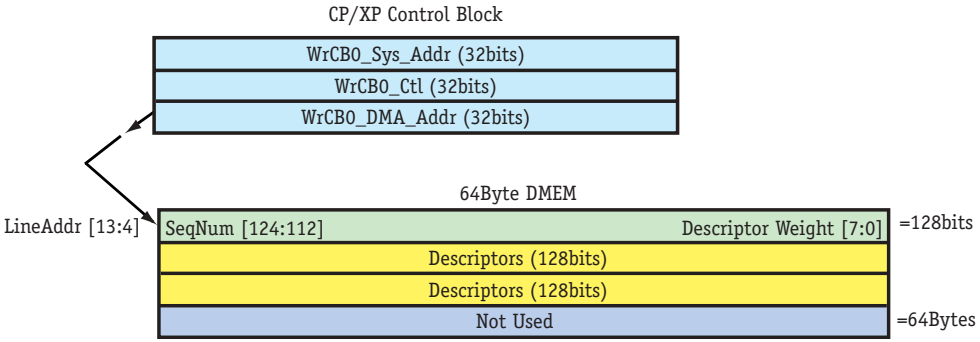
The bits for WrCB0_Sys_Addr, WrCB0_Ctl, and WrCB0_DMA_Addr are set as shown in Table 125 on page 435.

Table 125 WrCB0_ Settings for Unicast Enqueue

REGISTER/	FIELD NAME	BIT POSITION	DESCRIPTION
WrCB0_Ctl	Length	13:0	**Length** — Length of DMA transfer in Bytes. LENGTH (BYTES): 64
WrCB0_Sys_Addr	Mail Box#	28:24	**Mail Box Number** — Enter the number of the processor's mailbox. Legal range=0 to 16.
	Queue#	20:12	**Queue Number** — Queue to write to. Legal range=0 to 511.
	Cmd	10:8	**Command** — Enter 4 for Unicast Enqueue Operation.
WrCB0_DMA_Addr	PoolID	20:16	**PoolID** — Enter 31 for Operation.
	LineAddr	13:4	**DMEM Line Address** — DMEM 64Byte line address for DMA.

The WrCB0_DMA_Addr bits [13:0] *LineAddr* field refers to the Local DMEM buffer address that is generally 64Byte aligned. The Sequence Number [124:112], Descriptor Weight [7:0] are located in the first 128bit line followed with descriptors (depending on 12, 16, 24, or 32Byte sizes) located in the second and third 128bit lines inside the 64Byte DMEM as shown in Figure 96 on page 436. The fourth 128bit line is not used.

Figure 96 Unicast Enqueue Implementation



Refer Table 132 *on page 468* *for detail descriptions of these fields.*

Speculative Unicast Enqueue Operation

Speculative Unicast Enqueue uses a control block (WrCB0) to write the Descriptor data into a queue in the QMU's (SRAM) from the (DMEM) of either the requesting CP or XP.

Speculative Unicast Enqueue Example

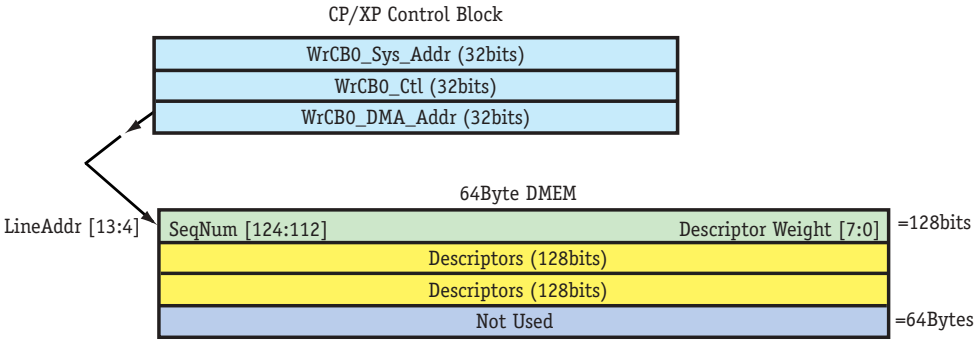
The bits for WrCB0_Sys_Addr, WrCB0_Ctl, and WrCB0_DMA_Addr are set as shown in Table 126 on page 437.

Table 126 WrCB0_ Settings for Speculative Unicast Enqueue

REGISTER/	FIELD NAME	BIT POSITION	DESCRIPTION
WrCB0_Ctl	Length	13:0	**Length** — Length of DMA transfer in Bytes. LENGTH (BYTES): 64
WrCB0_Sys_Addr	Mail Box#	28:24	**Mail Box Number** — Enter the number of the processor's mailbox. Legal range=0 to 16.
	Queue#	20:12	**Queue Number** — Queue to write to. Legal range=0 to 511.
	Cmd	10:8	**Command** — Enter 3 for Speculative Unicast Enqueue Operation.
WrCB0_DMA_Addr	PoolID	20:16	**PoolID** — Enter 31 for Operation.
	LineAddr	13:4	**DMEM Line Address** — DMEM 64Byte line address for DMA.

The WrCB0_DMA_Addr bits [13:0] *LineAddr* field refers to the Local DMEM buffer address that is generally 64Byte aligned. The Sequence Number [124:112], and Descriptor Weight [7:0] are located in the first 128bit line followed with descriptors (depending on 12, 16, 24, or 32Byte sizes) located in the second and third 128bit lines inside the 64Byte DMEM as shown in Figure 107 on page 465. The fourth 128bit line is not used.

Figure 97 Speculative Unicast Enqueue Implementation in Internal Mode



Refer Table 132 on page 468 *for detail descriptions of these fields.*

Multicast Enqueue Operation

Multicast Enqueue uses a control block (WrCB0) to write a Descriptor's data into multiple queues in the QMU's SRAM from the DMEM of either the requesting CP, or XP.

Multicast Enqueue Example

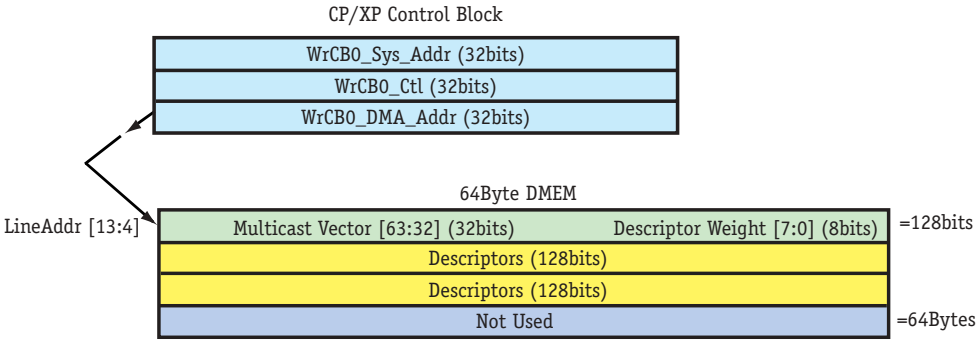
The bits for WrCB0_Sys_Addr, WrCB0_Ctl, and WrCB0_DMA_Addr are set as shown in Table 127 on page 439.

Table 127 WrCB0_ Settings for Multicast Enqueue

REGISTER	FIELD NAME	BIT POSITION	DESCRIPTION
WrCB0_Ctl	Length	13:0	**Length** — Length of DMA transfer in Bytes. LENGTH (BYTES) 64
WrCB0_Sys_Addr	Mail Box#	28:24	**Mail Box Number** — Enter the number of the processor's mailbox. Legal range=0 to 16.
	QLevel#	18:16	**Queue Level Number** — Index to the Queue Number to write to. Legal range=0 to7.
	Cmd	10:8	**Command** — Enter 6 for Multicast Enqueue Operation.
WrCB0_DMA_Addr	PoolID	20:16	**PoolID** — Enter 31 for Operation.
	LineAddr	13:4	**DMEM Line Address** — DMEM 64Byte line address for DMA.

The WrCB0_DMA_Addr bits [13:0] *LineAddr* field refers to the Local DMEM buffer address that is generally 64Byte aligned. The Multicast Vector [63:32], and Descriptor Weight [7:0] are located inside the first 128bit line followed with descriptors (12, 16, 24, or 32Byte sizes) located in the second and third 128bit lines inside the 64Byte DMEM as shown in Figure 98 on page 440. The fourth 128bit line is not used.

Figure 98 Multicast Enqueue Implementation



Refer Table 132 on page 468 *for detail descriptions of these fields.*

Dequeue Operation

Dequeue uses a control block (RdCB0) to read Descriptor data from a queue in the QMU's SRAM into the DMEM of the requesting CP, or XP. Dequeue frees a Descriptor Buffer from a queue.

Dequeue Example

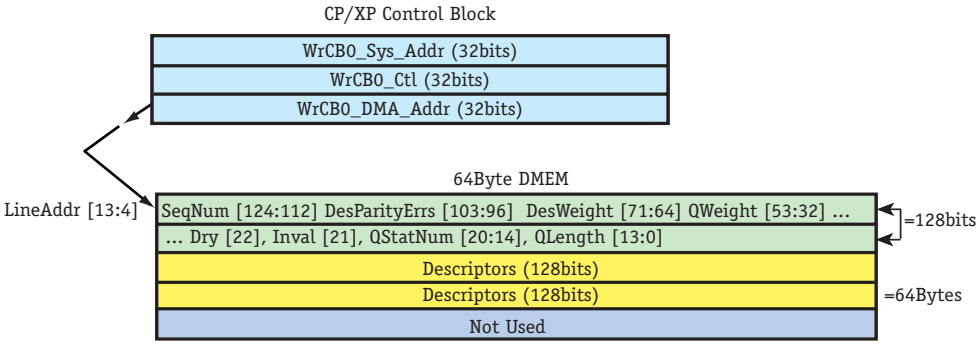
The bits for RdCB0_Sys_Addr, RdCB0_Ctl and RdCB0_DMA_Addr are set as shown in Table 128 on page 441.

Table 128 RdCB0_ Settings for Dequeue

REGISTER	FIELD NAME	BIT POSITION	DESCRIPTION
RdCB0_Ctl	Length	13:4	**Length** — Length of DMA transfer in Bytes. LENGTH (BYTES) 64
RdCB0_Sys_Addr	MailBox#	28:24	**Mail Box Number** — Enter the number of the processor's mailbox. Legal range=0 to 16.
	Queue#	20:12	**Queue Number** — Queue being Dequeued. Legal range=0 to 511.
	Cmd	10:8	**Command** — Enter 5 for Dequeue Operation.
RdCB0_DMA_Addr	PoolID	20:16	**PoolID** — Enter 31 for Queue Operation.
	LineAddr	13:4	**DMEM Line Address** — DMEM 64Byte line address for DMA.

The RdCB0_DMA_Addr bits [13:4] *LineAddr* field refers to the Local DMEM buffer address that is generally 64Byte aligned. The Sequence Number [124:112], Descriptor Parity Errors [103:96], Descriptor Weight [71:64], Queue Weight [53:32], Dry [22], Invalid [21], Queue Status Number [20:14], Queue Length [13:0] (14bits) are located inside the first 128bit line followed with descriptors (depending on 12, 16, 24, or 32Byte sizes) located in the second and third 128bit lines inside the 64Byte DMEM as shown in Figure 99 on page 442. The fourth 128bit line is not used.

Figure 99 Dequeue Implementation



Refer Table 132 on page 468 *for detail descriptions of these fields.*

QMU Multicast Support (Non-System Level)

Multicast enqueue operations take a single multicast descriptor from the DMEM of a processor (CPs, or XP) and copy it to the designated target queue numbers. To get from the DMEM to the selected targeted queue number requires two (2) items::

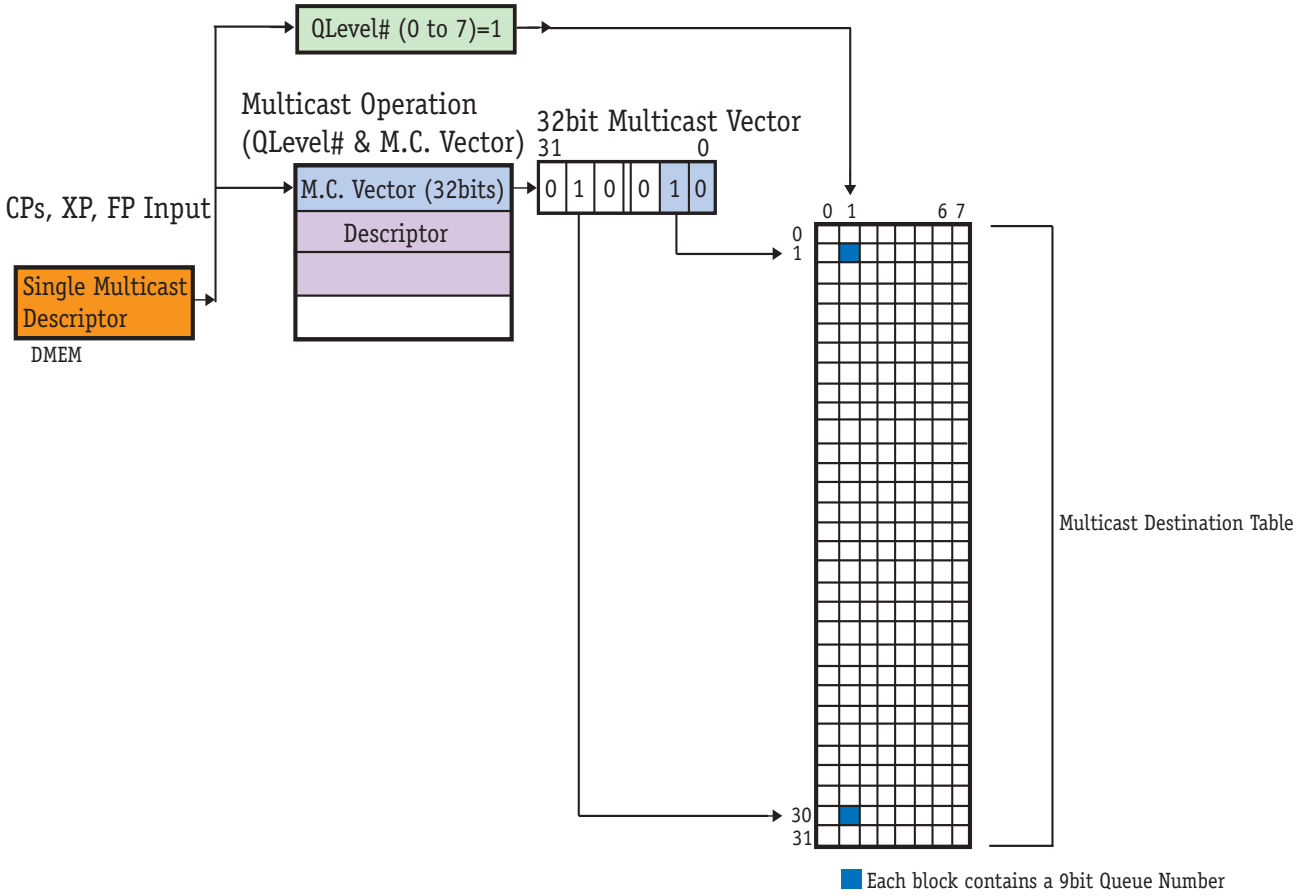
- User-defined target queue represented as a 32bit Multicast Vector
- Queue Level Number

The multicast operation is implemented using the Control Blocks (WrCB0_). Specifically, the *WrCB0_Sys_Addr* register, bits [18:16] *QLevel#* field that is used to select the queue level number (0 to 7). This allows eight (8) multicast queueing levels to be mapped to as many as eight (8) queues for each processor. Processors can have < 8 multicast-enabled queues if desired. Generally, the multicast vector is fetched from a multicast table in the TLU by the processor that built the descriptor.

The 32bit Multicast Vector, plus the QLevel#, are combined to provide the *Queue_Number* address (0 to 255) into the Multicast Destination Table. The table is set up with 9bit queue numbers, by using the *Multicast_Destination0* to *Multicast_Destination255* registers bits [8:0] *Queue_Number* field.

One of the QMU's setup steps is to map individual queue numbers (0 to 511) to their respective processors (CPs, XP, or FP) using the *Base_Queue_CP0* to *Base_Queue_CP15*, *Base_Queue_FP*, and *Base_Queue_XP* registers. This provides the association between the queue numbers and their mapped processors.

Figure 100 Multicast Enqueue Operation Example



Multicast elaboration is performed by copying the descriptor into a descriptor buffer for each of the appropriate queues. Other than the availability of *Free_Descriptor_Buffer_List* and *Dyn_Descriptor_Pooln_Usage*, the QMU imposes no limit on the number of descriptors that can simultaneously be in the queues as the result of multicast operations. Refer to "Multi-Use Counter (MUC) Management Transactions" on page 299. A single multicast enqueue can have a maximum of 32 targets. However, the external switching fabric must be capable of multicast replication for multiple fabric ports from a single multicast queue.

Multicast Operations Success or Failure

Descriptors enqueued in the QMU correspond to payload buffers located in the BMU. If a descriptor does not make it to a destination processor, the corresponding payload buffer is effectively lost as well because it is not deallocated. In a multicast operation, a payload buffer is assigned a destination count equal to the number of destination processors to receive a copy of the descriptor. When all the destination processors have received their copy of the descriptor, the payload buffer is released because the destination-count reaches zero (0).

It is critical that the QMU succeed at multicasting the descriptor to all the destination processors. If it does not, one or more processors never receive a copy of the buffer descriptor, and the payload buffer's destination count never reaches zero (0) and the payload buffer is never released. This would cause a "memory leak". If the QMU cannot successfully enqueue a descriptor to all the specified destinations, it rejects the entire multicast enqueue request. When a multicast enqueue request fails, the requesting processor can either retry the multicast enqueue, or drop the payload.

Multicast Operation Throughput Considerations

An N-way multicast enqueue operation cannot take longer than N unicast enqueues if the C-5e NP is to maintain its throughput. Therefore, the QMU cannot afford to take the time to visit each target queue to determine if its length is over its limit before actually beginning to do the series of enqueues. Since the QMU cannot check ahead of time and it cannot fail to enqueue to each queue, it cannot check the queue limit while it is doing the enqueues. This means that a multicast operation can push a queue above its length limit.

Before the QMU begins the individual enqueues of a multicast enqueue operation, it checks that there are enough free descriptor buffers (*Free_Descriptor_Buffer_List* register) and enough room within the dynamic pool (*Dyn_Des_Usage_Lim_Pooln* register) to complete the series of enqueues. As with the queue length limit, the QMU cannot know ahead of actually doing the individual enqueues how many of the target queues will be above their allowances and therefore need a dynamic descriptor buffer. The initial pool usage check insures that there will be enough dynamic buffers no matter what. However, the pool usage check might find that there are not enough buffer credits left if all the target queues need one, when, in actuality, there are enough pool credits available for the number of queues that actually need them. In this case, the multicast operation will be rejected when it could have succeeded.

Because the QMU only has time to check the dynamic pool of the first queue, all member queues of a multicast group must share the same dynamic buffer pool.

Queue Levels Supported in Multicast Operations

The QMU supports eight (8) levels (0 to 7) of multicast queueing. The purpose of using levels is to assign a specific queue to each level in order to implement a multicast operation. In a multicast operation, a single payload descriptor is simply copied to each of the targeted queues specified using the Queue Level Number field (QLevel#) in the *WrCB0_Sys_Addr*. The Queue Level Number is an index to the Queue Number to write to.

Although levels are available only through multicast commands, unicast descriptors can also use queueing levels if enqueued via a multicast command with a fanout of 1. A multicast descriptor specifies only the multicast vector and the queueing level; the QMU then maps that onto the specific queue corresponding to the given level for each target processor. The presumption is that a given descriptor is associated with a class-of-service that applies at each output port (CPs, XP, or FP). When different queueing levels are needed at different output ports, multiple multicast enqueues can be issued so that each one covers the destinations at each of the required queueing levels.

Multicast mapping uses a configurable look-up table(Multicast Destination Table) to store the queue number for each combination of target queue and queueing level. For the 32bit Multicast Vector and eight (8) queueing levels, this requires two-hundred-fifty-six (256) entries. Each entry holds a 9bit target queue number. With this scheme, each processor with eight (8) or more queues can have up to eight (8) queues for transmitting multicast traffic. These queues could also receive unicast traffic. Target processors can accept < 8 queues. Refer to "Multicast_Destination0 to Multicast_Destination255 Registers (QMU Configuration Function)" on page 651.

The actual mapping depends on the particular relationship of queueing levels to service classes in each application.

Multicasting to the Fabric Processor

The FP multicast vector bit in the descriptor is set by the originating processor and directs the descriptor to one FP queue at the appropriate queueing level.

QMU Configuration Space

The QMU has memory-mapped Configuration Space that contains a number of registers. The registers are used for four (4) purposes: mapping of queues to CPs, XP and FP, configuration of the QMU, QMU debugging, and collecting QMU statistics. Refer to Table 129 on page 447.

Table 129 QMU Registers

QMU REGISTER TYPES	REGISTER FUNCTION	SPECIFIC REGISTER DETAILS
CP's Queue Mapping	These registers specify the base address for a CP's queues.	See "Base_Queue_CP0 to Base_Queue_CP15 Registers (QMU CP's Queue Allocation Function)" on page 640.
XP's Queue Mapping	These registers specify the base address for a XP's queues.	See "Base_Queue_XP Register (QMU XP's Queue Allocation Function)" on page 641.
FP's Queue Mapping	These registers specify the base address for a FP's queues.	See "Base_Queue_FP Register (QMU FP's Queue Allocation Function)" on page 640.
QMU Configuration	Specifies the number of descriptor buffers to be available in the QMU.	See "Num_Descriptors Register (QMU Configuration Function)" on page 642.
	Specifies the maximum number of descriptors that can be enqueued dynamically to the queues associated with Pool*n*.	See "Dyn_Des_Usage_Lim_Pool0 Register (QMU Configuration Function)" on page 642.
	Specifies the operating mode of the QMU. Note: The External Mode is not supported.	See "Operation_Mode Register (QMU Configuration Function)" on page 643.
	Specifies the size of the data stored for each descriptor in an encoded form.	See "Descriptor_Size Register (QMU Configuration Function)" on page 644.
	Provides the mapping of the multicast destination port and queue level to a target queue number for each leaf of a multicast elaboration.	See "Multicast_Destination0 to Multicast_Destination255 Registers (QMU Configuration Function)" on page 651.

Table 129 QMU Registers (continued)

QMU REGISTER TYPES	REGISTER FUNCTION	SPECIFIC REGISTER DETAILS
QMU Status	Designates the total number of free descriptors.	See "Free_Descriptor_Buffer_List Register (QMU Control Function)" on page 652.
	Designates how many buffers are in use in Pool*n*.	See "Dyn_Descriptor_Pool0_Usage Register (QMU Status Function)" on page 653.
QMU Statistics	Count of Queue Configuration Operations.	See "Config_Q_Cnt Register (QMU Statistics Function)" on page 644.
	Count of Read Status operations.	See "Rd_Q_Status_Cnt Register (QMU Statistics Function)" on page 644.
	Count of Unicast Enqueues from the CPs.	See "CP_Uni_Enq_Cnt Register (QMU Statistics Function)" on page 645.
	Count of Multicast Enqueues from the CPs.	See "CP_Multi_Enq_Cnt Register (QMU Statistics Function)" on page 645.
	Count of Total Multicast Enqueues Targets from the CPs.	See "CP_Multi_Enq_Target_Cnt Register (QMU Statistics Function)" on page 645.
	Count of Dequeue operations from the CPs.	See "CP_Dequeue_Cnt Register (QMU Statistics Function)" on page 645.
	Count of Unicast Enqueues from the FP.	See "FP_Uni_Enq_Cnt Register (QMU Statistics Function)" on page 645.
	Count of Multicast Enqueues from the FP.	See "FP_Multi_Enq_Cnt Register (QMU Statistics Function)" on page 645.
	Count of Total Multicast Enqueues Targets from the FP.	See "FP_Multi_Enq_Target_Cnt Register (QMU Statistics Function)" on page 646.
	Count of Dequeue Operations from the FP.	See "FP_Dequeue_Cnt Register (QMU Statistics Function)" on page 646.
	Count of QMU Idle Clock Cycles.	See "QMU_Idle_Cycles Register (QMU Statistics Function)" on page 646.

Table 129 QMU Registers (continued)

QMU REGISTER TYPES	REGISTER FUNCTION	SPECIFIC REGISTER DETAILS
QMU Statistics (continued)	Count of Payload NACKs.	See "Payload_NACK_Cnt Register (QMU Statistics Function)" on page 646.
	Count of Global NACKs.	See "Global_NACK_Cnt Register (QMU Statistics Function)" on page 646.
	Count of Payload Read Failures.	See "Payload_Read_Failures_Cnt Register (QMU Statistics Function)" on page 646.
	Count of Command Processor Errors, illegal opcodes and out of range queue numbers.	See "Cmd_Processor_Err_Cnt Register (QMU Statistics Function)" on page 647.
	Counts errors in dequeued descriptors in both internal and external modes.	See "Dq_H_Par_Err_Cnt Register (QMU Sequence Numbers Function)" on page 647
	Counts errors in dequeued descriptors in internal mode	See "Dq_L_Par_Err_Cnt Register (QMU Sequence Numbers Function)" on page 648
	Clears the Statistics Registers.	See "Clear_Statistics Register (QMU Statistics Function)" on page 637.
QMU Sequence Numbers	Provides a 32bit field for a saturating count of the number of sequence numbers missing in front port enqueues to the QMU.	See "Missing_Front_Seq_Num_Cnt Register (QMU Sequence Numbers Function)" on page 648
	Provides fields for the ingress and egress sequence numbers used with front-ports.	See "Front_Seq_Num Register (QMU Sequence Numbers Function)" on page 649
	Provides a field for the egress sequence numbers used with back ports.	See "Back_Seq_Num Register (QMU Sequence Numbers Function)" on page 649
	Provides fields that control a time out associated with sequence number on enqueue.	See "Front_Seq_Num_Timeout Register (QMU Sequence Numbers Function)" on page 650

For complete details about specific registers go to their reference. Refer to "Queue Management Unit (QMU) Configuration Registers" on page 633.

QMU Setup

The QMU must be initialized to operate. Using Global Bus operations, various registers and memories must be written with appropriate values to allow the QMU to function. Initialization must be completed before the QMU can be put online. See Table 129 on page 447 for more complete descriptions of the registers listed below.

As with most C-5e NP components, QMU initialization is completely handled by Motorola's programming interface, so that programmers never directly deal with the registers listed in this section.

Initializing the QMU involves writing the following:

1 Configure the pools using the following registers:

- *Operation_Mode* register (Internal=0x1)

Warning: *The External Mode is not supported.*

- *Descriptor_Size* register (12, 16, 24, 32Bytes)
- *Num_Descriptors* register (1 to 16, 384)
- *Dyn_Des_Usage_Lim_Pool0* to *Dyn_Des_Usage_Lim_Pool3* registers for each of the four (4) *Dyn_Descriptor_Pooln_Usage* registers

2 Map the queues to their applicable processors (CPs, XP, or FP):

This maps specific processors to specific queues to be used for Unicast Enqueue Operations, as well as mapping specific processors to specific queues to be used for Multicast Enqueue Operations.

- *Base_Queue_CP0 to Base_Queue_CP15*, *Base_Queue_XP* and *Base_Queue_FP* registers for each of the eighteen (18) processors need to be initialized.
- 144 mapping table entries need to be written for Multicast Enqueue Operations.

3 Establish the *Free_Descriptor_Buffer_List*:

- This requires coordinated initialization of the *Free Descriptor Buffer List* register (head pointer, tail pointer, and length) and the internal descriptor linkage-list pointers. If there are 16,384 descriptors buffers, 16,383 linkage pointers must be initialized to point one to the next. The last descriptor's buffer linkage pointer is a *don't-care*.

4 Set the queue parameters:

- Queue Length Allowance — The guaranteed minimum amount (0 to 16K-1) of descriptor buffers available for this queue. Implemented using the Configure Queue Operation.
- Queue Length Limit — The maximum number of descriptor buffers available (0 to 16K-1) for a particular queue. In other words, Allowance is the minimum amount of the (0 to 16K-1) range, where as, Limit is the maximum of the (0 to 16K-1) range. Implemented using the Configure Queue Operation. Refer to "Configure Queue Operation" on page 431.

5 Set the queue Link-lists:

- The queue lengths must be initialized to zero. With a length of zero, the queue head and tail pointers are don't-cares.

6 Enable the *QMU_Run_Enable* bit:

- The QMU must be run enabled before it can process queueing operations. The *QMU_Run_Enable* register is written with a "1" to do this.

QMU Performance

The QMU uses pipelining, so that the execution latency of the queueing operations (i.e., the time between the request, at the QMU, until the descriptor is returned to the requester) is greater than the time interval between the beginning of the execution of the descriptors.

Execution Speed and Descriptor Size Relationship

Both the C-5e NP and C-3e NP use a separate external clock circuit. For the C-5e NP the clock runs at 160MHz (maximum), and for the C-3e NP it runs at 150MHz (maximum). The actual maximum number of queue operations achievable by the QMU is determined by both the clock frequency of the Queue Management Engine (QME) and the size of the descriptor. One (1) QME clock cycle for each (32bit word) is required for the descriptor storage/retrieval process. Their are four (4) allowed descriptor sizes: 12Bytes, 16Bytes, 24Bytes and 32Bytes.

For example, using a QME clock frequency of 160MHz and a 32Byte descriptor, the maximum number of queue operations is found using the formula shown in Figure 101 on page 452. Also, refer to Table 130 on page 452.

Figure 101 QMU Performance Formula

Number of Queue Operations per Second (M/s) = 1/((1/f) * (Descriptor Size/4))
Number of Queue Operations per Second (M/s) = 1/((1/160MHz) * (32/4))
Number of Queue Operations per Second (M/s) = 1/ ((6.25e^{-9}) * (8))
Number of Queue Operations per Second (M/s) =1/5e^{-8}
Number of Queue Operations per Second (M/s) = 20M/s

Using:
QME Clock Frequency=160MHz
Descriptor Size=32Bytes

Table 130 QMU Performance Results Using the Formula and Typical QMU Speeds

FREQUENCY OF THE QME CLOCK (MHZ)	FOR DESCRIPTOR SIZE =12BYTES	FOR DESCRIPTOR SIZE =16BYTES	FOR DESCRIPTOR SIZE =24BYTES	FOR DESCRIPTOR SIZE =32BYTES
150	50.00M/s	37.50M/s	25.00M/s	18.75M/s
160	53.33M/s	40.00M/s	26.67M/s	20.00M/s
175	58.33M/s	43.75M/s	29.17M/s	21.88M/s

Multicast Support (System Level)

The C-5e NP supports multicasting Ethernet packets and multi-pointing ATM frames or cells. Several C-5e NP components are involved in the multicast process.

Multicast elaboration from the Fabric Processor (FP) to the QMU is not supported for this version of the C-5e NP.

Multicast Flow in the C-5e NP

The overall multicast transaction flow is described below. It has been separated into the receive and the transmit portions for clarity.

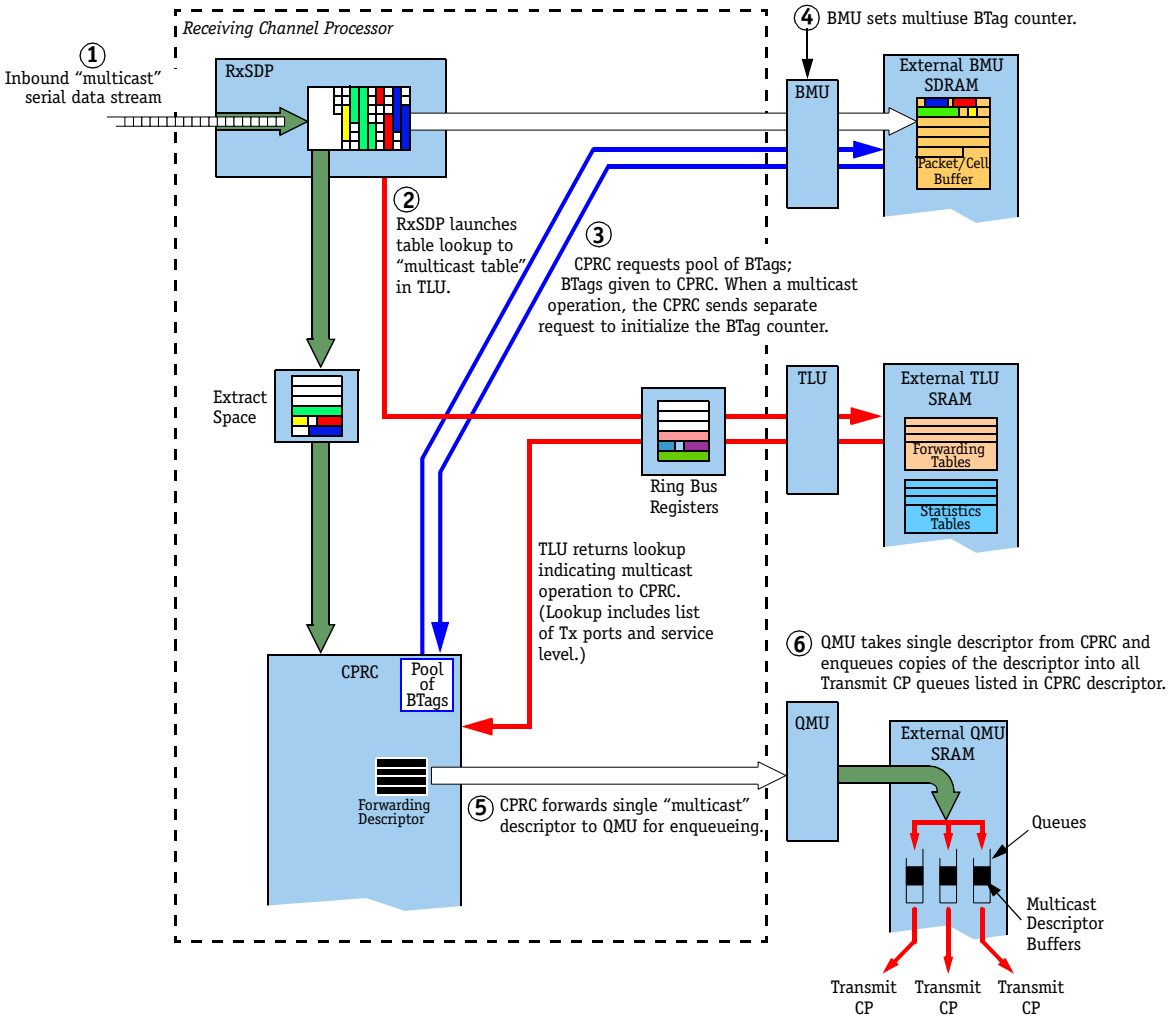
Multicast Receive Flow Transaction Process

The following describes the receive flow for a multicast transaction. Refer to Figure 102 on page 454.

1. RxSDP receives the “multicast/multipoint” packet/cell.
2. Based on a combination of SDP processing, table lookup, and CPRC processing, the CP determines that the packet/cell requires a multicast forwarding operation.
3. The CPRC requests and assigns a multiuse Buffer Tag (BTag) to the packet/cell and requests the BMU to associate a multi-use counter (MUC) BTag with the BTag.
4. The BMU assigns a multi-use counter (MUC) BTag (the count equals the number of transmit ports) and associates the counter with a buffer.
5. The CPRC can perform additional processing and then sends the descriptor to the QMU.
6. The QMU removes the multicast vector and queue level information and then tries to enqueue the descriptor to the specified output queues. It assesses the supply of descriptor buffers and if there are enough, it proceeds to do the enqueues.

   Each descriptor queue maintains an allocated descriptor count and an overall descriptor usage limit. The QMU checks the dynamic buffer pool for the first output queue, and if that pool has enough buffers to match the number of transmit ports, the QMU proceeds. Otherwise, it signals a failure of the enqueue operation and does not complete the multicast operation. Thus, all members of a multicast group should share the same dynamic buffer pool.

Figure 102 Multicast Application Receive Process Flow

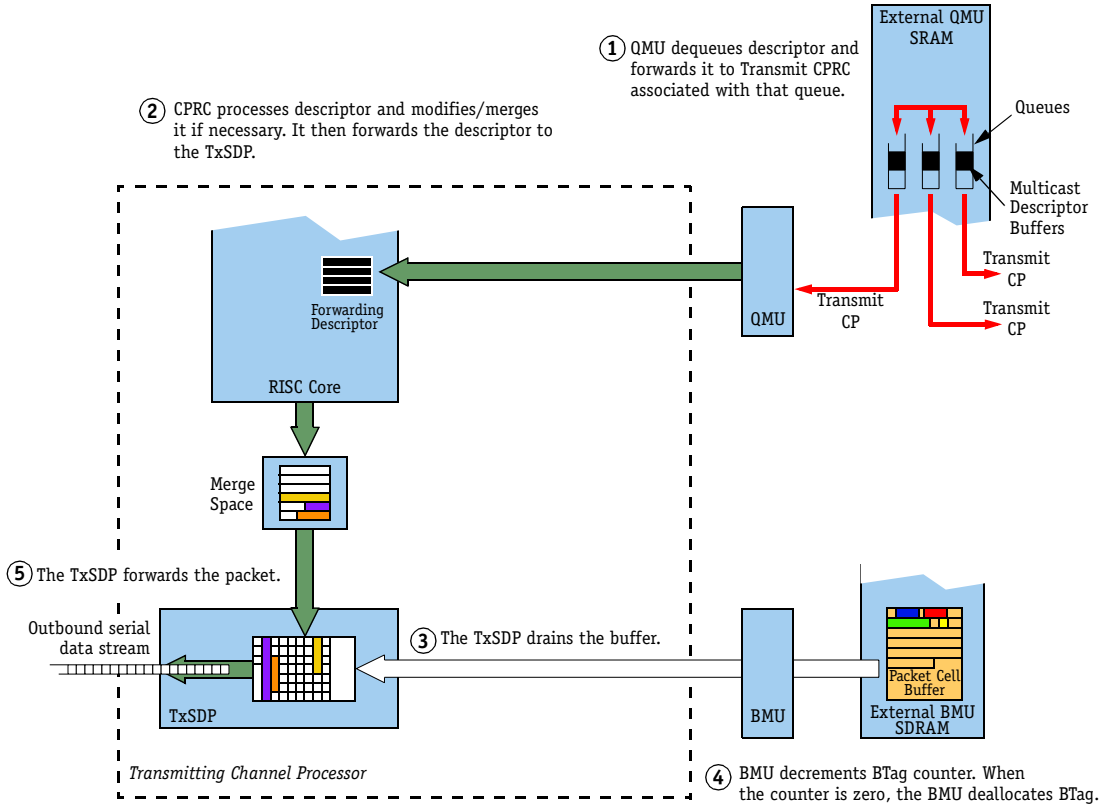


Multicast Transmit Flow Transaction Process

The multicast transmit process is identical to a unicast operation. The following describes the transmit flow for a multicast transaction. Refer to Figure 103 on page 456.

1. The CPRC requests the descriptor in its assigned queue.
2. The CPRC processes the descriptor (if necessary) and then forwards it to the TxSDP.
3. The TxSDP drains the corresponding buffer and appends the header.
4. The CPRC drains the buffer and then sends a message to the BMU to decrement the counter. When the multi-use counter (MUC) BTag reaches 0 (all buffers associated with this multicast operation have been drained), the BTag is deallocated.
5. The TxSDP forwards the packet. Each port sends the packet/cell out when it is able, based on the amount of traffic that is queued for that port.

Figure 103 Multicast Application Transmit Process Flow



External Scheduler Mode

The C-5e provides two modes for managing queues. They consist of:

- Internal Mode (using the internal QMU only)
- External Mode

The operation of the External Mode is described in the following section.

Warning: *Although the C-5e NP provides an External QMU Mode, it does not support an external traffic manager device. In addition, the associated registers and fields should not be implemented.*

Operation of the External Scheduler Mode

The External Mode uses the QMU's SRAM (200MHz, 55 signal) interface and internal structures are reconfigured and new internal data-paths are added. The interface is divided into two sections: one (1) for enqueues and one (1) for dequeues. Source-synchronous clocking is used to minimize skew and ease system analysis and design.

The QMU must have data-paths to bypass the normal processing of enqueues and pass these commands directly onto the external device. Additional information from the first beat of the queue message on the Payload Bus is passed to the external device along with the descriptor using some new fields. The QMU fills in the type, multicast bit, and *srcID* fields needed by the external devicein these enqueue data messages.

When handling speculative enqueues and the corresponding commit commands, the scrID field enables the external device to correlate the commit with a previous enqueue. Commits are transferred to the external device using a special one (1) word message on the descriptor enqueue path (NQD [23:0]). Commits to the external device are always sent sometime after the associated speculative enqueue; the commit can never precede the enqueue. The srcID in enqueue and commit messages determine the mailbox number.

The Q5 controls the content and rate of outgoing data by pushing descriptors to the transmitting ports. This is done by reconfiguring the memories for QMU buffer links (8k x 32b) and weights (4k x 32b) into descriptor Virtual Output Ports (VOPs) for a total of 48kBytes of storage. VOPs are descriptor FIFOs that are written by the external device when the credit based scheme indicates an open entry and read by a transmit port using conventional dequeue requests. The QMU supports 512 VOPs, each of which has an independently configured size. The QMU reports status of the VOPs to the transmit ports in the same way it reports queue status in internal mode, namely using broadcast empty

to non-empty queue status transitions and using dequeues to indicate that a VOP is empty.

Queue weights are not used when the QMU is in External Mode. When speculative enqueues are used, the external device will not push descriptors to the QMU until they are committed.

The items stored in VOPs are called *VOP-descriptors*. A VOP-descriptor contains a normal descriptor plus 8Bytes of auxiliary dequeue data. The auxiliary dequeue data can be used to pass information from the external device to first beat of the dequeue message on the Payload Bus. The external device uses new fields that can be passed for this purpose. VOP-descriptors use new fields to accommodate the extra information on the Payload Bus.

The total number of VOP-descriptors, legal range is (1024 to 2048), depends on the descriptor size configured. The number of descriptors in each VOP is configured through Configure-Queue operations. There is a new format of these operations in External Mode. VOP queue lengths are statically allocated, and there are *no* dynamic buffer pools.

Implementation of External Scheduler Mode

- Using the *Operation_Mode* register bits [3:0] *queueing mode* field (External mode=10, external scheduler mode).
- New enqueue and dequeue formats for Payload Bus.
- New configure queue function for the external scheduler mode. When in external scheduler mode, the configure queue function is used to specify the size of each VOP. This specifies the starting and ending descriptor number for each VOP. The range of descriptor numbers available depends on the *2bit encoded value* programmed in the *Descriptor _Size* register of the external device. Refer to Table 131 on page 459.

VOP Descriptors for CPs and/or FPTx

In addition to the normal descriptor, additional information associated with a PDU is needed by the transmit ports (CPs and/or FPTx). This additional information is contained in the new *VOP-descriptor*. The VOP-descriptor contains: the descriptor (12 to 32Bytes) and an auxiliary dequeue data (8Bytes).

The auxiliary dequeue data contains the following two (2) items:

- Two 24bit fields of external device dequeue data (*Q5DeqData0* bits [23:0] and *Q5DeqData1* bits [23:0]) that are sent from the external device along with each

dequeued descriptor. When this information is needed by the transmit engine (CPs and/or FPTx), it must be stored in the VOP.

- A 13bit field [12:0] *SeqNum* is used *only* for virtual queueing to the back port (FP). When the virtual queue bit is set in a external device dequeue message, a back port (FP) sequence number is generated by the QMU as the descriptor is placed in the VOP. The *SeqNum* field provides a way to carry these numbers to the FP. When the virtual queue is = 0, the *SeqNum* field is set to = 0. In contrast, for front ports (CPs) sequence numbering, the sequence numbers are generated when the descriptors are taken out of the VOP. Therefore, they do *not* need to be stored in the VOPs. Refer to Table 131 on page 459.

Table 131 VOP Descriptor Capacities

ENCODED VALUE	DESCRIPTOR SIZE (BYTES)	VOP DESCRIPTOR CAPACITY (WHEN IN EXTERNAL MODE
0	12	2048 Note: The External Mode is not supported.
1	16	2048 Note: The External Mode is not supported.
2	24	1536 Note: The External Mode is not supported.
3	32	1024 Note: The External Mode is not supported.

QMU Multicast in External Mode

When configuring the Multicast table, the values used are 9bit absolute queue numbers instead of 7bit offsets from the base_queue_number. Therefore, there is *no limitation* on the number of destinations per targeted port (CP, XP or FP).

Queue Organization in External Mode

Compared to the internal mode, three (3) of the internal SRAM data structures have changed, one (1) added and two (2) are not used when in the external mode. Refer to Figure 104 on page 461.

- Descriptor Link-list holds descriptors in external mode.

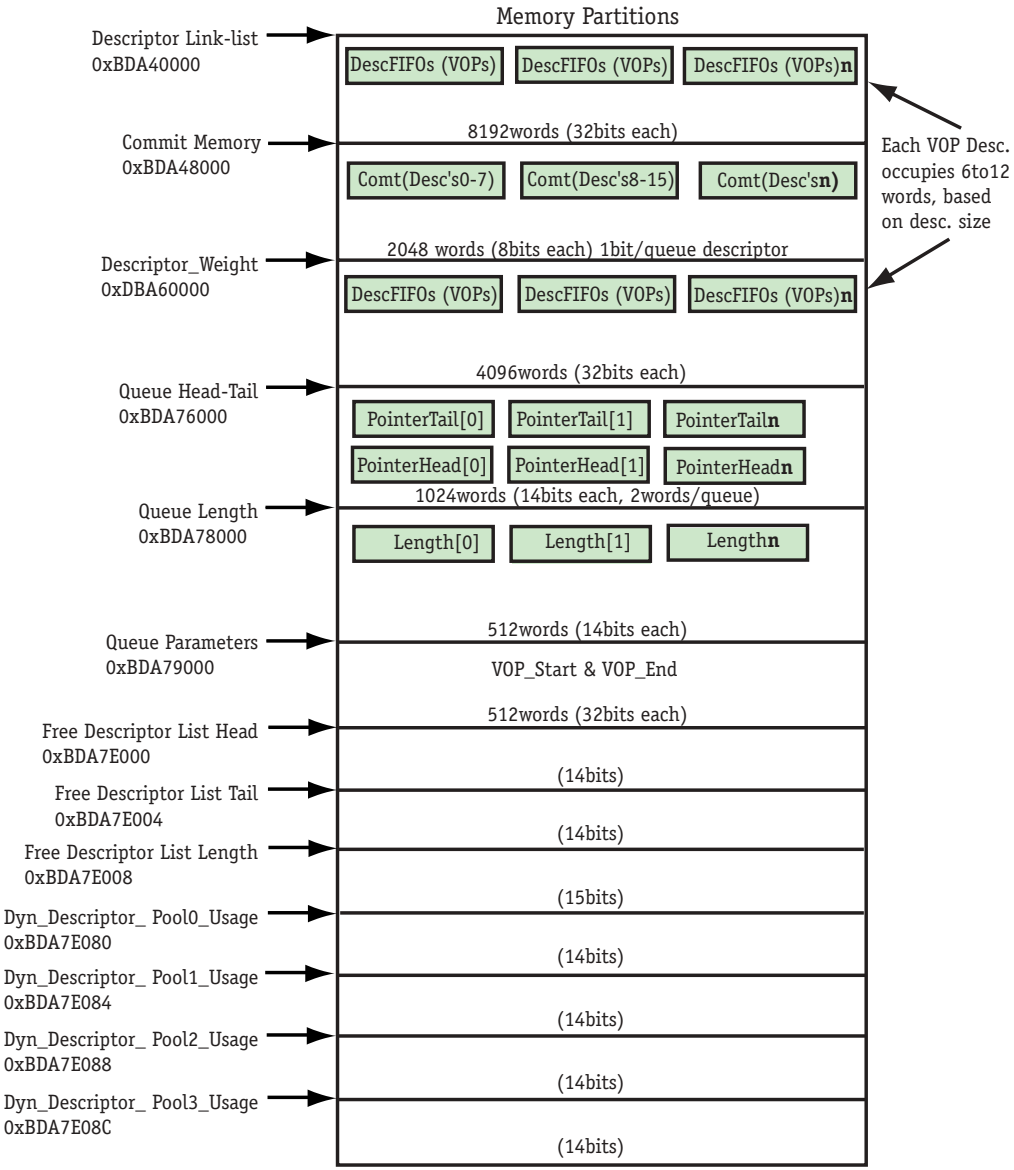
- Commit requires no configuration.
- Queue Parameters contain the starting and ending VOP-descriptor numbers for each Virtual Output Port (VOP). The VOP-descriptor numbers, from the starting number to the ending number (inclusive) are used as a FIFO for descriptors pushed out of the external device, but not yet read by the transmit port. The total number of VOP-descriptors available depends on the descriptor size selected using Descrip_Config register (12, 16, 24 and 32Bytes). Up to 512 VOPs can be programmed.

The starting to ending range should be programmed during initialization and not changed during data transmission. In addition, their programmed range for each VOP must not overlap with the range of any other VOP.

- Queue Head-Tail contains head and tail pointers for each VOP. The head pointer indicates where the next VOP-descriptor will be written. The tail pointer indicates where the next VOP-descriptor will be read. Each VOP consists of a range of VOP-descriptor numbers defined by the starting and ending values in queue parameter memory. This range is used as a ring buffer, i.e., the head and tail pointers are incremented from the Vop_Start value to the Vop_End value, and then the next increment causes them to wrap back to the Vop_Start value again. Head pointers are in odd addresses and tail pointers are in even addresses.

The head pointer and tail pointer for each VOP must be initialized to the Vop_Start value for that VOP. In addition, their programming should be done during initialization and not changed during transmission.

Figure 104 Internal SRAM Space Using External Mode



QMU Setup in External Mode

The QMU must be initialized to operate. Using Global Bus operations, various registers and memories must be written with appropriate values to allow the QMU to function. Initialization must be completed before the QMU can be put online.

- Reset the C-5e NP.
- Configure the external device dequeue interface to drive valid NQ_RDY, DQ_ARDY, and PARITY pins.
- Place C-5e/QMU in internal mode to perform configuration and initialization using *Operation_Mode* register (Internal=01).
- Set C-5e/QMU to external mode using the *Operation_Mode* register (External=10).
- Configure the external device enqueue interface to operate.

Warning: *Although the C-5e NP provides an External QMU Mode, it does not support an external traffic manager device. In addition, the associated registers and fields should not be implemented.*

Queue Management Transactions in External Mode

Queue transactions consist of five (5) different functions (Queue Status, Unicast Enqueue, Speculative Unicast Enqueue, Multicast Enqueue, and Dequeue). Queue transactions are invoked using Control Blocks (WrCB0, and RdCB0).

Warning: *Although the C-5e NP provides an External QMU Mode, it does not support an external traffic manager device. In addition, the associated registers and fields should not be implemented.*

Queue Transaction Functions (Operation and Examples) in External Mode

Each is described here along with examples.

Queue Status Operation in External Mode

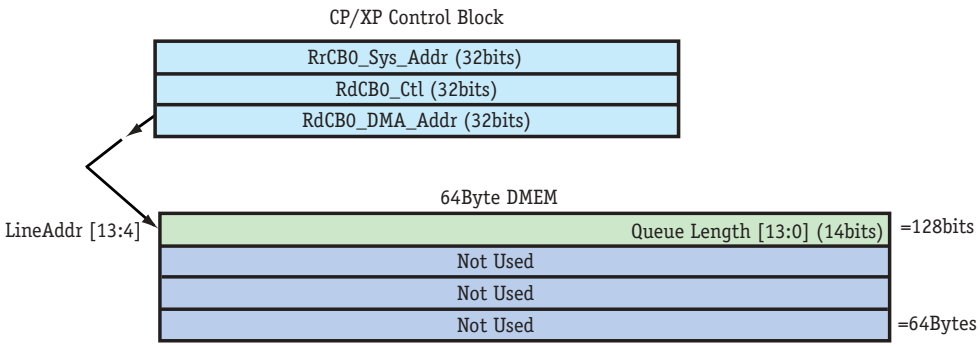
Queue Status uses a control block (RdCB0) to read a single queue's length from the QMU into the (DMEM) of the requesting CP, or XP.

Queue Status Example in External Mode

The bits for RdCB0_Sys_Addr, RdCB0_Ctl and RdCB0_DMA_Addr are set as shown in the internal mode.

The RdCB0_DMA_Addr bits [13:4] *LineAddr* field refers to the Local DMEM buffer address that is generally 64Byte aligned. The queue's length [13:0] (14bits) are located in the first 128bit line inside the 64Byte DMEM as shown in Figure 105 on page 463. The second, third and fourth 128bit lines are not used.

Figure 105 Queue Status Implementation in External Mode



Refer Table 132 on page 468 for detail descriptions of these fields.

Unicast Enqueue Operation in External Mode

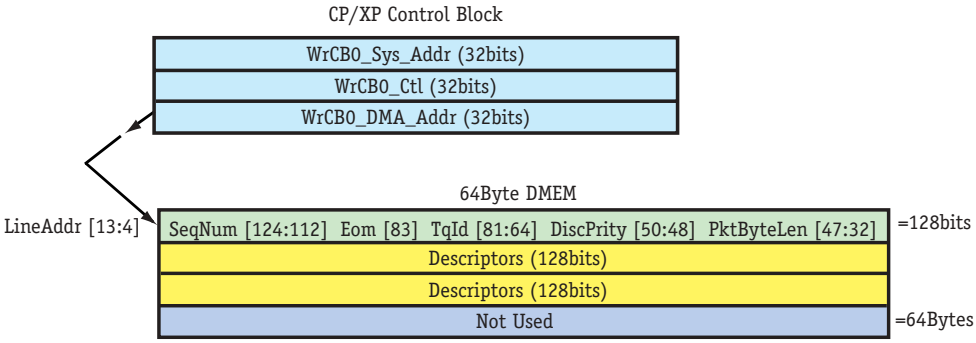
Unicast Enqueue uses a control block (WrCB0) to write the Descriptor data into a queue in the QMU's (SRAM) from the (DMEM) of either the requesting CP or XP.

Unicast Enqueue Example in External Mode

The bits for WrCB0_Sys_Addr, WrCB0_Ctl, and WrCB0_DMA_Addr are set as shown in internal mode.

The WrCB0_DMA_Addr bits [13:0] *LineAddr* field refers to the Local DMEM buffer address that is generally 64Byte aligned. The Sequence Number [124:112], End of Message [83], Traffic Queue Identifier [81:64], Discard Priority [50:48], and Packet Byte Length [47:32] are located in the first 128bit line followed with descriptors (depending on 12, 16, 24, or 32Byte sizes) located in the second and third 128bit lines inside the 64Byte DMEM as shown in Figure 106 on page 464. The fourth 128bit line is not used.

Figure 106 Unicast Enqueue Implementation in External Mode



Refer Table 132 on page 468 for detail descriptions of these fields.

Speculative Unicast Enqueue Operation in External Mode

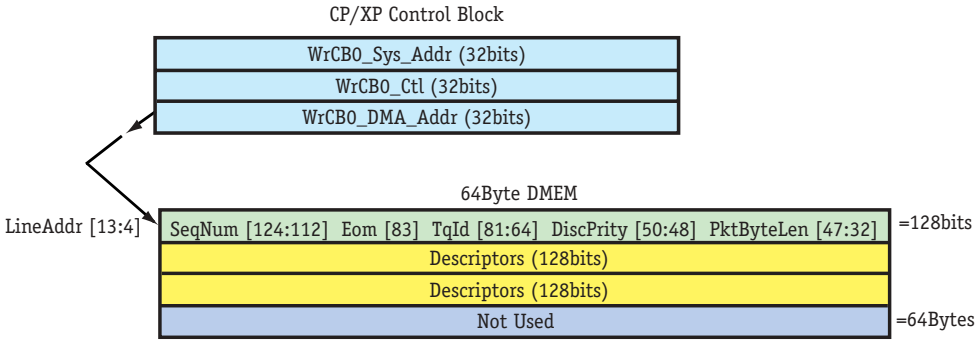
Speculative Unicast Enqueue uses a control block (WrCB0) to write the Descriptor data into a queue in the QMU's (SRAM) from the (DMEM) of either the requesting CP or XP.

Speculative Unicast Enqueue Example in External Mode

The bits for WrCB0_Sys_Addr, WrCB0_Ctl, and WrCB0_DMA_Addr are set as shown in internal mode.

The WrCB0_DMA_Addr bits [13:0] *LineAddr* field refers to the Local DMEM buffer address that is generally 64Byte aligned. The Sequence Number [124:112], End of Message [83], Traffic Queue Identifier [81:64], Discard Priority [50:48], and Packet Byte Length [47:32] are located in the first 128bit line followed with descriptors (depending on 12, 16, 24, or 32Byte sizes) located in the second and third 128bit lines inside the 64Byte DMEM as shown in Figure 107 on page 465. The fourth 128bit line is not used.

Figure 107 Speculative Unicast Enqueue Implementation in External Mode



Refer Table 132 on page 468 for detail descriptions of these fields.

Multicast Enqueue Operation in External Mode

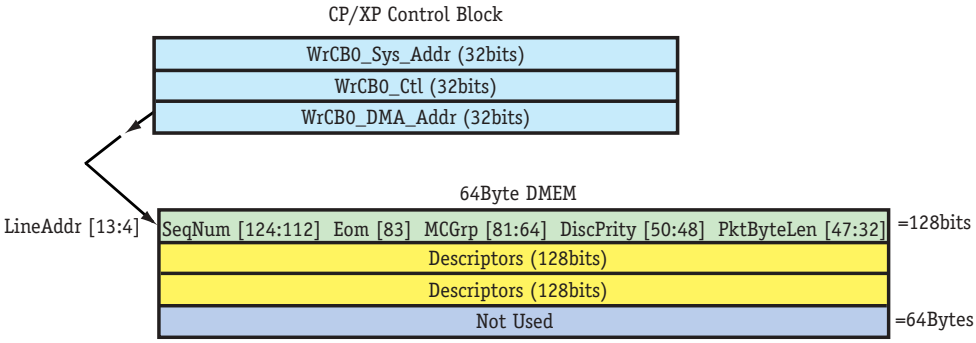
Multicast Enqueue uses a control block (WrCB0) to write a Descriptor's data into multiple queues in the QMU's SRAM from the DMEM of either the requesting CP, or XP.

Multicast Enqueue Example in External mode

The bits for WrCB0_Sys_Addr, WrCB0_Ctl, and WrCB0_DMA_Addr are set as shown in internal mode.

The WrCB0_DMA_Addr bits [13:0] *LineAddr* field refers to the Local DMEM buffer address that is generally 64Byte aligned. The Sequence Number [124:112], End of Message [83], Multicast Group [81:64], Discard Priority [50:48], and Packet Byte Length [47:32] are located inside the first 128bit line followed with descriptors (12, 16, 24, or 32Byte sizes) located in the second and third 128bit lines inside the 64Byte DMEM as shown in Figure 108 on page 466. The fourth 128bit line is not used.

Figure 108 Multicast Enqueue Implementation in External Mode



Refer Table 132 on page 468 for detail descriptions of these fields.

Dequeue Operation in External Mode

Dequeue uses a control block (RdCB0) to read Descriptor data from a queue in the QMU's SRAM into the DMEM of the requesting CP, or XP. Dequeue frees a Descriptor Buffer from a queue.

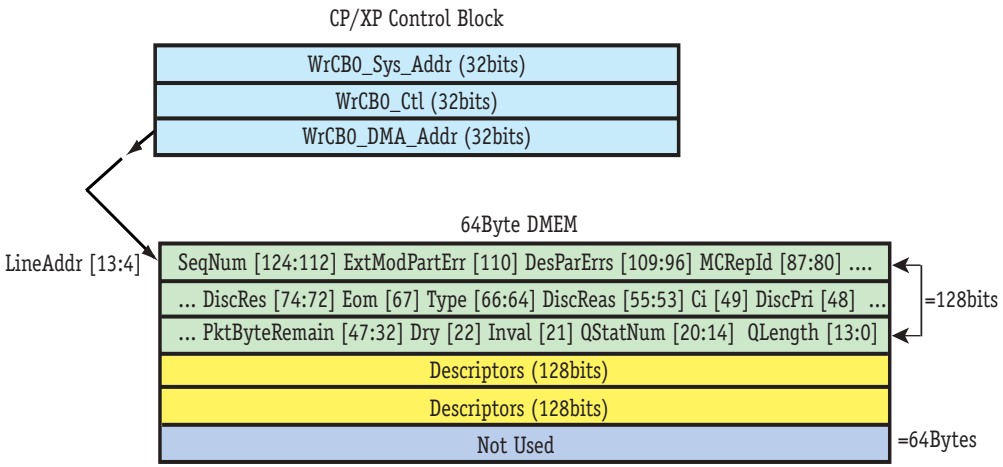
Dequeue Example in External Mode

The bits for RdCB0_Sys_Addr, RdCB0_Ctl and RdCB0_DMA_Addr are set as shown in internal mode.

The RdCB0_DMA_Addr bits [13:4] *LineAddr* field refers to the Local DMEM buffer address that is generally 64Byte aligned. The Sequence Number [124:112], External Mode Parity Error [110], Descriptor Parity Errors [109:96], Multicast Replication Identifier [87:80], Discard Reason [74:72], End of Message [67], Type [66:64], Discard Reason [55:53], Congestion Indicator [49], Discard Priority [48], Packet Byte Remainder [47:32], Dry [22], Invalid [21], Queue Status Number [20:14], Queue Length [13:0] are located inside the first 128bit line followed with descriptors (depending on 12, 16, 24, or 32Byte sizes) located in the second and third 128bit lines inside the 64Byte DMEM as shown in Figure 109 on page 467. The fourth 128bit line is not used.

The DiscardReason field is duplicated for internal reasons.

Figure 109 Dequeue Implementation in External Mode



Refer Table 132 on page 468 for detail descriptions of these fields.

Response Field Descriptions for Internal and External Modes

Table 132 on page 468 describes the individual response fields for both the internal and external modes.

Warning: *Although the C-5e NP provides an External QMU Mode, it does not support an external traffic manager device. In addition, the associated registers and fields should not be implemented.*

Table 132 Response Field Descriptions

FIELD NAME	DESCRIPTION
SeqNum	**Sequence Number** — Number used by QMU to order enqueues. Used only when the SeqNum bit is set in the QOP enqueue command. Otherwise this field is reserved.
DesParErrs	**Descriptor Parity Errors** — When bits are 1 they indicate parity errors received on the external device interface (covering external device pins DQD[23:0], DQPAR, NQRDY, and QARDY). Bit 109 indicates a parity error in the idle time before the descriptor was sent. The remaining bits are each for one word of the dequeue data message received from the external device dequeue interface, where bit 96 corresponds to the first word in the descriptor. Descriptors legal range= 3, 4, 6, or 8 words long. Unused bits =0, and errors =1's. Note: The External Mode is not supported.
Dry	**Dry** — Is 1 if the dequeue failed because the queue was empty. Otherwise is 0.
Inval	**Invalid** — Is 1 for a descriptor that was speculatively enqueued and then committed as invalid. Otherwise is 0.
QStatNum	**Queue Status Number** — Gives the queue number relative to the base queue number of the transmit engine that requested the dequeue.
QLength	**Queue Length** — Number of PDUs left on the queue after this dequeue.
ExtModeParErr	**External Mode Parity Error** — Is 1 if there has been at least one parity error reported from the external device interface since the QMU's last Payload Bus message. The error is not necessarily in the accompanying descriptor.
EOM	**End of Message** — Note: The External Mode is not supported.
TqId	**Traffic Queue Identifier** — Note: The External Mode is not supported.
DiscPriority	**Discard Priority** — Note: The External Mode is not supported.
PktByteLen	**Packet Byte Length** — Note: The External Mode is not supported.

Table 132 Response Field Descriptions (continued)

FIELD NAME	DESCRIPTION
MCastGrp	**Multicast Group** — Note: The External Mode is not supported.
MCastRepId	**Multicast Replication Identifier** — Note: The External Mode is not supported.
DiscReason	**Discard Reason** — Note: The External Mode is not supported.
Type	**Type** — Note: The External Mode is not supported.
Ci	**Congestion Indicator** — Note: The External Mode is not supported.
PktByteRemainder	**Packet Byte Remainder** — Note: The External Mode is not supported.

Chapter 8

INTERNAL BUSES

Chapter Overview

This chapter covers the following topics:

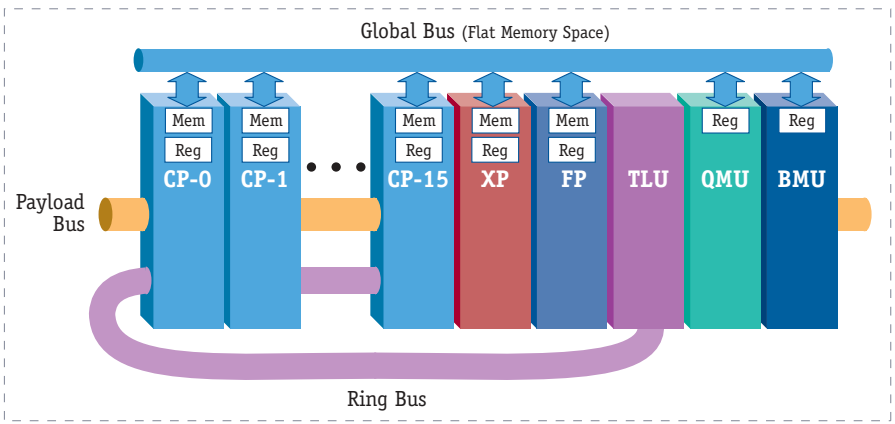
Internal Buses Overview

The C-5e NP contains three (3) independent data buses that provide internal communication paths between the C-5e NP's eighteen (18) processors (16CPs, XP, and FP) and three (3) coprocessors (TLU, QMU, and BMU), thus supporting concurrent processing. Refer to Table 133 on page 472. In addition, Figure 110 on page 472 shows the internal buses.

Table 133 C-5e NP Interconnect Components

ITEM	DEVICE TYPE	FUNCTION
Payload Bus	Slotted, multichannel, shared, arbitrated bus	Carries payload data and payload descriptors between the processors and the BMU and QMU.
Ring Bus	Slotted ring-topology bus	Provides bounded latency transactions between the processors and the TLU. It also supports inter-processor communication.
Global Bus	Slotted, multichannel, shared, arbitrated bus	Supports inter-processor communication via a conventional flat memory-mapped addressing scheme.

Figure 110 Internal Custom Buses



Internal Buses Characteristics

The three (3) buses have the bandwidth, bus width, and transfer size characteristics defined in Table 134 on page 473 for C-5e NP and Table 135 on page 473 for C-3e NP.

Table 134 C-5e NP Bus Characteristics Summary

BUS	BANDWIDTH	BUS WIDTH	TRANSFER SIZE
Payload Bus	54.5Gbps Max.	128bits	64Bytes
Ring Bus	17.0Gbps Max.	64bits	8Byte to 32Bytes
Global Bus	• 3.4Gbps to/from BMU/QMU Max. • 1.7Gbps among CPs Max.	32bits	4Bytes
Internal Aggregate Bandwidth (all three internal buses)	76.6Gbps total Max.	N/A	N/A

Table 135 C-3e NP Bus Characteristics Summary

BUS	BANDWIDTH	BUS WIDTH	TRANSFER SIZE
Payload Bus	36.9Gbps Max.	128bits	64Bytes
Ring Bus	11.5Gbps Max.	64bits	8Byte to 32Bytes
Global Bus	• 2.3Gbps to/from BMU/QMU Max. • 1.2Gbps among CPs Max.	32bits	4Bytes
Internal Aggregate Bandwidth (all three internal buses)	51.9Gbps total Max.	N/A	N/A

Bus Bandwidth General Formulas

The bandwidth values summarized above were derived using the general formulas in Figure 111 on page 474 and Figure 112 on page 474.

Figure 111 C-5e NP Bandwidth Formulas

Payload Bus:

266MHz * 2busses * 128bits/bus * .8TDM useage= 54.5Gbps Max.

Ring Bus:

266MHz * 64bits = 17.0Gbps Max.

Global Bus:

266MHz * 2busses * 32bits * .2TDM useage= 3.4Gbps to/from BMU/QMU Max.

266MHz * 32bits * .2TDM useage= 1.7Gbps among CPs Max.

Figure 112 C-3e NP Bandwidth Formulas

Payload Bus:

180MHz * 2busses * 128bits/bus * .8TDM useage= 36.9Gbps Max.

Ring Bus:

180MHz * 64bits = 11.5Gbps Max.

Global Bus:

180MHz * 2busses * 32bits * .2TDM useage= 2.3Gbps to/from BMU/QMU Max.

180MHz * 32bits * .2TDM useage= 1.2Gbps among CPs Max.

Payload Bus Overview

The Payload Bus is a slotted, multichannel, shared, arbitrated bus that provides a high bandwidth path for C-5e NP's (16CPs, XP, and FP) to shared services in the BMU and QMU. It has a guaranteed arbitration latency to satisfy CP programming constraints, a retry feature, and a bus acknowledgment to indicate when a transaction is complete.

Payload Bus Operation

The Payload Bus uses a 128bit wide data path in four-cycle bursts to transfer up to 64 Bytes of payload data, descriptors, buffer tags, and other information to or from a processor on each of two (2) independent channels. To achieve high bus utilization, operations are pipelined and reads are split into a read request and a write response. Typical payload operations are described in Table 136 on page 475.

Table 136 Typical Payload Operations

OPERATION	TYPE OF INFORMATION	QUANTITY/PDU
Rx Transactions		
Payload read	BTags (32 BTags are passed together)	1 per 32 PDUs
Payload write	Data	PDU size/64Bytes
Payload write	Descriptor	1
Tx Transactions		
Payload read	Descriptor	1
Payload read	Data	PDU size/64Bytes
Payload write	BTag	1

Payload Bus Latency

The Payload Bus arbitrates differently for the FP than for other clients (CPs and XP). This behavior is configurable by a *ZBFP* bit [9] in the *XP Miscellaneous Control* register. Refer to "XP Miscellaneous Control Register (XP Configuration Function)" on page 611. Setting this bit provides the FP with three (3) additional slots on the Payload Bus. Thus, you can optimize for greater FP access to the Payload Bus by setting this bit to 1, *or* optimize for better CP/XP access to the Payload Bus by setting this bit to 0 (the default configuration).

Payload Bus Latency (Default Mode)

In default mode (*ZBFP* = 0), the bus assigns/reserves one (1) bus slot for each processor. The default mode latency is shown in Table 137 on page 476.

Table 137 Payload Bus Arbitration Delay in Default Mode

LATENCY	READS	WRITES
Minimum	10 cycles	10 cycles
Maximum	CPs, XP, and FP = 110 cycles	CPs, XP, and FP = 110 cycles

Payload Bus Latency (FP Mode)

In FP mode (ZBFP = 1), additional bus slots are allocated to the TxFP for reads and to the RxFP for writes. This ensures that the FP can maintain a high data flow rate to the fabric. The FP mode latency is shown in Table 138 on page 476.

Table 138 Payload Bus Arbitration Delay in FP Mode

LATENCY	READS	WRITES
Minimum	10 cycles	10 cycles
Maximum	• TxFP = 40 cycles • CPs, XP, and RxFP = 140 cycles	• RxFP = 40 cycles • CPs, XP, and TxFP = 140 cycles

Ring Bus Overview

The C-5e NP implements a ring-topology bus for communication between the TLU and the eighteen (18) processors (16CPs, XP, and FP), each of which is a *node* on the ring. It uses a 64bit wide data path and is clocked at the C-5e NP core clock rate. The Ring Bus supports the following types of message transactions:

- Unacknowledged transaction
- Hardware acknowledged transaction
- Software acknowledged transaction

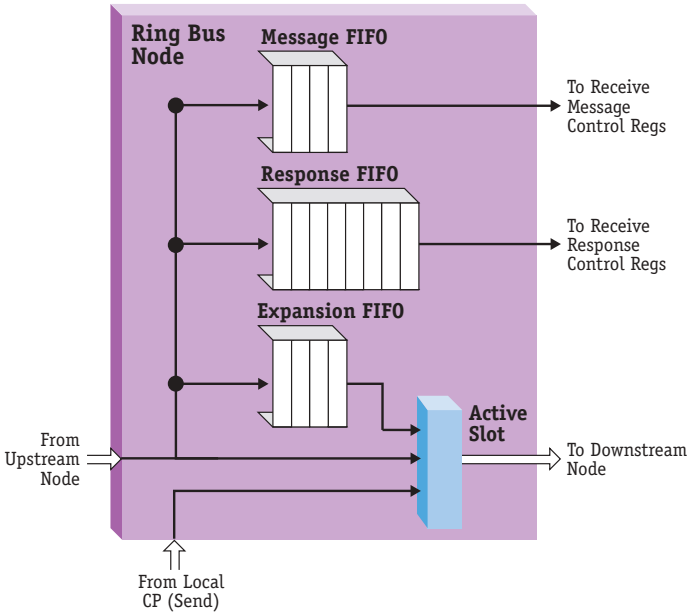
Ring Bus Major Components

A Ring Bus node consists of four (4) items. Refer to Table 139 on page 477 and Figure 113 on page 478.

Table 139 Ring Bus Components

ITEM	FUNCTION
Message FIFO	To pass messages to the *Receive Message* registers.
Response FIFO	To pass messages to the *Receive Response* registers.
Expansion FIFO	Where messages/responses passed from upstream are temporarily held when the active slot is busy.
Fixed-Size Active Slot	Where the upstream messages/responses or the local node's messages/responses are forwarded on the Ring Bus to the downstream node. The active slot comprises one (1) clock cycle, and the Ring Bus supports simultaneous node transmission and reception on each clock cycle.

Figure 113 Ring Bus Node Block Diagram



Ring Bus Node Operation

A Ring Bus node can perform the following actions:

- Sending Downstream
  - Send a new message or response to a downstream node.
- Receiving from Upstream
  - Receive a message destined for the local node.
  - Receive a response destined for the local node.
  - Pass through a message or response to a downstream node.

A node can send on the Ring Bus as long as the active slot (the slot currently available to the node) is free. Table 140 on page 479 lists the Ring Bus node IDs for the CPs, XP, FP, and TLU.

Table 140 Ring Bus Node IDs

UNIT	NODE ID	UNIT	NODE ID
CP0	0	CP10	10
CP1	1	CP11	11
CP2	2	CP12	12
CP3	3	CP13	13
CP4	4	CP14	14
CP5	5	CP15	15
CP6	6	XP	24
CP7	7	FP*	30
CP8	8	TLU	31
CP9	9		

* Transmit only. The FP cannot read messages on the Ring Bus. Thus any messages sent to the FP cannot be removed from the Ring Bus, eventually filling up the Ring Bus.

Sending Downstream

A local node uses its active slot to send downstream. When the local node is in the process of sending and then receives an upstream message targeted for a downstream node, the upstream message is placed in one of the local node's four (4) Expansion FIFO slots until the local node completes sending. Then the node passes the upstream message to the next node on the bus.

Contiguous "multi-slot" messages can be transmitted on the Ring Bus. Multi-slot messages are treated as one (1) message and are not divided as they move on the Ring Bus. The local node can send contiguous "multi-slot" messages (2 or 4 slots in length) as long as there are sufficient slots in the Expansion FIFO to hold the upstream messages. For example, if the local node wants to send a message requiring two slots, there must be two slots available in the Expansion FIFO. A four (4) slot message requires the expansion FIFO to be completely empty.

Receiving from Upstream

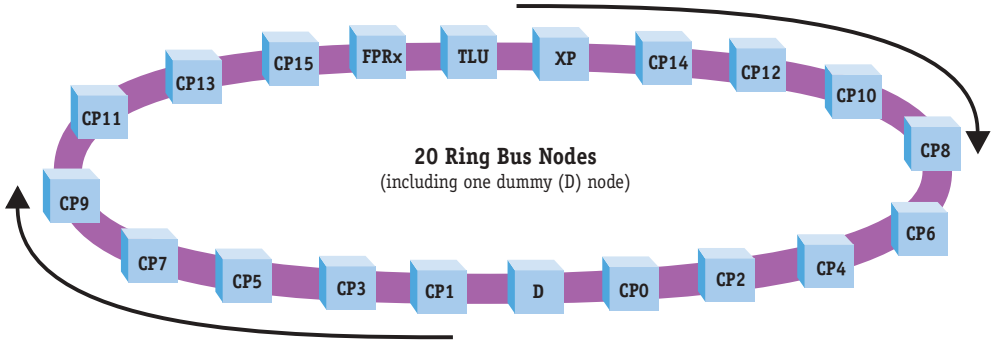
When a node receives a message or response from an upstream neighbor, it processes that message or response in one of three (3) ways:

- If a message is destined for the local node, it is forwarded through the four-slot (4) Message FIFO to the *Receive Message* registers. When the program reads these registers, the FIFO is popped. Refer to "Ring Bus (Rx) Receive Message Registers" on page 482.
- If a response is destined for this local node, it is forwarded through the eight-slot (8) Response FIFO to the *Receive Response* registers. A sequence number in the response dictates how the *Receive Response* registers are filled. If the target receive response block is already used in the register, then the FIFO can become blocked, possibly filling up the entire ring if incoming messages continue. The program must clear the *Receive Response* Register. Refer to "Ring Bus Receive (Rx) Response Registers" on page 483.
- If the node is simply forwarding a message/response downstream, it can send if the local node is not trying to send a message simultaneously. If its local node is sending a message, then the upstream message/response is placed in the local node's four-slot (4) Expansion FIFO. If this FIFO reaches capacity, the first item on the stack takes priority over the local node sending its own message/response and gets forwarded to the active slot.

Ring Bus Latency

When describing the Ring Bus's latency, it is important to understand that a "complete" transaction usually requires a round trip of the entire Ring Bus. For example, CP13 is located two nodes upstream from the TLU. If CP13 sends a request to the TLU, the "request" latency is three (3) clock cycles assuming that the TLU node is not busy. However, since the Ring Bus is unidirectional, the minimum latency to return data from the TLU to CP13 is 17 clock cycles. Thus round trip latency is 20 cycles, best case. Refer to Figure 114 on page 481.

Figure 114 Nodes on the Ring Bus



Because of the unidirectional nature of the Ring Bus and the fact that transactions on the Ring Bus are usually request/response, latency is not affected by which node delivers messages to the ring first. Therefore, the position of nodes on the Ring Bus should not be a consideration when designing your program.

The latency is also affected by the fact that the Ring Bus is expandable. Nodes (with the exception of the one (1) dummy node) can expand from one (1) slot to four (4) additional slots to increase Ring Bus node accessibility. The expansion is automatic. When upstream data is injected into a node, the node can expand to guarantee that the receiving node can still output data to the Ring Bus. This expansion enables the node to send four (4) slots of contiguous data.

Taking into account that Ring Bus message transactions are one-way operations and that nodes can expand up to four (4) additional slots, we can see that the worst case round trip latency is:

((19 nodes x 5 slots) + (the one dummy node)) = 96 clock cycles (assuming that the target node is not busy when the message arrives).

In rare cases, the latency might increase due to the target node being busy. In this case, the message continues around the bus and until it arrives at the target node again. If the target node is free, the transaction is completed.

A TLU response to the FPRx does not use the Ring Bus. Rather, these responses are sent via a dedicated path between the TLU and the FPRx.

Ring Bus Interface Registers

This section describes three (3) type of Ring Bus functions and their registers as shown in Table 141 on page 482.

Table 141 CP Registers by Function

CP FUNCTION	SPECIFIC REGISTERS
Ring Bus Tx Message Control	TxMsg0_Ctl, TxMsg0_Data_H, TxMsg0_Data_L; TxMsg1_Ctl, TxMsg1_Data_H, TxMsg1_Data_L; TxMsg2_Ctl, TxMsg2_Data_H, TxMsg2_Data_L; TxMsg3_Ctl, TxMsg3_Data_H, TxMsg3_Data_L
Ring Bus Rx Response Control	RxResp0_Ctl, RxResp0_DataH, RxResp0_DataL; RxResp1_Ctl, RxResp1_DataH, RxResp1_DataL; RxResp2_Ctl, RxResp2_DataH, RxResp2_DataL; RxResp3_Ctl, RxResp3_DataH, RxResp3_DataL; RxResp4_Ctl, RxResp4_DataH, RxResp4_DataL; RxResp5_Ctl, RxResp5_DataH, RxResp5_DataL; RxResp6_Ctl, RxResp6_DataH, RxResp6_DataL; RxResp7_Ctl, RxResp7_DataH, RxResp7_DataL
Ring Bus Rx Message Control	RxMsg_Ctl, RxMsg_FIFO

For complete details about specific registers go to their references. Refer to: "TxMsg0_Ctl Register (CP Ring Bus Tx Message Control Function)" on page 510, "TxMsg0_Data_H Register (CP Ring Bus Tx Message Control Function)" on page 512, "TxMsg0_Data_L Register (CP Ring Bus Tx Message Control Function)" on page 512, "RxResp0_Ctl Register (CP Ring Bus Rx Response Control Function)" on page 513, "RxResp0_Data_H Register (CP Ring Bus Rx Response Control Function)" on page 514, "RxResp0_Data_L Register (CP Ring Bus Rx Response Control Function)" on page 515, "RxMsg_Ctl Register (CP Ring Bus Rx Message Control Function)" on page 515, and "RxMsg_FIFO Register (CP Ring Bus Rx Message Control Function)" on page 517.

Ring Bus Transmit (Tx) Message Registers

Configuration Space includes four (4) sets of registers used to transmit messages on the Ring Bus. Refer to "Ring Bus Transmit (Tx) Messages Registers" on page 133.

Ring Bus (Rx) Receive Message Registers

Configuration Space includes a set of registers used to receive unsolicited messages. Refer to "Ring Bus (Rx) Receive Message Registers" on page 134.

Ring Bus Receive (Rx) Response Registers

Messages initiated by the CPRC as a request type expect to receive a subsequent response type message, for example TLU requests. Configuration space includes eight (8) sets of registers used to receive responses. Refer to “Ring Bus Receive (Rx) Response Registers” on page 134.

Global Bus Overview

The Global Bus is a slotted, multichannel, shared, arbitrated bus that supports inter-processor I/O using a single, flat memory model and provides direct access (via load/store operations) to all C-5e NP memory regions except CPRC IMEM, TLU table storage memory, SDP control stores, PCI configuration registers, and some XP configuration registers.

The Global Bus uses a 32bit wide data path with separate control and address path. It can transfer 4Bytes of data on each of two (2) independent data channels. It also has a guaranteed arbitration latency and a bus acknowledgment feature to indicate transaction completion. To achieve high bus utilization, operations are pipelined and reads are split into a read request and a write response. Retry selection is per CP (not per transaction).

Table 142 Global Bus Latency

WORST CASE	AVERAGE CASE	BEST
110 cycles	(Total global bandwidth/2) / (Num. of active processors)	10 cycles

Appendix A

C-5E NP REGISTERS

Appendix Overview

This appendix covers the following topics:

Although specific ranges of memory are allocated to specific functions, the entire area may not be used.

Channel Processor (CP) Configuration Registers

Configuration Space in the CPs is an area that contains a number of registers. The CPRC uses these registers to communicate with the SDP and the bus controllers (Payload Bus and Global Bus). The CP's registers can also be accessed by other components of the C-5e NP (XP and other CPs).

CP Registers

The following is a list of each CP register along with its address, function, and reference to its detailed parameters. The detailed parameters provide: purpose, field name, bit positions and descriptions. Refer to Table 143 on page 486.

Table 143 CP Registers

ADDRESS	REGISTER NAME	FUNCTION	DETAILED PARAMETERS
0xBC*n*00000	DMEM_Base	CP DMEM	See "Data Memory (DMEM)" on page 106
0xBC*n*04000 to 0xBC*n*0403C	RxSDP0_Ext0 to RxSDP0_Ext15	CP Rx Extract Space0	See page 492
0xBC*n*04080	RxCB0_Sys_Addr	CP Rx Control Block0	See page 493
0xBC*n*04084	RxCB0_Ctl		See page 494
0xBC*n*04088	RxCB0_DMA_Addr		See page 497
0xBC*n*0408C	RxCB0_SDP_Addr		See page 498
0xBC*n*04090	RxCtl0_Status		See page 498
0xBC*n*04100 to 0xBC*n*0413C	TxSDP0_Merge0 to TxSDP0_Merge15	CP Tx Merge Space0	See page 492
0xBC*n*04180	TxCB0_Sys_Addr	CP Tx Control Block0	See page 505
0xBC*n*04184	TxCB0_Ctl		See page 506
0xBC*n*04188	TxCB0_DMA_Addr		See page 507
0xBC*n*0418C	TxCB0_SDP_Addr		See page 508
0xBC*n*04190	TxCtl0_Status		See page 509
0xBC*n*04200 to 0xBC*n*0423C	RxSDP1_Ext0 to RxSDP1_Ext15	CP Rx Extract Space1	See Table 144 on page 492

Table 143 CP Registers (continued)

ADDRESS	REGISTER NAME	FUNCTION	DETAILED PARAMETERS
0xBC*n*04280	RxCB1_Sys_Addr	CP Rx Control Block1	See Table 146 on page 493
0xBC*n*04284	RxCB1_Ctl		See Table 148 on page 496
0xBC*n*04288	RxCB1_DMA_Addr		See Table 149 on page 497
0xBC*n*0428C	RxCB1_SDP_Addr		See Table 150 on page 498
0xBC*n*04290	RxCtl1_Status		See Table 151 on page 499
0xBC*n*04300 to 0xBC*n*0433C	TxSDP1_Merge0 to TxSDP1_Merge15	CP Tx Merge Space1	See Table 145 on page 493
0xBC*n*04380	TxCB1_Sys_Addr	CP Tx Control Block1	See Table 158 on page 505
0xBC*n*04384	TxCB1_Ctl		See Table 159 on page 507
0xBC*n*04388	TxCB1_DMA_Addr		See Table 160 on page 508
0xBC*n*0438C	TxCB1_SDP_Addr		See Table 161 on page 508
0xBC*n*04390	TxCtl1_Status		See Table 162 on page 509
0xBC*n*04400	WrCB0_Sys_Addr	CP Wr Control Block0	See page 499
0xBC*n*04404	WrCB0_Ctl		See page 500
0xBC*n*04408	WrCB0_DMA_Addr		See page 501
0xBC*n*04410	WrCB1_Sys_Addr	CP Wr Control Block1	See Table 152 on page 499
0xBC*n*04414	WrCB1_Ctl		See Table 153 on page 501
0xBC*n*04418	WrCB1_DMA_Addr		See Table 154 on page 501
0xBC*n*04420	RdCB0_Sys_Addr	CP Rd Control Block0	See page 502
0xBC*n*04424	RdCB0_Ctl		See page 503
0xBC*n*04428	RdCB0_DMA_Addr		See page 504
0xBC*n*04430	RdCB1_Sys_Addr	CP Rd Control Block1	See Table 155 on page 502
0xBC*n*04434	RdCB1_Ctl		See Table 156 on page 504
0xBC*n*04438	RdCB1_DMA_Addr		See Table 157 on page 504

Table 143 CP Registers (continued)

ADDRESS	REGISTER NAME	FUNCTION	DETAILED PARAMETERS
0xBC*n*04440	TxMsg0_Ctl	Ring Bus Tx Message Control	See page 510
0xBC*n*04448	TxMsg1_Ctl		See Table 164 on page 511
0xBC*n*04450	TxMsg2_Ctl		
0xBC*n*04458	TxMsg3_Ctl		
0xBC*n*04460	TxMsg0_Data_H		See page 512
0xBC*n*04464	TxMsg0_Data_L		See page 512
0xBC*n*04468	TxMsg1_Data_H		See Table 165 on page 512
0xBC*n*0446C	TxMsg1_Data_L		See Table 166 on page 512
0xBC*n*04470	TxMsg2_Data_H		See Table 165 on page 512
0xBC*n*04474	TxMsg2_Data_L		See Table 166 on page 512
0xBC*n*04478	TxMsg3_Data_H		See Table 165 on page 512
0xBC*n*0447C	TxMsg3_Data_L		See Table 166 on page 512
0xBC*n*04480	RxResp0_Ctl	Ring Bus Rx Response Control	See page 513
0xBC*n*04484	RxResp1_Ctl		See Table 167 on page 513
0xBC*n*04488	RxResp2_Ctl		
0xBC*n*0448C	RxResp3_Ctl		
0xBC*n*04490	RxResp4_Ctl		
0xBC*n*04494	RxResp5_Ctl		
0xBC*n*04498	RxResp6_Ctl		
0xBC*n*0449C	RxResp7_Ctl		
0xBC*n*044A0	RxResp0_Data_H		See page 514
0xBC*n*044A4	RxResp0_Data_L		See page 515
0xBC*n*044A8	RxResp1_Data_H		See Table 168 on page 514
0xBC*n*044AC	RxResp1_Data_L		See Table 169 on page 515

Table 143 CP Registers (continued)

ADDRESS	REGISTER NAME	FUNCTION	DETAILED PARAMETERS
0xBC*n*044B0	RxResp2_Data_H	Ring Bus Rx Response Control (continued)	See Table 168 on page 514
0xBC*n*044B4	RxResp2_Data_L		See Table 169 on page 515
0xBC*n*044B8	RxResp3_Data_H		See Table 168 on page 514
0xBC*n*044BC	RxResp3_Data_L		See Table 169 on page 515
0xBC*n*044C0	RxResp4_Data_H		See Table 168 on page 514
0xBC*n*044C4	RxResp4_Data_L		See Table 169 on page 515
0xBC*n*044C8	RxResp5_Data_H		See Table 168 on page 514
0xBC*n*044CC	RxResp5_Data_L		See Table 169 on page 515
0xBC*n*044D0	RxResp6_Data_H		See Table 168 on page 514
0xBC*n*044D4	RxResp6_Data_L		See Table 169 on page 515
0xBC*n*044D8	RxResp7_Data_H		See Table 168 on page 514
0xBC*n*044DC	RxResp7_Data_L		See Table 169 on page 515
0xBC*n*044E0	RxMsg_Ctl	Ring Bus Rx Message Control	See page 515
0xBC*n*044E4	RxMsg_FIFO		See page 517
0xBC*n*04500 to 0xBC*n*0457C	Rx_SONETOH0 to Rx_SONETOH31	SONET Rx Control	See page 517
0xBC*n*04580 to 0xBC*n*045FC	Tx_SONETOH0 to Tx_SONETOH31	SONET Tx Control	See page 517
0xBC*n*04600	RxCtl_ByteSeq0	SDP Rx Control	See page 518
0xBC*n*04604	RxCtl_ByteSeq1		See Table 170 on page 518
0xBC*n*04608	RxCtl_SyncSeq		See page 518
0xBC*n*0460C	RxCtl_BitSeq0		See page 518
0xBC*n*04610	RxCtl_BitSeq1		See Table 171 on page 519
0xBC*n*04620	TxCtl_ByteSeq0	SDP Tx Control	See page 519
0xBC*n*04624	TxCtl_ByteSeq1		See Table 172 on page 519
0xBC*n*0462C	TxCtl_BitSeq0	SDP Tx Control (continued)	See page 519
0xBC*n*04630	TxCtl_BitSeq1		See Table 173 on page 519

Table 143 CP Registers (continued)

ADDRESS	REGISTER NAME	FUNCTION	DETAILED PARAMETERS
0xBC*n*04640	CP_Mode0	CP Mode Configuration	See page 520
0xBC*n*04644	CP_Mode1		See page 523
0xBC*n*04648	SDP_Mode2		See page 526
0xBC*n*0464C	SDP_Mode3		See page 529
0xBC*n*04650	SDP_Mode4		See page 536
0xBC*n*04654	SDP_Mode5		See page 538
0xBC*n*04658	Debug_Mode		See page 544
0xBC*n*0465C	PIN_Mode		See page 546
0xBC*n*04660	Queue_Status0	CP Queue Status	See page 549
0xBC*n*04664	Queue_Status1		See Table 177 on page 549
0xBC*n*04668	Queue_Status2		
0xBC*n*0466C	Queue_Status3		
0xBC*n*04670	Queue_Update0		See page 550
0xBC*n*04674	Queue_Update1		See Table 178 on page 550
0xBC*n*04678	Queue_Update2		
0xBC*n*0467C	Queue_Update3		
0xBC*n*04680	Queue_Empty	Aggregated Queueing	See page 550
0xBC*n*04684	Event_Timer	CP Miscellaneous Control	See page 550
0xBC*n*04688	Cycle_Count_H		See page 551
0xBC*n*0468C	Cycle_Count_L		See page 551
0xBC*n*04690	Queue_Ctl	Aggregated Queueing	See page 551

Table 143 CP Registers (continued)

ADDRESS	REGISTER NAME	FUNCTION	DETAILED PARAMETERS
0xBC*n*046A0	Event0	CP Event and Interrupt Control	See page 552
0xBC*n*046A4	Event1		See page 555
0xBC*n*046A8	Event_Mask0		See page 557
0xBC*n*046AC	Event_Mask1		See Table 179 on page 557
0xBC*n*046B0	Event_Access		See page 557
0xBC*n*046B4	Mask_Access		See page 559
0xBC*n*046B8	Interrupt_Mask0		See page 559
0xBC*n*046BC	Interrupt_Mask1		See Table 180 on page 560
0xBC*n*046C0	SONET_Event		See page 560
0xBC*n*046C4	SONET_Mask		See page 569
0xBC*n*04700	RdCB0_BTag_Alloc	CP Rd Control Block0 Fixed	See page 569
0xBC*n*04704	RdCB0_Dequeue		See page 570
0xBC*n*04710	RdCB1_BTag_Alloc	CP Rd Control Block1 Fixed	See Table 182 on page 569
0xBC*n*04714	RdCB1_Dequeue		See Table 183 on page 570
0xBC*n*04720	WrCB0_BTag_Dealloc	CP Wr Control Block0 Fixed	See page 571
0xBC*n*04724	WrCB0_MUC_Alloc		See page 571
0xBC*n*04728	WrCB0_MUC_Decr		See page 572
0xBC*n*04730	WrCB0_Uni_Enq		See page 573
0xBC*n*04734	WrCB0_Multi_Enq		See page 574
0xBC*n*04738	WrCB0_Spec_Uni_Enq		See page 575
0xBC*n*04740	WrCB1_BTag_Dealloc	CP Wr Control Block1 Fixed	See Table 184 on page 571
0xBC*n*04744	WrCB1_MUC_Alloc		See Table 185 on page 572
0xBC*n*04748	WrCB1_MUC_Decr		See Table 186 on page 572
0xBC*n*04750	WrCB1_Uni_Enq		See Table 187 on page 573
0xBC*n*04754	WrCB1_Multi_Enq		See Table 188 on page 574
0xBC*n*04758	WrCB1_Spec_Uni_Enq		See Table 189 on page 575

CP Detailed Descriptions

The following is a detailed description of each of the CP registers and their individual parameters. The detailed parameters provide: purpose, field name, bit positions and descriptions.

RxSDP0_Ext0 to RxSDP0_Ext15 Registers (CP Rx Extract Space0 Function)

Purpose — Used to pass fields extracted from the receive data stream by the RxSDP to the CPRC. These registers are used only for receive datascope0. See Table 144 on page 492 for similar registers.

Address — 0xBC*n*04000 to 0xBC*n*0403C

Access — CPRC Read, CPRC Write only writable for test purposes when SDP_Mode3 RxResetx==0, SDP RxByte Processor Write - byte addressable.

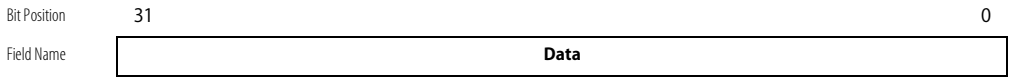


Table 144 RxSDP1_Ext0 to RxSDP1_Ext15 Registers (for Datascope1)

REGISTER NAME	PURPOSE	ADDRESS
RxSDP1_Ext0 to RxSDP1_Ext15	Same as registers *RxSDP0_Ext0* to *RxSDP0_Ext15*, but for datascope1.	0xBC*n*04200 to 0xBC*n*0423C

TxSDP0_Merge0 to TxSDP0_Merge15 Registers (CP Tx Merge Space0 Function)

Purpose — Used to pass fields from the CPRC to the TxSDP to merge in with the transmit data stream. These registers are used only for transmit datascope0. See Table 145 on page 493 for similar registers.

Address — 0xBC*n*04100 to 0xBC*n*0413C

Access — CPRC Read, CPRC Write, SDP TxByte Processor Read - byte addressable

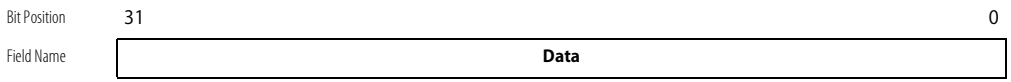


Table 145 TxSDP1_Merge0 to TxSDP1_Merge15 Registers (for Datascope1)

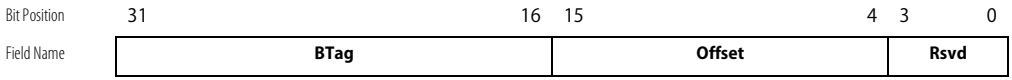
REGISTER NAME	PURPOSE	ADDRESS
TxSDP1_Merge0 to TxSDP1_Merge15	Same as registers *TxSDP0_Merge0* to *TxSDP0_Merge15*, but pertains to transmit datascope1.	0xBC*n*04300 to 0xBC*n*0433C

RxCB0_Sys_Addr Register (CP Rx Control Block0 Function)

Purpose Provides an address consisting of a Pool ID, BTag and offset for datascope0. See Table 146 on page 493 for similar register.

Address 0xBC*n*04080

Access CPRC Read/Write



FIELD NAME	BIT POSITION	DESCRIPTION
BTag	31:16	**Buffer Tag** — Address. Legal range is a physical limit= 0 to 65,532Bytes or 0 to 0xFFFF.
Offset	15:4	**Offset** — Address or Command. Refer to Table 23 on page 145 Legal range= 0 to 65,520Bytes, or 0 to 0xFFF0. Values must be 16Byte aligned.
Reserved	3:0	Read as zero.

Table 146 RxCB1_Sys_Addr Register (for Datascope1)

REGISTER NAME	PURPOSE	ADDRESS
RxCB1_Sys_Addr	Same as*RxCB0_Sys_Addr*, except for datascope1.	0xBC*n*04280

RxCB0_Ctl Register (CP Rx Control Block0 Function)

Purpose Controls DMA for payload receive operation for datascope0. See Table 148 on page 496 for similar register.

Address 0xBC*n*04084

Access CPRC Read/Write

Bit Position	31	30	29	28	27 24	23	22	21 20	19	18	17 16	15 0
Field Name	Avail	NoRetry	EOP	Protect EOP	Error	Own 1	Own 0	Ctx	SDP state	Rsvd	DMA state	RxLength
Reset Value	1	x	0	raz	x	x	x	x	0	raz	00	x

FIELD NAME	BIT POSITION	DESCRIPTION
Avail	31	**Availability Bit** — 1=RxCB is available to the CPRC, 0=Start the DRAM DMA engine.
NoRetry	30	**No Retry** — 1=Do not retry the transaction on bus NACK, 0=Retry, up to 16 (Max.) times before reporting an error.
EOP	29	**End-of-Packet** — Typically this is set by the SDP when scope is switched and cleared by the DMA engine when a transfer completes successfully.
Protect EOP	28	**Protect End-of-Packet** — When set during a RxCB_Ctl write, the EOP bit contained in field [29] is not written. When cleared during a RxCB_Ctl write, the EOP bit is written.
Error	27:24	**Error** — When a DMA operation completes and the *Avail* bit [31] =1, then a non-zero value indicates that the DMA operation completed with an error. Refer to Table 147 on page 495 for error code definitions.
Own1	23	**Block Ownership Bit** — 0=SDP owns, 1= DMA owns.
Own0	22	**Block Ownership Bit** — 0=SDP owns, 1= DMA owns.
Ctx	21:20	**Context** — Two bit field that software can use to provide context or identifying information per requests. When the Avail bit is cleared to launch the DRAM DMA operation the Ctx field is automatically set with the current CPRC Register File Context number (0 to 3). This filed has no impact on the operation.
SDP state	19	**SDP State** — Shows the state of the SDP engine: 0=Ready, 1=Wait for line update. During a RxCB*n*_Ctl write, this field is updated if *Avail* bit [31] is set, and not changed if *Avail* bit [31] is clear.
Reserved	18	Read as zero.

FIELD NAME	BIT POSITION	DESCRIPTION
State	17:16	**DMA State** — Shows the state of the DMA engine: 00 = Idle,10 = Request, 01 = Grant. During a RxCB*n*_Ctl write, this field is updated if *Avail* bit [31] is set, and not changed if *Avail* bit [31] is clear.
RxLength	15:0	**Receive Length** — Count of bytes in the receive payload.

Table 147 Transfer Control Block Error Codes

ERROR TYPE	ENCODING	READ/ WRITE	TARGET	DESCRIPTION
Success	0	Wr, Rx, Rd, Tx	Buffer memory: PID== 0-29 BMU:PID==30 QMU:PID==31	The payload bus transaction completed successfully.
RxSDP Error	8	Rx	Not Applicable	The SDP RxByte sequencer indicated an error by writing to the RxCtl_Status bit [30].
NACK Retry Limit	9	Wr, Rx, Rd, Tx	Buffer memory: PID== 0-29	The BMU was unable to accept a buffer memory transaction. The payload bus transaction was attempted until the NACK retry limit was reached.
NACK Retry Limit	9	Wr	BMU: PID==30	The BMU was unable to accept BTag command, or the multi-use counter table was full on an allocate command. The payload bus transaction was attempted until the NACK retry limit was reached.
NACK Retry Limit	9	Rd	BMU: PID==30	The BMU was unable to accept BTag or multi-use counter command. The payload bus transaction was attempted until the NACK retry limit was reached.
NACK Retry Limit	9	Wr	QMU:PID==31	The QMU was unable to accept a command because the write mailbox was full. The payload bus transaction was attempted until the NACK retry limit was reached.
NACK Retry Limit	9	Rd	QMU:PID==31	The QMU was unable to accept a command because the read mailbox was full. The payload bus transaction was attempted until the NACK retry limit was reached.
Bad Pool	A	Rd	BMU:PID==30	BTag allocate command requested a non-existent pool.

Table 147 Transfer Control Block Error Codes (continued)

ERROR TYPE	ENCODING	READ/ WRITE	TARGET	DESCRIPTION
Bad BMU Command	A	Wr	BMU:PID==30	Any of the following conditions occurred: • BMU command requested a non-existent pool, • BTag write found more BTags written than configured, • Multi-use counter allocate found a counter already allocated for this pool/BTag, • Multi-use counter decrement to a non-existent counter.
BTag Unavailable	C	Rd	BMU:PID==30	BTag allocate command found insufficient BTags to complete the allocation.
QMU Read Error	C	Rd	QMU:PID==31	QMU detected a dequeue of an empty queue.
Payload ECC Error	D	Rd	Buffer memory: PID==0-29	Un-correctable ECC error occurred on a buffer memory read.
BTag ECC Error	D	Rd	BMU:PID==30	Uncorrected ECC error occurred then reading memory for a BTag allocate command.
Non-Existent memory	E	Rd	Buffer memory: PID==0-29	A payload read of buffer memory that was out of bounds due to an error in the way software configured the BMU.
Non-Existent memory	E	Rd	BMU:PID==30	A BTag read from memory was out of bounds due to an error in the way software configured the BMU.
No Match on Multi-use Counter Read	F	Rd	BMU:PID==30	A multi-use counter read command of a non-existent counter.

Table 148 RxCB1_Ctl Register (for Datascope1)

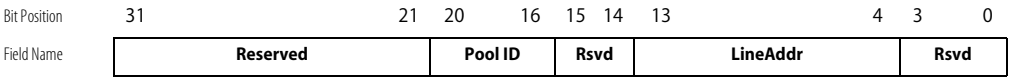
REGISTER NAME	PURPOSE	ADDRESS
RxCB1_Ctl	Same as *RxCB0_Ctl*, except for datascope1.	0xBC*n*04284

RxCB0_DMA_Addr Register (CP Rx Control Block0 Function)

Purpose Supplies the address of a 16Byte line in DMEM for DMA and the Pool ID of the buffer to write for datascope0. See Table 149 on page 497 for similar register.

Address 0xBC*n*04088

Access CPRC Read/Write



FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:21	Read as zero.
PoolID	20:16	**Pool ID** — Pool to write too. Legal range= 0 to 31.
Reserved	15:14	Read as zero.
LineAddr	13:4	**DMEM Line Address** — DMEM 16Byte line address for DMA. It is auto-incremented during DMA; bits [6:4] cleared by the DMA engine when a transfer completes successfully.
Reserved	3:0	Read as zero.

Table 149 RxCB1_DMA_Addr Register (for Datascope1)

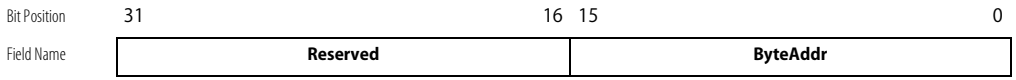
REGISTER NAME	PURPOSE	ADDRESS
RxCB1_DMA_Addr	Same as RxCB0_DMA_Addr, except for datascope1.	0xBC*n*04288

RxCB0_SDP_Addr Register (CP Rx Control Block0 Function)

Purpose Supplies the address of a byte in DMEM for DMA. It is auto-incremented during DMA for datascope0. See Table 150 on page 498 for similar register.

Address 0xBC*n*0408C

Access CPRC Read/Write



FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:16	Read as zero.
ByteAddr	15:0	**DMEM Byte Address** — DMEM byte address for DMA. It is auto-increment during DMA; bits [6:0] cleared by the DMA engine when a transfer completes successfully.

Table 150 RxCB1_SDP_Sys_Addr Register (for Datascope1)

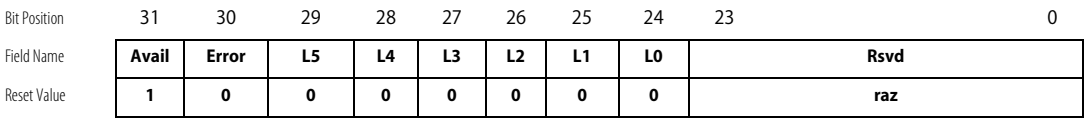
REGISTER NAME	PURPOSE	ADDRESS
RxCB1_SDP_Sys_Addr	Same as *RxCB0_SDP_Addr*, except for datascope1.	0xBC*n*0428C

RxCtl0_Status Register (CP Rx Control Block0 Function)

Purpose Semaphores governing SDP receive operation for datascope0. See Table 151 on page 499 for similar register.

Address 0xBC*n*04090

Access CPRC Read/Write, SDP Receive Byte Sequencer Read/Write - byte addressable



FIELD NAME	BIT POSITION	DESCRIPTION
Avail	31	**Availability Bit** — When the bit is 1, CPRC owns. When the bit is 0, SDP owns the scope.
Error	30	**RxSDP Error** — SDP sets this bit to indicate an error during receive processing.
L5 to L0	29:24	**SDP Level Bits** — SDP sets the corresponding bit to indicate level of processing, software defined.
Reserved	23:0	Read as zero.

Table 151 RxCtl1_Status Register (for Datascope1)

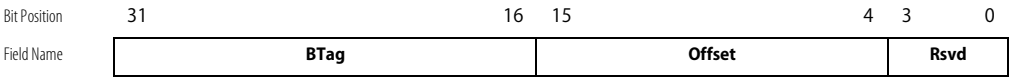
REGISTER NAME	PURPOSE	ADDRESS
RxCtl1_Status	Same as *RxCtl0_Status*, except for datascope1.	0xBC*n*04290

WrCB0_Sys_ Addr Register (CP Wr Control Block0 Function)

Purpose Provides an address consisting of a Pool ID, BTag and offset. See Table 152 on page 499 for similar register.

Address 0xBC*n*04400

Access CPRC Read/Write



FIELD NAME	BIT POSITION	DESCRIPTION
BTag	31:16	**Buffer Tag** — Legal range is a physical limit= 0 to 65,532Bytes or 0 to 0xFFFF.
Offset	15:4	**Offset** — Address or Command. Refer to "Using Multi-Use Control Blocks to Achieve Different Functions" on page 144. Legal range= 0 to 65,520Bytes, or 0 to 0xFFFF. Values must be 16Bytes aligned.
Reserved	3:0	Read as zero.

Table 152 WrCB1_Sys_Addr Register (for Control Block1)

REGISTER NAME	PURPOSE	ADDRESS
WrCB1_Sys_Addr	Same as *WrCB0_Sys_Addr*, except for control block1.	0xBC*n*04410

WrCB0_Ctl Register (CP Wr Control Block0 Function)

Purpose Controls DMA for payload write operation. See Table 153 on page 501 for similar register.

Address 0xBC*n*04404

Access CPRC Read/Write

Bit Position	31	30	29	28	27 24	23 22	21 20	19 18	17 16	15 14	13 0
Field Name	Avail	NoRetry	Mod64	Rsvd	Error	Rsvd	Ctx	Rsvd	State	Rsvd	Length
Reset Value	1	x	x	raz	x	raz	x	raz	0	raz	x

FIELD NAME	BIT POSITION	DESCRIPTION
Avail	31	**Availability Bit** — 1= WrCB is available to the CPRC, 0=Start the DRAM DMA engine.
NoRetry	30	**No Retry** — 1= Do not retry the transaction on bus NACK, 0=Retry 16 (Max.) times before reporting an error.
Mod64	29	**Modulo 64** — 1=Increment *WrCB0_Sys_Addr* bits [15:4] *Offset* field and *WrCB0_DMA_Addr* bits [13:4] *LineAddr* field modulo 64Bytes during DMA to perform a wrap. 0=Increment *WrCB0_Sys_Addr* bits [15:4] *Offset* field and *WrCB0_DMA_Addr* bits [13:4] *LineAddr* field linearly during DMA that steps through memory.
Reserved	28	Read as zero.
Error	27:24	**Error** — When *Avail* bit [31]=1 and a non-zero value is returned after a DMA operation completes, the DMA operation encountered an error. See Table 147 on page 495 for error code definitions.
Reserved	23:22	Read as zero.
Ctx	21:20	**Context** — Two bit field that software can use to provide context or identifying information per requests. When the Avail bit is cleared to launch the DRAM DMA operation the Ctx field is automatically set with the current CPRC Register File Context number (0 to 3). This filed has no impact on the operation.
Reserved	19:18	Read as zero.
State	17:16	**DMA State** — Shows the state of the DMA engine: 00 = Idle, 10 = Request, 01 = Grant. During a WrCB*n*_Ctl write, this field is updated if *Avail* bit [31] is set and not changed if *Avail* bit [31] is clear.
Reserved	15:14	Read as zero.

FIELD NAME	BIT POSITION	DESCRIPTION
Length	13:0	**Length** — Length of DMA transfer in Bytes. Legal range is a physical limit=12Kbytes.

Table 153 WrCB1_Ctl Register (for Control Block1)

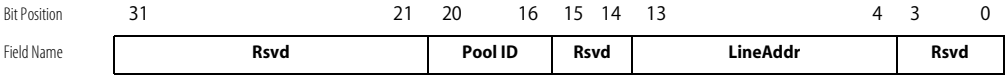
REGISTER NAME	PURPOSE	ADDRESS
WrCB1_Ctl	Same as *WrCB0_Ctl*, except for control block1.	0xBC*n*04414

WrCB0_DMA_Addr Register (CP Wr Control Block0 Function)

Purpose Supplies the address of a 16Byte line in DMEM for DMA and the Pool ID of the buffer to write. See Table 154 on page 501 for similar register.

Address 0xBC*n*04408

Access Global Read/Write



FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:21	Read as zero.
PoolID	20:16	**Pool ID** — Pool to write too. Legal range= 0 to 31.
Reserved	15:14	Read as zero.
LineAddr	13:4	**DMEM Line Address** — DMEM 16Byte line address for DMA. It is auto-increment during DMA.
Reserved	3:0	Read as zero.

Table 154 WrCB1_DMA_Addr Register (for Control Block1)

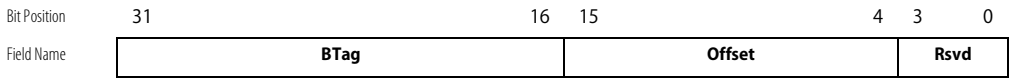
REGISTER NAME	PURPOSE	ADDRESS
WrCB1_DMA_Addr	Same as *WrCB0_DMA_Addr*, except for control block1.	0xBC*n*04418

RdCB0_Sys_Addr Register (CP Rd Control Block0 Function)

Purpose Provides an address consisting of a pool ID, BTag, and offset. See Table 155 on page 502 for similar register.

Address 0xBC*n*04420

Access CPRC Read/Write



FIELD NAME	BIT POSITION	DESCRIPTION
BTag	31:16	**Buffer Tag** — Address Legal range is a physical limit= 0 to 65,532Bytes or 0 to 0xFFFF.
Offset	15:4	**Offset** — Address or Command. Refer to "Using Multi-Use Control Blocks to Achieve Different Functions" on page 144. Legal range= 0 to 65,520Bytes, or 0 to 0xFFFF. Values must be 16Bytes aligned.
Reserved	3:0	Read as zero.

Table 155 RdCB1_Sys_Addr register (for Control Block1)

REGISTER NAME	PURPOSE	ADDRESS
RdCB1_Sys_Addr	Same as RdCB0_Sys_Addr, except for control block1.	0xBC*n*04430

RdCB0_Ctl Register (CP Rd Control Block0 Function)

Purpose Controls DMA for payload read operation. See Table 156 on page 504 for similar register.

Address 0xBC*n*04424

Access CPRC Read/Write

Bit Position	31	30	29	28	27 24	23 22	21 20	19 18	17 16	15 14	13 4	3 0
Field Name	Avail	NoRetry	Mod64	Rsvd	Error	Rsvd	Ctx	Rsvd	State	Rsvd	Length	Rsvd
Reset Value	1	x	x	raz	x	raz	x	raz	0	raz	x	

FIELD NAME	BIT POSITION	DESCRIPTION
Avail	31	**Availability Bit** — 1=RdCB is available to the CPRC, 0=Start the DRAM DMA engine.
NoRetry	30	**No Retry** — 1=Do not retry the transaction on bus NACK, 0=retry, up to 16 (Max.) times.
Mod64	29	**Modulo 64** — 1=Increment *RdCB0_Sys_Addr* bits [15:4] *Offset* field and *RdCB0_DMA_Addr* bits [13:4] *LineAddr* field modulo 64Bytes during DMA to perform a wrap. 0=Increment *RdCB0_Sys_Addr* bits [15:4] *Offset* field and *RdCB0_DMA_Addr* bits [13:4] *LineAddr* field linearly during DMA that steps through memory.
Reserved	28	Read as zero.
Error	27:24	**Error** — When *Avail* bit [31]=1 and a non-zero value is returned after a DMA operation completes, the DMA operation encountered an error. Refer to Table 147 on page 495 for error code definitions.
Reserved	23:22	Read as zero.
Ctx	21:20	**Context** — Two bit field that software can use to provide context or identifying information per requests. When the Avail bit is cleared to launch the DRAM DMA operation the Ctx field is automatically set with the current CPRC Register File Context number (0 to 3). This filed has no impact on the operation.
Reserved	19:18	Read as zero.
State	17:16	**DMA State** — Shows the state of the DMA engine (read only): 00 =Idle, 10 = Request, 01 = Grant. During a RdCB*n*_Ctl write, this field is updated if *Avail* bit [31] is set and not changed if *Avail* bit [31] is clear.

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	15:14	Read as zero.
Length	13:0	**Length** — Length of DMA transfer in bytes. Legal range is a physical limit=12KBytes.

Table 156 RdCB1_Ctl Register (for Control Block1)

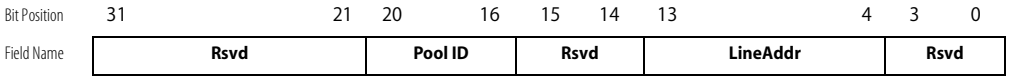
REGISTER NAME	PURPOSE	ADDRESS
RdCB1_Ctl	Same as *RdCB0_Ctl*, except for control block1.	0xBC*n*04434

RdCB0_DMA_Addr Register (CP Rd Control Block0 Function)

Purpose Supplies the address of a 16Byte line in DMEM for DMA and the Pool ID for buffer to read. See Table 157 on page 504 for similar register.

Address 0xBC*n*04428

Access CPRC Read/Write



FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:21	Read as zero.
PoolID	20:16	**Pool ID** — Pool to read too. Legal range= 0 to 31.
Reserved	15:14	Read as zero.
LineAddr	13:4	**DMEM Line Address** — DMEM 16Byte line address for DMA operation. It is auto-incremented during DMA.
Reserved	3:0	Read as zero.

Table 157 RdCB1_DMA_Addr Register (for Control Block1)

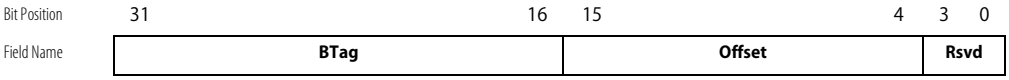
REGISTER NAME	PURPOSE	ADDRESS
RdCB1_DMA_Addr	Same as RdCB0_DMA_Addr, except for control block1.	0xBC*n*04438

TxCB0_Sys _Addr Register (CP Tx Control Block0 Function)

Purpose Provides an address consisting of a Pool ID, BTag and offset for datascope0. See Table 158 on page 505 for similar register.

Address 0xBC*n*04180

Access CPRC Read/Write



FIELD NAME	BIT POSITION	DESCRIPTION
BTag	31:16	**Buffer Tag** — Address Legal range is a physical limit= 0 to 65,532Bytes or 0 to 0xFFFF.
Offset	15:4	**Offset** — Address or Command. Refer to "Using Multi-Use Control Blocks to Achieve Different Functions" on page 144. Legal range= 0 to 65,520Bytes, or 0 to 0xFFF0. Values must be 16Byte aligned.
Reserved	3:0	Read as zero.

Table 158 TxCB1_Sys_Addr Register (for Datascope1)

REGISTER NAME	PURPOSE	ADDRESS
TxCB1_Sys_Addr	Same as *TxCB0_Sys_Addr*, except for datascope1.	0xBC*n*04380

TxCB0_Ctl Register (CP Tx Control Block0 Function)

Purpose Controls DMA for payload transmit operation for datascope0. See Table 159 on page 507 for similar register.

Address 0xBC*n*04184

Access CPRC Read/Write

Bit Position	31	30	29	28	27 24	23	22	21 20	19 18	17 16	15 0
Field Name	Avail	NoRetry	EOP	OOB	Error	Own1	Own0	Ctx	SDP state	DMA state	TxLength
Reset Value	1	x	0	x	x	0	0	x	0	00	x

FIELD NAME	BIT POSITION	DESCRIPTION
Avail	31	**Availability Bit** — 1=TxCB is available to the CPRC, 0=Start the DRAM DMA engine.
NoRetry	30	**No Retry** — 1=Do not retry the transaction on bus NACK, 0=Retry, up to 16 (Max.) times before reporting an error.
EOP	29	**End of Packet** — Typically this is set by the SDP when scope is switched and cleared by the DMA engine when a transfer completes successfully. During a TxCB*n*_Ctl write, this field is updated if *Avail* bit [31] is set, and not changed if *Avail* bit [31] is clear.
OOB	28	**Out of Band** — 1=Use out-of-band (OOB) bits, 0=Use the length field to determine end-of-frame.
Error	27:24	**Error** — When a DMA operation completes and the *Avail* bit [31] =1, then a non-zero value indicates that the DMA operation completed with an error. Refer to Table 147 on page 495 for error code definitions.
Own1	23	**Block Ownership Bit** — 1=SDP owns, 0=DMA owns.
Own0	22	**Block Ownership Bit** — 1=SDP owns, 0=DMA owns.
Ctx	21:20	**Context** — Two bit field that software can use to provide context or identifying information per requests. When the Avail bit is cleared to launch the DRAM DMA operation the Ctx field is automatically set with the current CPRC Register File Context number (0 to 3). This filed has no impact on the operation.

FIELD NAME	BIT POSITION	DESCRIPTION
SDP state	19:18	**SDP State** — Shows the state of the SDP engine for CPs , or PCI engine for XP. 0=Waiting for data, Non-zero value= Ready for transmit. During a TxCB*n*_Ctl write, this field is updated if *Avail* bit [31] is set, and not changed if *Avail* bit [31] is clear.
State	17:16	**DMA State** — Shows the state of the DMA engine: 00 = Idle,10 = Request, 01 = Grant. During a TxCB*n*_Ctl write, this field is updated if *Avail* bit [31] is set, and not changed if *Avail* bit [31] is clear.
TxLength	15:0	**Transmit Length** — Counts down the bytes of transmit payload.

Table 159 TxCB1_Ctl Register (for Datascope1)

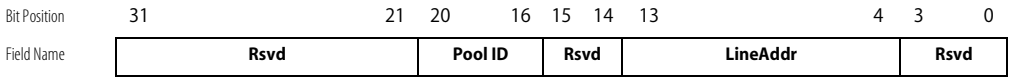
REGISTER NAME	PURPOSE	ADDRESS
TxCB1_Ctl	Same as *TxCB0_Ctl*, except for datascope1.	0xBC*n*04384

TxCB0_DMA_Addr Register (CP Tx Control Block0 Function)

Purpose — Supplies the address of a 16Byte line in DMEM for DMA and the Pool ID of buffer to read for datascope0. See Table 160 on page 508 for similar register.

Address — 0xBC*n*04188

Access — CPRC Read/Write



FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:21	Read as zero.
Pool ID	20:16	**Pool ID** — Pool to read too. Legal range= 0 to 31.
Reserved	15:14	Read as zero.
LineAddr	13:4	**DMEM Line Address** — DMEM 16Byte line address for DMA. It is auto-incremented during DMA; bits [6:4] cleared by DMA engine when a transfer completes successfully.
Reserved	3:0	Read as zero.

Table 160 TxCB1_DMA_Addr Register (for Datascope1)

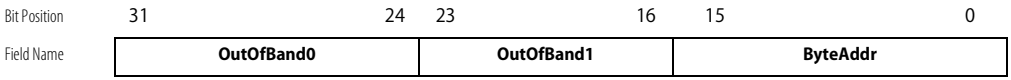
REGISTER NAME	PURPOSE	ADDRESS
TxCB1_DMA_Addr	Same as *TxCB0_DMA_Addr*, except for datascope1.	0xBC*n*04388

TxCB0_SDP_Addr Register (CP Tx Control Block0 Function)

Purpose Supplies the address of a byte in DMEM for DMA for transmit datascope0. See Table 161 on page 508 for similar register.

Address 0xBC*n*0418C

Access CPRC Read/Write



FIELD NAME	BIT POSITION	DESCRIPTION
OutOfBand0	31:24	**Out of Band0** — The eight (OOB) bits accompanying DMEM buffer 0.
OutOfBand1	23:16	**Out of Band1** — The eight (OOB) bits accompanying DMEM buffer 1.
ByteAddr	15:0	**DMEM Byte Address** — DMEM byte address for DMA. It is auto-incremented during DMA; bits [6:0] cleared by the DMA engine when a transfer completes successfully.

Table 161 TxCB1_SDP_Addr Register (for Datascope1)

REGISTER NAME	PURPOSE	ADDRESS
TxCB1_SDP_Addr	Same as *TxCB0_SDP_Addr*, except for datascope1.	0xBC*n*0438C

TxCtl0_Status Register (CP Tx Control Block0 Function)

Purpose Semaphores governing SDP transmit operation for data cope0. See Table 162 on page 509 for similar register.

Address 0xBC*n*04190

Access Read/Write

Bit Position	31	30	29	28	27	26	25	24	23 ... 0
Field Name	Avail	Error	L5	L4	L3	L2	L1	L0	Rsvd
Reset Value	1	x	x	x	x	x	x	x	raz

FIELD NAME	BIT POSITION	DESCRIPTION
Avail	31	**Availability Bit** — When the bit is 1, CPRC owns. When the bit is 0, SDP owns the scope.
Error	30	**TxSDP Error** — SDP sets this bit to indicate and error during transmit processing.
L5 - L0	29:24	**SDP Level Bits** — SDP sets the corresponding bit to indicate level of processing, software defined.
Reserved	23:0	Read as zero.

Table 162 TxCtl1_Status Register (for Datascope1)

REGISTER NAME	PURPOSE	ADDRESS
TxCtl1_Status	Same as *TxCtl0_Status*, except for datascope1.	0xBC*n*04390

TxMsg0_Ctl Register (CP Ring Bus Tx Message Control Function)

Purpose — Provides the control portion of an outgoing Ring Bus message. See Table 164 on page 511 for similar registers.

Address — 0xBC*n*04440

Access — CPRC Read/Write = Byte addressable, SDP Receive Byte Sequence: Write = field addressable (all fields), Read = field addressable (Available bit)

Bit Position	31	30 24	23	22 20	19 18	17 15	14 10	9 5	4 0
Field Name	Avail	Rsvd	Error	Rsvd	Type	Len	Seq	Dst	Src
Reset Value	1	raz	x	raz	x	x	x	x	x

FIELD NAME	BIT POSITION	DESCRIPTION
Avail	31	**Availability Bit** — When the bit is 1, slot is available to the CPRC or SDP. When the bit is 0, start the Ring Bus transmit engine.
Reserved	30:24	Read as zero.
ErrorFlag	23	**Error Flag** — Error bit of transmit message definable by the transmitter.
Reserved	22:20	Read as zero.
Type	19:18	**Transmit Message Type**: ENCODED VALUE / TYPE 0 Indication 1 Confirmation 2 Request 3 Response
Len	17:15	**Transmit Message Length** — Length field of transmit message in 8-byte slots. Valid values= 1, 2, and 4. Writing a length of 2 and 4, clears the Avail bit in subsequent slots.
Seq	14:10	**Transaction Sequence Number** — Transaction sequence number of transmit message. Note: That the low-order three bits of this field specify the Receive Response Slot on which the message to be transmitted will be received.

FIELD NAME	BIT POSITION	DESCRIPTION
Dst	9:5	**Transaction Message Destination** — Message destination, (Processor ID) typically the TLU.
Src	4:0	**Transmit Message Source** — Source of transmit message, typically the processor's Ring Bus node ID. See Table 163 on page 511.

Table 163 Ring Bus Processor IDs

PROCESSOR	RING BUS NODE ID	PROCESSOR	RING BUS NODE ID
CP0	0	CP10	10
CP1	1	CP11	11
CP2	2	CP12	12
CP3	3	CP13	13
CP4	4	CP14	14
CP5	5	CP15	15
CP6	6	XP	24
CP7	7	FP*	30
CP8	8	TLU	31
CP9	9		

* can only send messages on the Ring Bus. There is a direct connection from the TLU to the FP for responses. If some other node tries to send to the FP, the messages will circulate forever on the Ring Bus.

Table 164 TxMsg*n*_Ctl Registers (for Messages 1, 2 and 3)

REGISTER NAME	PURPOSE	ADDRESS
TxMsg1_Ctl	Same as *TxMsg0_Ctl*.	0xBC*n*04448
TxMsg2_Ctl	Same as *TxMsg0_Ctl*, but not writable from RxByte Processor.	0xBC*n*04450
TxMsg3_Ctl	Same as *TxMsg0_Ctl*, but not writable from RxByte Processor.	0xBC*n*04458

TxMsg0_Data_H Register (CP Ring Bus Tx Message Control Function)

Purpose Bits [63:32] of the transmit message data slot (big endian bytes 0-3). See Table 165 on page 512 for similar registers.

Address 0xBC*n*04460

Access CPRC Read/Write, RxByte Processor Read/Write, and is byte addressable

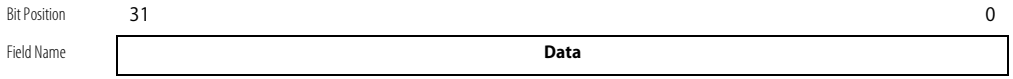


Table 165 TxMsg*n*_Data_H Registers (for Messages 1, 2 and 3)

REGISTER NAME	PURPOSE	ADDRESS
TxMsg1_Data_H	Same as *TxMsg0_Data_H*.	0xBC*n*04468
TxMsg2_Data_H	Same as *TxMsg0_Data_H*, but not writable form RxByte Processor.	0xBC*n*04470
TxMsg3_Data_H	Same as *TxMsg0_Data_H*, but not writable form RxByte Processor.	0xBC*n*04478

TxMsg0_Data_L Register (CP Ring Bus Tx Message Control Function)

Purpose Bits [31:0] of the transmit message data slot (big endian bytes 4-7). See Table 166 on page 512 for similar registers.

Address 0xBC*n*04464

Access CPRC Read/Write, RxByte Processor Read/Write, and is byte addressable

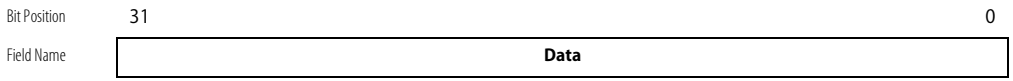


Table 166 TxMsg*n*_Data_L Registers (for Messages 1, 2 and 3)

REGISTER NAME	PURPOSE	ADDRESS
TxMsg1_Data_L	Same as TxMsg0_Data_l.	0xBCn0446C
TxMsg2_Data_L	Same as TxMsg0_Data_l, but not writable from RxByte Processor.	0xBCn04474
TxMsg3_Data_L	Same as TxMsg0_Data_l, but not writable from RxByte Processor.	0xBCn0447C

RxResp0_Ctl Register (CP Ring Bus Rx Response Control Function)

Purpose The control portion of an incoming Ring Bus response. See Table 167 on page 513 for similar registers.

Address 0xBC*n*04480

Access CPRC Read/Write

Bit Position	31	30 24	23	22 18	17 15	14 10	9 5	4 0
Field Name	Avail	Rsvd	Error	Rsvd	Len	Seq	Rsvd	Src
Reset Value	0	raz	x	raz		x	raz	x

FIELD NAME	BIT POSITION	DESCRIPTION
Avail	31	**Availability Bit** — When the bit is 1, slot is valid for the CPRC. When the bit is 0, allow Ring Bus to fill.
Reserved	30:24	Read as zero.
ErrorFlag	23	**Error Flag** — Error bit of response definable by user programming.
Reserved	22:18	Read as zero.
Len	17:15	**Receive Message Length** — Length field of receive message in 8Byte slots. Valid values= 1, 2, and 4. Writing a length of 2 and 4, clears the Avail bit in subsequent slots.
Seq	14:10	**Response Sequence Number** — Sequence number of the response, bits [12:10] match *RxRespCtl* register number.
Reserved	9:5	Read as zero.
Src	4:0	**Response Source** — Source field of the response, typically the processor's Ring Bus node ID. See Table 163 on page 511.

Table 167 RxResp*n*_Ctl Registers (for Ring Bus Responses 1, 2, 3, 4, 5, 6 and 7)

REGISTER NAME	PURPOSE	ADDRESS
RxResp1_Ctl	Same as *RxResp0_Ctl*.	0xBC*n*04484
RxResp2_Ctl	Same as *RxResp0_Ctl*.	0xBC*n*04488
RxResp3_Ctl	Same as *RxResp0_Ctl*.	0xBC*n*0448C
RxResp4_Ctl	Same as *RxResp0_Ctl*.	0xBC*n*04490
RxResp5_Ctl	Same as *RxResp0_Ctl*.	0xBC*n*04494

Table 167 RxResp*n*_Ctl Registers (for Ring Bus Responses 1, 2, 3, 4, 5, 6 and 7) (continued)

REGISTER NAME	PURPOSE	ADDRESS
RxResp6_Ctl	Same as *RxResp0_Ctl*.	0xBC*n*04498
RxResp7_Ctl	Same as *RxResp0_Ctl*.	0xBC*n*0449C

RxResp0_Data_H Register (CP Ring Bus Rx Response Control Function)

Purpose Bits [63:32] of the data portion of an incoming Ring Bus response (big endian bytes 0-3). See Table 168 on page 514 for similar registers.

Address 0xBC*n*044A0

Access CPRC Read/Write

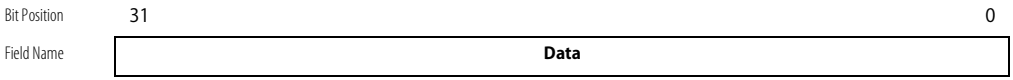


Table 168 RxResp*n*_Data_H Registers (for Ring Bus Responses 1, 2, 3, 4, 5, 6 and 7)

REGISTER NAME	PURPOSE	ADDRESS
RxResp1_Data_H	Same as *RxResp0_Data_H*.	0xBC*n*044A8
RxResp2_Data_H	Same as *RxResp0_Data_H*.	0xBC*n*044B0
RxResp3_Data_H	Same as *RxResp0_Data_H*.	0xBC*n*044B8
RxResp4_Data_H	Same as *RxResp0_Data_H*.	0xBC*n*044C0
RxResp5_Data_H	Same as *RxResp0_Data_H*.	0xBC*n*044C8
RxResp6_Data_H	Same as *RxResp0_Data_H*.	0xBC*n*044D0
RxResp7_Data_H	Same as *RxResp0_Data_H*.	0xBC*n*044D8

RxResp0_Data_L Register (CP Ring Bus Rx Response Control Function)

Purpose Bits [31:0] of the data portion of an incoming Ring Bus response (big endian bytes 4-7). See Table 169 on page 515 for similar registers.

Address 0xBC*n*044A4

Access CPRC Read/Write

Bit Position	31	0
Field Name	Data	

Table 169 RxResp*n*_Data_L Registers (for Ring Bus Responses 1, 2, 3, 4, 5, 6 and 7)

REGISTER NAME	PURPOSE	ADDRESS
RxResp1_Data_L	Same as *RxResp0_Data_L*.	0xBC*n*044AC
RxResp2_Data_L	Same as *RxResp0_Data_L*.	0xBC*n*044B4
RxResp3_Data_L	Same as *RxResp0_Data_L*.	0xBC*n*044BC
RxResp4_Data_L	Same as *RxResp0_Data_L*.	0xBC*n*044C4
RxResp5_Data_L	Same as *RxResp0_Data_L*.	0xBC*n*044CC
RxResp6_Data_L	Same as *RxResp0_Data_L*.	0xBC*n*044D4
RxResp7_Data_L	Same as *RxResp0_Data_L*.	0xBC*n*044DC

RxMsg_Ctl Register (CP Ring Bus Rx Message Control Function)

Purpose The top of the control portion of the Ring Bus receive message FIFO.

Address 0xBC*n*044E0

Access CPRC Read/Write

Bit Position	31 30	29 24	23	22 20	19 18	17 15	14 10	9 5	4 0
Field Name	State	Rsvd	Error	Rsvd	Type	Len	Seq	Rsvd	Src
Reset Value	0	raz	x	raz	x	x	x	raz	x

<table>
<tr><th>FIELD NAME</th><th>BIT POSITION</th><th>DESCRIPTION</th></tr>
<tr><td>State</td><td>31:30</td><td>Receive Message State: If State equals 1, there is at least one valid message is in the receive FIFO available to the CPRC process. State equals 1 for as long as any portion of a valid message remains in the FIFO.

<table>
<tr><th>ENCODED VALUE</th><th>STATE</th></tr>
<tr><td>00</td><td>Empty</td></tr>
<tr><td>10</td><td>High word</td></tr>
<tr><td>11</td><td>Low word</td></tr>
</table></td></tr>
<tr><td>Reserved</td><td>29:24</td><td>Read as zero.</td></tr>
<tr><td>ErrorFlag</td><td>23</td><td>Error Flag — Error bit of receive message definable by user programming.</td></tr>
<tr><td>Reserved</td><td>22:20</td><td>Read as zero.</td></tr>
<tr><td>Type</td><td>19:18</td><td>Receive Message Type:

<table>
<tr><th>ENCODED VALUE</th><th>TYPE</th></tr>
<tr><td>0</td><td>Indication</td></tr>
<tr><td>1</td><td>Confirmation</td></tr>
<tr><td>2</td><td>Request</td></tr>
</table></td></tr>
<tr><td>Len</td><td>17:15</td><td>Length — Length of receive message in 8Byte slots. Valid values are 1, 2, or 4.</td></tr>
<tr><td>Seq</td><td>14:10</td><td>Transaction Sequence Number — Transaction sequence number of receive message.</td></tr>
<tr><td>Reserved</td><td>9:5</td><td>Read as zero.</td></tr>
<tr><td>Src</td><td>4:0</td><td>Source — Source field of receive message, typically the processor’s Ring Bus node ID. See Table 163 on page 511.</td></tr>
</table>

RxMsg_FIFO Register (CP Ring Bus Rx Message Control Function)

Purpose The next four bytes of data from the Ring Bus receive message FIFO.

Address 0xBC*n*044E4

Access CPRC Read – Reading any portion of this register advances the receive FIFO. CPRC Write for test purposes only, writes the register but does not effect the receive FIFO.

Bit Position	31	0
Field Name	**Data**	

Rx_SONETOH0 to Rx_SONETOH31 Registers (CP SONET Rx Control Function)

Purpose Used for passing SONET Overhead fields extracted from the receive data stream by the framer to the CPRC.

Address 0xBC*n*04500 to 0xBC*n*0457C

Access CPRC Read, CPRC Write (during test), SDP Receive SONET Framer Write, and is byte addressable

Bit Position	31	0
Field Name	**Data**	

Tx_SONETOH0 to Tx_SONETOH31 Registers (CP SONET Tx Control Function)

Purpose Used for merging SONET Overhead fields from the CPRC into the transmit data stream.

Address 0xBC*n*04580 to 0xBC*n*045FC

Access CPRC Read, CPRC Write, SDP Transmit SONET Framer Read, and is byte addressable

Bit Position	31	0
Field Name	**Data**	

RxCtl_ByteSeq0 Register (CP SDP Rx Control Function)

Purpose Provides an area for passing information between the CPRC and RxByte Processor. See Table 170 on page 518 for similar register.

Address 0xBC*n*04600

Access CPRC Read/Write, SDP Receive Byte Sequencer Read/Write, and is byte addressable

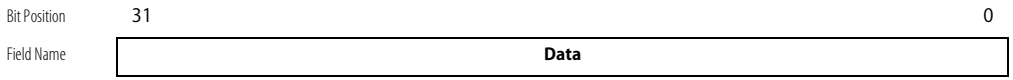


Table 170 RxCtl_ByteSeq1 Register (for Byte Sequence1)

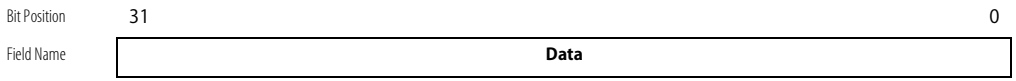
REGISTER NAME	PURPOSE	ADDRESS
RxCtl_ByteSeq1	Same as RxCtl_ByteSeq0.	0xBCn04604

RxCtl_SyncSeq Register (CP SDP Rx Control Function)

Purpose Provides an area for passing information between the CPRC and RxSync Processor.

Address 0xBC*n*04608

Access CPRC Read/Write, RxSync Sequencer Read/Write, and is byte addressable



RxCtl_BitSeq0 Register (CP SDP Rx Control Function)

Purpose Provides an area for passing information between the CPRC and RxBit Processor. See Table 171 on page 519 for similar register.

Address 0xBC*n*0460C

Access CPRC Read/Write, RxBit Sequencer Read/Write, and is byte addressable

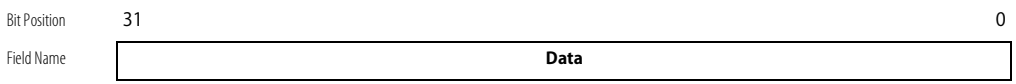


Table 171 RxCtl_BitSeq1 Register (for Bit Sequence1)

REGISTER NAME	PURPOSE	ADDRESS
RxCtl_BitSeq1	Same as *RxCtl_BitSeq0*.	0xBC*n*04610

TxCtl_ByteSeq0 Register (CP SDP Tx Control Function)

Purpose — Provides an area for passing information between the CPRC and TxByte Processor. See Table 172 on page 519 for similar register.

Address — 0xBC*n*04620

Access — CPRC Read/Write, TxByte Processor Read/Write, and is byte addressable

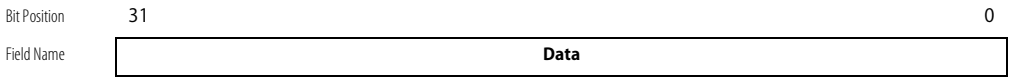


Table 172 TxCtl_ByteSeq1 Register (for Byte Sequence1)

REGISTER NAME	PURPOSE	ADDRESS
TxCtl_ByteSeq1	Same as *TxCtl_ByteSeq0*.	0xBC*n*04624

TxCtl_BitSeq0 Register (CP SDP Tx Control Function)

Purpose — Provides an area for passing information between the CPRC and TxBit Processor. See Table 173 on page 519 for similar register.

Address — 0xBC*n*0462C

Access — CPRC Read/Write, TxBit Processor Read/Write, and is byte addressable

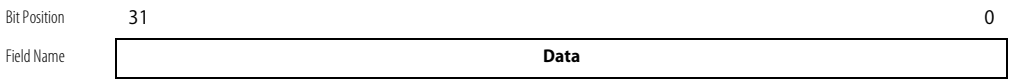


Table 173 TxCtl_BitSeq1 Register (for Bit Sequence1)

REGISTER NAME	PURPOSE	ADDRESS
TxCtl_BitSeq1	Same as *TxCtl_BitSeq0*.	0xBC*n*04630

CP_Mode0 Register (CP Mode Configuration Function)

Purpose Collects mode and control bits relevant to general CPRC and CP configuration.

Address 0xBC*n*04640

Bit Position	31	30	29	28	27 26	25	24	23 22	21 20	19	18 17	16	15 8	7	6	5	4	3	2	1	0
Field Name	RC Resetx	Wind Down	CP to XP IRQ	CP to all IRQ	Rsvd	SendSpecul Commit	Valid/ Invalid	QMU rdmbx	QMU wrmbx	Rsvd	Imode	Retry Global	WCS Write Byte	Scan Data Out	RxWCS Write	TxWCS Write	Rsvd	Scan Capture	Scan Update	Scan Datain1	Scan Datain0
Reset Value	0	raz	raz	raz	raz	0	0	0	0	raz	0	0	raz	x	raz	raz	raz	raz	raz	raz	raz

FIELD NAME	BIT POSITION	DESCRIPTION
RC_Resetx	31	**CPRC Resetx** — The active low *CPRC Resetx* bit powers up asserted, that is, = 0, so that the CPRC is in the reset state. It must be set by the XP or another CPRC to release the reset and enable the CPRC to begin the boot sequence. A CP can not enable itself.
WindDown	30	**Wind Down** — When the bit is written to 1, it asserts a global signal informing all chip functions to wind down as soon as possible, and to leave as much predictable error recovery state around as possible.
CPtoXPIRQ	29	**CP to XP IRQ** — When the bit is written to 1, it asserts a global signal causing an XP interrupt from this CP.
CPtoAllIRQ	28	**CP to All IRQ** — When the bit is written to 1, it asserts a global signal that causes an interrupt to every CP.
Reserved	27:26	Read as zero.
SendSpeculCommit	25	**Send Speculative Commit** — 1= Initialize transfer 0= No action.
Valid/Invalid	24	**Valid/Invalid** — 0= Commit, valid descriptor 1= Commit, invalid descriptor

<table>
<tr><th>FIELD NAME</th><th>BIT POSITION</th><th>DESCRIPTION</th></tr>
<tr><td>QMU rdmbx</td><td>23:22</td><td>QMU rdmbx Status — Legal ranges are detailed here:

<table>
<tr><th>ENCODED VALUE</th><th>STATUS</th></tr>
<tr><td>00</td><td>QMU idle or operation finished successfully</td></tr>
<tr><td>01</td><td>operation finished with error (probably resource error, see above)</td></tr>
<tr><td>10</td><td>busy, waiting to begin execution</td></tr>
<tr><td>11</td><td>busy, executing in QMU engine</td></tr>
</table>
</td></tr>
<tr><td>QMU wrmbx</td><td>21:20</td><td>QMU wrmbx Status — Legal ranges are detailed here:

<table>
<tr><th>ENCODED VALUE</th><th>STATUS</th></tr>
<tr><td>00</td><td>QMU idle or operation finished successfully</td></tr>
<tr><td>01</td><td>operation finished with error (probably resource error, see above)</td></tr>
<tr><td>10</td><td>busy, waiting to begin execution</td></tr>
<tr><td>11</td><td>busy, executing in QMU engine</td></tr>
</table>
</td></tr>
<tr><td>Reserved</td><td>19</td><td>Read as zero.</td></tr>
<tr><td>Imode</td><td>18:17</td><td>IMEM Configuration Mode — Legal ranges are detailed here:

<table>
<tr><th>ENCODED VALUE</th><th>MODE</th></tr>
<tr><td>00</td><td>Unshared memory</td></tr>
<tr><td>11</td><td>4-way shared memory</td></tr>
<tr><td>01</td><td>Unsupported</td></tr>
<tr><td>10</td><td>Unsupported</td></tr>
</table>

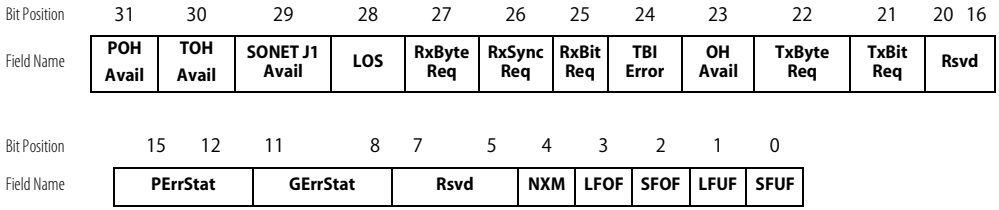
When using cluster memory, only CP0,4,8 & 12 use these 2 bits, therefore, CP1-3, 5-7, 9-11 & 13-15 do not use these 2 bits.</td></tr>
</table>

FIELD NAME	BIT POSITION	DESCRIPTION
RetryGlobal	16	**Global Bus Transaction Retry** — This bit causes global load and store operations through the Global bus controller to be retried up to 256 times when NACK'd. When 256 tries have been NACK'd, the bus controller terminates the operation and asserts a bus error.
WCS Write Byte	15:8	**Writable Control Store Write Byte** — Writing this byte coincident with writing bit 6 and/or bit 5 causes the byte data to be written to SDP control store.
ScanDataOut	7	**Scan Chain Data Out** — the value of this last bit in the SDP scan chain.
RxWCSWrite	6	**RxWritable Control Store Write** — When the bit is 1, it causes the data in the WCS Write Byte field to be loaded into RxSDP control store at a value pointed to by the internal SDP WCS load address register, and the address to increment.
TxWCSWrite	5	**TxWritable Control Store Write** — When the bit is 1, it causes the data in the WCS Write Byte field to be loaded into Transmit SDP control store at a value pointed to by the internal SDP WCS load address register, and the address to increment.
Reserved	4	Read as zero.
ScanCapture	3	**Scan Capture** — Parallel load of the SDP scan chain with chip state.
ScanUpdate	2	**Scan Update** — Parallel drive SDP scan chain data into chip state.
ScanDataIn1	1	**Scan Data In 1** — Shift a 1 serially into the SDP scan chain.
ScanDataIn0	0	**Scan Data In 0** — Shift a 0 serially into the SDP scan chain.

CP_Mode1 Register (CP Mode Configuration Function)

Purpose Collects mode and status bits relevant to general CP configuration.

Address 0xBC*n*04644



FIELD NAME	BIT POSITION	DESCRIPTION
POH Avail	31	**SONET Payload Overhead Address Avail** — When set, indicates that the final byte of path overhead (Z5) has just been written to the receive SONET overhead register area. All desired path overhead must be read out before the next time this signal is asserted or it will be overwritten
TOH Avail	30	**SONET Transport Overhead Address Avail** — When set, indicates that the final byte of transport overhead (E2) has just been written to the receive SONET overhead register area. All desired transport overhead must be read out before the next time this signal is set to a one or it will be overwritten. This bit is also available in SONET event register.
SONET J1 Avail	29	**SONET J1 Avail** — Indicates that the new j1 index written by the CPRC now has the corresponding J1 in the J1 overhead register area. This is cleared by writing a new j1 index. This is set by writing a J1 byte to the overhead register. This bit is also available in SONET event register.
LOS	28	**Loss of Signal** — This signal is either loss of synchronization or loss of frame as a function of other mode bits.

RXSONET ENABLE	LOS MODE	FUNCTION
0	0	Gigabit ethernet loss of sync
0	1	Fibre channel loss of sync
1	x	All zeros received for > 2.3 µseconds

FIELD NAME	BIT POSITION	DESCRIPTION
RxByteReq	27	**SDP RxByte Service Request** — When set, RxByte processor microcode needs attention of the CPRC. This is also connected to the CP event register. This bit is written by the RxByte processor.
RxSyncReq	26	**SDP RxSync Service Request** — When set, RxSync processor microcode needs attention of the CPRC. This is also connected to the CP event register. This bit is written by the RxSync processor.
RxBitReq	25	**SDP RxBit Service Request** — When set, RxBit processor microcode needs the attention of the CPRC. This is also connected to the CP event register. This bit is written by the RxBit processor.
TBI Error	24	**Ten Bit Interface Symbol Error** — When set to a one, indicates that there has been a ten bit symbol error since this bit was last cleared. This bit is set by the ten bit decoder and cleared by the CPRC by writing a one to this bit. It is readable by the CPRC. This cannot be used for error counting, but it does give some indication as to the health of the link.
OH Avail	23	**SONET Overhead Avail** — show the current value of the SDP Transmit SONET Framer's address bit 6 into the SONET overhead registers.
TxByteReq	22	**SDP TxByte Service Request** — When set, TxByte processor microcode needs attention of the CPRC. This is also connected to the CP event register. This bit is written by the TxByte processor.
TxBitReq	21	**SDP TxBit Service Request** — When set, TxBit processor microcode needs attention of the CPRC. This is also connected to the CP event register. This bit is written by the TxBit processor.
Reserved	20:16	Read as zero.
PErrStat	15:12	**Payload Error Status** — loaded when a Payload Error occurs and is locked until the CPRC Process clears the PErr bit. The individual control blocks can be interrogated to determine the specific offender. Write 1 to clear. Refer to Table 147 on page 495 for error code definitions.
GErrStat	11:8	**Global Error Status** — loaded when a Global Error occurs and locked until the RC process clears the GErr bit. Codes are shown in Table 174. Write 1 to clear.
Reserved	7:5	Read as zero.

FIELD NAME	BIT POSITION	DESCRIPTION
NXM	4	**Cluster Non-Existent Memory** — CPRC reference to cluster space addressed non-existent Data Memory (DMEM) or non-local configuration register space. Write 1 to clear.
LFOF	3	**Receive Large FIFO Overflow** — Receive Large FIFO overflow condition. Write 1 to clear.
SFOF	2	**Receive Small FIFO Overflow** —Receive Small FIFO overflow condition. Write 1 to clear.
LFUF	1	**Transmit Large FIFO Underflow** — Transmit Large FIFO underflow condition. Write 1 to clear.
SFUF	0	**Transmit Small FIFO Underflow** — Transmit Small FIFO underflow condition. Write 1 to clear.

Table 174 Global Bus Error Status Encoding

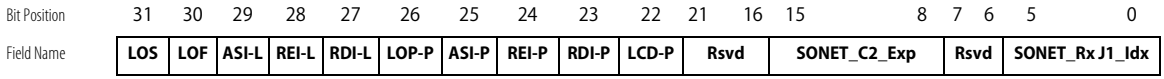
ERROR TYPE	ENCODING	READ/WRITE	DESCRIPTION
Success	0	R, W	The transaction completed successfully. If an error is reported, it is due to a non-cluster local DMEM reference.
NACK Retry Limit	9	R, W	The target was unable to accept a global memory request. The global bus transaction was attempted until the NACK retry limit was reached.
Non-Existent Memory	A	R, W	Reference made to an un-subscribed 1MByte CP block.

SDP_Mode2 Register (CP Mode Configuration Function)

Purpose Collects SONET/SDH alarm and status information.

Address 0xBC*n*04648

Access CPRC Read/Write



Bit Position	31	30	29	28	27	26	25	24	23	22	21	16	15	8	7	6	5	0
Field Name	LOS	LOF	ASI-L	REI-L	RDI-L	LOP-P	ASI-P	REI-P	RDI-P	LCD-P	Rsvd		SONET_C2_Exp		Rsvd		SONET_Rx J1_Idx	

FIELD NAME	BIT POSITION	DESCRIPTION
LOS	31	**Loss Of Signal** — This bit is meaningful for SONET/SDH, Gigabit Ethernet, and Fibre Channel applications. For SONET/SDH, it is defined as: the occurence of all-zeros (no line transitions) are detected on the receive SONET/SDH line: 2.3µs for OC-3c, or 4.6µs for OC12/c. LOS clears when the FRAMELOSS bit in the control/status register (ireg14) in RxBit programmable processor is cleared. This meets the Telcordia GR-253 standard. RxBit must set or clear FRAMELOSS based upon receiving the correct A1/A2 framing pattern. Note: The LOS indication requires the transmit SONET/SDH block be enabled and the SONET/SDH Transmit clock set to the correct frequency because LOS detection is timed through the TxSONET transmit logic. Note: That detection of LOS is dependent upon the transceiver's transmitting zeros when no optical power is being received. Check the optical transceiver documentation to implement the desired behavior. This bit is read-only.
LOF	30	**SONET Loss Of Framing** — This bit is set when a severely erred frame (SEF) condition has been detected for 3ms on the receive SONET/SDH stream. This bit is cleared when framing has been achieved for 3ms (A1, A2 bytes). This bit is controlled through the RxBit processor within the SDP. RxBit microcode must set or clear the FRAMELOSS bit in the control/status register (ireg14 bit 6) indicating whether it has achived or lost SONET/SDH frame synchronization (SEF). The SONET/SDH block then times out this condition for 3ms (24 frames) before setting the LOF bit. 24 valid frames are required to clear LOF. This bit is read-only. Note: The RxSONET block does *not* forward data when LOF is set.

FIELD NAME	BIT POSITION	DESCRIPTION
AIS-L	29	**SONET/SDH Line Alarm Indication Signal** — Set when five consecutive frames have been detected which contain the value (111) in bits 6,7, and 8 of the K2 byte. Cleared when five consecutive frames have been detection which do *not* contain the value (111) in bits 6,7, and 8 of the K2 byte. This bit is read-only.
REI-L	28	**SONET/SDH Line Remote Error Indicator** — Set when the M1 byte is between 1 and 24 for OC3c, or between 1 and 96 for OC12c for the current received SONET/SDH frame. M1 indicates the number of B2 errors detected on our transmitted SONET/SDH stream by the far-end equipment. This bit is cleared if zero errors are detected in the M1 byte or if the number of errors is outside the valid range. This bit is read-only.
RDI-L	27	**SONET/SDH Line Remote Defect Indicator** — Set when bits 6,7, and 8 of the K2 byte contain the value '110' for five consecutive frames. Cleared when bits 6,7, and 8 contain any pattern other than the code '110' in five consecutive frames. This bit is read-only.
LOP-P	26	**SONET/SDH Path Loss Of Pointer** — LOP-P is set when ten frames of invalid pointers or NDF enabled indications are observed. LOP-P is cleared when the same valid pointer with normal NDF is detected for 3 consecutive frames. This bit is read-only. Note: The RxSONET block does *not* forward data when LOP-P is set.
AIS-P	25	**SONET/SDH Path Alarm Indication Signal** — AIS-P is set when the H1 and H2 bytes for the first STS-1 contains an all-ones pattern in 3 consecutive frames. AIS-P is cleared after 3 good pointers have been received. This bit is read-only.
REI-P	24	**SONET/SDH Path Remote Error Indicator** — When G1[1:4] contains a value between 1 and 8 for the last frame, this bit is set. If G1 is a value other than those values, the bit is cleared.

<table>
<tr><th>FIELD NAME</th><th>BIT POSITION</th><th>DESCRIPTION</th></tr>
<tr><td>RDI-P</td><td>23</td><td>SONET/SDH Path Extended Remote Defect Indication — The following table illustrates what this bit is set to depending upon the value of G1[5:7]. 10 consecutive frames of the same ERDI-P code must be received before the value in the G1 RxSONET_OH overhead register is changed.

<table>
<tr><th>BITS [5:7] VALUES IN ERDI-P STATE</th><th>RDI_P BIT [23] SETTING</th></tr>
<tr><td>110</td><td>1</td></tr>
<tr><td>101</td><td>1</td></tr>
<tr><td>100</td><td>1 for backwards compatibility with RDI-P.</td></tr>
<tr><td>010</td><td>1</td></tr>
<tr><td>001</td><td>0</td></tr>
<tr><td>000</td><td>0 for backwards compatibility with RDI-P.</td></tr>
</table>
</td></tr>
<tr><td>LCD-P</td><td>22</td><td>Loss of Cell/Packet Delineation - Path — This bit is set when a Loss of Cell Delineation condition has been detected by the RxSync SDP microengine. This bit is controlled through the RxSync processor within the SDP. RxSync microcode must set or clear the DELINLOSS bit in the control/status register (ireg14 bit 6) indicating whether it has achieved or lost ATM cell synchronization (LCD) based upon verification of the ATM HEC. This bit is read-only. This bit is controlled through the RxSync processor within the SDP. When LCD_P is ON, and ManualFEBE is cleared, the appropriate ERDI-P code is transmitted. See the ManualFEBE bit in SDP_Mode3 for more information.</td></tr>
<tr><td>Reserved</td><td>21:16</td><td>Read as zero.</td></tr>
<tr><td>Sonet_C2_Exp</td><td>15:8</td><td>SONET C2 Expected — This is the value of the C2 Path Signal Label that is expected to be received from the far end SONET/SDH link. PLM-P and Path Unequipped alarms can be determined through the use of this register and the C2_ERROR bit in the SONET_Event register. See the SONET_Event register bit 8 C2_ERROR for more information.</td></tr>
<tr><td>Reserved</td><td>7:6</td><td>Read as zero.</td></tr>
</table>

FIELD NAME	BIT POSITION	DESCRIPTION
Sonet_J1_Idx	5:0	**RxSONET J1 Index** — Indicate which of the 64Byte path trace (J1) message should be written to the receive overhead location for J1. Changing this field clears *J1_AVAIL* in the *SONET_EVENT* register. *J1_AVAIL*becomes set when the J1 is actually written to the overhead register.

SDP_Mode3 Register (CP Mode Configuration Function)

Purpose Collects configuration mode bits relevant for programming the RxSDP machines.

Address 0xBC*n*0464C

Access CPRC Read/Write

Bit Position	31	30	29	28	27	26	25	24	23 22	21
Field Name	RxResetx	RxEnable	RxByteEna	RxBitEna	RxSonetEna	RxSyncEna	RxByte Loopback	RxBit Loop back	RxAgg Mode	RxSync CRC16/32
Reset Value	0	0	0	0	0	0	x	x	x	x

Bit Position	20	19	18	17	16	15	14	13	12 11	10
Field Name	RxSyncCRC Init	RxSyncFOL	RxSonet Concat	RxSync CRCInv	LOSenable	LOSmode	Rsvd	Manual_FEBE	RxBitInWidth	SonetOC
Reset Value	x	x	x	x	x	x	x	x	x	x

Bit Position	9	8	7	6	5	4	3	2	1	0
Field Name	SonetDscr	RxByteCRCInit	RxByteCRC 16/32	RxByteCRC Inv	RxByteFirst OnLeft	RxBitFirst OnLeft	Payload Dscr	RxFifo ExDis	Rx10Bit	Bit2Bit Source
Reset Value	x	x	x	x	x	x	x	x	x	x

FIELD NAME	BIT POSITION	DESCRIPTION
RxResetx	31	**RxSDP Master Reset** — When 0, puts RxSDP in reset state. When 1, RxSDP is in run state. This must be low (asserted) in order to load the microcode. NOTE: For CP0, CP4, CP8, and CP12 in each cluster, the RxSDP Master Reset bit [31] must be 1 (reset disasserted) for any cluster that is to be used for RxSDP processing. Without this, data is *not* passed from the receive pins to the individual CPs. For example, if CP10 is to be used for RxSDP processing, then the RxSDP Master Reset bit [31] in the CP8 SDP_Mode3 register must be set to 1, in addition to setting the SDP_Mode register for CP10.
RxEnable	30	**RxSDP Master Enable** — When 1, enables all Rx Sequencers. Allows all Rx processors and the SONET/SDH logic to be enabled at once.
RxByteEnable	29	**RxByte Processor Enable** — Enables the RxByte processor, when set to 1. Freezes the micropc at the current microaddress when disabled. Receive master enable must be set in order for this bit to have an effect. The RxByte processor should be enabled for all applications.
RxBitEnable	28	**RxBit Processor Enable** — Enables the RxBit processor, when set to 1. Freezes the processor at the current microaddress when disabled. Receive master enable must be set in order for this bit to have an effect. This processor should be enabled for all applications.
RxSonetEnable	27	**RxSONET Enable** — When a one, the SONET/SDH pointer interpreter, payload demultiplexer and overhead termination logic is enabled. When a zero, this SONET/SDH logic is disabled and bypassed. Effects operation of the loss of sync signal bit.
RxSyncEnable	26	**RxSync Enable** — Enables the rxsync processor, when set to a one. Freezes the micropc at the current microaddress when disabled. Receive master enable must be set in order for this bit to have an effect.
RxSDP Configuration	25:0	Each of the individual bits [25:0] are described in detail below:
RxByteLoopback	25	**RxByte Loopback Enable** — Connects network side of transmit large FIFO to network side of receive large FIFO, when set to a one.

FIELD NAME	BIT POSITION	DESCRIPTION
RxBitLoopback	24	**RxBit Loopback Enable** — Connects network side of transmit small FIFO to network side of receive small FIFO, when set to a one. Only works for aggregation mode = 0 (groups of one). The pin logic is not included in this loopback.
RxAggMode	23:22	**SDP Receive Aggregation Mode** — Controls the number of receive SDPs in a cluster that work as a unit. 00 = Each SDP works independently 01 = Reserved 10 = SDPs work as one group of four 11 = Reserved
RxSyncCRC16/32	21	**RxSync CRC 16/32** — When a one, selects CRC-16 operation. When a zero, selects CRC-32 operation.
RxSyncCRCInit	20	**RxSync CRC Initialize to Ones** — When a one, resetting the CRC register sets the CRC register to all ones. When a zero, resetting the CRC register sets the CRC register to all zeroes.
RxSyncFOL	19	**RxSync CRC First on Left** — Calculate CRC on the data assuming that the first bit received is the left-most bit of the byte. Must be set the same as *RxBitFirstOnLeft*.
RxSonetConcat	18	**RxSONET Concatenation Mode** — When in OC-12 mode and set to a one, the receive logic is configured to SONET OC-12c / SDH STM-4 VC-4-4c (one pipe). When in OC-12 mode and set to a zero, the receive logic is configured to SONET OC-12 (four OC-3c streams) / SDH STM-4 (four VC-4-1c streams). Note: When RXSONET OC-12/OC-3 is set to a zero (OC-3 mode), this bit must be set.
RxSyncCRCInv	17	**RxSync CRC Output Invert** — Read the ones complement of the CRC register, when a one. When a zero, read the CRC register directly.
LOSenable	16	**Loss of Synchronization Enable** — When set, the 8 bit 10 bit (TBI) decoder runs the loss of synchronization state machine on the incoming ten bit data. The loss of sync signal bit is a one if this enable is set to zero.

<table>
<tr><th>FIELD NAME</th><th>BIT POSITION</th><th>DESCRIPTION</th></tr>
<tr><td>LOSmode</td><td>15</td><td>Loss of Synchronization Mode — This signal is either loss of synchronization or loss of frame as a function of other mode bits
<table>
<tr><th>RXSONET ENABLE</th><th>LOS MODE</th><th>FUNCTION</th></tr>
<tr><td>0</td><td>0</td><td>Gigabit ethernet loss of sync</td></tr>
<tr><td>0</td><td>1</td><td>Fibre channel loss of sync</td></tr>
<tr><td>1</td><td>x</td><td>All zeros received for > 2.3 µseconds</td></tr>
</table></td></tr>
<tr><td>Reserved</td><td>14</td><td>Read as zero.</td></tr>
</table>

<table>
<tr><th>FIELD NAME</th><th>BIT POSITION</th><th>DESCRIPTION</th></tr>
<tr><td>Manual_FEBE</td><td>13</td><td>
Manual FEBE — When a one, the transmitted values of K2, M1 and G1 are completely determined by the contents of the TxSONET overhead registers. When a zero, all of M1, G1 and the Remote Error Indication portion of K2 are automatically generated by the hardware as a function of the receive data.
Note: The SONET/SDH byte terminology is as follows: M1[1:8] indicates bits 1 through 8 of the M1 byte where bit 1 is the msb and bit 8 is the lsb of the byte. Bytes are transmitted msb to lsb. This terminology is taken from Telcordia GR-253 core.
When Manual_FEBE is cleared the M1, G1 and K2 are affected as follows:
• M1[1:8] contains the number of B2 errors detected on the received SONET/SDH signal (0-24) OC3c or (0-96) OC12/c.
• G1[1:4] contains the number of B3 errors detected on the received SONET/SDH signal (0-8).
• G1[5:7] contains the Extended RDI-P value depending upon the state of the received SONET/SDH signal as detailed here:
<table>
<tr><th>RECEIVED SONET SIGNAL STATE ERDI-P TX BITS [5:7]</th><th>VALUE</th></tr>
<tr><td>AIS-P, LOP-P</td><td>101</td></tr>
<tr><td>UNEQ-P, TIM-P</td><td>110</td></tr>
<tr><td>PLM-P, LCD-P</td><td>010</td></tr>
<tr><td>No Defects</td><td>001</td></tr>
</table>
• K2[6:8] is written to the value 110 indicating RDI-L when LOS, LOF or AIS-L defects are detected on the received SONET/SDH signal.
Note: K2 reverts back to the value read from the TxSONET overhead register when an RDI-L condition is not being transmitted.
</td></tr>
</table>

<table>
<tr><th>FIELD NAME</th><th>BIT POSITION</th><th>DESCRIPTION</th></tr>
<tr><td>Manual_FEBE
(Continued)</td><td>13</td><td>K2 Operational Details:
If ForceLineAIS in the SDP_Mode5 Register is set, then the K2 byte reflects the AIS code (all 1's). Otherwise, if Manual_FEBE in the SDP_Mode3 is set, then K2 reflects the value in the Transmit SONET/SDH Transport Overhead K2 Byte Register. Otherwise, if RDI_L is asserted, then the 3 (lsb) are forced to value 6 and the upper 5 (msb) come from the Transmit SONET/SDH Transport Overhead K2 Byte Register. Finally, if none of the above conditions are present, then the default K2 reads its value from the Transmit SONET/SDH Transport Overhead K2 Byte Register.
Note: The signal used to select the RDI_L state above is actually filtered, so that if RDI_L is detected on the Receive (RX) side (sometime after the K1 byte), then it asserts RDI_L to this Transmit (TX) logic and holds it for at least 20 frames.</td></tr>
<tr><td>RxBitInWidth</td><td>12:11</td><td>RxBit Input Width — Determines the input data width received by RxBit. If the ten bit decoder is enabled, this field is ignored.

<table><tr><th>RXBIT INPUT WIDTH</th><th>INPUT WIDTH IN BITS</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>2</td></tr><tr><td>2</td><td>4</td></tr><tr><td>3</td><td>8</td></tr></table></td></tr>
<tr><td>SonetOC</td><td>10</td><td>RxSONET OC12/OC3 Select — When a one, the SONET/SDH receive logic is configured to receive SONET OC-12 / SDH STM-4. When a zero, this receive logic is configured to receive SONET OC-3c / SDH STM-1 VC-4-1c.</td></tr>
<tr><td>SonetDscr</td><td>9</td><td>SONET Descramble Enable — Enables the SONET / SDH descramble operation, when set to a one. When a zero, no descrambling occurs. Normally enabled if SONET/SDH is enabled.</td></tr>
<tr><td>RxByteCRCInit</td><td>8</td><td>RxByte CRC Initialize to Ones — When a one, resetting the CRC register sets the CRC register to all ones. When a zero, resetting the CRC register sets the CRC register to all zeroes.</td></tr>
</table>

FIELD NAME	BIT POSITION	DESCRIPTION
RxByteCRC16/32	7	**RxByte CRC 16/32** — When a one, selects CRC-16 operation. When a zero, selects CRC-32 operation.
RxByteCRCInv	6	**RxByte CRC Output Invert** — Read the ones complement of the CRC register, when a one. When a zero, read the CRC register directly.
RxByteFirstOnLeft	5	**RxByte CRC First on Left** — Calculate CRC on the data assuming that the first bit received is the left-most bit of the byte. Must be set the same as *RxBitFirstOnLeft*.
RxBitFirstOnLeft	4	**RxBit First on Left** — When a zero, the first bit on received is on the right-most bit of the byte. When a one, the first bit received is on the left-most bit of the byte. Set to zero for Ethernet. Set to one for SONET/SDH.
PayloadDscr	3	**RxSync Payload Descramble Enable** — When a one, self-synchronous payload descrambling is applied to the receive data if the microcode also enables it. When a zero, data is not descrambled.
RxFifoExDis	2	**RxFIFO Exception Disable** — When set to a zero, allows the RxSDP overflow handler to empty the RxSDP pipeline in the event of FIFO overflow. When set to a one, inhibit the SDP overflow handler from emptying the RxSDP in the event of FIFO overflow. Note: For diagnostics. This bit is normally set to zero.
Rx10Bit	1	**Fix 10 Bit Decoder Mode** — When set, the 8 bit to 10 bit decoder is enabled. The Rxbit input width is set to 10 bits. When clear, the 8 bit to 10 bit decoder is disabled and Rxbit input width is determined by the Rxbit input width field as described above. Note: This is used for gigabit ethernet 1000BaseX and Fibre Channel.
Bit2BitSource	0	**Receive Bit-tobit Source** — When set to one 1, selects the bit-to-bit signal from RxBit processor 0 of its neighbor cluster. If we call the four clusters 0, 1, 2, and 3. 0 and 1 are neighbors. Clusters 2 and 3 are also neighbors. When set to 0, selects the bit-to-bit signal from an RxBit processor in the local cluster. Which RxBit processor of the four is selected is a function of the aggregation mode. The bit-to-bit signal is readable by the RxBit processor.

SDP_Mode4 Register (CP Mode Configuration Function)

Purpose Configures the TxLarge FIFO and Aggregation Multiplexer.

Address 0xBC*n*04650

Access CPRC Read/Write

Bit Position	31	30 26	25 24	23 11	10	9	8	7	6 0
Field Name	FrameMode	Reserved	AgMux State	FrameCnt	Idle Cell	Idle Insert	Auto Token	Rsvd	Large FIFO Watermark
Reset Value	0	raz	0	0	x	x	x	raz	x

FIELD NAME	BIT POSITION	DESCRIPTION
FrameMode	31	**SONET APS Frame Mode** — When set, this bit turns FrameMode on. In FrameMode the FrameCnt field is used to determine the number of frames to wait before causing an interrupt when a B1, B2, B3, REI-L or REI-P error is observed in the incoming SONET/SDH stream. A rising edge on this bit causes the FrameCnt field to be re-read by the hardware. Additionally, the internal accumulated counters for B1, B2, B3, REI-L and REI-P are reset to zero. When set, alternative secondary definitions for the B1_Error, B2_Error and B3_Error SONET_Event register bits apply. See the SONET_Event register for these definitions.
Reserved	30:26	Read as zero.
AgMuxState	25:24	**Aggregation Multiplexer State** — Allows the base CPRC of an aggregate group to determine which CPRC of the aggregate group is currently selected by the aggregation multiplexer.
FrameCnt	23:11	**SONET APS Frame Count** — When FrameMode is enabled, this field indicates the number of frames the hardware delays before causing a B1_Error, B2_Error or B3_Error event to occur in the SONET_Event register if any of these errors have occurred in the last FrameCnt frames. This value indicates the number of complete SONET/SDH frames following the write to this register which causes an interrupt. Zero implies interrupt on the very next frame boundary. Note: The frameCnt value is updated after the last column of the Z5 POH byte is updated.

FIELD NAME	BIT POSITION	DESCRIPTION
Idle Cell	10	**Idle Cell Mode** — If the *Idle Insertion Enable* bit is set to 1, *Idle Cell Mode* determines what is inserted in to the data stream when no other data is available. 0 = insert a packet over SONET/SDH flag character 1 = insert an ATM idle cell
Idle Insert	9	**Idle Insertion Enable** — If set to 1, the transmit pipeline inserts either a packet over SONET/SDH flag (0x7E) or an ATM idle cell (if there is no other data available) depending on the state of the idle cell mode bit. This insertion occurs either before or after the aggregation multiplexer, depending on the state of the *Scramble/Insertion mode* bit in *SDP_MODE5* register. If set to a 0, no insertion takes place.
Auto Token	8	**AutoToken Enable** — When set to 1 and the aggregation group size is four, the token is passed without micro sequencer intervention whenever *data_nine* is read from the aggregation multiplexer output stage.
Reserved	7	Read as zero.
Large FIFO Watermark	6:0	**Large FIFO Watermark** — The TxLarge FIFO (128 words) has a watermark associated with it. Starting from the empty state, the FIFO remains empty until the entry count becomes greater than the watermark at which point it then becomes not-empty. The watermark depth is recalculated each time the last byte of a packet or cell has been transmitted. The last byte is defined as any byte with the 9th bit set. If the FIFO depth is less than the watermark number of bytes when the last byte is transmitted from the FIFO, the FIFO appears empty.

SDP_Mode5 Register (CP Mode Configuration Function)

Purpose Collects configuration mode bits relevant for programming the TxSDP machines.

Address 0xBC*n*04654

Access CPRC Read/Write

Bit Position	31	30	29	28	27	26	25	24	23 22
Field Name	TxResetx	TxEnable	TxByteEna	TxBitEna	TxSonetEna	ForceLineAIS	TxSonetOh Comp	ForceSonet PErr	TxAggMode
Reset Value	0	0	0	0	0	x	x	x	x

Bit Position	21	20 19	18	17	16	15	14	13	12 11	10
Field Name	TxFifoExDisab	TxBitPHY	TxSonet ConcMode	ForcePath AIS3	ForcePath AIS2	ForcePath AIS1	ForcePath AIS0	Sonet j1Mis	TxBitOut Width	TxSonet OC
Reset Value	x	x	x	x	x	x	x	x	x	x

Bit Position	9	8	7	6	5	4	3	2	1	0
Field Name	TxSonetScr	TxByteByte CRCInit1	TxByteCRC 16	TxByte CRCInv	TxByteFirst OnLeft	TxBitFirst OnLeft	Payload ScrEna	Payload ScrMode	Tx10bit	ForceLOS
Reset Value	X	x	x	x	x	x	x	x	x	x

FIELD NAME	BIT POSITION	DESCRIPTION
TxResetx	31	**TxSDP Master Reset** — When 0, puts TxSDP in reset state. When 1, puts TxSDP in run state. Must be low (asserted) in order to load the microcode.
TxEnable	30	**TxSDP Master Enable** — When 1, enables all Rx Sequencers. This allows all of the processors and the SONET/SDH logic to be started at the same time.
TxByteEna	29	**TxByte Enable** — Enables the TxByte processor, when set to 1. Freezes the micropc at the current microaddress when disabled. *TxEnable* must be set in order for this bit to have an effect. This processor should be enabled for all applications.
TxBitEna	28	**TxBit Enable** — Enables the TxBit processor, when set to a one. Freezes the micropc at the current microaddress when disabled. Transmit master enable must be set in order for this bit to have an effect. The TxBit processor should be enabled for all applications.

FIELD NAME	BIT POSITION	DESCRIPTION
TxSonetEna	27	**TxSONET Enable** — When one, enables the TxSONET logic. When a zero, TxSONET logic is disabled and bypassed. When enabled this logic inserts SONET/SDH transport and path overhead into the data stream.
ForceLineAIS	26	**TxSONET Force Line Alarm Indicator Signal** — When a one, force the transmission of line alarm indication signal (AIS-L). However, when set to a one when *sonet_force_path_alarm_indication_signal* is set to a one, this bit does not force AIS-L; it enables transmission of new data flag in H1.
TxSonetOhComp	25	**TxSONET Overhead Complete** — When set to a one, the final element (E2) of transmit path and transport overhead has been read from the SONET/SDH transmit overhead buffer. Indicates that one more J1 has been updated in the J1 portion of that buffer. Operation of this indicator is complicated by the following: The SONET/SDH transmit pointer is fixed at decimal 40. Because of this, a complete set of path overhead is read in this order: Z3 Z4 Z5 J1 B3 C2 G1 F2 H4, not in the expected order of J1... Z5 This bit is also available in the SONET event register.
ForceSONETPErr	24	**TxSONET Force Parity Error** — When a one, transmits parity errors on all bit lanes of B1 B2 and B3. For diagnostics.

FIELD NAME	BIT POSITION	DESCRIPTION
TxAggMode	23:22	**SDP Transmit Aggregation Mode** — Controls the number of TxSDPs in a cluster that work as a unit. 00 = Each SDP works independently 01 = Reserved 10 = SDPs work as one group of four 11 = SDPs work as one group of four, output is byte-interleaved. Output bytes are selected in round robin order from CP0,1,2,3 to create a merged output stream. When using this mode, all four txbyte processors provide one data byte in turn which are merged following the tx large fifo and presented to either the tx sonet block or the txbit processor (depending on the state of TxSonetEna). If the fifo watermark logic is also enabled, each valid byte must have the data nine flag asserted. One invalid byte must also be inserted into each fifo to indicate packet boundary for the watermark logic. This mode of operation is typically used with the M-5 Channel Adapter.
TxFifoExDisab	21	**TXFIFO Exception Disable** — When set to 0, allow the TxSDP underflow handler to transmit well-formed, but benign PDUs in the case of FIFO underflow. When set to 1, prevent this mechanism from operating, which will cause the last data in the Tx small FIFO to be repeated until new data is available. For diagnostics. This bit is normally set to zero.
TxBitPHY	20:19	**TxBit PHY Status Select** — Selects how the physical layer status bits from the PHY or transceiver chips are connected to the TxBit processor. The TxBit processor has two PHY status tests: *phy_status_0* and *phy_status_1*. See Table 175.
TxSonetConcMode	18	**TxSONET Concatenation Mode** — When in OC-12 mode and set to 1, the transmit logic is configured to SONET OC-12c / SDH STM-4 VC4-4c (one pipe). When in OC-12 mode and set to zero, the transmit logic is configured to SONET OC-12 (four OC-3c streams) / SDH STM-4 (four VC-4-1c streams). When *TxSonetOC* is set to a zero (OC-3 mode), this bit must be set.

FIELD NAME	BIT POSITION	DESCRIPTION
ForcePathAIS3	17	**SONET Force Path Alarm Indicator Signal 3** — When a one, forces the path alarm indication signal (AIS-P) to be transmitted. The operation of this varies according to the SONET mode selected. For OC-3c mode only, CPs 3, 7, 11, 15 - AIS is controlled by the setting in ForcePathAIS3. In OC-12c mode, the *ForcePathAIS0* of CP0 is the only bit that is used. In OC-12 non-concatenated mode, each of the four bits in CP0 only controls one of the four OC-3c tributaries. When set to 1 and the *sonet_force_line_alarm_indication_signal* is set to 1, this bit does not force AIS-P; it enables transmission of new data flag in H1.
ForcePathAIS2	16	**SONET Force Path Alarm Indicator Signal 2** — See *ForcePathAIS3*. For OC-3c mode only, CPs 2, 6, 10, 14 - AIS is controlled by the setting in ForcePathAIS2.
ForcePathAIS1	15	**SONET Force Path Alarm Indicator Signal 1** — See *ForcePathAIS3*. For OC-3c mode only, CPs 1, 5, 9, 13 - AIS is controlled by the setting in ForcePathAIS1.
ForcePathAIS0	14	**SONET Force Path Alarm Indicator Signal 0** — See *ForcePathAIS3*. For OC-3c mode only, CPs 0, 4, 8, 12 - AIS is controlled by the setting in ForcePathAIS0.
SonetJ1Mis	13	**SONET J1 Mismatch** — When 1, the path extended remote defect indication field (ERDI-P), is set to reflect this condition. This bit is to be set when the received 64Byte J1 message does not match the message that is expected.

<table>
<tr><th>FIELD NAME</th><th>BIT POSITION</th><th>DESCRIPTION</th></tr>
<tr><td>TxBitOutWidth</td><td>12:11</td><td>TxBit Output Width — Determines the output data width transmitter by TxBit.

<table><tr><th>TXBIT OUTPUT WIDTH</th><th>INPUT WIDTH IN BITS</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>2</td></tr><tr><td>2</td><td>4</td></tr><tr><td>3</td><td>8</td></tr></table>
Overridden if the ten bit encoder is enabled</td></tr>
<tr><td>TxSonetOC</td><td>10</td><td>TxSONET OC12/OC3 Select — When a one, the SONET/SDH transmit logic is configured to transmit SONET OC-12 / SDH STM-4. When a zero, this transmit logic is configured to transmit SONET OC-3c / SDH STM-1 VC-4-1c.</td></tr>
<tr><td>TxSonetScr</td><td>9</td><td>TxSONET Scramble Enable — When 1, the data is frame-synchronously scrambled. When zero, the data is not scrambled.</td></tr>
<tr><td>TxByteCRCInit1</td><td>8</td><td>TxByte CRC Initialize to Ones — When a one, resetting the CRC register sets the CRC register to all ones. When a zero, resetting the CRC register sets the CRC register to all zeroes.</td></tr>
<tr><td>TxByteCRC16</td><td>7</td><td>TxByte CRC 16/32 — When a one, selects CRC-16 operation. When a zero, selects CRC-32 operation.</td></tr>
<tr><td>TxByteCRCInv</td><td>6</td><td>TxByte CRC Output Invert — Read the ones complement of the CRC register, when a one. When a zero, read the CRC register directly.</td></tr>
<tr><td>TxByteFirstOnLeft</td><td>5</td><td>TxByte First on Left — Calculate CRC on the data assuming that the first bit received is the left-most bit of the byte. Must be set the same as RxBitFirstOnLeft.</td></tr>
<tr><td>TxBitFirstOnLeft</td><td>4</td><td>TxBit First on Left — When 0, the first bit transmitted is the right-most bit of the byte. When 1, the first bit transmitted is the left-most bit of the byte.
Set to 0 for Ethernet. Set to 1 for SONET/SDH.</td></tr>
</table>

FIELD NAME	BIT POSITION	DESCRIPTION
PayloadScrEna	3	**TxByte Payload Scramble Enable** — When set to a one, enables the self-synchronous payload scrambler if the microcode also enables it. Otherwise, data is transferred without scrambling. Usually set if SONET/SDH is enabled.
PayloadScrMode	2	**TxByte Payload Scramble Insertion Mode** — Selects whether optional insertion and/or payload scrambling of is done before or after aggregation multiplexing. When zero, insert/scramble before multiplexing. When one, insert/scramble after multiplexing. OC-12c applications *must* set this to a one. OC-3c and OC-12 non concatenated applications *must* set this to a zero.
Tx10bit	1	**Tx Ten Bit Enable** — When set, the 8 bit to 10 bit encoder is enabled. The TxBit output width is set to 10 bits. When clear, the 8 bit to 10 bit encoder is disabled and TxBit output width is determined by the TxBit output width field as described below. This is used for Gigabit Ethernet 1000BaseX and Fibre Channel.
ForceLOS	0	**TxSONET Force Loss Of Signal** — When a one, turns the entire SONET/SDH frame, post-scrambling to zeroes. For diagnostics. To the receiver this looks just like a fiber cut.

Table 175 PHY Status Bit - TxBit Processor Connections

SELECT FIELD	OPERATION		
0 or 3	*phy_status_0* is connected to receive data signal A *phy_status_1* is connected to receive data signal B Where signals A and B are the following:		
	PIN MODE	**SIGNAL A**	**SIGNAL B**
	DS1/3	frame sync	undefined
	OC-3	undefined	signal detect
	OC-12	frame pulse	PLL lock detect

Table 175 PHY Status Bit - TxBit Processor Connections (continued)

SELECT FIELD	OPERATION
1	*phy_status_0* is connected to carrier sense (RMII) *phy_status_1* is connected to collision (RMII)
2	*phy_status_0* is connected to carrier sense (GMII) *phy_status_1* is connected to collision (GMII)

Debug_Mode Register (CP Mode Configuration Function)

Purpose Configures the CP debug tap for the global debug counters.

Address 0xBC*n*04658

Access CPRC Read/Write

Bit Position	31	30 28	27 24	23	22 20	19 16	15	14 12	11 8	7	6 4	3 0
Field Name	Enb0	Rsvd	MUX0	Enb1	Rsvd	MUX1	Enb2	Rsvd	MUX2	Enb3	Rsvd	MUX3
Reset Value	0	raz	x	0	raz	x	0	raz	x	0	raz	x

FIELD NAME	BIT POSITION	DESCRIPTION
Enb0	31	**Global Debug Wire 0 Enable** — Enable the driver onto global debug wire 0.
reserved	30:28	Read as zero.
MUX0	27:24	**Global Debug Wire 0 MUX** — Select 1 of 16 debug events onto global debug wire 0.
Enb1	23	**Global Debug Wire 1 Enable** — Enable the driver onto global debug wire 1.
reserved	22:20	Read as zero.
MUX1	19:16	**Global Debug Wire 1 MUX** — Select 1 of 16 debug events onto global debug wire 1.
Enb2	15	**Global Debug Wire 2 Enable** — Enable the driver onto global debug wire 2.
reserved	14:12	Read as zero.
MUX2	11:8	**Global Debug Wire 2 MUX** — Select 1 of 16 debug events onto global debug wire 2.

FIELD NAME	BIT POSITION	DESCRIPTION
Enb3	7	**Global Debug Wire 3 Enable** — Enable the driver onto global debug wire 3.
reserved	6:4	Read as zero.
MUX3	3:0	**Global Debug Wire 3 MUX** — Select 1 of 16 debug events onto global debug wire 3.

There are four global debug wires that carry inputs to the global debug counter block. Each CP has a multiplexor that can select one of the 16 events enumerated in Table 176 to drive on each of the respective debug wires. Each multiplexor has a 4bit register to select what event to drive, and an enable bit to turn on the debug wire driver. Chip-wide, only one debug wire driver should be enabled at any time. Because of this restriction, the recommended procedure for software to manipulate debug taps is as follows:

1. Clear the master debug enable bit in the global debug configuration register space in the XP.
2. Clear the set driver enable bits. It may be safest to invoke a routine that clears all driver enable bits on every change regardless of the previous configuration.
3. Set one chip-wide driver enable bit and its corresponding multiplexor select value for each global debug wire.
4. Set up the global debug configuration bits and master debug enable.

Table 176 Debug Multiplexor Select Encodings

MUX INPUT	ENCODING	DESCRIPTION
1	15	Always selects 1 for multiplexor.
0	14:8	Always selects 0 for multiplexor (unused).
SDP service request	7	Logical OR of all SDP sequencers service requests.
SDP TXByte	6	SDP TxByte sequence moved a byte of payload from DMEM.
SDP RxByte	5	SDP RxByte sequence moved a byte of payload to DMEM.
Istall	4	CPRC instruction stall cycle.
Stall	3	CPRC data read or write stall cycle.
CPRC Read	2	CPRC data read.
CPRC Write	1	CPRC data write.

Table 176 Debug Multiplexor Select Encodings (continued)

MUX INPUT	ENCODING	DESCRIPTION
DebugMatch	0	The CPRC data, data address, or instruction address matches the programmed match registers in the CPRC.

PIN_Mode Register (CP Mode Configuration Function)

Purpose — Provides programmable pin configuration state.

Refer to Configuring the SDP Pin Logic on the C-5e NP document for configuring protocol services not supported in the CPI.

Address — 0xBC*n*0465C

Access — CPRC Read/Write

Bit Position	31 … 23	22 … 21	20 … 18	17 … 14	13 … 7	6 … 0
Field Name	Reserved	DataCnfg	RxClkMUX	TxClkMUX	Reserved	TxDataEna
Reset Value	raz	0	0	0	raz	0

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:23	Read as zero.
DataCnfg	22:21	**Pin Data Configuration** — Sets CP configuration pins: 0,1 = CP pins in read-only mode 2 = CP pins in LVTTL mode (all other configurations) 3 = CP pins in PECL mode (OC-3) Note: When in PECL mode, this enables CPn_2 and CPn_3 drivers, independent of the settings for *TxDataEna* field.

FIELD NAME	BIT POSITION	DESCRIPTION
RxClkMUX	20:18	**Receive Clock MUX Control** — For CP0, CP4, CP8, and CP12 in each cluster, the receive clock MUX control cannot be disabled (set to a value of 0) for any cluster that is to be used for either RxSDP or TxSDP processing since this is how clock is driven throughout the SDPs for that cluster. For example, if the cluster consisting of CP0 - CP3 is being used to receive data, CP0 could be set to *RxClkMux* = *TxClkMux* (5) and the *TxClkMux* must select a driven input clock. RxClkMUX configurations are: 1 = local (CPn_1) 2 = x2 aggregate (CPn_1 of the aggregation pair) 3 = TBI - recovered from CP2_1 and CP3_1 in a four CP cluster 4 = PECL (for OC-3, from CPn_0 and CPn_1 differential input) 5 = transmit clock (for RMII) 6 = inverted transmit clock (for local loopback) 7 = x4 aggregate (CP2_1 is the clock in a four CP Cluster) 0 = disable receive clock (set internally to 0)

<table>
<tr><th>FIELD NAME</th><th>BIT POSITION</th><th>DESCRIPTION</th></tr>
<tr><td>TxClkMUX</td><td>17:14</td><td>Transmit Clock MUX Control — Indicates the source of the TxClock used by the SDP:
<table>
<tr><th>ENCODED VALUE</th><th>SOURCE</th><th>SUGGESTED PROTOCOL FREQUENCIES</th><th>APPLICABLE NOTES</th></tr>
<tr><td>1</td><td>CCLK0</td><td>T1</td><td rowspan="4">N/A</td></tr>
<tr><td>2</td><td>CCLK1</td><td>E1</td></tr>
<tr><td>3</td><td>CCLK2</td><td>E3</td></tr>
<tr><td>4</td><td>CCLK3</td><td>T3</td></tr>
<tr><td>5</td><td>CCLK4</td><td>RMII</td><td rowspan="2">CCLK4-7 are internally tied to 0 in C-3e NP.</td></tr>
<tr><td>6</td><td>CCLK5</td><td>Fibre Channel</td></tr>
<tr><td>7</td><td>MII clk (CPn_1 in each cluster)</td><td>MII</td><td>To be used by MII Mode for GMII autonegotiate down to 100/10BaseT.</td></tr>
<tr><td>9</td><td>CCLK6</td><td>GMII/Gigabit Ethernet</td><td rowspan="2">CCLK4-7 are internally tied to 0 in C-3e NP</td></tr>
<tr><td>A</td><td>CCLK7</td><td>OC3</td></tr>
<tr><td>B</td><td>internal0</td><td rowspan="4"></td><td rowspan="2">Internal0 and internal1 are internally buffered versions of the receive clocks on CP4 and CP8, respectively.</td></tr>
<tr><td>C</td><td>internal1</td></tr>
<tr><td>D</td><td>receive clock</td><td>Use the 'even' receive clock in an 'even-odd' pair, for example CP8 and CP9 will use receive clock from CP8.</td></tr>
<tr><td>E</td><td>receive clock</td><td>N/A</td></tr>
<tr><td colspan="4">0,8,F = transmit clock disabled (internally set to 0) for both C-5e and C-3e</td></tr>
</table></td></tr>
<tr><td>Reserved</td><td>13:7</td><td>Read as zero.</td></tr>
</table>

FIELD NAME	BIT POSITION	DESCRIPTION
TxDataEna	6:0	**Transmit Data Enable** — Selects transmit or receive mode for each of the CP's pins: 0 = Receive 1 = Transmit Bits 0 - 6 map individually to each of the seven pins in each CP. Bit 1 maps CPn_1, bit 2 maps to CPn_2, and so on. Thus pins 0-4 could be in Rx mode while pins 5 and 6 could be in Tx mode. Note: When PECL mode is selected by *DataCnfg* field, then *TxDataEna* field is irrelevant. PECL drivers are predetermined to transmit on CPn_2/3 (tx data) and receive on CPn_0/1, CPn_4/5 and CPn_6 (clk, rx data, and signal_detect).

Queue_Status0 Register (CP Queue Status Function)

Purpose Stores queue status broadcast by the queue controller. See Table 177 on page 549 for similar registers.

Address 0xBC*n*04660

Access CPRC Read, Write one to clear

Bit Position	31	0
Field Name	**Status**	

Table 177 Queue_Status*n* Registers (for Queue Status 1, 2 and 3)

REGISTER NAME	PURPOSE	ADDRESS
Queue_Status1	Same as *Queue_Status0*.	0xBC*n*04664
Queue_Status2	Same as *Queue_Status0*.	0xBC*n*04668
Queue_Status3	Same as *Queue_Status0*.	0xBC*n*0466C

Queue_Update0 Register (CP Queue Status Function)

Purpose Address through which bits are set in the queue status registers. See Table 178 on page 550 for similar registers.

Address 0xBC*n*04670

Access CPRC write one to set the corresponding bit of the queue status register.

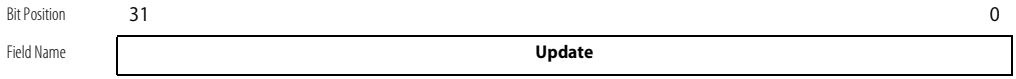


Table 178 Queue_Update*n* Registers (for Queue Updates 1, 2 and 3)

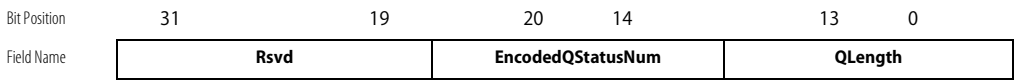
REGISTER NAME	PURPOSE	ADDRESS
Queue_Update1	Same as *Queue_Update0*.	0xBC*n*04674
Queue_Update2	Same as *Queue_Update0*.	0xBC*n*04678
Queue_Update3	Same as *Queue_Update0*.	0xBC*n*0467C

Queue_Empty Register (Aggregated Queueing Function)

Purpose Used to clear a bit of the Queue_Status registers.

Address 0xBC*n*04680

Access Write only

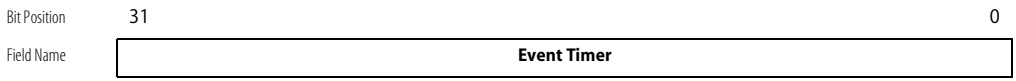


Event_Timer Register (CP Miscellaneous Control Function)

Purpose Cycle counter used to schedule timed events.

Address 0xBC*n*04684

Access CPRC Read/Write



Cycle_Count_H Register (CP Miscellaneous Control Function)

Purpose Most significant four bytes of the 64bit cycle counter, updated whenever Cycle_Count_L is read.

Address 0xBC*n*04688

Access CPRC Read

Bit Position	63	32
Field Name	**Frozen Count**	
Reset Value	**0**	

Cycle_Count_L Register (CP Miscellaneous Control Function)

Purpose Least significant four bytes of the 64bit cycle counter.

Address 0xBC*n*0468C

Access CPRC Read

Bit Position	31	0
Field Name	**Count**	
Reset Value	**0**	

Queue_Ctl Register (Aggregated Queueing Function)

Purpose Used to select queue aggregation level.

Address 0xBC*n*04690

Access Global Read/Write

Bit Position	31	2	1	0
Field Name	**Rsvd**		**QueueAggrMode**	

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:2	Read as zero.

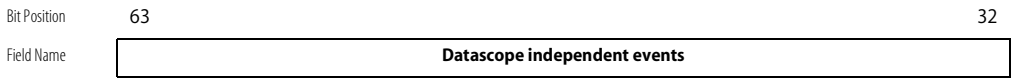
FIELD NAME	BIT POSITION	DESCRIPTION
QueueAggrMode	1:0	**Queue Aggregation Mode** — Selects queue aggregation mode as detailed here: ENCODED VALUE / MODE 00 — Single CP queueing (x1) (1CP) 01 — Cluster aggregation (x4) (4CPs clustered) 10 — Full-chip aggregation (x16) (all 16CPs clustered) 11 — Reserved

Event0 Register (CP Event and Interrupt Function)

Purpose Collects event bits relevant to datascope independent tasks.

Address 0xBC*n*046A0

Access CPRC Read, CPRC Write 1 bit to clear.



FIELD NAME	BIT POSITION	DESCRIPTION
WindDown	63	**Wind Down** — When unmasked, this global input is a request to wind down all CP activity as soon as possible, and leave as much predictable error recovery state around as possible.
GlobalError	62	**CPRC Global Reference Error** - when asserted, this bit means a CPRC Write received an error on the Global Bus or a non-existent memory error within the cluster.
PayloadError	61	**Payload Error** — Indicates an unrecoverable error occurred during a request sent to the BMU or QMU. An error status code is stored in the cp_mode_register, and also in the control block that initiated the request. Table 147 on page 495 for further description of causes of the error. Specifically, Error Codes A, C, D, E, and F cause MCError to be set.

FIELD NAME	BIT POSITION	DESCRIPTION
QMUError	60	**QMU Error** — Indicates an unrecoverable error occurred during a request sent to the QMU. An error status code is stored in the *cp_mode_register*.
XPInterrupt	59	**XP Interrupt Request** —The XP issued an interrupt request to this CP.
PayloadAlert	58	**Payload Request Alert** — A non-fatal bus error has occurred while trying to send a request to the BMU or QMU. Refer to Table 147 on page 495 for further description of causes of the alert. Specifically, the five (5) Error Codes encoded 9 cause PayloadAlert to be set.
DebugMatch	57	**CPRC Debug Match** — The CPRC data, or data address matched the programmed match registers in the CPRC.
TLUError	56	**TLU Error** — Indicates that an unrecoverable error occurred during a request sent to the TLU.
TxSDPError	55	**TxSDP Error** — Indicates that the SDP has encountered an error during processing. An error status code is stored in the *sdp_mode* register. This bit is the logical OR of the following conditions which are represented as bits in the *CP_MODE1* register: SDP TxBit service request [21] SDP TxByte service request [22] TxSmallFIFO underflow (SFUF) [0] TxLargeFIFO underflow (LFUF) [1] as well as these two bits: TxCtl0_Status [30] and TxCtl1_Status [30] error bit set (by TxByte)
RxSDPError	54	**RxSDP Error** — Indicates that the SDP has encountered an error during processing. An error status code is stored in the *sdp_mode* register.This bit is the logical OR of the following conditions which are represented as bits in the *CP_MODE1* register: SDP RxBit service request [25] SDP RxSync service request [26] SDP RxByte service request [27] RxSmallFIFO overflow (SFOF) [2] RxLargeFIFO overflow (LFOF) [3] as well as these two bits: RxCtl0_Status [30] and RxCtl1_Status [30] error bit set (by RxByte)

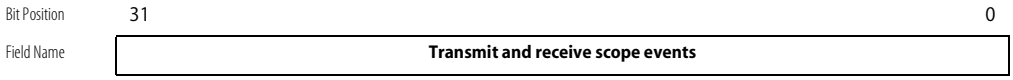
FIELD NAME	BIT POSITION	DESCRIPTION
RxMsgFIFO	53	**Ring Bus Receive Message Available** — This bit indicates the availability of a Ring Bus message in the receive FIFO, and corresponds to *RxMsgCtl.State* [31].
TimerEvent	52	**Event Timer Time-out** — This bit indicates the event timer counted down to 0.
AllCpInt	51	**All CPs Interrupt Request** — One CP has interrupted all other CPs.
SonetOH	50	**SONET Overhead Event** — Masked OR of all bits in the SONET Event register. This bit is level sensitive.
Reserved	49:48	Software controlled.
RxResp7-0	47:40	**Ring Bus Receive Response Available** — These eight bits correspond to the available bit for the eight Ring Bus receive response available bits. Bit 47 represents *RxResp7Ctl.Avail*, and bit 40 represents *RxRespCtl0.Ctl*. The events are triggered on a message basis, that is, one event for the beginning slot per multi-slot message.
TxMsg3-0	39:36	**Ring Bus Transmit Message Available** — These four bits correspond to the available bit for the four Ring Bus transmit message control registers. Bit 39 represents *TxMsg3Ctl.Avail*, and bit 36 represents *TxMsg0Ctl.Avail*.
WrCB	35:34	**Write Control Blocks 0/1** — These two bits correspond to the available bit for the two payload bus write control blocks. Bit 35 corresponds to *WrCB1Ctl.Avail*, and bit 34 corresponds to *WrCB0Ctl.Avail*.
RdCB	33:32	**Read Control Blocks 0/1** — These two bits correspond to the available bit for the two payload bus read control blocks. Bit 33 corresponds to *RdCB1Ctl.Avail*, and bit 32 corresponds to *RdCB0Ctl.Avail*.

Event1 Register (CP Event and Interrupt Function)

Purpose Collects together event bits relevant to transmit and receive datascopes.

Address 0xBC*n*046A4

Access CPRC Read, CPRC Write 1 bits to clear



FIELD NAME	BIT POSITION	DESCRIPTION
QRdMbxAvail	31	**Queue Read Mailbox Available** — This bit indicates that this CP's read mailbox in the QMU went from busy to available.
TxCB1_Avail	30	**TxCB1 Available** — Indicates that the available bit for datascope 1 Payload bus transmit control block *TxCB1Ctl.Avail* was set.
TxStatus1_Avail	29	**TxStatus1 Available** — Indicates that the TxSDP has set the *TxStatus1.Avail*.
TxStatus1_L1	28	**TxStatus1 Bit 1** — Indicates that the TxSDP has set the *TxStatus1* bit 1.
TxStatus1_L0	27	**TxStatus1 Bit 0** — Indicates that the TxSDP has set the *TxStatus1* bit 0.
QRdMbxBusy	26	**Queue Read Mailbox Busy** — This bit indicates that this CP's read mailbox in the QMU went from available to busy.
TxCB0_Avail	25	**TxCB0 Available** — Indicates that the available bit for datascope 0 payload bus transmit control block *TxCB0Ctl.Avail* was set.
TxStatus0_Avail	24	**TxStatus0 Available** — Indicates that the TxSDP has set the *TxStatus0.Avail*.
TxStatus0_L1	23	**TxStatus0 Bit 1** — Indicates that the TxSDP has set the *TxStatus0* bit 1.
TxStatus0_L0	22	**TxStatus0 Bit 0** — Indicates that the TxSDP has set the *TxStatus0* bit 0.
QueueStatus	21	**Queue Status** — This bit corresponds to the logical OR of all the bits in the four *Queue_Status* registers. The bit is level sensitive.

FIELD NAME	BIT POSITION	DESCRIPTION
SoftEvent2 [4:0]	20:16	**Software Events 2 [4:0]** — These five bits correspond to events that are set explicitly by software.
QWrMbxAvail	15	**Queue Write Mailbox Available** — This bit indicates that this CP's write mailbox in the QMU went from busy to available.
RxCB1_Avail	14	**RxCB1 Available** — Indicates that the available bit for datascope 1 payload bus receive control block *RxCB1Ctl.Avail* was set.
RxStatus1_Avail	13	**RxStatus1 Available** — Indicates that the RxSDP has set the *RxStatus1.Avail*.
RxStatus1_L1	12	**RxStatus1 Bit 1** — Indicates that the RxSDP has set the *RxStatus1* bit 1.
RxStatus1_L0	11	**RxStatus1 Bit 0** — Indicates that the RxSDP has set the *RxStatus1* bit 0.
QWrMbxBusy	10	**Queue Write Mailbox Busy** — This bit indicates that this CP's write mailbox in the QMU went from available to busy.
RxCB0_Avail	9	**RxCB0 Available** — Indicates that the available bit for datascope 0 payload bus receive control block *RxCB0Ctl.Avail* was set.
RxStatus0_Avail	8	**RxStatus0 Available** — Indicates that the RxSDP has set the *RxStatus0.Avail*.
RxStatus0_L1	7	**RxStatus0 Bit 1** — Indicates that the RxSDP has set the *RxStatus0* bit 1.
RxStatus0_L0	6	**RxStatus0 Bit 0** — Indicates that the RxSDP has set the *RxStatus0* bit 0.
Reserved	5	Software controlled.
SoftEvent3 [5:0]	4:0	**Software Events 3 [5:0]** — These six bits correspond to events that are set explicitly by software. Bit 5 corresponds to software_event3_5.

Event_Mask0 Register (CP Event and Interrupt Function)

Purpose — Provides mask that selects bits in the *Event0* register for event access. A 1 enables, a 0 disables interupts for the corresponding bit in the *Event0* register. See Table 179 on page 557 for similar register.

Address — 0xBC*n*046A8

Access — CPRC Read, CPRC Write

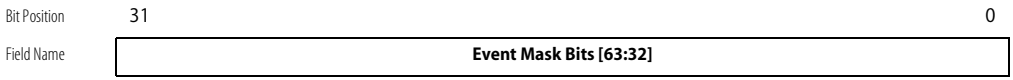


Table 179 Event_Mask1 Register (for Mask1)

REGISTER NAME	PURPOSE	ADDRESS
Event_Mask1	Same as *Event_Mask0*, except it masks events [31:0]	0xBC*n*046AC

Event_Access Register (CP Event and Interrupt Function)

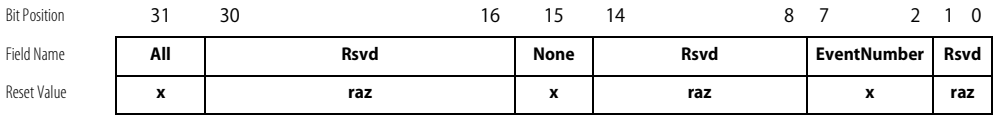
Purpose — Provides access to next high bit (1) of *Event* register pair (*Event0* and *Event1*) that is set in *Event_Mask* register pair *(Event_Mask0* and *Event_Mask1*) and also provides event summaries.

Note: The fields for this register change when performing a Read as opposed to performing a Write.

Address — 0xBC*n*046B0

Access — CPRC Read, CPRC Write to set/clear *Event* register bit.

The Read Format is:



FIELD NAME	BIT POSITION	DESCRIPTION
All	31	**All** — Provides logical-AND of *EVENT* register bits that are active in the mask.
Reserved	30:16	Read as zero.

FIELD NAME	BIT POSITION	DESCRIPTION
None	15	**None** — Provides logical-NOR of *EVENT* register bits that are active in the mask.
Reserved	14:8	Read as zero.
EventNumber	7:2	**Event Number** — Denotes the highest number selected event.
Reserved	1:0	Read as zero.

The Write Format is:

Bit Position	31 — 24	23 — 18	17 — 8	7 — 2	1 — 0
Field Name	**Rsvd**	**SetBit**	**Rsvd**	**ClearBit**	**Rsvd**
Reset Value	**raz**	**x**	**raz**	**x**	**raz**

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:24	Read as zero.
SetBit	23:18	**Set Bit** — Sets an individual bit in the *EVENT_MASKn* register.
Reserved	17:8	Read as zero.
ClearBit	7:2	**Clear Bit** — Clears an individual bit in the *EVENT_MASKn* register.
Reserved	1:0	Read as zero.

Mask_Access Register (CP Event and Interrupt Function)

Purpose Provides decoder to access bits in *Event_Mask* register pair (*Event_Mask0* and *Event_Mask1*)one at a time.

Address 0xBC*n*046B4

Access CPRC Write

Bit Position	31 24	23 18	17 8	7 2	1 0
Field Name	Rsvd	SetBit	Rsvd	ClearBit	Rsvd
Reset Value	raz	x	raz	x	raz

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:24	Read as zero.
SetBit	23:18	**Set Bit** — Sets an individual bit in the *EVENT_MASKn* register.
Reserved	17:8	Read as zero.
ClearBit	7:2	**Clear Bit** — Clears an individual bit in the *EVENT_MASKn* register.
Reserved	1:0	Read as zero.

Interrupt_Mask0 Register (CP Event and Interrupt Function)

Purpose Provides mask that select bits in the *Event* register pair (*Event0* and *Event1*) for interrupt reporting events [63:48] for IRQ0 and events [47:32] for IRQ1. See Table 180 on page 560 for similar register.

Address 0xBC*n*046B8

Access CPRC Read/ Write

Bit Position	31 16	15 0
Field Name	Interrupt Mask Bits [63:48] OR IRQ0	Interrupt Mask Bits [47:32] OR IRQ1

Table 180 Interrupt_Mask1 Register (for Mask Events [31:16] and [15:0])

REGISTER NAME	PURPOSE	ADDRESS
Interrupt_Mask1	Same as *Interrupt_Mask0*, except it masks events [31:16] for IRQ2 and events [15:0] for IRQ3.	0xBC*n*046BC

SONET_Event Register (CP Event and Interrupt Function)

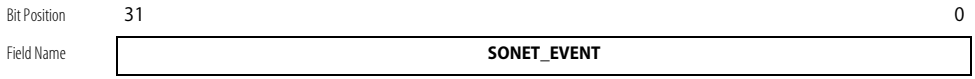
Purpose — Collects together SONET/SDH event bits from the rxSONET and txSONET blocks in the SDP.

For all the _DELTA fields only, a 1 indicates a change in that bit's state since it was last cleared. _DELTA bits require reading the corresponding *SDP_Mode2* register bits [31:22]. For the current values in the SONET/SDH receive overhead, see registers starting at *Rx_SONETOH0* to *Rx_SONETOH31* in Appendix C. All of the bits in the *SONET_Event* register can cause an interrupt to be generated on the CPRC. To enable these interrupts, unmask the corresponding bit in the *SONET_Mask* register and enable the SonetOH bit [50] in the *Event_Mask0* register.

Refer to Table 181 on page 568 for CP configurations for monitoring in an aggregated application.

Address — 0xBC*n*046C0

Access — CPRC Read, CRC Write 1 bit to clear.



FIELD NAME	BIT POSITION	DESCRIPTION
LOS_DELTA	31	**Loss of Signal State Changed** — Loss of Signal State Changed: LOS_DELTA indicates a change has occurred in the LOS state (either ON or OFF). This bit is latched and remains set until cleared by software. Write 1 to clear. LOS is determined through monitoring of edges detected on the line signal. See LOS in *SDP_Mode2* for the current state and detailed definition of SONET/SDH LOS.

FIELD NAME	BIT POSITION	DESCRIPTION
LOF_DELTA*	30	**Loss of Framing State Changed** — LOF_DELTA indicates a change has occurred in the LOF state. This bit is latched and remains set until cleared by software. Write 1 to clear. LOF is determined through monitoring of the A1 and A2 bytes (row, column, sts) (1,1,1) through OC3c(1,2,3) OC12/c(1,2,12). See LOF in *SDP_Mode2* for the current state and definition of SONET/SDH LOF.
AIS_L_DELTA	29	**Line Alarm Indication Signal State Changed** — AIS_L_DELTA indicates a change has occurred in the AIS_L state since the last time this bit was cleared. This bit is latched and remains set until cleared by software. Write 1 to clear. AIS_L is determined by information in the K2 byte (5,3,1). See AIS_L in *SDP_Mode2* for the current state and definition of SONET/SDH AIS_L.
REI_L/Z2_DELTA	28	**Line Remote Error Indicator** — • For OC3c: REI_L errors occurred since the last time this bit was cleared by software. REI_L errors are observed in the M1 line overhead byte (9,2,3). Up to 24 (OC3c) or 96 (OC12/c) far end errors can be detected via this byte. This bit is latched and remains set until cleared by software after an error occurs. Write 1 to clear. • For OC12/c - third CP of cluster, REI-L (STS-3) Third CP of cluster: REI_L errors occurred since the last time this bit was cleared by software. REI_L errors are observed in the M1 line overhead byte (9,2,3) on the third CP in the cluster. Up to 96 far end errors can be detected via this byte. This bit is latched and remains set until cleared by software after an error occurs. Write 1 to clear. **Z2_Delta** — For OC12/c first, second or fourth CP of cluster, Z2_Delta Z2 value changed: Z2 STS 1,2 or 4 has Changed: The indicated Z2 growth byte row 9, column 2, STS 1,2,4 value has changed. STS 9 is available on the first CP in the cluster and the remaining STS follow consecutively on the remaining CPs. Located in the Z2 growth byte, this bit indicates that the value of Z2 has changed since the last time this bit was cleared. This bit is latched and remains set until cleared by software. Write 1 to clear.

FIELD NAME	BIT POSITION	DESCRIPTION
RDI_L_DELTA	27	**Line Remote Defect Indicator State Changed** — RDI_L_DELTA indicates a change has occurred in the RDI_L state since the last time this bit was cleared. RDI_L is determined by information in the K2 byte (5,3,1). This bit is latched and remains set until cleared by software. Write 1 to clear. See RDI_L in *SDP_Mode2* for the current state and definition of RDI_L.
LOP_P_DELTA	26	**Path Loss Of Pointer State Changed** — LOP_P_DELTA indicates a change has occurred in the LOP_P state since the last time this bit was cleared. This bit is latched and remains set until cleared by software. Write 1 to clear. LOP_P is determined through information in the H1, H2 and H3 bytes (4,1,1) through (4,3,3) OC3c (4,3,12) OC12/c. See LOP_P in *SDP_Mode2* for the current state and definition of LOP_P.
AIS_P_DELTA	25	**Path Alarm Indication Signal State Changed** — AIS_P_DELTA indicates a change has occurred in the AIS_P state since the last time this bit was cleared. This bit is latched and remains set until cleared by software. Write 1 to clear. AIS_P is determined through the H1, H2 and H3 bytes (4,1,1) through OC3c (4,3,3) OC12/c (4,3,12). See AIS_P in *SDP_Mode2* for the current state and definition of AIS_P.
REI_P	24	**Path Remote Error Indicator** — REI_P errors occurred since the last time this bit was cleared by software. REI_P errors are observed in the G1 path overhead byte of the SPE (Synchronous Payload Envelope). For OC12, there are 4 G1 bytes located in the SPE. This bit is latched and remains set until cleared by software after an error occurs.
ERDI_P_DELTA	23	**Path Extended Remote Defect Indicator State Changed** — ERDI_P_DELTA indicates a change has occurred in the ERDI_P state since the last time this bit was cleared. This bit is latched and remains set until cleared by software. Write 1 to clear. See RDI_P in *SDP_Mode2* for the current state and definition of RDI_P.
LCD_P_DELTA	22	**Loss of Cell (or Packet) Delineation State Changed** — LCD_P_DELTA indicates a change has occurred in the LCD_P state since the last time this bit was cleared. This bit is latched and remains set until cleared by software. Write 1 to clear. See LCD_P in *SDP_Mode2* for the current state and definition of LCD_P.

FIELD NAME	BIT POSITION	DESCRIPTION
APS_ERROR	21	**APS Inconsistency Error** —Automatic Protection Switching Inconsistency Error: This bit indicates that in the previous 12 frames, there were *no* three consecutive frames that had the same value for K1 or K2. This bit is latched and remains set until cleared by software. Write 1 to clear.
B1_ERROR/FRAME CNT_REACHED	20	If *SDP_Mode4* FrameMode bit is set to 0, this field is defined as B1_ERROR. If the FrameMode bit is set to 1, this field is defined as FRAMECNT_REACHED. **B1 Section Parity Error** —The B1 byte is located in the B1 byte (2,1,1). This bit indicates that 1 or more B1 parity errors were detected on the incoming frame. See the Rx_SONETOH register 0xbcn04503 for the total number of errors (0-8) in the last frame. **Frame Count Reached** —The number of frames set in the SONET APS *FrameCnt* field in *SDP_Mode4* have completed. Note, that this interrupt could be used to perform some user specific overhead monitoring every *FrameCnt* frames. This bit is latched and remains set until cleared by software. Write 1 to clear.
B2_ERROR/ B2_ACCUM_FRAM ECNT	19	If *SDP_Mode4* FrameMode bit is set to 0, this field is defined as B2_ERROR. If the FrameMode bit is set to 1, this field is defined as B2_ACCUM_FRAMECNT. **B2 Line Parity Error** — B2 is located in (1,5,1) through (1,5,3) for OC3c and (1,5,12) for OC12/c B2 bytes. This bit indicates that 1 or more B2 parity errors were detected in the incoming frame. For OC3c there are 3 B2 bytes and for OC12/c there are 12 B2 bytes. See the Rx_SONETOH registers for the total number of errors (0-96) for OC12/c or (0-24) for OC3c. **B2 Accumulated Errors in Last Frame Count Interval** — The number of frames set in the SONET/SDH APS *FrameCnt* field in *SDP_Mode4* have completed and B2 errors have been observed in the last FrameCnt frames. See the associated RxSONET_OH accumulated B2 counts. Note: This bit is designed primarily for SONET APS SD/SF Bit Error Rate determination.

FIELD NAME	BIT POSITION	DESCRIPTION
Z2_1_DELTA	18	**Z2-1 Value Changed** — • For OC3c, Z2 STS 2 Changed: The indicated Z2 growth byte row 9, column 2, STS 2 has changed. Located in the Z2 growth byte, this bit indicates that the value of Z2 has changed since the last time this bit was cleared. This bit is latched and remains set until cleared by software. Write 1 to clear. • For OC12/c, Z2 STS 9,10,11 or 12 has Changed: The indicated Z2 growth byte row 9, column 2, STS 9,10,11,12 value has changed. STS 9 is available on the first CP in the cluster and the remaining STS follow consecutively on the remaining CPs. Located in the Z2 growth byte, this bit indicates that the value of Z2 has changed since the last time this bit was cleared. This bit is latched and will remain set until cleared by software. Write 1 to clear.
Z2_0_DELTA	17	**Z2-0 Value Changed** — • For OC3c, Z2 STS 1 Changed: The Z2 growth byte row 9, column 2, STS 1 has changed. Located in the Z2 growth byte, this bit indicates that the value of Z2 has changed since the last time this bit was cleared. This bit is latched and remains set until cleared by software. Write 1 to clear. • For OC12c, Z2 STS 5,6,7,8 Changed: The Z2 growth byte row 9, column 2, STS 5,6,7,8 value has changed. STS 5 is available on the first CP in the cluster and the remaining STS follow consecutively on the remaining CPs. Located in the Z2 growth byte, this bit indicates that the value of Z2 has changed since the last time this bit was cleared. This bit is latched and remains set until cleared by software. Write 1 to clear.

FIELD NAME	BIT POSITION	DESCRIPTION
Z1_2_DELTA	16	**Z1-1 Value Changed** — • For OC3c, Z1 STS 2 Changed: The Z1 growth byte row 9, column 1, STS 2 has changed. Located in the Z1 growth byte, this bit indicates that the value of Z2 has changed since the last time this bit was cleared. This bit is latched and remains set until cleared by software. Write 1 to clear. • For OC12c, Z1 STS 9,10,11,12 Changed: The Z1 growth byte row 9, column 1, STS 9,10,11,12 value has changed. STS 9 is available on the first CP in the cluster and the remaining STS follow consecutively on the remaining CPs. Located in the Z2 growth byte, this bit indicates that the value of Z1 has changed since the last time this bit was cleared. This bit is latched and remains set until cleared by software. Write 1 to clear.
Z1_1_DELTA	15	**Z1-0 Value Changed** — • For OC3c, Z1 STS 1 Changed: The Z1 growth byte row 9, column 1, STS 1 has changed. Located in the Z1 growth byte, this bit indicates that the value of Z2 has changed since the last time this bit was cleared. This bit is latched and remains set until cleared by software. Write 1 to clear. • For OC12 c, Z1 STS 5,6,7,8 Changed: The Z1 growth byte row 9, column 1, STS 5,6,7,8 value has changed. STS 5 is available on the first CP in the cluster and the remaining STS follow consecutively on the remaining CPs. Located in the Z2 growth byte, this bit indicates that the value of Z1 has changed since the last time this bit was cleared. This bit is latched and remains set until cleared by software. Write 1 to clear.
S1_DELTA	14	**S1 Value Changed** — S1 row 9, column 1 STS 1, value has changed. This bit indicates that the value of S1 has changed since the last time this bit was cleared. This bit is latched and remains set until cleared by software. Write 1 to clear.
APS_DELTA	13	**APS Value Changed** — The values of K1 (5,2,1) or K2 (5,3,1) bits have changed to a new value and remained consistent for 3 frames in a row. Note: If the value of K1 or K2 does not remain consistent for 3 frames in a row after changing to a new value, this bit is *not* set. The update to the K1 RxSONETOH registeris only made if the value remains consistent.

FIELD NAME	BIT POSITION	DESCRIPTION
J0_DELTA	12	**J0 Value Changed** — The value of J0 (1,3,1) has changed since the last time this bit was cleared. This bit is latched and remains set until cleared by software.
B3_ERROR/BIP_ACCUM_FRAMECNT	11	If *SDP_Mode4* FrameMode bit is set to 0, this field is defined as B3_ERROR. If the FrameMode bit is set to 1, this field is defined as BIP_ACCUM_FRAMECNT. **B3 Path Parity Error** — This bit indicates that 1 or more B3 parity errors were detected in the incoming frame. For OC3c/OC12c there is 1 B3 bytes and for OC12 there are 4 B3 bytes. See the Rx_SONETOH registers for the total number of errors detected in the last frame (0-8) for OC3c/OC12c or (32) for OC12. This bit is latched and remains set until cleared by software. **BIP Accumulated Errors in Last Frame Count Interval** — The number of frames set in the SONET/SDH APS *FrameCnt* field in *SDP_Mode4* have gone by and B1, B2, B3, REI-L or REI-P errors have been observed in the last FrameCnt frames. See the associated RxSONET_OH accumulated B1, B2, B3, REI-L and REI-P counts. Refer to Table 181 on page 568 that maps for an aggregated application, which CPs should monitor the indicated accumulated parity count.
PTR_JUST_EVENT	10	**Pointer Justification Event** — The value of the pointer (evaluating H1, H2, H3) has incremented or decremented (implementing the 8 of 10 rule). This bit is latched and remains set until cleared by software. Write 1 to clear.
NDF	9	**New Data Flag in Pointer** — If the most significant nibble of H1 is set to 9, a new data flag indication has been received. Under normal opeartion this nibble is set to 6. This bit is latched and remains set until cleared by software. Write 1 to clear.
C2_DELTA	8	**Path Payload Label Mismatch or Unequipped** — When the value of C2 received does not match that written to the *SDP_Mode2* Sonet_C2_Exp[15:8] value for 5 consecutive frames this bit is set. This bit is set if a transition from the states of matched, mismatched or unequipped occurs. This bit is latched and remains set until cleared by software. Write 1 to clear.

FIELD NAME	BIT POSITION	DESCRIPTION
J1_AVAIL	7	**Receive J1 Available** — The value of J1 indexed by the SDP_Mode2 SONET_J1_Idx[5:0] value (0-63) has been written to the rxSONETOH registers since the last time this bit was cleared. This bit is latched and remains set until cleared by software. J1 could be monitored as slowly as once every 64 frames. To do this, set the SONET_J1_Idx[5:0] value to zero after every J1_Available SONET event.
Z5_DELTA	6	**Z5 Value Changed** — The value of the path overhead growth byte Z5 has changed since the last time this event was cleared. This bit is latched and remains set until cleared by software. Write 1 to clear.
Z4_DELTA	5	**Z4 Value Changed** — The value of the path overhead growth byte Z4 has changed since the last time this event was cleared. This bit is latched and remains set until cleared by software. Write 1 to clear.
Z3_DELTA	4	**Z3 Value Changed** — The value of the path overhead growth byte Z3 has changed since the last time this event was cleared. This bit is latched and remains set until cleared by software. Write 1 to clear.
H4_DELTA	3	**H4 Value Changed** — The value of the H4 byte has changed since the last time this event was cleared. This bit is latched and remains set until cleared by software. Write 1 to clear.
TX_OH_COMPLETE	2	**Transmit Overhead Complete** — The last CS1 byte ((9,3,3) OC3c or (9,3,12) OC12c) has been transmiited for the current frame by the txSONET block. Software can use this as an indicator for when to write new transmit overhead data to the transmit SONET/SDH registers. Software has 37 * the number of STS (3 or 12) byte times to update transmit SONET/SDH overhead registers before the first byte of transport overhead is read by the TxSONET block.
RX_POH_AVAIL	1	**Receive Path Overhead Available** — The last Z5 byte or fixed stuff byte (OC12c) has been written to the RxSONET registers. Software can use this as an indicator for when to read the rxSONET registers for the current frame. Software has 90 * the number of STS (3 or 12) byte times to read the RxSONET overhead registers before the path overhead starts to be overwritten by the RxSONET block.

FIELD NAME	BIT POSITION	DESCRIPTION
RX_TOH_AVAIL	0	**Receive Transport Overhead Available** — The last CS1 byte has been written to the rxSONET registers. Software can use this as an indicator for when to read the rxSONET transport overhead registers for the current frame. Software has 87 * the number of STS (3 or 12) byte times to read the RxSONET overhead registers before the path overhead starts to be overwritten by theRxSONET block.

* The LOS indication requires the transmit logic enabled and the SONET/SDH clock set to the correct frequency.

Table 181 on page 568 indicates for an aggregated application, which CPs should monitor the indicated accumulated parity count.

Invalid results may be observed if not used in this configuration.

Table 181 CP Configurations to Monitor Accumulated Parity Counts in an Aggregated Application

BIP	CPn	CPn+1	CPn+2	CPn+3
B1	X			
B2			X	
B3	OC12c OC12nc	OC12nc	OC12nc	OC12nc
REI_L			X	
REI_P	OC12c OC12nc	OC12nc	OC12nc	OC12nc

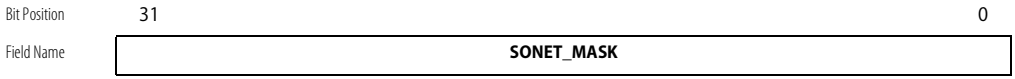
SONET_Mask Register (CP Event and Interrupt Function)

Purpose — Provides mask that selects bits in the *SONET_Event* register for event access. See bit field definitions in *SONET_Event* register.

When *SONET_Event* register logical AND *SONET_Mask* register results in any bit=1, then the CP *Event0* register *SONETOH Event* field bit [50]=1.

Address — 0xBC*n*046C4

Access — CPRC Read/ Write

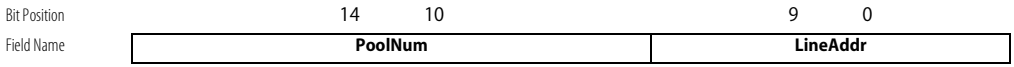


RdCB0_BTag_Alloc (CP Rd Control Block0 Fixed Function)

Purpose — Launches a 32 BTag allocate command with a single store. See Table 182 on page 569 for similar register.

Address — 0xBC*n*04700

Access — CPRC Write Only



FIELD NAME	BIT POSITION	DESCRIPTION
PoolNum	14:10	**Pool Number** — Pool number from which to allocate BTags.
LineAddr	9:0	**Line Address** — Block address in DMEM for allocated BTags.

Table 182 RdCB1_BTag_Alloc Register (for Control Block1)

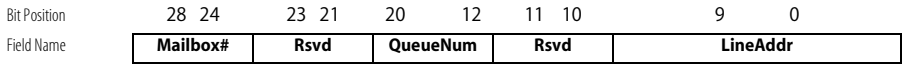
REGISTER NAME	PURPOSE	ADDRESS
RdCB1_BTag_Alloc	Same as *RdCB0_BTag_Alloc*, except for control block1.	0xBC*n*04710

RdCB0_Dequeue (CP Rd Control Block0 Fixed Function)

Purpose Launches a QMU dequeue command with a single store. Waits for QMU read mailbox to be ready before launching control block transfer. See Table 183 on page 570 for similar register.

Address 0xBC*n*04704

Access CPRC Write Only



FIELD NAME	BIT POSITION	DESCRIPTION
Mailbox#	28:24	**Mailbox Number** — QMU mailbox number, generally the CP_ID.
Reserved	23:21	Read as zero.
QueueNum	20:12	**Queue Number** — Queue number from which to dequeue.
Reserved	11:10	Read as zero.
LineAddr	9:0	**Line Address** — Block address in DMEM for descriptor data.

Table 183 RdCB1_Dequeue Register (for Control Block1)

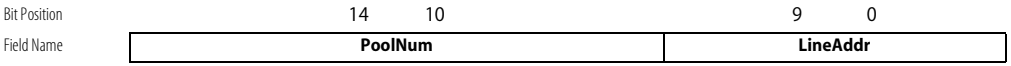
REGISTER NAME	PURPOSE	ADDRESS
RdCB1_Dequeue	Same as *RdCB0_Dequeue*, except for control block1.	0xBC*n*04714

WrCB0_BTag_Deallocate (CP Wr Control Block0 Fixed Function)

Purpose Launches a BTag deallocate command with a single store. See Table 184 on page 571 for similar register.

Address 0xBC*n*04720

Access CPRC Write Only



FIELD NAME	BIT POSITION	DESCRIPTION
PoolNum	14:10	**Pool Number** — Pool number from which to deallocate BTags.
LineAddr	9:0	**Line Address** — Block address in DMEM to allocated.

Table 184 WrCB1_BTag_Deallocate Register (for Control Block1)

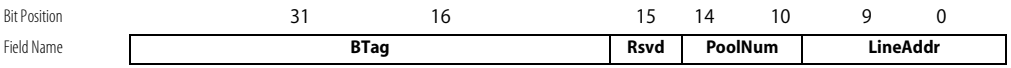
REGISTER NAME	PURPOSE	ADDRESS
WrCB1_BTag_Deallocate	Same as *WrCB0_BTag_Deallocate*, except for control block1.	0xBC*n*04740

WrCB0_MUC_Allocate (CP Wr Control Block0 Fixed Function)

Purpose Launches a Multi-Use Counter allocate command with a single store. See Table 185 on page 572 for similar register.

Address 0xBC*n*04724

Access CPRC Write Only



FIELD NAME	BIT POSITION	DESCRIPTION
BTag	31:16	**BTag** — BTag to associate with allocated counter.
Reserved	15	Read as zero.
PoolNum	14:10	**Pool Number** — Pool number from which to deallocate BTags.
LineAddr	9:0	**Line Address** — Block address in DMEM for allocated BTags.

Table 185 WrCB1_MUC_Allocate Register (for Control Block1)

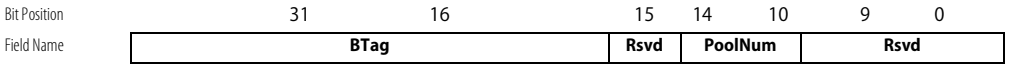
REGISTER NAME	PURPOSE	ADDRESS
WrCB1_MUC_Allocate	Same as *WrCB0_MUC_Allocate*, except for control block1.	0xBC*n*04744

WrCB0_MUC_Decrement (CP Wr Control Block0 Fixed Function)

Purpose Launches a Multi-Use Counter decrement command with a single store. See Table 186 on page 572 for similar register.

Address 0xBC*n*04728

Access CPRC Write Only



FIELD NAME	BIT POSITION	DESCRIPTION
BTag	31:16	**BTag** — BTag associated with counter to decrement.
Reserved	15	Read as zero.
PoolNum	14:10	**Pool Number** — Pool number associated with counter to decrement.
Reserved	9:0	Read as zero.

Table 186 WrCB1_MUC_Decrement Register (for Control Block1)

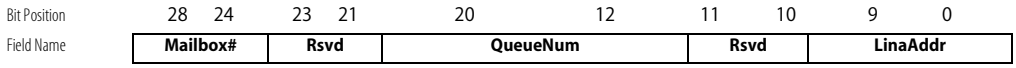
REGISTER NAME	PURPOSE	ADDRESS
WrCB1_MUC_Decrement	Same as *WrCB0_MUC_Decrement*, except for control block1.	0xBC*n*04748

WrCB0_Uni_Enq (CP Wr Control Block0 Fixed Function)

Purpose Launches a QMU Unicast Enqueue command with a single store. Waits for QMU write mailbox to be ready before launching control block transfer. See Table 187 on page 573 for similar register.

Address 0xBC*n*04730

Access CPRC Write Only



FIELD NAME	BIT POSITION	DESCRIPTION
Mailbox#	28:24	**Mailbox Number** — QMU mailbox number, generally the CP_ID.
Reserved	23:21	Read as zero.
QueueNum	20:12	**Queue Number** — Queue to enqueue.
Reserved	11:10	Read as zero.
LineAddr	9:0	**Line Address** — Block address in DMEM for descriptor data.

Table 187 WrCB1_Uni_Enq Register (for Control Block1)

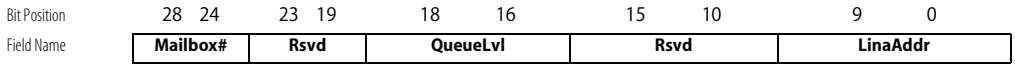
REGISTER NAME	PURPOSE	ADDRESS
WrCB1_Uni_Enq	Same as *WrCB0_Uni_Enq*, except for control block1.	0xBC*n*04750

WrCB0_Multi_Enq (CP Wr Control Block0 Fixed Function)

Purpose — Launches a QMU Multicast Enqueue command with a single store. Waits for QMU write mailbox to be ready before launching control block transfer. See Table 188 on page 574 for similar register.

Address — 0xBC*n*04734

Access — CPRC Write Only



FIELD NAME	BIT POSITION	DESCRIPTION
Mailbox#	28:24	**Mailbox Number** — QMU mailbox number, generally the CP_ID.
Reserved	23:19	Read as zero.
QueueLvl	18:16	**Queue Level** — Queue level to map through multicast enqueue table.
Reserved	15:10	Read as zero.
LineAddr	9:0	**Line Address** — Block address in DMEM for descriptor data.

Table 188 WrCB1_Multi_Enq Register (for Control Block1)

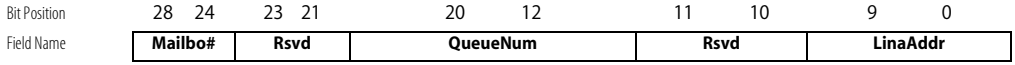
REGISTER NAME	PURPOSE	ADDRESS
WrCB1_Multi_Enq	Same as *WrCB0_Multi_Enq*, except for control block1.	0xBC*n*04754

WrCB0_Spec_Uni_Enq (CP Wr Control Block0 Fixed Function)

Purpose Launches a Speculative QMU Unicast Enqueue command with a single store. Waits for QMU write mailbox to be ready before launching control block transfer. See Table 189 on page 575 for similar register.

Address 0xBC*n*04738

Access CPRC Write Only



FIELD NAME	BIT POSITION	DESCRIPTION
Mailbox#	28:24	**Mailbox Number** — QMU mailbox number, generally the CP_ID.
Reserved	23:21	Read as zero.
QueueNum	20:12	**Queue Number** — Queue to enqueue.
Reserved	11:10	Read as zero.
LineAddr	9:0	**Line Address** — Block address in DMEM for descriptor data.

Table 189 WrCB1_Spec_Uni_Enq Register (for Control Block1)

REGISTER NAME	PURPOSE	ADDRESS
WrCB1_Spec_Uni_ _Enq	Same as *WrCB0_SPec_Uni_Enq*, except for control block1.	0xBC*n*04758

Executive Processor (XP) Configuration Registers

Configuration Space in the XP is an area that contains a number of registers. The XPRC uses these registers to communicate with the host and the bus controllers (Payload Bus and Global Bus). The XP's registers can also be accessed by other components of the C-5e NP (all CPs).

XPSlot 24 Configuration Registers

The following is a list of each XP Slot 24 register along with its address, function, and reference to its detailed parameters. The detailed parameters provide, purpose, field name, bit position, and descriptions. Refer to Table 190 on page 576.

Table 190 XP Registers

ADDRESS	REGISTER NAME	FUNCTION	DETAILED PARAMETERS
0xBD808000	PCI Device ID	PCI Configuration	See page 588
0xBD808002	PCI Vendor ID		See page 589
0xBD808004	PCI Status		See page 589
0xBD808006	PCI Command		See page 591
0xBD808008	PCI Class Code		See page 592
0xBD80800B	PCI Revision ID		See page 593
0xBD80800D	PCI Header Type		See page 594
0xBD80800E	PCI Latency Timer		See page 594
0xBD808010	PCI Inbound Memory Base Address0		See page 595
0xBD808014	PCI Inbound Memory Base Address1		See Table 193 on page 595.
0xBD808018	PCI Inbound Memory Base Address2		See page 596
0xBD80801C	PCI Inbound Memory Base Address3		See Table 194 on page 596.
0xBD808020	PCI Inbound Memory Base Address4		
0xBD808024	PCI Inbound Memory Base Address5		
0xBD80802C	PCI Subsystem ID (Read Only)		See page 597
0xBD80802E	PCI Subsystem Vendor ID (Read Only)		See page 597

Table 190 XP Registers (continued)

ADDRESS	REGISTER NAME	FUNCTION	DETAILED PARAMETERS
0xBD80803E	PCI Interrupt Pin	PCI Configuration (continued)	See page 597
0xBD80803F	PCI Interrupt Line		See page 597
0xBD808040	PCI Inbound BAR0 Translation		See page 598
0xBD808044	PCI Inbound BAR1 Translation		See page 598
0xBD808048	PCI Auxiliary Control		See page 599
0xBD80804C	PCI Subsystem ID (Read/Write)		See page 600
0xBD80804E	PCI Subsystem Vendor ID (Read/Write)		See page 600
0xBD808050	PCI Inbound Byte Swap Control		See page 600
0xBD808054	PCI Inbound BAR2 Translation		See page 601
0xBD808058	PCI Inbound BAR3 Translation		See Table 195 on page 602.
0xBD80805C	PCI Inbound BAR4 Translation		
0xBD808060	PCI Inbound BAR5 Translation		
0xBD808100	Serial Bus Configuration	XP Miscellaneous Control	See page 602
0xBD808104	Serial Bus Data		See page 603
0xBD808108	XP to CP Interrupt Request		See page 604
0xBD80810C	Software Warm Reset Request		See page 605
0xBD808200	Outbound PCI Base Address0	XP Configuration	See page 606
0xBD808204	Outbound PCI Base Address1		
0xBD808208	Outbound PCI Base Address2		
0xBD80820C	Outbound PCI Base Address3		
0xBD808210	Outbound PCI Base Address4		
0xBD808214	Outbound PCI Base Address5		
0xBD808218	Outbound PCI Base Address6		
0xBD80821C	Outbound PCI Base Address7		

Table 190 XP Registers (continued)

ADDRESS	REGISTER NAME	FUNCTION	DETAILED PARAMETERS
0xBD808220	Outbound BAR0 Translation	XP Configuration (continued)	See page 607
0xBD808224	Outbound BAR1 Translation		
0xBD808228	Outbound BAR2 Translation		
0xBD80822C	Outbound BAR3 Translation		
0xBD808230	Outbound BAR4 Translation		
0xBD808234	Outbound BAR5 Translation		
0xBD808238	Outbound BAR6 Translation		
0xBD80823C	Outbound BAR7 Translation		
0xBD808240	DMA Transmit Channel0 PCI Target		See page 608
0xBD808244	DMA Transmit Channel1 PCI Target		See Table 198 on page 609
0xBD808248	DMA Receive Channel0 PCI Target		See page 609.
0xBD80824C	DMA Receive Channel0 PCI Target Count		See page 609
0xBD808250	DMA Receive Channel1 PCI Target		See Table 199 on page 609
0xBD808254	DMA Receive Channel1 PCI Target Count		See Table 200 on page 610
0xBD808258	XP Miscellaneous Control		See page 611.
0xBD80825C	XP Auxiliary Event		See page 612
0xBD808260	Inbound PCI Mailbox0		See page 613
0xBD808264	Inbound PCI Mailbox1		
0xBD808268	Inbound PCI Mailbox2		
0xBD80826C	Inbound PCI Mailbox3		
0xBD808270	Inbound PCI Mailbox4		
0xBD808274	Inbound PCI Mailbox5		
0xBD808278	Inbound PCI Mailbox6		
0xBD80827C	Inbound PCI Mailbox7		

Table 190 XP Registers (continued)

ADDRESS	REGISTER NAME	FUNCTION	DETAILED PARAMETERS
0xBD808280	IMEM Overlay Target Address	XP Configuration (continued)	See page 614
0xBD808284	RxCB #25 Transfer Count		See page 614
0xBD808288	XP Diagnostic		See page 615
0xBD80828C	PCI Outbound Byte Swap Control		See page 616
0xBD808300	Debug Counter0 Start Value		See page 617
0xBD808304	Debug Counter1 Start Value		See Table 203 on page 620
0xBD808308	Debug Counter2 Start Value		
0xBD80830C	Debug Counter3 Start Value		
0xBD808340	Debug Counter0 Control		See page 618
0xBD808344	Debug Counter1 Control		See Table 203 on page 620
0xBD808348	Debug Counter2 Control		
0xBD80834C	Debug Counter3 Control		
0xBD808380	Debug Counter0 Current Value		See page 620
0xBD808384	Debug Counter1 Current Value		See Table 204 on page 620
0xBD808388	Debug Counter2 Current Value		
0xBD80838C	Debug Counter3 Current Value		
0xBD804080	RxCB0_Sys_Addr	XP DMEM#24 Transfer RxControl Block0 These registers are used to set up a DMA transaction from the PCI bus to SDRAM via DMEM#24. The fields of these registers are identical to their counterparts in the CP.	See "CP Registers" on page 486
0xBD804084	RxCB0_Ctl		
0xBD804088	RxCB0_DMA_Addr		
0xBD80408C	RxCB0_SDP_Addr		
0xBD804090	RxCtl0_Status	XP DMEM#24 Transfer RxControl Block0 (continued)	See page 621

Table 190 XP Registers (continued)

ADDRESS	REGISTER NAME	FUNCTION	DETAILED PARAMETERS
0xBD804180	TxCB0_Sys_Addr	XP DMEM#24 Transfer TxControl Block0 These registers are used to set up a DMA transaction from the SDRAM to the PCI bus via DMEM#24. The fields of these registers are identical to their counterparts in the CP.	See "CP Registers" on page 486 and page 622
0xBD804184	TxCB0_Ctl		
0xBD804188	TxCB0_DMA_Addr		
0xBD80418C	TxCB0_SDP_Addr		
0xBD804190	TxCtl0_Status	XP DMEM#24 Transfer TxControl Block0 (continued)	See page 622
0xBD804280	RxCB1_Sys_Addr	XP DMEM#24 Transfer RxControl Block1 These registers are used to set up a DMA transaction from the PCI bus to SDRAM via DMEM#24. The fields of these registers are identical to their counterparts in the CP.	See "CP Registers" on page 486
0xBD804284	RxCB1_Ctl		
0xBD804288	RxCB1_DMA_Addr		
0xBD80428C	RxCB1_SDP_Addr		
0xBD804290	RxCtl1_Status	XP DMEM#24 Transfer RxControl Block1 (continued)	See Table 205 on page 621

Table 190 XP Registers (continued)

ADDRESS	REGISTER NAME	FUNCTION	DETAILED PARAMETERS
0xBD804380	TxCB1_Sys_Addr	XP DMEM#24 Transfer TxControl Block1 These registers are used to set up a DMA transaction from the SDRAM to the PCI bus via DMEM#24. The fields of these registers are identical to their counterparts in the CP.	See "CP Registers" on page 486 and Table 206 on page 622
0xBD804384	TxCB1_Ctl		
0xBD804388	TxCB1_DMA_Addr		
0xBD80438C	TxCB1_SDP_Addr		
0xBD804390	TxCB1_Status	XP DMEM#24 Transfer TxControl Block1 (continued)	See Table 207 on page 623
0xBD804400	WrCB0_Sys_Addr	XP DMEM#24 Transfer WrControl Block0 (continued) These registers are identical to their counterparts in the CP except their addresses are different	See "CP Registers" on page 486
0xBD804404	WrCB0_Ctl		
0xBD804408	WrCB0_DMA_Addr		
0xBD804410	WrCB1_Sys_Addr	XP DMEM#24 Transfer WrControl Block1 (continued) These registers are identical to their counterparts in the CP except their addresses are different	See "CP Registers" on page 486
0xBD804414	WrCB1_Ctl		
0xBD804418	WrCB1_DMA_Addr		
0xBD804420	RdCB0_Sys_Addr	XP DMEM#24 Transfer RdControl Block0 (continued)	See "CP Registers" on page 486
0xBD804424	RdCB0_Ctl		
0xBD804428	RdCB0_DMA_Addr		

Table 190 XP Registers (continued)

ADDRESS	REGISTER NAME	FUNCTION	DETAILED PARAMETERS
0xBD804430	RdCB1_Sys_Addr	XP DMEM#24 Transfer RdControl Block1 (continued)	See "CP Registers" on page 486
0xBD804434	RdCB1_Ctl		
0xBD804438	RdCB1_DMA_Addr		
0xBD804440 to 0xBD8044E4	XP Ring Bus Control Configuration Registers	These registers are identical to their counterparts in the CP except their addresses are different.	See "CP Registers" on page 486
0xBD804640	XP_Mode	XP Mode Configuration	See page 623
0xBD804658	Debug_Mode		See page 625
0xBD804660	Queue_Status0	Queue Status These registers are identical to their counterparts in the CP except their addresses are different.	See "CP Registers" on page 486
0xBD804664	Queue_Status1		
0xBD804668	Queue_Status2		
0xBD80466C	Queue_Status3		
0xBD804670	Queue_Update0		
0xBD804674	Queue_Update1		
0xBD804678	Queue_Update2		
0xBD80467C	Queue_Update3		
0xBD804684	Event_Timer	Miscellaneous Control These registers are identical to their counterparts in the CP except their addresses are different.	See "CP Registers" on page 486
0xBD804688	Cycle_Counter_H		
0xBD80468C	Cycle_Counter_L		
0xBD8046A0	Event0	Event and Interrupt Control	See page 627
0xBD8046A4	Event1		See page 629

Table 190 XP Registers (continued)

ADDRESS	REGISTER NAME	FUNCTION	DETAILED PARAMETERS
0xBD8046A8	Event_Mask0	Event and Interrupt Control (continued) These registers are identical to their counterparts in the CP except their addresses are different.	See "CP Registers" on page 486
0xBD8046AC	Event_Mask1		
0xBD8046B0	Event_Access		
0xBD8046B4	Mask_Access		
0xBD8046B8	Interrupt_Mask0		
0xBD8046BC	Interrupt_Mask1		
0xBD804700	RdCB0_BTag_Alloc	XP DMEM#24 Rd Control Block0 Fixed These registers are identical to their counterparts in the CP except their addresses are different.	See "CP Registers" on page 486
0xBD804704	RdCB0_Dequeue		
0xBD804710	RdCB1_BTag_Alloc	XP DMEM#24 Rd Control Block1 Fixed These registers are identical to their counterparts in the CP except their addresses are different.	See "CP Registers" on page 486
0xBD804714	RdCB1_Dequeue		
0xBD804720	WrCB0_BTag_Dealloc	XP DMEM#24 Wr Control Block0 Fixed These registers are identical to their counterparts in the CP except their addresses are different.	See "CP Registers" on page 486
0xBD804724	WrCB0_MUC_Alloc		
0xBD804728	WrCB0_MUC_Decr		
0xBD804730	WrCB0_Uni_Enq		
0xBD804734	WrCB0_Multi_Enq		
0xBD804738	WrCB0_Spec_Uni_Enq		

Table 190 XP Registers (continued)

ADDRESS	REGISTER NAME	FUNCTION	DETAILED PARAMETERS
0xBD804740	WrCB1_BTag_Dealloc	XP DMEM#24 Wr Control Block1 Fixed These registers are identical to their counterparts in the CP except their addresses are different.	See "CP Registers" on page 486
0xBD804744	WrCB1_MUC_Alloc		
0xBD804748	WrCB1_MUC_Decr		
0xBD804750	WrCB1_Uni_Enq		
0xBD804754	WrCB1_Multi_Enq		
0xBD804758	WrCB1_Spec_Uni_Enq		
0xBD804880	RxCB0_Sys_Addr	XP DMEM#25 Transfer RxControl Block0 (continued) Theses registers are used to initialize SDRAM. The fields of these registers are identical to their counterparts in the CP.	See "CP Registers" on page 486
0xBD804884	RxCB0_Ctl		
0xBD804888	RxCB0_DMA_Addr		
0xBD80488C	RxCB0_SDP_Addr		
0xBD804890	RxCtl0_Status	XP DMEM#25 Transfer RxControl Block0 (continued)	See page 631
0xBD804980	TxCB0_Sys_Addr	XP DMEM#25 Transfer TxControl Block0 (continued) These registers are used to set up a DMA transaction from the SDRAM to the XP's IMEM. The fields of these registers are identical to their counterparts in the CP.	See "CP Registers" on page 486
0xBD804984	TxCB0_Ctl		
0xBD804988	TxCB0_DMA_Addr		
0xBD80498C	TxCB0_SDP_Addr		
0xBD804990	TxCtl0_Status	XP DMEM#25 Transfer Control Block0 (continued)	See page 632

Table 190 XP Registers (continued)

ADDRESS	REGISTER NAME	FUNCTION	DETAILED PARAMETERS
0xBD804A80	RxCB1_Sys_Addr	XP DMEM#25 Transfer Control Block1 (continued) Theses registers are used to initialize SDRAM. The fields of these registers are identical to their counterparts in the CP.	See "CP Registers" on page 486
0xBD804284	RxCB1_Ctl		
0xBD804A88	RxCB1_DMA_Addr		
0xBD804A8C	RxCB1_SDP_Addr		
0xBD80A290	RxCB1_Status	XP DMEM#25 Transfer Control Block1 (continued)	See page 632
0xBD804B80	TxCB1_Sys_Addr	XP DMEM#25 Transfer TxControl Block1 (continued) These registers are used to set up a DMA transaction from the SDRAM to the XP's IMEM. The fields of these registers are identical to their counterparts in the CP.	See "CP Registers" on page 486
0xBD804B84	TxCB1_Ctl		
0xBD804B88	TxCB1_DMA_Addr	XP DMEM#25 Transfer TxControl Block1 (continued) These registers are used to set up a DMA transaction from the SDRAM to the XP's IMEM. The fields of these registers are identical to their counterparts in the CP.	See "CP Registers" on page 486
0xBD804B8C	TxCB1_SDP_Addr		
0xBD804B90	TxCB1_Status	XP DMEM#25 Transfer Control Block1 (continued)	See page 632

Table 190 XP Registers (continued)

ADDRESS	REGISTER NAME	FUNCTION	DETAILED PARAMETERS
0xBD804C00	WrCB0_Sys_Addr	XP DMEM#25 Transfer Control Block0 (continued) These registers are identical to their counterparts in the CP except their addresses are different	See "CP Registers" on page 486
0xBD804C04	WrCB0_Ctl		
0xBD804C08	WrCB0_DMA_Addr		
0xBD804C10	WrCB1_Sys_Addr	XP DMEM#25 Transfer Control Block1 (continued) These registers are identical to their counterparts in the CP except their addresses are different	See "CP Registers" on page 486
0xBD804C14	WrCB1_Ctl		
0xBD804C18	WrCB1_DMA_Addr		
0xBD804C20	RdCB0_Sys_Addr	XP DMEM#25 Transfer Control Block0 (continued)	See "CP Registers" on page 486
0xBD804C24	RdCB0_Ctl		
0xBD804C28	RdCB0_DMA_Addr		
0xBD804C30	RdCB1_Sys_Addr	XP DMEM#25 Transfer Control Block1 (continued)	See "CP Registers" on page 486
0xBD804C34	RdCB1_Ctl		
0xBD804C38	RdCB1_DMA_Addr		
0xBD804F00	RdCB0_BTag_Alloc	XP DMEM#25 Rd Control Block0 Fixed These registers are identical to their counterparts in the CP except their addresses are different.	See "CP Registers" on page 486
0xBD804F04	RdCB0_Dequeue		

Table 190 XP Registers (continued)

ADDRESS	REGISTER NAME	FUNCTION	DETAILED PARAMETERS
0xBD804F10	RdCB1_BTag_Alloc	XP DMEM#25 Rd Control Block1 Fixed These registers are identical to their counterparts in the CP except their addresses are different.	See "CP Registers" on page 486
0xBD804F14	RdCB1_Dequeue		
0xBD804F20	WrCB0_BTag_Dealloc	XP DMEM#25 Wr Control Block0 Fixed These registers are identical to their counterparts in the CP except their addresses are different.	See "CP Registers" on page 486
0xBD804F24	WrCB0_MUC_Alloc		
0xBD804F28	WrCB0_MUC_Decr		
0xBD804F30	WrCB0_Uni_Enq		
0xBD804F34	WrCB0_Multi_Enq		
0xBD804F38	WrCB0_Spec_Uni_Enq		
0xBD804F40	WrCB1_BTag_Dealloc	XP DMEM#25 Wr Control Block1 Fixed These registers are identical to their counterparts in the CP except their addresses are different.	See "CP Registers" on page 486
0xBD804F44	WrCB1_MUC_Alloc		
0xBD804F48	WrCB1_MUC_Decr		
0xBD804F50	WrCB1_Uni_Enq		
0xBD804F54	WrCB1_Multi_Enq		
0xBD804F58	WrCB1_Spec_Uni_Enq		

XP Detailed Descriptions

The following is a detailed description of each of the XPSlot 24 registers and their individual parameters. The detailed parameters provide: purpose, field name, bit positions and descriptions.

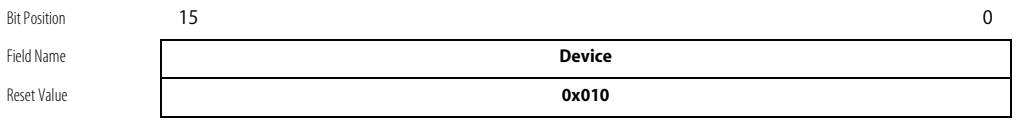
PCI Device ID Register (XP PCI Configuration Function)

Purpose Uniquely Identifies the Device.

Address 0xBD808000

Reset Vaule Refer to Table 191 on page 588.

Access Global Read



FIELD NAME	BIT POSITION	DESCRIPTION
Device ID	15:0	**Device ID** — Identifies the device.

Table 191 PCI Device ID (Reset Value)

NP REVISION	DEVICE ID (RESET VALUE) FOR 0XBD808000	REVISION ID (RESET VALUE) FOR 0XBD80800B
C-5 A0	0x1	0x10
C-5 A1		0x11
C-5 A2		0x12
C-5 B0		0x20
C-5 C0		0x30
C-5 D0		0x40
C-5e A0	0x2	0x10
C-5e A1		0x11
C-5e B0		0x20

Table 191 PCI Device ID (Reset Value)

NP REVISION	DEVICE ID (RESET VALUE) FOR 0XBD808000	REVISION ID (RESET VALUE) FOR 0XBD80800B
C-3e A0	0x3	0x10
C-3e A1		0x11
C-3e B0		0x20

PCI Vendor ID Register (XP PCI Configuration Function)

Purpose Uniquely Identifies the Device Vendor.

Address 0xBD808002

Access Global Read

Bit Position	15 ... 0
Field Name	Vendor
Reset Value	0x150E

FIELD NAME	BIT POSITION	DESCRIPTION
Vendor ID	15:0	**Vendor ID** — Identifies the vendor. This field is read-only.

PCI Status Register (XP PCI Configuration Function)

Purpose Captures Status Information for PCI bus related events.

Address 0xBD808004

Access Global Read/Write, write 1 to clear

Bit Position	15	14	13	12	11	10 ... 9	8	7	6	5	4	3 ... 0
Field Name	DPE	SSE	RMA	RTA	STA	DEVSEL	DPE	FBC	UDF	66M	NCP	Rsvd
Reset Value	0	0	0	0	0	01	0	1	0	1	0	0000

FIELD NAME	BIT POSITION	DESCRIPTION
DPE	15	**Detected Parity Error** — This bit is set by the device whenever it detects a parity error, even if parity error handling is disabled (as controlled by bit 6 in the Command Register).

FIELD NAME	BIT POSITION	DESCRIPTION
SSE	14	**Signaled System Error** — This bit is set whenever the device asserts *SERR#*.
RMA	13	**Received Master Abort** — This bit is set by the master whenever its transaction is terminated with a Master Abort.
RTA	12	**Received Target Abort** — This bit is set by the master whenever its transaction is terminated with a Target Abort.
STA	11	**Signaled Target Abort** — This bit is set by the target whenever it terminates a transaction wit a Target Abort.
DEVSEL	10:9	**DEVSEL Timing** — These two read-only bits are hardwired to "01" indicating the medium DEVSEL response time of this device.
DPE	8	**Data Parity Error Detected** — This bit is set when three conditions are met: 1) this device asserted *PERR#* itself or observed *PERR#* asserted; 2) this devices was acting as the bus master for the operation in which the error occurred; 3) the Parity Error Response bit (Command Register) is set.
FBC	7	**Fast Back-to-Back Capable** — This read-only bit is hardwired to 1 indicating that this device is capable of performing fast back-to-back transactions.
UDF	6	**UDF Supported** — This read-only bit is hardwired to 0 indicating that User Definable Features is not supported.
66M	5	**66MHz Capable** — This read-only bit is hardwired to 1 indicating that this device is capable of 66MHz operation.
NCP	4	**New Capabilities Pointer Support** — This read-only bit is hardwired to 0 indicating that there are no new capabilities pointers supported in the configuration register space.
Reserved	3:0	Read as zero.

PCI Command Register (XP PCI Configuration Function)

Purpose Provides control over the device's ability to generate and respond to PCI transactions.

Address 0xBD808006

Access Global Read/Write

Bit Position	15 10	9	8	7	6	5	4	3	2	1	0
Field Name	Rsvd	FB2B	SERR	WAIT	PERR	Rsvd	MWI	SPC	MST	MEM	IO
Reset Value	000000	0	0	0	0	0	0	0	0	0	0

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	15:10	Read as zero.
FB2B	9	**Fast Back-to-Back** — When the bit is 0, the PCI Master will generate no fast back-to-back transactions. When the bit is 1, the PCI Master will generate fast back-to-back transactions whenever possible.
SERR	8	**System Error** — When the bit is 0, the System Error indication is masked off. When the bit is 1, all System Errors will be reported.
WAIT	7	**Wait** — This bit is used to control whether or not a device does address/data stepping. This device never does address/data stepping, therefore this bit is hardwired to 0 and is read-only.
PERR	6	**Parity Error** — This bit controls the device's response to parity errors. When the bit is set to 1, the device will take its normal action when a parity error is detected. When the bit is 0, all parity errors are ignored.
Reserved	5	Read as zero.
MWI	4	**Memory Write and Invalidate** — This is an enable bit for using the Memory Write and Invalidate command. This device will never generate a MWI command, therefore this bit is hardwired to 0 and is read-only.
SPC	3	**Special Cycles** — This device always ignores special cycles, therefore this bit is hardwired to 0 and is read-only.
MST	2	**PCI Master** — This bit controls the device's ability to act as a master on the PCI bus. When the bit is 0, it disables the device from generating PCI accesses. When the bit is 1, it allows the device to behave as a bus master.

FIELD NAME	BIT POSITION	DESCRIPTION
MEM	1	**Memory Space** — This bit controls the device's response to Memory Space accesses. When the bit is 0, it disables the device's response. When the bit is 1, it allows the device to respond to Memory Space accesses
IO	0	**Input/Output Mapping** — This device requires no I/O spacing mappings, therefore this bit is hardwired to 0 and is read-only.

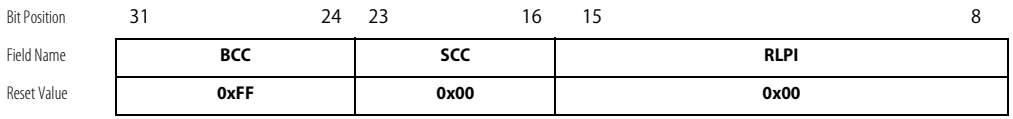
PCI Class Code Register (XP PCI Configuration Function)

Purpose Identifies the generic function of the device.

Address 0xBD808008

Reset Value 0xFF0000

Access Global Read



FIELD NAME	BIT POSITION	DESCRIPTION
BCC	31:24	**Base Class Code** — Hardwired to FFh indicating that the device does not fit in any of the PCI pre-defined classes.
SCC	23:16	**Sub-Class Code** — Hardwired to 00h. This field has no real meaning since the device has a base class code of FFh.
RLPI	15:8	**Register-Level Programming Interface** — Hardwired to 00h. This field has no real meaning since the device has a base class code of FFh

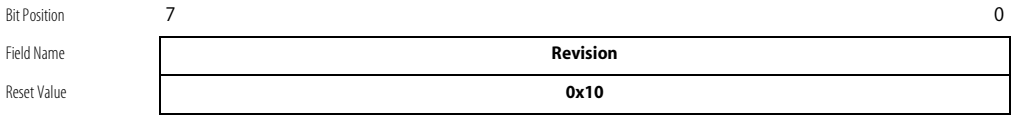
PCI Revision ID Register (XP PCI Configuration Function)

Purpose Provides a device specific revision identifier.

Address 0xBD80800B

Reset Value Refer to See Table 192 on page 593.

Access Global Read only



FIELD NAME	BIT POSITION	DESCRIPTION
Revision	7:0	**Revision** — Indicates the revision level of the device.

Table 192 PCI Revision ID (Reset Value)

NP REVISION	DEVICE ID (RESET VALUE) FOR 0XBD808000	REVISION ID (RESET VALUE) FOR 0XBD80800B
C-5 A0	0x1	0x10
C-5 A1		0x11
C-5 A2		0x12
C-5 B0		0x20
C-5 C0		0x30
C-5 D0		0x40
C-5e A0	0x2	0x10
C-5e A1		0x11
C-5e B0		0x20
C-3e A0	0x3	0x10
C-3e A1		0x11
C-3e B0		0x20

PCI Header Type Register (XP PCI Configuration Function)

Purpose Identifies the PCI Configuration Register layout.

Address 0xBD80800D

Reset Value 0x00

Access Global Read

PCI Latency Timer Register (XP PCI Configuration Function)

Purpose Limits the master's tenure on the bus in the presence of other bus access requests.

Address 0xBD80800E

Reset Value 0x00

Access Global Read/Write

Bit Position	7	3	2	0
Field Name	LAT		Rsvd	
Reset Value	00000		000	

FIELD NAME	BIT POSITION	DESCRIPTION
LAT	7:4	**Latency Timer Value** — The high order 5bits of an 8bit latency timer counting the number of PCI clock cycles that the master has tenure on the PCI bus.
Reserved	2:0	Read as zero.

PCI Inbound Memory Base Address0 Register (XP PCI Configuration Function)

Purpose Provides the Base Address for a 1MByte window into C-5e NP Address Space. See Table 193 on page 595 for similar register.

Address 0xBD808010

Reset Value 0x0008

Access Global Read/Write

Bit Position	31 20	19 4	3	2 1	0
Field Name	BA	Rsvd	PREF	TYPE	IO/M
Reset Value	0x000	0x0_000	1	00	0

FIELD NAME	BIT POSITION	DESCRIPTION
BA	31:20	**Base Address** — Provides the top 12bits of a 1MByte aligned address used for decoding and to identify PCI bus transactions for which this device is to act as a target.
Reserved	19:4	Read as zero.
PREF	3	**Prefetchable** — This read-only bit is hardwired to 1 indicating that there are no side effects on reads. The device returns all bytes on reads regardless of byte enables, and host bridges can merge processor writes into this range without causing errors.
TYPE	2:1	**Address Type** — These read-only bits are hardwired to 00 indicating that the base address can be set to locate this window anywhere in the 32bit PCI address space.
IO/M	0	**I/O or Memory Indicator** — This read-only bit is hardwired to 0 to indicate that this register is a memory space base address register.

Table 193 PCI Inbound Memory Base Address*n* Register (for Base Address1)

REGISTER NAME	PURPOSE	ADDRESS
PCI Inbound Memory Base Address1	Same as PCI Inbound Memory Base Address0, but for Base Address1.	0xBD808014

PCI Inbound Memory Base Address2 Register (XP PCI Configuration Function)

Purpose Provides the Base Address for a 4MByte window into C-5e NP Address Space. See Table 194 on page 596 for similar registers.

Address 0xBD808018

Reset Value 0x0008

Access Global Read/Write

Bit Position	31 22	21 4	3	2 1	0
Field Name	BA	Rsvd	PREF	TYPE	IO/M
Reset Value	0000_0000_00	00_0000_0000_0000_0000	1	00	0

FIELD NAME	BIT POSITION	DESCRIPTION
BA	31:22	**Base Address** — Provides the top 10bits of a 4MByte aligned address used for decoding, and to identify PCI bus transactions for which this device is to act as a target.
Reserved	21:4	Read as zero.
PREF	3	**Prefetchable** — This read-only bit is hardwired to 1 indicating that there are no side effects on reads. The device returns all bytes on reads regardless of byte enables, and host bridges can merge processor writes into this range without causing errors.
TYPE	2:1	**Address Type** — These read-only bits are hardwired to 00 indicating that the base address can be set to locate this window anywhere in the 32bit PCI address space.
IO/M	0	**I/O or Memory Indicator** — This read-only bit is hardwired to 0 to indicate that this register is a memory space base address register.

Table 194 PCI Inbound Memory Base Address*n* Register (for Base Address3, 4, and 5)

REGISTER NAME	PURPOSE	ADDRESS
PCI Inbound Memory Base Address3	Same as PCI Inbound Memory Base Address2, but for BAR3.	0xBD80801C
PCI Inbound Memory Base Address4	Same as PCI Inbound Memory Base Address2, but for BAR4.	0xBD808020
PCI Inbound Memory Base Address5	Same as PCI Inbound Memory Base Address2, but for BAR5.	0xBD808024

PCI Subsystem ID Register (Read Only) (XP PCI Configuration Function)

Purpose	Used to uniquely identify the subsystem where the PCI device resides.
Address	0xBD80802C
Reset Value	0x0000
Access	Global Read

PCI Subsystem Vendor ID Register (Read Only) (XP PCI Configuration Function)

Purpose	Used to uniquely identify the vendor of the subsystem where the PCI device resides.
Address	0xBD80802E
Reset Value	0x0000
Access	Global Read

PCI Interrupt Pin Register (XP PCI Configuration Function)

Purpose	Indicates that interrupt pin INTA# is the one being used.
Address	0xBD80803E
Reset Value	0x01
Access	Global Read/Write

PCI Interrupt Line Register (XP PCI Configuration Function)

Purpose	Used to communicate interrupt line routing information.
Address	0xBD80803F
Reset Value	0x00
Access	Global Read/Write

PCI Inbound BAR0 Translation Register (XP PCI Configuration Function)

Purpose Provides Address Translation and Control for the associated PCI Inbound Memory Base Address0 Register [Address: 10-13h].

Address 0xBD808040

Reset Value 0xA000

Access Global Read/Write

Bit Position	31 29	28 20	19 0
Field Name	101	TRANS	Rsvd
Reset Value	101	0_0000_0000	0000_0000_0000_0000_0000

FIELD NAME	BIT POSITION	DESCRIPTION
101	31:29	**"101" Translation Address** — Bits [31:29] of the transaction address are always translated to 101. This limits the translated address to the range of 0xA0000000 to 0xBFFFFFFF.
TRANS	28:20	**Translation Address** — Provides the replacement values for bits [28:20] of the transaction address to translate it to a different 1MByte window in C-5e NP Address Space.
Reserved	19:0	Read as zero.

PCI Inbound BAR1 Translation Register (XP PCI Configuration Function)

Purpose Provides Address Translation and Control for the associated PCI Inbound Memory Base Address1 Register [Address: 14-17h].

Address 0xBD808044

Reset Value 0xA000

Access Global Read/Write

Bit Position	31 29	28 20	19 0
Field Name	101	TRANS	Rsvd
Reset Value	101	0_0000_0000	0000_0000_0000_0000_0000

FIELD NAME	BIT POSITION	DESCRIPTION
101	31:29	**"101" Translation Address** — Bits [31:29] of the transaction address are always translated to 101. This limits the translated address to the range of 0xA0000000 to 0xBFFFFFFF.
TRANS	28:20	**Translation Address** — Provides the replacement values for bits [28:20] of the transaction address to translate it to a different 1MByte window in C-5e NP Address Space.
Reserved	19:0	Read as zero.

PCI Auxiliary Control Register (XP PCI Configuration Function)

Purpose Provides Control for Miscellaneous Functions within the PCI interface.

Address 0xBD808048

Reset Value 0x00

Access Global Read/Write

Bit Position	31 ... 3	2	1	0
Field Name	Rsvd	P2S	MA2S	TA2S
Reset Value	0000_0000_0000	0	0	0

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:3	Read as zero.
P2S	2	**Map Parity Errors to SERR#** — When the bit is 1, any PCI parity error that is detected and reported will result in the device signaling SERR#.
MA2S	1	**Map Master Abort to SERR#** — When the bit is 1, whenever the master has a transaction terminated by a Master Abort, the device will signal SERR#.
TA2S	0	**Map Target Abort to SERR#** — When the bit is 1, whenever the master has a transaction terminated by a Target Abort the device will signal SERR#.

PCI Subsystem ID Register (XP PCI Configuration Function)

Purpose Writable image of the PCI Subsystem ID.

Address 0xBD80804C

Reset Value 0x0000

Access Global Read/Write

PCI Subsystem Vendor ID Register (XP PCI Configuration Function)

Purpose Writable image of the PCI Subsystem Vendor ID.

Address 0xBD80804E

Reset Value 0x0000

Access Global Read/Write

PCI Inbound Byte Swap Control Register (XP PCI Configuration Function)

Purpose Provides control of the byte swapping feature for inbound transactions.

Address 0xBD808050

Reset Value 0x0000

Access Global Read/Write

Bit Position	31 … 6	5	4	3	2	1	0
Field Name	Rsvd	SWAP5	SWAP4	SWAP3	SWAP2	SWAP1	SWAP0
Reset Value	0000_0000_0000_0000_0000_0000_00	0	0	0	0	0	0

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:6	Read as zero.
SWAP5	5	**BAR5 Byte Swap Enable** — When this bit is set to a 1, byte swapping occurs as data passes through the PCI interface for any transaction decoded by *PCI Inbound Base Address5* register.
SWAP4	4	**BAR4 Byte Swap Enable** — When this bit is set to a 1, byte swapping occurs as data passes through the PCI interface for any transaction decoded by *PCI Inbound Base Address4* register.
SWAP3	3	**BAR3 Byte Swap Enable** — When this bit is set to a 1, byte swapping occurs as data passes through the PCI interface for any transaction decoded by *PCI Inbound Base Address3* register.

FIELD NAME	BIT POSITION	DESCRIPTION
SWAP2	2	**BAR2 Byte Swap Enable** — When this bit is set to a 1, byte swapping occurs as data passes through the PCI interface for any transaction decoded by *PCI Inbound Base Address2* register.
SWAP1	1	**BAR1 Byte Swap Enable** — When this bit is set to a 1, byte swapping occurs as data passes through the PCI interface for any transaction decoded by *PCI Inbound Base Address1* register.
SWAP0	0	**BAR0 Byte Swap Enable** — When this bit is set to a 1, byte swapping occurs as data passes through the PCI interface for any transaction decoded by *PCI Inbound Base Address0* register.

PCI Inbound BAR2 Translation Register (XP PCI Configuration Function)

Purpose — Provides Address Translation and Control for the associated PCI Inbound Memory Base Address2 Register [Address: 18-1Bh]. See Table 195 on page 602 for similar registers.

Address — 0xBD808054

Reset Value — 0xA000

Access — Global Read/Write

Bit Position	31 29	28 22	21 0
Field Name	101	TRANS	Rsvd
Reset Value	101	0_0000_00	00_0000_0000_0000_0000_0000

FIELD NAME	BIT POSITION	DESCRIPTION
101	31:29	**"101" Translation Address** — Bits [31:29] of the transaction address are always translated to 101. This limits the translated address to the range of 0xA0000000 to 0xBFFFFFFF.
TRANS	28:22	**Translation Address** — Provides the replacement values for bits [28: 22] of the transaction address to translate it to a different 4MByte window in C-5e NP Address Space.
Reserved	21:0	Read as zero.

Table 195 PCI Inbound BAR*n* Translation Register (for BAR3, 4 and 5)

REGISTER NAME	PURPOSE	ADDRESS
PCI Inbound BAR3 Translation	Same as PCI Inbound BAR2 translation, but for BAR3 and therefore, associated with the PCI Inbound Memory Base Address3 Register [Address: 1C-1Fh].	0xBD808058
PCI Inbound BAR4 Translation	Same as PCI Inbound BAR2 translation, but for BAR4 and therefore, associated with the PCI Inbound Memory Base Address4 Register [Address: 20-23h].	0xBD80805C
PCI Inbound BAR5 Translation	Same as PCI Inbound BAR2 translation, but for BAR5 and therefore, associated with the PCI Inbound Memory Base Address5 Register [Address: 24-27h].	0xBD808060

Serial Bus Configuration Register (XP Miscellaneous Control Function)

Purpose Sets the configuration of the Serial Bus.

Address 0xBD808100

Reset Value 0x000001F4

Access Global Read/Write

Bit Position	31 … 13	12	11	10	9	8 … 0
Field Name	Reserved	MDIO_Cycles	EN	PROTOCOL	PRE_SUPP	CLKDIV
Reset Value	0	0	0	0	0	1_1111_0100

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:13	Read as zero.
MDIO_Cycles	12	**MDIO Turn Around Cycles** — When deasserted, which is the default, enables one turnaround cycle. When asserted, enables two turnaround cycles on reads.
EN	11	**Serial Bus Enable** — This is an enable control bit for the serial bus. When the bit is 1, the bus is enabled.
PROTOCOL	10	**Protocol Select** — Determines which of 2 possible protocols will be followed by the C-5e NP on the serial bus. A value of 0 selects the MDIO protocol and a value of 1 selects the low-speed serial bus protocol.
PRE_SUPP	9	**MDIO Preamble Suppression** — When this bit is 1, the 32bit preamble pattern will be skipped during MDIO transfers.

FIELD NAME	BIT POSITION	DESCRIPTION
CLKDIV	8:0	**Core Clock Divider** — With the value *N* programmed into this 9bit field, the C-5e NP core clock will be divided by a factor of *4xN* to form the Serial bus clock. The default value of "*500*" causes the 166 MHz core clock to be divided by 4*500, generating a 83kHz Serial bus clock. Note: That when this *CLKDIV* field equals 0, the divider will be 512, resulting in a factor of 4*512=2048.

Serial Bus Data Register (XP Miscellaneous Control Function)

Purpose — Specifies the address, data, and write/read of a serial bus transfer. A transfer will be initiated when the REQ bit gets set.

Address — 0xBD808104

Reset Value — 0x00000000

Access — Global Read/Write

Bit Position	31	30 29	28	27 18	17	16	15 0
Field Name	REQ	Rsvd	WR	ADDR	AACK	DACK	DATA
Reset Value	0	00	0	00_0000_0000	0	0	0000_0000_0000_0000

FIELD NAME	BIT POSITION	DESCRIPTION
REQ	31	**Request** — Indicates that a serial bus transfer is pending. Software should set this bit to a 1 to begin a serial bus transfer. When the *REQ* bit is set, the transfer will be performed using the values of *WR* and *ADDR* in this register, and using the configuration from the Serial Bus Configuration Register. The *REQ* bit will remain a 1 until the transfer is done (also indicated by the *SB_TRANSFER_DONE* bit in XP *EVENT0*), after which it is cleared by the hardware. Writes to the serial bus data register are delayed until the serial bus is idle (and the *REQ* bit is deasserted). It is acceptable to set the *REQ* bit with the same register access that sets up the proper values of *WR*, *ADDR*, and *DATA* fields.
Reserved	30:29	Read as zero.
WR	28	**Write** — Specifies whether the serial bus transfer is a write or a read: 1 = write, 0 = read.

FIELD NAME	BIT POSITION	DESCRIPTION
ADDR	27:18	**Address** — Specifies the address of the serial bus transfer. In MDIO mode, bits [27:23] act as the PHY address, and bits [22:18] act as the register address. In low-speed serial bus mode, bits [24:18] make up the 7bit address (bits [27:25] are unused).
AACK	17	**Address Acknowledge** — This bit indicates the address acknowledge value captured on the serial bus. It is read-only (and is not affected by writes to this register) and applies only to the low-speed serial bus mode. When the bit is a 1, the previously attempted access failed due to a missing address acknowledgement.
DACK	16	**Data Acknowledge** — This bit indicates the data acknowledge value captured on the serial bus. It is read-only (and is not affected by writes to this register) and applies only to low-speed serial bus mode. When the bit is a 1, the previously attempted access failed due to a missing data acknowledgement.
DATA	15:0	**Data** — For writes to the serial bus, this field indicates the data to be written. For reads from the serial bus, this field contains the data that was read from the bus. The data is valid once the transfer is done, as indicated by the deassertion of the *REQ* bit and also by the *SB_TRANSFER_DONE* bit in XP *EVENT0*. Note: That bits [15:0] are valid for MDIO mode. For the low-speed serial bus mode, bits [15:8] are undefined and only bits [7:0] are valid.

XP to CP Interrupt Request Registers (XP Miscellaneous Control Function)

Purpose Initiates Interrupt Requests from the XP to individual CPs

Address 0xBD808108

Reset Value 0x00

Access Write only

Bit Position	31 … 19	18	17	16	15 … 0
Field Name	Rsvd	PCI_IRQ	Rsvd		XP2CP_IRQ
Reset Value	raz		raz		N/A

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:19	Read as zero.

FIELD NAME	BIT POSITION	DESCRIPTION
PCI_IRQ	18	**PCI Interrupt Request** — Provides a way for software to cause a PCI interrupt. When set to 1, the PCI Interrupt Line will be asserted.
Reserved	17:16	Read as zero.
XP2CP_IRQ	15:0	**XP to CP Interrupt Request Vector** — When the bit is 1, an interrupt request is sent to the CP corresponding to the bit number.

Software Warm Reset Request Register (XP Miscellaneous Control Function)

Purpose Trigger Reset Sequences for the C-5e NP and the XP.

Address 0xBD80810C

Reset Value 0x00

Access Write only

Bit Position	31	30	29	28 ... 0
Field Name	DCPRST	XPURST	WARM_XPUHOT	Rsvd
Reset Value	N/A	N/A	N/A	N/A

FIELD NAME	BIT POSITION	DESCRIPTION
NPRST	31	**NP Warm Reset Request** — When the bit is 1, a state machine is triggered that waits for PCI inbound and outbound activity to idle, asserts reset to the entire C-5e NP, waits for 128 C-5e NP core clock cycles, and then deasserts reset.
XPURST	30	**XP Warm Reset Request** — When the bit is 1, a state machine is triggered that waits for XP activity to idle, asserts reset to the XPRC, waits for 128 C-5e NP core clock cycles, and then deasserts reset.
WARM_XPUHOT	29	**Warm XPUHOT** — When a C-5e NP warm reset or XP warm reset is requested (using one of the above bits), this bit specifies whether the XP will be turned on (XPUHOT=1) or off (XPUHOT=0) after the warm reset completes. This only applies to warm resets; on cold resets, the XP will be turned on based on how the XPUHOT external pin is sampled.
Reserved	28:0	Read as zero

Outbound PCI Base Address0 Register (XP Configuration Function)

Purpose — Provides the Base Address for a variable size window for the XP into PCI Space. See Table 196 on page 606 for similar registers.

Address — 0xBD808200

Reset Value — 0x00

Access — Global Read/Write

Bit Position	31 — 14	13 — 0
Field Name	BA	Rsvd
Reset Value	0000_0000_0000_0000_00	00_0000_0000_0000

FIELD NAME	BIT POSITION	DESCRIPTION
BA	31:14	**Base Address** — Provides the high order address bytes used for decoding an access by the XP into a window into PCI Space. The number of bits used in the decode depends on the size specification for the window.
Reserved	13:0	Read as zero.

Table 196 Outbound PCI Base Address*n* Registers (for BAR 1, 2, 3, 4, 5, 6 and 7)

REGISTER NAME	PURPOSE	ADDRESS
Outbound PCI Base Address1	Same as *Outbound PCI Base Address0 Register*, except for BAR 1.	0xBD808204
Outbound PCI Base Address2	Same as *Outbound PCI Base Address0 Register*, except for BAR 2.	0xBD808208
Outbound PCI Base Address3	Same as *Outbound PCI Base Address0 Register*, except for BAR 3.	0xBD80820C
Outbound PCI Base Address4	Same as *Outbound PCI Base Address0 Register*, except for BAR 4.	0xBD808210
Outbound PCI Base Address5	Same as *Outbound PCI Base Address0 Register*, except for BAR 5.	0xBD808214
Outbound PCI Base Address6	Same as *Outbound PCI Base Address0 Register*, except for BAR 6.	0xBD808218
Outbound PCI Base Address7	Same as *Outbound PCI Base Address0 Register*, except for BAR 7.	0xBD80821C

Outbound BAR0 Translation Register (XP Configuration Function)

Purpose — Provides the Translation Address and control for a variable size window for the XP into the PCI Space. See Table 197 on page 608 for similar registers.

Address — 0xBD808220

Reset Value — 0x00

Access — Global Read/Write

Bit Position	31 — 14	13 — 4	3 — 1	0
Field Name	TRANS	Rsvd	SIZE	EN
Reset Value	0000_0000_0000_0000_00	raz	000	0

FIELD NAME	BIT POSITION	DESCRIPTION
TRANS	31:14	**Translation Address** — Provides the high order address bits to replace the high order address bits from the original address to create an address in PCI space. The number of bits replaced depends on the size specification for the window.
Reserved	13:4	Read as zero
SIZE	3:1	**Window Size** — Specifies the size of the address region viewed by the window. Note: That the base address and the translation address will be interpreted as being size aligned. ENCODED VALUE / WINDOW SIZE: 000 — 16 kB 001 — 32 kB 010 — 64 kB 011 — 128 kB 100 — 256 kB 101 — 512 kB 110 — 1 MB 111 — 2 MB

FIELD NAME	BIT POSITION	DESCRIPTION
EN	0	**BAR Enable** — This bit controls whether or not the corresponding BAR is used to decode XP accesses to PCI. When the bit is 1, the corresponding BAR is used in transaction decode. When the bit is 0, the BAR will be ignored.

Table 197 Outbound BAR*n* Translation Registers (for BAR1, 2, 3, 4, 5, 6 and 7)

REGISTER NAME	PURPOSE	ADDRESS
Outbound Bar1 Translation	Same as *Outbound BAR0 Translation Register*, except for BAR1.	0xBD808224
Outbound Bar2 Translation	Same as *Outbound BAR0 Translation Register*, except for BAR2.	0xBD808228
Outbound Bar3 Translation	Same as *Outbound BAR0 Translation Register*, except for BAR3.	0xBD80822C
Outbound Bar4 Translation	Same as *Outbound BAR0 Translation Register*, except for BAR4.	0xBD808230
Outbound Bar5 Translation	Same as *Outbound BAR0 Translation Register*, except for BAR5.	0xBD808234
Outbound Bar6 Translation	Same as *Outbound BAR0 Translation Register*, except for BAR6.	0xBD808238
Outbound Bar7 Translation	Same as *Outbound BAR0 Translation Register*, except for BAR7.	0xBD80823C

DMA Transmit Channel0 PCI Target Register (XP Configuration Function)

Purpose — Provides the PCI Target Address for the DMA Transmit Channel0. See Table 198 on page 609 for similar registers.

Address — 0xBD808240

Reset Value — 0x00

Access — Global Read/Write

Bit Position	31	4	3	0
Field Name	**ADDR**		**Rsvd**	
Reset Value	**0000_000**		**raz**	

FIELD NAME	BIT POSITION	DESCRIPTION
ADDR	31:4	**Target Address** — Provides a 16Byte aligned PCI address to start an outbound PCI write of data coming from Transmit Channel0.
Reserved	3:0	Read as zero.

Table 198 DMA Transmit Channel1 PCI Target Register (for Channel1)

REGISTER NAME	PURPOSE	ADDRESS
DMA Transmit Channel1 PCI Target	Same as *DMA Transmit Channel0 PCI Target*, except for channel1.	0xBD808244

DMA Receive Channel0 PCI Target Register (XP Configuration Function)

Purpose Provides the PCI Target Address for DMA Receive Channel0. See Table 199 on page 609 for similar registers.

Address 0xBD808248

Reset Value 0x00

Access Global Read/Write

Bit Position	31	4	3	0
Field Name	ADDR		Rsvd	
Reset Value	0000_000		raz	

FIELD NAME	BIT POSITION	DESCRIPTION
ADDR	31:4	**Target Address** — Provides a 16Byte aligned PCI address to start an outbound PCI read of data going to Receive Channel 0.
Reserved	3:0	Read as zero.

Table 199 DMA Receive Channel1 PCI Target Register (for Channel1)

REGISTER NAME	PURPOSE	ADDRESS
DMA Receive Channel1 PCI Target	Same as *DMA Receive Channel0 PCI Target*, except for channel1.	0xBD808250

DMA Receive Channel0 Transfer Count Register (XP Configuration Function)

Purpose Provides the transfer count for DMA Receive Channel0. See Table 200 on page 610 for similar register.

Address 0xBD80824C

Reset Value 0x0000

Access Global Read/Write

Bit Position	31 ... 14	13 ... 0
Field Name	Rsvd	COUNT
Reset Value	raz	00_0000_0000_0000

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:14	Read as zero.
COUNT	13:0	**Transfer Count** — Specifies the number of 4Byte transfers to be initiated on the PCI Bus when retrieving data for DMA Receive Channel 0. The transfer count legal range is 1 to 16k. The value of 16k is denoted by a programmed value of 0.

Table 200 DMA Receive Channel1 Transfer Count Register (for Channel1)

REGISTER NAME	PURPOSE	ADDRESS
DMA Receive Channel1 Transfer Count	Same as *DMA Receive Channel0 Transfer Count,* except for channel1.	0xBD808254

XP Miscellaneous Control Register (XP Configuration Function)

Purpose Provides control for the PROM and PCI interrupt line.

Address 0xBD808258

Reset Value 0x00

Access Global Read/Write

Bit Position	31 24	23	22	21	20	19 10	9	8 3	2 0
Field Name	Rsvd	PCI_IMSK3	PCI_IMSK2	PCI_IMSK1	PCI_IMSK0	Rsvd	ZBFP	Rsvd	PROMCLK
Reset Value	0	0	0	0	0	0	0	raz	000

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:24	Read as zero.
PCI_IMSK3	23	**PCI Interrupt Mask 3** — When the bit is 1, any interrupt intended for the XP on IRQ3 is redirected to the PCI Interrupt Line. When the bit is 0, the interrupt is directed to the XP.
PCI_IMSK2	22	**PCI Interrupt Mask 2** — When the bit is 1, any interrupt intended for the XP on IRQ2 is redirected to the PCI Interrupt Line. When the bit is 0, the interrupt is directed to the XP.
PCI_IMSK1	21	**PCI Interrupt Mask 1** — When the bit is 1, any interrupt intended for the XP on IRQ1 is redirected to the PCI Interrupt Line. When the bit is 0, the interrupt is directed to the XP.
PCI_IMSK0	20	**PCI Interrupt Mask 0** — When the bit is 1, any interrupt intended for the XP on IRQ0 is redirected to the PCI Interrupt Line. When the bit is 0, the interrupt is directed to the XP.
Reserved	19:10	Read as zero.
ZBFP	9	**Payload Bus Arbiter FP More Slots** — When the bit is 1, the FP is given more slots on the Payload Bus.
Reserved	8:3	Read as zero.
PROMCLK	2:0	**PROM Clock Divider** — Specifies the clock divider applied to the core clock to generate the PROM Interface serial clock. The default is zero which, sets the clock divider to 16. All other values result in a clock divider that is 2 times the value.

XP Auxiliary Event Register (XP Configuration Function)

Purpose Contains flags for CP Interrupts and PCI Mailbox Interrupts.

Address 0xBD80825C

Reset Value 0x00

Access Global Read/Write, write 1 clear

Bit Position	31	30	29	28	27	26	25	24	23 19	18	17	16	15 0
Field Name	R7	R6	R5	R4	R3	R2	R1	R0	Rsvd	PCI_INT	FPTx_INT	FPRX_INT	CP_INT
Reset Value	0	0	0	0	0	0	0	0	raz	0	0	0	0000

FIELD NAME	BIT POSITION	DESCRIPTION
R7	31	**Mailbox Register 7 Status Bit** — This bit is set by hardware when an inbound PCI transaction writes to *Inbound PCI Mailbox Register 7*, and is reset by hardware when the XP reads from *Inbound Mailbox Register 7*. This bit can also be reset by software by writing a 1 to the bit position.
R6	30	**Mailbox Register 6 Status Bit** — Same as R7 but pertains to *Inbound PCI Mailbox Register 6*.
R5	29	**Mailbox Register 5 Status Bit** — Same as R7 but pertains to *Inbound PCI Mailbox Register 5*.
R4	28	**Mailbox Register 4 Status Bit** — Same as R7 but pertains to *Inbound PCI Mailbox Register 4*.
R3	27	**Mailbox Register 3 Status Bit** — Same as R7 but pertains to *Inbound PCI Mailbox Register 3*.
R2	26	**Mailbox Register 2 Status Bit** — Same as R7 but pertains to *Inbound PCI Mailbox Register 2*.
R1	25	**Mailbox Register 1 Status Bit** — Same as R7 but pertains to *Inbound PCI Mailbox Register 1*.
R0	24	**Mailbox Register 0 Status Bit** — Same as R7 but pertains to *Inbound PCI Mailbox Register 0*.
Reserved	23:19	Read as zero.
PCI_INT	18	**PCI Interrupt** — Indicates that a PCI interrupt is active. (A PCI interrupt request can be made by setting a bit in the *XP To CP Interrupt Request* Register.)

FIELD NAME	BIT POSITION	DESCRIPTION
FPTx_INT	17	**TxFP Interrupt Status Bits** — When read as 1, indicates that an interrupt request was made by the TxFP unit.
FPRx_INT	16	**RxFP Interrupt Status Bits** — When read as 1, indicates that an interrupt request was made by the RxFP unit.
CP_INT	15:0	**CP Interrupt Status Bits** — When read as 1, indicates that an interrupt request was made by the CP corresponding to the bit number.

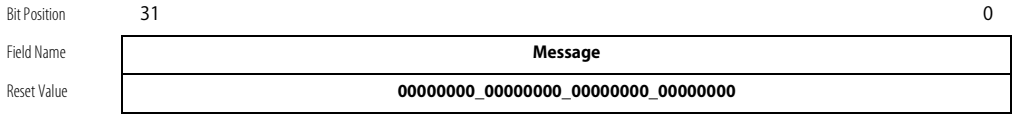
Inbound PCI Mailbox0 Register (XP Configuration Function)

Purpose Inbound PCI mailbox registers. See Table 201 on page 613 for similar registers.

Address 0xBD808260

Reset Value 0x0000

Access Global Read/Write



FIELD NAME	BIT POSITION	DESCRIPTION
Message	31:0	**Mailbox Message**

Table 201 Inbound PCI Mailbox*n* Registers (for Mailbox 1, 2, 3, 4, 5, 6 and 7)

REGISTER NAME	PURPOSE	ADDRESS
Inbound PCI Mailbox1	Same as *Inbound PCI Mailbox0 Register*, except for mailbox 1.	0xBD808264
Inbound PCI Mailbox2	Same as *Inbound PCI Mailbox0 Register*, except for mailbox 2.	0xBD808268
Inbound PCI Mailbox3	Same as *Inbound PCI Mailbox0 Register*, except for mailbox 3.	0xBD80826C
Inbound PCI Mailbox4	Same as *Inbound PCI Mailbox0 Register*, except for mailbox 4.	0xBD808270

Table 201 Inbound PCI Mailbox*n* Registers (for Mailbox 1, 2, 3, 4, 5, 6 and 7) (continued)

REGISTER NAME	PURPOSE	ADDRESS
Inbound PCI Mailbox5	Same as *Inbound PCI Mailbox0 Register*, except for mailbox 5.	0xBD808274
Inbound PCI Mailbox6	Same as *Inbound PCI Mailbox0 Register*, except for mailbox6.	0xBD808278
Inbound PCI Mailbox7	Same as *Inbound PCI Mailbox0 Register*, except for mailbox7.	0xBD80827C

IMEM Overlay Target Address Register (XP Configuration Function)

Purpose IMEM Overlay Target Address Register.

Address 0xBD808280

Access Global Read/Write

Bit Position	31 … 18	17 … 16	15 … 2	1 … 0
Field Name	IMEM ADDR0	Rsvd	IMEM ADDR1	Rsvd
Reset Value	0000_0000_0000_00	raz	0000_0000_0000_00	raz

FIELD NAME	BIT POSITION	DESCRIPTION
IMEM ADDR0	31:18	**IMEM ADDR0** – This is the target address into the IMEM that is used during transfers driven by DataScope0 of the TxCB#25.
Reserved	17:16	Read as zero.
IMEM ADDR1	15:2	**IMEM ADDR1** – This is the target address into the IMEM that is used during transfers driven by DataScope1 of the TxCB#25.
Reserved	1:0	Read as zero.

RxCB #25 Transfer Count Register (XP Configuration Function)

Purpose RxCB #25 Transfer Count Register.

Address 0xBD808284

Reset Value 0x0

Access Global Read/Write

Bit Position	31	0
Field Name	**Address**	
Reset Value	**0x00000000**	

XP Diagnostic Register (XP Configuration Function)

Purpose Retains data through warm reset. Used for diagnostic purposes.

Address 0xBD808288

Reset Value 0x0 (hard reset only; retains data on warm reset)

Access Global Read/Write

Bit Position	31	0
Field Name	**Address**	
Reset Value	**0x00000000**	

PCI Outbound Byte Swap Control Register (XP Configuration Function)

Purpose Provides control of the byte swapping feature for outbound transactions.

Address 0xBD80828C

Reset Value 0x0000

Access Global Read/Write

Bit Position	31 12	11	10	9	8	7	6	5	4	3	2	1	0
Field Name	Rsvd	TX1	TX0	RX1	RX0	BAR7	BAR6	BAR4	BAR4	BAR3	BAR2	BAR1	BAR0
Reset Value	0x00000	0	0	0	0	0	0	0	0	0	0	0	0

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:12	Read as zero.
TX1	11	**TX1 Byte Swap Enable** — When set to 1, byte swapping occurs as data passes through the PCI interface for any transaction associated with *DMA Transmit Channel 1* register.
TX0	10	**TX0 Byte Swap Enable** — When set to 1, byte swapping occurs as data passes through the PCI interface for any transaction associated with *DMA Transmit Channel 0* register.
RX1	9	**RX1 Byte Swap Enable** — When set to 1, byte swapping occurs as data passes through the PCI interface for any transaction associated with *DMA Receive Channel 1* register.
RX0	8	**RX0 Byte Swap Enable** — When set to 1, byte swapping occurs as data passes through the PCI interface for any transaction associated with *DMA Receive Channel 0* register.
BAR7	7	**BAR7 Byte Swap Enable** — When set to 1, byte swapping occurs as data passes through the PCI interface for any transaction decoded by *PCI Outbound Base Address 7* register.
BAR6	6	**BAR6 Byte Swap Enable** — When set to1, byte swapping occurs as data passes through the PCI interface for any transaction decoded by *PCI Outbound Base Address 6* register.
BAR5	5	**BAR5 Byte Swap Enable** — When set to 1, byte swapping occurs as data passes through the PCI interface for any transaction decoded by *PCI Outbound Base Address 5* register.
BAR4	4	**BAR4 Byte Swap Enable** — When set to 1, byte swapping occurs as data passes through the PCI interface for any transaction decoded by *PCI Outbound Base Address 4* register.

FIELD NAME	BIT POSITION	DESCRIPTION
BAR3	3	**BAR3 Byte Swap Enable** — When set to 1, byte swapping occurs as data passes through the PCI interface for any transaction decoded by *PCI Outbound Base Address 3* register.
BAR2	2	**BAR2 Byte Swap Enable** — When set to 1, byte swapping occurs as data passes through the PCI interface for any transaction decoded by *PCI Outbound Base Address 2* register.
BAR1	1	**BAR1 Byte Swap Enable** — When set to 1, byte swapping occurs as data passes through the PCI interface for any transaction decoded by *PCI Outbound Base Address 1* register.
BAR0	0	**BAR0 Byte Swap Enable** — When set to 1, byte swapping occurs as data passes through the PCI interface for any transaction decoded by *PCI Outbound Base Address 0* register.

Debug Counter0 Start Value Register (XP Configuration Function)

Purpose — Provides the start value for the associated 32bit debug event counter. See Table 202 on page 618 for similar registers.

Note: That following a RESET, this register contains the boot address read by the XP's IROM.

Address — 0xBD808300

Reset Value — 0xBFC00000

Access — Global Read/Write

Bit Position	31	0
Field Name	**Start Value**	
Reset Value	**0x0000_0000**	

Table 202 Debug Counter*n* Start Value Registers (for Debug Counter 1, 2 and 3)

REGISTER NAME	PURPOSE	ADDRESS	RESET VALUE
Debug Counter1 Start Value	Same as *DebugCounter0 Start Value Register*, except for debug counter 1.	0xBD808304	0x00000000
Debug Counter2 Start Value	Same as *DebugCounter0 Start Value Register*, except for debug counter 2.	0xBD808308	
Debug Counter3 Start Value	Same as *DebugCounter0 Start Value Register*, except for debug counter 3.	0xBD80830C	

Debug Counter0 Control Register (XP Configuration Function)

Purpose Provides control for the associated debug counter register. See Table 203 on page 620 for similar registers.

Address 0xBD808340

Reset Value 0x00008888

Access Global Read/Write

Bit Position	31	30 20	19	18	17	16	15 12	11 8	7 4	3 0
Field Name	DEBUG_EN	Rsvd	INC_ED	LD_ED	STRT_ED	STP_ED	INCSEL	LDSEL	STRTSEL	STPSEL
Reset Value		raz	0	0	0	0	1000	1000	1000	1000

FIELD NAME	BIT POSITION	DESCRIPTION
DEBUG_EN	31	**Debug Enable** — When the bit is 1, it enables all debug counters. When the bit is 0, it disables all debug counters. The Debug Enable bit [31] is only valid in debug counter0 control register, this bit in debug control register 1, 2, and 3 is *not* interpreted. This bit in debugcounter0 control register is the master enable bit, it enables all or none.
Reserved	30:20	Read as zero.
INC_ED	19	**Increment Control Edge Detect Enable** — When the bit is 1, a rising edge detect is performed on the signal selected as the increment signal for the debug counter, providing a one core clock long pulse for each rising edge appearing on the input signal. When the bit is 0, the raw signal is used as the increment signal.

<table>
<tr><th>FIELD NAME</th><th>BIT POSITION</th><th>DESCRIPTION</th></tr>
<tr><td>LD_ED</td><td>18</td><td>Load Control Edge Detect Enable — When the bit is 1, a rising edge detect is performed on the signal selected as the load signal for the debug counter, providing a one core clock long pulse for each rising edge appearing on the input signal. When the bit is 0, the raw signal is used as the load signal.</td></tr>
<tr><td>STRT_ED</td><td>17</td><td>Start Control Edge Detect Enable — When the bit is 1, a rising edge detect is performed on the signal selected as the start signal for the debug counter, providing a one core clock long pulse for each rising edge appearing on the input signal. When the bit is 0, the raw signal is used as the start signal.</td></tr>
<tr><td>STP_ED</td><td>16</td><td>Stop Control Edge Detect Enable — When the bit is 1, a rising edge detect is performed on the signal selected as the stop signal for the debug counter, providing a one core clock long pulse for each rising edge appearing on the input signal. When the bit is 0, the raw signal is used as the stop signal.</td></tr>
<tr><td>INCSEL</td><td>15:12</td><td>Increment Control Select — Selects the signal to use to control the increment line for the debug counter as listed below.
<table>
<tr><th>ENCODED VALUE</th><th>CONTROL VALUE</th></tr>
<tr><td>0000</td><td>Debug bus 0</td></tr>
<tr><td>0001</td><td>Debug bus 1</td></tr>
<tr><td>0010</td><td>Debug bus 2</td></tr>
<tr><td>0011</td><td>Debug bus 3</td></tr>
<tr><td>0100</td><td>Counter0 overflow</td></tr>
<tr><td>0101</td><td>Counter1 overflow</td></tr>
<tr><td>0110</td><td>Counter2 overflow</td></tr>
<tr><td>0111</td><td>Counter3 overflow</td></tr>
<tr><td>1000</td><td>0</td></tr>
<tr><td>1001</td><td>1</td></tr>
<tr><td>1010 -1111</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>LDSEL</td><td>11:8</td><td>Load Control Select — Selects the signal to use to control the load line for the debug counter. The selection values are the same as shown above for INCSEL.</td></tr>
</table>

FIELD NAME	BIT POSITION	DESCRIPTION
STRTSEL	7:4	**Start Control Select** — Selects the signal to use to enable the debug counter for counting. The selection values are the same as shown above for INCSEL.
STPSEL	3:0	**Stop Control Select** — Selects the signal to use to disable the debug counter for counting. The selection values are the same as shown above for INCSEL.

Table 203 Debug Counter*n* Control Registers (for Debug Counter 1, 2 and 3)

REGISTER NAME	PURPOSE	ADDRESS
Debug Counter1 Control	Same as *Debug Counter0 Control Register*, except for debug counter 1.	0xBD808344
Debug Counter2 Control	Same as *Debug Counter0 Control Register*, except for debug counter 2.	0xBD808348
Debug Counter3 Control	Same as *Debug Counter0 Control Register*, except for debug counter 3.	0xBD80834C

Debug Counter0 Current Value Register (XP Configuration Function)

Purpose Provides access to the current value of the associated 32bit debug event counter. See Table 204 on page 620 for similar registers.

Address 0xBD808380

Reset Value 0x00

Access Global Read

Bit Position	31	0
Field Name	**Current Value**	
Reset Value	**0x0000_0000**	

Table 204 Debug Counter*n* Current Value Registers (for Debug Counter 1, 2 and 3)

REGISTER NAME	PURPOSE	ADDRESS
Debug Counter1 Current Value	Same as *Debug Counter0 Current Value Register*, except for counter 1.	0xBD808384
Debug Counter2 Current Value	Same as *Debug Counter0 Current Value Register*, except for counter 2.	0xBD808388

Table 204 Debug Counter*n* Current Value Registers (for Debug Counter 1, 2 and 3) (continued)

REGISTER NAME	PURPOSE	ADDRESS
Debug Counter3 Current Value	Same as *Debug Counter0 Current Value Register*, except for counter 3.	0xBD80838C

All of the registers that pertain to XP DMEM #24 (0xBD804000 to 0xBD80443C) are identical to their counterparts in the Channel Processors (CP) except for those (6) registers documented here. These same (6) registers are also found in the (CP), however, the registers provide different functions for the XP versus the CP. By changing the use of the individual bits inside these registers they are capable of providing the different functions needed in the XP and CPs.

RxCtl0_Status Register (XP DMEM#24 Transfer Rx Control Block0 Function)

Purpose Semaphores governing PCI DMA receive operation for datascope0. See Table 205 on page 621 for similar register.

Address 0xBD804090

Reset Value 0x80000000

Access XP Read/Write

Bit Position	31	30 24	23 0
Field Name	Avail	Reserved	Reserved
Reset Value	1	raz	raz

FIELD NAME	BIT POSITION	DESCRIPTION
Avail	31	**Availability Bit** — When the bit is 1, the XP owns receive datascope 0. When the bit is 0, PCI DMA owns receive datascope0.
Reserved	30:24	Read as zero.
Reserved	23:0	Read as zero.

Table 205 RxCtl1_Status Register (for Datascope1)

REGISTER NAME	PURPOSE	ADDRESS
RxCtl1_Status	Same as RxCtl0_Status, but pertains to datascope1.	0xBD804290

TxCB0_Ctl Register (XP DMEM#24 Transfer Tx Control Block0 Function)

Purpose — Controls PCI DMA for payload transmit operation for datascope0. See Table 206 on page 622 for similar register.

Address — 0xBD804184

Reset Value — 1x0xx00x000x

Access — CPRC Read/Write. Usage is the same as for the CPs with the exception that when using *TxCB0_CTL* to perform DMAs on the PCI bus, the transfer length (*TxLength*) must be a multiple of four bytes.

Table 206 TxCB1_CTL Register

REGISTER NAME	PURPOSE	ADDRESS	ACCESS
TxCB1_CTL	Same as TxCB0_CTL, but pertains to transmit datascope1.	0xBD804384	Same as TxCB0_CTL

TxCtl0_Status Register (XP DMEM#24 Transfer Tx Control Block0 Function)

Purpose — Semaphores governing PCI DMA transmit operation for datascope0. See Table 207 on page 623 for similar register.

Address — 0xBD804190

Reset Value — 0x80000000

Access — XP Read/Write

Bit Position	31	30 … 24	23 … 0
Field Name	Avail	Reserved	Reserved
Reset Value	1	raz	raz

FIELD NAME	BIT POSITION	DESCRIPTION
Avail	31	**Availability Bit** — When the bit is 1, the XP owns transmit datascope 0. When the bit is 0, PCI DMA owns transmit datascope 0.
Reserved	30:24	Read as zero.
Reserved	23:0	Read as zero.

Table 207 TxCtl1_Status Register (for Datascope1)

REGISTER NAME	PURPOSES	ADDRESS
TxCtl1_ Status	Same as TxCtl0_Status, but pertains to transmit datascope1.	0xBD804390

All of the registers that pertain to XP Mode, Queue Status and Event (0xBD804500 to 0xBD8046C0) are identical to their counterparts in the Channel Processors (CP) except for those (4) registers documented here. These same (4) registers are also found in the (CP), however, the registers provide different functions for the XP versus the CP. By changing the use of the individual bits inside these registers they are capability of providing the different functions needed in the XP and CPs.

XP_Mode Register (XP Mode Configuration Function)

Purpose Collects mode and error status bits relevant to general XP configuration.

Address 0xBD804640

Access Upper 16 bits are XP Read/Write, Lower 16 bits are Write 1 to clear, hardware update, except for *QMU rdmbx* and *QMU wrmbx* which are read only.

Bit Position	31	30	29 17	23 22	21 20	19 17	16	15 13	12	11	10 8	7	6 4	3	2 0
Field Name	XP Reset	Wind Down	Rsvd	QMU rdmbx	QMU wrmbx	Rsvd	Retry Global	Rsvd	NXM	PErr #24	PErr Status #24	PErr #25	PErr Status #25	GErr	GErr Status
Reset Value	XPUHOT	raz	raz	0	0	raz	0	raz		0	000	0	000	0	000

FIELD NAME	BIT POSITION	DESCRIPTION
XP Reset	31	**XP Reset** — When the bit is 0, the XP is held in reset state. If the XPUHOT pin is held low upon cold reset, then this bit must be set to 1 with a PCI inbound transaction to bring the XP out of reset. Upon warm reset, the state of this bit is copied from the WARM_XPUHOT bit in the "Software Warm Reset Request Register (XP Miscellaneous Control Function)" on page 605.
WindDown	30	**WindDown** — When the bit is 1, this bit asserts a global signal informing all chip functions to wind down as soon as possible, and leave as much predictable error recovery state around as possible. This bit is readable, allowing a process to determine that the global signal was caused by a process setting this bit, however, the write causes only a single global wind down request.

<table>
<tr><th>FIELD NAME</th><th>BIT POSITION</th><th>DESCRIPTION</th></tr>
<tr><td>Reserved</td><td>29:24</td><td>Read as zero.</td></tr>
<tr><td>QMU rdmbx</td><td>23:22</td><td>QMU Read Mailbox Status (read only):

<table><tr><th>ENCODED VALUE</th><th>STATUS</th></tr><tr><td>00</td><td>QMU idle or operation finished successfully</td></tr><tr><td>01</td><td>operation finished with error (probably resource error, see above</td></tr><tr><td>10</td><td>busy, waiting to begin execution</td></tr><tr><td>11</td><td>busy, executing in QMU engine</td></tr></table></td></tr>
<tr><td>QMU wrmbx</td><td>21:20</td><td>QMU Write Mailbox Status (read only):

<table><tr><th>ENCODED VALUE</th><th>STATUS</th></tr><tr><td>00</td><td>QMU idle or operation finished successfully</td></tr><tr><td>01</td><td>operation finished with error (probably resource error, see above</td></tr><tr><td>10</td><td>busy, waiting to begin execution</td></tr><tr><td>11</td><td>busy, executing in QMU engine</td></tr></table></td></tr>
<tr><td>Reserved</td><td>19:17</td><td>Read as zero.</td></tr>
<tr><td>RetryGlobal</td><td>16</td><td>Global Bus Transaction Retry — This bit causes global load and store operations through the Global bus controller to be retried up to 256 times when NACK'd. When 256 tries have been NACK'd, the bus controller terminates the operation and asserts a bus error.</td></tr>
<tr><td>Reserved</td><td>15:13</td><td>Read as zero.</td></tr>
<tr><td>NXM</td><td>12</td><td>Non-Existent Memory — Indicates that an access has occurred to a non-existent memory location (NXM). This bit is write 1 to clear.</td></tr>
<tr><td>PErr #24</td><td>11</td><td>Payload #24 Error — An error was detected on a payload bus read or write on Payload Bus Node #24.</td></tr>
</table>

FIELD NAME	BIT POSITION	DESCRIPTION
PErr Status #24	10:8	**Payload #24 Error Status** — Loaded when a Payload Error occurs in Payload Bus Node #24, and is locked until the XPRC clears the Payload #24 Error bit. The individual control blocks can be interrogated to determine the specific offender.
PErr #25	7	**Payload #25 Error** — An error was detected on a payload bus read or write on Payload Bus Node #25.
PErr Status #25	6:4	**Payload #25 Error Status** — Loaded when a Payload Error occurs in Payload Bus Node #25, and locked until the XPRC clears the Payload #25 Error bit. The individual control blocks can be interrogated to determine the specific offender.
GErr	3	**Global Bus Controller Error** — An error was detected on a Global read or write attempted by the XPRC.
GErr Status	2:0	**Global Bus Error Status** — Loaded when a global error occurs, and is locked until the XPRC process clears the global error bit. Status codes are identical to those for the CPs.

XP Debug Mode Register (XP Mode Configuration Function)

Purpose Configures the XP debug tap for the global debug counters.

Address 0xBD804658

Access XP Read/Write

Bit Position	31	30 28	27 24	23	22 20	19 16	15	14 12	11 8	7	6 4	3 0
Field Name	Enb0	Rsvd	MUX0	Enb1	Rsvd	MUX1	Enb2	Rsvd	MUX2	Enb3	Rsvd	MUX 3
Reset Value	0	raz	x	0	raz	x	0	raz	x	0	raz	x

FIELD NAME	BIT POSITION	DESCRIPTION
Enb0	31	**Enable Driver 0** — Enable the driver onto global debug wire 0.
Reserved	30:28	Read as zero.
MUX0	27:24	**Mux 0** — Select 1 of 16 debug events onto global debug wire 0.
Enb1	23	**Enable Driver 1** — Enable the driver onto global debug wire 1.
reserved	22:20	Read as zero.
MUX1	19:16	**Mux 1** — Select 1 of 16 debug events onto global debug wire 1.
Enb2	15	**Enable Driver 2** — Enable the driver onto global debug wire 2.

FIELD NAME	BIT POSITION	DESCRIPTION
reserved	14:12	Read as zero.
MUX2	11:8	**Mux 2** — Select 1 of 16 debug events onto global debug wire 2.
Enb3	7	**Enable Driver 3** — Enable the driver onto global debug wire 3.
reserved	6:4	Read as zero.
MUX3	3:0	**Mux 3** — Select 1 of 16 debug events onto global debug wire 3.

There are four (4) global debug wires that carry inputs to the global debug counter block. Each CP and XP have a multiplexor that can select one of the 16 events. The selectable events in the XP to drive on each of the respective debug wires are enumerated in Table 208. Each multiplexor has a 4bit register to select what event to drive, and an enable bit to turn on the debug wire driver. Since chip-wide only one debug wire driver should be enabled at any time, the recommended procedure to use the debug taps is:

1. Clear the master debug enable bit in the global debug configuration register space in the XP.
2. Clear the set driver enable bits. It may be safest to invoke a routine that clears all driver enable bits on every change regardless of the previous configuration.
3. Set one chip-wide driver enable bit and its corresponding multiplexor select value for each global debug wire.
4. Set up the global debug configuration bits and master debug enable.

Table 208 XP Debug Multiplexor Select Encodings

MUX INPUT	ENCODING	DESCRIPTION
0	16:13	Select 0 for multiplexor
ISM_TRAN	12	PCI Master Transaction Initiated
ISM_WRXFER	11	PCI Master has completed a write data phase
ISM_RDXFER	10	PCI Master has completed a read data phase
ISM_DISC	9	PCI Master transaction has been disconnected by addressed target
TSM_TRAN	8	PCI Target has decoded a new inbound transaction
TSM_WRXFER	7	PCI Target has completed a write data phase
TSM_RDXFER	6	PCI Target has completed a read data phase

Table 208 XP Debug Multiplexor Select Encodings (continued)

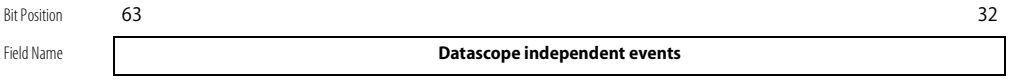
MUX INPUT	ENCODING	DESCRIPTION
TSM_LATTO	5	PCI Target has disconnected due to a data latency time-out
Bubble	4	XPRC has inserted a bubble into its pipeline
Stall	3	XPRC data read or write stall cycle
RC Read	2	XPRC data Read
RC Write	1	XPRC data Write
Debug/Match	0	The XPRC data, data address, or instruction address matches the programmed match registers in the XPRC

Event0 Register (Event and Interrupt Control Function)

Purpose Collects together event bits relevant to datascope independent tasks.

Address 0xBD8046A0

Access XPRC Read/Write, write 1 to clear.



FIELD NAME	BIT POSITION	DESCRIPTION
WindDown	63	**Wind Down** — When unmasked, this global input is a request to wind down all CP activity as soon as possible, and leave as much predictable error recovery state around as possible.
GlobalError	62	**CPRC Global Reference Error—** When asserted, this bit means a CPRC Write received an error on the Global Bus or a non-existent memory error within the cluster.
PayloadError	61	**Payload Error** — Indicates an unrecoverable error occurred during a request sent to the BMU. An error status code is stored in the *xp_mode_register*, and also in the control block that initiated the request. Refer to Table 147 on page 495 for further description of causes of the error. Specifically, Error Codes A, C, D, E, and F cause MCError to be set.
QMUError	60	**QMU Error** — Indicates an unrecoverable error occurred during a request sent to the QMU. An error status code is stored in the *xp_mode_register*.

FIELD NAME	BIT POSITION	DESCRIPTION
CP_Interrupt	59	**CP Interrupt Request** — One of the CPs issued an interrupt request to the XP.
PayloadAlert	58	**Payload Request Alert** — A non-fatal bus error has occurred while trying to send a request to the BMU or QMU. Refer to Table 147 on page 495 for further description of causes of the alert. Specifically, the five (5) Error Codes encoded 9 cause PayloadAlert to be set.
DebugMatch	57	**XPRC Debug Match** — The XPRC data, or data address matched the programmed match registers in the XPRC.
TLUError	56	**TLU Error** — Indicates that an unrecoverable error occurred during a request sent to the TLU.
SB_TransDone	55	**Serial Bus Transfer Done** — Indicates that a read or write transfer completed on the serial bus.
Reserved	54	Read as zero.
RxMsgFIFO	53	**Ring Bus Receive Message Available** — This bit indicates the availability of a Ring Bus message in the receive FIFO, and corresponds to *RxMsgCtl.State* [31].
TimerEvent	52	**Event Timer Time-out** — This bit indicates the event timer counted down to 0.
PCI Mailbox	51	**PCI Mailbox Interrupt** — One of the PCI mailbox registers has been written to by an inbound PCI transaction and contains a pending message.
Reserved	50	Read as zero.
Reserved	49:48	Software controlled.
RxResp7-0	47:40	**Ring Bus Receive Response Available** — These eight bits correspond to the available bit for the eight Ring Bus receive response available bits. Bit 47 represents RxResp7Ctl.Avail, and bit 40 represents RxRespCtl0.Ctl.
TxMsg3-0	39:36	**Ring Bus Transmit Message Available** — These four bits correspond to the available bit for the four Ring Bus transmit message control registers. Bit 39 represents TxMsg3Ctl.Avail, and bit 36 represents *TxMsg0Ctl.Avail*.
WrCB	35:34	**Write Control Blocks 0/1** — These two bits correspond to the available bit for the two payload bus write control blocks. Bit 35 corresponds to *WrCB1Ctl.Avail*, and bit 34 corresponds to *WrCB0Ctl.Avail*.

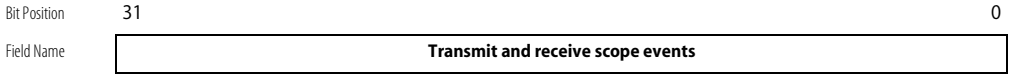
FIELD NAME	BIT POSITION	DESCRIPTION
RdCB	33:32	**Read Control Blocks 0/1** — These two bits correspond to the available bit for the two payload bus read control blocks. Bit 33 corresponds to *RdCB1Ctl.Avail*, and bit 32 corresponds to *RdCB0Ctl.Avail*.

Event1 Register (Event and Interrupt Control Function)

Purpose Collects together event bits relevant to transmit and receive datascopes.

Address 0xBD8046A4

Access CPRC Read/Write, write 1 to clear



FIELD NAME	BIT POSITION	DESCRIPTION
QRdMbxAvail	31	**Queue Read Mailbox Available** — This bit indicates that this XP's read mailbox in the QMU went from busy to available.
TxCB1_Avail (DMEM #24)	30	**Transmit Control Block Available —** Indicates that the available bit for datascope1 Payload Bus transmit control block #24 (TxCB1Ctl.Avail) was set.
TxStatus1_Avail (DMEM #24)	29	**PCI Transmit Datascope1 Available —** Indicates that the PCI transmit state machine has set TxStatus1.Avail.
TxCB1_Avail (DMEM #25)	28	**Transmit Control Block Available** — Indicates that the available bit for datascope1 Payload Bus transmit control block #25 (TxCB1Ctl.Avail) was set.
TxStatus1_Avail (DMEM #25)	27	**IMEM Loader Datascope1 Available** — Indicates that the IMEM Loader state machine has set TxStatus1_Avail.
QRdMbxBusy	26	**Queue Read Mailbox Busy** — This bit indicates that this XP's read mailbox in the QMU went from available to busy.
TxCB0_Avail (DMEM #24)	25	**Transmit Control Block Available** — Indicates that the available bit for datascope0 payload bus transmit control block #24 (TxCB0Ctl.Avail) was set.
TxStatus0_Avail (DMEM #24)	24	**PCI Transmit datascope0 Available** — Indicates that the PCI transmit state machine has set TxStatus0.Avail.

FIELD NAME	BIT POSITION	DESCRIPTION
TxCB0_Avail (DMEM #25)	23	**Transmit Control Block Available** — Indicates that the available bit for datascope0 payload bus transmit control block #25 (TxCB0Ctl.Avail) was set.
TxStatus0_Avail (DMEM #25)	22	**IMEM Loader Datascope0 Available** — Indicates that the IMEM Loader state machine has set TxStatus0_Avail.
QueueStatus	21	**Queue Status** — This bit corresponds to the logical OR of the bits within each of the four *Queue_Status* registers. The bit is level sensitive.
Reserved	20	Read as zero.
WrCB1-0 (DMEM #25)	19:18	**Write Control Block Available** — These two bits correspond to the available bit for the two payload bus write control blocks associated with DMEM #25. Bit 19 corresponds to WrCB1Ctl, and bit 18 corresponds to WrCB0Ctl.Avail.
RdCB1-0 (DMEM #25)	17:16	**Read Control Block Available** — These two bits correspond to the available bit for the two payload bus read control blocks associated with DMEM #25. Bit 19 corresponds to RdCB1Ctl, and bit 18 corresponds to RdCB0Ctl.Avail.
QWrMbxAvail	15	**Queue Write Mailbox Available** — This bit indicates that this XP's write mailbox in the QMU went from busy to available.
RdCB1_Avail (DMEM #24)	14	**Receive Control Block Available** — Indicates that the available bit for datascope1 payload bus receive control block #24 (RxCB1_Ctl.Avail) was set.
RxStatus1_Avail (DMEM #24)	13	**PCI Receive Datascope1 Available** — Indicates that the PCI receive state machine has set RxStatus1.Avail.
RdCB1_Avail (DMEM #25)	12	**Receive Control Block Available** — Indicates that the available bit for datascope1 payload bus receive control block #25 (RxCB1_Ctl.Avail) was set.
RxStatus1_Avail (DMEM #25)	11	**#25 Receive Datascope1 Available** — Indicates that the #25 receive state machine has set RxStatus1.Avail.
QWrMbxBusy	10	**Queue Write Mailbox Busy** — This bit indicates that this XP's write mailbox in the QMU went from available to busy.
RxCB0_Avail (DMEM #24)	9	**Receive Control Block Available** — Indicates that the available bit for datascope0 payload bus receive control block #24 (RxCB0_Ctl.Avail) was set.
RxStatus0_Avail (DMEM #24	8	**PCI Receive Datascope0 Available** — Indicates that the PCI receive state machine has set RxStatus0.Avail.

FIELD NAME	BIT POSITION	DESCRIPTION
RxCB0_Avail (DMEM #25)	7	**Receive Control Block Available** — Indicates that the available bit for datascope0 payload bus receive control block #25 (RxCB0_Ctl.Avail) was set.
RxStatus0_Avail (DMEM #25	6	**#25 Receive Datascope0 Available** — Indicates that the #25 receive state machine has set RxStatus0.Avail.
Ext_Interrupt	5	**External PHY Interrupt** — A PHY generated external interrupt that comes in through the XPU_HOT pin when not in reset, causes an event in this bit.
PCI Master Abort	4	**PCI Master Transaction Aborted** — The PCI Master has experienced either a Master Abort, a Target Abort, or an abort caused by the PCI Master being disabled.
Debug Events	3:0	**Debug Events** — Bits 3:0 correspond to debug counters 3:0, and are set whenever a debug counter triggers its event signal.

All of the registers that pertain to XP DMEM #25 (0xBD804800 to 0xBD804C3C) are identical to their counterparts in the Channel Processors (CP) except for those (4) registers documented here. These same (4) registers are also found in the (CP), however, the registers provide different functions for the XP versus the CP. By changing the use of the individual bits inside these registers they are able of providing the different functions needed in the XP and CPs.

RxCtl0_Status Register (XP DMEM#25 Transfer Control Block0 Function)

Purpose Semaphores governing PCI DMA receive operation for datascope0. See Table 209 on page 632 for similar register.

Address 0xBD804890

Reset Value 0x80000000

Access XP Read/Write

Bit Position	31	30 24	23 0
Field Name	**Avail**	**Reserved**	**Reserved**
Reset Value	**1**	**raz**	**raz**

FIELD NAME	BIT POSITION	DESCRIPTION
Avail	31	**Availability Bit** — When the bit is 1, the XP owns receive datascope0. When the bit is 0, PCI DMA owns receive datascope0.
Reserved	30:24	Read as zero.
Reserved	23:0	Read as zero.

Table 209 RxCtl1_Status Register

REGISTER NAME	PURPOSE	ADDRESS
RxCtl1_Status	Same as *RxCtl0_Status*, but for datascope1.	0xBD80A290

TxCtl0_Status Register (XP DMEM#25 Transfer Control Block0 Function)

Purpose Semaphores governing PCI DMA transmit operation for datascope0. See Table 210 on page 632 for similar register.

Address 0xBD804990

Reset Value 0x80000000

Access XP Read/Write

Bit Position	31	30 24	23 0
Field Name	**Avail**	**Reserved**	**Reserved**
Reset Value	**1**	**raz**	**raz**

FIELD NAME	BIT POSITION	DESCRIPTION
Avail	31	**Availability Bit** — When the bit is 1, the XP owns transmit datascope0. When the bit is 0, PCI DMA owns transmit datascope0.
Reserved	30:24	Read as zero.
Reserved	23:0	Read as zero.

Table 210 TxCtl1_Status Register

REGISTER NAME	PURPOSE	ADDRESS
TxCtl1_Status	Same as *TxCtl0_Status*, but pertains to transmit datascope1.	0xBD804B90

Queue Management Unit (QMU) Configuration Registers

Configuration Space in the QMU is an area that contains a number of registers. The QMU occupies 1MByte within the C-5e NP's Configuration Space starting at 0xBDA00000 to 0xBDAFFFFF. The QMU only supports 32bit aligned operations. The QMU uses these registers to: map queues to CPs, XP and FP, configure the QMU, debug the QMU, and collect QMU statistics. Processor registers (WrCB0, and RdCB0) are described in Chapter 2 to move data through the internal QMU and external QMU to/from SRAM from/to the DMEM of either the requesting CPs, or XP.

These registers (WrCB0_Sys_Addr, WrCB0_Ctl, WrCB0_DMA_Addr and RdCB0_Sys_Addr, RdCB0_Ctl, RdCB0_DMA_Addr) are physically located in the Configuration Space of their respective CPs, or XP and not in the QMU Configuration Space.

Warning: *When the QMU is run-enabled, an attempt to read or write many of the internal registers and memories will interfere with the operation of the QMU.*

Warning: *Although the C-5e NP provides an External QMU Mode, it does not support an external traffic manager device. In addition, the associated registers and fields should not be implemented.*

QMU Registers

The following is a list of each QMU register along with its address, function, and reference to its detailed parameters. The detailed parameters provide: purpose, field name, bit position, and descriptions.

Table 211 QMU Registers

ADDRESS	REGISTER NAME	FUNCTION	DETAILED PARAMETERS
0xBDA00000	QMU_Run_Enable	Enables QMU	See page 637
0xBDA00020	Clear_Statistics	QMU Statistics	See page 637
0xBDA00040	Base_Queue_CP0	CP's Queue Allocation	See page 637
0xBDA00044	Base_Queue_CP1		
0xBDA00048	Base_Queue_CP2		
0xBDA0004C	Base_Queue_CP3		
0xBDA00050	Base_Queue_CP4		
0xBDA00054	Base_Queue_CP5		
0xBDA00058	Base_Queue_CP6		
0xBDA0005C	Base_Queue_CP7		
0xBDA00060	Base_Queue_CP8		
0xBDA00064	Base_Queue_CP9		
0xBDA00068	Base_Queue_CP10		
0xBDA0006C	Base_Queue_CP11		
0xBDA00070	Base_Queue_CP12		
0XBDA00074	Base_Queue_CP13		
0xBDA00078	Base_Queue_CP14		
0xBDA0007C	Base_Queue_CP15		
0xBDA000C0	Base_Queue_FP	FP's Queue Allocation	See page 640
0xBDA000C8	Base_Queue_XP	XP's Queue Allocation	See page 641

Table 211 QMU Registers (continued)

ADDRESS	REGISTER NAME	FUNCTION	DETAILED PARAMETERS
0xBDA000D0	Num_Queues	QMU Configuration	See page 641
0xBDA000D4	Num_Descriptors		See page 642
0xBDA000DC	Dyn_Des_Usage_Lim_Pool0		See page 642
0xBDA000E0	Dyn_Des_Usage_Lim_Pool1		
0xBDA000E4	Dyn_Des_Usage_Lim_Pool2		
0xBDA000E8	Dyn_Des_Usage_Lim_Pool3		
0xBDA000F0	Operation_Mode Note: The External Mode is not supported.		See page 643
0xBDA000F4	Descriptor_Size		See page 644
0xBDA00180	Config_Q_Cnt	QMU Statistics	See page 644
0xBDA00184	Rd_Q_Status_Cnt		See page 644
0xBDA00188	CP_Uni_Enq_Cnt		See page 645
0xBDA0018C	CP_Multi_Enq_Cnt		See page 645
0xBDA00190	CP_Multi_Enq_Target_Cnt		See page 645
0xBDA00194	CP_Dequeue_Cnt		See page 645
0xBDA00198	FP_Uni_Enq_Cnt		See page 645
0xBDA0019C	FP_Multi_Enq_Cnt		See page 645
0xBDA001A0	FP_Multi_Enq_Target_Cnt		See page 646
0xBDA001A4	FP_Dequeue_Cnt		See page 646
0xBDA001A8	QMU_Idle_Cycles		See page 646
0xBDA001AC	Payload_NACK_Cnt		See page 646
0xBDA001B0	Global_NACK_Cnt		See page 646
0xBDA001B4	Payload_Rd_Failures_Cnt		See page 646
0xBDA001B8	Cmd_Processor_Err_Cnt		See page 647
0xBDA001C0	Dq_H_Par_Err_Cnt		See page 647
0xBDA001C4	Dq_L_Par_Err_Cnt		See page 648

Table 211 QMU Registers (continued)

ADDRESS	REGISTER NAME	FUNCTION	DETAILED PARAMETERS
0xBDA001C8	Missing_Front_Seq_Num_Cnt	QMU Sequence Numbers	See page 648
0xBDA001CC	Front_Seq_Num		See page 649
0xBDA001D0	Back_Seq_Num		See page 649
0xBDA001D4	Front_Seq_Num_Timeout		See page 650
0xBDA00400 to 0xBDA0063C	Multicast_Destination0 to Multicast_Destination255	QMU Configuration	See page 651
0xDBA7E000	Free_Descriptor_List _Head	QMU Control	See page 651
0xDBA7E004	Free_Descriptor_List_Tail		See page 652
0xBDA7E008	Free_Descriptor_Buffer_List		See page 652
0xBDA7E080	Dyn_Descriptor_Pool0_Usage	QMU Status	See page 653
0xBDA7E084	Dyn_Descriptor_Pool1_Usage		See Table 215 on page 653.
0xBDA7E088	Dyn_Descriptor_Pool2_Usage		
0xBDA7E08C	Dyn_Descriptor_Pool3_Usage		

QMU Detailed Descriptions

The following is a detailed description of each of the QMU registers and their individual parameters. The detailed parameters provide: purpose, field name, bit positions and descriptions.

QMU_Run_Enable Register (QMU Enable Queue Function)

Purpose When this one bit wide register is 1, it enables the QMU's processing of queue operations. When this bit is 0, it disables the QMU's execution of queue operations. The QMU powers up when this bit is clear. This bit must be set to a "1" before the QMU can process any queue operations.

Address 0xBDA00000

Access Global Read/Write

Bit Position	31 ... 1	0
Field Name	Rsvd	Enable

Clear_Statistics Register (QMU Statistics Function)

Purpose Enables/Disables and clears QMU statistics counters. There are 16 counters as defined here.

Address 0xBDA00020

Access Global Read/Write

Bit Position	31 ... 16	15 ... 0
Field Name	Reserved	QMU Statistics Counter Bits
Reset Value	0000_0000_0000-0000	0000_0000_0000-0000

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:16:	Read as zero.
DeqParErrCtr	15	**Dequeue Parity Error Count/Clear** — When this bit is set to a 0, the DqHParErrCnt and DqLParErrCnt registers begin counting parity errors. WHen set to a 1, these registers are cleared and the counting is disabled.

FIELD NAME	BIT POSITION	DESCRIPTION
CmdProcErrCtr	14	**Command Processor Error Count/Clear** — When this bit set to a 0, the Cmd_Processor_Err_Cnt register begins counting Cmd_Processor errors, illegal opcodes, and out of range queue numbers. When set to a 1, the Cmd_Processor_Err_Cnt register is cleared and the counting is disabled.
PayRdFailCtr	13	**Payload Read Failure Count/Clear** — When this bit set to a 0, the Payload_Read_Failures_Cnt register begins counting payload read failures. When set to a 1, the Payload_Read_Failures_Cnt register is cleared and the counting is disabled.
GlbNACKCtr	12	**Global NACK Count/Clear** — When this bit set to a 0, the Global_Nack_Cnt register begins counting global NACKs. When set to a 1, the Global_Nack_Cnt register is cleared and the counting is disabled.
PayNACKCtr	11	**Payload NACK Count/Clear** — When this bit set to a 0, the Payload_Nack_Cnt register begins counting global NACKs. When set to a 1, the Payload_Nack_Cnt register is cleared and the counting is disabled.
QMUIdlCycCtr	10	**QMU Idle Cycles Count/Clear** — When this bit set to a 0, the Qmu_Idle_Cycles register begins counting QMU idle cycles. When set to a 1, the Qmu_Idle_Cycles register is cleared and the counting is disabled.
FPDeqCtr	9	**FP Dequeue Count/Clear** — When this bit set to a 0, the Fp_Dequeue_Cnt register begins counting FP dequeues. When set to a 1, the Fp_Dequeue_Cnt register is cleared and the counting is disabled.
FPMultiEnqTarCtr	8	**FP Multicast Enqueue Target Count/Clear** — When this bit set to a 0, the Fp_Multi_Enq_Target_Cnt register begins counting FP multicast enqueue targets. When set to a 1, the Fp_Multi_Enq_Target_Cnt register is cleared and the counting is disabled.
FPMultiEnqCtr	7	**FP Multicast Enqueue Count/Clear** — When this bit set to a 0, the Fp_Multi_Enq_Cnt register begins counting FP multicast enqueue s. When set to a 1, the Fp_Multi_Enq_Cnt register is cleared and the counting is disabled.
FPUniEnqCtr	6	**FP Unicast Enqueue Count/Clear** — When this bit set to a 0, the Fp_Uni_Enq_Cnt register begins counting FP unicast enqueue s. When set to a 1, the Fp_Uni_Enq_Cnt register is cleared and the counting is disabled.

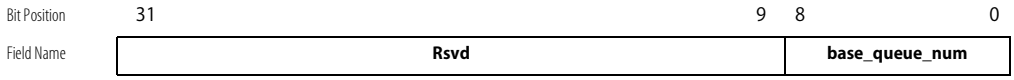
FIELD NAME	BIT POSITION	DESCRIPTION
CPDeqCtr	5	**CP Dequeue Count/Clear** — When this bit set to a 0, the Cp_Dequeue_Cnt register begins counting CP dequeues. When set to a 1, the Cp_Dequeue_Cnt register is cleared and the counting is disabled.
CPMultiEnqTarCtr	4	**CP Multicast Enqueue Target Count/Clear** — When this bit set to a 0, the Cp_Multi_Enq_Target_Cnt register begins counting CP mutlicast enqueue targets. When set to a 1, the Cp_Multi_Enq_Target_Cnt register is cleared and the counting is disabled.
CPMultiEnqCtr	3	**CP Multicast Enqueue Count/Clear** — When this bit set to a 0, the Cp_Multi_Enq_Cnt register begins counting CP mutlicast enqueues. When set to a 1, the Cp_Multi_Enq_Cnt register is cleared and the counting is disabled.
CPUniEnqCtr	2	**CP Unicast Enqueue Count/Clear** — When this bit set to a 0, the Cp_Uni_Enq_Cnt register begins counting CP unicast enqueues. When set to a 1, the Cp_Uni_Enq_Cnt register is cleared and the counting is disabled.
RdQueStatCtr	1	**Read Queue Status Count/Clear** — When this bit set to a 0, the Rd_Q_Status_Cnt register begins counting read status operations. When set to a 1, the Rd_Q_Status_Cnt register is cleared and the counting is disabled.
QueConfCtr	0	**Queue Configuration Count/Clear** — When this bit set to a 0, the Config_Q_Cnt register begins counting queue configuration operations. When set to a 1, the Config_Q_Cnt register is cleared and the counting is disabled.

Base_Queue_CP0 to Base_Queue_CP15 Registers (QMU CP's Queue Allocation Function)

Purpose These registers specify the base address for a CP's queues.

Addresses 0xBDA00040 (CP 0 base address), 0xBDA00044 (CP 1 base address)
0xBDA00048 (CP 2 base address), 0xBDA0004C (CP 3 base address)
0xBDA00050 (CP 4 base address), 0xBDA00054 (CP 5 base address)
0xBDA00058 (CP 6 base address), 0xBDA0005C (CP 7 base address)
0xBDA00060 (CP 8 base address), 0xBDA00064 (CP 9 base address)
0xBDA00068 (CP 10 base address), 0xBDA0006C (CP 11 base address)
0xBDA00070 (CP 12 base address), 0xBDA00074 (CP 13 base address)
0xBDA00078 (CP 14 base address), 0xBDA0007C (CP 15 base address)

Access Global Read/Write



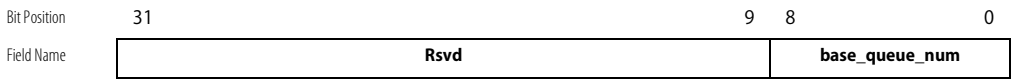
FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:9	Read as zero.
Base_Queue_Num	8:0	Establishes the base queue address for the CP (0 to 15).

Base_Queue_FP Register (QMU FP's Queue Allocation Function)

Purpose This register specifies the base address for the FP's queues.

Address 0xBDA000C0

Access Global Read/Write



FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:9	Read as zero.
Base_Queue_Num	8:0	Specifies the base address for the FP's queues

Base_Queue_XP Register (QMU XP's Queue Allocation Function)

Purpose This register specifies the base address for the XP queues. It has the same data format as *Base_Queue_FP* register.

Address 0xBDA000C8

Access Global Read/Write

Num_Queues Register (QMU Configuration Function)

Purpose This nine bit wide field specifices the total number of queues in the QMU. This register is checked in the command processor to determine whether a queue specified in a command is in the valid configured range. The number of queues specified does *not* have to equal the maximum number of queues supported by the operating mode.

In internal-queue mode, the maximum number of queues is 512. Legal range= 0 to 511 as detailed here:

PROGRAMMED VALUE	NUMBER OF QUEUES
0 • • • 511	1 • • • 512

Address 0xBDA000D0

Access Global Read/Write

Bit Position	31	9	8	0
Field Name	Rsvd		Data	

Num_Descriptors Register (QMU Configuration Function)

Purpose This 14 bit wide register specifies the number of descriptor buffers to be available in the QMU. Legal range= 0 to 16,383 as detailed here:

PROGRAMMED VALUE	NUMBER OF DESCRIPTORS
0 • • • 16,383	1 • • • 16,384

Address 0xBDA000D4

Access Global Read/Write

Bit Position	31 ... 14	13 ... 0
Field Name	**Rsvd**	**Data**

Dyn_Des_Usage_Lim_Pool0 Register (QMU Configuration Function)

Purpose Specify the maximum number of descriptors that can be enqueued dynamically to Pool0. Legal range 0 to 16K -1.

The total number of Descriptors allocated among all four (4) pools of the *Dyn_Des_Usage_Lim_Pool0* to *Dyn_Des_Usage_Lim_Pool3* registers should be < the number of dynamically enqueued descriptors. See Table 212 on page 643 for similar registers.

Address 0xBDA000DC

Access Global Read/Write

Bit Position	31 ... 14	13 ... 0
Field Name	**Rsvd**	**Data**

Table 212 Dyn_Des_Usage_Lim_Pool*n* Registers (for Descriptor Pools 1, 2 and 3)

REGISTER NAME	PURPOSE	ADDRESS
Dyn_Des_Usage_Lim_Pool1	Same as *Dyn_Des_Usage_Lim_Pool0*, except for descriptor pool1.	0xBDA000E0
Dyn_Des_Usage_Lim_Pool2	Same as *Dyn_Des_Usage_Lim_Pool0*, except for descriptor pool2.	0xBDA000E4
Dyn_Des_Usage_Lim_Pool3	Same as *Dyn_Des_Usage_Lim_Pool0*, except for descriptor pool2.	0xBDA000E8

Operation_Mode Register (QMU Configuration Function)

Purpose This four bit wide register specifies the operating mode of the QMU. The codes for the modes are listed in Table 213.

Address 0xBDA000F0

Access Global Read/Write

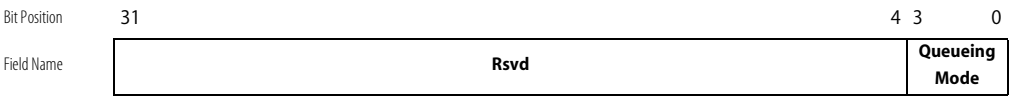


Table 213 Queue Operating Mode Codes

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:4	Read as zero.
QueueingMode	3:0	**Queueing Mode:** ENCODED VALUE / MODE 01 — Internal queueing mode (QMU performs queueing using external descriptor memory. 10 — External scheduler mode. Note: The External Mode is not supported. 0011 to 1111 — Reserved

Descriptor_Size Register (QMU Configuration Function)

Purpose This two bit wide register specifies the size of the data stored for each descriptor in an encoded form. When in external mode, it indicates the VOP-Descriptor capacity.

Note: The External Mode is not supported.

Address 0xBDA000F4

Access Global Read/Write

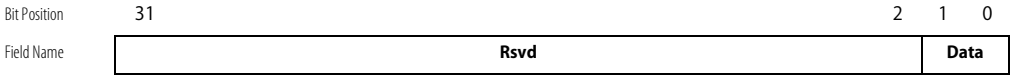


Table 214 Descriptor Size and VOP-Descriptor Capacity Values

ENCODED VALUE	DESCRIPTOR SIZE (BYTES)	VOP-DESCRIPTOR CAPACITY (WHEN IN EXTERNAL MODE
0	12	2048 Note: The External Mode is not supported.
1	16	2048 Note: The External Mode is not supported.
2	24	1536 Note: The External Mode is not supported.
3	32	1024 Note: The External Mode is not supported.

Config_Q_Cnt Register (QMU Statistics Function)

Purpose Count of Queue Configuration operations.

Address 0xBDA00180

Access Global Read

Rd_Q_Status_Cnt Register (QMU Statistics Function)

Purpose Count of Read Status operations.

Address 0xBDA00184

Access Global Read

CP_Uni_Enq_Cnt Register (QMU Statistics Function)

Purpose Count of Unicast Enqueues from the CPs.

Address 0xBDA00188

Access Global Read

CP_Multi_Enq_Cnt Register (QMU Statistics Function)

Purpose Count of Multicast Enqueues from the CPs.

Address 0xBDA0018C

Access Global Read

CP_Multi_Enq_Target_Cnt Register (QMU Statistics Function)

Purpose Count of Total Multicast Enqueues Targets from the CPs.

Address 0xBDA00190

Access Global Read

CP_Dequeue_Cnt Register (QMU Statistics Function)

Purpose Count of Dequeue operations from the CPs.

Address 0xBDA00194

Access Global Read

FP_Uni_Enq_Cnt Register (QMU Statistics Function)

Purpose Count of Unicast Enqueues from the FP.

Address 0xBDA00198

Access Global Read

FP_Multi_Enq_Cnt Register (QMU Statistics Function)

Purpose Count of Total Multicast Enqueues from the FP.

Address 0xBDA0019C

Access Global Read

FP_Multi_Enq_Target_Cnt Register (QMU Statistics Function)

Purpose Count of Multicast Enqueue Targets from the FP.

Address 0xBDA001A0

Access Global Read

FP_Dequeue_Cnt Register (QMU Statistics Function)

Purpose Count of Dequeue operations from the FP.

Address 0xBDA001A4

Access Global Read

QMU_Idle_Cycles Register (QMU Statistics Function)

Purpose Count of QMU Idle Clock Cycles.

Address 0xBDA001A8

Access Global Read

Payload_NACK_Cnt Register (QMU Statistics Function)

Purpose Count of payload NACKs.

Address 0xBDA001AC

Access Global Read

Global_NACK_Cnt Register (QMU Statistics Function)

Purpose Count of Global NACKs.

Address 0xBDA001B0

Access Global Read

Payload_Read_Failures_Cnt Register (QMU Statistics Function)

Purpose Count of payload read failures.

Address 0xBDA001B4

Access Global Read

Cmd_Processor_Err_Cnt Register (QMU Statistics Function)

Purpose Count of command processor errors, illegal opcodes and out of range queue numbers.

Address 0xBDA001B8

Access Global Read

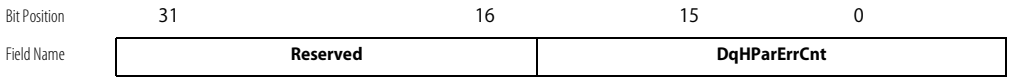
Dq_H_Par_Err_Cnt Register (QMU Sequence Numbers Function)

Purpose Counts errors in dequeued descriptors in both internal and external modes.

Note: The External Mode is not supported.

Address 0xBDA001C0

Access Global Read



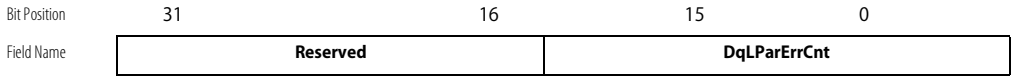
FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:16	Read as zero.
DqHParErrCnt	15:0	**Dequeue High Parity Error Count** — In Internal Mode = counts the number of parity errors received on the high half of the external data bus (QD [31:16]). In External Mode= counts the number of descriptors with parity errors received from the Q-5 (covers the 24bit data bus, NQRDY and QARDY). Parity error sensing for a descriptor also covers any idle time before the descriptor. Note: The External Mode is not supported. Note: This is a saturating counter.

Dq_L_Par_Err_Cnt Register (QMU Sequence Numbers Function)

Purpose Counts errors in dequeued descriptors in internal mode.
Note: The External Mode is not supported.

Address 0xBDA001C4

Access Global Read



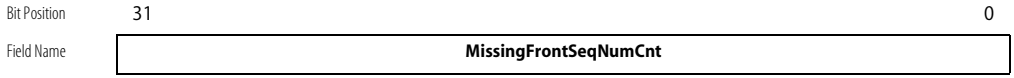
FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:16	Read as zero.
DqLParErrCnt	15:0	**Dequeue Low Parity Error Count** — In Internal Mode = counts the number of parity errors received on the low half of the external data bus (QD [15:0]). Note: Not used in External Mode. Note: This is a saturating counter.

Missing_Front_Seq_Num_Cnt Register (QMU Sequence Numbers Function)

Purpose Provides a 32bit field for a saturating count of the number of sequence numbers missing in front port enqueues to the QMU.

Address 0xBDA001C8

Access Global Read/Write



FIELD NAME	BIT POSITION	DESCRIPTION
MissingFrontSeqNumCnt	31:0	**Missing Front Sequence Number Count** — A saturating count of the number of sequence numbers missing in front port enqueues to the QMU.

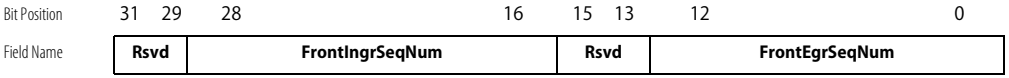
Front_Seq_Num Register (QMU Sequence Numbers Function)

Purpose — Provides fields for the ingress and egress sequence numbers used with front-ports.

Note: There is no need to monitor or set these values. Access is provided for test and initialization purposes only. Any writes during normal operation results in loss of a large number of PDUs.

Address — 0xBDA001CC

Access — Global Read/Write



FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:29	Read as zero.
FrontIngrSeqNum	28:16	**Front Ingress Sequence Number** — Contains the next sequence number the QMU looks for in enqueueing descriptors from the front ports.
Reserved	15:13	Read as zero.
FrontEgrSeqNum	12:0	**Front Egress Sequence Number** — Contains the next sequence number the QMU supplies with a descriptor sent to the front ports.

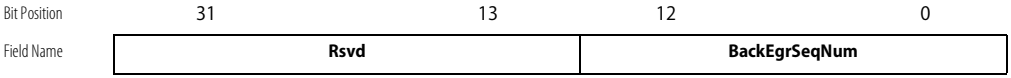
Back_Seq_Num Register (QMU Sequence Numbers Function)

Purpose — Provides a field for the egress sequence numbers used with back ports.

Note: There is no need to monitor or set these values. Access is provided for test and initialization purposes only. Any writes during normal operation results in loss of a large number of PDUs.

Address — 0xBDA001D0

Access — Global Read/Write



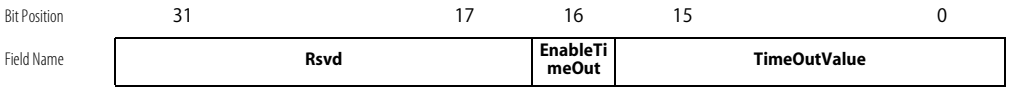
FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:13	Read as zero.
BackEgrSeqNum	12:0	**Back Egress Sequence Number** — Contains the next sequence number the QMU supplies with a descriptor sent to the back ports.

Front_Seq_Num_Timeout Register (QMU Sequence Numbers Function)

Purpose Provides fields that control a time out associated with sequence number on enqueue.

Address 0xBDA001D4

Access Global Read/Write



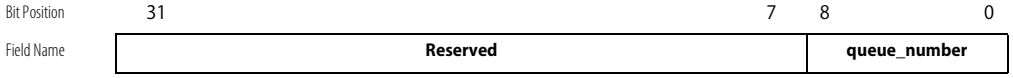
FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:17	Read as zero.
EnableTimeOut	16	**Enable Time Out** — When set =1 and front port sequence numbers are enabled, then the missing sequence number time out is enabled. Each time an enqueue is read from a mail box the time out starts. If when the counter reaches zero (0) there are still some write mail boxes not-empty, then the FrontIngrSeqNum increments and the time out is reset to the initial value given in [15:0] of this register.
TimeOutValue	15:0	**Time Out Value** — A 16bit value that is loaded into a decrementing counter as described above. The counter is decremented each core clock cycle when enabled.

Multicast_Destination0 to Multicast_Destination255 Registers (QMU Configuration Function)

Purpose Provide the mapping of the multicast destination port and queue level number to target a queue number for each leaf of a multicast elaboration.

Address 0xBDA00400 — 0xBBA007FC

Access Global Read/Write



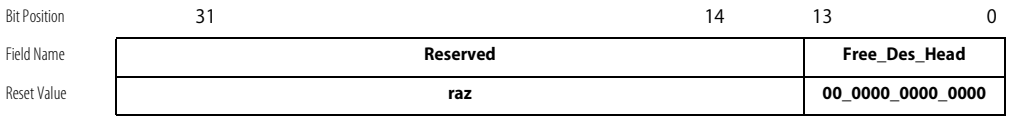
FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:7	Read as zero.
queue_number	8:0	**Destination Queue Number** — Provide the mapping of the multicast destination port and queue level number to target a queue number for each leaf of a multicast elaboration.

Free_Descriptor_List_Head Register (QMU Control Function)

Purpose The descriptor number at the head of the linked list of free descriptors (the next free descriptor to be used).

Address 0xBDA7E000

Access Global Read/Write (writes are for diagnostics purposes only). Any writes to this register are likely to corrupt all traffic, resuiring a hard reset.



Free_Descriptor_List_Tail Register (QMU Control Function)

Purpose The descriptor number at the tail of the linked list of free descriptors.

Address 0xBDA7E004

Access Global Read/Write (writes are for diagnostics purposes only). Any writes to this register are likely to corrupt all traffic, resuiring a hard reset.

Bit Position	31 ... 14	13 ... 0
Field Name	**Reserved**	**Free_Des_Tail**
Reset Value	**raz**	**00_0000_0000_0000**

Free_Descriptor_Buffer_List Register (QMU Control Function)

Purpose Designates the total number of free descriptors.

Address 0xBDA7E008

Access Global Read/Write (writes are for diagnostics purposes only). Any writes to this register are likely to corrupt all traffic, resuiring a hard reset.

Bit Position	31 ... 14	13 ... 0
Field Name	**Reserved**	**Free_Des_Buffer**
Reset Value	**raz**	**00_0000_0000_0000**

Dyn_Descriptor_Pool0_Usage Register (QMU Status Function)

Purpose — Designates how many dynamic descriptor buffers are in use in Pool0. Legal range= 0 to 16K-1. For example, if the queue allowance is 10 and the limit is 15, the 11th descriptor-enqueue will trigger the usage register to move from 0 to 1 (and up to 5 if all 15 descriptors are enqueued). See Table 215 on page 653 for similar register.

Address — 0xBDA7E080

Access — Global Read/Write

Table 215 Dyn_Descriptor_Buffer_Usage_Pool*n* Register (for Pool1, 2 and 3)

REGISTER NAME	PURPOSE	ADDRESS
Dyn_Descriptor_Pool1_Usage	Same as Dyn_Descriptor_Pool0_Usage, but for pool1.	0xBDA7E084
Dyn_Descriptor_Pool2_Usage	Same as Dyn_Descriptor_Pool0_Usage, but for pool2.	0xBDA7E088
Dyn_Descriptor_Pool3_Usage	Same as Dyn_Descriptor_Pool0_Usage, but for pool3.	0xBDA7E08C

Buffer Management Unit (BMU) Configuration Registers

Configuration Space in the BMU contains a number of registers. The BMU uses these registers to configure and operate the BMU. The BMU uses others registers (WrCB0, RdCB0, RxCB0 and TxCB0) as described in Chapter 2 to move data through the BMU to/from SDRAM from/to the DMEM of either the requesting CPs, XP or FP.

These registers (WrCB0_Sys_Addr, WrCB0_Ctl, WrCB0_DMA_Addr, WrCB0_SDP_Addr; RdCB0_Sys_Addr, RdCB0_Ctl, RdCB0_DMA_Addr, RdCB0_SDP_Addr; RxCB0_Sys_Addr, RxCB0_Ctl, RxCB0_DMA_Addr, RxCB0_SDP_Addr; and TxCB0_Sys_Addr, TxCB0_Ctl, TxCB0_DMA_Addr, TxCB0_SDP_Addr) *are physically located in the Configuration Space of their respective CPs and not in the BMU Configuration Space.*

The BMU registers are located in the KSEG1 region, (0xA0000000 to 0xBFFFFFFF), which is uncached, starting at address 0xBDB00000. Refer to "C-5e NP Address Mapping" on page 69.

Warning: *Attempting to access a buffer pool before it is setup results in unpredictable behavior.*

BMU Registers

The following is a list of each register along with its address, function and reference to its detailed parameters. The detailed parameters provide: purpose, field name, bit positions and descriptions.

Table 216 BMU Registers

ADDRESS	REGISTER NAME	FUNCTION	DETAILED PARAMETERS
0xBDB00000	Pool0 Base	Buffer Pool Base Address	See page 660
0xBDB00004	Pool1 Base		
0xBDB00008	Pool2 Base		
0xBDB0000C	Pool3Base		
0xBDB00010	Pool4 Base		
0xBDB00014	Pool5 Base		
0xBDB00018	Pool6 Base		
0xBDB0001C	Pool7 Base		
0xBDB00020	Pool8 Base		
0xBDB00024	Pool9 Base		
0xBDB00028	Pool10 Base		
0xBDB0002C	Pool11 Base		
0xBDB00030	Pool12 Base		
0xBDB00034	Pool13 Base		
0xBDB00038	Pool14 Base		
0xBDB0003C	Pool15 Base		
0xBDB00040	Pool16 Base		
0xBDB00044	Pool17 Base		
0xBDB00048	Pool18 Base		
0xBDB0004C	Pool19 Base		

Table 216 BMU Registers (continued)

ADDRESS	REGISTER NAME	FUNCTION	DETAILED PARAMETERS
0xBDB00050	Pool20 Base	Buffer Pool Base Address (continued)	See page 660
0xBDB00054	Pool21 Base		
0xBDB00058	Pool22 Base		
0xBDB0005C	Pool23 Base		
0xBDB00060	Pool24 Base		
0xBDB00064	Pool25 Base		
0xBDB00068	Pool26 Base		
0xBDB0006C	Pool27 Base		
0xBDB00070	Pool28 Base		
0xBDB00074	Pool29 Base		
0xBDB10000	Pool0 BTag Shift	Encoded Buffer Size	See page 661
0xBDB10004	Pool1 BTag Shift		
0xBDB10008	Pool2 BTag Shift		

Table 216 BMU Registers (continued)

ADDRESS	REGISTER NAME	FUNCTION	DETAILED PARAMETERS
0xBDB1000C	Pool3 BTag Shift	Encoded Buffer Size I (continued)	See page 661
0xBDB10010	Pool4 BTag Shift		
0xBDB10014	Pool5 BTag Shift		
0xBDB10018	Pool6 BTag Shift		
0xBDB1001C	Pool7 BTag Shift		
0xBDB10020	Pool 8 BTag Shift		
0xBDB10024	Pool9 BTag Shift		
0xBDB10028	Pool10 BTag Shift		
0xBDB1002C	Pool11 BTag Shift		
0xBDB10030	Pool12 BTag Shift		
0xBDB10034	Pool13 BTag Shift		
0xBDB10038	Pool14 BTag Shift		
0xBDB1003C	Pool15 BTag Shift		
0xBDB10040	Pool16 BTag Shift		
0xBDB10044	Pool17 BTag Shift		
0xBDB10048	Pool18 BTag Shift		
0xBDB1004C	Pool19 BTag Shift		
0xBDB10050	Pool20 BTag Shift		
0xBDB10054	Pool21 BTag Shift		
0xBDB10058	Pool22 BTag Shift		
0xBDB1005C	Pool23 BTag Shift		
0xBDB10060	Pool24 BTag Shift		
0xBDB10064	Pool25 BTag Shift		
0xBDB10068	Pool26 BTag Shift		
0xBDB1006C	Pool27 BTag Shift		
0xBDB10070	Pool28 BTag Shift		
0xBDB10074	Pool29 BTag Shift		
0xBDB20000	BTag FIFO Base0	BTag FIFO Base Address	See page 662

Table 216 BMU Registers (continued)

ADDRESS	REGISTER NAME	FUNCTION	DETAILED PARAMETERS
0xBDB20004	BTag FIFO Base1	BTag FIFO Base Address (continued)	See page 662
0xBDB20008	BTag FIFO Base2		
0xBDB2000C	BTag FIFO Base3		
0xBDB20010	BTag FIFO Base4		
0xBDB20014	BTag FIFO Base5		
0xBDB20018	BTag FIFO Base6		
0xBDB2001C	BTag FIFO Base7		
0xBDB20020	BTag FIFO Base8		
0xBDB20024	BTag FIFO Base9		
0xBDB20028	BTag FIFO Base10		
0xBDB2002C	BTag FIFO Base11		
0xBDB20030	BTag FIFO Base12		
0xBDB20034	BTag FIFO Base13		
0xBDB20038	BTag FIFO Base14		
0xBDB2003C	BTag FIFO Base15		
0xBDB20040	BTag FIFO Base16		
0xBDB20044	BTag FIFO Base17		
0xBDB20048	BTag FIFO Base18		
0xBDB2004C	BTag FIFO Base19		
0xBDB20050	BTag FIFO Base20		
0xBDB20054	BTag FIFO Base21		
0xBDB20058	BTag FIFO Base22		
0xBDB2005C	BTag FIFO Base23		
0xBDB20060	BTag FIFO Base24		
0xBDB20064	BTag FIFO Base25		
0xBDB20068	BTag FIFO Base26		
0xBDB2006C	BTag FIFO Base27		
0xBDB20070	BTag FIFO Base28		
0xBDB20074	BTag FIFO Base29		

Table 216 BMU Registers (continued)

ADDRESS	REGISTER NAME	FUNCTION	DETAILED PARAMETERS
0xBDB30000	Num BTag0	Number of BTags in a Pool	See page 662
0xBDB30004	Num BTag1		
0xBDB30008	Num BTag2		
0xBDB3000C	Num BTag3		
0xBDB30010	Num BTag4		
0xBDB30014	Num BTag5		
0xBDB30018	Num BTag6		
0xBDB3001C	Num BTag7		
0xBDB30020	Num BTag8		
0xBDB30024	Num BTag9		
0xBDB30028	Num BTag10		
0xBDB3002C	Num BTag11		
0xBDB30030	Num BTag12		
0xBDB30034	Num BTag13		
0xBDB30038	Num BTag14		
0xBDB3003C	Num BTag15		
0xBDB30040	Num BTag16		
0xBDB30044	Num BTag17		
0xBDB30048	Num BTag18		
0xBDB3004C	Num BTag19		
0xBDB30050	Num BTag20		
0xBDB30054	Num BTag21		
0xBDB30058	Num BTag22		
0xBDB3005C	Num BTag23		
0xBDB30060	Num BTag24		
0xBDB30064	Num BTag25		
0xBDB30068	Num BTag26		
0xBDB3006C	Num BTag27		

Table 216 BMU Registers (continued)

ADDRESS	REGISTER NAME	FUNCTION	DETAILED PARAMETERS
0xBDB30070	Num BTag28	Number of BTags in a Pool (continued)	See page 662
0xBDB30074	Num BTag29		
0xBDB40000	Memory Size	Physical Memory Configuration, Test and Debug	See page 663
0xBDB40008	SDRAM Config		See page 664
0xBDB4000C	Single ECC Errors		See page 665
0xBDB40010	ECC Enable and Test Enable		See page 665
0xBDB40014	Debug Config		See page 666
0xBDB40018	Wr_Mem_Violation_Hi		See page 667
0xBDB4001C	Wr_Mem_Violation_Lo		See page 667

BMU Detailed Descriptions

The following is a detailed description of each of the BMU registers and their individual parameters. The detailed parameters provide: purpose, field name, bit positions and descriptions.

Pool0 Base to Pool29 Base Registers (Buffer Pool Base Address Function)

Buffer pools must be configured during system initialization. Unpredictable behavior results when a pool is accessed prior to its initialization. The following registers are used to initialize buffer pools.

Purpose — Buffer pool base address. Width depends upon physical memory size: minimum value is 0, maximum value is the physical memory limit.

Software is responsible for ensuring that there is enough space to hold all of the pool's buffers.

Note: That the buffer pool "ends" at the next allocated piece of memory.

Address — 0xBDB00000 - 0xBDB00074

Access — Global Read/Write

Bit Position	31	24	23	0
Field Name	Rsvd		Pool Base	

Pool0 BTag Shift to Pool29 BTag Shift Registers (Buffer Size for a Pool Function)

Purpose BTag shift amount for address calculation. This value encodes the buffer size for a pool. Table 217 lists legal buffer sizes and their encodings.

Minimum value is 0. Maximum value is 10.

Address 0xBDB10000 - 0xBDB10074

Access Global Read/Write

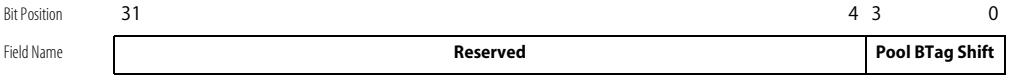


Table 217 BTag Shift Values and Corresponding Buffer Sizes

BTAG SHIFT	BUFFER SIZE	BTAG SHIFT	BUFFER SIZE
0	64kB	6	1kB
1	32kB	7	512B
2	16kB	8	256B
3	8kB	9	Not Supported
4	4kB	10	64B
5	2kB		

BTag FIFO Base0 to BTag FIFO Base29 Registers (BTag FIFO Base Address Function)

Purpose Buffer pool BTag FIFO base address. Used for BTag FIFO management.

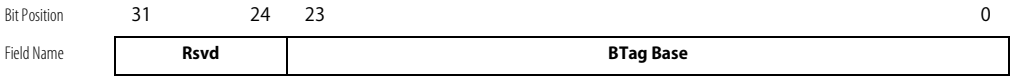
Software is responsible for ensuring that there is enough space to hold all of the pool's BTags.

Note: That the FIFO "ends" at the next allocated piece of memory. Each BTag consumes two bytes of memory.

Minimum value is 0. Maximum value is the physical memory limit.

Address 0xBDB20000 - 0xBDB20074

Access Global Read/Write

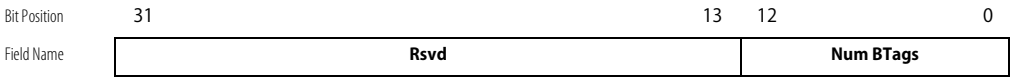


Num BTags0 to Num BTags29 Registers (Number of BTags in a Pool Function)

Purpose Number of BTags in pool, in multiples of eight. Default value of 0 = 0 BTags allocated.

Address 0xBDB30000 - 0xBDB30074

Access Global Read/Write



Memory Size Register (Miscellaneous Function)

Purpose — Physical memory size in bytes. Software determines the amount of physical memory by writing and reading bit patterns to SDRAM. This configuration register is written with a value representing the amount of physical memory that software had determined that was present in the system.

Address — 0xBDB40000

Reset Value — 10

Access — Global Read/Write

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:2	Read as zero.
phyMemSize	1:0	**Physical Memory Size** — Size of physical memory in bytes. Legal ranges are detailed here: ENCODED VALUE / SIZE 00 — 64MB 01 — 128MB 10 — 256MB 11 — Reserved

SDRAM Config Register (Miscellaneous Function)

Purpose SDRAM controller configuration register.

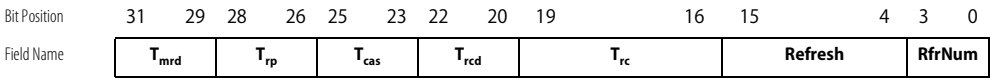
A write to this register tells the SDRAM controller the timing properties of the SDRAM and also initiates the SDRAM configuration process.

Note: That SDRAMs require a manufacturer specific initial settling time before the configuration process can begin. It is software's responsibility to ensure that this time has elapsed before the SDRAM configuration register is written.

Address 0xBDB40008

Reset Value 0

Access Global Read/Write



FIELD NAME	BIT POSITION	DESCRIPTION
T_{mrd}	31:29	**Timing Mode Register Delay**
T_{rp}	28:26	**Precharge Command Period Timing**
T_{cas}	25:23	**CAS Timing** — CAS timing.
T_{rcd}	22:20	**Timing From Active to Command** — Active to command timing.
T_{rc}	19:16	**Timing RAS Cycle Time** — RAS cycle time.
Refresh	15:4	**Refresh Period** — Refresh rate of SDRAM. For Micron SDRAM memory parts, this value is computed as 15.625 µsec times the clock rate for the memory. For example: for 100MHz Micron SDRAM parts, this value must be less than or equal to 1562.
RfrNum	3:0	**Refresh Number** — Number of initial refreshes.

Single ECC Errors Register (Miscellaneous Function)

Purpose This read only register counts the number of single Error Correction Code (ECC) errors that have occurred.

Address 0xBDB4000C

Reset Value 0

Access Global Read

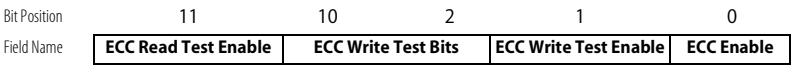
ECC Enable and Test Enable Register (Miscellaneous Function)

Purpose During normal operation, the Single Error Correction/Double Error Detecting (SECDED) error code if bit [0] is set to 1. ECC is disabled if bit [0] is set to 0. ECC test modes are controlled by bits [11:1].

Address 0xBDB40010

Reset Value 0

Access Global Read/Write



FIELD NAME	BIT POSITION	DESCRIPTION
ECC Read Test Enable	11	**ECC Read Test Enable** – This enables the ECC read test function, placing ECC bits directly on the Payload Bus.
ECC Write Test Bits	10:2	**ECC Write Test Bits** – Provides ECC bits for ECC write test.
ECC Write Test Enable	1	**ECC Write Test Enable** –This bit enables the ECC write test function, writing ECC write test bits directly to SDRAM.
ECC Enable	0	**ECC Enable** – This bit enables ECC checking during normal operation.

Debug Config Register (Miscellaneous Function)

Purpose BMU C-5e NP debug register in canonical format.

Address 0xBDB40014

Access Global Read/Write

Bit Position	31	30 28	27 24	23	22 20	19 16	15	14 12	11 8	7 6	4 3 0
Field Name	Enb0	Rsvd	MUX0	Enb1	Rsvd	MUX1	Enb2	Rsvd	MUX2	Enb3	Rsvd	MUX3
Reset Value	0	raz	0	0	raz	0	0	raz	0	0	raz	0

Table 218 BMU Debug Inputs

MUX/V VALUE	EVENT CHOSEN
15	Payload read
14	Payload write
13	Global read
12	Global write
11	BTag read
10	BTag write
9	BTag deallocation
8	Counter allocation
7	Counter decrement
6	Global read to SDRAM
5	Global write to SDRAM
4	BTag write to SDRAM
3	BTag deallocation to SDRAM
2	Counter decrement deallocation to SDRAM
1	BTag read to SDRAM
0	Write causing a memory violation

Wr_Mem_Violation_Hi Register (Miscellaneous Function)

Purpose Captures the global or payload address of transactions that led to the write error. Used in conjunction with Wr_Mem_Violation_Lo register.

Address 0xBDB40018

Access Global Read

Bit Position	7	6	5 ... 0
Field Name	**Error**	**Bus**	**Payload_Addr_Ctl [37:32]**
Reset Value	**0**	**0**	**0**

FIELD NAME	BIT POSITION	DESCRIPTION
Error	7	**Error** — 1=Error, 0=No error.
Bus	6	**Bus** — 1=Global bus, 0=Payload bus.
Bus Addr	0:5	**Bus Addr** — Records the high order of the Payload bus error bits that caused the error.

Wr_Mem_Violation_Lo Register (Miscellaneous Function)

Purpose Captures the global or payload address of transactions that led to the write error. Used in conjunction with Wr_Mem_Viloation_Hi.

Address 0xBDB4001C

Access Global Read

Bit Position	31 ... 0
Field Name	**Global_Addr [31:0] or Payload_Addr [31:0]**
Reset Value	**0**

FIELD NAME	BIT POSITION	DESCRIPTION
Bus Addr	31:0	**Bus Addr** — Records the Global or Payload bus that caused the error. If Payload this register capture only the low order bits. Where as, the high order are recorded in the *Wr_Mem_Violation_Hi* register.

Fabric Processor (FP) Configuration Registers

Configuration Space in the FP is an area that contains a number of registers. The FP uses these registers to communicate with the SDP and the bus controllers (Payload Bus and Global Bus). The FP performs flow mapping and management to and from the switching fabric.

FP Registers

The following is a list of each FP register and its address, function, and reference to its detailed parameters. The detailed parameters provide: purpose, field name, bit positions and descriptions. You should also refer to "TxByte Processor's Memory Space and Registers" on page 194.

Table 219 Fabric Processor Registers

ADDRESS	REGISTER NAME	FUNCTION	DETAILED PARAMETERS
0xBDD04000	TxFP_Enable	FP Tx Enable	See page 672
0xBDD04004	TxFI_Configuration	FP Tx Configuration	See page 673
0xBDD04008	TxDescInfo		See page 675
0xBDD0400C	TxDM_Header/Payload Delimiter		See page 675
0xBDD04010	TxQueueWeight_Configuration		See page 676
0xBDD04014	TxSysConfig		See page 678
0xBDD04018	TxFI_CRC		See page 678
0xBDD0401C	TxFCE_Configuration		See page 679
0xBDD04020	TxFP_Debug_Mux_Control	FP Tx Debug	See page 681
0xBDD04024	TxWCS_CAM	FP TxWCS CAM	See page 683
0xBDD0402C	TxFlowTbl	FP Tx Debug	See page 684
0xBDD04030	TxFlowTbl_Data_Low		See page 684
0xBDD04034	TxFlowTbl_Data_High		See page 685
0xBDD04038	TxFlowCAM	FP Tx Configuration	See page 685
0xBDD0403C	TxMergeAddr	FP Tx Debug	See page 687
0xBDD04040	TxMergeData		See page 687
0xBDD04044	TxIdleData	FP Tx Configuration	See page 688
0xBDD04048	TxByte_Ctl0	FP TxByte General Purpose	See page 688
0xBDD0404C	TxByte_Ctl1		See page 689

Table 219 Fabric Processor Registers (continued)

ADDRESS	REGISTER NAME	FUNCTION	DETAILED PARAMETERS
0xBDD04050	TxDebug_Internal_State	FP Tx Debug	See page 689
0xBDD04054	Absolute Priority_Configuration	FP Tx Configuration	See page 690
0xBDE04000 to 0xBDE0407F	RxExtractSpace0	FP RxByte Processor0	See page 691
0xBDE04090	RxStatus0		See page 692
0xBDE04094	RxFlowSeg0		See page 693
0xBDE04098	RxFlowSize0		See page 694
0xBDE0409C	RxTxCgs0		See page 695
0xBDE04200 to 0xBDE0427F	RxExtractSpace1	FP RxByte Processor1	See Table 221 on page 691
0xBDE04290	RxStatus1		See Table 222 on page 692
0xBDE04294	RxFlowSeg1		See Table 223 on page 694
0xBDE04298	RxFlowSize1		See Table 224 on page 694
0xBDE0429C	RxTxCgs1		See Table 225 on page 695
0xBDE04600	RxFP_Enable	FP Rx Enable	See page 696
0xBDE04604	RxFI_Configuration	FP Rx Configuration	See page 696
0xBDE04608	RxDS_Header_Change1		See page 699
0xBDE0460C	RxDS_Header_Change2		See Table 226 on page 699
0xBDE04610	RxDS_Header/Payload_Delimiter0		See page 700
0xBDE04614	RxDS_Header/Payload_Delimiter1		See Table 227 on page 700
0xBDE04618	RxDS_Header/Payload_Delimiter2		See Table 227 on page 700
0xBDE0461C	RxDS_Configuration		See page 701
0xBDE04620	RxFI_CRC		See page 703

Table 219 Fabric Processor Registers (continued)

ADDRESS	REGISTER NAME	FUNCTION	DETAILED PARAMETERS
0xBDE04624	RxWCS_CAM	FP RxWCS CAM	See page 704
0xBDE04628	RxByte0 General Purpose Configuration	FP Rx Configuration	See page 705
0xBDE0462C	RxByte1 General Purpose Configuration		See Table 228 on page 706
0xBDE04630	RxFCE_Configuration0		See page 706
0xBDE04634	RxFCE_Configuration1		See page 707
0xBDE04638	RxFCE_Configuration2		See page 709
0xBDE04640	Pool0_CFG0	FP Rx Pool Configuration	See page 710
0xBDE04648	Pool1_CFG0		See Table 229 on page 710.
0xBDE04650	Pool2_CFG0		
0xBDE04658	Pool3_CFG0		
0xBDE04644	Pool0_CFG1		See page 711
0xBDE0464C	Pool1_CFG1		See Table 230 on page 711.
0xBDE04654	Pool2_CFG1		
0xBDE0465C	Pool3_CFG1		
0xBDE04660	RxByte_Shared_Low	FP RxByte Shared	See page 712
0xBDE04664	RxByte_Shared_High		See page 712
0xBDE04680	RxFP_Interrupt_Event	FP Rx Interrupt	See page 713
0xBDE04684	RxFP_Interrupt_Enable		See page 714
0xBDE04688	RxFP_Debug_Mux_Control	FP Rx Debug	See page 714
0xBDE04690	RxMemory_Address		See page 717
0xBDE04694	RxMemory_Data		See page 717
0xBDE04698	RxPDU_ID_CAM		See page 718
0xBDE046A0	SEGs_Rcvd	FP Rx Statistics	See page 719.
0xBDE046A4	PDUs_Rcvd		
0xBDE046A8	SEGs_Lost		
0xBDE046AC	PDUs_Lost		

Table 219 Fabric Processor Registers (continued)

ADDRESS	REGISTER NAME	FUNCTION	DETAILED PARAMETERS
0xBDE046C0	CParity_Err	FP Rx Statistics (continued)	See Table 232 on page 719
0xBDE046C4	Err_Hdr		
0xBDE046C8	Parity_Err		
0xBDE046CC	Length_Err		
0xBDE046D0	Reserved		
0xBDE046D4	CRC_Err		
0xBDE046D8	Odd_PDU		
0xBDE046DC	Seq_Err		
0xBDE046E0	Seq_Dis		
0xBDE046E4	Lost_PDU		
0xBDE046E8	No_Flow_Tbl		
0xBDE046EC	No_BTag		
0xBDE046F0	BTag_Err		
0xBDE046F4	Alloc_Err		
0xBDE046F8	Enque_Err		
0xBDE04700	RxDebug_Internal_State	FP Rx Debug	See page 722

FP Details Descriptions

The following is a detailed description of each of the FP registers and their individual parameters. The detailed parameters provide: purpose, field name, bit positions and descriptions.

TxFP_Enable Register (FP Tx Enable Function)

The transmit path is enabled via the *TxEnable* register.

The FPTx cannot be re-enabled after it has been disabled.

Purpose	Provides TxFP enable/disable.
Address	0xBDD04000
Access	Global Read/Write

Bit Position	31	30 ... 0
Field Name	**Enable**	**Reserved**
Reset Value	**0**	**0**

FIELD NAME	BIT POSITION	DESCRIPTION
Enable	31	**Tx Fabric Port Enable** — 1 enables the internal FPTx logic; 0 disables the FPTx and holds it in reset.
Reserved	30:0	Read/write.

TxFI_Configuration Register (FP Tx Configuration Function)

Purpose Allows physical configuration of the TxFP Interconnect.

Address 0xBDD04004

Access Global Read/Write

Bit Position	31	30 26	25	24	23	22	21	20	19	18	17	16	15 8	7	6 5	4	3	2	1	0
Field Name	CFIEnable	Rsvd	U2PHY Tri Enable	U2Mode	Variable CellSize	Rsvd	PRIZMA	Rsvd	PowerX	IdleCell	Rsvd	Rsvd	SEG Size	Rsvd	Bus Width	BigEnd	ATM	OddP	Rsvd	RegIn
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

FIELD NAME	BIT POSITION	DESCRIPTION
CFIEnable	31	**CFI Enable** — 1 enables the external Freescale Fabric Interface; 0 disables (external pins are tri-stated).
Reserved	30:26	Read/write
U2PHYTriEnable	25	**UTOPIA2PHYTri-state Enable** — Enables tri-stating of FPTx output pins. Should be set to 1 when the FPTx is configured for U2PHY mode. Must be set to 0 for U2ATM and all other modes.
U2Mode	24	**UTOPIA2 Mode Enable** — 1 enables UTOPIA2 mode; 0 disables UTOPIA2 mode.
VariableCellSizes	23	**Variable Cell Size Enable** — 1 enables variable length cells; 0 disables.
Reserved	22	Read/write
PRIZMA	21	**PRIZMA Mode Enable** — 1 enables PRIZMA fabric mode; 0 disables the PRIZMA fabric mode. Cannot be set with PowerX bit [19].
Reserved	20	Read/write
PowerX	19	**PowerX (CSIX-L0) Mode Enable** — 1 enables PowerX mode; 0 deselects PowerX mode. Cannot be set with PRIZMA bit [21].
IdleCell	18	**Idle Cell Enable** — 1 generates idle cell when the transmit FIFO is empty; 0 inhibits generation of idle cell. Note: Idle cells are *only* supported for PRIZMA mode, and *must* be set in PRIZMA mode.
Reserved	17	Read/write
Reserved	16	Must be set to a 1.

<table>
<tr><th>FIELD NAME</th><th>BIT POSITION</th><th>DESCRIPTION</th></tr>
<tr><td>SEG Size</td><td>15:8</td><td>Segment Size — Configures segment size of fabric (legal values are from 40 to 204Bytes).
Note: The segment size must be a multiple of four Bytes (1 word). Unpredictable results may occur for values outside of this range or non word aligned.</td></tr>
<tr><td>CSIX-L1</td><td>7</td><td>CSIX-L1 Mode Enable — 1 enables CSIX-L1 mode; 0 deselects CSIX-L1 mode.</td></tr>
<tr><td>Bus Width</td><td>6:5</td><td>Bus Width — Specifies Fabric bus width. Legal values are detailed here:
<table><tr><th>ENCODED VALUE</th><th>WIDTH (BITS)</th></tr><tr><td>00</td><td>8</td></tr><tr><td>01</td><td>16</td></tr><tr><td>10</td><td>undefined</td></tr><tr><td>11</td><td>32</td></tr></table></td></tr>
<tr><td>BigEnd</td><td>4</td><td>Select Big Endian — 1 selects big endian; 0 selects little endian.</td></tr>
<tr><td>ATM</td><td>3</td><td>ATM — Selects ATM or PHY mode for UTOPIA. 1 selects ATM; 0 selects PHY.</td></tr>
<tr><td>OddP</td><td>2</td><td>Odd Parity — 1 selects odd parity; 0 selects even parity.</td></tr>
<tr><td>Reserved</td><td>1</td><td>Read/write</td></tr>
<tr><td>RegIn</td><td>0</td><td>Select Registered Inputs — 1 selects registered inputs; 0 selects non-registered inputs. Configurations are detailed here based on the mode:
<table><tr><th>SETTING</th><th>MODE</th></tr><tr><td>0</td><td>UTOPIA2</td></tr><tr><td>1</td><td>UTOPIA3, CSIX-L1, PRIZMA, PowerX</td></tr></table>Note: Must be configured to 0 for UTOPIA2 mode. Must be a 1 for UTOPIA3 and PRIZMA. This bit is a “don’t care” for PowerX(CSIX-L0) and CSIX-L1 modes since there are no inputs to the FPTx., however, it is recommended to set the bit to a 1.</td></tr>
</table>

TxDescInfo Register (FP Tx Configuration Function)

Purpose Describes descriptor layout allowing TxFCE to extract key information.
Address 0xBDD04008
Access Global Read/Write

Bit Position	31 24	23 16	15 8	7 0
Field Name	Multicast Position	Length Position	Pool Position	BTag Position
Reset Value	0	0	0	0

FIELD NAME	BIT POSITION	DESCRIPTION
Multicast Position	31:24	**Multicast Position** — Offset bit position in descriptor of multicast bit.
Length Position	23:16	**Length Position** — Offset bit position in descriptor of PDU length.
Pool Position	15:8	**Pool Position** — Offset bit position in descriptor of Pool ID.
BTag Position	7:0	**BTag Position** — Offset bit position in descriptor of BTag.

TxDM_Header/Payload Delimiter Register (FP Tx Configuration Function)

Purpose Used by the Data Merge hardware to prepend the header to the payload.
Address 0xBDD0400C
Access Global Read/Write

Bit Position	31 29	28 24	23 16	15 8	7 0
Field Name	Reserved	MinSOF-SOF	IdleCellLen	HeaderLen2	HeaderLen1
Reset Value	0	0	0	0	0

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:29	Read/write
MinSOF-SOF Spacing	28:24	**Minimum Cell Size** — Specifies minimum Start of Frame (SOF)-to-SOF timing, in terms of fabric clocks for short cells in PowerX mode.
IdleCellLen	23:16	**Idle Cell Length** — Length of Idle Cell in bus cycles. (Used in conjunction with *TxFI Configuration* register's *Idle Cell Enable* field and the *Idle Cell Header* register). Only in PRIZMA mode.
HeaderLen2	15:8	**Header Length 2** — Length of middle and last segment headers in Bytes.

FIELD NAME	BIT POSITION	DESCRIPTION
HeaderLen1	7:0	**Header Length 1** — Length of first and only segment headers in Bytes.

TxQueueWeight_Configuration Register (FP Tx Configuration Function)

Purpose A globally accessible register that allows the configuration of FP weight counters and minimum quantum counters.

Address 0xBDD04010

Access Global Read/Write

Bit Position	31 17	16	15	14 11	10 7	6 0
Field Name	Reserved	Write	Reserved	MQShiftVal	WgtCtrVal	QueueNumber
Reset Value	raz	0	raz	1111	0001	0

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:17	Read as zero.
Write	16	**Write** — Setting this bit to a 1 causes the *MQShiftVal* [14:11] and *WgtCtrVal* [10:7] to be written into the FPTx weighting algorithm memory for the associated Queue Number. To fully configure the weighting algorithm, perform one (1) write for each queue (up to 128). This *Write* bit [16] is automatically cleared by hardware after each assertion. NOTE: Write to both fields, *MQShiftVal* [14:11] and *WgtCtrVal* [10:7] for a given queue, to make a change to either field.
Reserved	15	Read as zero.

<table>
<tr><th>FIELD NAME</th><th>BIT POSITION</th><th>DESCRIPTION</th></tr>
<tr><td>MQShiftVal</td><td>14:11</td><td>Minimum Quantum Shift Value — This field contains the amount to shift the initial weight counter value (WgtCtrVal [10:7]) for the minimum quantum counter of the queue specified. Legal ranges are detailed here:
<table>
<tr><th>ENCODED VALUE</th><th>RESULTING MINIMUM QUANTUM COUNTER VALUE</th></tr>
<tr><td>0000</td><td>0 WgtCtrVal >>0</td></tr>
<tr><td>0001</td><td>1 WgtCtrVal >>1</td></tr>
<tr><td>0010</td><td>2 WgtCtrVal >>2</td></tr>
<tr><td>•</td><td>•</td></tr>
<tr><td>•</td><td>•</td></tr>
<tr><td>1111</td><td>15 WgtCtrVal >>15</td></tr>
</table></td></tr>
<tr><td>WgtCtrVal</td><td>10:7</td><td>Weight Counter Value — This field contains the initial value for the weight counter (also the refresh value) for the queue specified. Legal ranges are detailed here:
<table>
<tr><th>ENCODED VALUE</th><th>WEIGHT COUNTER (16K*N BYTES)</th></tr>
<tr><td>0001</td><td>16K</td></tr>
<tr><td>0010</td><td>32K</td></tr>
<tr><td>0011</td><td>48K</td></tr>
<tr><td>•</td><td>•</td></tr>
<tr><td>•</td><td>•</td></tr>
<tr><td>1111</td><td>16K * 15 (240KBytes)</td></tr>
</table></td></tr>
<tr><td>QueueNumber</td><td>6:0</td><td>Queue Number — Number of the queue to be accessed.</td></tr>
</table>

TxSysConfig Register (FP Tx Configuration Function)

Purpose Sets the base queue number for the FP.

Address 0xBDD04014

Access Global Read/Write

Bit Position	31 25	24 16	15 0
Field Name	Reserved	QueueOffset	FabricID
Reset Value	0	0	0

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:25	Read/write
Queue ffset	24:16	**Queue Offset** — Base queue number for FP block of 128 queues in QMU.
FabricID	15:0	Reserved

TxFI_CRC Register (FP Tx Configuration)

Purpose The *TxFI_CRC* register configures the CRC function. The result can be configured in two (2) ways:

- Initial value of the CRC accumulator (0 or all 1s)
- Inverted (one's complement)

Also, the *TxFI_CRC* provides index fields that allow the CRC to be calculated over any sequential portion of the segment, and then appended or inserted anywhere afterward.

Address 0xBDD04018

Access Global Read/Write

Bit Position	31	30 28	27	26	25	24	23 16	15 8	7 0
Field Name	Enable	Reserved	Rsvd	CRC Ini1	Rsvd	Invert	FirstIndex	LastIndex	Append Index
Reset Value	0	0	1	0 or 1	0	0 or 1	0	0	0

FIELD NAME	BIT POSITION	DESCRIPTION
Enable	31	**CRC Enable** — 1 enables CRC mechanism; 0 disables and leaves the CRC mechanism in reset.
Reserved	30:28	Read/write

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	27	Must be set to 1.
Init1	26	**CRC Initialization** — 1 initializes the CRC register to all 1s; 0 initializes it to a 0.
Reserved	25	Must be set to 0
Invert	24	**Invert CRC** — 1 selects CRC to be inverted prior to being appended to Segment; 0 selects not inverted.
FirstIndex	23:16	**First Index** — Offset from Byte 0 of segment to start of CRC accumulation Byte. FirstIndex must be a multiple of 4.
LastIndex	15:8	**Last Index** — Must be equal to (cell size - 4). This represents the offset (plus 1 byte) from byte 0 of segment to the last byte to be part of the CRC accumulation.
Append Index	7:0	**Append Index** — Byte offset from byte 0 of segment to appended CRC (currently not supported).

TxFCE_Configuration Register (FP Tx Configuration Function)

Purpose Configures FCE descriptor size, queue configuration, and flow mask.

Address 0xBDD0401C

Access Global Read/Write. Bits [31:28] are read only from a global perspective; they are written by FP hardware.

Bit Position	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	0
Field Name	QMU Parity Error	RdError	WrError	QMU Error	IntAck	IntEnable	DescSize	Rsvd			QMU Parity Error Enable	PDU Pause	FCEnab.	Queue Depth	Rsvd	Debug Shift Enable	Flow Mask	
Reset Value	0	0	0	0	0	0	0	0			0	0	0	0	1	0	0	

FIELD NAME	BIT POSITION	DESCRIPTION
QMUParityError	31	**QMU Parity Error** — Indicates QMU dequeue parity error status.
RdError	30	**Read Error** — Indicates the TxFCE Read Control Block transfer failed.
WrError	29	**Write Error** — Indicates the TxFCE Write Control Block transfer failed.
QMUError	28	**QMU Error** — Indicates the TxFCE dequeue operation failed.
IntAck	27	**Interrupt Acknowledge** — Set to a 1 to acknowledge and clear an interrupt, then set to 0.

<table>
<tr><th>FIELD NAME</th><th>BIT POSITION</th><th>DESCRIPTION</th></tr>
<tr><td>IntEnable</td><td>26</td><td>Interrupt Enable — 1 enables interrupts to be generated to the XP for the following errors:
• QMU parity error, bit [31] of this register.
• Read error, bit [30] of this register.
• Write error, bit [29] of this register.
• QMU error, bit [28] of this register.</td></tr>
<tr><td>DescSize</td><td>25:24</td><td>Descriptor Size — Legal ranges are detailed here:<table><tr><th>ENCODED VALUE</th><th>SIZE (BYTES)</th></tr><tr><td>00</td><td>12</td></tr><tr><td>01</td><td>16</td></tr><tr><td>10</td><td>24</td></tr><tr><td>11</td><td>32</td></tr></table></td></tr>
<tr><td>Reserved</td><td>23:22</td><td>Read/write</td></tr>
<tr><td>QMU Parity Error Enable</td><td>21</td><td>QMU Parity Error Enable — 1 enables QMU dequeue parity error checking, 0 disables parity checking.</td></tr>
<tr><td>PDU Pause</td><td>20</td><td>Reserved. Must be set to zero</td></tr>
<tr><td>FCEnable</td><td>19</td><td>Flow Control Enable — 1 enables per-queue flow control; 0 disables per-queue flow control.</td></tr>
<tr><td>Queue Depth</td><td>18</td><td>Queue Depth — 1 selects 16 queues x eight priorities; 0 selects 32 queues x four priorities.</td></tr>
<tr><td>Reserved</td><td>17</td><td>Must be set to 1.</td></tr>
</table>

FIELD NAME	BIT POSITION	DESCRIPTION
Debug Shift Enable	16	**Debug Shift Enable** — Must be set =0. When this bit is set (=1) the weight counter and minimum quantum counter values are shifted right by eight (8) on initialization and on refresh. Note: This is a Test-Only mode for debug purposes.
Flow Mask	15:0	**Flow Mask** — Used to mask 'x' bits of 16bit Flow ID during CAM operation on congestion messages. When matching entries in the Tx Flow ID CAM using the *TxFlowCAM* register (0xBDD0438), you must be sure to also set this register's *Flow Mask* field. The value of the *Flow Mask* is *AND'ed* with the value of the *TxFlowCAM* register's *Match* field to obtain the result submitted to the CAM. The default value for the *Flow Mask* is zero. Hence failing to set the *Flow Mask* means you will never match any entry in the CAM.

TxFP_Debug_Mux_Control Register (FP Tx Debug Function)

For the purposes of debug, most of the internal registers are made visible via Global Address Space, and a limited number of events (usually used to count) can be viewed via the *TxFP_Debug_Mux_Control* register.

Purpose — Allows you to monitor events for FPTx debug purposes. These events are selectable. See Table 220 on page 682. Any event can be viewed in association with any of the four (4) selection fields, including simultaneously being selected in more than one field (that is, viewed multiple times).

For the purposes of debug, specific monitoring points with in the FPTx are wired to the event register as selected by the *TxFP_Debug_Mux_Control* register.

Address — 0xBDD04020

Access — Global Read, TxByte Processor Read/Write,

Bit Position	31	30 28	27 24	23	22 20	19 16	15	14 12	11 8	7	6 4	3 0
Field Name	EN0	RSVD	SEL0	EN1	RSVD	SEL1	EN2	RSVD	SEL2	EN3	RSVD	SEL3
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0

FIELD NAME	BIT POSITION	DESCRIPTION
EN0	31	**TxDebug Event Mux Control Enable 0** — 1 enables the associated selected events; 0 disables the associated event from being viewed.
Reserved	30:28	Read/write
SEL0	27:24	**TxDebug Event Mux Control Select 0** — Selects one (1) of the eight (8) FPTx events to be viewed for the corresponding field.
EN1	23	**TxDebug Event Mux Control Enable 1** — 1 enables the associated selected events; 0 disables the associated event from being viewed.
Reserved	22:20	Read/write
SEL1	19:16	**TxDebug Event Mux Control Select 1** — Selects one (1) of the eight (8) FPTx events to be viewed for the corresponding field.
EN2	15	**TxDebug Event Mux Control Enable 2** — 1 enables the associated selected events; 0 disables the associated event from being viewed.
Reserved	14:12	Read/write
SEL2	11:8	**TxDebug Event Mux Control Select 2** — Selects one of the eight (8) FP Tx events to be viewed for the corresponding field.
EN3	7	**TxDebug Event Mux Control Enable 3** — 1 enables the associated selected events; 0 disables the associated event from being viewed.
Reserved	6:4	Read/write
SEL3	3:0	**TxDebug Event Mux Control Select 3** — Selects one (1) of the eight (8) FPTx events to be viewed for the corresponding field.

Table 220 FPTx_Debug Monitored Events

SELECT VALUE	MONITORED EVENT	DESCRIPTION
0	Per-queue pause	Pulses once per per-queue pause request from the FPRx.
1	Cell complete	Pulses once per cell.
2	PDU complete	Pulses once per complete PDU transmitted.
3	DMA request	Pulses once per payload DMA.
4	PDU resume	Pulses once per PDU resumed after a per -queue pause.

Table 220 FPTx_Debug Monitored Events (continued)

SELECT VALUE	MONITORED EVENT	DESCRIPTION
5	Fabric FIFO empty	Active for every clock that the transmit FIFO is empty.
6	Send pause	Pulses once per FPRx request to send a link-level pause.
7	Pause	Pulses once per FPRx request to link-level pause.

TxWCS_CAM (Tx WCS CAM Function)

Purpose Interface in global address space to initialize FP TxByte Processor's WCSs and CAMs.

Address 0xBDD04024

Access Global Read / Write (bits [7:6] are read only)

Bit Position	31 16	15 8	7 6	5	4	3	2	1	0
Field Name	Reserved	WCSWriteData	WCSScanOut	WCSWriteCmd	CAMReset	CAMUpdate	WCS/CAM Capture	WCS ATA IN1	WCSDATA IN0
Reset Value	0	0	0	0	0	0	0	0	0

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:16	Read/write
WCSWriteData	15:8	**TxWCS_CAM WCS Write** — Data to be written to both WCSes via the Byte write interface.
WCSScanOut	7:6	**TxWCS_CAM WCS Scan** — Read only. WCS scan shift out for Byte processors 1 & 0 (in that order).
WCSWriteCmd	5	**TxWCS_CAM WCS Write Cmd** — Setting this to a 1 launches a Byte write to both WCSes. Cleared by hardware.
CAMReset	4	**TxWCS_CAM Reset** — Setting this to a 1 resets the TxByte CAM.
CAMUpdate	3	**TxWCS_CAM Update** — Setting this to a 1 updates the CAM array from the CAM's shift registers. Cleared by hardware.
WCS/CAMCapture	2	**TxWCS_CAM WCS/CAM Capture** — Setting this to a 1 launches a WCS and CAM scan capture for diagnostic purposes (loads data into the WCS and CAM shift registers). Cleared by hardware.
WCSDATAIN1	1	**TxWCS_CAM WCS DATAIN1** — Setting this to a 1 shifts a 1 into the WCSes. Cleared by hardware.

FIELD NAME	BIT POSITION	DESCRIPTION
WCSDATAIN0	0	**TxWCS_CAM WCS DATAIN0** — Setting this to a 1 shifts a 0 into the WCSes. Cleared by hardware.

TxFlowTbl Register (FP Tx Debug Function)

Purpose Allows access to global address space read flow table inside the TxFCE.

Address 0xBDD0402C

Access Global Read/Write, 128 60bit entries

Bit Position	31 … 17	16	15 … 8	7 … 0
Field Name	Reserved	WT	Reserved	ADDR
Reset Value	0	0	0	0

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:17, 15:8	Read/write.
WT	16	**Write Flow Table** — Writing a 1 initiates a flow table write. This bit is automatically cleared by hardware.
ADDR	7:0	**Flow Table Address** — Address of flow table to write or read. Note: Only the first128 entries are valid.

TxFlowTbl_Data_Low Register (FP Tx Debug Function)

Purpose Least significant word of Tx flow table data.

Address 0xBDD04030

Access Global Read/Write

Bit Position	31 … 0
Field Name	DATA_LOW
Reset Value	Undefined

FIELD NAME	BIT POSITION	DESCRIPTION
DATA_LOW	31:0	This is the low order data which was read from a flow table read, or the data to be written on a flow table write.

TxFlowTbl_Data_High Register (FP Tx Debug Function)

Purpose Most significant word of Tx flow table data.

Address 0xBDD04034

Access Global Read/Write

Bit Position	31 28	27 0
Field Name	Reserved	DATA_HIGH
Reset Value	raz	Undefined

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:28	Read As Zero (raz)
DATA_HIGH	27:0	This is the high order data which was read from a flow table read, or the data to be written on a flow table write.

TxFlowCAM Register (FP Tx Debug Function)

Purpose Interface to global address space to initialize the TxByte Processor's TxFlow CAM store.

Address 0xBDD04038

Access Global Read/Write

Bit Position	31 27	26	25	24	23 8	7 0
Field Name	Reserved	WT	DEL	SRCH	Match	CAM WT Data
Reset Value	raz	raz	raz	raz	raz (except bit 8)	0

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:27	Read as zero.
WT	26	**Write CAM Location** — This blt is always read as zero. Writes the location matched, or the next free location if nothing matches (for diagnostic purposes only). Setting this bit to a 1 launches a CAM write of the write data. After the CAM write, the bit is cleared by the hardware. 1 = write Cam entry 0 = do not write Cam entry

<table>
<tr><th>FIELD NAME</th><th>BIT POSITION</th><th>DESCRIPTION</th></tr>
<tr><td>DEL</td><td>25</td><td>Delete CAM Entry — This bit is always read as zero.Deletes CAM entry matched (for diagnostic purposes only). Setting this bit to a 1 launches a CAM delete of the entry corresponding to the previous match value. The Match [23:8] value must be set first, and then the DEL CAM [25] bit set with a separate write to this register. After the CAM write, the bit is cleared by the hardware.

<table>
<tr><th>ENCODED VALUE</th><th>CAM ACTION</th></tr>
<tr><td>1</td><td>Delete Cam entry</td></tr>
<tr><td>0</td><td>Do not delete Cam entry</td></tr>
</table>
</td></tr>
<tr><td>SRCH</td><td>24</td><td>CAM Search — 1 selects CAM search. This bit is always read as zero.</td></tr>
<tr><td>Match</td><td>23:8</td><td>16Bit CAM Match Value — Value to search on. On reads, bits 23:9 return zeros, while bit 8 returns the CAM free indication.
When matching entries in the Tx Flow ID CAM using this register, you must be sure to also set the TxFCE Configuration (0xBDD0438) register's Flow Mask field. The value of the Flow Mask is AND'ed with the value of the TxFlowCam register's Match field to obtain the result submitted to the CAM. The default value for the Flow Mask is zero. Hence, failing to set the Flow Mask means you will never match any entry in the CAM.</td></tr>
<tr><td>CAM WT Data</td><td>7:0</td><td>CAM Write Data — CAM data read from the CAM, or to be written to the CAM. Field is 6:0, but bit 7 is always set to 0.</td></tr>
</table>

TxMergeAddr Register (FP Tx Debug Function)

Purpose Used to select the Merge Block address and access (read/write). Bit [5] selects merge block 1/ 0, and bits [4:0] select the word offset (0-31).

Global Address 0xBDD0403C

Access Global Read, Global Write

Bit Position	31 … 17	16	15 … 8	5 … 0
Field Name	Reserved	WMB	Reserved	ADDR
Reset Value	raz	0	raz	0

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:17	Read as zero.
WMB	16	**Write Merge Block** — 1 selects write; 0 selects read.
Reserved	15:8	Read as zero.
ADDR	5:0	**Merge Space Address** — Address of Merge Space to write/read.

TxMergeData Register (FP Tx Debug Function)

Purpose Used to Write / Read Data from each Merge block.

Global Address 0xBDD04040

Access Global Read, Global Write

Bit Position	31 … 0
Field Name	DATA
Reset Value	0

TxIdleData Register (FP Tx Configuration Function)

Purpose	Provides data for idle cells.
Address	0xBDD04044
Access	Global Read/Write

This register is meaningful only in PRIZMA mode.

Bit Position	31	0
Field Name	Idle Cell Data	
Reset Value	0	

FIELD NAME	BIT POSITION	DESCRIPTION
Idle Cell Data	31:0	**Idle Cell Data** — Four Bytes of data for Idle cell generation. (Used in conjunction with the *TxFI Configuration* register's *Idle Cell Enable* field, and the *TxDM Header Payload Delimiter* register).

TxByte_Ctl0 Register (FP TxByte General Purpose Function)

Purpose	General purpose data which is accessible globally and by the TxByte Processor. There are two (2) 32bit registers (TxByte_CTL0, TxByte_CTL1).
Address	0xBDD04048
TxByte Processor Address	0x0A0 - 0x0A7
Access	Global Read/ Write, TxByte Processor Read

Bit Position	31	0
Field Name	Data	
Reset Value	0	

FIELD NAME	BIT POSITION	DESCRIPTION
Data	31:0	Global Read/Write, TxByte Processor Read

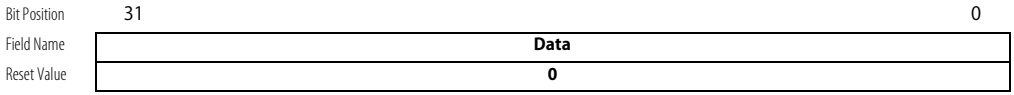
TxByte_Ctl1 Register (FP TxByte General Purpose Function)

Purpose General purpose data which is accessible globally and by the TxByte Processor. There are two (2) 32bit registers (TxByte_CTL0, TxByte_CTL1).

Address 0xBDD0404C

TxByte Processor Address 0x0A0 - 0x0A7

Access Global Read/ Write, TxByte Processor Read



FIELD NAME	BIT POSITION	DESCRIPTION
Data	31:0	Global Read/Write, TxByte Processor Read

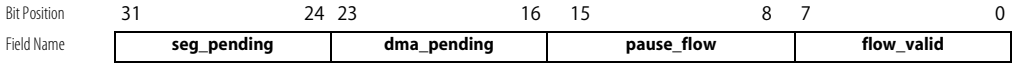
TxDebug_Internal_State Register (FP Tx Debug Function)

Purpose The *TxDebug_Internal_State* register has four (4) 8bit fields that provide a internal signal debug information for the eight (8) internal scopes.

While the FPTx handles 128 flows, only eight flows (flows 0 to 7) are *active* at any one time. Thus the bit fields indicate the state of each active flow (within each field the LSB identifies flow 0 and the MSB identifies flow 7).

Address 0xBDD04050

Access Global Read, Written by FPTx hardware



FIELD NAME	BIT POSITION	DESCRIPTION
seg_pending	31:24	**Segment Pending** — 1 indicates the flow(s) (0 to 7) have started to transmit a segment and are waiting for the segment transmit to complete prior to getting the PDU segment.
dma_pending	23:16	**DMA Pending** — 1 indicates the flow(s) (0 to 7) have requested a PDU segment to be DMA'ed from DRAM to DMEM and are currently awaiting the segment DMA to complete so that it can be transmitted.
pause_flow	15:8	**Pause Flow** — 1 indicates the flow(s) (0 to 7) is being paused (that is, flow controlled) and will be moved to the 'sleep' state to allow another ready flow to go *active*. Only asserted for one core clock cycle.
flow_valid	7:0	**Flow Valid** — 1 indicates the flow(s) (0 to 7) are actively transmitting. 0 indicates that the active flow slot is not being used, that is, there are no additional queues ready to be serviced.

Absolute Priority_Configuration Register (FP Tx Configuration Function)

Purpose A globally accessible register that allows enabling of absolute queues.

Address 0xBDD04054

Access Global Read/Write

FIELD NAME	BIT POSITION	DESCRIPTION
AbsPriEnable	31:0	**Absolute Priority Queue Enable** — 1= Enables corresponding port's absolute queue, 0= Disable corresponding port's absolute priority queue. For a 16x8 configuration: bit[0] is port 0, bit[15] is port 15, and bits [31:16] are unused. For a 32x4 configuration: bit[0] is port 0 and bit[31] is port 31.

RxExtractSpace0 Space (FP RxByte Processor0 Function)

Purpose	Used for Extracting data from the headers by RxByte0 Processor and storing it for building descriptors . See Table 221 on page 691 for similar register.
Address	0xBDE04000 to 0xBDE0407F
RxByte Processor Address	0x00 to 0x1F (for 4 scopes per RxByte Processor), or 0x00 to 0x0F (for 8 scopes per RxByte Processor).
Access	Global Read/Write, RxByte Processor Write, Hardware Read

Bit Position	31	0
Field Name	Data	
Reset Value	0	

Table 221 RxExtractSpace0 Space (for RxByte Processor1)

REGISTER NAME	PURPOSE	ADDRESS	RXBYTE PROCESSOR ADDRESS
RxExtractSpace1	Same as *RxExtractSpace0*, except for RxByte Processor1.	0xBDE04200 to 0xBDE0427F	• 0x00 to 0x1F (for 4 scopes per RxByte Processor), or • 0x00 to 0x0F (for 8 scopes per RxByte Processor).

RxStatus0 Register (FP RxByte Processor0 Function)

Purpose — Read/modified Writes governing RxByte Processor receive operation for datascope0. See Table 222 on page 692 for similar register.

Address — 0xBDE04090

RxByte Processor Address — 0x80 to 0x83

Access — Global Read/Write, RxByte Processor Read/Write, Hardware Read/Write

Bit Position	7	6	1	0
Field Name	Own	Reserved		Reserved
Reset Value	1	raz		0

FIELD NAME	BIT POSITION	DESCRIPTION
Own	7	**Extract Space Ownership** — 1 = FCE owns; 0 = RxByte Processor owns.
Reserved	6:1	Read as zero.
Reserved	0	**Reserved** — Must be set to 0.

Table 222 RxStatus1 Register (for RxByte Processor1)

REGISTER NAME	PURPOSE	ADDRESS	RXBYTE PROCESSOR ADDRESS
RxStatus1	Same as *RxStatus0*, except for RxByte Processor1.	0xBDE04290	0x80 to 0x83

RxFlowSeg0 Register (FP RxByte Processor Function)

The Flow Segment0 register is used by the RxByte Processor to associate a segment with a flow. Specifically, the RxByte Processor writes the Flow ID, Segment type, and whether the segment is part of a multicast flow.

Purpose — Associates a received Segment to a Flow. See Table 223 on page 694 for similar register.

Address — 0xBDE04094

RxByte Processor Address — 0x84 to 0x87

Access — Global Read/Write, RxByte Procesor Write, Hardware Read

Bit Position	31 27	26	25 24	23 18	17	16	15 0
Field Name	Reserved	Multicast	Seg Type	Reserved	FlowDisc	FlowError	PDU ID
Reset Value	raz	0	0	raz	0	0	0

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:27	Read as zero.
Mulitcast	26	**Multicast Enable** — Reserved. Must be set to 0.
Seg Type	25:24	**Segment Type** — Defines the Segment (PDU Segment). Legal values are detailed here: ENCODED VALUE / TYPE 00 — Continuation of Message (COM), (middle) segment 01 — End of Message (EOM), (last) segment 10 — Beginning of Message (BOM), (first) segment 11 — First and only Message (FOM), (first and last) segment
Reserved	23:18	Read as zero.
FlowDisc	17	**Flow Discard** — 1 = Discards flow; 0 = does not discard flow.

FIELD NAME	BIT POSITION	DESCRIPTION
FlowError	16	**Flow Error** — 1 = RxByte Processor detects error (packet discarded with error); 0 = RxByte Processor detects no error.
PDU ID	15:0	**PDU ID** — The 16bit PDU ID.

Table 223 RxFlowSeg1 Register (for RxByte Processor1)

REGISTER NAME	PURPOSE	ADDRESS	RXBYTE PROCESSOR ADDRESS
RxFlowSeg1	Same as *RxFlowSeg0*, except for RxByte Processor1.	0xBDE04294	0x84 to 0x87

RxFlowSize0 Register (FP Rx Byte Processor Function)

Purpose — Identifies the Length of the PDU for a given flow. See Table 224 on page 694 for similar register.

Address — 0xBDE04098

RxByte Processor Address — 0x88 to 0x8B

Access — Global Read/Write, RxByte Processor Write, Hardware Read

Bit Position	15 … 0
Field Name	Rx Flow Size
Reset Value	0

FIELD NAME	BIT POSITION	DESCRIPTION
Rx Flow Size	15:0	**Receive Flow Size** — Length in Bytes of PDU of the current flow.

Table 224 RxFlowSize1 Register (for RxByte Processor1)

REGISTER NAME	PURPOSE	ADDRESS	RXBYTE PROCESSOR ADDRESS
RxFlowSize1	Same as *RxFlowSize0*, except for RxByte Processor1.	0xBDE04298	0x88 to 0x8B

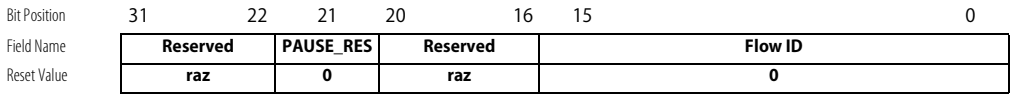
RxTxCgs0 Register (FP Rx Byte Processor Function)

Purpose — Transmit Congestion Flow ID, signals transmit side that a specific flow has congestion and should be stopped. See Table 225 on page 695 for similar register.

Address — 0xBDE0409C

RxByte Processor Address — 0x8C to 0x8F

Access — Global Read/Write, RxByte Processor Write, Hardware Read



Bit Position	31 … 22	21	20 … 16	15 … 0
Field Name	Reserved	PAUSE_RES	Reserved	Flow ID
Reset Value	raz	0	raz	0

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:22	Read as zero.
PAUSE_RES	21	**Pause Resume** — 1 = disable transmit (Pause), 0 = enables or resumes transmit. Note: Writing the *PAUSE_RES* bit validates the data in the register. This bit MUST be written last.
Reserved	20:16	Read as zero.
Flow ID	15:0	**Flow ID** — 16bit Flow ID (FID) that is sent to the FPTx, where it is masked and then mapped to a 7bit queue number by the FPTx Flow Control CAM.

Table 225 RxTxcgs1 Register (for RxByte Processor1)

REGISTER NAME	PURPOSE	ADDRESS	RXBYTE PROCESSOR ADDRESS
RxTxcgs1	Same as *RxTxcgs0*, except for RxByte Processor1.	0xBDE0429C	0x8C to 0x8F

RxFP_Enable Register (FP Rx Enable Function)

The internal FP Rx logic is enabled via the RxEnable Register.

A separate enable bit exists in the RxFI register for the FPRx external fabric interface.

Purpose Setting the enable to a 1 turns on the internal FPRx logic.

Address 0xBDE04600

Access Global Read/Write

Bit Position	31	30 … 0
Field Name	**Enable**	**Reserved**
Reset Value	**0**	**raz**

FIELD NAME	BIT POSITION	DESCRIPTION
Enable	31	**FPRx Enable** — 1 enables FPRx; 0 disables FPRx and holds it in reset.

RxFI_Configuration Register (FP Rx Configuration Function)

The FP contains eight (8) configuration registers residing in global address space allocated to the FP block. The eight (8) registers include: *RxFI_Configuration*, R*xDS_Header_Change1*, *RxDS_Header_Change2*, *RxDS_Header/Payload_Delimiter0*, *RxDS_Header/Payload_Delimiter1*, *RxDS_Header/Payload_Delimiter2*, *RxDS_Configuration*, and *RxFI_CRC*. These registers enable configuration of each of the stages and are defined below.

Purpose Allows physical configuration of the FPRx Interconnect.

Address 0xBDE04604

Access Global Read/ Write

Bit Position	31	30 27	26	25	24	23	22 16	15 8	7	6	5 4	3	2	1	0
Field Name	**CFI Enable**	**Inf**	**ByteParity**	**Odd Parity**	**Check Parity**	**Fixed Size Segment**	**Parity Ctl Mask**	**SEG Size**	**CLH**	**Fab Size**	**Fabric Width**	**BigEnd**	**Rsvd**	**ATM**	**RegInput**
Reset Value	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**raz**	**0**	**1**

<table>
<tr><th>FIELD NAME</th><th>BIT POSITION</th><th>DESCRIPTION</th></tr>
<tr><td>CFIEnable</td><td>31</td><td>Interface Enable Bit — 1 enables the Fabric Interface state machines and I/O buffers; 0 disables Fabric Interface (external pins are tri-stated). This enable, separate from the Rx Enable, allows software to delay the asynchronous payload from entering the port until the C-5e NP synchronous portions of the FP are ready.</td></tr>
<tr><td>Interface</td><td>30:27</td><td>Interface Type — Selects the Interface type. Legal values are detailed here:
<table>
<tr><th>ENCODED VALUE</th><th>INTERFACE</th></tr>
<tr><td>0x0</td><td>UTOPIA3</td></tr>
<tr><td>0x1</td><td>Reserved</td></tr>
<tr><td>0x2</td><td>CSIX-L1</td></tr>
<tr><td>0x3</td><td>PowerX (CSIX-L0)</td></tr>
<tr><td>0x4</td><td>PRIZMA</td></tr>
<tr><td>0x5</td><td>UTOPIA1 and 2</td></tr>
<tr><td>0x6 to 0xF</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>ByteParity</td><td>26</td><td>Byte Parity — 1 selects parity on each Byte lane. 0 deselects Byte lane parity. Used in PowerX mode only. When selected, one bit of parity is matched for each Byte of fabric width (pins CFI_CTL [6:3] = P[0:3]). Must be 1 for PowerX mode, 0 for all other modes.</td></tr>
<tr><td>Odd Parity</td><td>25</td><td>Odd Parity — 1 selects odd parity check, 0 selects even parity.</td></tr>
<tr><td>Check Parity</td><td>24</td><td>Check Parity — 1 enables parity checking, 0 disables parity checking.</td></tr>
<tr><td>FixedSizeSegments</td><td>23</td><td>Fixed Size Segments — 1 selects fixed sized segments, 0 selects variable sized segments.
Must be set to 1 for UTOPIA2, UTOPIA3 and PRIZMA. Can be set to 0 for UTOPIA3Like to M-5, CSIX-L1 and PowerX modes if desired.</td></tr>
<tr><td>ParityCtl Mask</td><td>22:16</td><td>Parity Control Mask — Only used in PowerX mode and must be set to 7. Must be set to 0 in non-PowerX modes. Mask used to identify the valid CFI control pins that parity is calculated over. (Parity of selected control pins is always XOR’ed to least significant parity bit).</td></tr>
</table>

FIELD NAME	BIT POSITION	DESCRIPTION
SEG Size	15:8	**Segment Size** — Configures segment size of fabric (legal values are from 8 to 204Bytes). Note: The segment size must be a multiple of four Bytes (1 word). Unpredictable results may occur for values outside of this range or non word aligned.
CLH	7	**Cell Level Handshake** — *Must* be set to 1. When selected, flow control threshold values in RxDS Header register (see page 699) enable flow control on nearest cell boundary. When not selected, flow control threshold values are directly used to assert flow control.
Fabric Size	6	Reserved
Fabric Width	5:4	**Fabric Bus Width** — Specifies Fabric bus width. Legal ranges are detailed here: **ENCODED VALUE** / **WIDTH (BITS)** 00 / 8 01 / 16 10 / Reserved 11 / 32
BigEnd	3	**Big Endian** — 1 selects Big Endian mode, 0 selects Little Endian mode.
Reserved	2	Read as zero.
ATM	1	**ATM Mode** — 1 selects ATM mode, 0 selects PHY mode. This bit is meaningful only in UTOPIA mode.
RegInput	0	**Register Control and Data Input Pins** — Asserted in all modes except UTOPIA1 and 2. 1 selects Registered Input; 0 selects non-Registered Input.

RxDS_Header_Change1 Register (FP Rx Configuration Function)

Purpose Configures Data Separator use of Payload Delimiter0 and Payload Delimiter1 registers. See Table 226 on page 699 for similar register.

Address 0xBDE04608

Access Global Read/Write

Bit Position	31	30 24	23 16	15 8	7 0
Field Name	Change	Reserved	Change Index	Change Value	Change Mask
Reset Value	0	raz	0	0	0

FIELD NAME	BIT POSITION	DESCRIPTION
Change	31	**Change** — Enable ability to switch from *RxDS Header Delimiter0* register to *RxDS Header Delimiter1* register.
Reserved	30:24	Read as zero.
Change Index	23:16	**Change Index** — Index into segment for match comparison Byte.
Change Value	15:8	**Change Value** — Value to match against.
Change Mask	7:0	**Change Mask** — Mask to apply to match comparison Byte.

Table 226 RxDS_Header_Change2 Register

REGISTER NAME	PURPOSE	ADDRESS
RxDS_Header_Change2	Same as RxDS_Header_Change1.	0xBDE0460C

RxDS_Header/Payload_Delimiter0 Register (FP Rx Configuration Function)

Purpose Used by the Data Separator hardware to separate the header from the payload. SeeTable 227 on page 700 for similar registers.

Address 0xBDE04610

Access Global Read/Write

Bit Position	31 24	23 16	15 8	7 0
Field Name	Reserved	Header Last Index	Payload First Index	Payload Last Index
Reset Value	raz	0	0	0

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:24	Read as zero.
Header Last Index	23:16	**Header Last Index** — Byte position offset from 0 Byte (first Byte) of cell identifying the last Byte of the header. Must be less than or equal to (Segment Size -5). .
Payload First Index	15:8	**Payload First Index** — Byte position offset from 0 Byte (first Byte) of cell identifying the first Byte of the payload.
Payload Last Index	7:0	**Payload Last Index** — Byte position offset from 0 Byte (first Byte) of cell identifying the last Byte of the payload. Must be > 3, and must be > Payload First Index and Payload Last Index must be > Header Last Index. Note: The Payload Last Index typically equals (Segment Size -1), using the *Segment Size* value programmed into the *RxFI_Configuration* register. Refer to RxFI_Configuration Register (FP Rx Configuration Function).

Table 227 RxDS_Header/Payload_Delimiter1 and 2 (for Payload Delimiter1 and 2)

REGISTER NAME	PURPOSE	ADDRESS
RxDS_Header/Payload_ Delimiter1	Same as *RxDSHeader/Payload Delimiter0,* except for Payload Delimiter1.	0xBDE04614
RxDS_Header/Payload_ Delimiter2	Same as *RxDSHeader/Payload Delimiter0,* except for Payload Delimiter2.	0xBDE04618

RxDS_Configuration Register (FP Rx Configuration Function)

Purpose Configures the Data Separator Hardware.

Address 0xBDE0461C

Access Global Read/Write

Bit Position	31	30	29 25	29 28	27 25	26 23	22 20	19	18	17	16	15 8	7 0
Field Name	QueGntDis	SharMem GntDis	Rsvd	NumPri	Rsvd	HdrFIFO XOFF	Ctl Word Size	CtlWord Disable	Rsvd	Drop Hdr	Rx Byte EOH	Data XOFF Threshold	Data XON Threshold
Reset Value	0	0	raz	0	raz	0	0		raz		0	32	32

FIELD NAME	BIT POSITION	DESCRIPTION
QueGntDis	31	**Queue Grant Disable, (PRIZMA mode only)** — Must be set to 1. 0 enables hardware interpretation of Queue Grants to link-level flow control Tx side of C-5e NP. 1 disables hardware so that RxByte microcode can invoke per-flow flow control. When enabled, link-level flow control is asserted whenever any of the Queue Grants are not enabled.
SharMemGntDis	30	**Shared Memory Grant Disable, (PRIZMA mode only)** — Must be set to 0. A 0 value enables hardware interpretation of Shared Memory Pool Grants to link-level flow control Tx side of C-5e NP. 1 disables hardware so that RxByte microcode can invoke per-flow flow control. When this bit is zero, link-level flow control is asserted whenever any of the Shared Memory Grants are not asserted.
NumPri	29:28	**Number of Priorities** — This is the number of shared memory grant priorities that are enabled. Note: This is only applicable to PRIZMA/UDASL applications. Encoded values are detailed here: **ENCODED VALUE** / **NUMBER OF PRIORITIES ENABLED** 1 — 1 priority (priority "0", which is the highest level) 2 — 2 priorities (0 and 1) 3 — 3 priorities (0, 1 and 2) 0 — All 4 priorities (0, 1, 2 and 3)
Reserved	27:25	Read as zero.

FIELD NAME	BIT POSITION	DESCRIPTION
HdrFIFOXOFF	26:23	**Header FIFO Flow Control Threshold for XOFF** — The level of 4Byte words available in either header FIFO at which flow control is asserted. When the available count drops to this threshold or below, a link-level pause is asserted. When the available count rises above this, a resume occurs. When a header FIFO is empty, its available count is 16 (64Bytes). When full, its count is 0 (0Bytes). The valid range for the threshold is 15 to 0.
CtlWordSize	22:20	**Control Word Size**, (**PowerX only)** — Must be set to 2. Indicates the size of control words for fabrics which support them. For instance, 0=disabled, 010b=2 bytes, 100b=4 bytes. Control words are directed to the appropriate Byte processor between cells. Between cells, a Byte processor needs to test a control word condition prior to each cell being processed. The Byte processor needs to handle all control words prior to handling the next cell.
CtlWordDisable	19	**Control Word Disable** — Must be set to 0 for PowerX and 1 for all other modes. 1 deselects the Control word FIFO operation; 0 enables CTL word FIFO operation. By deselecting the CTL word FIFO, control words are ignored by the Data Splitter logic and not presented to the Byte processor.
Reserved	18	Read as zero.
DropHdr	17	**Drop Header** — Must be set to 0. 1 selects header to be dropped when payload is being dropped due to congestion; 0 selects header to be forwarded to Byte processors even when payload is being dropped.
Rx Byte processor EOH	16	**Receive Byte Processor End of Header Indication** — 1 selects RxByte Byte processor Data9 test condition to be set true coincident with the last Byte of the header (PDU + SFR). Refer to *C-Ware Microcode Programming Guide (part number CSTMCPG-UG/D)*. 0 selects RxByte Data9 test condition to be set true coincident with the first Byte of the header.
Data XOFF Threshold	15:8	**Payload FIFO Flow Control threshold for XOFF** — Number of 32bit words available in the payload input FIFO when flow control is asserted. Valid Range 0-126. Note: The XOFF threshold must be less than the XON threshold. When empty, there are 128 words available in the Payload FIFO (512Bytes).

FIELD NAME	BIT POSITION	DESCRIPTION
Data XON Threshold	7:0	**Payload FIFO Flow Control threshold for XON** — Number of 32bit words -1 available in the payload input FIFO when flow control is deasserted. Valid Range: 1-127. Note: The XON should always be greater than the Data XOFF Threshold. If Data XON Threshold is less than Data XOFF, unpredictable results will occur. When empty, there are 128 words available in the Payload FIFO (512Bytes).

RxFI_CRC Register (FP Rx Configuration Function)

Purpose — The *RxFI_CRC* register configures the CRC function.

- Initial Value of the FP accumulator (0 or all 1s)
- Inverted (ones complement)

The *RxFI_CRC* provides INDEX fields which allow the CRC to be calculated over any sequential portion of the Segment and then appended or inserted anywhere afterward.

Address — 0xBDE04620

Access — Global Read/Write

Bit Position	31	30 28	27	26	25	24	23 16	15 8	7 0
Field Name	Enable	Rsvd	Rsvd	Init1	Reflect	Invert	FirstIndex	LastIndex	Rsvd
Reset Value	0	raz	0	0	0	0	0	0	0

FIELD NAME	BIT POSITION	DESCRIPTION
Enable	31	**CRC Enable** — 1 enables CRC mechanism; 0 disables and leaves the CRC mechanism in reset.
Reserved	30:28	Read/write
Reserved	27	Must be set to 1
Init1	26	**CRC Initialization** — 1 initializes the CRC register to all 1s; 0 initializes it to a 0.
Reflect	25	Reserved. Must be set to 0.
Invert	24	**Invert CRC** — 1 selects CRC to be inverted prior to being appended to Segment; 0 selects not inverted.
FirstIndex	23:16	**First Index** — Offset from byte 0 of segment to start of CRC accumulation byte. FirstIndex must be a multiple of 4.

FIELD NAME	BIT POSITION	DESCRIPTION
LastIndex	15:8	**Last Index** — Must be set to (cell size - 1). Represents the offset from byte 0 of segment to the last byte of the CRC value itself (not the last byte of the CRC accumulation region). Must be equal to or less than (cell size -1), even if CRC is not enabled.
Reserved	7:0	**Append Index** — Unused. Byte offset from byte 0 of segment to appended CRC.

RxWCS_CAM Register (FP RxWCS CAM Function)

Purpose Interface in Global address space to initialize the FP RxByte Processor's WCSs and CAMs along with the DBE WCS.

Address 0xBDE04624

Access Global Read / Write

Bit Position	31 … 24	23	22	21	20	19	18	17	16
Field Name	DBE Data In	DBE Scan Out	Rsvd	DBE Write	Rsvd	DBE Scan Capture	Rsvd	DBE Scan1 In	DBE Scan0 In
Reset Value	0	x	raz	0	raz	0	R/W	0	0

Bit Position	15 … 8	7	6	5	4	3	2	1	0
Field Name	WCS Data In	WCS Scan1 Out	WCS Scan0 Out	WCS Write	Rsvd	WCS /CAM Scan Capture	WCS /CAM Scan Update	WCS /CAM Scan1 In	WCS /CAM Scan0 In
Reset Value	0	x	x	0	raz	0	0	0	0

FIELD NAME	BIT POSITION	DESCRIPTION
DBE Data in	31:24	**DBE Data In** — DBE WCS Byte wide data
DBE Scan Out	23	**DBE Scan Out** — Output of DBE Scan Chain (Read Only).
DBE Write	21	**DBE Write** — DBE WCS Byte Write (1 Selects Write)
DBE Scan Capture	19	**DBE Scan Capture** — Capture data from selected address field from the DBE WCS to the DBE Scan Chain
Reserved	18	Read and Write.
DBE Scan1 In	17	**DBE Scan1 In** — Shift a 1 into the DBE Scan Chain.
DBE Scan0 in	16	**DBE Scan0 In**— Shift a 0 into the DBE Scan Chain.
Reserved	22, 20, 4	Read as Zero.

FIELD NAME	BIT POSITION	DESCRIPTION
WCS Data in	15:8	**WCS Data In** — WCS Byte wide data
WCS Scan1 Out	7	**WCS Scan1 Out** — Output of WCS1 Scan Chain (Read Only).
WCS Scan0 Out	6	**WCS Scan0 Out** — Output of WCS0 Scan Chain (Read Only).
WCS Write	5	**WCS Write** — WCS Byte Write (1-Selects Write)
WCS/CAM Scan Capture	3	**WCS/CAM Scan Capture** — Capture data from selected address fields from WCS/CAM to WCS/CAM Scan Chain.
WCS/CAM Scan Update	2	**WCS/CAM Scan Update** — Store or Update the CAM entry as defined on the CAM Addr bits with the 36 bits of data in the CAM Group, CAM Pattern, and CAM Tag.
WCS/CAM Scan1 In	1	**WCS/CAM Scan1 In** — Shift a 1 into the Scan Chain.
WCS/CAM Scan0 in	0	**WCS/CAM Scan0 In** —Shift a 0 into the Scan Chain.

RxByte0 General Purpose Configuration Register (FP Rx Configuration Function)

Purpose — The *FP RxByte0 General Purpose Configuration* register provides an area for passing information between the XP and the RxByte Processor0. SeeTable 228 on page 706 for similar register.

Global Address — 0xBDE04628

RxByte Processor Address — 0xA0 – 0xA3

Access — Global Read/Write, RxByte Processor Read – Byte addressable

Bit Position	31	0
Field Name	Data	
Reset Value	0	

Table 228 RxByte1 General Purpose Configuration Register (for RxByte Processor1)

REGISTER NAME	PURPOSE	ADDRESS	RXBYTE PROCESSOR ADDRESS
RxByte1 General Purpose Configuration	Same as *RxByte0 General Purpose Configuration*, except for RxByte Processor1.	0xBDE0462C	0xA0 – 0xA3

RxFCE_Configuration0 Register (FP Rx Configuration Function)

The RxFCE contains configuration registers that reside in global address space and are allocated to the FP block. These registers enable configuration of descriptors and Buffer Pools as defined below.

Purpose	Configures RxFCE descriptor size, Ring Bus response size, and flow mask.
Address	0xBDE04630
Access	Global Read/Write,

Bit Position	31 – 26	25 – 24	23	22 – 18	17 – 16	15 – 0
Field Name	Reserved	DescSize	TLU Resp	Reserved	Resp Size	PDUIDMask
Reset Value	raz	10	0	R/W	10	0xFFFF

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:26	Read as zero.
DesSize	25:24	**QMU Descriptor Size** — Selects the Descriptor Size and implicitly selects the extract space. For 16Byte x 16 scopes (*Desc Size* = 12 or 16). For 32Byte x 8 scopes (*Desc Size* = 24 or 32). Legal ranges are detailed here: **ENCODED VALUE** / **SIZE (BYTES)** 00 / 32 01 / 12 10 / 16 11 / 24

FIELD NAME	BIT POSITION	DESCRIPTION
TLU Resp	23	**Enable TLU Response** — Force DBE to wait until TLU response is complete before beginning to build a descriptor. 1= enabled 0 = disabled
Reserved	22:18	Read and Write.
Resp Size	17:16	**Response Size** — Selects the TLU Response Size. For 16Byte x 16 scopes, (*Resp Size* = 16). For 32Byte x 8 scopes, (*Resp Size* = 16 or 32). Legal ranges are detailed here: **ENCODED VALUE** / **SIZE (BYTES)** 00 / 32 01 / 8 10 / 16 11 / reserved
PDUID Mask	15:0	**PDU ID Mask** — Used to mask bits of PDU ID. Note: This could be used in lieu of microcode to automatically mask a PDU ID of less than 16bits.

RxFCE_Configuration1 Register (FP Rx Configuration Function)

Purpose Configures the RxFCE.

Address 0xBDE04634

Access Global Write Only

Bit Position	31 16	15	14	13 10	9	8	7 0
Field Name	PDU Size	DfltSzEn	PDULEen CkDis	Rsvd	DropOnFlow	Drop on BTag	Allocation Delay
Reset Value	0	raz	0	raz	0	0	0

FIELD NAME	BIT POSITION	DESCRIPTION
PDU_Size	31:16	**Default PDU Size** — Used in place of the *RxFlowSz* register value if bit 15 (*Dflt SzEn*) is selected.
DfltSzEn	15	**Default Size Enable** — 1 selects default size; 0 requires RxByte processor code to fill out PDU length (usually obtained from first PDU segment header) in the *RxFlowSz* register.
PDULenCkDis	14	**PDU Length Check Disable** — 1 disables PDU length check; 0 enables PDU length check. The PDU Length Check needs to be disabled for applications that do *not* specify an accurate PDU length. Typically this applies to applications that use a Default PDU Size. With the PDU Length Check disabled, the PDUs are enqueued without error even when the number of bytes received is fewer than expected.
Reserved	13:10	Read as zero.
DropOnFlow	9	**Drop on Flow** — This bit must be set to 1. 1 selects drop segment if a Flow Table CAM entry is not available; 0 selects do not drop flow and wait for a Flow Table entry to become available.
Drop on BTag	8	**Drop on BTag** — This bit should be set to 1. 1 selects drop segment if BTag is not available; 0 selects wait for BTag to become available. Leaving this set to 0 presumes that under a worst case scenario, the BTag will become available prior to the first segment fully arriving.
Allocation Delay	7:0	**Allocation Delay** — This bit must be set to 0. Back-off time delay between retries when the FPRx Buffer Pool Manager is replenishing BTags. The default is 0. Can change using (number of core clocks 8), but not recommended.

RxFCE_Configuration2 Register (FP Rx Configuration Function)

Purpose Configures the FCE.

Address 0xBDE04638

Access Global Write Only (Global Reads return inaccurate data).

Bit Position	31 … 23	22	21	20	19	18	17 … 0
Field Name	Reserved	RetryEnq	XOFFNoScope En	EnqWaitWrCB	Force 64Byte WrCB Transfers	DefaultQEn	DefaultQNum
Reset Value	R/W	0	0	0	0	0	0

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:23	Read and Write.
RetryEnq	22	**Retry Enqueue —** When set to 1, the FPRx retries failed enqueue operations to the QMU until they succeed. When set to 0, the FPRx drops the PDU upon a failed enqueue operation.
XOFFNoScopeEn	21	**XOFF No Scope Enable —** When enabled, the FPRx asserts a link-level pause when a “no scope available” condition is encountered in an RxByte Processor; a resume is issued when the scope becomes available.
EnqWaitWrCB	20	**Enqueue Wait WrCB —** When set to 1, enqueue requests are stalled until all WrCB DMAs associated with the PDU are complete. When set to 0, enqueue requests are issued as early as possible. Typical applications should deassert this bit.
Force 64Byte WrCBTransfers	19	**Force 64Byte WrCB Transfers —** When asserted, the FPRx write control blocks (WrCB’s) always perform a 64Byte DMA transfer to SDRAM. Otherwise, DMA transfers are in 16Byte increments (16, 32, 48 or 64).
DefaultQEn	18	**Default Queue Enable —** 1 selects the use of the Default Queue (see *Default Queue* field) in the event of a TLU error; 0 disables the use of the default queue. Note: To use this feature ensure that the DBE microcode does *not* set the Drop bit in the descriptor control word, or else the PDU will be dropped after a TLU error.
DefaultQNum	17:0	**Default Queue Number** — Queue number to be used if *DfltQEn* bit set *and* a TLU_ERROR occurs. In QMU Internal Mode the queue is represented by bits [8:0]. In QMU External Mode, the queue is represented by bits [17:0].

Pool0_CFG0 Register (FP Rx Pool Configuration Function)

Associated with each of the four (4) buffer pools are two (2) configuration registers *PooIn_CFG0* and *PooIn_CFG1*, where n=0,1,2,3.

Purpose	Configures FCE Buffer Pool0 ID and buffer size parameters. See Table 229 on page 710 for similar registers.
Address	0xBDE04640
Access	Global Read/Write

Bit Position	31 21	20 16	15 6	5 0
Field Name	Reserved	Pool ID	Buffer Size	Reserved
Reset Value	raz	0	0	raz

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:21	Read as zero.
Pool Id	20:16	**Pool ID** — Maps to BMU Pool.
Buffer Size	15:6	**Buffer Size** — Must match BMU buffer size (between 64 and 64Kbytes) for specified pool ID. The sizes are in terms of 64 Byte blocks. Buffer Size[15:6]=1 corresponds to a size of 64. Buffer Size[15:6]=0b1111111111 corresponds to a size of 64K-64. Note: A size of 0 is not applicable; Buffer Size[15:6]=0 corresponds to a size of 64K.
Reserved	5:0	Read as zero.

Table 229 Pool*n*_CFG0 Registers (for Pools 1, 2, and 3)

ADDRESS	REGISTER NAME	PURPOSE	COUNTER WIDTH (BITS)
0xBDE04648	Pool1_CFG0	Same as *Pool0_CFG0*, except for pool 1.	32
0xBDE04650	Pool2_CFG0	Same as *Pool0_CFG0*, except for pool 2.	32
0xBDE04658	Pool3_CFG0	Same as *Pool0_CFG0*, except for pool 3.	32

Pool0_CFG1 Register (FP Rx Pool Configuration Function)

Purpose Configures FCE Buffer Pool0, allocation threshold, and last block parameters. See Table 230 on page 711 for similar registers.

Address 0xBDE04644

Access Global Read/Write

Bit Position	31 26	25 24	23 11	10 8	7 3	2 0
Field Name	Reserved	Reserved	Reserved	Alloc Threshold	Rsvd	Last Block
Reset Value	raz	n	raz	0	raz	0

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:26	Read as zero.
Reserved	25:24	Must be set to *n*. Where *n* is the Pool*n*_CFG1 register. This field resets to *n*. For example, *n*=0 for Pool0, *n*=1 for Pool1, *n*=2 for Pool2, *n*=3 for Pool3.
Reserved	23:11	Read as zero.
Alloc Threshold	10:8	**Allocation Threshold** — Specifies a threshold value such that when the number of blocks drops below that threshold, additional BTags are fetched until the 'Last Block' level is reached. Rules: Allocation Threshold must *ALWAYS* be ≤ Last Block. Last Block = 0 disables pool refill.
Reserved	7:3	Read as zero.
Last Block	2:0	**Last Block of BTags** — Specifies the blocks (0-7) that are to be filled. Specifying a Last block of 0 disables a given pool. Therefore the useful range is 1 to 7, where a value of 1 with an Allocation Threshold = 1 fills two blocks (0 and 1).

Table 230 Pool*n*_CFG1 Registers (for Pools 1, 2 and 3)

REGISTER NAME	PURPOSE	ADDRESS
Pool1_CFG1	Same as *Pool0_CFG1*, except for pool 1.	0xBDE0464C
Pool2_CFG1	Same as *Pool0_CFG1*, except for pool 2.	0xBDE04654
Pool3_CFG1	Same as *Pool0_CFG1*, except for pool 3.	0xBDE0465C

RxByte_Shared_Low Register (FP Rx Shared Function)

The RxByte Processor provides two (2), 4Byte shared registers (*RxByte_Shared_Low* and *RxByte _Shared_High*) that are accessible by both the RxByte Processors (RxByte Processor0 and RxByte Processor1). Therefore, a total of 8Bytes is accessible by either RxByte Processor0 or RxByte Processor1. The registers are Byte writable from each of the Byte processors.

The use of the *RxByte_Shared_Low* and *RxByte _Shared_High* registers are application specific. For example, they could be used by the RxByte Processor's microcode to pass information between the RxByte Processor0 and RxByteProcessor1.

One Byte of these registers could serve as a read/modified write, (where the value of the read/modified write is passed to each Byte Processor from the RxByte General Purpose Configuration register), if needed by the application.

Purpose	Provides 4Bytes for passing information between the FP's RxByte Processor0 and RxByte Processor1.
Global Address	0xBDE04660
RxByte Processor Address	0xB0 – 0xB3
Access	Global Read, RxByte Processor Read/Write – Byte addressable

Bit Position	31	0
Field Name	**Data**	
Reset Value	**0**	

RxByte_Shared_High Register (FP Rx Shared Function)

Purpose	Same as the *RxByte_Shared_Low* register, except that this register provides an *additional* 4Bytes for passing information between the FP's RxByte Processor0 and the RxByte Processor1.
Global Address	0xBDE04664
RxByte Processor Address	0xB4 – 0xB7

RxFP_Interrupt_Event Register (FP Rx Interrupt Function)

Purpose Access to interrupt events (interrupts directed to the XP via the *RxFP_Interrupt_Enable* register).

Address 0xBDE04680

Access Global Read/Write, Write 1 to Clear

Bit Position	31 … 6	5	4	3	2	1	0
Field Name	Reserved	ParityError	No BTags on Alloc	WrFail	BTagPRG	BTagECC	Alloc Timeout
Reset Value	raz	0	0	0	0	0	0

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:6	Read as zero.
ParityError	5	**Parity Error** — Indicates a parity error was detected on the Freescale Fabric interface.
No BTags on Allocation	4	**No BTags on Allocation** — BTag allocation failed because of no BTags available from the BMU.
WrFail	3	**Payload Fail** — Indicates a Payload Bus error during a Payload Write via the WrCB (that is, time-out after 16 retries).
BTagPRG	2	**BTag PRG** — Indicates either bad pool request (code 0x2) or a non-existent memory location (code 0x6) was attempted by the RdCB. These result from an inconsistent programming of the FPRx Pools relative to the FPRx BTag.
BTagECC	1	**BTag ECC** — Indicates a double ECC error was returned as a result of an attempt by the RxCB to transfer BTags from the BMU. A code of 0x5 is returned by the RdCB.
Allocation Timeout	0	**Allocation Time-out**— Indicates the number of BTag allocation retries exceeded, (16 max.)

RxFP_Interrupt_Enable Register (FP Rx Interrupt Function)

Purpose Enable interrupts for the corresponding bits in the *RxFP_Interrupt_Event* register. Set a bit to 1 to enable the interrupt.

Address 0xBDE04684

Access Global Read/ Write

Bit Position	31 ... 6	5	4	3	2	1	0
Field Name	Reserved	Parity ErrorEN	No BTags on Alloc EN	WrFailEN	BTagPRG EN	BTagECC EN	AllocTime outEN
Reset Value	raz	0	0	0	0	0	0

RxFP_Debug_Mux_Control Register (FP Rx Debug Function)

Four (4) of thirteen (13) events can be viewed via the *RxFP_Debug_ Mux_Control* register. The selectable events are shown in Table 231. Many of these events relate to FIFO full conditions which, if the FPRx is programmed correctly, should never occur.

Any event can be viewed in association with any of the four (4) selection fields, including simultaneously being selected in more than one field, that is, viewed multiple times.

Purpose For the purposes of diagnostics and debug. Enables key test points to the system event register to be viewed.

Address 0xBDE04688

Access Global Read, RxByte Processor Read/Write,

Bit Position	31	30 28	27 24	23	22 20	19 16	15	14 12	11 8	7	6 4	3 0
Field Name	En0	Rsvd	Sel0	En1	Rsvd	Sel1	En2	Rsvd	Sel2	En3	Rsvd	Sel3
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0

FIELD NAME	BIT POSITION	DESCRIPTION
En0	31	**Enable0** — 1 enables the associated selected events; 0 disables the associated event from being viewed.
Reserved	30:28	Read as zero.
Sel0	27:24	**Select0** — Selects one of the thirteen (13) FPRx events to be viewed for the corresponding field. See Table 231 on page 715.
En1	23	**Enable1** — 1 enables the associated selected events; 0 disables the associated event from being viewed.

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	22:20	Read as zero.
Sel1	19:16	**Select1** — Selects one (1) of the thirteen (13) FPRx events to be viewed for the corresponding field. See Table 231 on page 715.
En2	15	**Enable2** — 1 enables the associated selected events; 0 disables the associated event from being viewed.
Reserved	14:12	Read as zero.
Sel2	11:8	**Select2** — Selects one (1) of the thirteen (13) FPRx events to be viewed for the corresponding field. See Table 231 on page 715.
En3	7	**Enable3** — 1 enables the associated selected events; 0 disables the associated event from being viewed.
Reserved	6:4	Read as zero.
Sel3	3:0	**Select3** — Selects one (1) of the thirteen (13) RxFP events to be viewed for the corresponding field. See Table 231 on page 715.

Table 231 RxFP Thirteen (13) Viewable Events

SELECT VALUE	EVENT NAME	DESCRIPTION
0	RxNextSeg	Indicates that internal hardware is requesting the next segment for processing. This event should occur normally as a result of correct FP operation for each segment received independent of whether an error is detected in the segment or not. The only exception is a parity error that occurs on the first word of a segment; in such case the data is dropped at the fabric interface and no segment is received.
1	RxFlow	Indicates a good Queue status has been received from the QMU as a result of an enqueue operation. This event normally occurs as a result of correct operation of the RxFP for non-errored PDUs.
2	RxSegmentErr	Indicates one or more of the errors (as defined in the error statistics table) has occurred on a segment. This event occurs operationally as a result of the non-fatal interface type errors.

Table 231 RxFP Thirteen (13) Viewable Events (continued)

SELECT VALUE	EVENT NAME	DESCRIPTION
3	RxPDUErr	Indicates that a DPU is being dropped and the BTag is being deallocated. This occurs anytime a PDU encounters an error *and* the BTag has already been allocated for that PDU.
4	BPMErrAlloc	Indicates that a BTag allocate error occurred due to no BTags available in the BMU.
5	AsyncFIFOFull	Indicates Async Fabric Interface FIFO is full. This should never occur under any circumstances and would indicate a hardware failure. This FIFO is 'deallocated' upon the condition that it is not empty, that is, data is moved out on the following clock cycle. Data is dropped at the synchronous payload FIFO under congested operation.
6	HdrFIFOFull	Indicates one of the header FIFOs is full. This FIFO is filled by the Data Splitter with the selected header Bytes as configured in the header/payload Delimiter registers.
7	EFIFOFull	Indicates an internal FIFO that detects segment errors is full. This condition should never occur.
8	PFIFOPause	Indicates the payload FIFO pause (XOFF) threshold has been exceeded. This typically occurs during a congested or flow-controlled situation.
9	FlowFull	Either of two Internal 'Flow FIFOs' are full. Each of the Rx Byte processors, upon selecting scope 'Avail', push the current flow forward into the flow handling hardware of the FP Rx. A 'Flow Full' event indicates that one of these FIFOs is full and in effect the flow handling hardware is stalled for some reason. Typically this is due to Internal hardware pending upon lack of some resource such as BTags (although DROP_ON_BTAG should be selected to prevent this). This event should never occur.
10	DBEFull	An Internal FIFO that passes segments information to the DBE. This event should never occur. This type of an event could occur if the DBE or TLU is improperly programmed so that the DBE is stalled, for example, waiting for a TLU operation to complete.
11	EnqFull	Indicates that the 16-deep enqueue request FIFO is full. Typically this would only occur in Retry-Enqueue mode, where the FPRx keeps retrying a single enqueue request when the QMU indicates the enqueue failed (that is, QMU queue is full).

Table 231 RxFP Thirteen (13) Viewable Events (continued)

SELECT VALUE	EVENT NAME	DESCRIPTION
12	DeqFull	FPRx Dequeue Request FIFO Full. This event should never occur. This indicates that FPTx is requesting Dequeue operations to the FPRx faster than they can be sent via the FP->QMU interface.

RxMemory_Address Register (FP Rx Debug Function)

Purpose Configures the target memory address for accessing the Rx Flow Table or Descriptor Table. Refer to "Rx Flow Table and Descriptor Table Access" on page 269.

Address 0xBDE04690

Access Global Read/Write

Bit Position	31	0
Field Name	Address	
Reset Value	0	

RxMemory_Data Register (FP Rx Debug Function)

Purpose Used to write or read target Rx Flow Table or Descriptor Table locations. Refer to Table 75 on page 270.

Address 0xBDE04694

Access Global Read/Write

Bit Position	31	0
Field Name	Data	
Reset Value	0	

RxPDU_ID_CAM Register (FP Rx Debug Function)

Purpose — Provides access to the FPRx PDU_ID CAM for debug purposes.

The PDU_ID CAM is a 160-entry CAM in the FPRx that maps a 16bit PDU ID to an 8bit internal PDU index used by hardware. The CAM can be accessed by software only for debug purposes.

Address — 0xBDE04698

Access — Global Read/Write. The CAM hardware updates the Free bit continuously. The hardware updates the CAM data field after a Search operation.

Bit Position	31:28	27	26	25	24	23:8	7:0
Field Name	Reserved	Free	Write	Delete	Search	Match	CAM Data
Reset Value	raz	0	raz	raz	0	0	0

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31:28	Read as zero.
Free	27	**Free** - Indicates that at least one entry in the CAM is free (available for use). This bit is read-only.
Write	26	**Write CAM Location** — Writes the location matched, or the next free location if nothing matches (for diagnostic purposes only). Setting this bit to a 1 launches a CAM write of the write data. The bit will then be cleared by the hardware. It is always read as zero.
Delete	25	**Delete CAM Entry** — Deletes CAM entry matched (for diagnostic purposes only). Setting this bit to a 1 launches a CAM delete of the entry corresponding to the match value. The bit will then be cleared by the hardware. It is always read as zero. Legal values are detailed here:

ENCODED VALUE	CAM ACTION
1	Delete CAM entry
0	Do *not* delete CAM entry

FIELD NAME	BIT POSITION	DESCRIPTION
Search	24	**Search** — 1 selects CAM search, using the match value. After the search, the result is returned via the CAM data field in this register.
Match	23:8	**16bit CAM Match Value** — Value to search on.
CAM Data	7:0	**CAM Data -** Contains the 8bit value that was read out of the CAM after the most recent Search operation. Alternatively, this represents the data that is written into the CAM on a Write operation.

RxFP_Statistics Registers (FP Rx Statistics Function)

The FPRx provides nineteen (19) registers that accumulate statistics. These registers are only accessible via the global bus. Table 232 defines these registers and provides their Global Bus address. The RxFP Statistics Registers are read in 32bit quantities. For counters that are only 16bits, the upper 16bits of the register are read as zero (raz) and the lower 16bits hold the contents of the requested counter. A global write to one of these registers, regardless of the write data, resets the counter to 0.

When a counter reaches its maximum value, it rolls over to 0 and starts counting again.

Table 232 Global Bus Receive FP Statistics Registers Map

ADDRESS	NAME	DESCRIPTION	COUNTER WIDTH (BITS)
0xBDE046A0	Segs_Rcvd	Number of segments received.	32
0xBDE046A4	PDUs_Enq	Number of PDUs successfully enqueued.	32
0xBDE046A8	Segs_Err	Number of segments with an error.	32
0xBDE046AC	PDUs_Err	Number of PDUs that were dropped due to an error *and* had their BTag deallocated. (does *not* include PDUs that were dropped *before* a BTag could be allocated, for instance due to an early parity error).	32
0xBDE046C0	CParity_Err	Number of Control Word Parity Errors (PowerX only).	16
0xBDE046C4	Err_Hdr	Number of Headers lost due to a header FIFO overflow.	16

Table 232 Global Bus Receive FP Statistics Registers Map (continued)

ADDRESS	NAME	DESCRIPTION	COUNTER WIDTH (BITS)
0xBDE046C8	Parity_Err	Number of received parity errors. If a parity error occurs *during* an assertion of SOF (the first clock cycle of a new segment), the parity error counter is incremented and the data is dropped immediately (no header or payload data is received into the FIFOs). If a parity error occurs *after* the SOF and while the segment is being received, one parity error is logged (only one per segment); the RxByte Processor would still have to process the header, after which the segment and its associated PDU are automatically dropped.	16
0xBDE046CC	Length_Err	Number of segment length errors.	16
0xBDE046D0	Reserved	Reserved. Read as zero.	16
0xBDE046D4	CRC_Err	Number of CRC errors. Each error causes the segment and its associated PDU to be dropped.	16
0xBDE046D8	Odd_Seg	Number of odd Segments (COMs and EOMs received without an associated BOM).	16
0xBDE046DC	Seq_Err	Number of sequencer errors (set by RxByte processor).	16
0xBDE046E0	Seq_Dis	Number of sequencer discards (set by RxByte processor).	16
0xBDE046E4	Lost_Len_Err	Number of PDUs dropped due to a PDU length error. There are three (3) cases that cause this error: • The number of received length bytes do not match the expected payload length (only flagged if PDU length checking is enabled). • A second BOM is received for an already active PDU. • A segment is labelled as a BOM or COM, implying that there are more segments to come, yet the entire PDU length has already been received.	16
0xBDE046E8	No_Flow_Tbl	Number of times no Flow Table entry available (all 160 flows are in use).	16
0xBDE046EC	No_BTag	Number of times no BTag available from the Pool Cache in the FPRx.	16

Table 232 Global Bus Receive FP Statistics Registers Map (continued)

ADDRESS	NAME	DESCRIPTION	COUNTER WIDTH (BITS)
0xBDE046F0	BTag_Err	Bits [23:16] represent the number of BTag Programming Errors, for example, because of a bad Pool ID. Bits [7:0] represent the number of BTag ECC errors. The remaining bits are unused.	32 (contains two 8bit counters)
0xBDE046F4	Alloc_Err	Number of BTag allocation errors, due to lack of available BTags in the BMU.	16
0xBDE046F8	Enque_Err	Number of enqueue errors (QMU responded with a NAK, that is, due to a full queue).	16

RxDebug_Internal_State Register (FP Rx Statistics Function)

Purpose Enables viewing key internal states including: RxByte0/1 Program Counter, Buffer State Machine States, Descriptor Build Engine Program Counter, Transfer Control Block Programming States, and FP-QMU State Machine States.

Address 0xBDE04700

Access Global Read Only

Bit Position	31	30 25	24 19	18 14	13 8	7 6	5 0
Field Name	Rsvd	RxByte0_PCtr	RxByte1_PCtr	BfrState	DBEPrgCtr	XCB	FP-QMU
Reset Value	0	0	0	0	0	0	0

FIELD NAME	BIT POSITION	DESCRIPTION
Reserved	31	Read a zero.
RxByte0_PCtr	30:25	**RxByte 0 Program Counter** — Program Counter0 for the RxByte0 Processor [5:0], bit 6 is not available.
RxByte1_PCtr	24:19	**RxByte1 Program Counter** — Program Counter1 for the RxByte1 Processor [5:0], bit 6 is not available.
BfrState	18:14	**Buffer State Machine States** — See Table 235 on page 723.
DBEPrgCtr	13:8	**Descriptor Build Engine Program Counter** — Program counter for the Descriptor Build Engine.
XCB	7:6	**Transfer Control Block Programming States** — See Table 234 on page 723.
FP-QMU	5:0	**FP-QMU State Machine States** — See Table 233 on page 723.

Each state machine runs off the core clock and requires at least one clock cycle to transition to the exit state. States are either *conditional*, that is, waiting for one or more conditions to become true so they can exit to an alternate state, or they are *transitional* whereby they will always exit to an alternate state in a single clock cycle.

While transitional states may have conditions that allow them to transition to one or more possible states, if no condition is true, a default exit state ensures that they cannot remain in the current state. The importance of this distinction is that when asynchronously polling the *RxDebug_Internal_State* register, it is important to realize that by chance you may hit upon transitional states that indicate that the machine is operating and that a cell is being processed by the FPRx.

Consistently seeing a conditional state indicates that the machine is most likely waiting for a condition to be fulfilled. This can help you identify a related illegal configuration.

Whether a state is conditional (C) or transitional (T) is called out in the following Machine State tables.

Table 233 FP-QMU State Machine States

STATE NUMBER	STATE TYPE	STATE NAME	DESCRIPTION / EXIT CONDITION
0	C	Idle	Idle state awaiting enqueue request, or dequeue request from the FPTx. This same state machine handles both enqueue and dequeue operations on the FP->QMU interface.
1	T	Pend	Pend State provided for timing purposes. Used by both enqueue and dequeue operations.
10	T	PrgCtl	Program Control Information to QMU; such as PDU length, and information from the Descriptor Control Word.
18 - 11	T	Enq8-Enq1	Program Words 8 – 1. Note: For 32Byte descriptors states Enq8-1 are transitioned, 24Byte descriptors Enq6-1, 16Byte descriptors Enq4-1, 12 Byte descriptors Enq3-1. Once an Enq state is entered they cycle down to Enq1, then Pend, then back to idle, all one clock per state.

Table 234 Transfer Control Block Programing States

STATE NUMBER	STATE TYPE	STATE NAME	DESCRIPTION / EXIT CONDITION
0	C	Idle/ PrgDMA	Idle State awaiting BFR Payload DMEM flush to DRAM. Upon which Write CB DMA Register being programmed.
1	T	PrgSys	Write CB System Register is programmed as state is exited.
2	T	PrgCtl	Write CB Control Register being programmed.

Table 235 Buffer State Machine States

STATE NUMBER	STATE TYPE	STATE NAME	DESCRIPTION / EXIT CONDITION
0	C	Idle	Idle, waiting for Valid Segment Indication from RxByte0 Processor and RxByte Processor1 in alternating order.
1	T	CAMWait	PDU ID (PDU ID) CAM lookup delay state.

Table 235 Buffer State Machine States (continued)

STATE NUMBER	STATE TYPE	STATE NAME	DESCRIPTION / EXIT CONDITION
2	C	Search	Match PDU ID to produce Flow Index (FIN), Determines Segment type. Requires available PDU ID CAM entry *OR DropOnFlow* bit enabled (bit [9] selected to 1 in the *RxFCE_Configuration1* register).
3	T	RdTable	FIN is used to index into RxFlow Table.
4	C	GetBTag	For beginning of messages (firsts), state to get a BTag. Technically this state is Transitional, however if no BTags exist, the Bfr will alternate between states 2 and 4 until BTags with correctly sized buffers become available.
5	C	XfrData	Transfer Data from Payload FIFO to 64Byte DMEM Buffer. Xfr DMEM to DRAM every 64Byte boundary. Stays in this state until End of Segment (EOS) or End of Packet (EOP).
6	C	DropData	Discard state for pad Bytes of last segment or discard due to Segment/Flow error. Exit upon EOS.
7	C	EnqQueue	Enqueue Assessment state. Waits if Enqueue and no room in Enqueue Request FIFO.
8	T	Error	Delay state to wait for worst case parity / FP errors.
9	C	NewBfr	State to get a new 64Byte DMEM buffer as XFR to buffer DRAM BTag. Wait in this state until a new DMEM buffer is available.
10	C	PendBfr	Special case where end of segment and 64 byte DMEM buffer are full. Need to get a new buffer before updating state table. Wait in this state until new DMEM buffer is available.
11	T	RtnPend	Special delay state to allow DMEM buffer to be returned during transfer if an error detected.

Appendix B

USING AGGREGATE MODE

Appendix Overview

This appendix covers the following topics:

Purpose of the C-5e NP Channel Aggregate Mode

The C-5e NP Aggregate Mode enables you to scale serial bandwidths. The CPs can be aggregated into parallel clusters for wider data streams. The C-5e NP's 16 CPs can be partitioned into four (4) groups of four (4) CPs called *clusters*. Clusters allow the CPs to share resources (IMEM and DMEM) and support aggregation. A cluster of CPs can be configured, for example, to work together to support one physical interface (such as OC-12), or either the receive or transmit portion of one physical interface (such as Gigabit Ethernet).

The types of physical interfaces that require aggregation in the C-5e NP include: Gigabit Ethernet, FibreChannel, OC-12 and OC-12c.

Aggregate Mode Requirements on the C-5e NP

Aggregation requires that the C-5e NP fulfill two (2) requirements for processing data from high-speed physical interface. These are needed to ensure that the C-5e NP is able to keep up with the speed of these high-bandwidth physical interfaces, as well as effectively use the processing power of the C-5e NP; the following requirements must be met:

1. Individual packets and cells must be distributed among the CPs in a cluster to effectively share the processing load.
2. Packet and cell ordering between the ingress and egress physical interfaces must be preserved.

Supporting these two (2) requirements have the following implications for the Serial Data Processor (SDP) and Channel Processor RISC Core (CPRC) components as described in Table 236 on page 726.

Table 236 Aggregate Mode Implications (for SDP and CPRC)

COMPONENT	IMPLICATION
RxSDP	The RxSDP must distribute incoming packets and cells to the different CPs within a cluster in a round robin fashion.
CPRC	The CPRC receive program must ensure that order is maintained with enqueue operations to the QMU within a cluster.
	The CPRC transmit program must ensure that order is maintained with dequeue operations from the QMU within a cluster.
TxSDP	The TxSDP must serialize outgoing packets and cells from the CPs within a cluster to a single physical interface correctly.

Packet/Cell Ordering Handling for Rx in Aggregate Mode

Most network protocols at Layer 2 and Layer 3 require that the forwarding component maintain ordering, but some do not. At the physical layer the distinction between these two (2) scenarios cannot be made. To solve this problem, the C-5e NP maintains ordering of all packets and cells that it processes from a physical interface when aggregated.

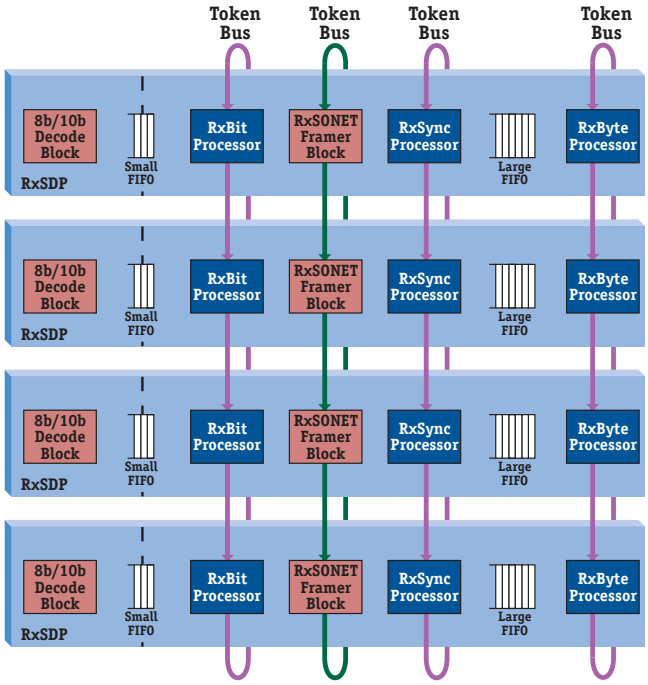
To support distribution of packets or cells evenly to the CPs within a cluster, the C-5e NP uses two (2) types of tokens:

- Hardware tokens in the RxSDP to deliver packets and cells to CPs in a round-robin fashion
- The CPRC software receive programs use software tokens to serialize enqueue operations. This maintains the ordering of descriptors to the QMU

Hardware Receive Tokens

The RxSDP processor provides four (4) *token buses* that run among the RxBit processors, the RxSONET Framer blocks, the RxSync processors, and the RxByte processors within a cluster. Refer to Figure 115 on page 728. The token buses in the RxSDP pass tokens between sequencers. A sequencer cannot forward a packet or cell upstream in the RxSDP until it has the token. The token passing function is asserted by a microprogram running in an RxSDP sequencer, and the token is typically passed by a microprogram based on packet or cell delineation. This ensures that different packets and cells are delivered to different CPs within a cluster in a round robin fashion.

Figure 115 RxSDP Token Buses



Software Receive Tokens

The CPRC receive program is typically notified of packet or cell arrival from the RxSDP. It then makes a forwarding decision and enqueues a descriptor to the QMU for forwarding to the egress interface. In the aggregated case, the software program running on the receive CPRC must ensure that the QMU receives application-defined descriptors in the same order that the packets that the descriptors represent arrived from the physical interface. To achieve this requirement, the software programs running on the CPRC must ensure that the descriptor enqueue operations are serialized.

The CPRC programs achieve this by using a piece of shared DMEM as a software token. A CPRC program only enqueues when it owns the token. The program passes the token to its neighbor after enqueueing the descriptor.

This operation ensures that transactions to the QMU are in the same order in which the packets arrived on the physical interface.

Packet/Cell Ordering Handling for Tx in Aggregate Mode

Like receive aggregation, transmit aggregation must maintain ordering from the QMU to the physical interface. This requires the same level of synchronization and ordering as receive aggregation.

Hardware Transmit Tokens

There is a single hardware token bus that is used in the TxByte processor. Refer to Figure 116 on page 730. The management of this token bus is controlled by the microcode running in the TxByte processor. This token controls the draining of the large FIFO downstream of the TxByte processor.

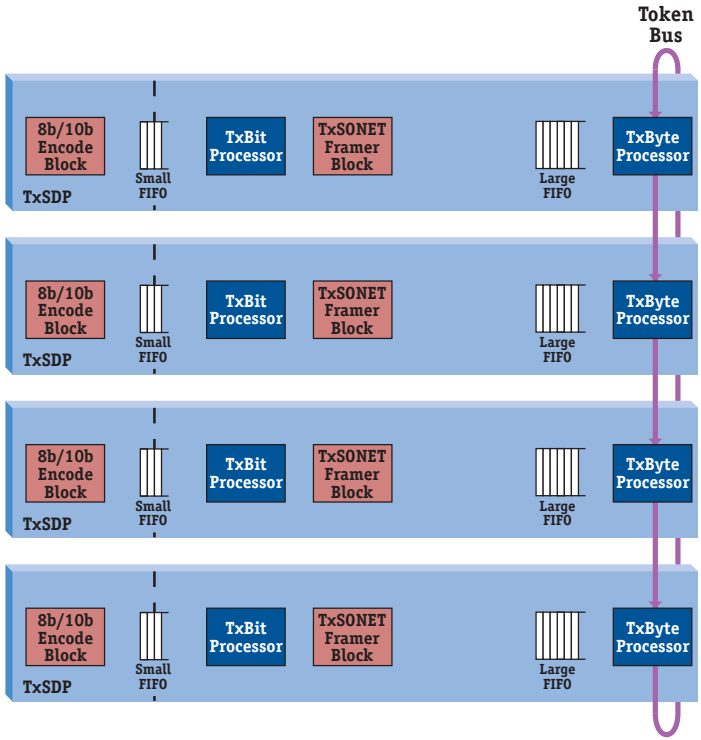
This function allows the TxByte processors to prime the FIFOs even if they do not happen to own the token. When the TxByte processor does own the token, the large FIFO is filled with enough data to keep the TxBit processor — and the physical interface — full.

There is only one (1) TxBit processor that operates per aggregated cluster. This processor gets the data from the large FIFO selected by the TxByte token that is being passed in a round-robin fashion.

Software Transmit Tokens

The CPRC transmit program maintains a software token in shared DMEM for the cluster doing transmit processing for the aggregated physical interface. The transmit program can only send the packet or cell to its TxSDP when it owns the token. After the TxSDP begins packet or cell transmission, the CPRC transmit program passes the token to its neighbor so it can begin the same process.

Figure 116 TxSDP Token Bus



Clock Distribution in Aggregate Mode

The pad data and clocks are wired so that they can be accessed equally by all CPs in the cluster.

Latching is performed at the pads using the configured clock, with the latched data circulated to the SDPs. The pads are grouped such that two (2) CPs have sufficient pads to meet the needs of either transmit or receive. During aggregation, the receive clock from one (1) CP cluster becomes the master receive clock for all of the pads in the two (2) CP clusters. In addition, the PHY chip in OC-12c generates the transmit clock, so the clock is received and then forwarded through the transmit clock mux to become the master transmit clock for the cluster.

The transmit data and clock are available to all four (4) transmit SDPs in CPs 0 to 3 through busing from the pins. Similarly, the receive data and clock is available for all four (4) receive SDPs in CPs 4 to 7 through busing from the pins. OC-12 aggregation of four (4) CPs has the transmit pins allocated from CPs 0 and 2 and receive pins allocated from CPs 1 and 3.

Aggregate Mode Application Examples

Each application that uses CP aggregation (Gigabit Ethernet, FibreChannel, OC-12 and OC-12c) has a slightly different implementation. This section provides details on each.

Gigabit Ethernet and FibreChannel Applications

Both Gigabit Ethernet and FibreChannel use the TBI (ten-bit interface) physical layer encoding. Even though Layer 2 and above of the protocol is different, each of these protocols implements aggregation in the same fashion. GMII for Gigabit Ethernet also follows this same aggregation scheme.

PHY Connectivity

The pins that connect the TBI/GMII and the C-5e NP are spread across all of the pins in a CP cluster. It so happens that the transmit pins from the TBI/GMII are on CP0 and CP1 of the cluster and the receive pins are on CP2 and CP3 of the cluster. Since every CP only has 7 pins associated with it, this configuration is necessary for processing of TBI/GMII protocols. For complete descriptions of Gigabit Ethernet and FibreChannel pinouts, see the *C-5e NP Data Sheet*.

After the pins are routed inside the C-5e NP, they are muxed together and sent as a single 10bit stream to each CP in the cluster for processing as a single stream. Refer to Figure 117 on page 733.

SDP Components

The RxSDP components that are used by these protocols are the 8b/10b Decode block, the RxBit processor, the RxSync processor, and the RxByte processor. The TxSDP components that are used by these protocols are the TxByte processor, the TxBit processor, and the 8b/10b Encode block. The receive path is shown in Figure 117 on page 733, and the transmit path is shown in Figure 118 on page 735.

8b/10b Decode Block

Since Gigabit Ethernet and FibreChannel are 10bit protocols, they each use the 8b/10b Decode block in the RxSDP to convert the 10bit physical layer encoding into the 8bit data on which the C-5e NP can operate.

FibreChannel has a slightly different Loss-of-Synchronization state machine than Gigabit Ethernet that is implemented in this block.

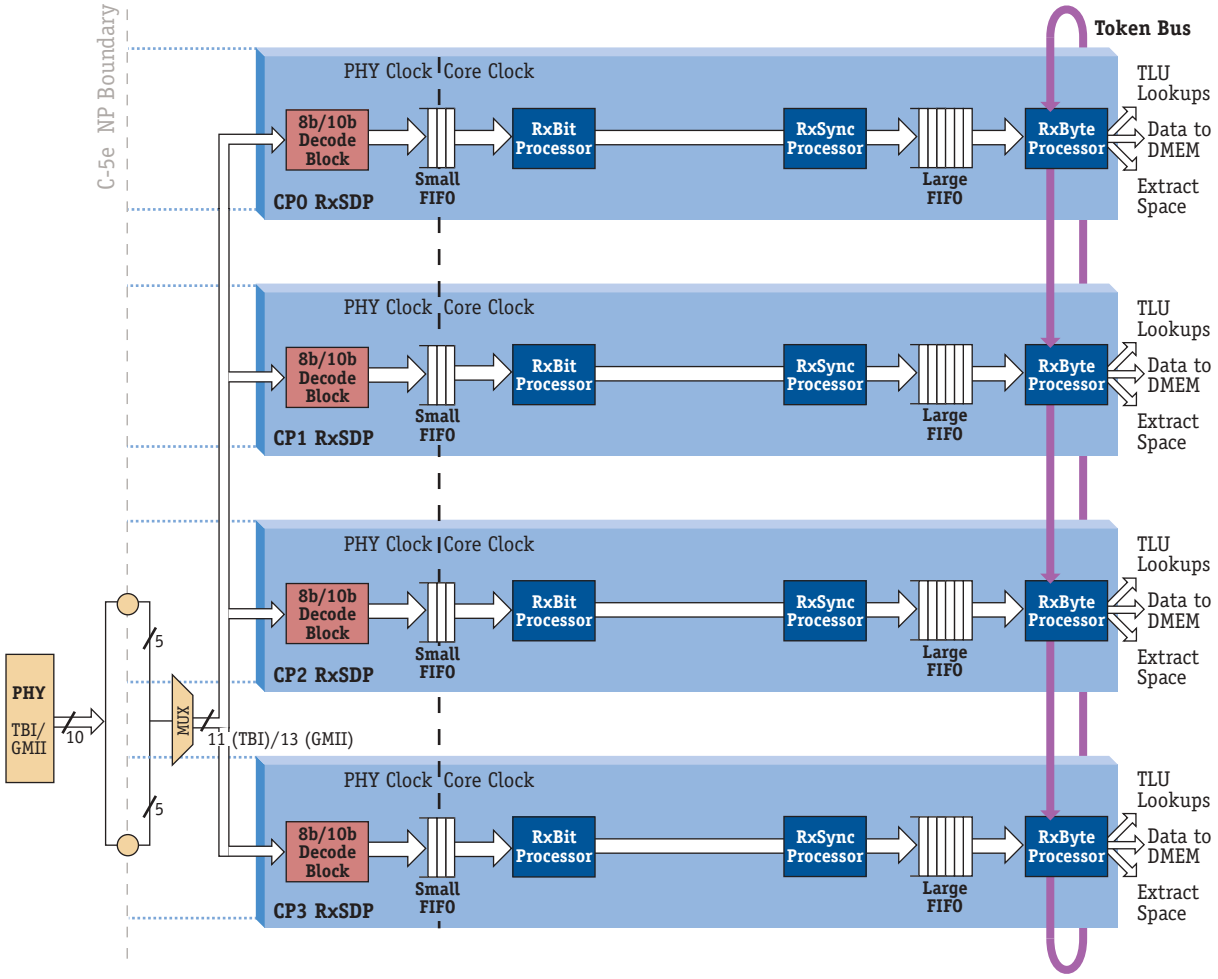
RxBit Processor

The RxBit processor delivers received packets to the RxSync processor. Functionally, the RxBit processor is used in these applications for frame delineation, stripping of control characters, and preamble for delivery of the packet without physical layer or control information into the RxSync processor.

RxSync and RxByte Processors

The RxSync processor is used in both FibreChannel and Gigabit for auxiliary processing. The RxByte processor is used for function parsing and processing of Layer 2 and above in Gigabit Ethernet and FibreChannel. In addition, the RxByte processor forwards packets when it owns the token. It then passes the token to the next RxByte processor in line. If the token is not owned by the RxByte processor, the packet is dropped.

Figure 117 SDP Receive Path for Gigabit Ethernet and FibreChannel



TxByte Processor

The TxByte processor is used in these protocols to distribute packets from the cluster of CPs to a single physical interface by way of the TxBit processor. This is done by each of the TxByte processors in a cluster filling the large FIFO with bytes, but not allowing the TxBit processor to empty its FIFO until it owns the hardware token.

Functionally, the TxByte processor does header updating and replacement, CRC and checksum recalculation of data flowing through to the physical interface.

TxBit Processor

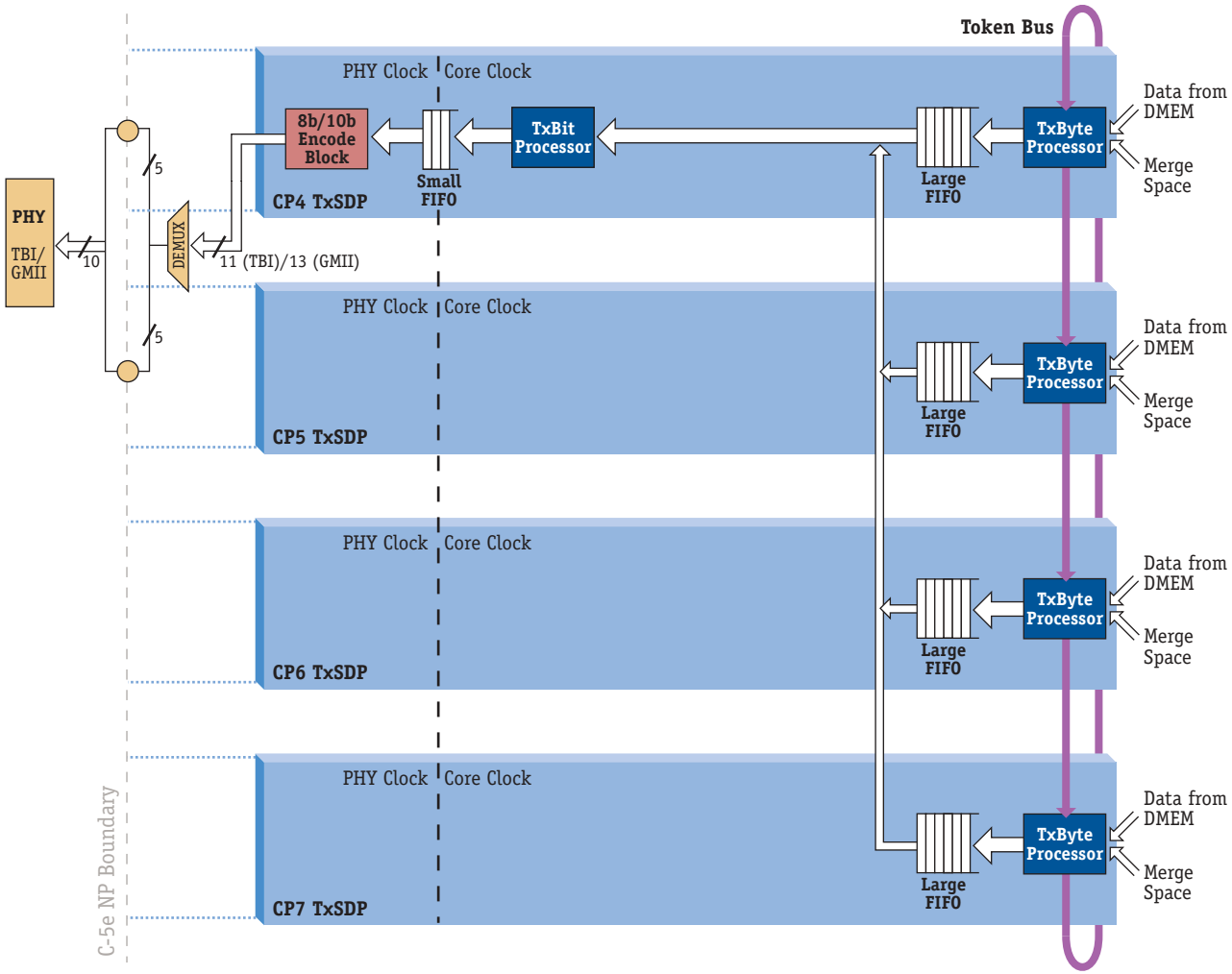
The TxBit processor is used to send the 8bit data bytes to the 8b/10b block for 10bit encoding and transmission out to the physical interface. The TxBit processor appears to receive a single stream of data from the set of large FIFOs from other TxSDPs (in order of what TxByte processor owns the hardware token). Functionally, the TxBit processor is responsible for inserting idle characters, control characters, and inter-packet gaps.

8b/10b Encode Block

The 8b/10b Encode block takes the 8bit data from the TxBit processor and does the proper 10bit encoding before transmission to the physical interface.

In the case of FibreChannel, this encoder inserts the correct End-of-Frame (EOF) word based on the 8b/10b running disparity. There is no such requirement in Gigabit Ethernet.

Figure 118 SDP Transmit Path for Gigabit Ethernet and FibreChannel



Implementation Options

Both Gigabit Ethernet and FibreChannel can be implemented in two (2) different designs.

Non-blocking Operation

This is how the reference applications in the *C-Ware Applications Library* are implemented. Clusters 0 and 2 are dedicated to receive processing and clusters 1 and 3 are dedicated to transmit processing.

The physical design implications are that the transmit pins on CP0 and CP1 on clusters 0 and 2 are tied down since they are not used. Likewise, the receive pins on CP2 and CP3 on clusters 1 and 3 are tied down since they are not used.

Blocking Operation

To have a system with twice the port density, but potentially blocks the C-5e NP, can be configured to have four (4) Gigabit Ethernet or FibreChannel ports by wiring all the receive and transmit data pins on a cluster.

That is each cluster would have a full-duplex PHY connected to it by wiring all the pins on the cluster to one (1) PHY.

OC-12 and OC-12c Applications

Both OC-12 and OC-12c applications, which include Packet-Over-SONET (POS) and ATM, can be implemented with the C-5e NP using CP aggregation. Both of these protocols use the SDP in the same configuration but have subtle differences in serialization and synchronization of the SDPs.

PHY Connectivity

The pins that the physical interface are wired to on the C-5e NP are spread out over the cluster. This is because each CP on the C-5e NP only has 7 external pins associated with it and that is too few to support OC-12 or OC-12c. Internally, the OC-12 or OC-12c data pins are replicated and sent to each of the CPs within the cluster for processing. For complete descriptions of OC-12/OC-12c pinouts, see the *C-5e NP Data Sheet*.

SDP Components

The RxSDP components that are used by these protocols are the RxBit processor, the RxSONET Framer block, the RxSync processor, and the RxByte processor. The TxSDP components that are used by these protocols are the TxByte processor, the TxSONET Framer block, and the TxBit processor. The receive path is shown in Figure 119 on page 738, and the transmit path is shown in Figure 120 on page 740.

RxBit Processor

The RxBit processor is used to perform SONET Frame delineation, that is, it locates the A1/A2 bytes and determines when it is in the Loss-Of-Frame (LOF).

RxSONET Framer

The RxSONET Framer block has different responsibilities in OC-12 and OC-12c.

- In OC-12, each SONET frame contains four (4) independent OC-3c streams. Each RxSONET Framer block processes the SONET overhead for its OC-3c stream and sends the associated payload up to the RxSync processor.
- In OC-12c, each RxSONET Framer block processes the overhead for the entire OC-12c SONET frame and sends all SONET payload up to the RxSync processor.

RxSync Processor

The RxSync processor performs ATM cell delineation for an STM cell stream, or a Point-to-Point Protocol (PPP) packet processing for Packet-over-SONET (POS) data stream.

The data stream is sightly different between OC-12 and OC-12c applications.

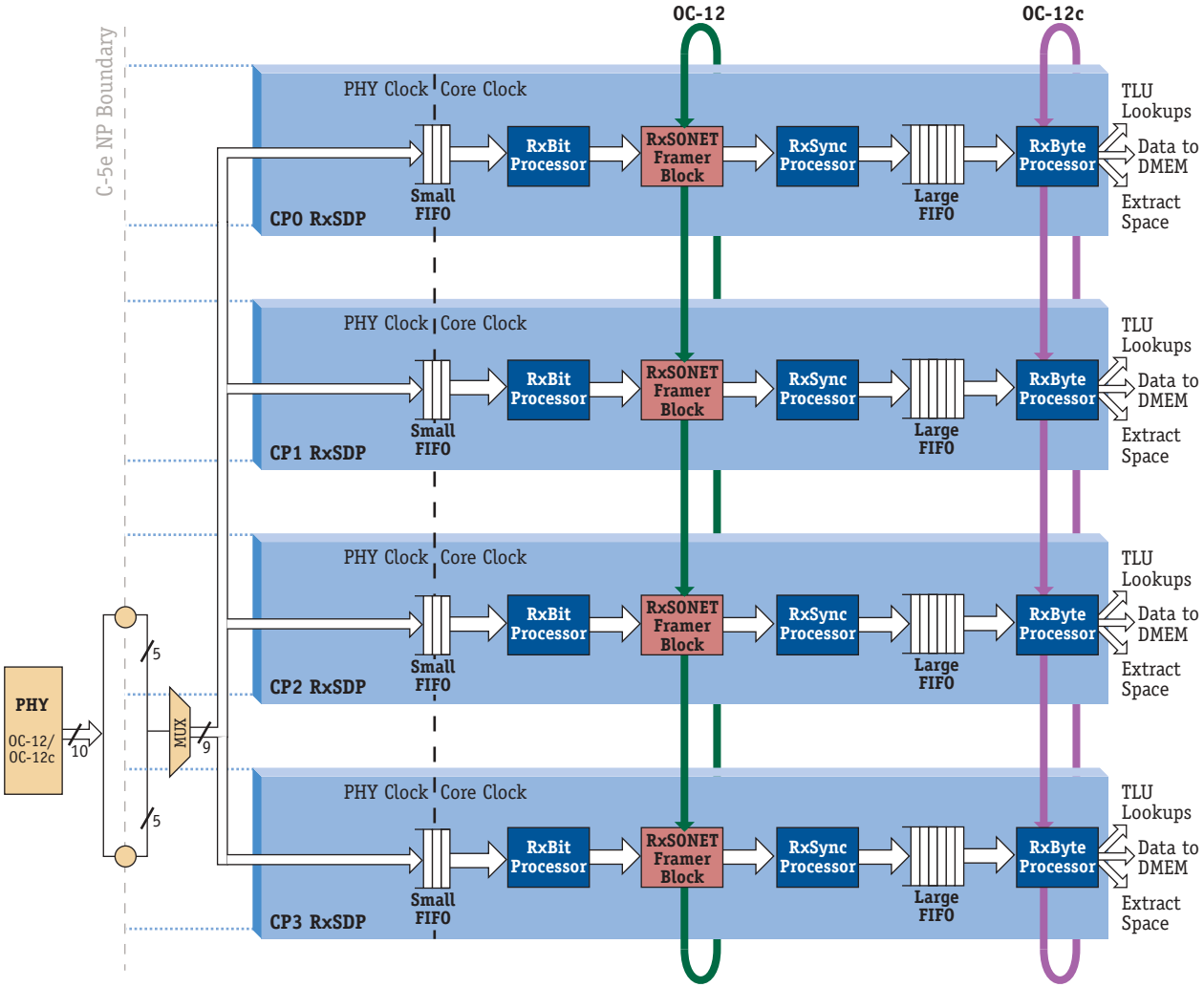
- In OC-12, the RxSync processor receives one (1) of the four (4) OC-3c payload streams.

- In OC-12c, each RxSync processor receives *all* of the payload for the entire OC-12c stream.

RxByte Processor

The RxByte processor does functional parsing of ATM cells or PPP packets and has no special aggregation function.

Figure 119 SDP Receive Path for OC-12 and OC-12c



TxByte Processor
Functionally, the TxByte processor does any type of translation, re-encapsulation, or checksum/CRC regeneration for the protocol.

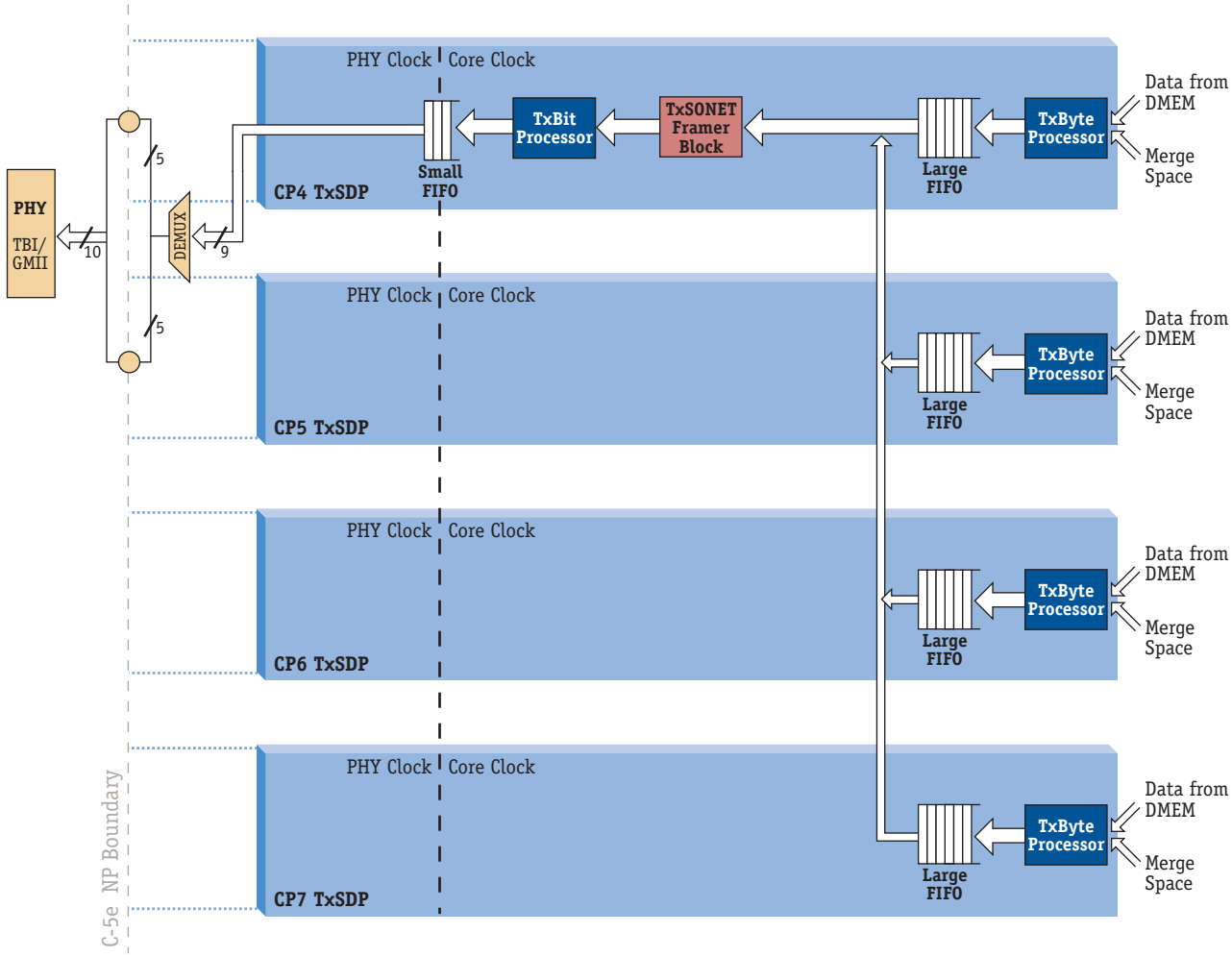
Transmit FIFO Automatic Token Passing:
The C-5e NP uses a automatic token passing mechanism. The method is for the TxLargeFIFO to check whether the *Merge9* bit is set on a payload byte. When it sees the ninth bit set, it automatically passes the token enabling the TxLargeFIFO on the next Channel Processor.

To enable this feature for the C-5e NP, set the *Auto Token Enable* bit in the *SDP_MODE4* register. Refer to "SDP_Mode4 Register (CP Mode Configuration Function)" on page 536.

TxSONET Framer
The TxSONET Framer block in these applications adds the correct SONET overhead to the OC-12 or OC-12c payload for transmission out onto the physical medium. The output of the TxSONET Framer block goes to the TxBit processor in the base CP of the cluster.

TxBit Processor
The TxBit processor sends the data bytes to the physical interface for transmission by way of the small FIFO.

Figure 120 SDP Transmit Path for OC-12 and OC-12c



Appendix C

SONET/SDH CP SUPPORT

Appendix Overview

This appendix covers the following topics:

C-5e NP SONET Support Overview

SONET monitoring is available to all SONET applications for OC-3, OC-12 and OC-12c rates. The C-5e NP RxSonet and TxSonet SDP logic blocks perform a portion of the SONET monitoring task. The tasks are performed using both hardware and software as described here:

- Specific SONET monitoring tasks are performed by *hardware* portions of the C-5e NP during monitoring include:
  - SONET OC-3c/OC-12c framing/mapping (RxSonet/TxSonet Block in SDP).
  - Identification and notification of defect conditions by the SONET framer (RxSonet block writes CREGs). The hardware times out defect conditions, not "soak" conditions.
  - Configurable SONET interrupt to notify the CPRC of interesting SONET events.
  - Configurable automatic forwarding of RDI_L, RDI_P, REI_L, and REI_P via clearing of Manual FEBE bit in SDP_Mode3 register.
  - Visibility into received SONET framers via RxSonet CREGs.
  - Ability to insert transmit overhead into the transmit SONET stream via TxSonet CREGs.
  - Accumulates up to one (1) second of B1, B2, B3, REI_P, REI_L errors for Automatic Protection Switch (APS).
  - The remaining portion of the SONET monitoring is shared between the XP and the CPRC.
- The C-Ware API's SONET Protocol Services, *software*, include a module that provides access and configuration control to the SONET blocks in the hardware. Specific monitoring tasks are performed by software portions of the C-Ware API's SONET Protocol Services during monitoring include:
  - Configure fixed transmit overhead.
  - Monitor and soak defects, for example, LOS, LOF and others.

Also, SONET Reference Applications are provided that implement the features of the C-5e NP and the API.

SONET/SDH Overview

SONET/SDH (Synchronous Optical Network/Synchronous Digital Hierarchy) is a standard for optical transport that defines optical carrier levels and their electrically equivalent synchronous transport signals. SONET allows for a multi-vendor environment and positions the network for transport of new services, synchronous networking, and enhanced Operation, Administration, Maintenance and Provisioning (OAM&P).

The C-5e NP SONET/SDH transmit support consists of inserting payload into the SONET/SDH frame on the transmit side. The SONET/SDH frame overhead data is read from the Channel Processor (CP) Configuration registers.

The SONET/SDH receive support consists of extracting payload from the SONET/SDH frame and forwarding this payload to the large FIFO of the CPRC. The SONET/SDH frame overhead data is written to the Channel Processor (CP) Configuration registers.

The C-5e NP allows access to a large portion of the SONET/SDH Overhead Read Directory by the CPRC or the XP/Host via the Global Bus. This allows a given application the flexibility to add code to support such features as Orderwire or Data Communication Channels.

The C-5e NP Supports three (3) SONET/SDH configurations internally:

- OC-3c,
- OC-12c,
- OC-12 with 4 separate OC-3c embedded streams.

With the addition of the M-5 Channel Adapter and an external OC-48 framer device the C-5e NP can also support OC-48.

The C-5e NP conforms with both SONET and SDH standards. Therefore, OC-3 is used instead of (STS-3/STM-1), OC-12 is used instead of (STS-12/STM-4, and OC-48 is used instead of (STS-48/STM-16).

In addition, to the SONET/SDH Overhead access on both the transmit (Tx) and receive (Rx) side, the SONET/SDH block provides a function for defect monitoring purposes. SONET/SDH defect events are flagged in the *SONET_Event* register. An example of the types of events supported are listed in Table 237 on page 744.

Table 237 Example of Events Reported in the SONET_Event Register

EVENT CATEGORY	EXAMPLES	USE
Defects (Non-Pointer related)	LOS, LOF, C2 error (PLM-P), LCD-P	Near-end fault detection.
	AIS-L, RDI-L, AIS-P, RDI-P	Far-end fault detection.
Counters	B1, B2, B3	Near-end error rate detection.
	REI-L, REI-P	Far-end error rate detection.
Pointer Defects	LOP-P, PTR-Change, NDF, H4 Change	Near-end fault detection.
APS Support	APS-Error, K2 Change	Switching protection.
Other OH Support	Z1, Z2, Z3, Z4, Z5 Change, S1 Change	Synchronization states and country specific SDH support.
Trail Trace Support	J0 Change, J1 Available	J0/J1 Path & Section trace support.
General Support	Tx overhead complete, Rx transport overhead available, Rx path overhead available	Useful for updating overhead on Tx and checking other SONET overhead on Rx when no other interrupt is available.

In addition, the SONET block can automatically forward far-end alarm indications on the same port when errors are detected on the line (for example, LOS, LOF, B1, etc.) This is done when the *SDP_Mode3* register bit [13] *Manual_FEBE* field is set to 0.

SONET/SDH Overhead Access

Configuration Space of the CPs includes a number of registers that pertain to implementing and using the SONET/SDH functions. Primarily, sixty-four (64) (4Byte) (256Byte total) registers are used for storing receive (Rx) and transmit (Tx) SONET overhead. The SDP RxSONET Framer block writes bytes to the space in a manner similar to way the SDP RxByte Processor writes to Extract Space. The CPRC can read the Receive SONET registers at any time. The CPRC can write to the Receive SONET registers, but only during initialization and test periods (when the *SDP_Mode3* register bit [30] *RxEnable* field is clear).

The SDP TxSONET Framer block reads bytes from the space, similar to the way merge registers are used by the SDP TxByte Processor. The CPRC process can read or write the Transmit SONET registers at any time. Refer to "Rx_SONETOH0 to Rx_SONETOH31 Registers (CP SONET Rx Control Function)" on page 517 and "Tx_SONETOH0 to Tx_SONETOH31 Registers (CP SONET Tx Control Function)" on page 517.

For detail descriptions of the SONET/SDH transport overhead bytes. Refer to *Telcordia GR-253 Core Synchronous Optical Network (SONET) Transport Systems: Common Generic Criteria, (Issue 3, September 2000)*.

SONET/SDH Frame Format Overview

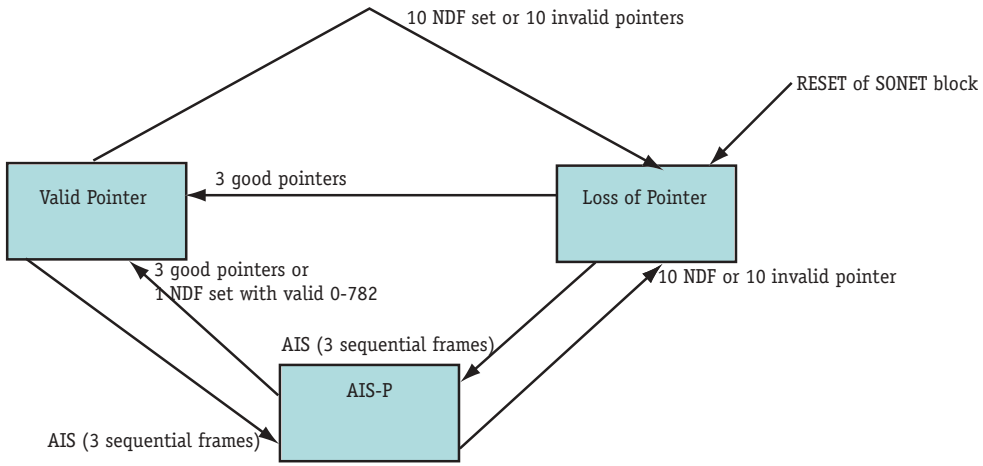
The basic frame format consists of 9 x 90Bytes (9rows x 90columns) called an STS-1. An STS-1 frame contains 810Bytes and is transmitted in 125µs for a bit rate of 51.840Mbps. Higher aggregate data rates can be accommodated by combining multiple STS-1 frames to create STS-N frames. STS-N frames consist of N Byte interleaved STS-1 frames. To carry single higher rate payloads, frames can be interleaved with fixed phase alignment to form concatenated frames. For example, an STS-3c (concatenated) frame can be used to transport a payload with bit rate up to 155.52Mbps where a STS-3 frame can carry three (3) distinct 51.840Mbps payloads.

The SONET/SDH frame format is divided into two (2) main areas: Synchronous Payload Envelope (SPE) and Transport Overhead (TOH). The SPE contains the information being transported by the frame. The TOH supports the Operation, Administration, Maintenance and Provisioning (OAM&P) functions of SONET/SDH, and includes a data communication channel that provides an OAM&P communication path between multiple interconnected SONET/SDH network elements.

The ninety (90) columns are allocated as follows: the first three (3) columns are reserved for the TOH, while the remaining eighty-seven (87) columns are for the SPE. Transmission is row by row starting with the byte in the upper left corner and ending with the byte in the lower right corner. The nine (9) rows of the TOH are allocated: first three (3) are the STS-1 Section Overhead, while the remaining six (6) rows are for the STS-1 Line Overhead. The TOH is able to couple the functions of certain overhead bytes to the network architecture by using the Section OH and the Line OH. Also the TOH provides a pointer or pointers to the SPE(s). The STS Path Overhead (STS POH) is part of the SPE. The STS POH has the task of monitoring quality and indicating the contents of SPE. The TOH's pointer points to a different destination based on the STS level. For OC-3c and OC-12c, it points to the STS-3c or STS-12c Path overhead J1 byte. For OC-12, four (4) separate pointers identify four (4) separate J1 bytes. Refer to Figure 122 on page 747.

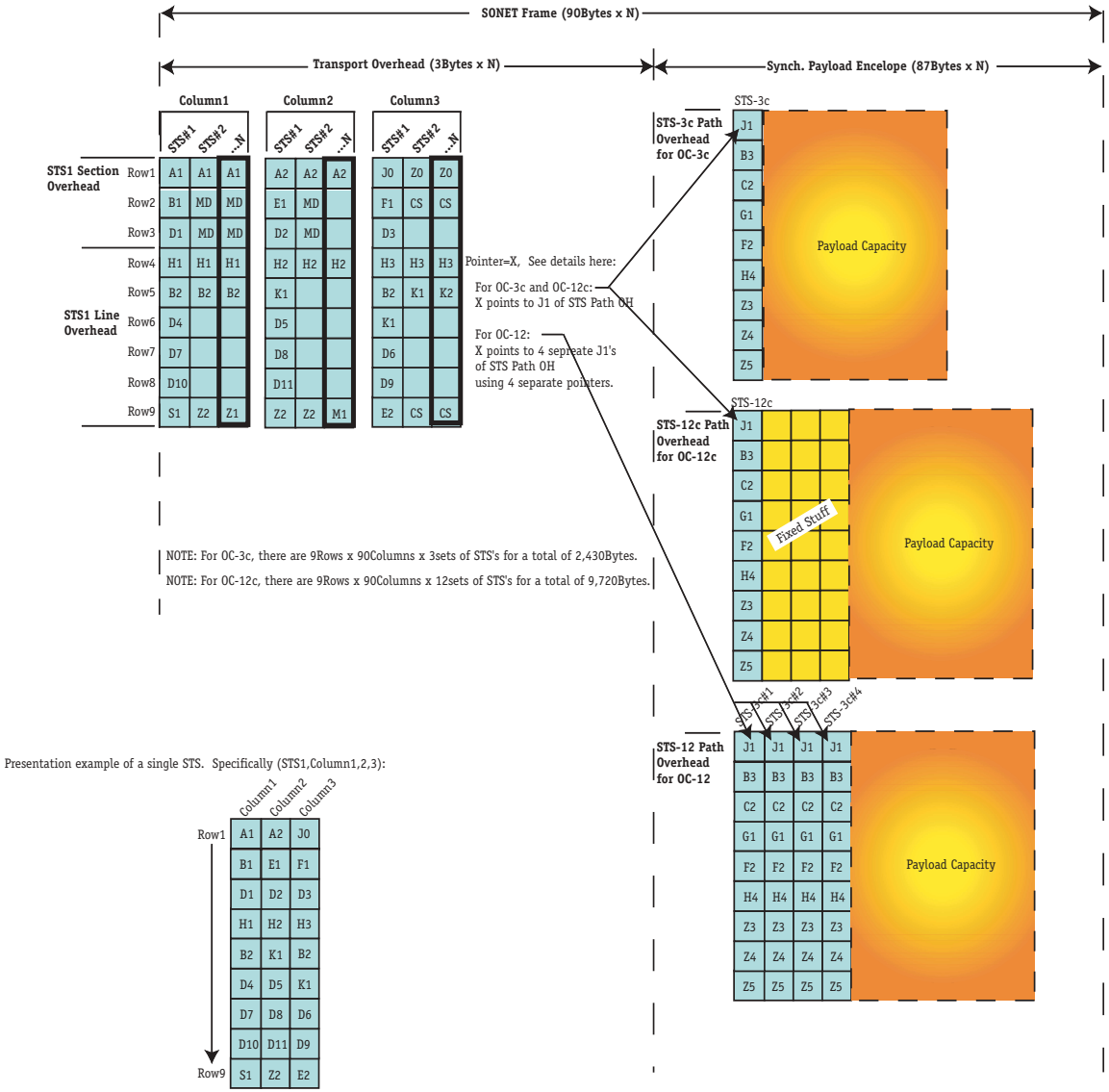
Figure 121 on page 746 shows the receive SONET pointer state machines operation. Refer to Table 239 on page 750 and Table 244 on page 763 for the corresponding transport overhead byte H1, STS #1 for receive SONET/SDH OC-3c and OC-12/OC-12c.

Figure 121 Receive SONET Pointer State Machine



Invalid pointer is declared as follows:
If all ones, Valid pointer
If 8 of 10 rule is met for increment or decrement decision - VALID Pointer
Else If out of range (0-782) INVALID Pointer
If NDF clear, and doesn't match previous pointer - INVALID Pointer

Figure 122 SONET/SDH Frame Format



The SONET OC-3c, SONET OC-12 and OC-12c Overhead positions and definitions for both the Rx (readable) and Tx (writable) sides, as well as, detail mapping information listing the SONET/SDH overhead definitions and C-5e NP addresses and whether the overhead contents are Transport or Path bytes are covered in the following sections. Refer to Table 238 on page 748.

Table 238 Quick Reference to Applicable SONET/SDH Information

STANDARD	FLOW	POSITIONS	TRANSPORT DEFINITIONS	PATH DEFINITIONS	STATISTICS COUNTER DEFINITIONS (TRANSPORT & PATH)
SONET Overhead OC-3c	Rx	See Figure 123 on page 749.	See Table 239 on page 750.	See Table 240 on page 754.	See Table 241 on page 755.
	Tx	See Figure 124 on page 757.	See Table 242 on page 758.	See Table 243 on page 761.	N/A
SONET Overhead OC-12/OC-12c	Rx	See Figure 125 on page 762.	See Table 244 on page 763.	See Table 245 on page 774.	See Table 246 on page 777.
	Tx	See Figure 126 on page 781.	See Table 247 on page 782.	See Table 248 on page 790.	N/A

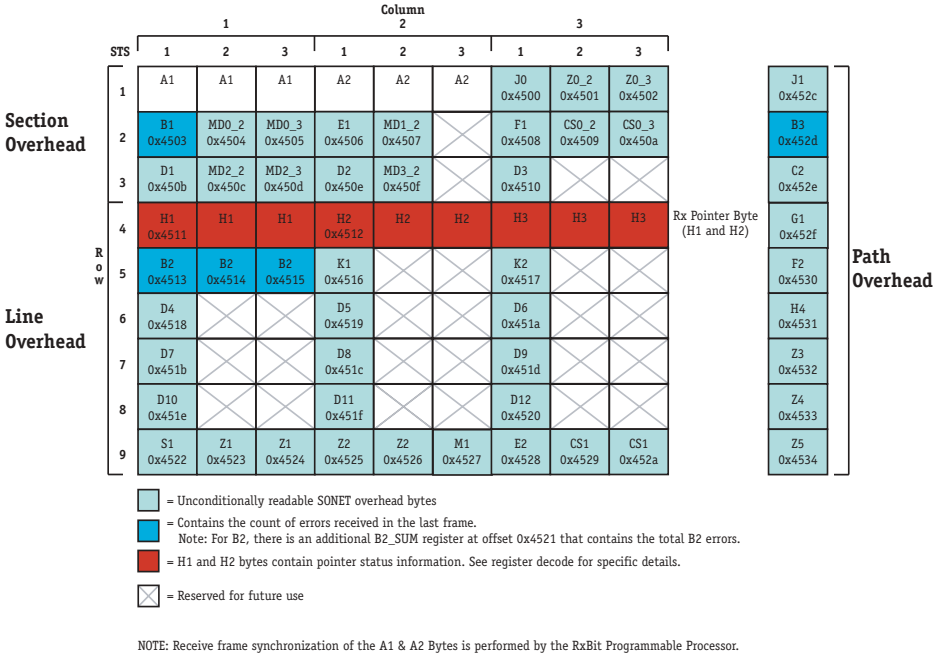
SONET/SDH OC-3c Overhead Bytes

This section provides the SONET/SDH OC-3c Overhead positions and definitions for both the Rx (readable) and Tx (writable) sides. Also, the detail mapping information listing the SONET/SDH overhead definitions and C-5e NP addresses and whether the overhead contents are Transport or Path bytes. For the Rx side, refer to Figure 123 on page 749, Table 239 on page 750, Table 240 on page 754 and Table 241 on page 755. For the Tx side, refer to Figure 124 on page 757, Table 242 on page 758, Table 243 on page 761 and Table 246 on page 777.

Receive OC-3c Readable Overhead Bytes Positions

Figure 123 on page 749 shows the readable bytes in the OC-3c SONET/SDH Overhead.

Figure 123 Rx SONET/SDH OC-3c Readable Overhead Bytes



NOTE: Receive frame synchronization of the A1 & A2 Bytes is performed by the RxBit Programmable Processor.

Receive OC-3c Transport Overhead Definitions

Table 239 on page 750 lists the SONET/SDH Transport Overhead definitions and addresses.

Table 239 Receive SONET/SDH OC-3c Transport Overhead Byte Addresses

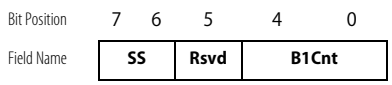
TRANSPORT OVERHEAD BYTE	C-5E NP ADDRESS	NOTES
J0	0xBC*n*04500	The J0 value received in (1,3,1) (row, column, STS) position of the last frame is written to this location.
Z0, STS #2	0xBC*n*04501	Z0 growth byte value received in last frame for position (1,3,2).
Z0, STS #3	0xBC*n*04502	Z0 growth byte value received in last frame for position (1,3,3).
B1_SSH1	0xBC*n*04503	The actual B1 (2,1,1) parity byte value is not written to the B1_SSH1 register. Instead, the number of bit lanes in error is provided. The number of errors reported is therefore 0 through 8 reported in the lower 5 bits of B1_SSH1. The upper 2 bits of this register indicate the value of the SS bits received in the H1 byte. 
MD0, STS #2	0xBC*n*04504	SDH Media Dependent byte value received in last frame for position (2,1,2).
MD0, STS #3	0xBC*n*04505	SDH Media Dependent byte value received in last frame for position (2,1,3).
E1	0xBC*n*04506	Orderwire E1 byte value received in last frame for position (2,2,1).
MD1, STS #2	0xBC*n*04507	SDH Media Dependent byte value received in last frame for position (2,2,2).
F1	0xBC*n*04508	F1 byte value received in last frame for position (2,3,1).
CS0, STS #2	0xBC*n*04509	SDH Country Specific byte value received in last frame for position (2,3,2).
CS0, STS #3	0xBC*n*0450A	SDH Country Specific byte value received in last frame for position (2,3,3).
D1	0xBC*n*0450B	Datacom Channel 1 byte value received in last frame for position (3,1,1).

Table 239 Receive SONET/SDH OC-3c Transport Overhead Byte Addresses (continued)

TRANSPORT OVERHEAD BYTE	C-5E NP ADDRESS	NOTES
MD2, STS #2	0xBC*n*0450C	SDH Media Dependent byte value received in last frame for position (3,1,2).
MD2, STS #3	0xBC*n*0450D	SDH Media Dependent byte value received in last frame for position (3,1,3).
D2	0xBC*n*0450E	Datacom Channel 2 byte value received in last frame for position (3,2,1).
MD3, STS #2	0xBC*n*0450F	SDH Media Dependent byte value received in last frame for position (3,2,2).
D3	0xBC*n*04510	Datacom Channel 3 byte value received in last frame for position (3,3,1).
H1, STS #1	0xBC*n*04511	For this H1 (4,1,1) location, it is not the actual values of H1 that is written, but the pointer processing results listed here: **BITS** / **DESCRIPTION** [7:6] Pointer State Where: 0=Valid Pointer 2=AIS-P is observed (pointer is all 1's) 3= Loss of Pointer (LOP-P) [5] New Data Flag occurred (NDF) [4] Pointer Increment occurred [3] Pointer Decrement occurred [2] Zero [1] Current Pointer Value bit9 [0] Current Pointer Value bit8 The SONET pointer interpreter meets the following GR-253 Issue 3 requirements: • R3-98 - NDF 3 of 4 bits correct determines NDF set - tolerant of single bit errors. • O3-96 - 8 of 10 voting for pointer increment decrement decision. • R3-104 - is met for pointer interpretation.

Table 239 Receive SONET/SDH OC-3c Transport Overhead Byte Addresses (continued)

TRANSPORT OVERHEAD BYTE	C-5E NP ADDRESS	NOTES
H2, STS #1	0xBC*n*04512	For this H2 location, it is not the actual values of H2 (4,2,1) that is written, but the value of the current pointer value bits 7:0.
B2_CNT, STS #1	0xBC*n*04513	The actual B2 parity byte value in (5,1,1) is not written to the B2_CNT STS #1 register. Instead, the sum of the number of bit lanes in error for B2 STS #1 is provided. The number of errors reported is therefore 0 through 8.
B2_CNT, STS #2	0xBC*n*04514	The actual B2 parity byte value in (5,1,2) is not written to the B2_CNT STS #2 register. Instead, the sum of the number of bit lanes in error for B2 STS #2. This value ranges from 0-8.
B2_CNT, STS #3	0xBC*n*04515	The actual B2 parity byte value in (5,1,3) is not written to the B2_CNT STS #3 register. Instead, the sum of the number of bit lanes in error for B2 STS #3 is provided. This value ranges from 0-8.
K1	0xBC*n*04516	The K1 byte (5,2,1) is written to the registers only when three identical bytes have been received in consecutive frames.
K2	0xBC*n*04517	The K2 byte (5,3,1) is written to the registers only when three identical bytes have been received in consecutive frames.
D4	0xBC*n*04518	Datacom Channel 4 byte value received in last frame for position (6,1,1).
D5	0xBC*n*04519	Datacom Channel 5 byte value received in last frame for position (6,2,1).
D6	0xBC*n*0451A	Datacom Channel 6 byte value received in last frame for position (6,3,1).
D7	0xBC*n*0451B	Datacom Channel 7 byte value received in last frame for position (7,1,1).
D8	0xBC*n*0451C	Datacom Channel 8 byte value received in last frame for position (7,2,1).
D9	0xBC*n*0451D	Datacom Channel 9 byte value received in last frame for position (7,3,1).
D10	0xBC*n*0451E	Datacom Channel 10 byte value received in last frame for position (8,1,1).

Table 239 Receive SONET/SDH OC-3c Transport Overhead Byte Addresses (continued)

TRANSPORT OVERHEAD BYTE	C-5E NP ADDRESS	NOTES
D11	0xBC*n*0451F	Datacom Channel 11 byte value received in last frame for position (8,2,1).
D12	0xBC*n*04520	Datacom Channel 12 byte value received in last frame for position (8,3,1).
B2_SUM	0xBCn04521	The sum of all B2 errors received in the last frame is written to this register. For OC3c, this is the same value as that received in B2, STS #1 above.
S1	0xBC*n*04522	S1 Synchronization Status byte value received in last frame for position (9,1,1).
Z1, STS #2	0xBC*n*04523	Z1 growth byte value received in last frame for position (9,1,2).
Z1, STS #3	0xBC*n*04524	Z1 growth byte value received in last frame for position (9,1,3).
Z2, STS #1	0xBC*n*04525	Z2 growth byte value received in last frame for position (9,2,1).
Z2, STS #2	0xBC*n*04526	Z2 growth byte value received in last frame for position (9,2,2).
M1	0xBC*n*04527	M1 REI-L byte value received in last frame for position (9,2,3).
E2	0xBC*n*04528	Orderwire E2 byte value received in last frame for position (9,3,1).
CS1, STS #2	0xBC*n*04529	SDH Country Specific 1 byte value received in last frame for position (9,3,2).
CS1, STS #3	0xBC*n*0452A	SDH Country Specific 1 byte value received in last frame for position (9,3,3).

Receive OC-3c Path Overhead Definitions

Table 240 on page 754 lists the SONET/SDH Path Overhead definitions and addresses.

Table 240 Receive SONET/SDH OC-3c Path Overhead Byte Addresses

PATH OVERHEAD BYTE	C-5E NP ADDRESS	NOTES
J1, STS #1	0xBC*n*0452C	The J1 register contains the Nth J1 of the 64 (or 16) byte path trace message. The value N is obtained by the hardware from the RxSonet J1 Index field of the SDP_Mode2 register and is under CPRC software control. To read the complete message for the first time, increment the index N after every frame.
B3_CNT, STS #1	0xBC*n*0452D	The actual B3 parity byte value is not written to the B3_CNT STS #1 register. Instead, the sum of the number of bit lanes in error for B3 STS #1 is provided. The number of errors reported is therefore 0 through 8.
C2, STS #1	0xBC*n*0452E	C2 path signal label value received in the last frame.
G1, STS #1	0xBC*n*0452F	G1 path overhead value received in the last frame.
F2, STS #1	0xBC*n*04530	F2 path overhead value received in the last frame.
H4, STS #1	0xBC*n*04531	H4 path overhead value received in the last frame.
Z3, STS #1	0xBC*n*04532	Z3 path overhead value received in the last frame.
Z4, STS #1	0xBC*n*04533	Z4 path overhead value received in the last frame.
Z5, STS #1	0xBC*n*04534	Z5 path overhead value received in the last frame.

Receive OC-3c Statistics Counters for Both Transport and Path Overhead

Table 241 on page 755 lists the SONET/SDH Statistics Counters for both Transport and Path Overhead. Their definitions and addresses are provided. Statistics Counters collect both SONET/SDH Transport and Path Overhead on a per frame basis and can be read on a one second basis.

Table 241 Receive SONET/SDH OC-3c Statistics Counters Byte Addresses

TRANSPORT AND PATH OVERHEAD BYTES	C-5E NP ADDRESS	NOTES
B1_ACCUM[15:8]	0xBCn04536	Bits [15:8] of B1_ACCUM, the Accumulated B1 Bit Interleaved Parity Error registers. The accumulated B1 registers contain a running total of the number of B1 errors. If a rising edge is observed on the FrameMode signal, this count is cleared to zero and begins incrementing again as errors are observed. Note: These registers are sized to allow up to 1 seconds worth of errors to be accumulated before wrapping.
B1_ACCUM[7:0]	0xBCn04537	Bits [7:0] of B1_ACCUM, the Accumulated B1 Bit Interleaved Parity Error registers.
B2_ACCUM[19:16]	0xBCn04538	Bits [19:16] of B2_ACCUM, the Accumulated B2 Bit Interleaved Parity Error registers. The accumulated B2 registers contain a running total of the number of B2 errors. If a rising edge is observed on the FrameMode signal, this count is cleared to zero and begins incrementing again as errors are observed. Note that these registers are sized to allow up to 1 seconds worth of errors to be accumulated before wrapping.
B2_ACCUM[15:8]	0xBCn04539	Bits [15:8] of B2_ACCUM, the Accumulated B2 Bit Interleaved Parity Error registers.
B2_ACCUM[7:0]	0xBCn0453A	Bits [7:0] of B2_ACCUM, the Accumulated B2 Bit Interleaved Parity Error registers.
B3_ACCUM[15:8]	0xBCn0453C	Bits [15:8] of B3_ACCUM, the Accumulated B3 Bit Interleaved Parity Error registers. The accumulated B3 registers contain a running total of the number of B3 errors. If a rising edge is observed on the FrameMode signal, this count is cleared to zero and begins incrementing again as errors are observed. Note: These registers are sized to allow up to 1 seconds worth of errors to be accumulated before wrapping.

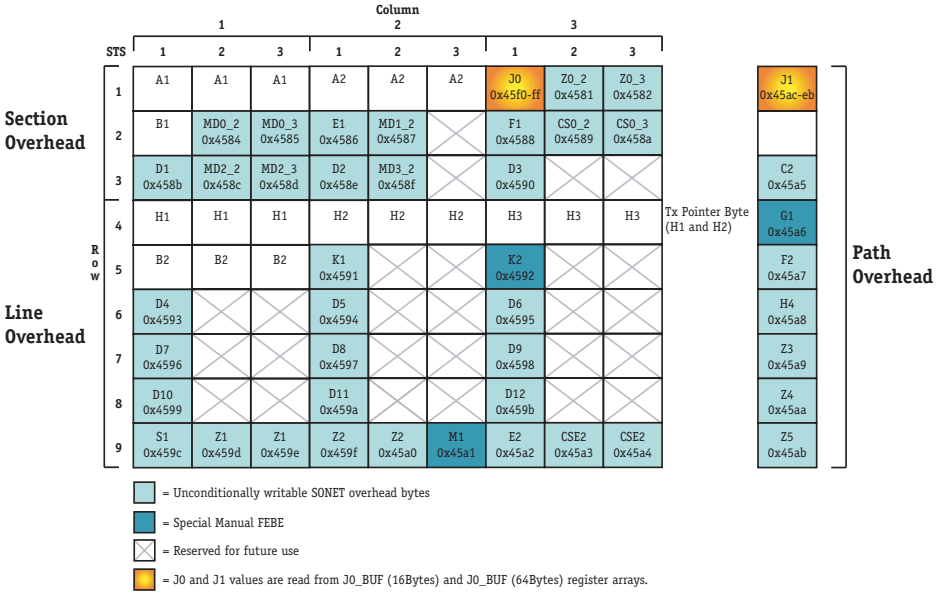
Table 241 Receive SONET/SDH OC-3c Statistics Counters Byte Addresses (continued)

TRANSPORT AND PATH OVERHEAD BYTES	C-5E NP ADDRESS	NOTES
B3_ACCUM[7:0]	0xBCn0453D	Bits [7:0] of B3_ACCUM, the Accumulated B3 Bit Interleaved Parity Error registers.
REI_P_ACCUM[15:8]	0xBCn0453E	Bits [15:8] of REI_P_ACCUM, the Accumulated Path Remote Error Indication registers. The accumulated REI_P registers contain a running total of the number of REI_P errors seen by the far end. If a rising edge is observed on the FrameMode signal, this count is cleared to zero and begins incrementing again as errors are observed. Note: These registers are sized to allow up to 1 seconds worth of errors to be accumulated before wrapping.
REI_P_ACCUM[7:0]	0xBCn0453F	Bits [7:0] of REI_P_ACCUM, the Accumulated Path Remote Error Indication registers.
REI_L_ACCUM[19:16]	0xBCn04541	Bits [19:16] of REI_L_ACCUM, the Accumulated Line Remote Error Indication registers. The accumulated REI_L registers contain a running total of the number of REI_L errors seen by the far end. If a rising edge is observed on the FrameMode signal, this count is cleared to zero and begins incrementing again as errors are observed. Note: These registers are sized to allow up to 1 seconds worth of errors to be accumulated before wrapping.
REI_L_ACCUM[15:8]	0xBCn04542	Bits [15:8] of REI_L_ACCUM, the Accumulated Line Remote Error Indication registers.
REI_L_ACCUM[7:0]	0xBCn04543	Bits [7:0] of REI_L_ACCUM, the Accumulated Line Remote Error Indication registers.

Transmit OC-3c Writable Overhead Bytes Positions

Figure 124 on page 757 shows the writable bytes in the OC-3c SONET/SDH Overhead.

Figure 124 Tx SONET/SDH OC-3c Writable Overhead Bytes



Transmit OC-3c Transport Overhead Definitions

Table 242 on page 758 lists the SONET/SDH Transport Overhead definitions and addresses.

Table 242 Transmit SONET/SDH OC-3c Transport Overhead Byte Addresses

TRANSPORT OVERHEAD BYTE	C-5E NP ADDRESS	NOTES
Z0, STS #2	0xBC*n*04581	Z0 growth byte value to be sent in the next frame for position (1,3,2) (row, column, STS).
Z0, STS #3	0xBC*n*04582	Z0 growth byte value to be sent in the next frame for position (1,3,3).
MD0, STS #2	0xBC*n*04584	SDH Media Dependent byte position (2,1,2).
MD0, STS #3	0xBC*n*04585	SDH Media Dependent byte position (2,1,3).
E1	0xBC*n*04586	Orderwire E1 byte value to be sent in the next frame for position (2,2,1).
MD1, STS #2	0xBC*n*04587	SDH Media Dependent byte value to be sent in the next frame for position (2,2,2).
F1	0xBC*n*04588	F1 byte value to be sent in the next frame for position (2,3,1).
CS0, STS #2	0xBC*n*04589	SDH Country Specific byte value to be sent in the next frame for position (2,3,2).
CS0, STS #3	0xBC*n*0458A	SDH Country Specific byte value to be sent in the next frame for position (2,3,3).
D1	0xBC*n*0458B	Datacom Channel 1 byte value to be sent in the next frame for position (3,1,1).
MD2, STS #2	0xBC*n*0458C	SDH Media Dependent byte value to be sent in the next frame for position (3,1,2).
MD2, STS #3	0xBC*n*0458D	SDH Media Dependent byte value to be sent in the next frame for position (3,1,3).
D2	0xBC*n*0458E	Datacom Channel 2 byte value to be sent in the next frame for position (3,2,1).
MD3, STS #2	0xBC*n*0458F	SDH Media Dependent byte value to be sent in the next frame for position (3,2,2).
D3	0xBC*n*04590	Datacom Channel 3 byte value to be sent in the next frame for position (3,3,1).
K1	0xBC*n*04591	The K1 byte value to be sent in the next frame for position (5,2,1).

Table 242 Transmit SONET/SDH OC-3c Transport Overhead Byte Addresses (continued)

TRANSPORT OVERHEAD BYTE	C-5E NP ADDRESS	NOTES
K2	0xBC*n*04592	The K2 byte value to be sent in the next frame for position (5,3,1). See *Manual_FEBE* description in *SDP_Mode3* register for exceptions.
D4	0xBC*n*04593	Datacom Channel 4 byte value to be sent in the next frame for position (6,1,1).
D5	0xBC*n*04594	Datacom Channel 5 byte value to be sent in the next frame for position (6,2,1).
D6	0xBC*n*04595	Datacom Channel 6 byte value to be sent in the next frame for position (6,3,1).
D7	0xBC*n*04596	Datacom Channel 7 byte value to be sent in the next frame for position (7,1,1).
D8	0xBC*n*04597	Datacom Channel 8 byte value to be sent in the next frame for position (7,2,1).
D9	0xBC*n*04598	Datacom Channel 9 byte value to be sent in the next frame for position (7,3,1).
D10	0xBC*n*04599	Datacom Channel 10 byte value to be sent in the next frame for position (8,1,1).
D11	0xBC*n*0459A	Datacom Channel 11 byte value to be sent in the next frame for position (8,2,1).
D12	0xBC*n*0459B	Datacom Channel 12 byte value to be sent in the next frame for position (8,3,1).
S1	0xBC*n*0459C	S1 Synchronization Status byte value to be sent in the next frame for position (9,1,1).
Z1, STS #2	0xBC*n*0459D	Z1 growth byte value to be sent in the next frame for position (9,1,2).
Z1, STS #3	0xBC*n*0459E	Z1 growth byte value to be sent in the next frame for position (9,1,3).
Z2, STS #1	0xBC*n*0459F	Z2 growth byte value to be sent in the next frame for position (9,2,1).
Z2, STS #2	0xBC*n*045A0	Z2 growth byte value to be sent in the next frame for position (9,2,2).
M1	0xBC*n*045A1	M1 REI-L byte value to be sent in the next frame for position (9,2,3). See *Manual_FEBE* description in *SDP_Mode3* register for exceptions.

Table 242 Transmit SONET/SDH OC-3c Transport Overhead Byte Addresses (continued)

TRANSPORT OVERHEAD BYTE	C-5E NP ADDRESS	NOTES
E2	0xBC*n*045A2	Orderwire E2 byte value to be sent in the next frame for position (9,3,1).
CSE2, STS #2	0xBC*n*045A3	SDH Country Specific 1 byte value to be sent in the next frame for position (9,3,2).
CSE2, STS #3	0xBC*n*045A4	SDH Country Specific 1 byte value to be sent in the next frame for position (9,3,3).
J0_BUF	0xBC*n*045F0 to FF	The J0 section trace message to be transmitted is written to these registers.
H1_SS, STS #1	0xBC*n*45EC	For SDH, the transmit SS bits *must* be initialized for backward compatibility. This register allows the SS bits of the pointer to be initialized. The two LSB of the register are written to the H1 SS bits of the Transmit SONET/SDH frame.

Transmit OC-3c Path Overhead Definitions

Table 243 on page 761 lists the SONET/SDH Path Overhead definitions and addresses.

Table 243 Transmit SONET/SDH OC-3c Path Overhead Byte Addresses

PATH OVERHEAD BYTE	C-5E NP ADDRESS	NOTES
C2, STS #1	0xBC*n*045A5	C2 path signal label value to be transmitted in the next frame.
G1, STS #1	0xBC*n*045A6	G1 path overhead value to be transmitted in the next frame. See *Manual_FEBE* description in *SDP_Mode3* register for exceptions.
F2, STS #1	0xBC*n*045A7	F2 path overhead value to be transmitted in the next frame.
H4, STS #1	0xBC*n*045A8	H4 path overhead value to be transmitted in the next frame.
Z3, STS #1	0xBC*n*045A9	Z3 path overhead value to be transmitted in the next frame.
Z4, STS #1	0xBC*n*045AA	Z4 path overhead value to be transmitted in the next frame.
Z5, STS #1	0xBC*n*045AB	Z5 path overhead value to be transmitted in the next frame.
J1_BUF, STS #1	0xBC*n*045AC to 0xBC*n*045EB	The J1 path trace message to be transmitted is written to these registers. • To support a 64Byte path, write the complete 64Byte message to these registers. • To support a 16Byte path trace, write the 16Byte message consecutively 4 times to these registers.

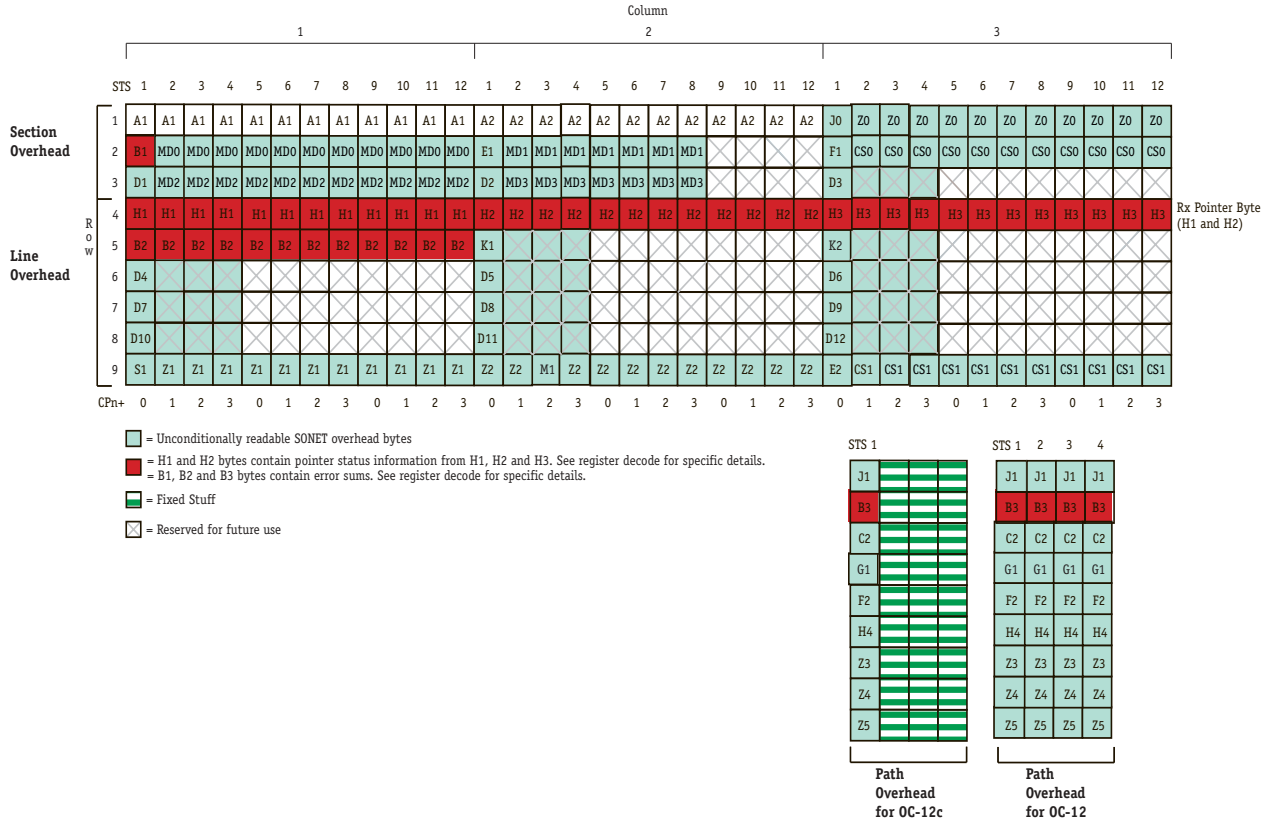
SONET/SDH OC-12 and OC-12c Overhead Bytes

This section provides the SONET/SDH OC-12 and OC-12c Overhead positions and definitions for both the Rx (readable) and Tx (writable) sides. Also, the detail mapping information listing the SONET/SDH overhead definitions and C-5e NP addresses and whether the overhead contents are Transport or Path bytes. For the Rx side, refer to Figure 125 on page 762, Table 244 on page 763 and Table 245 on page 774. For the Tx side, refer to Figure 126 on page 781, Table 247 on page 782 and Table 248 on page 790.

Receive OC-12/OC-12c Readable Overhead Bytes

Figure 125 on page 762 shows the readable bytes in the OC-12/OC-12c SONET/SDH Overhead.

Figure 125 Rx SONET/SDH OC-12/OC-12c Readable Overhead Bytes



Receive OC-12/OC-12c Transport Overhead Definitions

Table 244 on page 763 lists the SONET/SDH Transport Overhead definitions and addresses.

Table 244 Receive SONET/SDH OC-12 and OC-12c Transport Overhead Byte Addresses

TRANSPORT OVERHEAD BYTE	CP# WITHIN A CLUSTER	C-5E NP ADDRESS*	NOTES
J0	0	0xBC*n*04500	The J0 value received in (1,3,1) (row, column, STS) position of the last frame is written to this location.
Z0, STS #2	1	0xBC*(n+1)*04500	Z0 growth byte value received in last frame for position (1,3,2).
Z0, STS #3	2	0xBC*(n+2)*04500	Z0 growth byte value received in last frame for position (1,3,3).
Z0, STS #4	3	0xBC*(n+3)*04500	Z0 growth byte value received in last frame for position (1,3,4).
Z0, STS #5	0	0xBC*n*04501	Z0 growth byte value received in last frame for position (1,3,5).
Z0, STS #6	1	0xBC*(n+1)*04501	Z0 growth byte value received in last frame for position (1,3,6).
Z0, STS #7	2	0xBC*(n+2)*04501	Z0 growth byte value received in last frame for position (1,3,7).
Z0, STS #8	3	0xBC*(n+3)*04501	Z0 growth byte value received in last frame for position (1,3,8).
Z0, STS #9	0	0xBC*n*04502	Z0 growth byte value received in last frame for position (1,3,9).
Z0, STS #10	1	0xBC*(n+1)*04502	Z0 growth byte value received in last frame for position (1,3,10).
Z0, STS #11	2	0xBC*(n+2)*04502	Z0 growth byte value received in last frame for position (1,3,11).
Z0, STS #12	3	0xBC*(n+3)*04502	Z0 growth byte value received in last frame for position (1,3,12).

Table 244 Receive SONET/SDH OC-12 and OC-12c Transport Overhead Byte Addresses

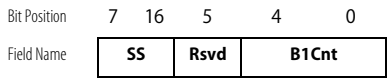
TRANSPORT OVERHEAD BYTE	CP# WITHIN A CLUSTER	C-5E NP ADDRESS*	NOTES
B1_SSH1	0	0xBC*n*04503	The actual B1 parity byte value is not written to the B1_SSH1 register. Instead, the number of bit lanes in error is provided. The number of errors reported is therefore 0 through 8 reported in the lower 5 bits of B1_SSH1. The upper 2 bits of this register indicate the value of the SS bits received in the H1 byte. 
MD0, STS #2	1	0xBC*(n+1)*04503	SDH Media Dependent byte value received in last frame for position (2,1,2).
MD0, STS #3	2	0xBC*(n+2)*04503	SDH Media Dependent byte value received in last frame for position (2,1,3).
MD0, STS #4	3	0xBC*(n+3)*04503	SDH Media Dependent byte value received in last frame for position (2,1,4).
MD0, STS #5	0	0xBC*n*04504	SDH Media Dependent byte value received in last frame for position (2,1,5).
MD0, STS #6	1	0xBC*(n+1)*04504	SDH Media Dependent byte value received in last frame for position (2,1,6).
MD0, STS #7	2	0xBC*(n+2)*04504	SDH Media Dependent byte value received in last frame for position (2,1,7).
MD0, STS #8	3	0xBC*(n+3)*04504	SDH Media Dependent byte value received in last frame for position (2,1,8).
MD0, STS #9	0	0xBC*n*04505	SDH Media Dependent byte value received in last frame for position (2,1,9).
MD0, STS #10	1	0xBC*(n+1)*04505	SDH Media Dependent byte value received in last frame for position (2,1,10).
MD0, STS #11	2	0xBC*(n+2)*04505	SDH Media Dependent byte value received in last frame for position (2,1,11).
MD0, STS #12	3	0xBC*(n+3)*04505	SDH Media Dependent byte value received in last frame for position (2,1,12).
E1	0	0xBC*n*04506	Orderwire E1 byte value received in last frame for position (2,2,1).

Table 244 Receive SONET/SDH OC-12 and OC-12c Transport Overhead Byte Addresses

TRANSPORT OVERHEAD BYTE	CP# WITHIN A CLUSTER	C-5E NP ADDRESS*	NOTES
MD1, STS #2	1	0xBC*(n+1)*04506	SDH Media Dependent byte value received in last frame for position (2,2,2).
MD1, STS #3	2	0xBC*(n+2)*04506	SDH Media Dependent byte value received in last frame for position (2,2,3).
MD1, STS #4	3	0xBC*(n+3)*04506	SDH Media Dependent byte value received in last frame for position (2,2,4).
MD1, STS #5	0	0xBC*n*04507	SDH Media Dependent byte value received in last frame for position (2,2,5).
MD1, STS #6	1	0xBC*(n+1)*04507	SDH Media Dependent byte value received in last frame for position (2,2,6).
MD1, STS #7	2	0xBC*(n+2)*04507	SDH Media Dependent byte value received in last frame for position (2,2,7).
MD1, STS #8	3	0xBC*(n+3)*04507	SDH Media Dependent byte value received in last frame for position (2,2,8).
F1	0	0xBC*n*04508	F1 byte value received in last frame for position (2,3,1).
CS0, STS #2	1	0xBC*(n+1)*04508	SDH Country Specific byte value received in last frame for position (2,3,2).
CS0, STS #3	2	0xBC*(n+2)*04508	SDH Country Specific byte value received in last frame for position (2,3,3).
CS0, STS #4	3	0xBC*(n+3)*04508	SDH Country Specific byte value received in last frame for position (2,3,4).
CS0, STS #5	0	0xBC*n*04509	SDH Country Specific byte value received in last frame for position (2,3,5).
CS0, STS #6	1	0xBC*(n+1)*04509	SDH Country Specific byte value received in last frame for position (2,3,6).
CS0, STS #7	2	0xBC*(n+2)*04509	SDH Country Specific byte value received in last frame for position (2,3,7).
CS0, STS #8	3	0xBC*(n+3)*04509	SDH Country Specific byte value received in last frame for position (2,3,8).
CS0, STS #9	0	0xBC*n*0450A	SDH Country Specific byte value received in last frame for position (2,3,9).
CS0, STS #10	1	0xBC*(n+1)*0450A	SDH Country Specific byte value received in last frame for position (2,3,10).

Table 244 Receive SONET/SDH OC-12 and OC-12c Transport Overhead Byte Addresses

TRANSPORT OVERHEAD BYTE	CP# WITHIN A CLUSTER	C-5E NP ADDRESS*	NOTES
CS0, STS #11	2	0xBC*(n+2)*0450A	SDH Country Specific byte value received in last frame for position (2,3,11).
CS0, STS #12	3	0xBC*(n+3)*0450A	SDH Country Specific byte value received in last frame for position (2,3,11).
D1	0	0xBC*n*0450B	Datacom Channel 1 byte value received in last frame for position (3,1,1).
MD2, STS #2	1	0xBC*(n+1)*0450B	SDH Media Dependent byte value received in last frame for position (3,1,2).
MD2, STS #3	2	0xBC*(n+2)*0450B	SDH Media Dependent byte value received in last frame for position (3,1,3).
MD2, STS #4	3	0xBC*(n+3)*0450B	SDH Media Dependent byte value received in last frame for position (3,1,4).
MD2, STS #5	0	0xBC*n*0450C	SDH Media Dependent byte value received in last frame for position (3,1,5).
MD2, STS #6	1	0xBC*(n+1)*0450C	SDH Media Dependent byte value received in last frame for position (3,1,6).
MD2, STS #7	2	0xBC*(n+2)*0450C	SDH Media Dependent byte value received in last frame for position (3,1,7).
MD2, STS #8	3	0xBC*(n+3)*0450C	SDH Media Dependent byte value received in last frame for position (3,1,8).
MD2, STS #9	0	0xBC*n*0450D	SDH Media Dependent byte value received in last frame for e position (3,1,9).
MD2, STS #10	1	0xBC*(n+1)*0450D	SDH Media Dependent byte value received in last frame for position (3,1,10).
MD2, STS #11	2	0xBC*(n+2)*0450D	SDH Media Dependent byte value received in last frame for position (3,1,11).
MD2, STS #12	3	0xBC*(n+3)*0450D	SDH Media Dependent byte value received in last frame for position (3,1,12).
D2	0	0xBC*n*0450E	Datacom Channel 2 byte value received in last frame for position (3,2,1).
MD3, STS #2	1	0xBC*(n+1)*0450E	SDH Media Dependent byte value received in last frame for position (3,2,2).
MD3, STS #3	2	0xBC*(n+2)*0450E	SDH Media Dependent byte value received in last frame for position (3,2,3).

Table 244 Receive SONET/SDH OC-12 and OC-12c Transport Overhead Byte Addresses

TRANSPORT OVERHEAD BYTE	CP# WITHIN A CLUSTER	C-5E NP ADDRESS*	NOTES
MD3, STS #4	3	0xBC*(n+3)*0450E	SDH Media Dependent byte value received in last frame for position (3,2,4).
MD3, STS #5	0	0xBC*n*0450F	SDH Media Dependent byte value received in last frame for position (3,2,5).
MD3, STS #6	1	0xBC*(n+1)*0450F	SDH Media Dependent byte value received in last frame for position (3,2,6).
MD3, STS #7	2	0xBC*(n+2)*0450F	SDH Media Dependent byte value received in last frame for position (3,2,7).
MD3, STS #8	3	0xBC*(n+3)*0450F	SDH Media Dependent byte value received in last frame for position (3,2,8).
D3	0	0xBC*n*04510	Datacom Channel 3 byte value received in last frame for position (3,3,1).

Table 244 Receive SONET/SDH OC-12 and OC-12c Transport Overhead Byte Addresses

TRANSPORT OVERHEAD BYTE	CP# WITHIN A CLUSTER	C-5E NP ADDRESS*	NOTES
H1, STS #1	0	0xBC*n*04511	For this H1 location, it is not the actual values of H1 (4,1,1) that is written, but the pointer processing results listed here: **BITS** / **DESCRIPTION** [7:6] Pointer State Where: 0=Valid Pointer 2=AIS-P is observed (pointer is all 1's) 3=Loss of Pointer (LOP-P) [5] New Data Flag occurred (NDF) [4] Pointer Increment occurred [3] Pointer Decrement occurred [2] Zero [1] Current Pointer Value bit9 [0] Current Pointer Value bit8 The SONET pointer interpreter meets the following GR-253 Issue 3 requirements: • R3-98 - NDF 3 of 4 bits correct determines NDF set - tolerant of single bit errors. • O3-96 - 8 of 10 voting for pointer increment decrement decision. • R3-104 - is met for pointer interpretation.
H1, STS #2	1	0xBC*(n+1)*04511	For OC12nc, this byte contains the pointer processing results for the second of 4 OC3c flows. See H1, STS #1 for decode.
H1, STS #3	2	0xBC*(n+2)*04511	For OC12nc, this byte contains the pointer processing results for the third of 4 OC3c flows. See H1, STS #1 for decode.
H1, STS #4	3	0xBC*(n+3)*04511	For OC12nc, this byte contains the pointer processing results for the last of 4 OC3c flows. See H1, STS #1 for decode.

Table 244 Receive SONET/SDH OC-12 and OC-12c Transport Overhead Byte Addresses

TRANSPORT OVERHEAD BYTE	CP# WITHIN A CLUSTER	C-5E NP ADDRESS*	NOTES
H2, STS #1	0	0xBC*n*04512	For this H2 location, it is not the actual values of H2 (4,1,2) that is written, but the value of the current pointer value bits 7:0.
H2, STS #2	1	0xBC*(n+1)*04512	For OC12nc, this byte contains the pointer processing results for the second of 4 OC3c flows. For this H2 location, it is not the actual values of H2 that is written, but the value of the current pointer value bits 7:0.
H2, STS #3	2	0xBC*(n+2)*04512	For OC12nc, this byte contains the pointer processing results for the third of 4 OC3c flows. For this H2 location, it is not the actual values of H2 that is written, but the value of the current pointer value bits 7:0.
H2, STS #4	3	0xBC*(n+3)*04512	For OC12nc, this byte contains the pointer processing results for the last of 4 OC3c flows. For this H2 location, it is not the actual values of H2 that is written, but the value of the current pointer value bits 7:0.
B2, STS #1	0	0xBC*n*04513	The actual B2 parity byte value in (5,1,1) is not written to the B2_CNT STS #1 register. Instead, the sum of the number of bit lanes in error for B1 STS #1 is provided. The number of errors reported is therefore 0 through 8.
B2, STS #2	1	0xBC*(n+1)*04513	The actual B2 parity byte value in (5,1,2) is not written to the B2_CNT STS #2 register. Instead, the sum of the number of bit lanes in error for B2 STS #2. This value ranges from 0-8.
B2, STS #3	2	0xBC*(n+2)*04513	The actual B2 parity byte value in (5,1,3) is not written to the B2_CNT STS #3 register. Instead, the sum of the number of bit lanes in error for B2 STS #3 is provided. This value ranges from 0-8.
B2, STS #4	3	0xBC*(n+3)*04513	The actual B2 parity byte value in (5,1,4) is not written to the B2_CNT STS #4 register. Instead, the sum of the number of bit lanes in error for B2 STS #4 is provided. This value ranges from 0-8.
B2, STS #5	0	0xBC*n*04514	The actual B2 parity byte value in (5,1,5) is not written to the B2_CNT STS #5 register. Instead, the sum of the number of bit lanes in error for B2 STS #5 is provided. This value ranges from 0-8.

Table 244 Receive SONET/SDH OC-12 and OC-12c Transport Overhead Byte Addresses

TRANSPORT OVERHEAD BYTE	CP# WITHIN A CLUSTER	C-5E NP ADDRESS*	NOTES
B2, STS #6	1	0xBC*(n+1)*04514	The actual B2 parity byte value in (5,1,6) is not written to the B2_CNT STS #6 register. Instead, the sum of the number of bit lanes in error for B2 STS #6 is provided. This value ranges from 0-8.
B2, STS #7	2	0xBC*(n+2)*04514	The actual B2 parity byte value in (5,1,7) is not written to the B2_CNT STS #7 register. Instead, the sum of the number of bit lanes in error for B2 STS #7 is provided. This value ranges from 0-8.
B2, STS #8	3	0xBC*(n+3)*04514	The actual B2 parity byte value in (5,1,8) is not written to the B2_CNT STS #8 register. Instead, the sum of the number of bit lanes in error for B2 STS #8 is provided. This value ranges from 0-8.
B2, STS #9	0	0xBC*n*04515	The actual B2 parity byte value in (5,1,9) is not written to the B2_CNT STS #9 register. Instead, the sum of the number of bit lanes in error for B2 STS #9 is provided. This value ranges from 0-8.
B2, STS #10	1	0xBC*(n+1)*04515	The actual B2 parity byte value in (5,1,10) is not written to the B2_CNT STS #10 register. Instead, the sum of the number of bit lanes in error for B2 STS #10 is provided. This value ranges from 0-8.
B2, STS #11	2	0xBC*(n+2)*04515	The actual B2 parity byte value in (5,1,11) is not written to the B2_CNT STS #11 register. Instead, the sum of the number of bit lanes in error for B2 STS #11 is provided. This value ranges from 0-8.
B2, STS #12	3	0xBC*(n+3)*04515	The actual B2 parity byte value in (5,1,12) is not written to the B2_CNT STS #12 register. Instead, the sum of the number of bit lanes in error for B2 STS #12 is provided. This value ranges from 0-8.
K1	0	0xBC*n*04516	The K1 byte (5,2,1) is written to the registers only when three identical bytes have been received in consecutive frames. For non-base CPs, STS positions (5,2,2), (5,2,3) and (5,2,4) are written.
K2	0	0xBC*n*04517	The K2 byte (5,3,1) is written to the registers only when three identical bytes have been received in consecutive frames. For non-base CPs, STS positions (5,3,2), (5,3,3) and (5,3,4) are written.
D4	0	0xBC*n*04518	Datacom Channel 4 byte value received in last frame for position (6,1,1).

Table 244 Receive SONET/SDH OC-12 and OC-12c Transport Overhead Byte Addresses

TRANSPORT OVERHEAD BYTE	CP# WITHIN A CLUSTER	C-5E NP ADDRESS*	NOTES
D5	0	0xBC*n*04519	Datacom Channel 5 byte value received in last frame for position (6,2,1).
D6	0	0xBC*n*0451A	Datacom Channel 6 byte value received in last frame for position (6,3,1).
D7	0	0xBC*n*0451B	Datacom Channel 7 byte value received in last frame for position (7,1,1).
D8	0	0xBC*n*0451C	Datacom Channel 8 byte value received in last frame for position (7,2,1).
D9	0	0xBC*n*0451D	Datacom Channel 9 byte value received in last frame for position (7,3,1).
D10	0	0xBC*n*0451E	Datacom Channel 10 byte value received in last frame for position (8,1,1).
D11	0	0xBC*n*0451F	Datacom Channel 11 byte value received in last frame for position (8,2,1).
D12	0	0xBC*n*04520	Datacom Channel 12 byte value received in last frame for position (8,3,1).
B2_SUM	0	0xBC*n*04521	The sum of all B2 errors received in the last frame is written to this register.
S1	0	0xBC*n*04522	S1 Synchronization Status byte value received in last frame for position (9,1,1).
Z1, STS #2	1	0xBC*(n+1)*04522	Z1 growth byte value received in last frame for position (9,1,2).
Z1, STS #3	2	0xBC*(n+2)*04522	Z1 growth byte value received in last frame for position (9,1,3).
Z1, STS #4	3	0xBC*(n+3)*04522	Z1 growth byte value received in last frame for position (9,1,4).
Z1, STS #5	0	0xBC*n*04523	Z1 growth byte value received in last frame for position (9,1,5).
Z1, STS #6	1	0xBC*(n+1)*04523	Z1 growth byte value received in last frame for position (9,1,6).
Z1, STS #7	2	0xBC*(n+2)*04523	Z1 growth byte value received in last frame for position (9,1,7).
Z1, STS #8	3	0xBC*(n+3)*04523	Z1 growth byte value received in last frame for position (9,1,8).

Table 244 Receive SONET/SDH OC-12 and OC-12c Transport Overhead Byte Addresses

TRANSPORT OVERHEAD BYTE	CP# WITHIN A CLUSTER	C-5E NP ADDRESS*	NOTES
Z1, STS #9	0	0xBC*n*04524	Z1 growth byte value received in last frame for position (9,1,9).
Z1, STS #10	1	0xBC*(n+1)*04524	Z1 growth byte value received in last frame for position (9,1,10).
Z1, STS #11	2	0xBC*(n+2)*04524	Z1 growth byte value received in last frame for position (9,1,11).
Z1, STS #12	3	0xBC*(n+3)*04524	Z1 growth byte value received in last frame for position (9,1,12).
Z2, STS #1	0	0xBC*n*04525	Z2 growth byte value received in last frame for position (9,2,1).
Z2, STS #2	1	0xBC*(n+1)*04525	Z2 growth byte value received in last frame for position (9,2,2).
M1	2	0xBC*(n+2)*04525	M1 REI-L byte value received in last frame for position (9,2,3).
Z2, STS #4	3	0xBC*(n+3)*04525	Z2 growth byte value received in last frame for position (9,2,4).
Z2, STS #5	0	0xBC*n*04526	Z2 growth byte value received in last frame for position (9,2,5).
Z2, STS #6	1	0xBC*(n+1)*04526	Z2 growth byte value received in last frame for position (9,2,6).
Z2, STS #7	2	0xBC*(n+2)*04526	Z2 growth byte value received in last frame for position (9,2,7).
Z2, STS #8	3	0xBC*(n+3)*04526	Z2 growth byte value received in last frame for position (9,2,8).
Z2, STS #9	0	0xBC*n*04527	Z2 growth byte value received in last frame for position (9,2,9).
Z2, STS #10	1	0xBC*(n+1)*04527	Z2 growth byte value received in last frame for position (9,2,10).
Z2, STS #11	2	0xBC*(n+2)*04527	Z2 growth byte value received in last frame for position (9,2,11).
Z2, STS #12	3	0xBC*(n+3)*04527	Z2 growth byte value received in last frame for position (9,2,12).
E2	0	0xBC*n*04528	Orderwire E2 byte value received in last frame for position (9,3,1).

Table 244 Receive SONET/SDH OC-12 and OC-12c Transport Overhead Byte Addresses

TRANSPORT OVERHEAD BYTE	CP# WITHIN A CLUSTER	C-5E NP ADDRESS*	NOTES
CS1, STS #2	1	0xBC*(n+1)*04528	SDH Country Specific 1 byte value received in last frame for position (9,3,2).
CS1, STS #3	2	0xBC*(n+2)*04528	SDH Country Specific 1 byte value received in last frame for position (9,3,3).
CS1, STS #4	3	0xBC*(n+3)*04528	SDH Country Specific 1 byte value received in last frame for position (9,3,4).
CS1, STS #5	0	0xBC*n*04529	SDH Country Specific 1 byte value received in last frame for position (9,3,5).
CS1, STS #6	1	0xBC*(n+1)*04529	SDH Country Specific 1 byte value received in last frame for position (9,3,6).
CS1, STS #7	2	0xBC*(n+2)*04529	SDH Country Specific 1 byte value received in last frame for position (9,3,7).
CS1, STS #8	3	0xBC*(n+3)*04529	SDH Country Specific 1 byte value received in last frame for position (9,3,8).
CS1, STS #9	0	0xBC*n*0452A	SDH Country Specific 1 byte value received in last frame for position (9,3,9).
CS1, STS #10	1	0xBC*(n+1)*0452A	SDH Country Specific 1 byte value received in last frame for position (9,3,10).
CS1, STS #11	2	0xBC*(n+2)*0452A	SDH Country Specific 1 byte value received in last frame for position (9,3,11).
CS1, STS #12	3	0xBC*(n+3)*0452A	SDH Country Specific 1 byte value received in last frame for position (9,3,12).

* *n* can be CP0, CP4, CP8, or CP12

Receive OC-12/OC-12c Path Overhead Definitions

Table 245 on page 774 lists the SONET/SDH Path Overhead definitions and addresses.

Table 245 Receive SONET/SDH OC-12 and OC-12c Path Overhead Byte Addresses

PATH OVERHEAD BYTE	CP# WITHIN A CLUSTER	C-5E NP ADDRESS*	NOTES
J1, STS #1	0	0xBC*n*0452C	For OC12c and OC12nc: The J1 register contains the Nth J1 of the 64 (or 16) byte path trace message. The value N is obtained by the hardware from the RxSonet J1 Index field of the SDP_Mode2 register and is under CPRC software control. To read the complete message for the first time, increment the index N after every frame.
J1, STS #2	1	0xBC*(n+1)*0452C	For OC12c: One of the fixed stuff bytes. For OC12nc: J1 for the second OC3c of 4.
J1, STS #3	2	0xBC*(n+2)*0452C	For OC12c: One of the fixed stuff bytes. For OC12nc: J1 for the third OC3c of 4.
J1, STS #4	3	0xBC*(n+3)*0452C	For OC12c: One of the fixed stuff bytes. For OC12nc: J1 for the last OC3c of 4.
B3_CNT, STS #1	0	0xBC*n*0452D	For OC12c and OC12nc: The actual B3 parity byte value is not written to the B3_CNT STS #1 register. Instead, the sum of the number of bit lanes in error for B3 STS #1 is provided. The number of errors reported is therefore 0 through 8.
B3_CNT, STS #2	1	0xBC*(n+1)*0452D	For OC12c: One of the fixed stuff bytes. For OC12nc: B3 for the second OC3c of 4.
B3_CNT, STS #3	2	0xBC*(n+2)*0452D	For OC12c: One of the fixed stuff bytes. For OC12nc: B3 for the third OC3c of 4.
B3_CNT, STS #4	3	0xBC*(n+3)*0452D	For OC12c: One of the fixed stuff bytes. For OC12nc: B3 for the last OC3c of 4.
C2, STS #1	0	0xBC*n*0452E	For OC12c: The C2 value. For OC12nc: C2 for the first OC3c of 4.
C2, STS #2	1	0xBC*(n+1)*0452E	For OC12c: The C2 value. For OC12nc: C2 for the second OC3c of 4.

Table 245 Receive SONET/SDH OC-12 and OC-12c Path Overhead Byte Addresses (continued)

PATH OVERHEAD BYTE	CP# WITHIN A CLUSTER	C-5E NP ADDRESS*	NOTES
C2, STS #3	2	0xBC*(n+2)*0452E	For OC12c: The C2 value. For OC12nc: C2 for the third OC3c of 4.
C2, STS #4	3	0xBC*(n+3)*0452E	For OC12c: The C2 value. For OC12nc: C2 for the last OC3c of 4.
G1, STS #1	0	0xBC*n*0452F	For OC12c: The G1 value. For OC12nc: G1 for the first OC3c of 4.
G1, STS #2	1	0xBC*(n+1)*0452F	For OC12c: The G1 value. For OC12nc: G1 for the second OC3c of 4.
G1, STS #3	2	0xBC*(n+2)*0452F	For OC12c: The G1 value. For OC12nc: G1 for the third OC3c of 4.
G1, STS #4	3	0xBC*(n+3)*0452F	For OC12c: The G1 value. For OC12nc: G1 for the last OC3c of 4.
F2, STS #1	0	0xBC*n*04530	For OC12c: The F2 value. For OC12nc: F2 for the first OC3c of 4.
F2, STS #2	1	0xBC*(n+1)*04530	For OC12c: The F2 value. For OC12nc: F2 for the second OC3c of 4.
F2, STS #3	2	0xBC*(n+2)*04530	For OC12c: The F2 value. For OC12nc: F2 for the third OC3c of 4.
F2, STS #4	3	0xBC*(n+3)*04530	For OC12c: The F2 value. For OC12nc: F2 for the last OC3c of 4.
H4, STS #1	0	0xBC*n*04531	For OC12c: The H4 value. For OC12nc: H4 for the first OC3c of 4.
H4, STS #2	1	0xBC*(n+1)*04531	For OC12c: The H4 value. For OC12nc: H4 for the second OC3c of 4.
H4, STS #3	2	0xBC*(n+2)*04531	For OC12c: The H4 value. For OC12nc: H4 for the third OC3c of 4.
H4, STS #4	3	0xBC*(n+3)*04531	For OC12c: The H4 value. For OC12nc: H4 for the last OC3c of 4.
Z3, STS #1	0	0xBC*n*04532	For OC12c: The Z3 value. For OC12nc: Z3 for the first OC3c of 4.

Table 245 Receive SONET/SDH OC-12 and OC-12c Path Overhead Byte Addresses (continued)

PATH OVERHEAD BYTE	CP# WITHIN A CLUSTER	C-5E NP ADDRESS*	NOTES
Z3, STS #2	1	0xBC*(n+1)*04532	For OC12c: The Z3 value. For OC12nc: Z3 for the second OC3c of 4.
Z3, STS #3	2	0xBC*(n+2)*04532	For OC12c: The Z3 value. For OC12nc: Z3 for the third OC3c of 4.
Z3, STS #4	3	0xBC*(n+3)*04532	For OC12c: The Z3 value. For OC12nc: Z3 for the last OC3c of 4.
Z4, STS #1	0	0xBC*n*04533	For OC12c: The Z4 value. For OC12nc: Z4 for the first OC3c of 4.
Z4, STS #2	1	0xBC*(n+1)*04533	For OC12c: The Z4 value. For OC12nc: Z4 for the second OC3c of 4.
Z4, STS #3	2	0xBC*(n+2)*04533	For OC12c: The Z4 value. For OC12nc: Z4 for the third OC3c of 4.
Z4, STS #4	3	0xBC*(n+3)*04533	For OC12c: The Z4 value. For OC12nc: Z4 for the last OC3c of 4.
Z5, STS #1	0	0xBC*n*04534	For OC12c: The Z5 value. For OC12nc: Z5 for the first OC3c of 4.
Z5, STS #2	1	0xBC*(n+1)*04534	For OC12c: The Z5 value. For OC12nc: Z5 for the second OC3c of 4.
Z5, STS #3	2	0xBC*(n+2)*04534	For OC12c: The Z5 value. For OC12nc: Z5 for the third OC3c of 4.
Z5, STS #4	3	0xBC*(n+3)*04534	For OC12c: The Z5 value. For OC12nc: Z5 for the last OC3c of 4.

* *n* can be CP0, CP4, CP8, or CP12

Receive OC-12/OC-12c Statistics Counters for Both Transport and Path Overhead

Table 246 on page 777 lists the SONET/SDH Statistics Counters for both Transport and Path Overhead. Their definitions and addresses are provided. Statistics Counters collect both SONET/SDH Transport and Path Overhead on a per frame basis and can be read on a one second basis.

When using the frame count mechanism to interrupt errors every n frame (frame mode=1), the internal accumulators written to the statistics counters are automatically cleared. This allows the number of errors in the last n frames to be read. In this case, there is no need to manually clear the statistics counters with the frame mode bit.

Multi-byte counters are aligned so as to be read in one 16bit or 32bit read operation.

Table 246 Receive SONET/SDH OC-12 and OC-12c Statistics Counters Byte Addresses

TRANSPORT AND PATH OVERHEAD BYTES	CP# WITH A CLUSTER	C-5E NP ADDRESS	NOTES
B1_ACCUM[15:8]	0	0xBCn04536	Bits [15:8] of B1_ACCUM, the Accumulated B1 Bit Interleaved Parity Error registers. The accumulated B1 registers contain a running total of the number of B1 errors. If a rising edge is observed on the FrameMode signal, this count is cleared to zero and begins incrementing again as errors are observed. Note: These registers are sized to allow up to 1 seconds worth of errors to be accumulated before wrapping.
B1_ACCUM[7:0]	0	0xBCn04537	Bits [7:0] of B1_ACCUM, the Accumulated B1 Bit Interleaved Parity Error registers.
B2_ACCUM[19:16]	2	0xBC(*n+2*)04538	Bits [19:16] of B2_ACCUM, the Accumulated B2 Bit Interleaved Parity Error registers. The accumulated B2 registers contain a running total of the number of B2 errors. If a rising edge is observed on the FrameMode signal, this count is cleared to zero and begins incrementing again as errors are observed. Note that these registers are sized to allow up to 1 seconds worth of errors to be accumulated before wrapping.
B2_ACCUM[15:8]	2	0xBC(*n+2*)04539	Bits [15:8] of B2_ACCUM, the Accumulated B2 Bit Interleaved Parity Error registers.

Table 246 Receive SONET/SDH OC-12 and OC-12c Statistics Counters Byte Addresses (continued)

TRANSPORT AND PATH OVERHEAD BYTES	CP# WITH A CLUSTER	C-5E NP ADDRESS	NOTES
B2_ACCUM[7:0]	2	0xBC(*n+2*)0453A	Bits [7:0] of B2_ACCUM, the Accumulated B2 Bit Interleaved Parity Error registers.
B3_ACCUM[15:8], STS#1	0	0xBCn0453C	Bits [15:8] of B3_ACCUM, the Accumulated B3 Bit Interleaved Parity Error registers. The accumulated B3 registers contain a running total of the number of B3 errors. If a rising edge is observed on the FrameMode signal, this count is cleared to zero and begins incrementing again as errors are observed. Note: These registers are sized to allow up to 1 seconds worth of errors to be accumulated before wrapping.
B3_ACCUM[7:0], STS#1	0	0xBC(*n*)0453D	Bits [7:0] of B3_ACCUM, the Accumulated B3 Bit Interleaved Parity Error registers.
B3_ACCUM[15:8], STS#2	1	0xBC(*n+1*)0453C	For OC12c: Unused. For OC12nc: B3 for second OC3 of 4.
B3_ACCUM[7:0], STS#2	1	0xBC(*n+1*)0453D	For OC12c: Unused. For OC12nc: B3 for second OC3 of 4.
B3_ACCUM[15:8], STS#3	2	0xBC(*n+2*)0453C	For OC12c: Unused. For OC12nc: B3 for third OC3 of 4.
B3_ACCUM[7:0], STS#3	2	0xBC(*n+2*)0453D	For OC12c: Unused. For OC12nc: B3 for third OC3 of 4.
B3_ACCUM[15:8], STS#4	3	0xBC(*n+3*)0453C	For OC12c: Unused. For OC12nc: B3 for last OC3 of 4.
B3_ACCUM[7:0], STS#4	3	0xBC(*n+3*)0453D	For OC12c: Unused. For OC12nc: B3 for last OC3 of 4.

Table 246 Receive SONET/SDH OC-12 and OC-12c Statistics Counters Byte Addresses (continued)

TRANSPORT AND PATH OVERHEAD BYTES	CP# WITH A CLUSTER	C-5E NP ADDRESS	NOTES
REI_P_ACCUM[15:8], STS#1	0	0xBCn0453E	Bits [15:8] of REI_P_ACCUM, the Accumulated Path Remote Error Indication registers. The accumulated REI_P registers contain a running total of the number of REI_P errors seen by the far end. If a rising edge is observed on the FrameMode signal, this count is cleared to zero and begins incrementing again as errors are observed. Note: These registers are sized to allow up to 1 seconds worth of errors to be accumulated before wrapping.
REI_P_ACCUM[7:0], STS#1	0	0xBCn0453F	Bits [7:0] of REI_P_ACCUM, the Accumulated Path Remote Error Indication registers.
REI_P_ACCUM[15:8], STS#2	1	0xBC(*n+1)*0453E	For OC12c: Unused. For OC12nc: B3 for second OC3 of 4.
REI_P_ACCUM[7:0], STS#2	1	0xBC(*n+1)*0453F	For OC12c: Unused. For OC12nc: B3 for second OC3 of 4.
REI_P_ACCUM[15:8], STS#3	2	0xBC(*n+2)*0453E	For OC12c: Unused. For OC12nc: B3 for third OC3 of 4.
REI_P_ACCUM[7:0], STS#3	2	0xBC(*n+2)*0453F	For OC12c: Unused. For OC12nc: B3 for third OC3 of 4.
REI_P_ACCUM[15:8], STS#4	3	0xBC(*n+3)*0453E	For OC12c: Unused. For OC12nc: B3 for last OC3 of 4.
REI_P_ACCUM[7:0], STS#4	3	0xBC(*n+3)*0453F	For OC12c: Unused. For OC12nc: B3 for last OC3 of 4.

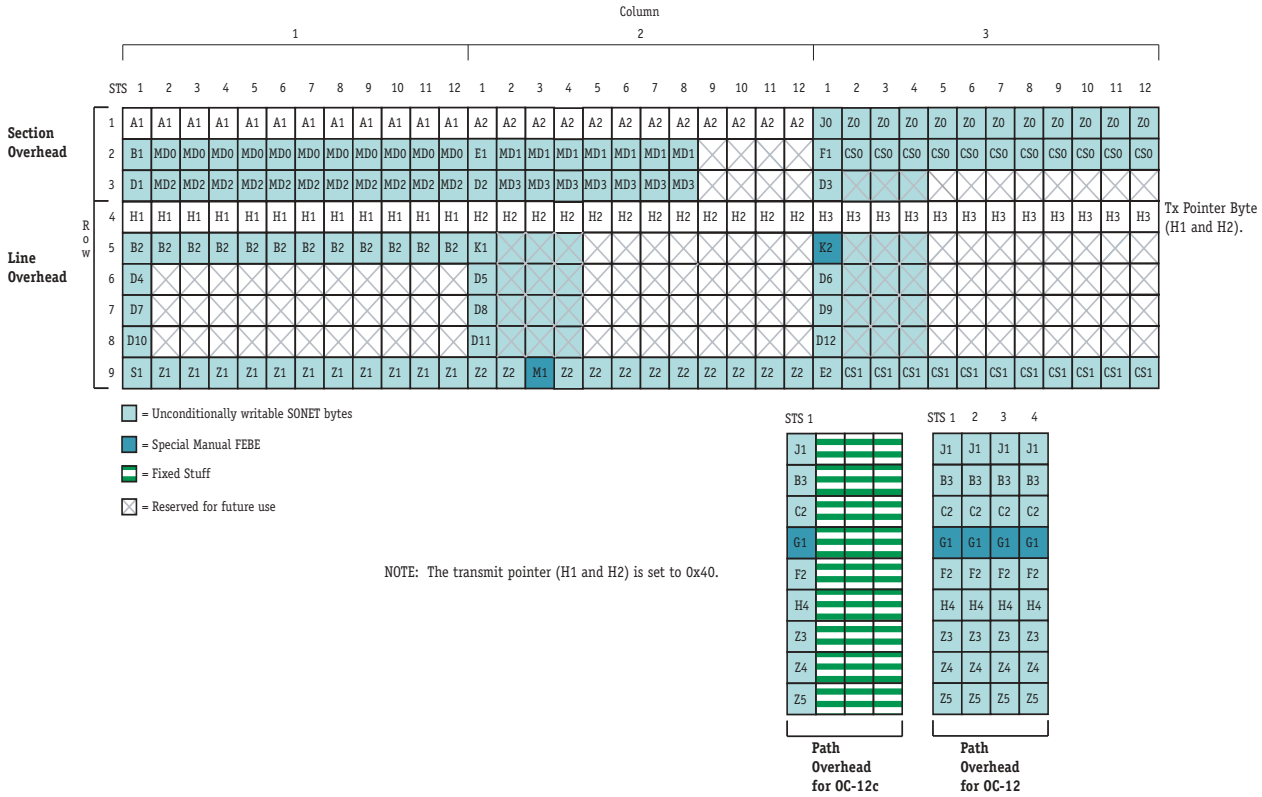
Table 246 Receive SONET/SDH OC-12 and OC-12c Statistics Counters Byte Addresses (continued)

TRANSPORT AND PATH OVERHEAD BYTES	CP# WITH A CLUSTER	C-5E NP ADDRESS	NOTES
REI_L_ACCUM[19:16]	2	0xBC(*n+2)*04541	Bits [19:16] of REI_L_ACCUM, the Accumulated Line Remote Error Indication registers. The accumulated REI_L registers contain a running total of the number of REI_L errors seen by the far end. If a rising edge is observed on the FrameMode signal, this count is cleared to zero and begins incrementing again as errors are observed. This count can be read as a single 32bit word at address 0xBCn04540. Note: These registers are sized to allow up to 1 seconds worth of errors to be accumulated before wrapping.
REI_L_ACCUM[15:8]	2	0xBC(*n+2)*04542	Bits [15:8] of REI_L_ACCUM, the Accumulated Line Remote Error Indication registers.
REI_L_ACCUM[7:0]	2	0xBC(*n+2)*04543	Bits [7:0] of REI_L_ACCUM, the Accumulated Line Remote Error Indication registers.

Transmit OC-12/OC-12c Writable Overhead Bytes Positions

Figure 126 on page 781 shows the writable bytes in the OC-12/OC-12c SONET/SDH Overhead.

Figure 126 Tx SONET/SDH OC-12/OC-12c Writable Overhead Bytes



Transmit OC-12/OC-12c Transport Overhead Definitions

Table 247 on page 782 lists the SONET/SDH Transport Overhead definitions and addresses.

Table 247 Transmit SONET/SDH OC-12 and OC-12c Transport Overhead Byte Addresses

TRANSPORT OVERHEAD BYTE	CP# WITHIN A CLUSTER	C-5E NP ADDRESS*	NOTES
Z0, STS #2	1	0xBC*(n+1)*04580	Z0 growth byte value to be sent in the next frame for position (1,3,2).
Z0, STS #3	2	0xBC*(n+2)*04580	Z0 growth byte value to be sent in the next frame for position (1,3,3).
Z0, STS #4	3	0xBC*(n+3)*04580	Z0 growth byte value to be sent in the next frame for position (1,3,4).
Z0, STS #5	0	0xBC*n*04581	Z0 growth byte value to be sent in the next frame for position (1,3,5).
Z0, STS #6	1	0xBC*(n+1)*04581	Z0 growth byte value to be sent in the next frame for position (1,3,6).
Z0, STS #7	2	0xBC*(n+2)*04581	Z0 growth byte value to be sent in the next frame for position (1,3,7).
Z0, STS #8	3	0xBC*(n+3)*04581	Z0 growth byte value to be sent in the next frame for position (1,3,8).
Z0, STS #9	0	0xBC*n*04582	Z0 growth byte value to be sent in the next frame for position (1,3,9).
Z0, STS #10	1	0xBC*(n+1)*04582	Z0 growth byte value to be sent in the next frame for position (1,3,10).
Z0, STS #11	2	0xBC*(n+2)*04582	Z0 growth byte value to be sent in the next frame for position (1,3,11).
Z0, STS #12	3	0xBC*(n+3)*04582	Z0 growth byte value to be sent in the next frame for position (1,3,12).
MD0, STS #2	1	0xBC*(n+1)*04583	SDH Media Dependent byte value to be sent in the next frame for position (2,1,2).
MD0, STS #3	2	0xBC*(n+2)*04583	SDH Media Dependent byte value to be sent in the next frame for position (2,1,3).
MD0, STS #4	3	0xBC*(n+3)*04583	SDH Media Dependent byte value to be sent in the next frame for position (2,1,4).
MD0, STS #5	0	0xBC*n*04584	SDH Media Dependent byte value to be sent in the next frame for position (2,1,5).

Table 247 Transmit SONET/SDH OC-12 and OC-12c Transport Overhead Byte Addresses

TRANSPORT OVERHEAD BYTE	CP# WITHIN A CLUSTER	C-5E NP ADDRESS*	NOTES
MD0, STS #6	1	0xBC*(n+1)*04584	SDH Media Dependent byte value to be sent in the next frame for position (2,1,6).
MD0, STS #7	2	0xBC*(n+2)*04584	SDH Media Dependent byte value to be sent in the next frame for position (2,1,7).
MD0, STS #8	3	0xBC*(n+3)*04584	SDH Media Dependent byte value to be sent in the next frame for position (2,1,8).
MD0, STS #9	0	0xBC*n*04585	SDH Media Dependent byte value to be sent in the next frame for position (2,1,9).
MD0, STS #10	1	0xBC*(n+1)*04585	SDH Media Dependent byte value to be sent in the next frame for position (2,1,10).
MD0, STS #11	2	0xBC*(n+2)*04585	SDH Media Dependent byte value to be sent in the next frame for position (2,1,11).
MD0, STS #12	3	0xBC*(n+3)*04585	SDH Media Dependent byte value to be sent in the next frame for position (2,1,12).
E1	0	0xBC*n*04586	Orderwire E1 byte value to be sent in the next frame for position (2,2,1).
MD1, STS #2	1	0xBC*(n+1)*04586	SDH Media Dependent byte value to be sent in the next frame for position (2,2,2).
MD1, STS #3	2	0xBC*(n+2)*04586	SDH Media Dependent byte value to be sent in the next frame for position (2,2,3).
MD1, STS #4	3	0xBC*(n+3)*04586	SDH Media Dependent byte value to be sent in the next frame for position (2,2,4).
MD1, STS #5	0	0xBC*n*04587	SDH Media Dependent byte value to be sent in the next frame for position (2,2,5).
MD1, STS #6	1	0xBC*(n+1)*04587	SDH Media Dependent byte value to be sent in the next frame for position (2,2,6).
MD1, STS #7	2	0xBC*(n+2)*04587	SDH Media Dependent byte value to be sent in the next frame for position (2,2,7).
MD1, STS #8	3	0xBC*(n+3)*04587	SDH Media Dependent byte value to be sent in the next frame for position (2,2,8).
F1	0	0xBC*n*04588	F1 byte value to be sent in the next frame for position (2,3,1).
CS0, STS #2	1	0xBC*(n+1)*04588	SDH Country Specific byte value to be sent in the next frame for position (2,3,2).

Table 247 Transmit SONET/SDH OC-12 and OC-12c Transport Overhead Byte Addresses

TRANSPORT OVERHEAD BYTE	CP# WITHIN A CLUSTER	C-5E NP ADDRESS*	NOTES
CS0, STS #3	2	0xBC*(n+2)*04588	SDH Country Specific byte value to be sent in the next frame for position (2,3,3).
CS0, STS #4	3	0xBC*(n+3)*04588	SDH Country Specific byte value to be sent in the next frame for position (2,3,4).
CS0, STS #5	0	0xBC*n*04589	SDH Country Specific byte value to be sent in the next frame for position (2,3,5).
CS0, STS #6	1	0xBC*(n+1)*04589	SDH Country Specific byte value to be sent in the next frame for position (2,3,6).
CS0, STS #7	2	0xBC*(n+2)*04589	SDH Country Specific byte value to be sent in the next frame for position (2,3,7).
CS0, STS #8	3	0xBC*(n+3)*04589	SDH Country Specific byte value to be sent in the next frame for position (2,3,8).
CS0, STS #9	0	0xBC*n*0458A	SDH Country Specific byte value to be sent in the next frame for position (2,3,9).
CS0, STS #10	1	0xBC*(n+1)*0458A	SDH Country Specific byte value to be sent in the next frame for position (2,3,10).
CS0, STS #11	2	0xBC*(n+2)*0458A	SDH Country Specific byte value to be sent in the next frame for position (2,3,11).
CS0, STS #12	3	0xBC*(n+3)*0458A	SDH Country Specific byte value to be sent in the next frame for position (2,3,12).
D1	0	0xBC*n*0458B	Datacom Channel 1 byte value to be sent in the next frame for position (3,1,1).
MD2, STS #2	1	0xBC*(n+1)*0458B	SDH Media Dependent byte value to be sent in the next frame for position (3,1,2).
MD2, STS #3	2	0xBC*(n+2)*0458B	SDH Media Dependent byte value to be sent in the next frame for position (3,1,3).
MD2, STS #4	3	0xBC*(n+3)*0458B	SDH Media Dependent byte value to be sent in the next frame for position (3,1,4).
MD2, STS #5	0	0xBC*n*0458C	SDH Media Dependent byte value to be sent in the next frame for position (3,1,5).
MD2, STS #6	1	0xBC*(n+1)*0458C	SDH Media Dependent byte value to be sent in the next frame for position (3,1,6).
MD2, STS #7	2	0xBC*(n+2)*0458C	SDH Media Dependent byte value to be sent in the next frame for position (3,1,7).

Table 247 Transmit SONET/SDH OC-12 and OC-12c Transport Overhead Byte Addresses

TRANSPORT OVERHEAD BYTE	CP# WITHIN A CLUSTER	C-5E NP ADDRESS*	NOTES
MD2, STS #8	3	0xBC*(n+3)*0458C	SDH Media Dependent byte value to be sent in the next frame for position (3,1,8).
MD2, STS #9	0	0xBC*n*0458D	SDH Media Dependent byte value to be sent in the next frame for position (3,1,9).
MD2, STS #10	1	0xBC*(n+1)*0458D	SDH Media Dependent byte value to be sent in the next frame for position (3,1,10).
MD2, STS #11	2	0xBC*(n+2)*0458D	SDH Media Dependent byte value to be sent in the next frame for position (3,1,11).
MD2, STS #12	3	0xBC*(n+3)*0458D	SDH Media Dependent byte value to be sent in the next frame for position (3,1,12).
D2	0	0xBC*n*0458E	Datacom Channel 2 byte value to be sent in the next frame for position (3,2,1).
MD3, STS #2	1	0xBC*(n+1)*0458E	SDH Media Dependent byte value to be sent in the next frame for position (3,2,2).
MD3, STS #3	2	0xBC*(n+2)*0458E	SDH Media Dependent byte value to be sent in the next frame for position (3,2,3).
MD3, STS #4	3	0xBC*(n+3)*0458E	SDH Media Dependent byte value to be sent in the next frame for position (3,2,4).
MD3, STS #5	0	0xBC*n*0458F	SDH Media Dependent byte value to be sent in the next frame for position (3,2,5).
MD3, STS #6	1	0xBC*(n+1)*0458F	SDH Media Dependent byte value to be sent in the next frame for position (3,2,6).
MD3, STS #7	2	0xBC*(n+2)*0458F	SDH Media Dependent byte value to be sent in the next frame for position (3,2,7).
MD3, STS #8	3	0xBC*(n+3)*0458F	SDH Media Dependent byte value to be sent in the next frame for position (3,2,8).
D3	0	0xBC*n*04590	Datacom Channel 3 byte value to be sent in the next frame for position (3,3,1).
K1	0	0xBC*n*04591	K1 byte position (5,2,1) read from this register is transmitted in the next frame.
K2	0	0xBC*n*04592	K2 byte position (5,3,1) read from this register is transmitted in the next frame. See *Manual_FEBE* description in *SDP_Mode3* register for exceptions.

Table 247 Transmit SONET/SDH OC-12 and OC-12c Transport Overhead Byte Addresses

TRANSPORT OVERHEAD BYTE	CP# WITHIN A CLUSTER	C-5E NP ADDRESS*	NOTES
D4	0	0xBC*n*04593	Datacom Channel 4 byte value to be sent in the next frame for position (6,1,1).
D5	0	0xBC*n*04594	Datacom Channel 5 byte value to be sent in the next frame for position (6,2,1).
D6	0	0xBC*n*04595	Datacom Channel 6 byte value to be sent in the next frame for position (6,3,1).
D7	0	0xBC*n*04596	Datacom Channel 7 byte value to be sent in the next frame for position (7,1,1).
D8	0	0xBC*n*04597	Datacom Channel 8 byte value to be sent in the next frame for position (7,2,1).
D9	0	0xBC*n*04598	Datacom Channel 9 byte value to be sent in the next frame for position (7,3,1).
D10	0	0xBC*n*04599	Datacom Channel 10 byte value to be sent in the next frame for position (8,1,1).
D11	0	0xBC*n*0459A	Datacom Channel 11 byte value to be sent in the next frame for position (8,2,1).
D12	0	0xBC*n*0459B	Datacom Channel 12 byte value to be sent in the next frame for position (8,3,1).
S1	0	0xBC*n*0459C	S1 Synchronization Status byte value to be sent in the next frame for position (9,1,1).
Z1, STS #2	1	0xBC*(n+1)*0459C	Z1 growth byte value to be sent in the next frame for position (9,1,2).
Z1, STS #3	2	0xBC*(n+2)*0459C	Z1 growth byte value to be sent in the next frame for position (9,1,3).
Z1, STS #4	3	0xBC*(n+3)*0459C	Z1 growth byte value to be sent in the next frame for position (9,1,4).
Z1, STS #5	0	0xBC*n*0459D	Z1 growth byte value to be sent in the next frame for position (9,1,5).
Z1, STS #6	1	0xBC*(n+1)*0459D	Z1 growth byte value to be sent in the next frame for position (9,1,6).
Z1, STS #7	2	0xBC*(n+2)*0459D	Z1 growth byte value to be sent in the next frame for position (9,1,7).
Z1, STS #8	3	0xBC*(n+3)*0459D	Z1 growth byte value to be sent in the next frame for position (9,1,8).

Table 247 Transmit SONET/SDH OC-12 and OC-12c Transport Overhead Byte Addresses

TRANSPORT OVERHEAD BYTE	CP# WITHIN A CLUSTER	C-5E NP ADDRESS*	NOTES
Z1, STS #9	0	0xBC*n*0459E	Z1 growth byte value to be sent in the next frame for e position (9,1,9).
Z1, STS #10	1	0xBC*(n+1)*0459E	Z1 growth byte value to be sent in the next frame for position (9,1,10).
Z1, STS #11	2	0xBC*(n+2)*0459E	Z1 growth byte value to be sent in the next frame for position (9,1,11).
Z1, STS #12	3	0xBC*(n+3)*0459E	Z1 growth byte value to be sent in the next frame for position (9,1,12).
Z2, STS #1	0	0xBC*n*0459F	Z2 growth byte value to be sent in the next frame for position (9,2,1).
Z2, STS #2	1	0xBC*(n+1)*0459F	Z2 growth byte value to be sent in the next frame for position (9,2,2).
M1	2	0xBC*(n+2)*0459F	M1 REI-L byte value to be sent in the next frame for position (9,2,3). See *Manual_FEBE* description in *SDP_Mode3* register for exceptions.
Z2, STS #4	3	0xBC*(n+3)*0459F	Z2 growth byte value to be sent in the next frame for position (9,2,4).
Z2, STS #5	0	0xBC*n*045A0	Z2 growth byte value to be sent in the next frame for position (9,2,5).
Z2, STS #6	1	0xBC*(n+1)*045A0	Z2 growth byte value to be sent in the next frame for position (9,2,6).
Z2, STS #7	2	0xBC*(n+2)*045A0	Z2 growth byte value to be sent in the next frame for position (9,2,7).
Z2, STS #8	3	0xBC*(n+3)*045A0	Z2 growth byte value to be sent in the next frame for position (9,2,8).
Z2, STS #9	0	0xBC*n*045A1	Z2 growth byte value to be sent in the next frame for position (9,2,9).
Z2, STS #10	1	0xBC*(n+1)*045A1	Z2 growth byte value to be sent in the next frame for position (9,2,10).
Z2, STS #11	2	0xBC*(n+2)*045A1	Z2 growth byte value to be sent in the next frame for position (9,2,11).
Z2, STS #12	3	0xBC*(n+3)*045A1	Z2 growth byte value to be sent in the next frame for position (9,2,12).

Table 247 Transmit SONET/SDH OC-12 and OC-12c Transport Overhead Byte Addresses

TRANSPORT OVERHEAD BYTE	CP# WITHIN A CLUSTER	C-5E NP ADDRESS*	NOTES
E2	0	0xBC*n*045A2	Orderwire E2 byte value to be sent in the next frame for position (9,3,1).
CSE2, STS #2	1	0xBC*(n+1)*045A2	SDH Country Specific 1 byte value to be sent in the next frame for position (9,3,2).
CSE2, STS #3	2	0xBC*(n+2)*045A2	SDH Country Specific 1 byte value to be sent in the next frame for position (9,3,3).
CSE2, STS #4	3	0xBC*(n+3)*045A2	SDH Country Specific 1 byte value to be sent in the next frame for position (9,3,4).
CS2, STS #5	0	0xBC*n*045A3	SDH Country Specific 1 byte value to be sent in the next frame for position (9,3,5).
CS2, STS #6	1	0xBC*(n+1)*045A3	SDH Country Specific 1 byte value to be sent in the next frame for position (9,3,6).
CS2, STS #7	2	0xBC*(n+2)*045A3	SDH Country Specific 1 byte value to be sent in the next frame for position (9,3,7).
CS2, STS #8	3	0xBC*(n+3)*045A3	SDH Country Specific 1 byte value to be sent in the next frame for position (9,3,8).
CS3, STS #9	0	0xBC*n*045A4	SDH Country Specific 1 byte value to be sent in the next frame for position (9,3,9).
CS3, STS #10	1	0xBC*(n+1)*045A4	SDH Country Specific 1 byte value to be sent in the next frame for position (9,3,10).
CS3, STS #11	2	0xBC*(n+2)*045A4	SDH Country Specific 1 byte value to be sent in the next frame for position (9,3,11).
J0_BUF	0	0xBC*n*045F0 to FF	The J0 section trace message to be transmitted is written to these registers.
H1_SS, STS #1	0	0xBC*n*045EC	For SDH, the transmit SS bits *must* be initialized for backward compatibility. This register allows the SS bits of the pointer to be initialized. The two LSB of the register are written to the H1 SS bits of the Transmit SONET/SDH frame.
H1_SS, STS #2	1	0xBC(n+1)045EC	For OC12c: The H1 value. For OC12nc: H1 for second OC3 of 4.
H1_SS, STS #3	2	0xBC(n+2)045EC	For OC12c: The H1 value. For OC12nc: H1 for third OC3 of 4.

Table 247 Transmit SONET/SDH OC-12 and OC-12c Transport Overhead Byte Addresses

TRANSPORT OVERHEAD BYTE	CP# WITHIN A CLUSTER	C-5E NP ADDRESS*	NOTES
H1_SS, STS #4	3	0xBC(n+3)045EC	For OC12c: The H1 value. For OC12nc: H1 for last OC3 of 4.

* *n* can be CP0, CP4, CP8, or CP12

Transmit OC-12/OC-12c Path Overhead Definitions

Table 248 on page 790 lists the SONET/SDH Path Overhead definitions and addresses.

Table 248 Transmit SONET/SDH OC-12 and OC-12c Path Overhead Byte Addresses

PATH OVERHEAD BYTE	CP# WITHIN A CLUSTER	C-5E NP ADDRESS*	NOTES
C2, STS #1	0	0xBC*n*045A5	For OC12c: The C2 value to be transmitted in the next frame. For OC12nc: C2 for first OC3c of 4.
C2, STS #2	1	0xBC*(n+1)*045A5	For OC12c: The C2 value. For OC12nc: C2 for second OC3c of 4.
C2, STS #3	2	0xBC*(n+2)*045A5	For OC12c: The C2 value. For OC12nc: C2 for third OC3c of 4.
C2, STS #4	3	0xBC*(n+3)*045A5	For OC12c: The C2 value. For OC12nc: C2 for last OC3c of 4.
G1, STS #1	0	0xBC*n*045A6	For OC12c: The G1 value to be transmitted in the next frame. See *Manual_FEBE* description in *SDP_Mode3* register for G1 transmit exceptions. For OC12nc: G1 for first OC3c of 4.
G1, STS #2	1	0xBC*(n+1)*045A6	For OC12c: The G1 value. For OC12nc: G1 for second OC3c of 4.
G1, STS #3	2	0xBC*(n+2)*045A6	For OC12c: The G1 value. For OC12nc: G1 for third OC3c of 4.
G1, STS #4	3	0xBC*(n+3)*045A6	For OC12c: The G1 value. For OC12nc: G1 for last OC3c of 4.
F2, STS #1	0	0xBC*n*045A7	For OC12c: The F2 value to be transmitted in the next frame. For OC12nc: F2 for first OC3c of 4.
F2, STS #2	1	0xBC*(n+1)*045A7	For OC12c: The F2 value. For OC12nc: F2 for second OC3c of 4.
F2, STS #3	2	0xBC*(n+2)*045A7	For OC12c: The F2 value. For OC12nc: F2 for third OC3c of 4.
F2, STS #4	3	0xBC*(n+3)*045A7	For OC12c: The F2 value. For OC12nc: F2 for last OC3c of 4.

Table 248 Transmit SONET/SDH OC-12 and OC-12c Path Overhead Byte Addresses (continued)

PATH OVERHEAD BYTE	CP# WITHIN A CLUSTER	C-5E NP ADDRESS*	NOTES
H4, STS #1	0	0xBC*n*045A8	For OC12c: The H4 value to be transmitted in the next frame. For OC12nc: H4 for first OC3c of 4.
H4, STS #2	1	0xBC*(n+1)*045A8	For OC12c: The H4 value. For OC12nc: H4 for second OC3c of 4.
H4, STS #3	2	0xBC*(n+2)*045A8	For OC12c: The H4 value. For OC12nc: H4 for third OC3c of 4.
H4, STS #4	3	0xBC*(n+3)*045A8	For OC12c: The H4 value. For OC12nc: H4 for last OC3c of 4.
Z3, STS #1	0	0xBC*n*045A9	For OC12c: The Z3 value to be transmitted in the next frame. For OC12nc: Z3 for first OC3c of 4.
Z3, STS #2	1	0xBC*(n+1)*045A9	For OC12c: The Z3 value. For OC12nc: Z3 for second OC3c of 4.
Z3, STS #3	2	0xBC*(n+2)*045A9	For OC12c: The Z3 value. For OC12nc: Z3 for third OC3c of 4.
Z3, STS #4	3	0xBC*(n+3)*045A9	For OC12c: The Z3 value. For OC12nc: Z3 for last OC3c of 4.
Z4, STS #1	0	0xBC*n*045AA	For OC12c: The Z4 value to be transmitted in the next frame. For OC12nc: Z4 for first OC3c of 4.
Z4, STS #2	1	0xBC*(n+1)*045AA	For OC12c: The Z4 value. For OC12nc: Z4 for second OC3c of 4.
Z4, STS #3	2	0xBC*(n+2)*045AA	For OC12c: The Z4 value. For OC12nc: Z4 for third OC3c of 4.
Z4, STS #4	3	0xBC*(n+3)*045AA	For OC12c: The Z4 value. For OC12nc: Z4 for last OC3c of 4.
Z5, STS #1	0	0xBC*n*045AB	For OC12c: The Z5 value to be transmitted in the next frame. For OC12nc: Z5 for first OC3c of 4.

Table 248 Transmit SONET/SDH OC-12 and OC-12c Path Overhead Byte Addresses (continued)

PATH OVERHEAD BYTE	CP# WITHIN A CLUSTER	C-5E NP ADDRESS*	NOTES
Z5, STS #2	1	0xBC*(n+1)*045AB	For OC12c: The Z5 value. For OC12nc: Z5 for second OC3c of 4.
Z5, STS #3	2	0xBC*(n+2)*045AB	For OC12c: The Z5 value. For OC12nc: Z5 for third OC3c of 4.
Z5, STS #4	3	0xBC*(n+3)*045AB	For OC12c: The Z5 value. For OC12nc: Z5 for last OC3c of 4.
J1_BUF, STS #1	0	0xBC*n*045AC to 0xBC*n*045EB	The J1 path trace message to be transmitted is written to these registers. • To support a 64Byte path, write the complete 64Byte message to these registers. • To support a 16Byte path trace, write the 16Byte message consecutively 4 times to these registers.
J1_BUF, STS #2	1	0xBC*(n+1)*045AC to 0xBC*(n+1)*045EB	For OC12c: The J1 path trace message. For OC12nc: J1 for second OC3c of 4.
J1_BUF, STS #3	2	0xBC*(n+2)*045AC to 0xBC*(n+2)*045EB	For OC12c: The J1 path trace message. For OC12nc: J1 for third OC3c of 4.
J1_BUF, STS #4	3	0xBC*(n+3)*045AC to 0xBC*(n+3)*045EB	For OC12c: The J1 path trace message. For OC12nc: J1 for last OC3c of 4.

* *n* can be CP0, CP4, CP8, or CP12

CP Configuration Space (SONET/SDH Specific)

In addition to the previous mentioned SONET registers that relate to SONET overhead, there are additional registers that relate to Configuration Space of the CPs. Some of them are described here.

CP Mode (SONET/SDH Specific Enable) Registers

The RxSONET and TxSONET blocks are enabled and disabled using the *SDP_Mode3* register bit [31] *RxRestx* field, bit [30] *RxEnable* field, bit [27] *RxSonetEna* field and the *SDP_Mode5* register bit [31] *TxRestx* field, bit [30] *TxEnable* field, bit [27] *TxSonetEna* field. The mode registers also contain other configuration bits which control SONET scrambling, OC-12/OC-12c/OC-3c modes, automatic insertion of far-end alarms (via clearing the *Manual_FEBE* bit) and other SONET configurations.

Table 249 SONET/SDH Specific Configuration Registers

REGISTER NAME	PURPOSE	ADDRESS	DETAILS
SDP_Mode3	Collects configuration mode bits relevant for programming the RxSDP machines.	0xBC*n*0464C	See "SDP_Mode3 Register (CP Mode Configuration Function)" on page 529.
SDP_Mode5	Collects configuration mode bits relevant for programming the TxSDP machines.	0xBC*n*04654	See "SDP_Mode5 Register (CP Mode Configuration Function)" on page 538.

CP Event and Interrupt (SONET/SDH Specific Event) Registers

SONET/SDH events including LOS, LOF, LOP, RDI-L, RDI-P, AIS-L, and AIS-P are monitored via the *SONET_Event*, *SONET_Event_Mask*, and *SDP_Mode2* registers. The *SONET_Event* register indicates whether a change in the state of any SONET event has occurred. The *SONET_Event_Mask* register is used to select which SONET events are of interest. If a SONET event occurs and that particular event has been enabled in the *SONET_Event_Mask* register, the *Event0* register bit [50] *SONETOH* field is set. This mechanism can be used to generate interrupts on state changes for various SONET defect conditions. Refer to "Event0 Register (CP Event and Interrupt Function)" on page 552.

The *SDP_Mode2* register bits [31:22] *SONET State* field detail the current defect condition (ON or OFF). Using this mechanism, defects can be monitored via interrupts on the CPRC (where interrupts are generated when the defect condition goes ON or OFF).

Table 250 SONET/SDH Specific Event Registers

REGISTER NAME	PURPOSE	ADDRESS	DETAILS
SDP_Mode2	Collects SONET/SDH alarm and status information.	0xBC*n*04648	See "SDP_Mode2 Register (CP Mode Configuration Function)" on page 526.
SONET_Event	Collects together SONET/SDH event bits from the SDP's.	0xBC*n*046C0	See "SONET_Event Register (CP Event and Interrupt Function)" on page 560.
SONET_Mask	Provides mask that selects bits in the SONET_Event register for event access.	0xBC*n*046C4	See "SONET_Mask Register (CP Event and Interrupt Function)" on page 569.

SONET/SDH Monitoring Example

The following is an example of a procedure for monitoring a given defect in the C-5e NP. Specifically, to detect loss of signal (LOS) in an interrupt handler.

1. Set bit [31] *Loss of Signal* field in the *SONET_Mask* register. This allows LOS to be flagged in the *SONET_Event* register.
2. Set bit [50] *SONETOH Event* field in the *Event_Mask0* register. This allows interrupts to be generated from changes in the *SONET_Event* register.
3. Using *KsEventRegisterInterrupt(),* link the interrupt handler function with the interrupt using the Freescale CPI software. This allows the CPRC to call the SONET interrupt handler when the *SONETOH Event* field, bit [50] state changes, in the *Event_Mask0* register.
4. In the interrupt handler function, check bit [31] *Loss of Signal* field in the *SDP_Mode2* register to determine the state of the defect (on or off).
5. Set bit [31] *Loss of Signal* field in the *SONET_Event* register to clear the event, so that future LOS events are not missed.
6. Perform any notification services required to the XPRC, or write information to local DMEM.
7. Exit the interrupt handler function.
8. The CPI clears the *SONETOH Event* field bit [50] before exiting, thus enabling new SONET/SDH events to generate future interrupts.

Automatic Protection Switch (APS) Overview

Linear Automatic Protection Switch (APS) is defined for SONET/SDH to provide protection switching at the line layer. This means that all SPEs (Synchronous Payload Envelopes) carried in an OC-N signal are protected together and all SPEs are switched simultaneously.

There are 2 types of linear APS architectures: 1+1 and 1:*n*.

- In the 1+1 architecture, the transmitted signal is continuously bridged at the electrical level to a working and a protection line so that the same payloads are transmitted identically to the far end. The receiving equipment chooses either the working or the protection signal as the one from which to select the traffic.
- In the 1:*n* architecture, any of *n* working channels can be bridged to a single protection line. The protection line can be used to transport extra traffic.

Signal Failure (SF) Definition

Signal Failure (SF) is "hard failure" condition resulting from the Line BER exceeding a pre-selected threshold. The BER threshold for an SF condition shall be configurable over the range of 10^{-3} to 10^{-5}.

Signal Degrade (SD) Definition

Signal Degrade (SD) is a "soft failure" condition resulting from the Line BER exceeding a pre-selected threshold. The BER threshold for an SD condition shall be configurable over the range of 10^{-5} to 10^{-9}.

Bit Error Rate algorithms should be designed such that the number of errors detected in a particular time period is less than the threshold for detecting an SD or SF condition and a corresponding probability that the number of errors are greater than or equal to the threshold. The goals below were written assuming a Poisson distribution of errors.

The goals are as follows:

- When the actual BER is greater than or equal to an SD or SF threshold, it is quickly detected and a switch is initiated.
- The probability of detecting a threshold crossing when the actual BER is below the threshold is very small and tolerant to burst errors.

One optional requirement is that for SD/SF conditions based on the BER, the probability that the switch initiation time is less than the objective curve should be greater than .95.

Switch Initiation Timing

There are timing requirements for switch initiation. Since statistically, a large sampling period is preferred to meet the above goals, the maximum timing requirements come into play. There are two (2) GR-253 requirements which detail this:

- For SF and SD conditions based on BER, the switch initiation time shall be below the "maximum" curve. Refer to Table 251 on page 797 with data from the maximum curve.

Table 251 Maximum Switch Initiation Times

ERROR RATE	OC-3C	OC-12C
10^{-3}	8ms	8ms
10^{-4}	13ms	8ms
10^{-5}	100ms	25ms
10^{-6}	1s	250ms
10^{-7}	10s	2.5s
10^{-8}	83s	21s
10^{-9}	667s	167s

- Switch *completion* time is required to be *50ms or less once initiated*.

Clearing of SD /SF Conditions

These rules *must be* implemented based upon the conditions listed:

- The clearing threshold for an SD or SF condition based on the BER shall be one-tenth the threshold for declaring the SD or SF.
- If an SD/SF condition is detected and the incoming signal's BER is greater than or equal to that SD or SF threshold, the probability that the Lite Terminating Equipment (LTE) detects that the BER is less than the SD or SF clearing threshold within the "maximum clearing" time listed shall be less than or equal to 10^{-6}. Refer to Table 252 on page 798.
- If an SD or SF condition is detected and the incoming signal's BER is less than or equal to the SD or SF clearing threshold, the probability that the LTE detects that the BER is less than that threshold within the "maximum clearing" time listed shall be greater than or equal to 0.99. Refer to Table 252 on page 798.

Table 252 Maximum Switch Clearing Time

SIGNAL DEGRADE/SIGNAL FAILURE THRESHOLD	CLEARING THRESHOLD	MAXIMUM/OBJECTIVE
10^{-3}	10^{-4}	10ms/none
10^{-4}	10^{-5}	100ms/25ms (OC12)
10^{-5}	10^{-6}	1s/250ms (OC12)
10^{-6}	10^{-7}	10s/2.5s (OC12)
10^{-7}	10^{-8}	100s/21s (OC12) 83s (OC3)
10^{-8}	10^{-9}	1,000s/167s (OC12) 667s (OC3)
10^{-9}	10^{-10}	10,000s/1,333s (OC12) 5,360s (OC3)

APS Protocol Using the K1 and K2 Bytes

Once detecting the SD/SF condition, a protection switch is initiated. This may be signaled to the far-end with the K1 and K2 bytes but is not required for 1+1. K1 indicates the request by a channel for a switch action. K2 indicates the bridging actions performed at the LTE and the provisioned architecture and mode of operation. For more information, see section 5.3.5 APS Channel Protocol in GR-253 core.

Determining Signal Degrade/Signal Failure Conditions with C-5e NP

As described in the earlier section describing GR-253 requirements, Bit Error Rates (BER) are noted in units of number of bits in error for every x bits. For example, 1bit error in 1000bits (10^{-3}). In order to determine the bit error rate, SONET/SDH standards assume a Poisson or Binomial distribution of errors. These can be approximated using a Normal (Gaussian) distribution when collected over a large sample period. For the purposes of this discussion, a binomial curve is approximated using the normal distribution.

Because of the random nature of the bit errors, it is impossible to specify a perfect maximum detection time. The standard reads that the detection time should be less than the objective in more than 95% of error events.

The normal curve can be used to obtain thresholds for given detection times to obtain the appropriate threshold values. An example follows to illustrate this. A normal random variable with a mean of 0 and a variance of 1 is called a *standard* normal random variable and this is denoted as Z.

The following equation is used to convert from a binomial distribution to the approximated normal distribution. If X is a binomial random variable, then Z is approximately a *standard* normal random variable where *np* is the mean and the square root of *np(1-p)* is the standard deviation. The approximation is good when *n* is large relative to *p*. For our discussion, *n* is the number of bits received and *p* is the error rate. Refer to Figure 127 on page 799.

Figure 127 Converting from Binomial to Approximate Normal Distribution Formula

$$Z = \frac{(X - np)}{\sqrt{np(1-p)}}$$

To determine the values to use for the detection threshold for bit error rate detection for SD/SF with a surety of 95% correctness, the following equation is used to convert from the standard normal distribution to the normal distribution we have.

If X is a normal random variable with E(X)= mean and V(X)= variance, then the random variable is a normal random variable with E(Z)=0 and V(Z)=1. That is, Z is a *standard* normal random variable. Refer to Figure 128 on page 800.

Figure 128 Converting from Standard Normal to Normal Distribution Formula

$$Z = \frac{(X - \mu)}{\sigma}$$

If this is true, the probability of 95% surety can be obtained using a standardization table.

Looking up .95 in the standardized normal distribution table, a standard deviation of less than 1.65 is required. To calculate the threshold once the mean and standard deviation are determined, use the following equation to determine the detection threshold. Refer to Figure 129 on page 800.

Figure 129 Detection Threshold Formula

$$X = \mu + 1.645\sigma$$

Let's take an OC-3c case and run some numbers through the equation for a given detection time:

np = (2430 bytes/frame) * (32 frames) * (8 bits/byte) (1 x 10^{-3} BER) = 622.08

Figure 130 OC-3c Detection Example

$$\sqrt{np(1-p)} = \sqrt{622(1-1\times10^{-3})} = 24.929$$

So, the detection threshold is 622.08 + (1.645)24.929 = 663 errors in 32frames or 4ms and an error rate of $1x10^{-3}$.

Table 253 on page 800 indicates the possible settings for detection times which could be used for OC-3c.

Table 253 Possible Settings for OC-3c Detection Times

OC-3C ERROR RATE	MAXIMUM TIME ALLOWED	EQUIVALENT NUMBER OF FRAMES	CHOSEN FRAME COUNT	THRESHOLD
10^{-3}	8ms	64	32	663
10^{-4}	13ms	104	52	85
10^{-5}	100ms	800	400	63
10^{-6}	1s	8000	4000	63

Table 254 on page 801 indicates the possible settings for detection times which could be used for OC-12c.

Table 254 Possible Settings for OC-12c Detection Times

OC-12C ERROR RATE	MAXIMUM TIME ALLOWED	EQUIVALENT NUMBER OF FRAMES	CHOSEN FRAME COUNT	THRESHOLD
10^{-3}	8ms	64	32	2417
10^{-4}	8ms	64	32	224
10^{-5}	25ms	200	100	64
10^{-6}	250ms	2000	1000	64

The chosen frame count value is not the same as the maximum time allowed. This still meets the requirements statistically, if the sample is large. The examples use half of the maximum time allowed value for the chosen frame count values.

For lower error rate thresholds (10^{-7} to 10^{-9}), the hardware is limited by the maximum frame count value of 8000 frames. To work around this issue in software, set the frame count value to the maximum (8000) and set the threshold to 1 and accumulate the larger range in software. Refer to Table 255 on page 801 and Table 256 on page 801.

Table 255 OC-3c Desired Thresholds for Lower Error Rates

OC-3C ERROR RATE	MAXIMUM TIME ALLOWED	EQUIVALENT NUMBER OF FRAMES	CHOSEN FRAME COUNT	THRESHOLD
10^{-7}	10s	80000	40000	64
10^{-8}	83s	664000	332000	52
10^{-9}	667s	5336000	2668000	41

Table 256 OC-12c Desired Thresholds for Lower Error Rates

OC-12C ERROR RATE	MAXIMUM TIME ALLOWED	EQUIVALENT NUMBER OF FRAMES	CHOSEN FRAME COUNT	THRESHOLD
10^{-7}	2.5s	20000	10000	64
10^{-8}	21s	168000	84000	53
10^{-9}	167s	1336000	668000	41

The information in "Determining Signal Degrade/Signal Failure Conditions with C-5e NP" is preliminary and the values in Table 253 on page 800, Table 254 on page 801, Table 255 on page 801 *and* Table 256 on page 801 *are subject to change.*

Appendix D

RISC CORE CUSTOM INSTRUCTIONS

Appendix Overview

This appendix covers the following topic:

- RISC Core Enhancements

RISC Core Enhancements

Twenty-four (24) total instructions have been added to the CPRC and XPRC. Sixteen (16) instructions are in addition to the subset of a MIPS™1 instruction set (excluding multiply, divide, floating point, unaligned loads and stores, move to hi and move to lo), mentioned earlier in this manual. In addition, eight (8) Branch Likely instructions from the standard MIPS™ 2 instruction set are provided in the CPRC and XPRC. They include: BEQL (Branch on Equal Likely), BNEL (Branch on Not Equal Likely), BLEZL (Branch on Less than or Equal to Zero Likely), BGTZL (Branch on Greater Than Zero Likely), BLTZL Branch on Less Than Zero Likely), BGEZL (Branch on Greater than or Equal to Zero Likely), BLTZALL (Branch on Less Than Zero And Link Likely), and BGEZALL (Branch on Greater than or Equal to Zero And Link Likely). Refer to the MIPS™ 2 RISC architecture manual for their detail descriptions.

The unaligned load and store instructions LWL (Load Word Left), LWR (Load Word Right), SWL (Store Word Left), and SWR (Store Word Right) are not supported C-5e NP. Execution of these instructions results in an illegal instruction exception.

Individual Custom Instructions

The name, format, description and operation for each custom instruction is provided in this section.

CLZ - Count leading zeros

Bit Position	31 26	25 21	20 16	15 11	10 6	5 0
Field Name	SPECIAL2 011100	rs	00000	rd	00000	CLZ 100000
Number of Bits	6	5	5	5	5	6

Format:
CLZ rd, rs

Description:
Register rs is scanned from bit 31 to bit 0, and the number of zeros before the first set bit is stored in register rd. If register rs is zero, then a value of 32 is stored in register rd.

Operation:
T: GPR[rd] <- # of leading zeros{GPR[rs]}

CSWAP - Context swap

Bit Position	31 26	25 21	20 16	15 11	10 6	5 0
Field Name	COPO 010000	10000	rt	unused	unused	CSWAP 010001
Number of Bits	6	5	5	5	5	6

Format:
CSWAP rt

Description:
The program unconditionally jumps to the Program Counter (PC) saved for “new context” and switches to the register file saved for “new context”. The PC for current context for the instruction after the delay slot is saved. The “new context” value is stored in rt and its legal range is 0 to 3. Fixed mapping between the context and coprocessor zero’s (CP0) registers are as follows:

Context0 is mapped to CP0 register16,
Context1 is mapped to CP0 register17,
Context2 is mapped to CP0 register18,
Context3 is mapped to CP0 register19.

In addition, CP0 register3 holds the current context number as it changes. Its content can be read.

Operation:
T: SAVED_PC[“active context”] <- PC+8
T+1: PC <- SAVED_PC[RT1:0]
“active context” <- RT[1:0]

BEQNL - Branch on equal not likely

Bit Position	31 26	25 21	20 16	15 0
Field Name	**BEQNL 011000**	**rs**	**rt**	**offset**
Number of Bits	**6**	**5**	**5**	**16**

Format:
BEQNL rs, rt, offset

Description:
A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16bit offset, shifted left two bits and sign-extended to 32 bits. The contents of general register *rs* and the contents of general register *rt* are compared. If the two registers are equal, the target address is branched to, with a delay of one instruction. If the conditional branch *is* taken, the instruction in the branch delay slot is nullified.

Operation:
T: target <- $(offset_{15})^{14}$ ||offset||0^2
Condition <- (GPR[rs] = GPR[rt])
T+1: if condition then
PC <- PC + target
NullifyCurrentInstruction
endif

BGEZALNL - Branch on greater than or equal to zero and link not likely

Bit Position	31 26	25 21	20 16	15 0
Field Name	REGIMM	rs	BGEZANLNL 10111	offset
Number of Bits	6	5	5	16

Format:
BGEZALNL rs, offset

Description:
A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16bit offset, shifted left two bits and sign-extended to 32 bits. Unconditionally, the address of the instruction after the delay slot is placed in the link register, r31. If the contents of general register rs have the sign bit cleared, then the program branches to the target address, with a delay of one instruction.

General register rs may not be general register 31, because such an instruction is not restartable. An attempt to execute this instruction is not trapped, however. If the conditional branch *is* taken, the instruction in the branch delay slot is nullified.

Operation:
T: target <- $(offset_{15})^{14}$ ||offset||0^2
Condition <- $GPR[rs]_{31} = 0$
GPR[31] <- PC+8
T+1: if condition then
PC <- PC + target
NullifyCurrentInstruction
endif

BGEZNL - Branch on greater than or equal to zero not likely

Bit Position	31 26	25 21	20 16	15 0
Field Name	**REGIMM**	**rs**	**BGEZNL 00111**	**offset**
Number of Bits	**6**	**5**	**5**	**16**

Format:
BGEZNL rs, offset

Description:
A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16bit offset, shifted left two bits and sign-extended to 32 bits. If the contents of general register rs have the sign bit cleared, then the program branches to the target address, with a delay of one instruction. If the conditional branch *is* taken, the instruction in the branch delay slot is nullified.

Operation:
T: target <- $(offset_{15})^{14}$ ||offset||0^2
Condition <- $GPR[rs]_{31} = 0$
T+1: if condition then
PC <- PC + target
NullifyCurrentInstruction
endif

BGTZNL - Branch on greater than zero not likely

Bit Position	31 26	25 21	20 16	15 0
Field Name	BGTZNL 011001	rs	00000	offset
Number of Bits	6	5	5	16

Format:
BGTZNL rs, offset

Description:
A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16bit offset, shifted left two bits and sign-extended to 32 bits. The contents of general register rs are compared to zero. If the contents of general register rs have the sign bit cleared and not equal to zero, then the program branches to the target address, with a delay of one instruction. This instruction is only valid when rt=0. If the conditional branch *is* taken, the instruction in the branch delay slot is nullified.

Operation:
T: target <- $(offset_{15})^{14}$ ||offset||0^2
Condition <- $GPR[rs]_{31}$ = 0 *and* GPR[rs] != 0^{32}
T+1: if condition then
PC <- PC + target
NullifyCurrentInstruction
endif

BLEZNL - Branch on less than or equal to zero not likely

Bit Position	31 26	25 21	20 16	15 0
Field Name	**BLEZNL 011010**	**rs**	**00000**	**offset**
Number of Bits	**6**	**5**	**5**	**16**

Format:
BLEZNL rs, offset

Description:
A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16bit offset, shifted left two bits and sign-extended to 32 bits. The contents of general register rs are compared to zero. If the contents of general register rs have the sign bit set or are equal to zero, then the program branches to the target address, with a delay of one instruction. This instruction is only valid when rt=0. If the conditional branch *is* taken, the instruction in the branch delay slot is nullified.

Operation:
T: target <- $(offset_{15})^{14}$ ||offset||0^2
Condition <- $GPR[rs]_{31} = 1$ *or* $GPR[rs] = 0^{32}$
T+1: if condition then
PC <- PC + target
NullifyCurrentInstruction
endif

BLTZALNL - Branch on less than zero and link not likely

Bit Position	31 26	25 21	20 16	15 0
Field Name	REGIMM	rs	BLTZALNL 10110	offset
Number of Bits	6	5	5	16

Format:
BLTZALNL rs, offset

Description:
A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16bit offset, shifted left two bits and sign-extended to 32 bits. Unconditionally, the address of the instruction after the delay slot is placed in the link register, r31. If the contents of general register rs have the sign bit set, then the program branches to the target address, with a delay of one instruction.

General register rs may not be general register 31, because such an instruction is not restartable. An attempt to execute this instruction is not trapped, however. If the conditional branch *is* taken, the instruction in the branch delay slot is nullified.

Operation:
T: target <- $(offset_{15})^{14}$ ||offset||0^2
Condition <- $GPR[rs]_{31} = 1$
GPR[31] <- PC+8
T+1: if condition then
PC <- PC + target
NullifyCurrentInstruction
endif

BLTZNL - Branch on less than zero not likely

Bit Position	31 26	25 21	20 16	15 0
Field Name	**REGIMM**	**rs**	**BLTZNL 00110**	**offset**
Number of Bits	**6**	**5**	**5**	**16**

Format:
BLTZNL rs, offset

Description:
A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16bit offset, shifted left two bits and sign-extended to 32 bits. If the contents of general register rs have the sign bit set, then the program branches to the target address, with a delay of one instruction. If the conditional branch *is* taken, the instruction in the branch delay slot is nullified.

Operation:
T: target <- $(offset_{15})^{14}$ ||offset||0^2
Condition <- $GPR[rs]_{31} = 1$
T+1: if condition then
PC <- PC + target
NullifyCurrentInstruction
endif

BNENL - Branch on not equal not likely

Bit Position	31 26	25 21	20 16	15 0
Field Name	BNENL 011011	rs	rt	offset
Number of Bits	6	5	5	16

Format:
BNENL rs, rt, offset

Description:
A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16bit offset, shifted left two bits and sign-extended to 32 bits. The contents of general register rs and the contents of general register rt are compared. If the two registers are not equal, the target address is branched to, with a delay of one instruction. If the conditional branch *is* taken, the instruction in the branch delay slot is nullified.

Operation:
T: target <- $(offset_{15})^{14}$ ||offset||0^2
Condition <- GPR[rs] != GPR[rt]
T+1: if condition then
PC <- PC + target
NullifyCurrentInstruction
endif

BBIT0 - Branch on bit clear

Bit Position	31 26	25 21	20 16	15 0
Field Name	**BBIT0 011110**	**rs**	**p**	**offset**
Number of Bits	**6**	**5**	**5**	**16**

Format:
BBIT0 rs, p, offset

Description:
A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16bit offset, shifted left two bits and sign-extended to 32 bits. If the contents of general register rs have bit p clear, then the program branches to the target address, with a delay of one instruction.

Operation:
T: target <- $(offset_{15})^{14}$ ||offset||0^2
Condition <- GPR[rs] [p] =0
T+1: if condition then
PC <- PC + target
endif

BBIT1 - Branch on bit set

Bit Position	31 26	25 21	20 16	15 0
Field Name	BBIT1 011111	rs	p	offset
Number of Bits	6	5	5	16

Format:
BBIT1 rs, p, offset

Description:
A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16bit offset, shifted left two bits and sign-extended to 32 bits. If the contents of general register rs have bit p set, then the program branches to the target address, with a delay of one instruction.

Operation:
T: target <- $(offset_{15})^{14}$ ||offset||0^2
Condition <- GPR[rs] [p] =1
T+1: if condition then
PC <- PC + target
endif

INS - Insert bit field

Bit Position	31 26	25 21	20 16	15 11	10 6	5 0
Field Name	SPECIAL2 011100	rs	rt	len	p	INS 100010
Number of Bits	6	5	5	5	5	6

Format:
INS rt, rs, p, len

Description:
The contents of general purpose register rt from bit location p + (len-1) to bit location p are replaced with the contents of general purpose register rs from bit location len-1 to bit location 0. If p +(len-1) >31 then the results are undefined.

Operation:
T: GPR[rt][p+(len-1): p] <- GPR[rs] [(len-1):0]

CINS - Clear then insert bit field

Bit Position	31 26	25 21	20 16	15 11	10 6	5 0
Field Name	SPECIAL2 011100	rs	rt	len	p	INS 100101
Number of Bits	6	5	5	5	5	6

Format:
CINS rt, rs, p, len

Description:
The contents of general purpose register rt are zeroed and the contents of register rt from bit location p + (len-1) to bit location p are replaced with the contents of general purpose register rs from bit location len-1 to bit location 0. If p +(len-1) >31 then the results are undefined.

Operation:
T: GPR[rt] <- 0
GPR[rt][p+(len-1): p] <- GPR[rs] [(len-1):0]

EXTU - Extract bit field unsigned

Bit Position	31 26	25 21	20 16	15 11	10 6	5 0
Field Name	SPECIAL2 011100	rs	rt	len	p	EXTU 100011
Number of Bits	6	5	5	5	5	6

Format:
EXTU rt, rs, p, len

Description:
The contents of general purpose register rt are replaced with the contents of general purpose register rs from bit location p+(len-1) to bit location p shifted right by p bits and zero extended to 32 bits. If p +(len-1) >31 then the results are undefined.

Operation:
T: GPR[rt] <- zero extend {GPR[rs] [p+(len-1):p] >> p }

EXTS - Extract bit field signed

Bit Position	31 26	25 21	20 16	15 11	10 6	5 0
Field Name	SPECIAL2 011100	rs	rt	len	p	EXTS 100100
Number of Bits	6	5	5	5	5	6

Format:
EXTS rt, rs, p, len

Description:
The contents of general purpose register rt are replaced with the contents of general purpose register rs from bit location p+(len-1) to bit location p shifted right by p bits and sign extended to 32 bits. If p +(len-1) >31 then the results are undefined.

Operation:
T: GPR[rt] <- sign extend {GPR[rs] [p+(len-1):p] >> p }

Appendix E

PCI BYTE SWAPPING

Appendix Overview

This appendix covers the following topics:

PCI Byte Swapping Overview

The C-5e NP provides a byte swapping feature used to move data between the PCI Bus Little Endian environment and the C-5e NP Big Endian environment.

Most endianess issues come from the difference in the addressing (and/or byte enable assignment) and the position of bytes within a 32bit double word between the two (2) environments as shown in Figure 131 on page 822.

Figure 131 Little Endian vs. Big Endian

PCI Little Endian Environment

Bits	31:24	23:16	15:8	7:0
Address	3	2	1	0
Contents	0xAA	0xBB	0xCC	0xDD

C-5e NP Big Endian Environment

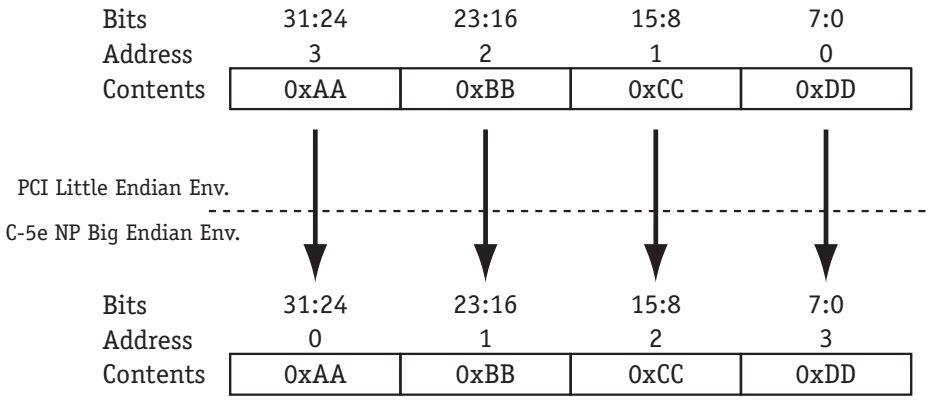
Bits	31:24	23:16	15:8	7:0
Address	0	1	2	3
Contents	0xAA	0xBB	0xCC	0xDD

These differences create a problem when 32bit double word and byte transactions pass from one environment to the other.

Default Mode

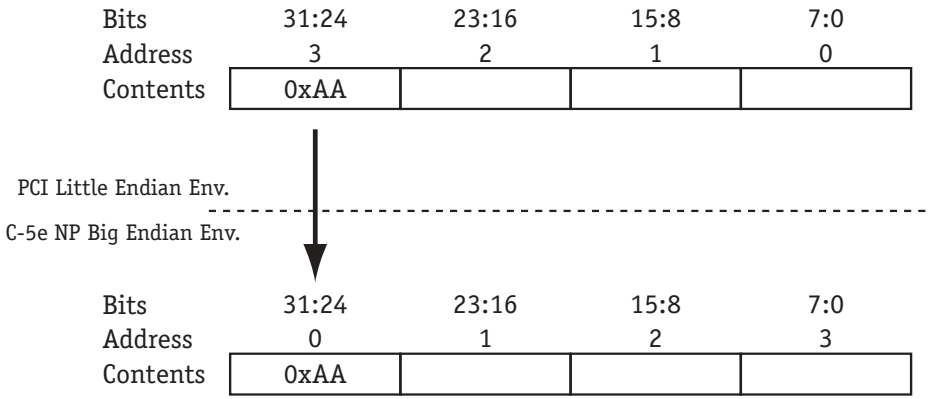
The current hardware handles this transition by maintaining the data byte positions within the 32bit double word and effectively swapping the byte address of the bytes. Figure 132 on page 823 illustrates a 32bit double word write from the PCI bus into a register somewhere in the Big Endian C-5e NP environment. Notice how the data that was in bits [31:24] on the PC I bus are stored in bits [31:24] in the C-5e NP register even though the address of that byte is 3 on the PCI Bus and 0 within the C-5e NP.

Figure 132 PCI 32bit Aligned Double Word Access to C-5e NP



This means that software must be very careful with the address when performing accesses to smaller than a 32bit double word. For example, in Figure 133 on page 823 the PCI performs a byte write to address 3 on the PCI Bus. Again the data on bits [31:24] on the PCI Bus arrives in bits [31:24] in the C-5e NP register; however, as far as a processing element within the C-5e NP is concerned that byte is at address 0.

Figure 133 PCI Byte Access to C-5e NP (PCI Address 3)



The same rules apply when data flows in the opposite direction. Byte position remain the same, but the addresses associated with the bytes swap as the data crosses the interface.

Figure 134 on page 824 illustrates a 32bit double word access and Figure 135 on page 824 shows a byte access flowing from the C-5e NP to the PCI.

Figure 134 C-5e NP 32bit Aligned Double Word Access to PCI

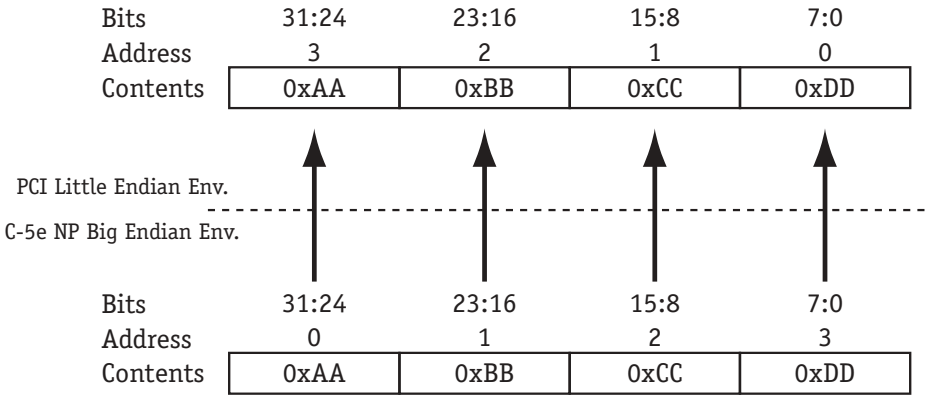
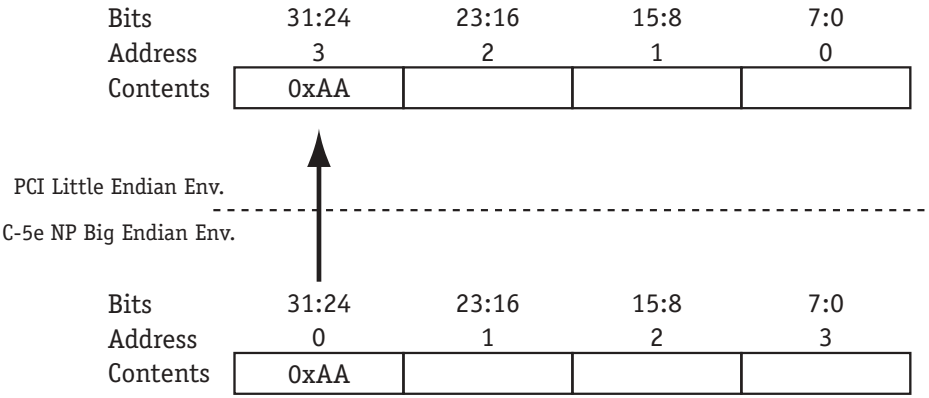


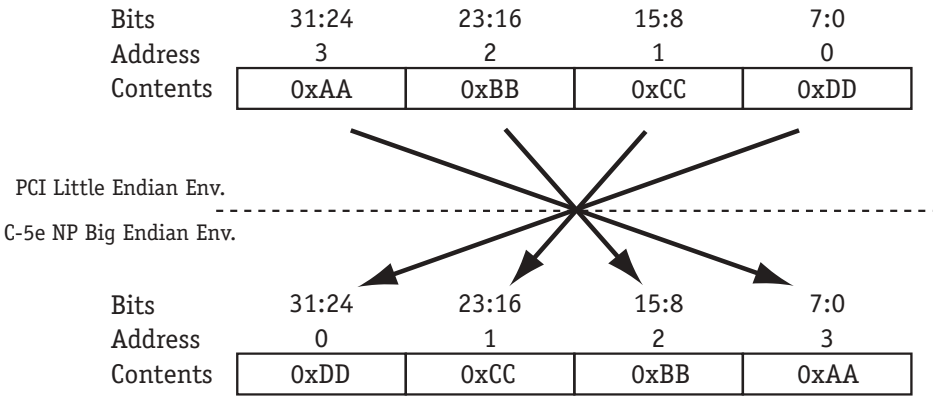
Figure 135 C-5e NP Byte Access to PCI (C-5e NP Address 0)



Byte Swapping Mode

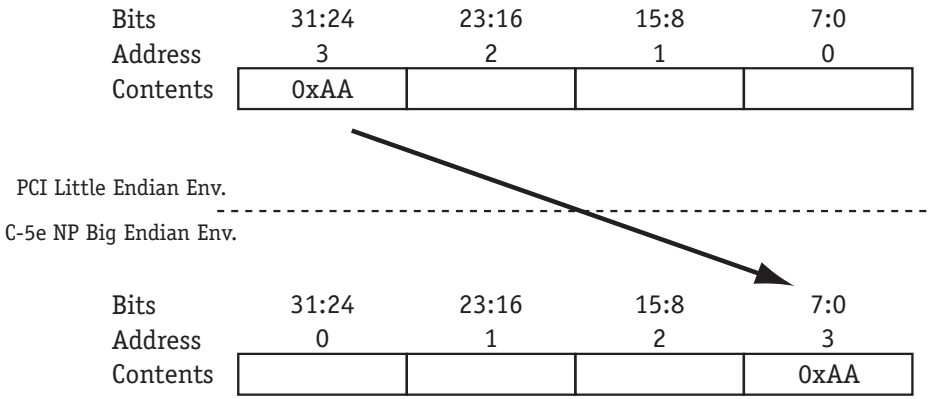
The Byte Swapping Mode is an alternative to the Default Mode that handles this transition by maintaining consistent byte addresses and swapping the data byte positions. Figure 136 on page 825 illustrates a 32bit double word write from the PCI bus into a register somewhere in the Big Endian C-5e NP environment. Notice how the data that was in bits [31:24] on the PC I bus are stored in bits [7:0] in the C-5e NP register, which is the byte at the same address as that on the PCI bus. This means that if a C-5e NP internal processing element accesses the same 32bit double word, the data will be byte swapped from what the PCI originally wrote.

Figure 136 PCI 32bit Aligned Double Word Access to C-5e NP



In this mode, the software does not have to worry about the byte addresses, because they remain consistent across the interface. For example, in Figure 137 on page 826 the PCI is doing a byte write to address 3 on the PCI Bus. Again the data on bits [31:24] on the PCI Bus arrive in bits [7:0] in the C-5e NP register, which is also C-5e NP internal byte address 3.

Figure 137 PCI Byte Access to C-5e NP (PCI Address 3)



The same rules apply when data is flowing in the opposite direction. Byte address remains the same, but the byte positions are swapped as the data crosses the interface. Figure 138 on page 826 illustrates a 32bit double word access and Figure 139 on page 827 shows a byte access flowing from the C-5e NP to the PCI.

Figure 138 C-5e NP 32bit Aligned Double Word Access to PCI

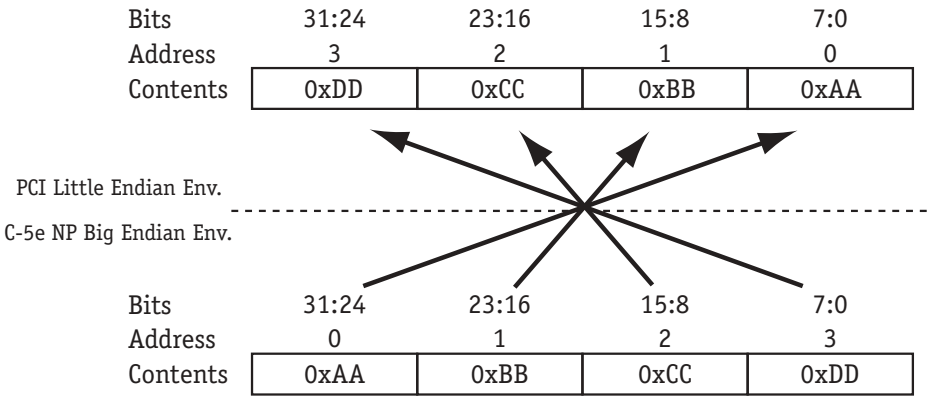
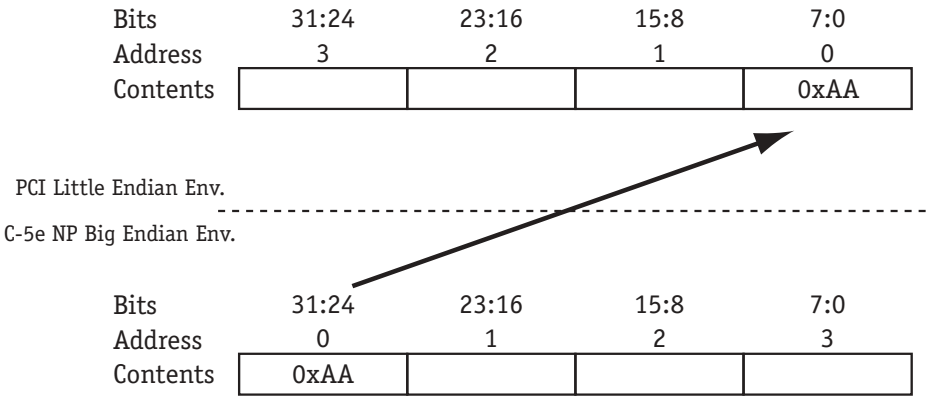


Figure 139 C-5e NP Byte Access to PCI (C-5e NP Address 0)

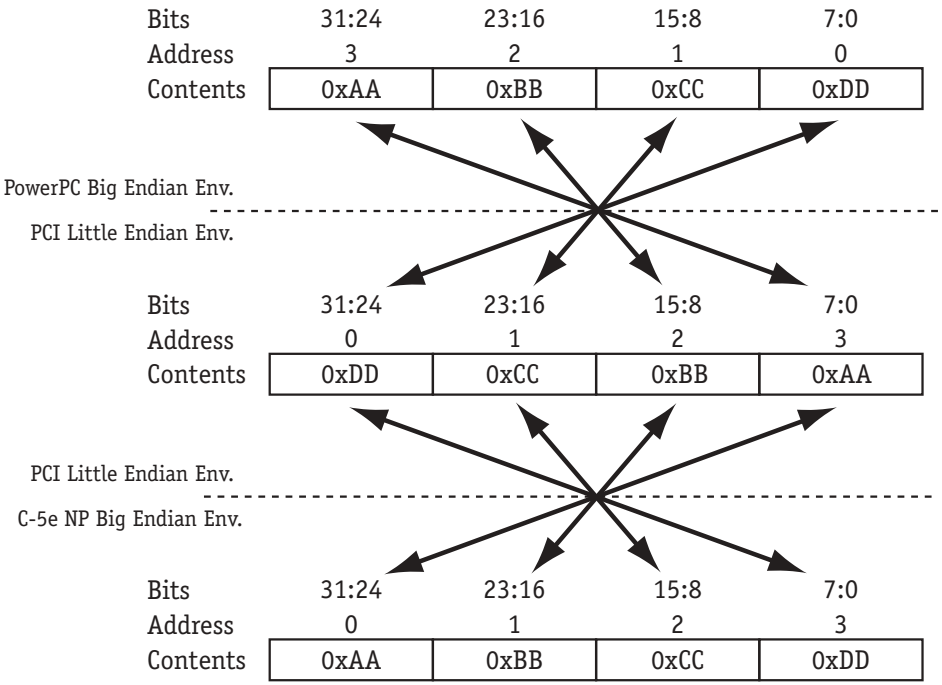


Primary Application Using Byte Swapping Mode

The primary application of the Byte Swapping Mode is when the host processor in the system is a Big Endian processor and the processor's bridge to the Little Endian PCI Bus performs byte swapping, as illustrated in Figure 140 on page 828.

The two (2) byte swapping operations that occur as data passes from or to the PowerPC Big Endian Environment, over the PCI Bus and to or from the C-5e NP Big Endian Environment cancel each other out. This provides both processing environments with a consistent view of all of the data.

Figure 140 C-5e NP 32bit Aligned Double Word Access to PCI



Implementing Byte Swapping Mode

The default configuration for access to or from the C-5e NP does *not* perform byte swapping, that is, the Default Mode. The Byte Swapping Mode can be enabled for specific accesses based on programmable bits within the two (2) *PCI Inbound BARn Translation* registers, the eight (8) *PCI Outbound BARn Translation* registers, or the control registers associated with the PCI transmit and receive transfer control blocks. Table 257 on page 829 defines the details for each transaction source/target pair that uses the PCI interface. Table 258 on page 830 lists the Inbound and Outbound Bar Translation registers.

Table 257 Byte Swapping Support Specification

SOURCE	TARGET	BYTE SWAPPING SUPPORT
PCI Bus	C-5e NP PCI Config. Registers	No Byte Swapping
PCI Bus	XP Config. Registers C-5e NP Serial Bus PROM Interface CPs via Global Bus C-5e NP Ring Bus DMEM24 DMEM25	Byte swapping controlled by bit 0 (BAR0 accesses) and bit 1 (BAR 1 accesses) of the PCI Inbound Byte Swap Control register. If the bit is set to a 1, byte swapping occurs as the data passes through the PCI. Each enable bit control s the byte swapping for only transactions decoded by its corresponding base address register.
XP/RC	C-5e NP PCI Config Regs	No Byte Swapping
XP/RC	External PCI Space	Byte swapping controlled by bits 0 through 7 of the PCI Outbound Byte Swap Control register. If the bit is set to a 1, byte swapping occurs as the data passes through the PCI. Each enable bit controls the byte swapping for only transactions decoded by its corresponding base address register, where bits 0 through 7 correspond with BARs 0 through 7, respectively.
Rx XCB#24 (Scope 0)	External PCI Space	Byte swapping controlled by bit 8 of PCI Outbound Byte Swap Control register. If the bit is set to a 1, byte swapping occurs as the data passes through the PCI interface.
Rx XCB#24 (Scope 1)	External PCI Space	Byte swapping controlled by bit 9 of PCI Outbound Byte Swap Control register. If the bit is set to a 1, byte swapping occurs as the data passes through the PCI interface.
Tx XCB #24 (Scope 0)	External PCI Space	Byte swapping controlled by bit 10 of PCI Outbound Byte Swap Control register. If the bit is set to a 1, byte swapping occurs as the data passes through the PCI interface.
Tx XCB #24 (Scope 1)	External PCI Space	Byte swapping controlled by bit 11 of PCI Outbound Byte Swap Control register. If the bit is set to a 1, byte swapping occurs as the data passes through the PCI interface.

Table 258 Inbound and Outbound Barn Transaction Registers

ADDRESS	REGISTER NAME	DETAILS
0xBD808040	PCI Inbound BAR0 Translation	See "PCI Inbound BAR0 Translation Register (XP PCI Configuration Function)" on page 598
0xBD808044	PCI Inbound BAR1 Translation	See "PCI Inbound BAR1 Translation Register (XP PCI Configuration Function)" on page 598
0xBD808220 to 0xBD80823C	Outbound BAR0 Transaction to Outbound Bar7 Transaction	See "Outbound BAR0 Translation Register (XP Configuration Function)" on page 607

PCI Inbound and Outbound Byte Swap Registers

The control bits for byte swapping are located in dedicated configuration registers, to allow these bits to be configured and then "forgotten". This makes it possible for more of the code to be written without knowing about the proper setting of these bits. Table 259 on page 831 lists the two (2) PCI Inbound and Outbound Byte Swap Control registers.

Table 259 PCI Inbound and Outbound Byte Swap Control Registers

ADDRESS	REGISTER NAME	DETAILS
0xBD808050	PCI Inbound Byte Swap Control	See "PCI Inbound Byte Swap Control Register (XP PCI Configuration Function)" on page 600.
0xBD80828C	PCI Outbound Byte Swap Control	See "PCI Outbound Byte Swap Control Register (XP Configuration Function)" on page 616.

Appendix F

C-5E NP SYSTEM CONFIGURATION

Appendix Overview

This appendix covers the following topics:

- C-5e NP System Configuration and Overview
- C-5e and M-5 Configuration Types and Their Options
- C-5e Methods for Handling High Speed (OC-48) PDUs
- M-5 Channel Adapter Overview

C-5e NP System Configuration and Overview

The C-5e NP product offers OC-48c line speeds using the M-5 Channel Adapter. Table 260 on page 834 describes each device and its role within the C-5e NP system.

Table 260 Roles of Each Device Within the C-5e NP System

ITEM	FUNCTION
C-5e	The C-5e individual channels each support up to a single OC-3c stream. The C-5e has 16 CPs that can be clustered together into four (4) clusters. These four (4) clusters can be grouped together (aggregation) to support higher bandwidth streams. Through aggregation, a cluster can support a single OC-12c stream. A cluster's CPs each represents a stage in a pipeline, thus providing the processor cycles required by each PDU. For the C-5e to support OC-48c with the front ports, all 16 CPs have to be aggregated together (4 clusters, each running OC-12c =OC-48c). A cluster, due to its logical and physical organization, has the ability to share data and synchronize in order to support cluster aggregation. However, this communication does *not* exist across cluster boundaries. The 16 CPs required for OC-48c cannot share data, or enforce order across the CPs when descriptors are enqueued for PDU transmission (Rx or Tx). The C-5e uses a method called *Aggregation Queueing*. Aggregation queueing provides two benefits: it allows clustered CPs to share queues, and allows all 16 CPs to share one (1) single queue to transmit one (1) single concatenated stream (up to four (4) clusters can transmit from one (1) queue). Another method used by the C-5e is called *Speculative Enqueue*. Speculative Enqueue allows a fixed latency for PDU streams, from the start to enqueue. This accommodates the QMU since it uses sequence numbers to order PDU enqueues in a concatenated stream. Also, this method prevents uneven PDU flows that can cause overruns of PDUs, and large gaps in the streams, thus, providing a more efficient bandwidth of PDU flow. The enqueues are speculative because at the time they are enqueued, the CRC is *not* calculated. Typically, speculative enqueues are used on CPs for enqueue (only) operations.
M-5	The Channel Adapter (M-5), used with the C-5e provides the required PDU ordering for both packet and cells for the front ports (CPs). In addition, it provides bus translation from multi-physical layer (MPHY), 32bit UTOPIA Level 3 (UTOPIA-L3) to the C-5e front ports (CPs) and back port (FP), as well as, bus translation from Saturn POS-PHY Level 3 (POS-PHY-L3) to the C-5e front ports (CPs) and back port (FP), thus allowing OC-48c bandwidths. Additionally, the front ports (CPs) use a Gigabit Media Independent Interface (GMII) that can be modified to provide flow control that interfaces with the SDPs. The method that the M-5 uses to provide PDU ordering is called *Sequence Numbers*. Sequencer numbers are 13bits long and are included as part of the enqueue and dequeue operations. Sequence Numbers are used whenever a flow is spread across more than one cluster in the case of the front ports (CPs). The FPRx does not need sequence numbers since it maintains strict ordering internally. Refer to "M-5 Channel Adapter Overview" on page 843.

C-5e and M-5 Configuration Types and Their Options

Numerous configurations are supported using the C-5e with the M-5 companion device.

In general, three (3) types of system configurations are supported using chip set combinations. The three (3) system configuration types are:

- Basic Port Aggregation !OC-48 System Configuration, Using 1 C-5e
- Basic OC-48c to OC-48c System Configurations, Using 2 C-5e and 2 M-5s
- Basic OC-48(c) to Switch Fabric System Configurations, Using 1 C-5e, and 1 M-5

Table 261 on page 837 indicates the supported C-5e system configurations and details some of their parameters. Table 261 on page 837 groups the C-5e supported system configurations into three (3) types. Illustrations of the three (3) system configuration types are shown in Figure 141 on page 839, Figure 142 on page 840, and Figure 143 on page 841.

The configurations listed in Table 261 on page 837 are specified by the mode of the front ports (CPs) and back port (FP). Also, the configurations specify three (3) C-5e methods that are used. They include: queueing model, sequence numbers queueing aggregation, and external scheduler mode. Notes are provided that pertain to specific configurations. These items are described here:

Front Ports (CPs) and Back Port (FP) Configurations

The modes of the front ports (CPs) consist of:

- Clustered (M-5)
- Configurable (non-clustered, completely configurable)
- Requires Freescale Reference Application Software

Clustered means that the front ports (CPs) are connected to a M-5, the four (4) CPs are aggregated together for OC-12c traffic, and the SDP are set for Gigabit Media Independent Interface (GMII). Configurable means the front ports (CPs) are set to support any mode up to and including two 1Gbit Ethernet ports. Requires Freescale Reference Application Software, means that certain application software is needed.

Freescale Reference Application Software refers to a segmented, interleaved application that manages the bandwidth and individual flow control of up to 3 STS-1 (Synchronous Transport Signal level 1) flows for up to 16 STS-1s per cluster. Any combination of STS-1, STS-3, STS-12, and Gbit are supported.

External Scheduler Mode

The C-5e provides two (2) modes for managing queues:

- Internal Mode (using the internal QMU only)
- External Mode

The Internal Mode (QMU) functions as described in the QMU section.

Refer to "External Scheduler Mode" on page 457 for information on the External Mode operation.

Queueing Model Configurations

The Queueing Model refers to the organization of either the QMU or the Q-5. Three (3) configurations are possible:

- Clustered front ports (CPs) and QMU= the CPs share up to 128 queues for transmit using the queue aggregation method
- Clustered Front ports (CPs) and QMU= the CPs share one Queue in 16 way Aggregation.
- Normal configured back port (FP) and QMU = 32 port groups with 4 priorities *or* 16

Sequence Numbers Configurations

The Sequence Numbers are generated values associated with PDUs to maintain their order. Sequence numbers are required in one (1) configuration:

- For front ports (CPs), when one (1) flow is spread across multiple cluster's of CPs (>1)

C-5e Methods

Three (3) C-5e methods are used to support the various system configurations. In general, these methods are used to handle high speed (OC-48) PDU streams.

- Sequence Numbers
- Aggregated Queueing (x16)
- Speculative Enqueue

Refer to "C-5e Methods for Handling High Speed (OC-48) PDUs" on page 842 for descriptions of each method.

Notes

To ensure proper system configuration, refer to the notes for that particular configuration.

Table 261 Supported C-5e System Configurations

FRONT PORT (CPS) (CONFIGURATIONS	BACK PORT (FP)/OTHER PORT (>1 C-5E) (CONFIGURATIONS)	FRONT PORT (CONFIGURATIONS)	QUEUEING MODEL (FRONT PORT)	QUEUEING MODEL (BACK PORT)	SEQUENCE NUMBERS FOR FRONT PORT (ENQUEUE/DEQUEUE)	SEQUENCE NUMBERS FOR BACK PORT (ENQUEUE/DEQUEUE)	REQUIRED C-5E METHODS	NOTES
Basic Port Aggregation !OC-48 System Configuration, Using 1 C-5e								
Standard Ports (Not including Gbit Ethernet)	UTOPIA L3	Configurable	<=128 queues per port	4<= queues <=48	No/No	No/No		Refer to Figure 141 on page 839.
Basic OC-48c to OC-48c System Configurations (allows max.aggregation configuration), Using 2 C-5es, 2M-5s								
OC-48c SPHY	OC-48c SPHY*	Clustered (M-5)	<= 128 queues for all CPs	N/A	Yes/Yes	N/A N/A	x16 Queue Aggregation, Speculative Enqueues	Needs two C-5e's half-duplex configuration Refer to Figure 142 on page 840.
OC-48c SPHY	OC-48c SPHY*	Clustered (M-5)	1 Queue for all CPs	N/A	Yes/Yes	N/A N/A	x16 Queue Aggregation, Speculative Enqueues, External Q-5	Needs two C-5e's half-duplex configuration Refer to Figure 142 on page 840.
OC-48 MPHY	OC-48c MPHY*	Requires Freescale Reference Application Software and (M-5)	<= 128 queues per CP/Cluster	No	No/No	N/A N/A		Needs two C-5e's half-duplex configuration Refer to Figure 142 on page 840.

Table 261 Supported C-5e System Configurations (continued)

FRONT PORT (CPS) (CONFIGURATIONS)	BACK PORT (FP)/OTHER PORT (>1 C-5E) (CONFIGURATIONS)	FRONT PORT (CONFIGURATIONS)	QUEUEING MODEL (FRONT PORT)	QUEUEING MODEL (BACK PORT)	SEQUENCE NUMBERS FOR FRONT PORT (ENQUEUE/DEQUEUE)	SEQUENCE NUMBERS FOR BACK PORT (ENQUEUE/DEQUEUE)	REQUIRED C-5E METHODS	NOTES
OC-48 MPHY	OC-48c MPHY*	Requires Freescale Reference Application Software and (M-5)	1Queue per CP/Cluster	N/A	No/No	N/A N/A		Needs two C-5e's half-duplex configuration Refer to Figure 142 on page 840
Basic OC-48(c) to Switch Fabric System Configurations, Using 1 C-5e, 1 M-5								
OC-48(c) SPHY	Switch Fabric (like UTOPIA, CSIX, etc.)	Clustered (M-5)	<= 128queues for all CPs	32x4 or, 16x8	Yes/Yes	No/No	Speculative Enqueues, x16 Queue Aggregation	OC-12 max flow Refer to Figure 143 on page 841.
OC-48(c) SPHY	Switch Fabric (like UTOPIA, CSIX, etc.)	Clustered (M-5)	1 Queue for all CPs	32 or 16 Queues	Yes/Yes	No/No	Speculative Enqueues, x16 Queue Aggregation	OC-12 max flow Refer to Figure 143 on page 841.
OC-48 MPHY	Switch Fabric (like UTOPIA, CSIX, etc.)	Requires Freescale Reference Application Software and (M-5)	<= 128 queues per CP/Cluster	32x4 or, 16x8	No/No	No/No		OC-12 max flow Refer to Figure 143 on page 841.
OC-48 MPHY	Switch Fabric (like UTOPIA, CSIX, etc.)	Requires Freescale Reference Application Software and (M-5)	1 Queue per CP/Cluster	32 or, 16 Queues	No/No	No/No	Speculative Enqueues	OC-12 max flow Refer to Figure 143 on page 841.

* Other port refers to other front port of a second C-5e in a two chip set configuration.

Figure 141 Basic Port Aggregation !OC-48 System Configuration, Using 1 C-5e

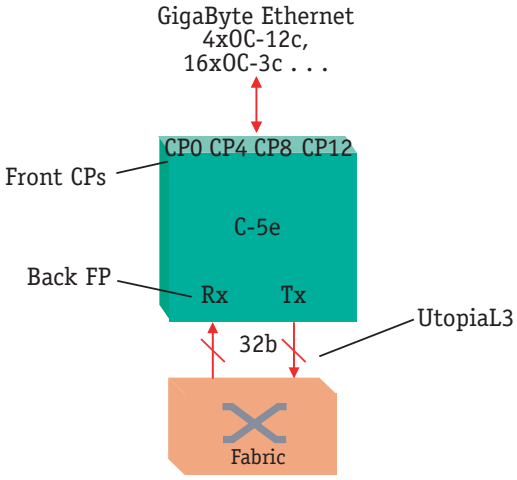
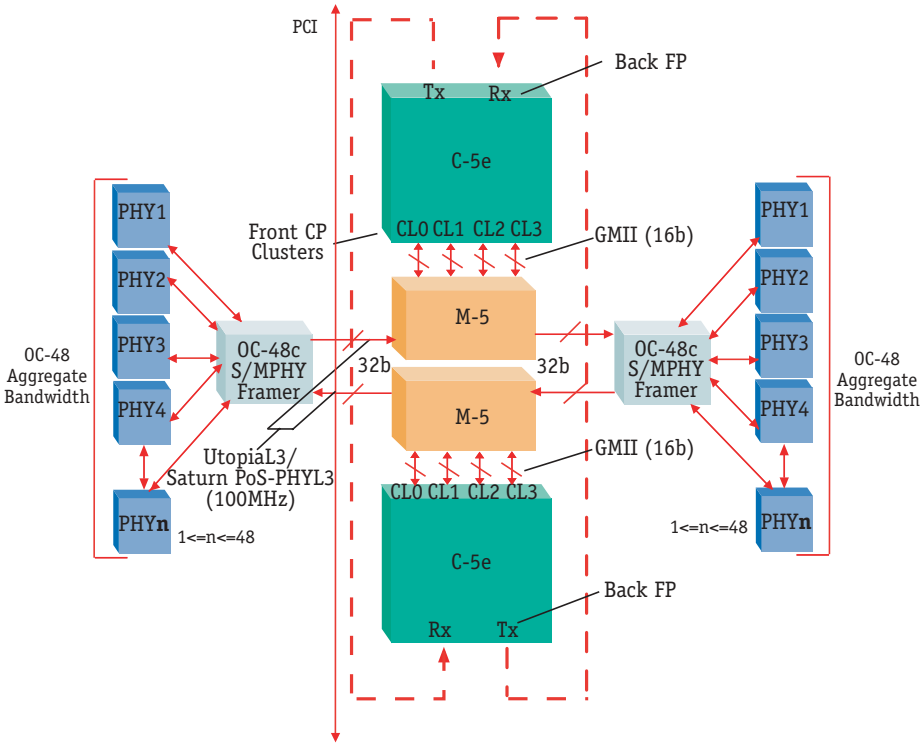
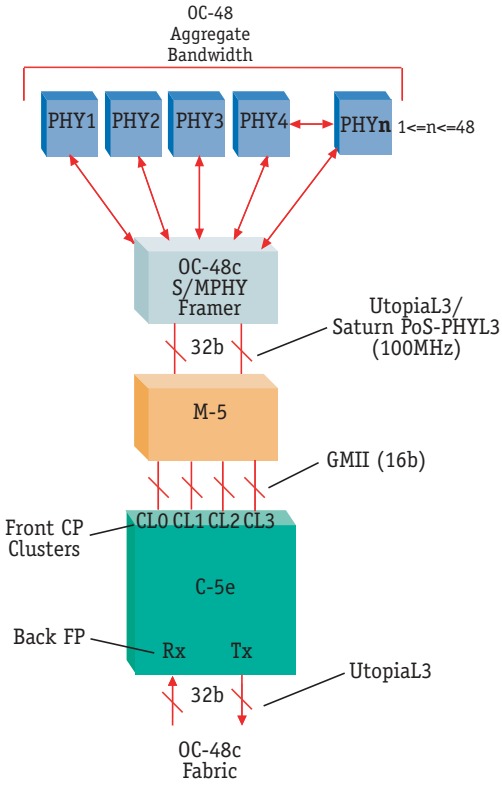


Figure 142 OC-48c to OC-48c Configuration, Using 2 C-5es, and 2 M-5s



Full line rate for OC-48c applications are not realized when both the CP's and the FP are used. However, the full OC-48c line rate (using 64byte packets) is realized when only using the CPs.

Figure 143 OC-48(c) to Switch Fabric Configuration, Using 1 C-5e, and 1 M-5



Full line rate for OC-48c applications are not realized when both the CP's and the FP are used. However, the full OC-48c line rate (using 64byte packets) is realized when only using the CPs.

C-5e Methods for Handling High Speed (OC-48) PDUs

This section provides an overview of the three (3) C-5e methods used to implement higher line speeds. Some are used in conjunction with an external companion device (M-5). Table 262 on page 842 lists the methods and some related aspects as an overview.

Table 262 C-5e Methods and Purpose in Relation to Components, Operation, and Companion Devices

C-5E METHODS	PURPOSE	APPLICABLE C-5E COMPONENT	APPLICABLE OPERATION	USED WITH	DETAILS
Sequence Numbers	Provides ordering inside the C-5e that enables extracting and merging of the associated descriptors with their payloads.	CPs	Enqueue and Dequeue	M-5	Refer to "Sequence Numbers for CPs" on page 148.
Aggregated Queueing	Provides two benefits: allows clustered CPs to share queues, and allows all 16 CPs to share a single queue in order to transmit a single concatenated stream. Both are used in order to reduce latency when operating at higher line speeds.	CPs	Dequeue	N/A	Refer to "Aggregated Queueing for CPs" on page 150.
Speculative Enqueue	Allows a fixed latency for PDU streams, from the start to enqueue. It prevents uneven PDU flows that can cause overruns of PDUs, and large gaps in the streams. This provides a more efficient bandwidth of PDU flow. The enqueues are speculative because at the time they are enqueued, the CRC is *not* calculated.	CPs	Enqueue Only	M-5 Optional	Refer to "Speculative Enqueues for CPs" on page 153.

M-5 Channel Adapter Overview

The Channel Adapter (M-5) is a programmable coprocessor that is used for OC-48c line speed applications. The M-5 accepts both PoS Level3 and Utopia Level3 framer interfaces (refer to Figure 144 on page 843) into the C-5e's 16 clustered CPs, as well as, its FP port at OC-48c wire line speeds. Both SPHY and MPHY framer are supported on the CPs, and the FP supports SPHY framer. OC-1, OC-3, OC-12c stream configurations are also accommodated. Packet-based and cell-based PDU streams are provided for through configuration to furnish the flexibility of different protocols. Table 263 on page 844 provides some specifications on the M-5. For more detail information, refer to the *M-5 Channel Adapter Architecture Guide (part number M5CAARCH-RM/D)*.

Figure 144 M-5 Block Diagram

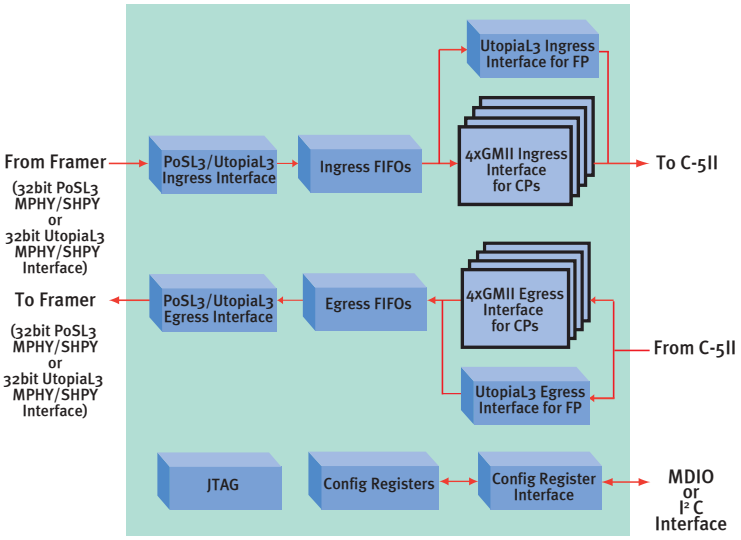


Table 263 M-5 Specifications

Electrical	Power Consumption	2.0W@100MHz Typical (Preliminary)
	Frequency	80MHz Typical, 104MHz Maximum
Processing	Throughput	5Gbps aggregate
Physical	Layout	352 pin Ball Grid Array (BGA) package
Environmental	Operating Temperature	-40° to +85°C
Configuration	Ingress Channels	1 to 48
	Channel Rates Supported	For CPs OC-1 OC-3c OC-12c OC-48 OC-48c For FP OC-48
	PDU Stream Interfaces	For CPs 32bit @ 80 to 104MHz Utopia Level3 with optional parity (fixed-sized) ATM for C-5e CPs (16), with both direct and polling status modes 32bit @ 80 to 104MHz PoS-PHY Level3 with optional parity and packet transfer control (variable-sized) IP for C-5e CPs (16) For FP 32bit @ 80 to 104MHz Utopia Level3 with optional parity (fixed-sized) ATM for C-5e CPs (16), with direct status mode 32bit @ 80 to 104MHz PoS-PHY Level3 with optional parity and packet transfer control (variable-sized) IP for C-5e CPs (16)
	Mapping	OC-1 maps as 3M-5 Ingress channel to 1 C-5e CP channel OC-3c maps as 1 M-5 Ingress channel to 1 C-5e CP channel OC-12c maps as 1 M-5 Ingress channel to 1 C-5e CP Cluster (4 CP channels) OC-48c maps as 1 M-5 Ingress channel to 4 C-5e CP Clusters (16 CP channels) or maps as 1 M-5 Ingress channel to 1 C-5e FP channel
	External PHY Buses	Up to 4 MPHY for C-5e CPs 1 SPHY for C-5e FP

Table 263 M-5 Specifications (continued)

Configuration (continued)	Selected Configurations for 1 Aggregated OC-48 Channel (C-5e CPs only)	48 channels @ OC-1 with total aggregated data rate of OC-48 16 channels @ OC-3c with total aggregated data rate of OC-48 4 channels @ OC-12c with total aggregated data rate of OC-48 3 channels @ OC-1, 3 channels @ OC-3c, 3 channels @ OC-12c with total aggregated data rate of OC-48 12 channels @ OC-1, 12 channels @ OC-3c with total aggregated data rate of OC-48
	PDU Sizes	For ATM 52Byte cells For IP 28Bytes to 9216Bytes packets
	Error Detection	Ingress Errors Parity errors reported via status byte in packet frame Protocol errors reported via PoS or Utopia interfaces FIFO overrun, packet FCS error, packet abort, and illegal packet configuration size, reported via packet frame status byte Egress Errors FCS packet error, optionally can abort packet FCS cell error, optionally can drop cell

GLOSSARY

Aggregate Channel Mode — A mode of the C-5e NP that use the CPs in parallel clusters for wider data stream processing for higher OC speeds like OC-12 or Gigabit Ethernet.

Aggregated Queueing — A method that allows clustered CPs to share queues, and allows all 16 CPs to share a single queue in order to transmit a single concatenated stream. Both are used in order to reduce latency when operating at higher line speeds. Used with CPs.

Allocate — To assign a BTag to a given requester (CP, XP or FP) from the BTags configured in BMU.

Buffer — A partitioned area of the BMU's SDRAM that holds data.

Buffer Tag (BTag) — A partitioned area of the BMU's SDRAM that is used to identify a buffer's location in a pool. It points the buffer.

Buffer Pool — A partitioned area of SDRAM that contains buffers.

Common Switch Interface Level (CSIX-L1) Mode — A FP mode that interfaces to a n industry standard switching fabric as specified by the Network Processing Forum, formerly CSIX/CPIX.

Configure Queue — To send configuration information from the DMEM of a requesting processor (CP, or XP).

Congestion Indicator — The dequeue descriptor message contains a discard priority field used for tagging non-conforming packet descriptors and a congestion indicator field for tagging packet descriptors that experienced congestion.

Constant Bit Rate (CBR) Specifies a fixed bit rate so that data is sent in a steady stream. This is analogous to a leased line.

Content Addressable Memory (CAM) A memory area that contains programmable content that can be searched.

Deallocate To return a BTag from a given requester (CP, XP or FP) back to the BMU.

Descriptor A specific type of data used for traffic forwarding.

Descriptor Buffer A specific type of data used for traffic forwarding stored in a buffer in the QMU's external SRAM.

Discard Priority The dequeue descriptor message contains a discard priority field used for tagging non-conforming packet descriptors.

Discard Reason The dequeue descriptor message defined by the traffic management interface (TMI) contains a type field that specifies the discard reason for packet descriptors that have been selected for discard.

Dequeue To read descriptor data from a queue in QMU's SRAM into the DMEM of the requesting processor (CP, or XP).

Dynamic Descriptor Buffer Pool A partitioned area of the QMU's external SRAM that contains descriptor buffers.

End of Message The enqueue message received from the traffic management interface that contains an end of message indicator bit used to identify the last packet descriptor of a larger message.

FP Payload A portion of segment, after the header, that is taken from or stored into a BMU buffer.

FP Protocol Data Unit (PDU) Data to be transmitted from or received into a BMU buffer.

FP PDU ID An identifier that allows a receiving FP to associate segments for reassembly. This ID need only be unique in the FPRx while the PDU is being reassembled; that is, a given PDU ID can be reused by a subsequent PDU. Typically, a "flow" of PDUs, transmitted from a single queue of a Network Processor, might all use the same PDU ID.

FP Segment A basic package of FP data and header information that is transferred on fabric interface, usually a fixed size.

FP Segment Header The beginning portion of a segment containing information used to route the segment through the fabric and allow the receiving FP to reassemble the PDU from segments.

FP Segment Type Indicates the portion of the PDU that the segment carries. There are four types; first, middle, last, and only (first and last).

FP Scope A set of internal hardware resources. The FPTx has 8 scopes. The FPRx can be configured for either 8 or 16 scopes.

Fixed-Use Control Blocks A group of programming short-cuts dedicated to make data moves to/from SDRAM, the BMU, or the QMU invoked by using WrCB_ and RdCB_.

Link-List Tracks the free descriptor buffers, used descriptor buffers for queueing, and the location of the data (descriptor data) in queues in the SRAM.

Literals A value written exactly as it's meant to be interpreted.

Multicast Enqueue To write a single descriptor's data into multiple queues in the QMU's SRAM from the DMEM of the requesting processor (CP, or XP).

Multicast Group A bit in the traffic management interface enqueue message is used to indicate that an enqueue operation is destined for a multicast group. The enqueue processor enqueues the descriptor from the enqueue message to each destination traffic queue in a multicast group or discards it.

Multicast Replication Identifier When multiple copies of an enqueued descriptor are multicast elaborated to multiple traffic queues destined for the same virtual output port, the receiver of the dequeued descriptor on the traffic management interface can require a unique identifier to distinguish one dequeued descriptor from another. In this situation, a multicast replication identifier is specified in each destination traffic queue's parameter block. The destination traffic queue's multicast replication identifier is placed in each of its traffic management interface dequeue messages for use by the receiver.

Multi-Use Control Blocks A group of registers that can be programmed to make data moves to/from SDRAM, the BMU, or the QMU. (WrCB0_Sys_Addr, WrCB0_Ctl, WrCB0_DMA_Addr, WrCB0_SDP_Addr; RxCB0_Sys_Addr, RxCB0_Ctl, RxCB0_DMA_Addr, RxCB0_SDP_Addr; RdCB0_Sys_Addr, RdCB0_Ctl, RdCB0_DMA_Addr, RdCB0_SDP_Addr; and TxCB0_Sys_Addr, TxCB0_Ctl, TxCB0_DMA_Addr, TxCB0_SDP_Addr).

Multi-Use Counter When a BTag is assigned to more than one target (CP, XP or FP), a counter is needed to track the multi-use BTag.

Packet Byte Length An enqueued packet's Byte count comes from the packet Byte length field of its associated enqueue message, which is received through the traffic management interface.

Packet Byte Remainder The dequeue messages will contain a remainder field which indicates how much of the encapsulated packet remained before this segment was transmitted.

Pipeline Channel Mode A mode of the C-5e NP that links the individual CPs together for processing a single data stream to achieve higher processing speeds.

PowerX Mode (CSIXL-0) A FP mode that interfaces to a Power X TeraChannel® switch fabric product.

PRIZMA Mode A FP mode that interfaces to a IBM PRIZMA-E™ or PRIZMA-EP™ switch fabric product.

Queue A FIFO that contains descriptor data.

Queue Level A index to the queue number with a port to a processor (CP, or XP). It's purpose is to copy a single descriptor to multiple queues mapped to multiple processors.

Queue Status To read a single queue's length and weight from the QMU into the DMEM of the requesting processor (CP, or XP).

Recirculation A method used to pipeline your C-5e NP. To configure the C-5e Channel Processors (CPs) RxSDP and TxSDP so that the output from the TxSDP is routed to the input of its corresponding RxSDP.

Reference Count An initial count that is entered by hardware into the 8bit counter that is used to track multi-use BTags.

Sequence Numbers A method that provides ordering inside the C-5e NP that enables extracting and merging of the associated descriptors with their payloads. Used with CPs and/or FPTx.

Single Channel Mode A mode of the C-5e NP where the CPs operate independently of each other at full duplex and can support for example, OC-3.

Speculative Enqueue A method that allows a fixed latency for PDU streams, from the start to enqueue. It prevents uneven PDU flows that can cause overruns of PDU's, and large gaps in the stream. Thus, it provides a more efficient bandwidth of PDU flow. The enqueues are speculative because at the time they are enqueued, the CRC is not calculated. Used with CPs.

Traffic Queue Identifier Traffic queue identifiers are 18bits wide and a link memory word holds 17bits. The following restriction is made to reduce the number of bits of storage required for traffic queue links to 17bits:

Type The dequeue descriptor message defined by the traffic management interface (TMI) contains a type field that specifies the discard reason for packet descriptors that have been selected for discard.

Unicast Enqueue To write descriptor data into a queue in the QMU's SRAM from the DMEM of the requesting processor (CP, or XP).

User-Defined Inter-processor Message — Small fixed-sized (12, 16, 24, or 32Bytes) data structures that contain user defined information. Generally, inter-processor messages are used to orchestrate control plane activities such as flow control, statistics gathering, or table maintenance.

Variable Bit Rate (VBR) — A specified throughput capacity but data is not sent evenly. This is a popular choice for voice and video-conference data.

Virtual Queuing — A method that allows one (1) transmit queue to be spread out over multiple output ports (CPs and/or FPTx).

INDEX

D

E

F

G

H

I

M

N

O

P

Q

R

S

T

V

W

X

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

freescale™
semiconductor