Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!

Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832
Email & Skype: info@chipsmall.com Web: www.chipsmall.com
Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China

RapidIO 2.1 Serial Endpoint IP Core User's Guide

June 2011
IPUG84_01.3

Table of Contents

The RapidIO 2.1 interconnect specification is an open standard developed by the RapidIO Trade Association. It has been designed to offer a reliable high-performance, packet-switched, interconnect for the embedded industry providing scalable chip-to-chip and board-to-board communication for microprocessors, digital signal processors (DSPs), network processors, and memories. Rapid I/O offers the generic memory and device addressing concepts of a processor bus at the user level where at the Serial RapidIO transaction level, processing is managed completely by the protocol and hidden from the user's application. The LatticeECP3™ family of low-cost, low-power FPGA devices provides a very economical platform for developing Serial RapidIO interconnect strategies.

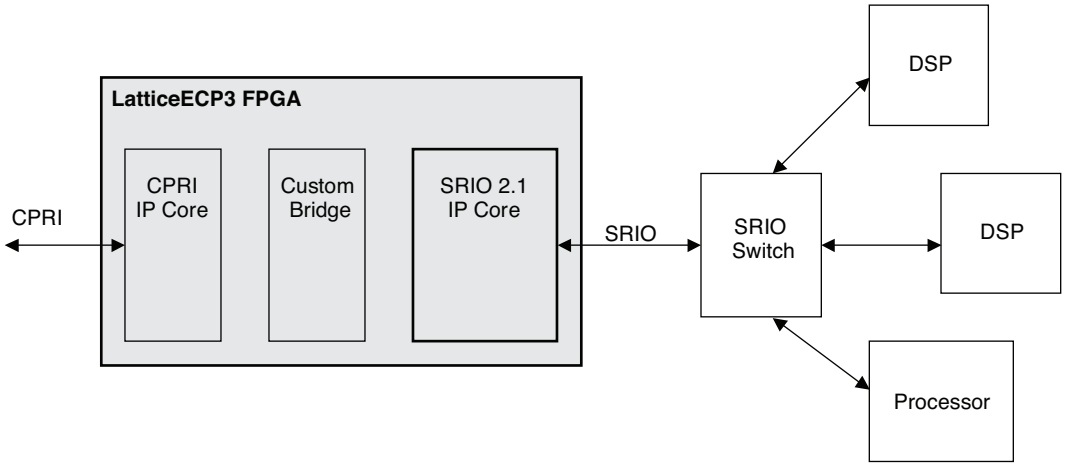
The Lattice Serial RapidIO (SRIO) Endpoint IP core is fully compliant to the following RapidIO version 2.1 Interconnect Specifications:

- Part 1: Input/Output Logical Specification
- Part 2: Message Passing Logical Specification
- Part 3: Common Transport Specification
- Part 6: LP-Serial Physical Layer Specification
- Part 8: Error Management Extensions Specification

Because the core is completely compliant to these specifications, this user's guide relies heavily upon them to aid in the explanation of core behavior and functional content. Required RapidIO behavior and functionality explained in these specifications is not replicated in this user's guide. It is recommended that users have access to, and a working level knowledge of these specifications before using the SRIO IP core.

Figure 1-1 shows an example of a typical system application.

Figure 1-1. System Application



Quick Facts

Table 1-1 gives quick facts about the SRIO IP core.

Table 1-1. Quick Facts

Core Requirements	FPGA Families Supported	SRIO IP Configuration			
		1X Mode / 2X Mode / 4X Mode			
		LatticeECP3			
Resources Utilization	Target Device	LFE3-35EA	LFE3-70EA	LFE3-95EA	LFE3-150EA
	Data Path Width	64			
	LUTs	15,749			
	sysMEM™ EBRs	16			
	Registers	10,199			
Design Tool Support	Lattice Implementation	Lattice Diamond™ 1.2			
	Synthesis	Synplicity Simplify Pro® for Lattice E-2010.09L-SP2			
	Simulation	Aldec® Active-HDL® 8.2 SP1 Lattice Edition			
		Mentor Graphics® ModelSim® 6.5B			

Features

General Features

- Compliant with RapidIO Trade Association, RapidIO Interconnect Specification, Revision 2.1, August 2009
- Includes compliance with Interconnect Specification Part 8: Error Management Extensions
- Supports all capability (CARs) and configuration and status registers (CSRs) for all three RapidIO layers
- Supports 8-bit or 16-bit device IDs
- Supports incoming and outgoing multi-cast events
- Transmitter controlled flow control
- Generic 32-bit processor interface for local register access
- Generic 32-bit interface supporting user definable register expansion accessible locally or remotely via the SRIO link.

Physical Layer Features

- 1x/2x/4x serial lane support with integrated transceivers
- Serial data rates supported: 1.25, 2.5, 3.125 GBaud
- Receive/transmit packet buffering, flow control, error detection, packet assembly and delineation
- Automatic freeing of resources used by acknowledged packets
- Automatic retransmission of retried packets
- Autonomous error recovery
- Full control over integrated transceiver parameters via SERDES SCI interface

Transport Layer Features

- Supports up to three logical layer user ports.
- Scheduling of transmission from logical layer ports based on two different selectable priority schemes - fixed or rotating.

Logical Layer Features

- Built-in maintenance request decoder/response encoder supporting SRIO link access of local CAR and CSR registers.
- Autonomous Doorbell generation on event.
- Built-in logical layer packet source/sink capability referred to as the Soft Packet Interface (SPI) accessed via a local processor control interface (AMI).

Errata

The logN_rlnk_dst_dsc_n signal may not function as expected with small sized packets. It is not recommended to use this signal in your design.

Supported Transactions

All legal SRIO logical layer transaction types are supported via the Logical Layer user interface ports and though the Soft Packet Interface.

What Is Not Supported

The core does not support the following optional features:

- Software Assisted Error Recovery
- Critical Request Flow (CRF)
- Baud Rate Discovery
- Lane n Status 1 to 7 CSRs
- Enable Inactive Lanes
- Port Initialization Test Mode
- Virtual Channels (VCs) 1 - 8

Conventions

The nomenclature used in this document is based on the Verilog language. This includes radix indications, and logical operators.

Data Ordering and Data Types

Data ordering is per the RapidIO specification, and also follows PowerPC conventions. This includes:

- Byte ordering is big-endian
- The most significant bit within data types is numbered 0

Table 1-2. Data Types

Name	Size
byte or octet	8 bits
hword – half word	16 bits
word	32 bits
dword – double word	64 bits

Signal Names

- Signal names that end with “_n” or “_N” are active low.

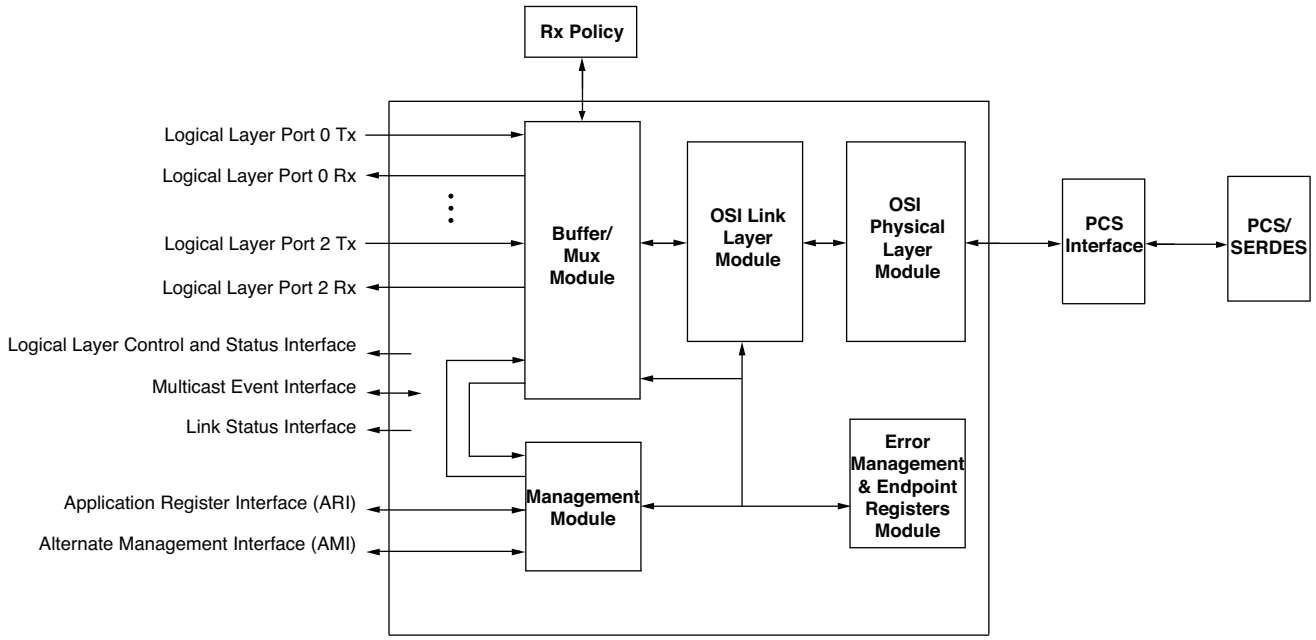
This section provides detailed descriptions of functionality that interfaces with user logic.

Overview

The SRIO IP core implements the functionality described in the RapidIO 2.1 specification for the Physical, Transport, Error Management, and Logical layers. Figure 2-1 shows the major functional blocks that make up the core as well as how it interfaces to the high-speed transceivers and user-defined logical layer functions. Built-in logical layer functionality is provided for maintenance transaction support and for a low bandwidth control plane based packet source/sink feature referred to as the Soft Packet Interface (SPI). Three bi-directional logical layer interface ports are available for user defined functions such as Initiator, Target, and Doorbell functions. The core consists of the following major functional blocks:

- **OSI Physical Layer Module** – Implements RapidIO physical layer functions that are defined in the OSI physical layer.
- **OSI Link Layer Module** – Implements RapidIO physical layer functions that are defined in the OSI link layer.
- **Buffer Mux Module** – Implements buffering for packet transmission and reception. It also manages the multiplexing and de-multiplexing of traffic to and from the Logical Layer ports.
- **Management Module** – Implements the functions needed to access Control and Status Registers (CSRs) both locally or remotely and provides an external interface for user defined register expansion. It also includes a subsystem used to monitor interface events.
- **Error Management and Endpoint Registers Module** – Implements logical layer error management CSRs as well as various endpoint CSRs.

Figure 2-1. LP-Serial Endpoint Core Architecture



User Interface

The core interacts with the user application through eight types of interfaces:

- **Logical Layer Ports** – The logical layer ports are used to connect user implemented logical layer functions to the core. Each Logical layer port consists of a 64-bit receive and transmit interface, an error reporting interface supporting error management extensions, and a User Event interface.
- **Alternate Management Interface** – This interface is used to access local CSRs within the core and user defined registers outside the core via the ARI described below. It is also used to access the Soft-Packet Interface (SPI) FIFO feature that allows creation and termination of RapidIO packets though software.
- **Application Register Interface** – This interface is used by the core to access user defined CSRs. This interface provides an efficient way for users to add additional registers to their design and can be accessed through maintenance commands as well from the far-end.
- **Logical Layer Common Control and Status Interface** – This interface presents information contained in the logical layer control CSRs contained in the core, transmit flow control information, and general logical port interface status.
- **Multicast Event Interface** – This interface provides support for generating RapidIO control symbols containing the multicast event function code. It also indicates when a control symbol has been received that contains the multicast event function code.
- **Link Status Interface** – The status interface includes both generic information about Tx and Rx activity and RapidIO port state as well as interface training information specific to LP-Serial ports.
- **Receive Policy Interface** – This interface controls the de-multiplexing of request and response packets received from the RapidIO link to Logical link and Management Rx ports on the Buffer Mux Module.
- **Transceiver Interface** – The transceiver interface connects the SRIO IP core to the PCS interface logic that interfaces with the built-in SERDES.

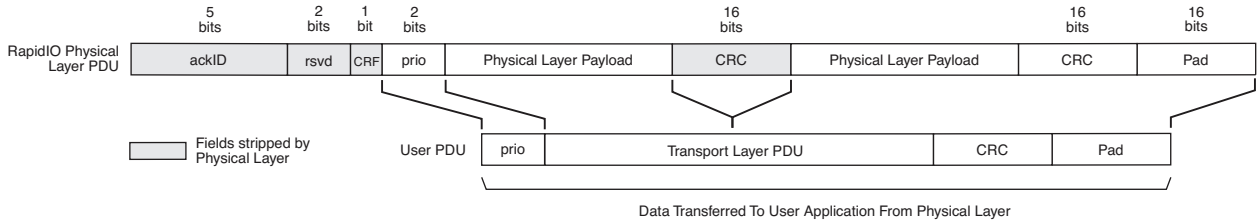
In addition to the functions included in the core proper, there is a user customizable Receive Policy module that is delivered as Verilog source code. A Physical Coding Sub-layer interface module is also provided in Verilog format and provides the interface between the core and the device built-in SERDES functions.

Packet Transmission and Reception

The SRIO core presents a reliable transport to logical layer functions connected to the logical layer ports. This means that link retries due to error recovery and congestion are handled by the physical layer and are transparent to logical layer functions.

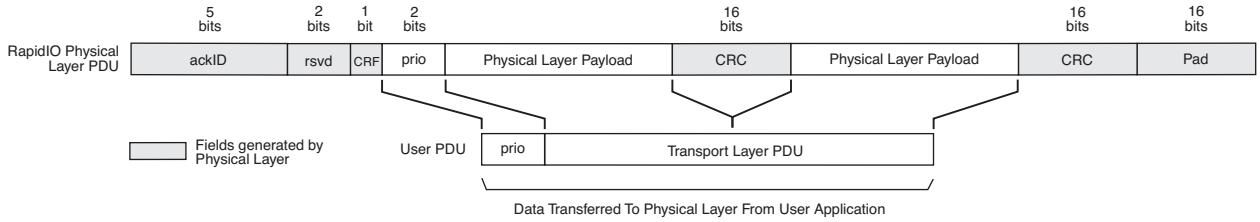
Received packets have the first eight bits of the physical layer overhead stripped along with the mid-span CRC, if present. The final CRC and possible pad are not stripped and can be discarded by the logical layer. Figure 2-2 illustrates the handling of received packets.

Figure 2-2. Receive Packet Handling



For transmitted packets the physical layer core adds the first octet of the physical layer overhead along with the mid-span CRC, final CRC, and pad as needed. Figure 2-3 illustrates the handling of transmitted packets.

Figure 2-3. Transmit Packet Handling



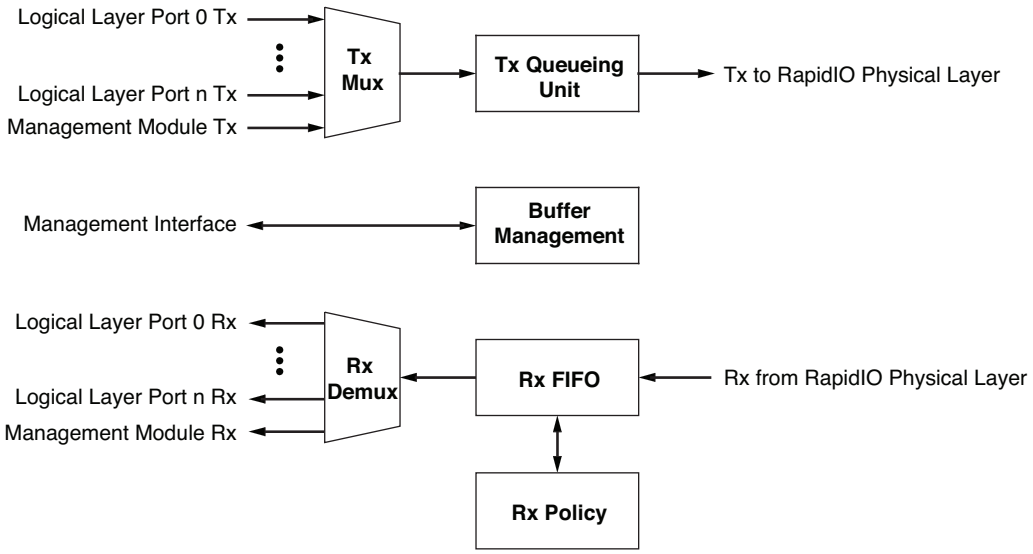
Module Descriptions

Buffer Mux Module

The Buffer/Mux module contains the following functional blocks:

- **Tx Queuing Unit** – Buffers outgoing RapidIO Logical layer packets for transmission, and handles retransmission in response to retry and error indications from the far end of the link. The maximum number of maximum length RapidIO packets that can be buffered is fixed in this version of the IP core to 14.
- **Tx Mux** – Multiplexes packets from the external logical layer functions, and the internal Management Module to the Tx FIFO.
- **Rx FIFO** – Buffers incoming RapidIO Logical layer packets. This buffer not only provides elasticity, but it also filters out packets with receive errors so they are not visible to logical layer functions. The FIFO can store up to seven, maximum length, RapidIO packets.
- **Rx Demux** – De-multiplexes incoming packets to one of the Logical layer Rx ports or to the internal Management Module Rx port.
- **Rx Policy** – Controls the operation of the Rx Demux and determines whether a packet will be stored in the Rx FIFO based on user defined criteria. This block is implemented outside the core itself. A reference implementation of an Rx Policy in Verilog source code form is included with the core.

Figure 2-4. Buffer Mux Module Block Diagram

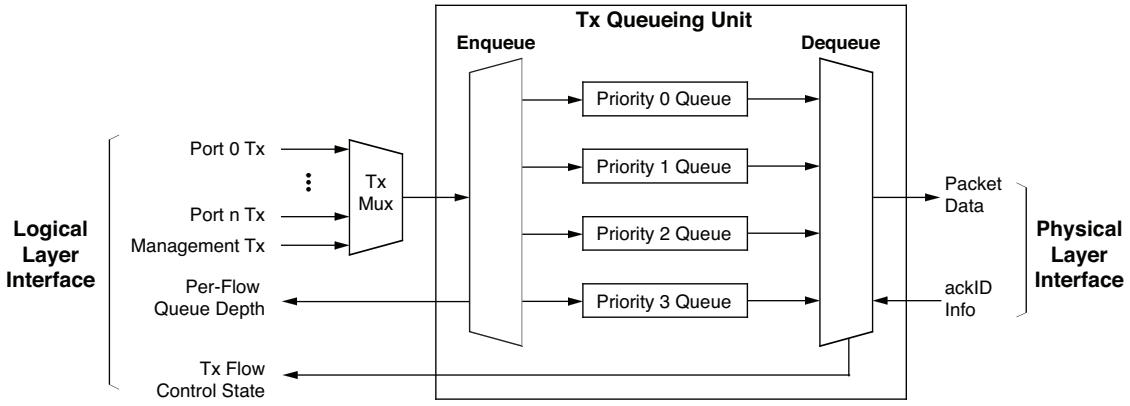


Packet Transmission

The transmission logic multiplexes logical layer packets generated by logical layer functions and queues them for transmission. This buffering eliminates the need for the logical layer functions to resend packets in the case of link errors or retries due to link congestion. There is no cut-through forwarding of packets. That is to say all packets pre-

sented at the logical layer Tx ports are completely loaded into the Tx queues before they are eligible for transmission.

Figure 2-5. Buffer Mux Tx Datapath Functional Block Diagram



Transmit Multiplexer – Tx Mux

The transmit multiplexer selects the next packet for en-queue from the packets that are currently presented on the Logical Port Transmit Interfaces as well as the management interface Transmit link. In cases where multiple packets are queued on these interfaces for transmission, the Transmit Multiplexer selects the next packet for en-queue using the algorithm shown in Figure 2-6. While the packet that was selected is being en-queued, traffic presented at the other logical layer ports is stalled. This serializes the en-queue of packets from the logical layer ports.

Arbitration is affected by the RapidIO priority of traffic queued at the multiplexer ports, and the relative priority of the Transmit Multiplexer ports themselves. Arbitration effectively occurs in two phases with the first being RapidIO priority arbitration, and the second port priority arbitration.

RapidIO Priority Arbitration

The first thing considered in the arbitration algorithm is the RapidIO priority of queued traffic. This priority information is taken from log_tlnk_d[0:1] bits from each port. Packets are forwarded in strict RapidIO priority order with RapidIO priority 3 being the highest. The highest RapidIO priority of packets enqueued at the start of an arbitration cycle is referred to as the winning RapidIO priority. If there is only one packet queued at the winning RapidIO priority then that packet will be enqueued.

Port Priority Arbitration

If there are multiple packets queued at the winning RapidIO priority, then the packet selected for forwarding is based on the relative port priorities of ports with packets enqueued at the winning RapidIO priority. There are two schemes supported (fixed and rotating) for port arbitration, with the scheme that is used configured via GUI selection during core generation:

- **Fixed Priority** – Each port has a fixed relative priority with port 0 having the highest priority and the management port having the lowest.
- **Rotating Priority** – Each port has a fixed relative priority, but the absolute priorities may update after an arbitration cycle. This is described in more detail below.

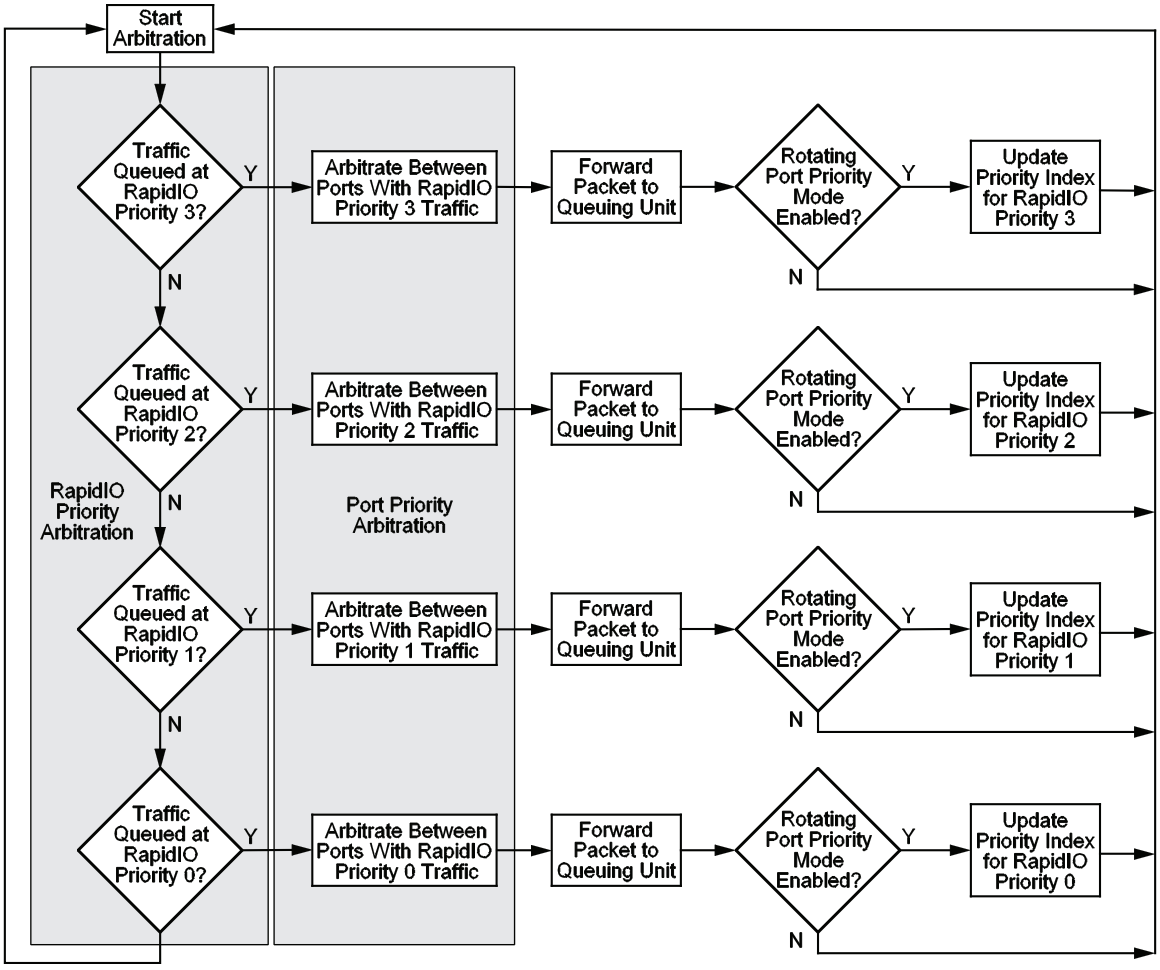
When configured for rotating priority the relative priorities of the ports remain constant, but the port that won the last port arbitration contest is assigned the lowest priority for the next contest. This has the effect of rotating the port priority. An internal variable called the priority index determines which port is assigned the highest port priority during arbitration. After each arbitration contest this value is updated to equal the winning port number plus one. A separate priority index is kept for each RapidIO priority level.

Table 2-1 shows an example of how the priority index would be updated and port priorities change over several contests at the same RapidIO priority level.

Table 2-1. Rotating Port Priority Example

	Arc	Contest 1	Contest 2	Contest 3	Contest 4
Port Priority	Highest	Logical Port 0	Logical Port 3	Management Port	Logical Port 2
		Logical Port 1	Management Port	Logical Port 0	Logical Port 3
		Logical Port 2	Logical Port 0	Logical Port 1	Management Port
		Logical Port 3	Logical Port 1	Logical Port 2	Logical Port 0
	Lowest	Management Port	Logical Port 2	Logical Port 3	Logical Port 1
Priority Index		Port 0	Port 3	Management Port	Logical Port 2
Winning Port		Port 2	Port 3	Port 1	

Figure 2-6. Transmit Multiplexer Arbitration Algorithm



Queuing Unit

The Queuing Unit manages the buffering of packets for transmission. Packets are stored in the Tx buffer memory implemented inside the core. This memory is divided into fixed size packet buffers of 280 bytes each. Each of these buffers is capable of storing a maximum sized RapidIO packet. The total number of buffers available for storage across all priority queues (see below) is equal to the transmit buffer memory size in bytes mod 280. For the initial design, the buffer memory is fixed at 4096 bytes and so a total of 14 buffers are available.

Packets are stored in one of four logical queues, one for each of the four RapidIO priority levels. Each of these queues can contain up to the fourteen packet maximum for the initial IP core release. It is up to user logical layer functions to control the maximum number of packets that should be loaded at each priority level so that there are buffers available when they are needed for other priorities. It may not make sense, for example, to use up all 14 buffers at the low priority queue level if other priority traffic must also be supported. In addition, at least one of the fourteen buffers should always be left available for response packets associated with the Management unit (maintenance requests) in order to avoid dead-lock conditions. This procedure is referred to as the ingress queue threshold policy.

The ingress queue policy should also take into consideration the fill levels of buffers at the far-end of the link when Transmitter Flow Control is enabled. In order support the implementation of the ingress queue threshold policy the core provides the following information to the user logical layer functions via the Logical Layer Common Control and Status Interface:

- The current depth of each of the local priority queues as well as a total depth. This information is presented on the tx_flow_depth vector.
- The current state of the Tx flow control state machine. This information is presented on the tx_flow_ctrl_state vector and provides the fill levels of the buffers at the far-end of the link when Transmitter Flow Control is being used. If Transmitter Flow Control is not being used, this vector can be ignored.

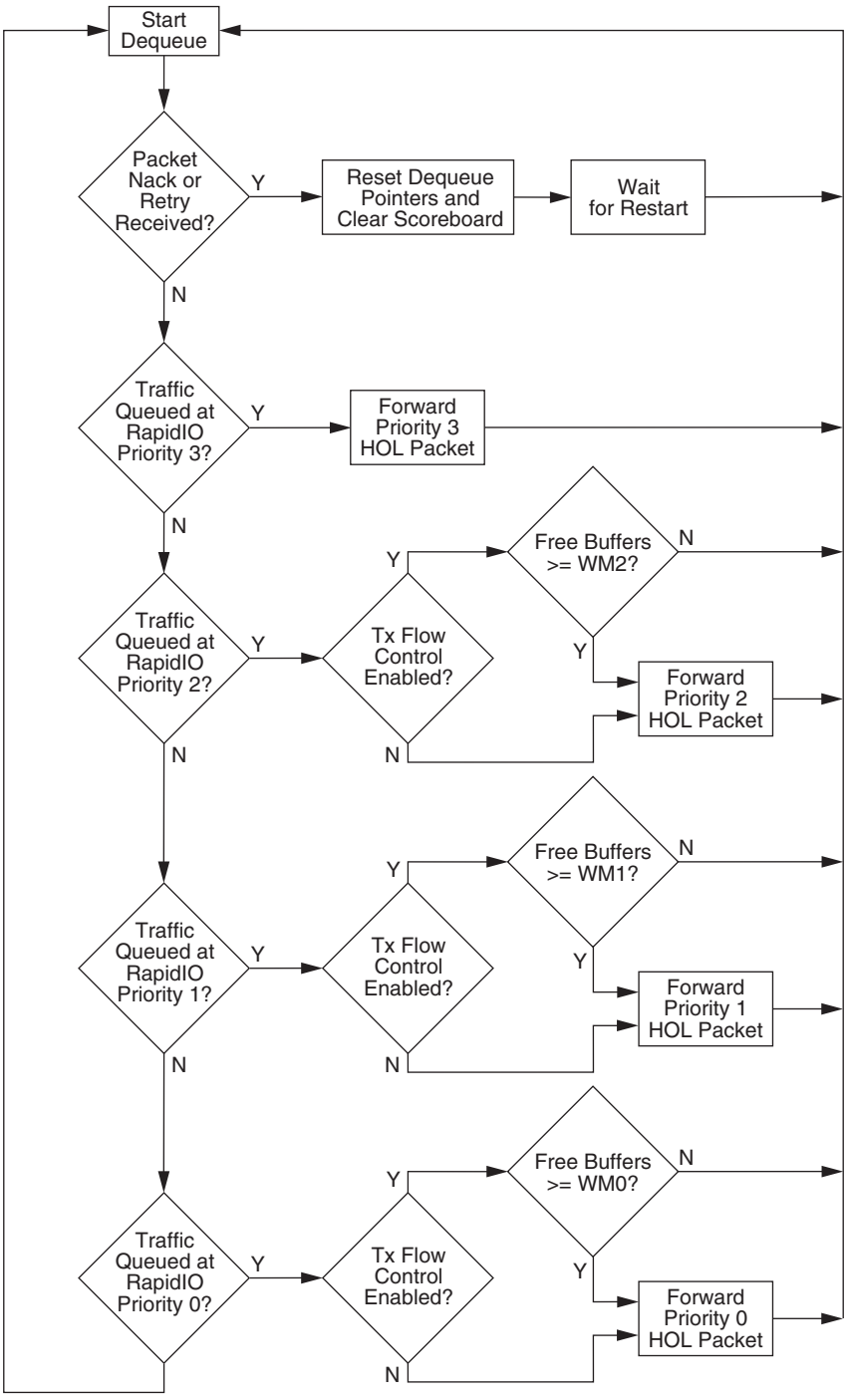
Enqueue Policy
Packets delivered by the Tx Mux to the Queuing Unit are queued based on their RapidIO priority. This priority information is captured from log_tlnk_d[0:1] on the first beat of a packet transfer. Packets are not accepted for enqueue when all packet buffers in the queuing unit are currently filled.

Dequeue Policy
The Dequeue Policy selects queued packets for forwarding over the link. Factors that affect which packet is selected include:

- Packet priorities
- Tx flow control, if enabled
- Retransmission state based on errors or retries

The algorithm used by the Dequeue Policy for packet selection is shown in Figure 2-7.

Figure 2-7. Dequeue Policy Algorithm



Packet Reception

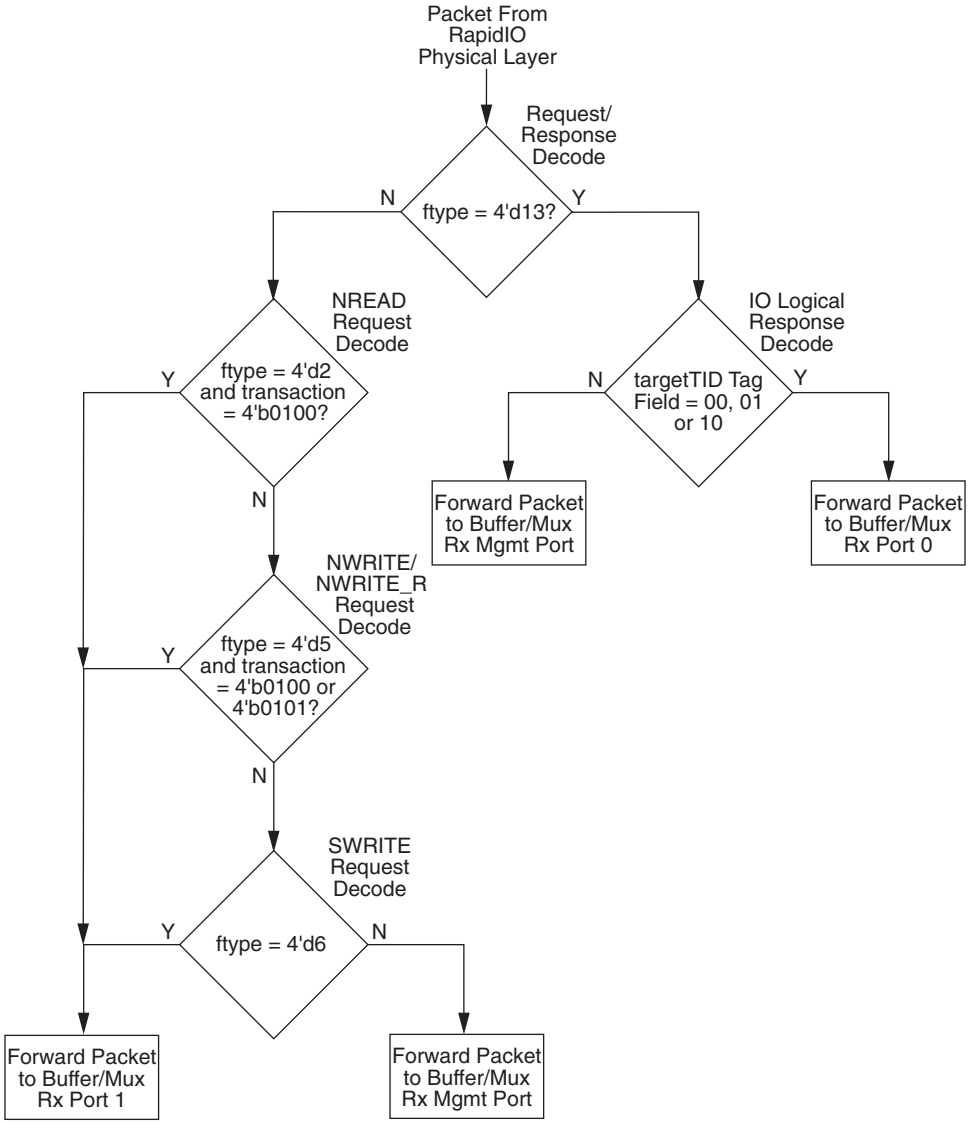
Rx Policy

This section describes how the Receive Policy is used to de-multiplex received packets to the Logical Layer ports and Management unit. The policy description shown in Figure 2-8 assumes the following configuration of logical layer functions:

- I/O Logical Layer initiator block connected to Logical Layer Port 0.
- I/O Logical Layer target block connected to Logical Layer Port 1.
- No logical layer functions connected to Logical Layer Port 2.

The reference receive policy does not force any retries of received packets. The receive policy that is provided as a reference design with the IP core design package is slightly different than the policy described below in that it also supports a doorbell unit connected to Logical Layer port 2. This reference policy can be modified to suit the arrangement of user logical layer functions connected to the core.

Figure 2-8. Reference Policy Demux Decision Tree

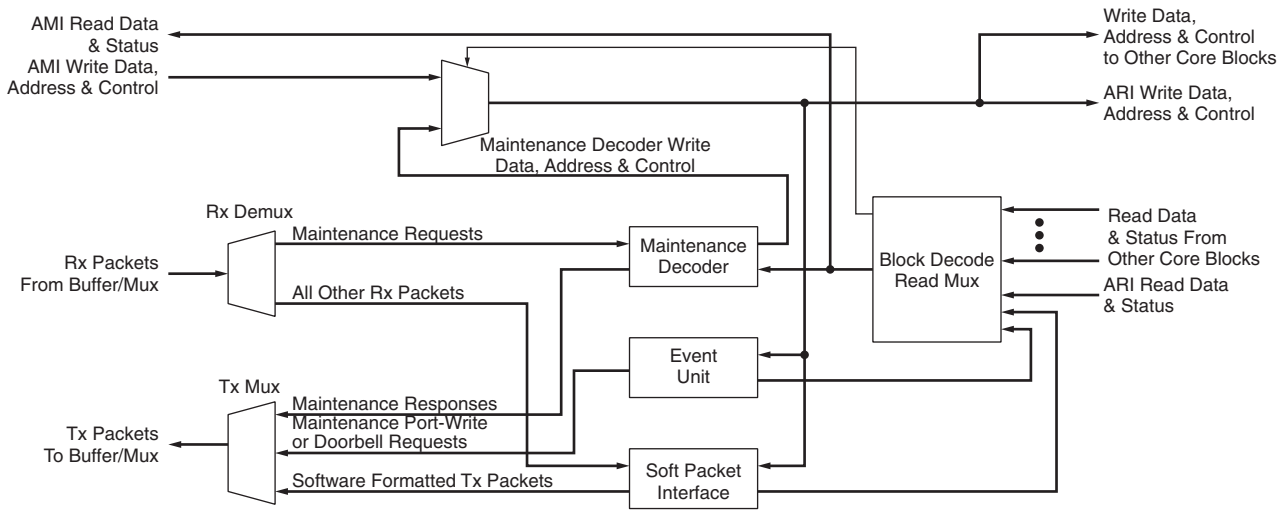


Management Module

The Management Module Implements the infrastructure needed to manage an endpoint remotely using RapidIO maintenance transactions or locally through the AMI processor interface. It also contains facilities for remote status reporting and software packet generation and decoding. The Management Module consists of the following sub-modules:

- **Maintenance Decoder** – Decoding and responding to maintenance transactions received from the RapidIO link.
- **Event Unit** – Implements logging and reporting of system events.
- **Block Decode / Read Mux** – Block level address decoding for modules. Arbitrating access to CSRs between the maintenance decoder and the Alternate Management Interface (AMI).
- **Soft Packet Interface (SPI)** – This module provides a means of generating and decoding RapidIO packets under software control.

Figure 2-9. Management Module Block Diagram



Maintenance Decoder

The maintenance decoder gives remote endpoints access to the control and status registers in the core. It does this by decoding incoming maintenance packets and generates read or write cycles on the management interface. In the case of maintenance read transactions the decoder formats response packets with the read data.

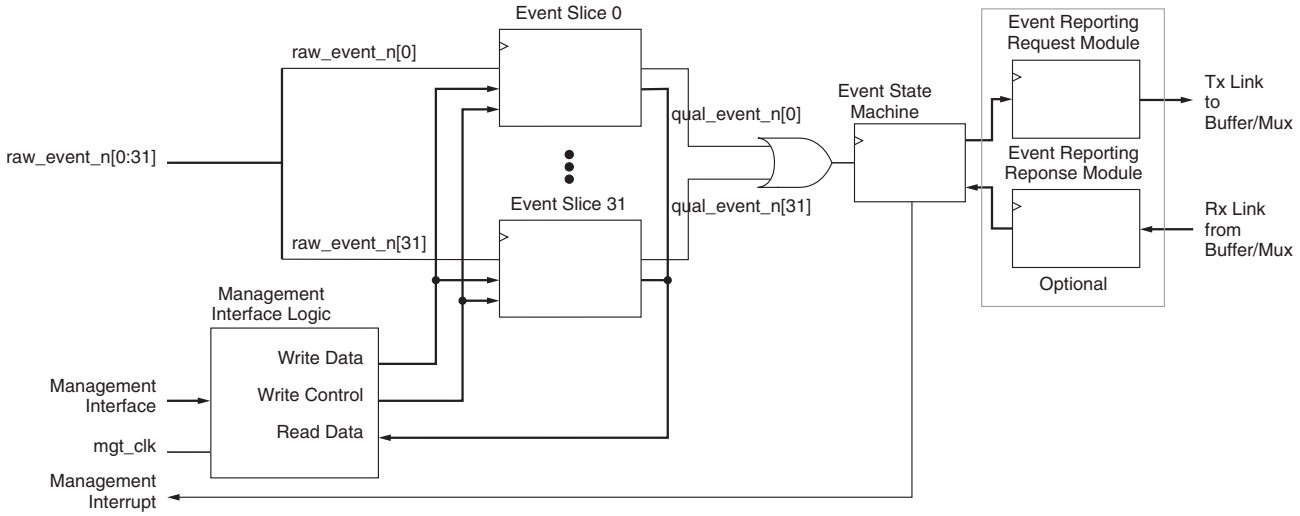
Event Unit

The Event unit logs and reports the various system level events that can occur within the endpoint. These events include:

- **Error Management Events** – These include errors at the physical, transport, and logical layer as defined in the Error Management Extensions specification.
- **Soft Packet Interface Events** – These include reception of a packet, as well as the completion of packet transmission.
- **User-Defined Events** – Each logical layer interface includes an event indication that can be used to flag events that are significant to the user application.

A complete list of events, both standard and optional, is shown in Table 2-2. Event logging consists of capturing the assertion of events on the event bus in the Event Status Register, and in some cases the Event Overflow Register. Depending on whether the event has been masked it will be reported to the system locally and remotely as described below. The structure of the Event Unit is shown in Figure 2-10.

Figure 2-10. Event Unit Block Diagram

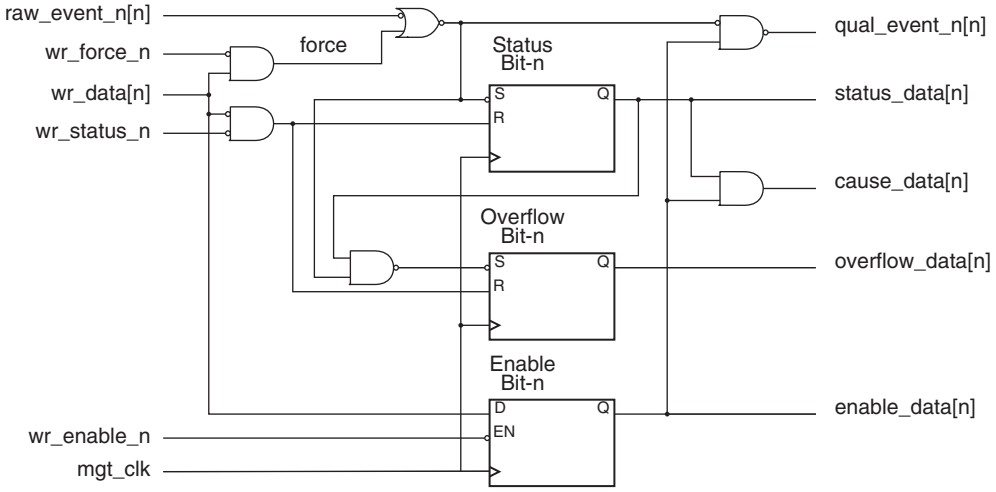


Event Logging

The logging of events is managed by five CSRs located in the event unit. Each bit in each of the CSRs corresponds to one of the system events.

- **Status Register** – The Status CSR shows which events have been asserted. This is a read/write register, and events can be cleared by writing to this register.
- **Overflow Register** – The Overflow CSR shows which events have occurred more than once since the corresponding bit was set in the status register.
- **Mask Register** – The Mask CSR controls whether the assertion of an event results in its being reported by one of the standard event reporting mechanisms.
- **Cause Register** – The Cause CSR shows which events caused an event to be reported. Essentially this CSR shows the state of the Status CSR masked by the state of the Mask CSR.
- **Force Register** – The Force CSR is used to force the assertion of one or more events, and is typically used for testing.

Figure 2-11. Event Slice Logic Block Diagram



Event Reporting

There are two methods of event reporting:

- Assertion of the interrupt signal on the Alternate Management Interface.
- Transmission of a Doorbell message. The doorbell message will contain the event number which corresponds to the active event listed in Table 2-2. The lower five bits of the information field is used to carry the event number. Care should be taken if using Doorbells in the system for other purposes to reserve this portion of the doorbell information field. In the initial version of the core, there is no way to disable generation of the doorbell message. This feature will be added in a future release of the core.

  Transmission of a Port-Write maintenance packet as described in the Error Management Extensions specification is also not available in the initial version of the core. This feature will also be made available as an option in a future release.

Table 2-2. Endpoint Layer Core Events

Event Number	Description
0	Physical Layer Error Management Event
1	Logical/Transport Layer Error Management Event
2	Soft Packet Interface Rx Event
3	Soft Packet Interface Tx Event
4-15	Reserved
16	Logical Port 0, User Event 0
17	Logical Port 0, User Event 1
18	Logical Port 0, User Event 2
19	Logical Port 0, User Event 3
20	Logical Port 1, User Event 0
21	Logical Port 1, User Event 1
22	Logical Port 1, User Event 2
23	Logical Port 1, User Event 3
24	Logical Port 2, User Event 0
25	Logical Port 2, User Event 1
26	Logical Port 2, User Event 2
27	Logical Port 2, User Event 3
28	Logical Port 3, User Event 0
29	Logical Port 3, User Event 1
30	Logical Port 3, User Event 2
31	Logical Port 3, User Event 3

Soft Packet Interface Module

The Soft Packet Interface (SPI) provides a means of sending and receiving RapidIO packets under software control. Formatting and decoding of these packets is performed by system software. The format of packet data transmitted and received is the same as that used on the Logical Layer port interfaces used to connected logical layer functions, and is shown in Figure 2-2 and Figure 2-3.

Specific examples of RapidIO Logical Layer packets can be seen in “Packet Formats and Descriptions” on page 25. Transmission and reception of packets by software is done using the CSRs summarized in Table 2-3. The Soft Packet Interface (SPI) supports packet transfers based on either polled or interrupt driven schemes.

Table 2-3. Soft Packet Interface CSR Summary

CSR Name	Summary Description	Details in Section
SP_TX_CTRL	Soft Packet Transmit Control	"Soft Packet FIFO Transmit Control (IR_SP_TX_CTRL) CSR" on page 75
SP_TX_STAT	Soft Packet Transmit Status	"Soft Packet FIFO Transmit Status (IR_SP_TX_STAT) CSR" on page 76
SP_TX_DATA	Soft Packet Transmit Data	"Soft Packet FIFO Receive Data (IR_SP_RX_DATA) CSR" on page 77
SP_RX_CTRL	Soft Packet Receive Control	"Soft Packet FIFO Receive Control (IR_SP_RX_CTRL) CSR" on page 76
SP_RX_STAT	Soft Packet Receive Status	"Soft Packet FIFO Receive Status (IR_SP_RX_STAT) CSR" on page 77
SP_RX_DATA	Soft Packet Receive Data	"Soft Packet FIFO Receive Data (IR_SP_RX_DATA) CSR" on page 77

Packet Transmission

Transmission of packets using the soft packet FIFO consists of the following steps:

1. Verify that the FIFO is ready to accept another packet by reading the Transmit Status CSR.
2. Write control information to Transmit Control CSR.
3. Write packet data to the Transmit Data CSR.

Loading of packet data in the transmit queue is managed by the Transmit FIFO State Machine. It is important to understand the operation of this state machine in order to write software that interacts with it. Figure shows the states and transitions that this machine implements and Table 2-17 describes the conditions that cause state transitions.

Table 2-4. Transmit FIFO State Machine State Diagram

Arc	Current State	Next State	Cause	Comments
1	Idle	Idle	The state machine is parked in this state until there is a write to the Transmit Control CSR.	This is the initial state after reset.
2	Idle	Armed	There has been a write to the Transmit Control CSR.	
3	Armed	Armed	The state machine is parked in this state until there is a write to the Transmit Data CSR.	The number of bytes in the packet to be transmitted is written to the
4	Armed	Active	There has been a write to the Transmit Data CSR.	
5	Active	Active	The state machine is parked in this state until the Octets Remaining field of the Transmit Status CSR equals zero.	Octets Remaining field of the Transmit Status CSR is decremented by four each time there is a write to the Transmit Data CSR.
6	Active	Idle	The Octets Remaining field of the Transmit Status CSR equals zero	If the Event Enable bit is set, a Soft Packet Interface Tx Event is generated during this transition.

Packet Reception

Reception of packets using the soft packet FIFO consists of the following steps:

1. Verify that the FIFO has packet data ready by reading the Receive Status CSR.
2. Read packet data from the Receive Data CSR.

Unloading packet data from the receive queue is managed by the Receive FIFO State Machine. It is important to understand the operation of this state machine in order to write software that interacts with it. Figure 2-12 shows the states and transitions that this machine implements and Table 2-5 describes the conditions that cause state transitions.

Figure 2-12. Receive FIFO State Machine State Diagram

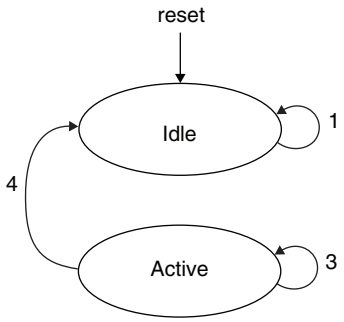


Table 2-5. Receive FIFO State Machine Transition Table

Arc	Current State	Next State	Cause	Comments
1	Idle	Idle	The state machine is parked in this state until there is a read of the Receive Data CSR.	This is the initial state after reset.
2	Idle	Active	There has been a read of the Receive Data CSR.	
3	Active	Active	The state machine is parked in this state until the Octets Remaining field of the Receive Status CSR equals zero.	Octets Remaining field of the Receive Status CSR is decremented by four each time there is a read of the Receive Data CSR.
4	Active	Idle	The Octets Remaining field of the Receive Status CSR equals zero	

Error Management

The core includes support for the RapidIO Error Management Extensions Specification. Error management support consists of the following components:

- A complete implementation of the physical layer error management extensions implemented in the OLLM and OPLM blocks.
- Implementation of logical and transport layer extensions for logical layer functions implemented in the core.
- Infrastructure for handling logical layer errors detected by user defined logical layer functions external to the core. Please refer to "Logical Port Error Capture Interface" on page 23 for detail. Errors detected by the Logical Layer are reported to the core using this 128-bit vector format.

Physical Layer Extensions

The core implements all of the physical layer error management extension described in the specification. These extensions are summarized in Table 2-6.

Table 2-6. Physical Layer Error Management Extension Summary

Error	Comments
Received control symbol	Received a control symbol with a bad CRC value.
Received out-of-sequence acknowledge control symbol	Received an acknowledge control symbol with an unexpected ackID (packet-accepted or packet_retry).
Received packet-not-accepted control symbol	
Received packet with unexpected ackID	Received packet with an ackID value that was either out-of-sequence or not outstanding.
Received packet with bad CRC	
Received packet exceeds 276 Bytes	Received packet which exceeds the maximum allowed size.

Table 2-6. Physical Layer Error Management Extension Summary (Continued)

Received Non-outstanding ackID	Link_response received with an ackID that is not outstanding.
Protocol error	An unexpected packet or control symbol was received.
Delineation error	Received unaligned /SC/ or /PD/ or undefined code-group.
Unsolicited acknowledge control symbol	An unexpected acknowledge control symbol was received.
Link time-out	An acknowledge or link-response control symbol is not received within the specified time-out interval.

The physical layer extensions are controlled by the following CSRs, which appear in the Error Reporting Block:

- Port 0 Error Detect CSR, described in section "Port 0 Error Detect (ERB_ERR_DET) CSR" on page 68
- Port 0 Error Rate Enable CSR, described in section "Port 0 Error Rate Enable (ERB_ERR_RATE_EN) CSR" on page 69
- Port 0 Attributes Capture CSR, described in section "Port 0 Attributes Capture (ERB_ATTR_CAPT) CSR" on page 70
- Port 0 Packet/Control Symbol Capture CSR, described in section "Port 0 Packet/Control Symbol Capture (ERB_PACK_SYM_CAPT) CSR" on page 71
- Port 0 Packet Capture CSR 1, described in section "Port 0 Packet Capture 1 (ERB_PACK_CAPT_1) CSR" on page 71
- Port 0 Port Packet Error Capture CSR 2, described in section "Port 0 Packet Capture 2 (ERB_PACK_CAPT_2) CSR" on page 71
- Port Packet Error Capture CSR 3, described in section "Port 0 Packet Capture 3 (ERB_PACK_CAPT_3) CSR" on page 71

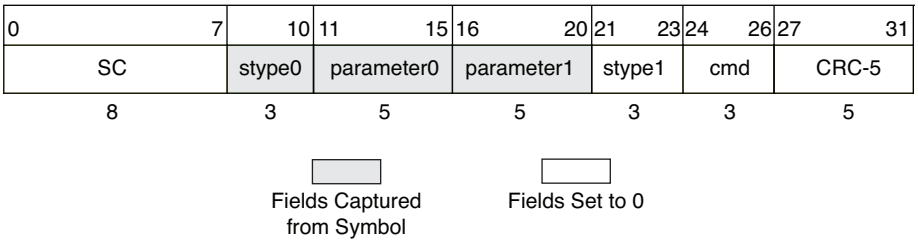
Physical Layer Error Decoding and Capture

RapidIO Physical layer errors are decoded by either receive or transmit logic depending on the error type. This has an effect on the data that is captured when the error occurs. In the case of errors decoded by the transmit logic only the fields relevant to the specific error are capture in the Port Packet/Control Symbol Error Capture CSR 0. Table 2-7 itemizes which errors are decode by the receive logic and which are decoded by the transmit logic.

Table 2-7. Physical Layer Error Capture

Error Name	Decoded By	What is captured
Corrupt Control Symbol	Rx Logic	Complete Control Symbol
Acknowledgement with Unexpected ackID	Tx Logic	Partial Control Symbol
Packet-not-accepted	Tx Logic	Partial Control Symbol
Packet with Unexpected ackID	Rx Logic	Complete Packet Header
Packet with Bad CRC	Rx Logic	Complete Packet Header
Packet greater than 276 bytes	Rx Logic	Complete Packet Header
Illegal or Invalid Character	**Not implemented**	**Not implemented**
Data Character in Idle 1 Sequence	**Not implemented**	**Not implemented**
Loss of Descrambler Synchronization	**Not implemented**	**Not implemented**
Link_response with non-outstanding ackID	Tx Logic	Partial Control Symbol
Protocol error	Tx Logic	Partial Control Symbol
Deliniation error	Rx Logic	Complete Control Symbol
Unsolicited acknowledgement	Tx Logic	Partial Control Symbol
Link time-out	Tx Logic	**Partial Packet Header**

Figure 2-13. Partial Control Symbol Capture Fields



User-Implemented Logical Transport Extensions

The core provides a common infrastructure for reporting logical layer errors detected by user defined logic connected to the core. Interaction between the core and these functions is done through the following interfaces:

- Logical Layer Common Control and Status Interface
- Logical Port Error Capture Interface

Logical Layer Common Control and Status Interface

The Logical Layer Common Control and Status Interface presents information contained in the logical layer control CSRs contained in the core. From the standpoint of error management there are two signals on this interface that are significant:

- **lt_error_en[0:31]** – This vector reflects the contents of the Logical/Transport Layer Error Enable CSR. User defined logical layer functions must monitor this vector to verify that they have detected has been enabled before reporting it.
- **resp_time_out[0:23]** – This vector reflects the state of bits 0-23 of the Port Response Time-out Control CSR. RapidIO logical layer functions that initiate transactions must use this value to set the timeout period for responses.

Logical Port Error Capture Interface

This interface is used to report errors detected by the user defined logical layer functions back to the core for reporting. When a logical layer error is detected that has not been masked by the corresponding bit in the lt_error_en vector, the user function should present the error information on the logN_error_capt_data vector and assert the logN_error_capt_trigd_req_n indication. This information must be held stable until acknowledged by the logN_error_capt_trigd_ack_n indication. The format of the logN_error_capt_data vector is shown in 10. Note that if another logical layer error was already reported and the Logical/Transport Capture were loaded with that error data and locked, this error data will be discarded.

Table 2-8. Error Capture Vector Field Definitions

Bit Field	Name	Description
logN_error_capt_data[0:31]	address[0:31]	Most significant 32 bits of the address associated with the error (for requests, for responses if available). This field is loaded into bits 0 to 31 of the Logical/Transport Layer High Address Capture CSR.
logN_error_capt_data[32:60]	address[32:60]	Least significant 29 bits of the address associated with the error (for requests, for responses if available). This field is loaded into bits 0 to 28 of the Logical/Transport Layer Address Capture CSR.
logN_error_capt_data[61]	reserved	Reserved field. Implementations should set this bit-field to 0. This field is loaded into bit 29 of the Logical/Transport Layer Address Capture CSR.

Table 2-8. Error Capture Vector Field Definitions (Continued)

logN_error_capt_data[62:63]	xamsbs	Extended address bits of the address associated with the error (for requests, for responses if available). This field is loaded into bits 30 to 31 of the Logical/Transport Layer Address Capture CSR.
logN_error_capt_data[64:71]	MSB destinationID	Most significant byte of the destinationID associated with the error (large transport systems only). This field is loaded into bits 0 to 7 of the Logical/Transport Layer Device ID Capture CSR.
logN_error_capt_data[72:79]	destinationID	The destinationID associated with the error. This field is loaded into bits 8 to 15 of the Logical/Transport Layer Device ID Capture CSR.
logN_error_capt_data[80:87]	MSB sourceID	Most significant byte of the sourceID associated with the error (large transport systems only). This field is loaded into bits 16 to 23 of the Logical/Transport Layer Device ID Capture CSR.
logN_error_capt_data[88:95]	sourceID	The sourceID associated with the error. This field is loaded into bits 24 to 31 of the Logical/Transport Layer Device ID Capture CSR.
logN_error_capt_data[96:99]	ftype	Format type associated with the error. This field is loaded into bits 0 to 3 of the Logical/Transport Layer Control Capture CSR.
logN_error_capt_data[100:103]	ttype	Transaction type associated with the error. This field is loaded into bits 4 to 7 of the Logical/Transport Layer Control Capture CSR.
logN_error_capt_data[104:111]	msg info	letter, mbox, and msgseg for the last Message request received for the mailbox that had an error (Message errors only). This field is loaded into bits 8 to 15 of the Logical/Transport Layer Control Capture CSR.
logN_error_capt_data[112:122]	implementation defined	Implementation defined field. This field is loaded into bits 16 to 26 of the Logical/Transport Layer Control Capture CSR.
logN_error_capt_data[123:127]	error type	This field indicates the type of error that has been detected. It is used to set the appropriate bit (indexed by this 5-bit vector) in the Logical/Transport Layer Error Detect CSR. It is also loaded into bits 27 to 31 of the Logical/Transport Layer Control Capture CSR. This 5-bit vector defines the bit number (0 to 31) to set in the Logical Layer Error Detect CSR.