ECMA Standard-342

February 2003

ECMA International

Standardizing Information and Communication Systems

RapidIO^{TM} Interconnect Specification

ECMA Standard-342

February 2003

ECMA International

Standardizing Information and Communication Systems

RapidIO^{TM} Interconnect Specification

Phone: +41 22 849.60.00 - **Fax:** +41 22 849.60.01 - **URL:**http://www.ecma.ch - **Internet:** helpdesk@ecma.ch

Brief History

The RapidIO™ architecture was developed to address the need for a high-performance low pin count packet-switched system level interconnect to be used in a variety of applications as an open standard. The architecture is targeted toward networking, telecom, and high performance embedded applications. It is intended primarily as an intra-system interface, allowing chip-to-chip and board-to-board communications at Gigabyte per second performance levels. It provides a rich variety of features including high data bandwidth, low-latency capability and support for high-performance I/O devices, as well as providing globally shared memory, message passing, and software managed programming models. In its simplest form, the interface can be implemented in a FPGA end point. The interconnect defines a protocol independent of a physical implementation. The physical features of an implementation utilizing the interconnect are defined by the requirements of the implementation, such as I/O signaling levels, interconnect topology, physical layer protocol, error detection, and so forth. The architecture is intended and partitioned to allow adaptation to a multitude of applications.

This ECMA Standard has been adopted by the General Assembly in February 2003.

Overview of the standard

This overview explains each of the three layers of the RapidIO architecture, their interrelationships, an the system and device interoperability:

1. Logical layer—The logical layer defines the overall protocol and packet formats, the types of transactions that can be carried out with RapidIO, how addressing is handled. The logical specifications are partitioned into two partitions:

   — *Partition I: Input/Output Logical Specification*

   — *Partition II: Message Passing Logical Specification*

   — *Partition V: Globally Shared Memory Logical Specification*

2. Transport layer—The transport layer provides the necessary route information for a packet to move from one point to another. This information is covered in *Partition III: Common Transport Specification.*
3. Physical layer—The physical layer contains the device level interface such as packet transport mechanisms, flow control, electrical characteristics, and low-level error management. This standard covers these topics in *Partition IV: Physical 8/16 LP-LVDS Specification,* and in *Partition VI: Physial Layer 1X/4X LP-Serial Specification.*
4. Inter-operability — This consists of a standard setod device and system design solutions to provide for interoperability. The specification is given in *Patition VII: Inter-operability Specification System and Device.*

NOTE

RapidIO specifications are structured so that additions can be made to each without affecting the others. For example, each logical specification is independent and can be implemented alone.

Partitions I, II and V: Logical Specifications

In RapidIO, the logical layer is subdivided into two specifications that support distributed I/O processing. Partition I: Input/Output Logical Specification explains how RapidIO supports input-output systems and Partition II: Message Passing Logical Specification describes the message passing features of the RapidIO interconnect. Additionally, Partition V: Globally Shared Memory Logical Specification, specifies an extension for applications that support cache-coherency and multi-processing.

The logical specifications do not imply a specific transport or physical interface, therefore they are specified in a bit stream format. Necessary bits are added to the logical encodings for each lower layer in the hierarchy.

Because all logical layers fulfill the same data communication functions no matter what programming model they support, specifications written to this logical level address similar issues. In RapidIO, this similarity among the logical specifications is reflected in the chapter contents, with each of the logical specifications containing the following chapters:

- Chapter 1, " System Models," provides explanations and figures of the types of systems that can use a RapidIO interface.
- Chapter 2, "Operation Descriptions," describes the sets of operations and transactions supported by RapidIO message passing and input/output protocols.
- Chapter 3, "Packet Format Descriptions," breaks down packets into the two basic classes of request and response packets and then discusses and illustrates the format types within each class for each logical specification.
- Chapter 4, "Message Passing Registers," and Chapter 4, "Input/Output Registers," provides a memory map of registers used in the message passing and I/O specifications, and then subsections that discuss and illustrate each register.

The message passing logical specification has an annex added that describes in greater detail two examples of RapidIO message passing models, one a simple model and one a more extended model.

The extension to the logical specifications as given in Partition V contains the following chapters:

- Chapter 1, "Overview," describes the set of operations and transactions supported by the RapidIO globally shared memory protocols.

- Chapter 2, “System Models,” introduces some possible devices that could participate in a RapidIO GSM system environment. The chapter explains the memory directory-based mechanism that tracks memory accesses and maintains cache coherence. Transaction ordering and deadlock prevention are also covered.
- Chapter 3, “Operation Descriptions,” describes the set of operations and transactions supported by the RapidIO globally-shared memory (GSM) protocols.
- Chapter 4, “Packet Format Descriptions,” contains the packet format definitions for the GSM specification. The two basic types, request and response packets, with their sub-types and fields are defined. The chapter explains how memory read latency is handled by RapidIO.
- Chapter 5, “Globally Shared Memory Registers,” describes the visible register set that allows an external processing element to determine the globally shared memory capabilities, configuration, and status of a processing element using this logical specification. Only registers or register bits specific to the GSM logical specification are explained. Refer to the other RapidIO logical, transport, and physical specifications of interest to determine a complete list of registers and bit definitions.
- Chapter 6, “Communication Protocols,” contains the communications protocol definitions for this GSM specification.
- Chapter 7, “Address Collision Resolution Tables,” explains the actions necessary under the RapidIO GSM model to resolve address collisions.

Partition III: Common Transport Specification

Partition III: Common Transport Specification contains three chapters:

The introduction to Partition III: Common Transport Specification offers a general understanding of the features and functions of the transport specification.

- Chapter 1, “Transport Format Description,” describes the routing methods used in RapidIO for sending packets across the systems of switches described in this chapter.
- Chapter 2, “Common Transport Registers,” describes the visible register set that allows an external processing element to determine the capabilities, configuration, and status of a processing element using this RapidIO transport layer definition.

Partitions IV and VI: Physical Layer 8/16 LP-LVDS and 1x/4x LP-Serial Specifications

Partition IV: Physical Layer 8/16 LP-LVDS Specification contains eight chapters and an annex:

- The introduction to *Partition IV: Physical Layer 8/16 LP-LVDS Specification* offers a general understanding of the features and functions of the physical layer specification.
- Chapter 1, “Physical Layer Protocol,” describes the physical layer protocol for packet delivery to the RapidIO fabric, including packet transmission, flow control, error management, and link maintenance protocols.
- Chapter 2, “Packet and Control Symbol Transmission,” defines packet and control symbol delineation and alignment on the physical port and mechanisms to control the pacing of a packet.
- Chapter 3, “Control Symbol Formats,” explains the physical layer control formats that manage the packet delivery protocols mentioned in Chapter 2.
- Chapter 4, “8/16 LP-LVDS Registers,” describes the register set that allows an external processing element to determine the physical capabilities and status of an 8/16 LP-LVDS RapidIO implementation.
- Chapter 5, “System Clocking Considerations,” discusses the RapidIO synchronous clock and how it is distributed in a typical switch configuration.
- Chapter 6, “Board Routing Guidelines,” explains board layout guidelines and application environment considerations for the RapidIO architecture.
- “Chapter 7,” contains the signal pin descriptions for a RapidIO end point device.
- Chapter 8, “Electrical Specifications,” describes the low voltage differential signaling (LVDS) electrical specifications of the RapidIO 8/16 LP-LVDS device.
- Annex A, “Interface Management (Informative),” contains information pertinent to interface management in a RapidIO system, including SECDED error tables, error recovery, link initialization, and packet retry state machines.

Partition VI: Physical Layer 1x/4x LP-Serial Specification containssight chapters and two annexes:

- Chapter 1, “Overview”, offers a general understanding of the futures of the physical layer specification.
- Chapter 2, “Packets”, defines the LP-Serial packet format and the fields that are added by the LP-Serial physical layer.
- Chapter 3, “Control Symbols”, defines the format of the two classes of control symbols which are the message elements sed by ports connected by an LP-Serial link to manage all aspects of the link operation.

- Chapter 4, “PCS and PMA Layers”, defines the fonctions provided by the Physical Coding Sublayer (PSC) and the Physical Media Attachment (PMA) sublayer, comprising encoding, link transmission rules, serialization and link initialization.
- Chapter 5, “LP-Serial Protocol”, defines how packets, control symbols, and the PSC/PMA sublayers are used to implement the physical layer protocol that provides the reliable delivery of packets between two RapidIO devices that are connected by an LP-Serial link.
- Chapter 6, “LP-Serial Registers”, defines the physical layer control and status register set. By accessing this LP-Serial Command and Status Register (CSR) set a processing element may query the capabilities and status, and configure another processing element.
- Chapter 7, “Signal Descriptions”, describes the signal pin for end point devices and shows the connectivity between processing elements with 1x ports and those with 4x ports.
- Chapter 8, “A.C. Electrical Specifications”, defines the electrical requirements for the LP-Serial device, comprosing two transmission types and three speed grades.
- Annex A, “Interface Management”, describes state machines showing examples for error recovery, link initialization and packet retry.
- Annex B, “Bibliography”.

Partition VII: Inter-operability Specification System and Device

Partition VII: Inter-operability Specification System and Device contains chapters:

- Chapter 1, “Overview”, provides a short survey about one way of interworking of system components.
- Chapter 2, “System Exploration and Initialization”, describes a system that is explored and configured at boot time by processors, in order to support relatively fast-changing plug-and-play or hot-swap systems.
- Chapter 3, “816 LP-LVDS Device Class Requirements”, describes the requirements for devices adhering to the 8/16 LP-LVDS physical layer specification, and defines three classes of devices with increasing levels of functionality.
- Chapter 4, “PCI Considerations”, describes architectural considerations for an implementation of a RapidIO to PCI (PVCI 2.2 and PCI x1.0) bridge processing element.
- Chapter 5, “Globally Shared Memory (GSM) Devices”, defines the portions of the GMS protocol necessary to implement different processing elements (devices). Additionally, this chapter contains the 8/16 LP-LVDS and 1x/4x LP-Serial transaction to priority mappings to guarantee that a system maintains cache coherence and is deadlock free.

Partition VIII: Error Management Extensions Specification

Partition VIII: Error Management Extensions Specification contains two chapters and one annex:

- Chapter 1, “Error Management Extensions”, defines the additional requirements for all physical and logical layers, and describes the behavior of a device when an error is detected and how the new registers and bits are managed by sofware and hardware.
- Chapter 2, “Error Management Registers”, describes the Error Management Extended Features block, and a number of new bits to the existing standard physical layer registers.
- Annex A provides information and background on the application of the error management capabilities.

Extensions

Extensions to this base set of RapidIO specifications will be published periodically under separate cover.

Terminology

Refer to the Glossary at the back of this document.

Conventions

||	Concatenation, used to indicate that two fields are physically associated as consecutive bits
ACTIVE_HIGH	Names of active high signals are shown in uppercase text with no overbar. Active-high signals are asserted when high and not asserted when low.
$\overline{\text{ACTIVE\_LOW}}$	Names of active low signals are shown in uppercase text with an overbar. Active low signals are asserted when low and not asserted when high.
italics	Book titles in text are set in italics.
REG[FIELD]	Abbreviations or acronyms for registers are shown in uppercase text. Specific bits, fields, or ranges appear in brackets.

TRANSACTION	Transaction types are expressed in all caps.
operation	Device operation types are expressed in plain text.
n	A decimal value.
[n-m]	Used to express a numerical range from n to m.
0bnn	A binary value, the number of bits is determined by the number of digits.
0xnn	A hexadecimal value, the number of bits is determined by the number of digits or from the surrounding context; for example, 0xnn may be a 5, 6, 7, or 8 bit value.

- I -

Summary

This summary is intended for anyone who needs a high-level understanding of the RapidIO architecture. The layers of the architecture are described, and the functional, physical, and performance features are presented.

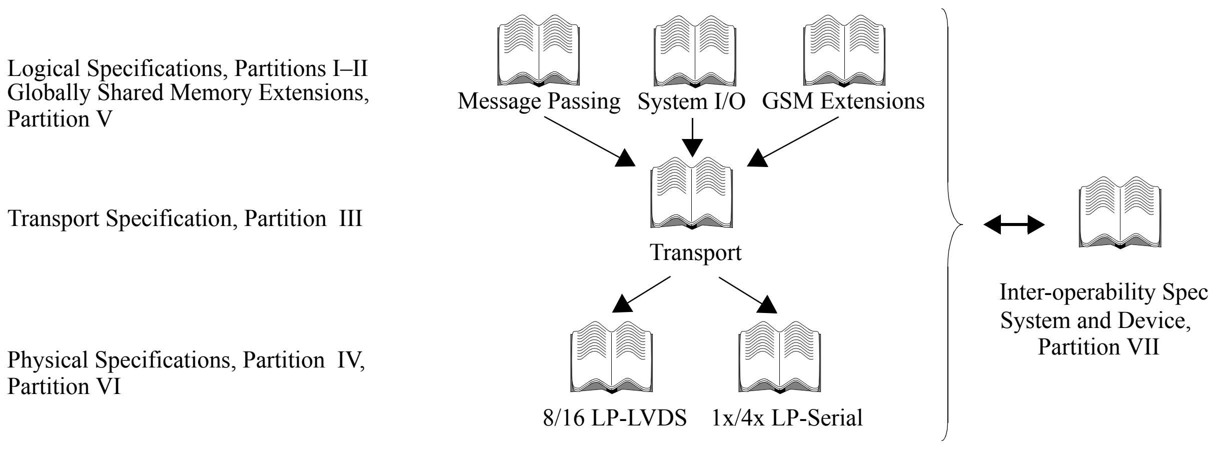
Introduction

RapidIO is a packet-switched interconnect intended primarily as an intra-system interface for chip-to-chip and board-to-board communications at Gigabyte-per-second performance levels. Uses for the architecture can be found in connected microprocessors, memory, and memory mapped I/O devices that operate in networking equipment, memory subsystems, and general purpose computing.

The RapidIO interconnect is targeted toward memory-mapped distributed memory systems and subsystems. Such systems consist of multiple independent devices that use DMA engines to communicate data and maintain their consistency by passing messages back and forth among the devices. The majority of applications written today use such a DMA and message passing programming model.

RapidIO is a definition of a system interconnect. System concepts such as processor programming models, caching, system reset, and interrupt programming models are beyond the scope of the RapidIO architecture. However, these functions may use facilities provided within the RapidIO interconnect to support the necessary behavior. For example, the RapidIO architecture provides the necessary operations to support processor programming models ranging from strong consistency through total store ordering to weak ordering. Any reference to these areas within the RapidIO architecture specification is for illustration only. Subsequent revisions of the RapidIO specifications may further define these system functions.

Although this set of RapidIO specifications is built for the distributed memory system, further RapidIO specifications extend the capabilities of the interface and address topics such as a serial physical layer, globally shared memory, and inter-operability requirements. These specifications are available under separate covers from the specifications contained in this book.


Logical Specifications, Partitions I–II
Globally Shared Memory Extensions, Partition V
Message Passing
System I/O
GSM Extensions
Transport Specification, Partition III
Transport
Inter-operability Spec System and Device, Partition VII
Physical Specifications, Partition IV, Partition VI
8/16 LP-LVDS
1x/4x LP-Serial

RapidIO Layered Hierarchy

The RapidIO architecture is specified in a three-layer hierarchy consisting of logical, common transport, and physical specifications:

- Logical specifications—The logical specifications define the operation protocols required by the end points to carry out the targeted operation and the necessary transaction packet formats. The logical specifications do not imply a specific transport nor physical interface, therefore they are specified in a bit stream format. Necessary bits are added to the logical formats for each lower layer in the hierarchy.
- Because applications are written using different programming models, the RapidIO architecture subdivides its specifications to support them. Currently the RapidIO logical specifications include the following:
- System I/O specifications in *Partition I: Input/Output Logical Specification*
- Message passing specifications in *Partition II: Message Passing Logical Specification*
- Additional logical layer specifications under separate covers

- Transport specification—The common transport specification describes the packet addressing scheme for delivery of RapidIO packets from one end point to another. The common transport specification is common to all of RapidIO and is described in *Partition III: Common Transport Specification.*
- Physical specification—A set of physical layer specifications define the interface between two devices and the packet transport mechanisms, flow control, and electrical characteristics. This specification is described in *Partition IV: Physical 8/16 LP-LVDS Specification.* Additional physical layer specifications are under separate covers.

RapidIO Feature Set

The RapidIO feature set and protocols are based upon a number of considerations for both general computing and embedded applications. In each of the three layers, these features are broken down into three categories: functional, physical, and performance.

Logical Layer Features

Message passing and direct-memory access (DMA) devices can improve the interconnect efficiency if larger non-coherent data quantities are encapsulated within a single packet, so RapidIO supports a variety of data sizes within the packet formats. Because the message passing programming model is fundamentally a non-coherent non-shared memory model, in a RapidIO device, portions of the memory space are only directly accessible by a processor or a local device controlled message passing interface.

Packet headers are as small as possible to minimize the control overhead and are organized for fast, efficient assembly and disassembly. As the amount of data included in a packet increases, packet efficiency increases. RapidIO supports data payloads up to 256 bytes. Messages are very important for embedded control applications, so a variety of large and small data fields and multiple packet messages are supported.

Multiple transactions are allowed concurrently in the system, not only through the ability to pipeline transactions from a single device, but also through spatial reuse of interfaces between different devices in the system. Without this, a majority of the potential system throughput is wasted.

Functional Features

The following are RapidIO logical layer functional features:

- Many embedded systems are multiprocessor systems, not multiprocessing systems, and prefer a message passing or software-based coherency programming model over the traditional computer-style globally shared memory programming model in order to support their distributed I/O and processing requirements, especially in the networking and routing markets. RapidIO supports all of these programming models.
- System sizes from very small to very large are supported in the same or compatible packet formats—RapidIO plans for future expansion and requirements.
- Read-modify-write atomic operations are useful for synchronization between processors or other system elements.
- The RapidIO architecture supports 50- and 66-bit addresses as well as 34-bit local addresses for smaller systems.
- Message passing and DMA devices can improve the interconnect efficiency if larger non-coherent data quantities can be encapsulated within a single packet, so RapidIO supports a variety of data sizes within the packet formats.
- Because the message passing programming model is fundamentally a non-coherent non-shared memory model, RapidIO can assume that portions of the memory space are only directly accessible by a processor or device local to that memory space. A remote device that attempts to access that memory space must do so through a local device controlled message passing interface.

Physical Features

The following are features of the RapidIO logical layer designed to satisfy the needs of the physical layer requirements for various applications and systems:

- The RapidIO packet definition is independent of the width of the physical interface to other devices on the interconnect fabric.
- The protocols and packet formats are independent of the physical interconnect topology. The protocols work whether the physical fabric is a point-to-point ring, a bus, a switched multi-dimensional network, a duplex serial connection, and so forth.
- RapidIO is not dependent on the bandwidth or latency of the physical fabric.

- The protocols handle out-of-order packet transmission and reception.
- No requirement exists in RapidIO for geographical addressing; a device's identifier does not depend on its location in the address map but can be assigned by other means.
- Certain devices have bandwidth and latency requirements for proper operation. RapidIO should not preclude an implementation from imposing these constraints within the system.

Performance Features

Following are performance features at the logical layer:

- Messages are very important for networking applications, so a variety of large and small data fields and multiple packet messages are supported for efficiency.
- Packet headers are as small as possible to minimize the control overhead and are organized for fast, efficient assembly and disassembly.
- Multiple transactions are allowed concurrently in the system, preventing much potential system input from being wasted.

Transport Layer Features

The transport layer functions of the RapidIO interconnect have been addressed by incorporating the following functional, physical, and performance features.

Functional Features

Functional features at the transport layer include the following:

- System sizes from very small to very large are supported in the same or compatible packet formats.
- Because RapidIO has only a single transport specification, compatibility among implementations is assured.
- The transport specification is flexible, so that it can be adapted to future applications.
- Packets are assumed, but not required, to be directed from a single source to a single destination.

Physical Features

The following are physical features of the RapidIO fabric that apply at the transport layer:

- The transport definition is independent of the width of the physical interface between devices in the interconnect fabric.
- No requirement exists in RapidIO for geographical addressing; a device's identifier does not depend on its location in the address map but can be assigned by other means.

Performance Features

Performance features that apply to the transport layer include the following:

- Packet headers are as small as possible to minimize the control overhead and are organized for fast, efficient assembly and disassembly.
- Broadcasting and multicasting can be implemented by interpreting the transport information in the interconnect fabric.

Physical Layer Features

The physical layer defines the signal definitions, flow control and error management for RapidIO. Initially an 8-bit and 16-bit parallel (8/16 LP-LVDS), point-to-point interface is deployed. An 8/16 LP-LVDS device interface contains a dedicated 8- or 16-bit input port with clock and frame signals, and a 8- or 16-bit output port with clock and frame signals. A source-synchronous-clock signal clocks packet data on the rising and falling edges. A frame signal provides a control reference. Differential signaling is used to reduce interface complexity, provide robust signal quality, and promote good frequency scalability across printed circuit boards and connectors.

Functional Features

Following is a functional feature of the physical layer of RapidIO:

- RapidIO provides a flow control mechanism between devices that communicate on the RapidIO interconnect fabric, because infinite data buffering is not available in a device.

Physical Features

The following are physical features of the RapidIO physical layer:

- Connections are point-to-point unidirectional, one in and one out, with 8-bit or 16-bit ports
- Physical layer protocols and packet formats are to some degree independent of the topology of the physical interconnect; however; the physical structure is assumed to be link-based.
- There is no dependency in RapidIO on the bandwidth or latency of the physical fabric.
- Physical layer protocols handle out-of-order and in-order packet transmission and reception.
- Physical layer protocols are tolerant of transient errors caused by high frequency operation of the interface or excessive noise in the system environment.

Performance Features

The following are performance features of the RapidIO physical layer:

- Physical protocols and packet formats allow for the smallest to the largest data payload sizes
- Packet headers are as small as possible to minimize the control overhead and are organized for fast, efficient assembly and disassembly.
- Multiple transactions are allowed concurrently in the system, preventing much potential system input from being wasted.
- The electrical specification allows for the fastest possible speed of operation for future devices.

Table of contents

Partition I: Input/Output Logical Specification

I Partition I

Partition I is intended for users who need to understand the input/output system architecture of the RapidIO interconnect.

I.1 Overview

The *Input/Output Logical Specification* is part of RapidIO's logical layer specifications that define the interconnect's overall protocol and packet formats. This layer contains the transaction protocols necessary for end points to process a transaction. Another RapidIO logical layer specification is described in *Partition II: Message Passing Logical Specification.*

The logical specifications do not imply a specific transport or physical interface; therefore they are specified in a bit stream format. At the lower levels in the RapidIO three-layer hierarchy, necessary bits are added to the logical encoding for the transport and physical layers.

RapidIO is targeted toward memory-mapped distributed memory systems. A message passing programming model is supported to enable distributed I/O processing. *Partition I: Input/Output Logical Specification* defines the basic I/O system architecture of RapidIO.

I.2 Contents

Following are the contents of *Partition I: Input/Output Logical Specification*:

- Chapter 1, "System Models," introduces some devices that might be part of a RapidIO system environment. System issues, such as the ordered and unordered systems that can be built using RapidIO, are explained.
- Chapter 2, "Operation Descriptions," describes the set of transactions and operations supported by the I/O protocols.
- Chapter 3, "Packet Format Descriptions," contains the packet format definitions for the Input/Output specification. The two basic types, request and response packets, with their sub-types and fields are defined.
- Chapter 4, "Input/Output Registers," describes the visible register set that allows an external processing element to determine the I/O capabilities, configuration, and status of a processing element using this logical specification. Only registers or register bits specific to the Input/Output specification are explained. Refer to the other RapidIO logical, transport, and physical specifications of interest to determine a complete list of registers and bit definitions.

1 Chapter 1 - System Models

This overview introduces some possible devices in a RapidIO system.

1.1 Processing Element Models

"Figure 1-1. A Possible RapidIO-Based Computing System" on page 4 describes a possible RapidIO-based computing system. The processing element is a computer device such as a processor attached to a local memory and to a RapidIO system interconnect "Figure i. RapidIO Layered Hierarchy," . The bridge part of the system provides I/O subsystem services such as high-speed PCI interfaces and gigabit Ethernet ports, interrupt control, and other system support functions.

The following sections describe several possible processing elements.

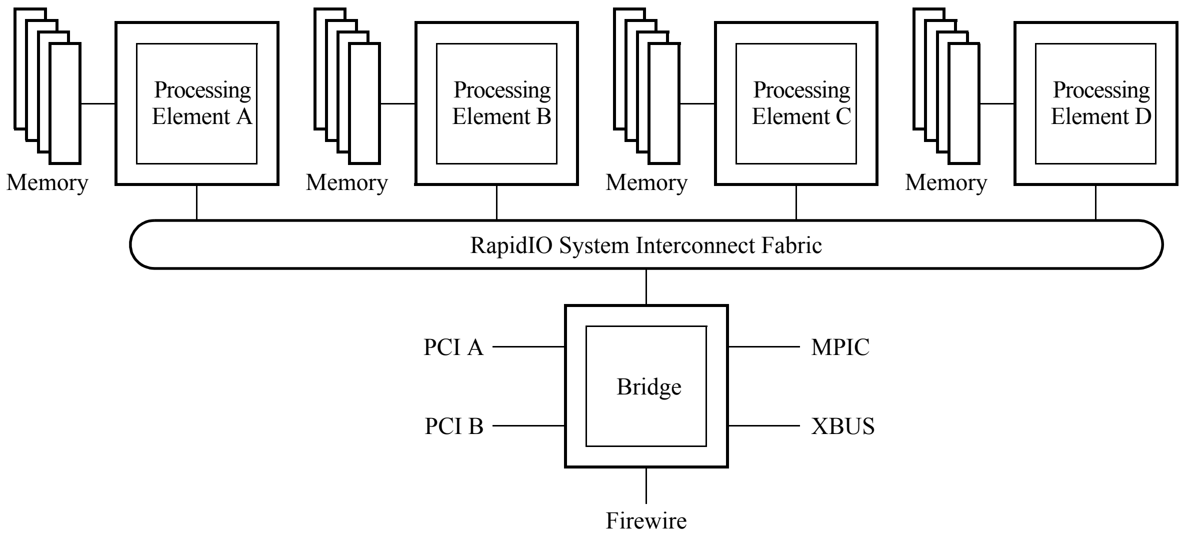


Figure 1-1. A Possible RapidIO-Based Computing System

1.1.1 Processor-Memory Processing Element Model

Figure 1-2 shows an example of a processing element consisting of a processor connected to an agent device. The agent carries out several services on behalf of the processor. Most importantly, it provides access to a local memory that has much lower latency than memory that is local to another processing element (remote memory accesses). It also provides an interface to the RapidIO interconnect to service those remote memory accesses.

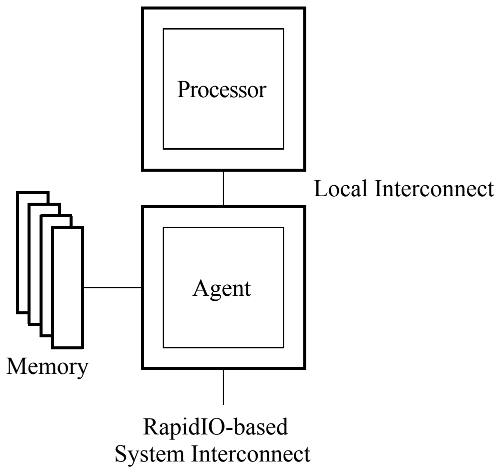


Figure 1-2. Processor-Memory Processing Element Example

1.1.2 Integrated Processor-Memory Processing Element Model

Another form of a processor-memory processing element is a fully integrated component that is designed specifically to connect to a RapidIO interconnect system as shown in Figure 1-3. This type of device integrates a memory system and other support logic with a processor on the same piece of silicon or within the same package.

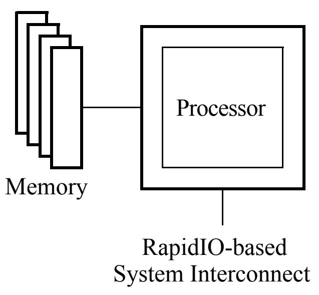


Figure 1-3. Integrated Processor-Memory Processing Element Example

1.1.3 Memory-Only Processing Element Model

A different processing element may not contain a processor at all, but may be a memory-only device as shown in Figure 1-4. This type of device is much simpler than a processor; it only responds to requests from the external system, not to local requests as in the processor-based model. As such, its memory is remote for all processors in the system.

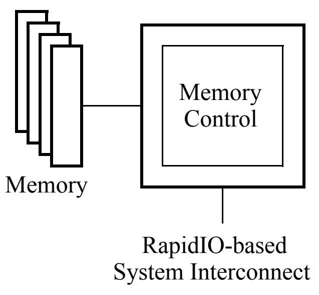


Figure1-4. Memory-Only Processing Element Example

1.1.4 Processor-Only Processing Element

Similar to a memory-only element, a processor-only element has no local memory. A processor-only processing element is shown in Figure 1-5

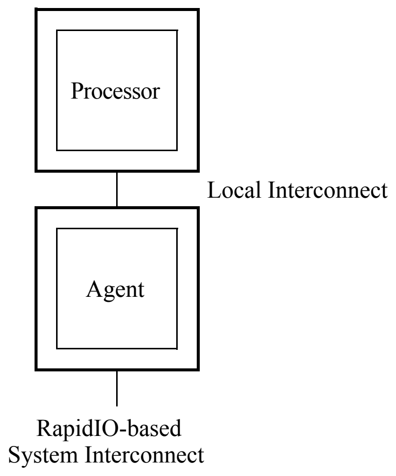


Figure 1-5. Processor-Only Processing Element Example

1.1.5 I/O Processing Element

This type of processing element is shown as the bridge in Figure 1-1. This device has distinctly different behavior than a processor or a memory device. An I/O device only needs to move data into and out of local or remote memory.

1.1.6 Switch Processing Element

A switch processing element is a device that allows communication with other processing elements through the switch. A switch may be used to connect a variety of RapidIO-compliant processing elements. A possible switch is shown in Figure1-6. Behavior of the switches, and the interconnect fabric in general, is addressed in the *RapidIO Common Transport Specification*.

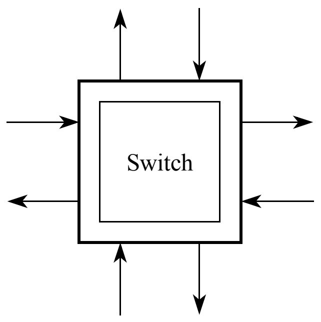


Figure 1-6. Switch Processing Element Example

1.2 System Issues

The following sections describe transaction ordering and system deadlock considerations in a RapidIO system.

1.2.1 Operation Ordering

Most operations in an I/O system do not have any requirements as far as completion ordering. There are, however, several tasks that require events to occur in a specific order. As an example, a processing element may wish to write a set of registers in another processing element. The sequence in which those writes are carried out may be critical to the operation of the target processing element. Without some specific system rules there would be no guarantee of completion ordering of these operations. Ordering is mostly a concern for operations between a specific source and destination pair.

In certain cases a processing element may communicate with another processing element or set of processing elements in different contexts. A set or sequence of operations issued by a processing element may have requirements for completing in order at the target processing element. That same processing element may have another sequence of operations that also requires a completion order at the target processing element. However, the issuing processing element has no requirements for completion order between the two sequences of operations. Further, it may be desirable for one of the sequences of operations to complete at a higher priority than the other sequence. The term “transaction request flow” is defined as one of these sequences of operations.

A transaction request flow is defined as a ordered sequence of non-maintenance request transactions from a given source (as indicated by the source identifier) to a given destination (as indicated by the transaction destination identifier), where a maintenance request is a special system support request. Each packet in a transaction request flow has the same source identifier and the same destination identifier.

There may be multiple transaction request flows between a given source and destination pair. When multiple flows exist between a source and destination pair, the flows are distinguished by a flow indicator (flowID). Rapid IO allows multiple transaction request flows between any source and destination pair. The flows between each source and destination pair are identified with alphabetic characters beginning with A.

The flows between each source and destination pair are prioritized. The flow priority increases alphabetically with flow A having the lowest priority, flow B having the next to lowest priority, etc. When multiple transaction request flows exist between a given source and destination pair, transactions of a higher priority flow may pass transactions of a lower priority flow, but transactions of a lower priority flow may not pass transactions of a higher priority flow.

Maintenance transactions are not part of any transaction request flow. However, within a RapidIO fabric, maintenance transactions may not pass other maintenance transactions of the same or higher priority taking the same path through the fabric.

Response transactions are not part of any transaction request flow. There is no ordering between any pair of response transactions and there is no ordering between any response transaction and any request transaction that did not cause the generation of the response.

To support transaction request flows, all devices that support the RapidIO logical specification shall comply as applicable with the following Fabric Delivering Ordering and End point Completion Ordering rules.

Fabric Delivery Ordering Rules

1. Non-maintenance request transactions within a transaction request flow (same source identifier, same destination identifier, and same flowID) shall be delivered to the logical layer of the destination in the same order that they were issued by the logical layer of the source.

2. Non-maintenance request transactions that have the same source (same source identifier) and the same destination (same destination identifier) but different flowIDs shall be delivered to the logical layer of the destination as follows.

- A transaction of a higher priority transaction request flow that was issued by the logical layer of the source before a transaction of a lower priority transaction request flow shall be delivered to the logical layer of the destination before the lower priority transaction.
- A transaction of a higher priority transaction request flow that was issued by the logical layer of the source after a transaction of a lower priority transaction request flow may be delivered to the logical layer of the destination before the lower priority transaction.

3. Request transactions that have different sources (different source identifiers) or different destinations (different destination identifiers) are unordered with respect to each other.

End point Completion Ordering Rules

1. Write request transactions in a transaction request flow shall be completed at the logical layer of the destination in the same order that the transactions were delivered to the logical layer of the destination.

2. A read request transaction with source A and destination B shall force the completion at the logical layer of B of all write requests in the same transaction request flow that were received by the logical layer of B before the read request transaction.

Read request transactions need not be completed in the same order that they were received by the logical layer of the destination. As a consequence, read response transactions need not be issued by the logical layer of the destination in the same order that the associated read request transactions were received.

Write response transactions will likely be issued at the logical level in the order that the associated write request was received. However, since response transactions are not part of any flow, they are not ordered relative to one another and may not arrive at the logical level their destination in the same order as the associated write transactions were issued. Therefore, write response transactions need not be issued by the logical layer in the same order as the associated write request was received.

It may be necessary to impose additional rules in order to provide for inter operability with other interface standards or programming models. However, such additional rules are beyond the scope of this specification.

1.2.2 Transaction Delivery

There are two basic types of delivery schemes that can be built using RapidIO processing elements: unordered and ordered. The RapidIO logical protocols assume that all outstanding transactions to another processing element are delivered in an arbitrary order. In other words, the logical protocols do not rely on transaction interdependencies for operation. RapidIO also allows completely ordered delivery systems to be constructed. Each type of system puts different constraints on the implementation of the source and destination processing elements and any intervening hardware. The specific mechanisms and definitions of how RapidIO enforces transaction ordering are discussed in the appropriate physical layer specification.

1.2.2.1 Unordered Delivery System Issues

An unordered delivery system is defined as an interconnect fabric where transactions between a source and destination pair can arbitrarily pass each other during transmission through the intervening fabric.

Operations in the unordered system that are required to complete in a specific order shall be properly managed at the source processing element. For example, enforcing a specific sequence for writing a series of configuration registers, or preventing a subsequent read from bypassing a preceding write to a specific address are cases of ordering that may need to be managed at the source. The source of these transactions shall issue them in a purely serial sequence, waiting for completion notification for a write before issuing the next transaction to the interconnect fabric. The destination processing element shall guarantee that all outstanding non-coherent operations from that source are completed before servicing a subsequent non-coherent request from that source.

1.2.2.2 Ordered Delivery System Issues

Ordered delivery systems place additional implementation constraints on both the source and destination processing elements as well as any intervening hardware. Typically an ordered system requires that all transactions between a source/destination pair be completed in the order generated, not necessarily the order in which they can be accepted by the destination or an intermediate device. In one example, if several requests are sent before proper receipt is acknowledged the destination or intermediate device shall retry all following transactions until the first retried packet is retransmitted and accepted. In this case, the source shall "unroll" its outstanding transaction list and retransmit the first one to maintain the proper system ordering. In another example, an interface may make use of explicit transaction tags which allow the destination to place the transactions in the proper order upon receipt.

1.2.3 Deadlock Considerations

A deadlock can occur if a dependency loop exists. A dependency loop is a situation where a loop of buffering devices is formed, in which forward progress at each device is dependent upon progress at the next device. If no device in the loop can make progress then the system is deadlocked.

The simplest solution to the deadlock problem is to discard a packet. This releases resources in the network and allows forward progress to be made. RapidIO is designed to be a reliable fabric for use in real time tightly coupled systems, therefore discarding packets is not an acceptable solution.

In order to produce a system with no chance of deadlock it is required that a deadlock free topology be provided for response-less operations. Dependency loops to single direction packets can exist in unconstrained switch topologies. Often the dependency loop can be avoided with simple routing rules. Topologies like hypercubes or three-dimensional meshes physically contain loops. In both cases, routing is done in several dimensions (x,y,z). If routing is constrained to the x dimension, then y, then z (dimension ordered routing), topology related dependency loops are avoided in these structures.

In addition, a processing element design shall not form dependency links between its input and output ports. A dependency link between input and output ports occurs if a processing element is unable to accept an input packet until a waiting packet can be issued from the output port.

RapidIO supports operations, such as read operations, that require responses to complete. These operations can lead to a dependency link between a processing element's input port and output port.

As an example of a input to output port dependency, consider a processing element where the output port queue is full. The processing element can not accept a new request at its input port since there is no place to put the response in the output port queue. No more transactions can be accepted at the input port until the output port is able to free entries in the output queue by issuing packets to the system.

The method by which a RapidIO system maintains a deadlock free environment is described in the appropriate Physical Layer specification.

2 Chapter 2 - Operation Descriptions

This chapter describes the set of operations and their associated transactions supported by the I/O protocols of RapidIO. The transaction types, packet formats, and other necessary transaction information are described in Chapter 3, "Packet Format Descriptions."

The I/O operation protocols work using request/response transaction pairs through the interconnect fabric. A processing element sends a request transaction to another processing element if it requires an activity to be carried out. The receiving processing element responds with a response transaction when the request has been completed or if an error condition is encountered. Each transaction is sent as a packet through the interconnect fabric. For example, a processing element that requires data from another processing element sends an NREAD transaction in a request packet to that processing element, which reads its local memory at the requested address and returns the data in a DONE transaction in a response packet. Note that not all requests require responses; some requests assume that the desired activity will complete properly.

Two possible response transactions can be received by a requesting processing element:

- A DONE response indicates to the requestor that the desired transaction has completed and it also returns data for read-type transactions as described above.
- An ERROR response means that the target of the transaction encountered an unrecoverable error and could not complete the transaction.

Packets may contain additional information that is interpreted by the interconnect fabric to route the packets through the fabric from the source to the destination, such as a device number. These requirements are described in the appropriate RapidIO transport layer specification, and are beyond the scope of this specification.

Depending upon the interconnect fabric, other packets may be generated as part of the physical layer protocol to manage flow control, errors, etc. Flow control and other fabric-specific communication requirements are described in the appropriate RapidIO transport and physical layer specifications and are beyond the scope of this document.

For most transaction types, a request transaction sent into the system is marked with a transaction ID that is unique for each requestor and responder processing element pair. This transaction ID allows a response to be easily matched to the original request when it is returned to the requestor. An end point cannot reuse a transaction ID value to the same destination until the response from the original transaction has been received by the requestor. The number of outstanding transactions that may be supported is implementation dependent.

Transaction IDs may also be used to indicate sequence information if ordered reception of transactions is required by the destination processing element and the interconnect fabric can reorder packets. The receiving device can either retry subsequent out-of-order requests, or it can accept and not complete the subsequent out-of-order requests until the missing transactions in the sequence have been received and completed.

2.1 I/O Operations Cross Reference

Table 2-1 contains a cross reference of the I/O operations defined in this RapidIO specification and their system usage.

Table 2-1. I/O Operations Cross Reference

Operation	Transactions Used	Possible System Usage	Request Transaction Classification for Completion Ordering Rules	Description	Packet Format
Read	NREAD, RESPONSE	Read operation	Read	Section 2.2.1	Type 2 Section 3.1.5
Write	NWRITE	Write operation	Write	Section 2.2.2	Type 5 Section 3.1.7
Write-with-response	NWRITE_R, RESPONSE	Write operation	Write	Section 2.2.3	Type 5 Section 3.1.7
Streaming-write	SWRITE	Write operation	Write	Section 2.2.2	Type 6 Section 3.1.8
Atomic (read-modify-write)	ATOMIC, RESPONSE	Read-modify-write operation	Write	Section 2.2.4	Type 2 Section 3.1.5 Type 5 Section 3.1.7
Maintenance	MAINTENANCE	System exploration, initialization, and maintenance operation	not applicable	Section 2.3.1	Type 8 Section 3.1.10

2.2 I/O Operations

The operations described in this section are used for I/O accesses to physical addresses in the target of the operation. Examples are accesses to non-coherent memory, ROM boot code, or to configuration registers that do not participate in any globally shared system memory protocol. These accesses may be of any specifiable size allowed by the system.

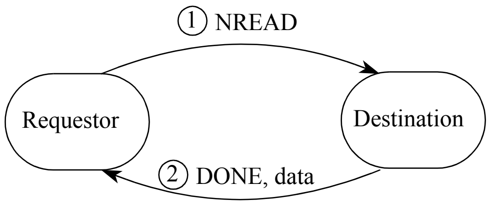
All data payloads that are less than 8 bytes shall be padded and have their bytes aligned to their proper byte position within the double-word, as in the examples shown in Figure 2-6 throughFigure 2-8.

The described behaviors are the same regardless of the actual target physical address.

2.2.1 Read Operations

The read operation, consisting of the NREAD and RESPONSE transactions (typically a DONE response) as shown in Figure 2-1 is used by a processing element that needs to read data from the specified address. The data returned is of the size requested.

If the read operation is to memory, data is returned from the memory regardless of the state of any system-wide cache coherence mechanism for the specified cache line or lines, although it may cause a snoop of any caches local to the memory controller.



Fgure 2-1. Read Operation

2.2.2 Write and Streaming-Write Operations

The write and streaming-write operations, consisting of the NWRITE and SWRITE transactions as shown in Figure 2-2, are used by a processing element that needs to write data to the specified address. The NWRITE transaction allows multiple double-word, word, half-word and byte writes with properly padded and aligned (to the 8-byte boundary) data payload. The SWRITE transaction is a double-word-only version of the NWRITE that has less header overhead. The write size and alignment for the NWRITE transaction are specified in Table 3-4. Non-contiguous and unaligned writes are not supported. It is the requestor's responsibility to break up a write operation into multiple transactions if the block is not aligned.

NWRITE and SWRITE transactions do not receive responses, so there is no notification to the sender when the transaction has completed at the destination.

If the write operation is to memory, data is written to the memory regardless of the state of any system-wide cache coherence mechanism for the specified cache line or lines, although it may cause a snoop of any caches local to the memory controller.

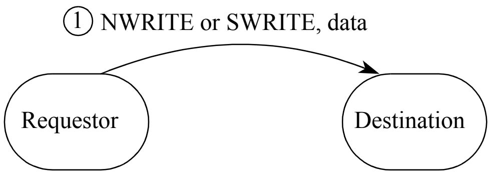


Figure 2-2. Write and Streaming-Write Operations

2.2.3 Write-With-Response Operations

The write-with-response operation, consisting of the NWRITE_R and RESPONSE transactions (typically a DONE response) as shown in Figure 2-3, is identical to the write operation except that it receives a response to notify the sender that the write has completed at the destination. This operation is useful for guaranteeing read-after-write and write-after-write ordering through a system that can reorder transactions and for enforcing other required system behaviors.

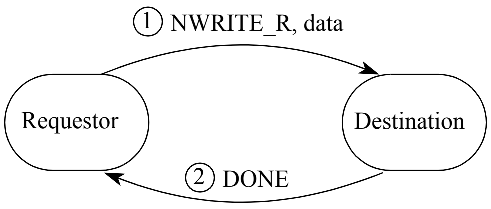


Figure 2-3. Write-With-Response Operation

2.2.4 Atomic (Read-Modify-Write) Operations

The read-modify-write operation, consisting of the ATOMIC and RESPONSE transactions (typically a DONE response) as shown in Figure 2-4 is used by a number of cooperating processing elements to perform synchronization using non-coherent memory. The allowed specified data sizes are one word (4 bytes), one half-word (2 bytes) or one byte, with the size of the transaction specified in the same way as for an NWRITE transaction. Double-word (8-byte) and 3, 5, 6, and 7 byte ATOMIC transactions may not be specified.

The atomic operation is a combination read and write operation. The destination reads the data at the specified address, returns the read data to the requestor, performs the required operation to the data, and then writes the modified data back to the specified address without allowing any intervening activity to that address. Defined operations are increment, decrement, test-and-swap, set, and clear (See bit settings in Table 4-9 and Table 4-10). Of these, only test-and-swap require the requesting processing element to supply data. The target data of an atomic operation may be initialized using an NWRITE transaction.

If the atomic operation is to memory, data is written to the memory regardless of the state of any system-wide cache coherence mechanism for the specified cache line or lines, although it may cause a snoop of any caches local to the memory controller.

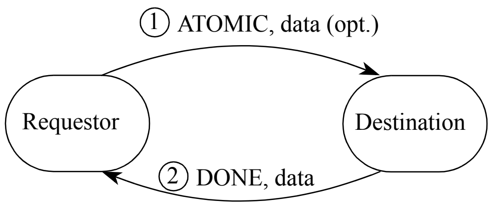


Figure 2-4. Atomic (Read-Modify-Write) Operation

2.3 System Operations

All data payloads that are less than 8 bytes shall be padded and have their bytes aligned to their proper byte position within the double-word, as in the examples shown in Figure 2-6 through Figure 2-8.

2.3.1 Maintenance Operations

The maintenance operation, which can consist of more than one MAINTENANCE transaction as shown in Figure 2-5 is used by a processing element that needs to read or write data to the specified CARs, CSRs, or locally-defined registers or data structures. If a response is required, MAINTENANCE requests receive a MAINTENANCE response rather than a normal response for both read and write operations. Supported accesses are in 32 bit quantities and may optionally be in double-word and multiple double-word quantities to a maximum of 64 bytes.

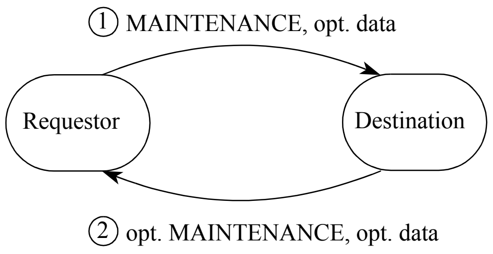
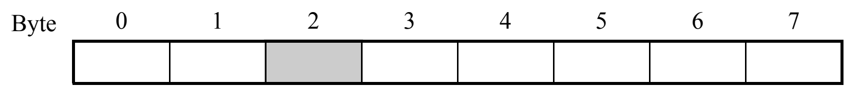


Figure 2-5. Maintenance Operation

2.4 Endian, Byte Ordering, and Alignment

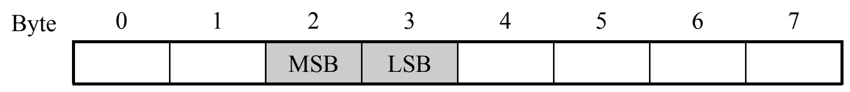
RapidIO has double-word (8-byte) aligned big-endian data payloads. This means that the RapidIO interface to devices that are little-endian shall perform the proper endian transformation to format a data payload.

Operations that specify data quantities that are less than 8 bytes shall have the bytes aligned to their proper byte position within the big-endian double-word, as in the examples shown in Figure 2-6 through Figure 2-8.



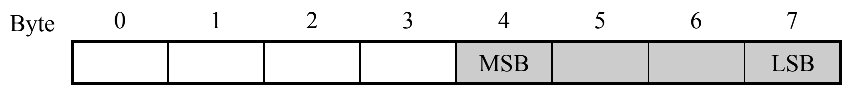
Byte address 0x0000_0002, the proper byte position is shaded.

Figure 2-6. Byte Alignment Example



Half-word address 0x0000_0002, the proper byte positions are shaded.

Figure 2-7. Half-Word Alignment Example



Word address 0x0000_0004, the proper byte positions are shaded.

Figure 2-8. Word Alignment Example

For write operations, a processing element shall properly align data transfers to a double-word boundary for transmission to the destination. This alignment may require breaking up a data stream into multiple transactions if the data is not naturally aligned. A number of data payload sizes and double-word alignments are defined to minimize this burden.
Figure 2-9 shows a 48-byte data stream that a processing element wishes to write to another processing element through the interconnect fabric. The data displayed in the figure is big-endian and double-word aligned with the bytes to be written shaded in grey. Because the start of the stream and the end of the stream are not aligned to a double-word boundary, the sending processing element shall break the stream into three transactions as shown in the figure.

The first transaction sends the first three bytes (in byte lanes 5, 6, and 7) and indicates a byte lane 5, 6, and 7 three-byte write. The second transaction sends all of the remaining data except for the final sub-double-word. The third transaction sends the final 5 bytes in byte lanes 0, 1, 2, 3, and 4 indicating a five-byte write in byte lanes 0, 1, 2, 3, and 4.

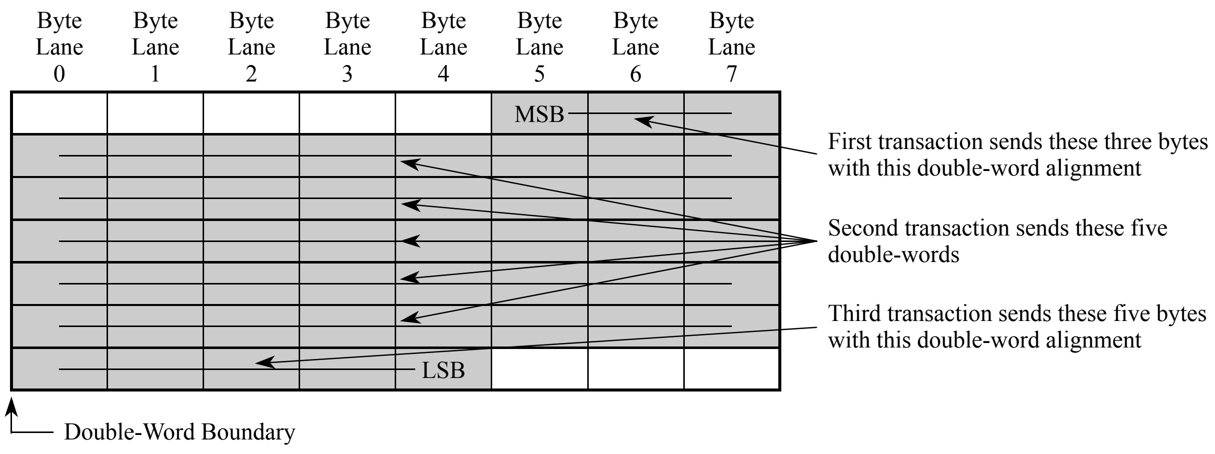

Byte Lane 0
Byte Lane 1
Byte Lane 2
Byte Lane 3
Byte Lane 4
Byte Lane 5
Byte Lane 6
Byte Lane 7
MSB
LSB
First transaction sends these three bytes with this double-word alignment
Second transaction sends these five double-words
Third transaction sends these five bytes with this double-word alignment
Double-Word Boundary

Figure 2-9. Data Alignment Example

3 Chapter 3 - Packet Format Descriptions

This chapter contains the packet format definitions for the RapidIO Input/Output Logical Specification. Four types of I/O packet formats exist:

- Request
- Response
- Implementation-defined
- Reserved

The packet formats are intended to be interconnect fabric independent so the system interconnect can be anything required for a particular application. Reserved formats, unless defined in another logical specification, shall not be used by a device.

3.1 Request Packet Formats

A request packet is issued by a processing element that needs a remote processing element to accomplish some activity on its behalf, such as a memory read operation. The request packet format types and their transactions for the *RapidIO Input/Output Logical Specification* are shown in Table 3-1 below.

Table 3-1. Request Packet Type to Transaction Type Cross Reference

Request Packet Format Type	Transaction Type	Definition	Document Section No.
Type 0	Implementation-defined	Defined by the device implementation	Section 3.1.3
Type 1	—	Reserved	Section 3.1.4
Type 2	ATOMIC	Read-modify-write operation on specified address	Section 3.1.5
	NREAD	Read specified address	
Type 3-4	—	Reserved	Section 3.1.6
Type 5	ATOMIC test-and-swap	Read-test=0-swap-write operation on specified address	Section 3.1.7
	NWRITE	Write specified address	
	NWRITE_R	Write specified address, notify source of completion	
Type 6	SWRITE	Write specified address	Section 3.1.8
Type 7	—	Reserved	Section 3.1.9
Type 8	MAINTENANCE	Read or write device configuration registers and perform other system maintenance tasks	Section 3.1.10
Type 9-11	—	Reserved	Section 3.1.11

3.1.1 Addressing and Alignment

The size of the address is defined as a system-wide parameter; thus the packet formats do not support mixed local physical address fields simultaneously. The least three significant bits of all addresses are not specified and are assumed to be logic 0.

All transactions are aligned to a byte, half-word, word, or double-word boundary. Read and write request addresses are aligned to any specifiable double-word boundary and are not aligned to the size of the data written or requested. Data payloads start at the first double-word and proceed linearly through the address space. Sub-double-word data payloads shall be padded and properly aligned within the 8-byte boundary. Non-contiguous or unM0aligned transactions that would ordinarily require a byte mask are not supported. A sending device that requires this behavior shall break the operation into multiple request transactions. An example of this is shown in Section 2.4, “Endian, Byte Ordering, and Alignment.”

3.1.2 Field Definitions for All Request Packet Formats

Table 3-2 through Table 3-4 describe the field definitions for all request packet formats. Bit fields that are defined as "reserved" shall be assigned to logic 0s when generated and ignored when received. Bit field encodings that are defined as "reserved" shall not be assigned when the packet is generated. A received reserved encoding is regarded as an error if a meaningful encoding is required for the transaction and function, otherwise it is ignored. Implementation-defined fields shall be ignored unless the encoding is understood by the receiving device. All packets described are bit streams from the first bit to the last bit, represented in the figures from left to right respectively.

Table 3-2. General Field Definitions for All Request Packets

Field	Definition
ftype	Format type, represented as a 4-bit value; is always the first four bits in the logical packet stream.
wdptr	Word pointer, used in conjunction with the data size (rdsize and wrsize) fields—see Table 3.3, Table 3.4 and Section 2.4.
rdsize	Data size for read transactions, used in conjunction with the word pointer (wdptr) bit—see Table 3.3 and Section 2.4.
wrsize	Write data size for sub-double-word transactions, used in conjunction with the word pointer (wdptr) bit—see Table 3.4 and Section 2.4. For writes greater than one double-word, the size is the maximum payload that should be expected by the receiver.
rsrv	Reserved
srcTID	The packet's transaction ID
transaction	The specific transaction within the format class to be performed by the recipient; also called type or ttype.
extended address	Optional. Specifies the most significant 16 bits of a 50-bit physical address or 32 bits of a 66-bit physical address.
xamsbs	Extended address most significant bits. Further extends the address specified by the address and extended address fields by 2 bits. This field provides 34-, 50-, and 66-bit addresses to be specified in a packet with the xamsbs as the most significant bits in the address.
address	Least significant 29 bits (bits [0-28] of byte address [0-31]) of the double-word physical address

Table 3-3. Read Size (rdsize) Definitions

wdptr	rdsize	Number of Bytes	Byte Lanes
0b0	0b0000	1	0b10000000
0b0	0b0001	1	0b01000000
0b0	0b0010	1	0b00100000
0b0	0b0011	1	0b00010000
0b1	0b0000	1	0b00001000
0b1	0b0001	1	0b00000100
0b1	0b0010	1	0b00000010
0b1	0b0011	1	0b00000001
0b0	0b0100	2	0b11000000
0b0	0b0101	3	0b11100000
0b0	0b0110	2	0b00110000
0b0	0b0111	5	0b11111000
0b1	0b0100	2	0b00001100
0b1	0b0101	3	0b00000111
0b1	0b0110	2	0b00000011

Table 3-3. Read Size (rdsize) Definitions(Continued)

wdptr	rdsize	Number of Bytes	Byte Lanes
0b1	0b0111	5	0b00011111
0b0	0b1000	4	0b11110000
0b1	0b1000	4	0b00001111
0b0	0b1001	6	0b11111100
0b1	0b1001	6	0b00111111
0b0	0b1010	7	0b11111110
0b1	0b1010	7	0b01111111
0b0	0b1011	8	0b11111111
0b1	0b1011	16	
0b0	0b1100	32	
0b1	0b1100	64	
0b0	0b1101	96	
0b1	0b1101	128	
0b0	0b1110	160	
0b1	0b1110	192	
0b0	0b1111	224	
0b1	0b1111	256	

Table 3-4. Write Size (wrsize) Definitions

wdptr	wrsize	Number of Bytes	Byte Lanes
0b0	0b0000	1	0b10000000
0b0	0b0001	1	0b01000000
0b0	0b0010	1	0b00100000
0b0	0b0011	1	0b00010000
0b1	0b0000	1	0b00001000
0b1	0b0001	1	0b00000100
0b1	0b0010	1	0b00000010
0b1	0b0011	1	0b00000001
0b0	0b0100	2	0b11000000
0b0	0b0101	3	0b11100000
0b0	0b0110	2	0b00110000
0b0	0b0111	5	0b11111000
0b1	0b0100	2	0b00001100
0b1	0b0101	3	0b00000111
0b1	0b0110	2	0b00000011
0b1	0b0111	5	0b00011111
0b0	0b1000	4	0b11110000

Table 3-4. Write Size (wrsize) Definitions(Continued)

wdptr	wrsize	Number of Bytes	Byte Lanes
0b1	0b1000	4	0b00001111
0b0	0b1001	6	0b11111100
0b1	0b1001	6	0b00111111
0b0	0b1010	7	0b11111110
0b1	0b1010	7	0b01111111
0b0	0b1011	8	0b11111111
0b1	0b1011	16 maximum	
0b0	0b1100	32 maximum	
0b1	0b1100	64 maximum	
00b	0b1101	reserved	
0b1	0b1101	128 maximum	
0b0	0b1110	reserved	
0b1	0b1110	reserved	
0b0	0b1111	reserved	
0b1	0b1111	256 maximum	

3.1.3 Type 0 Packet Format (Implementation-Defined)

The type 0 packet format is reserved for implementation-defined functions such as flow control.

3.1.4 Type 1 Packet Format (Reserved)

The type 1 packet format is reserved.

3.1.5 Type 2 Packet Format (Request Class)

The type 2 format is used for the NREAD and ATOMIC transactions as specified in the transaction field defined in Table 3-5. Type 2 packets never contain a data payload.

The data payload size for the response to an ATOMIC transaction is 8 bytes. The addressing scheme defined for the read portion of the ATOMIC transaction also controls the size of the atomic operation in memory so the bytes shall be contiguous and shall be of size byte, half-word (2 bytes), or word (4 bytes), and be aligned to that boundary and byte lane as with a regular read transaction. Double-word (8-byte), 3, 5, 6, and 7 byte ATOMIC transactions are not allowed.

Note that type 2 packets don’t have any special fields.

Table 3-5. Transaction Fields and Encodings for Type 2 Packets

Encoding	Transaction Field
0b0000–0011	Reserved
0b0100	NREAD transaction
0b0101–1011	Reserved
0b1100	ATOMIC inc: post-increment the data
0b1101	ATOMIC dec: post-decrement the data

Figure 3-1 displays the type 2 packet with all its fields. The field value 0b0010 in Figure 3-1 specifies that the packet format is of type 2.

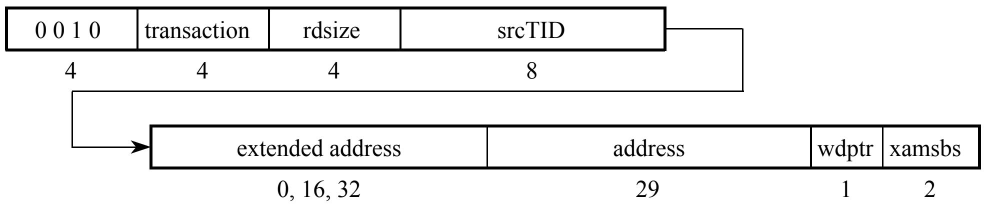


Figure 3-1. Type 2 Packet Bit Stream Format

3.1.6 Type 3–4 Packet Formats (Reserved)

The type 3–4 packet formats are reserved.

3.1.7 Type 5 Packet F3.1.7 ormat (Write Class)

Type 5 packets always contain a data payload. A data payload that consists of a single double-word or less has sizing information as defined in Table 3-4. The wrsize field specifies the maximum size of the data payload for multiple double-word transactions. The data payload may not exceed that size but may be smaller if desired. The ATOMIC, NWRITE, and NWRITE_R transactions use the type 5 format as defined in Table 3-6. NWRITE request packets do not require a response. Therefore, the transaction ID (srcTID) field for a NWRITE request is undefined and may have an arbitrary value.

The ATOMIC test-and-swap transaction is limited to one double-word (8 bytes) of data payload. The addressing scheme defined for the write transactions also controls the size of the atomic operation in memory so the bytes shall be contiguous and shall be of size byte, half-word (2 bytes), or word (4 bytes), and be aligned to that boundary and byte lane as with a regular write transaction. Double-word (8-byte) and 3, 5, 6, and 7 byte ATOMIC test-and-swap transactions are not allowed.

Note that type 5 packets don't have any special fields.

Table 3-6. Transaction Fields and Encodings for Type 5 Packets

Encoding	Transaction Field
0b0000–0011	Reserved
0b0100	NWRITE transaction
0b0101	NWRITE_R transaction
0b0110–1101	Reserved
0b1110	ATOMIC test-and-swap: read and return the data, compare to 0, write with supplied data if compare is true
0b1111	Reserved

Figure 3-2 displays the type 5 packet with all its fields. The field value 0b0101 in Figure 3-2 specifies that the packet format is of type 5-

.

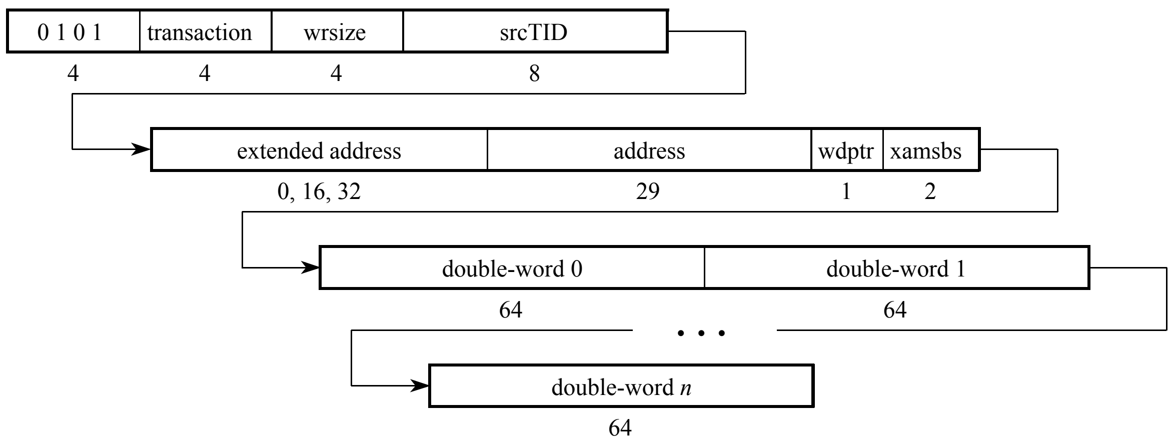


Figure 3-2. Type 5 Packet Bit Stream Format

3.1.8 Type 6 Packet Format (Streaming-Write Class)

The type 6 packet is a special-purpose type that always contains data. The data payload always contains a minimum of one complete double-word. Sub-double-word data payloads shall use the type 5 NWRITE transaction. Type 6 transactions may contain any number of double-words up to the maximum defined in Table 3-4.

Because the SWRITE transaction is the only transaction to use format type 6, there is no need for the transaction field within the packet. There are also no size or transaction ID fields.

Figure 3-3 displays the type 6 packet with all its fields. The field value 0b0110 in Figure 3-3 specifies that the packet format is of type 6.

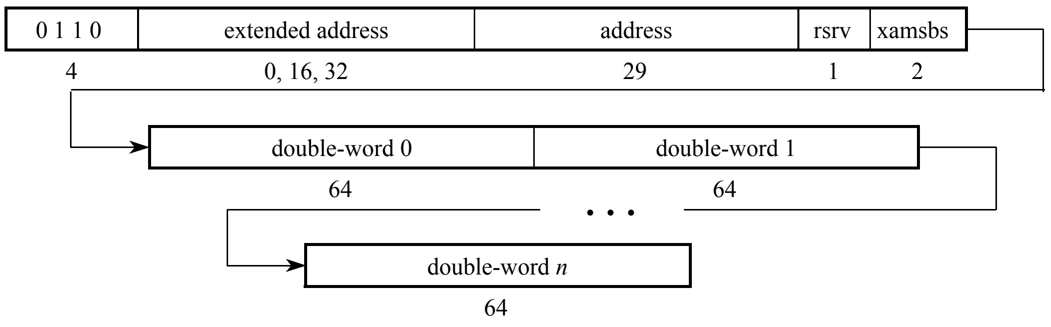


Figure 3-3. Type 6 Packet Bit Stream Format

3.1.9 Type 7 Packet Format (Reserved)

The type 7 packet format is reserved.

3.1.10 Type 8 Packet Format (Maintenance Class)

The type 8 MAINTENANCE packet format is used to access the RapidIO capability and status registers (CARs and CSRs) and data structures. Unlike other request formats, the type 8 packet format serves as both the request and the response format for maintenance operations. Type 8 packets contain no addresses and only contain data payloads for write requests and read responses. All configuration register read accesses are performed in word (4-byte), and optionally double-word (8-byte) or specifiable multiple double-word quantities up to a limit of 64 bytes. All register write accesses are also performed in word (4-byte), and optionally double-word (8-byte) or multiple double-word quantities up to a limit of 64 bytes.

Read and write data sizes are specified as shown in Table 3-3 and Table 3-4. The wrsize field specifies the maximum size of the data payload for multiple double-word transactions. The data payload may not exceed that size but may be smaller if desired. Both the maintenance read and the maintenance write request generate the appropriate maintenance response.

The maintenance port-write operation is a write operation that does not have guaranteed delivery and does not have an associated response. This maintenance operation is useful for sending messages such as error indicators or status information from a device that does not contain an end point, such as a switch. The data payload is typically placed in a queue in the targeted end point and an interrupt is typically generated to a local processor. A port-write request to a queue that is full or busy servicing another request may be discarded.

Definitions and encodings of fields specific to type 8 packets are provided in Table 3-7. Fields that are not specific to type 8 packets are described in Table 3-2.

Figure 3-7. Specific Field Definitions and Encodings for Type 8 Packets

Type 8 Fields	Encoding	Definition
transaction	0b0000	Specifies a maintenance read request
	0b0001	Specifies a maintenance write request
	0b0010	Specifies a maintenance read response
	0b0011	Specifies a maintenance write response
	0b0100	Specifies a maintenance port-write request
	0b0101–1111	Reserved
config_offset	—	Double-word offset into the CAR/CSR register block for reads and writes
srcTID	—	The type 8 request packet’s transaction ID (reserved for port-write requests)
targetTID	—	The corresponding type 8 response packet’s transaction ID
status	0b0000	DONE—Requested transaction has completed successfully
	0b0001–0110	Reserved
	0b0111	ERROR—Unrecoverable error detected
	0b1000–1011	Reserved
	0b1100–1111	Implementation-defined—Can be used for additional information such as an error code

Figure 3-4 displays a type 8 request (read or write) packet with all its fields. The field value 0b1000 in Figure 3-4 specifies that the packet format is of type 8. The srcTID and config_offset fields are reserved for port-write requests.

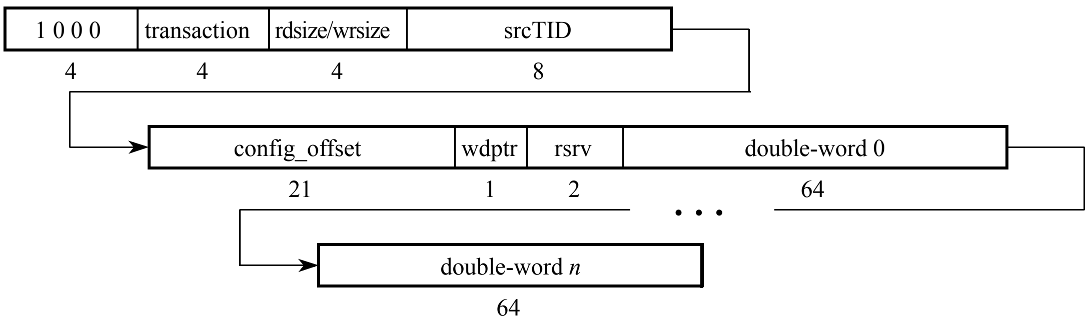


Figure 3-4. Type 8 Request Packet Bit Stream Format

Figure 3.5 displays a type 8 response packet with all its fields.

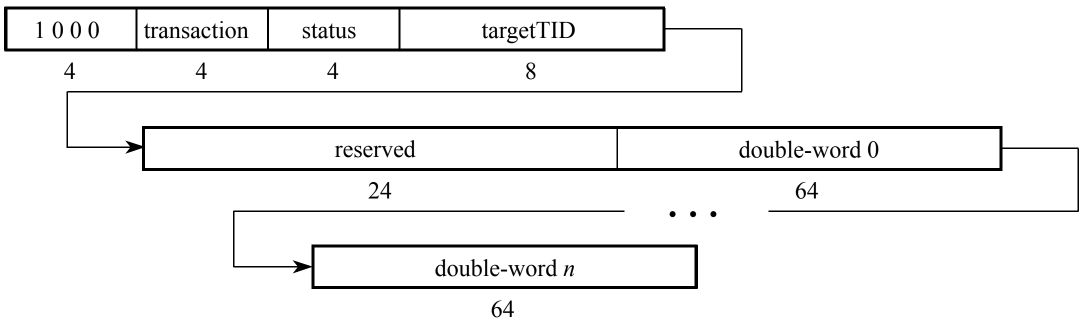


Figure 3-5. Type 8 Response Packet Bit Stream Format

3.1.11 Type 9–11 Packet Formats (Reserved)

The type 9–11 packet formats are reserved.

3.2 Response Packet Formats

A response transaction is issued by a processing element when it has completed a request made to it by a remote processing element. Response packets are always directed and are transmitted in the same way as request packets. Currently two packet format types exist, as shown in Table 3-8..

Table 3-8. Response Packet Type to Transaction Type Cross Reference

Response Packet Format Type	Transaction Type	Definition	Document Section Number
Type 12	—	Reserved	Section 3.2.2
Type 13	RESPONSE	Issued by a processing element when it completes a request by a remote element.	Section 3.2.3
Type 14	—	Reserved	Section 3.2.4
Type 15	Implementation-defined	Defined by the device implementation	Section 3.2.5

3.2.1 Field Definitions for All Response Packet Formats

The field definitions in Table 3-9 apply to more than one of the response packet formats.

Table 3-9. Field Definitions and Encodings for All Response Packets

Field	Encoding	Sub-Field	Definition
transaction	0b0000		RESPONSE transaction with no data payload
	0b0001–0111		Reserved
	0b1000		RESPONSE transaction with data payload
	0b1001–1111		Reserved
targetTID	—		The corresponding request packet's transaction ID
status	Type of status and encoding		
	0b0000	DONE	Requested transaction has been successfully completed
	0b0001–0110	—	Reserved
	0b0111	ERROR	Unrecoverable error detected
	0b1000–1011	—	Reserved
	0b1100–1111	Implementation	Implementation defined—Can be used for additional information such as an error code

3.2.2 Type 12 Packet Format (Reserved)

The type 12 packet format is reserved.

3.2.3 Type 13 Packet Format (Response Class)

The type 13 packet format returns status, data (if required), and the requestor's transaction ID. A RESPONSE packet with an "ERROR" status or a response that is not expected to have a data payload never has a data payload. The type 13 format is used for response packets to all request packets except maintenance and response-less writes.

Note that type 13 packets do not have any special fields.

Figure 3-6 illustrates the format and fields of type 13 packets. The field value 0b1101 in Figure 3-6 specifies that the packet format i is of type 13 .

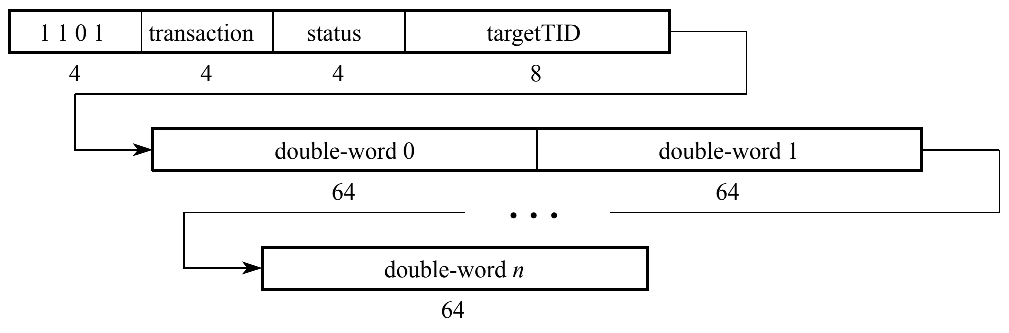


Figure 3-6. Type 13 Packet Bit Stream Format

3.2.4 Type 14 Packet Format (Reserved)

The type 14 packet format is reserved.

- 3.2.5 Type 15 Packet Format (Implementation-Defined)

The type 15 packet format is reserved for implementation-defined functions such as flow control.

4 Chapter 4 - Input/Output Registers

This chapter describes the visible register set that allows an external processing element to determine the capabilities, configuration, and status of a processing element using this logical specification. This chapter only describes registers or register bits defined by this specification. Refer to the other RapidIO logical, transport, and physical specifications of interest to determine a complete list of registers and bit definitions. All registers are 32-bits and aligned to a 32-bit boundary.

4.1 Register Summary

Table 4-1 shows the register map for this RapidIO specification. These capability registers (CARs) and command and status registers (CSRs) can be accessed using RapidIO maintenance operations. Any register offsets not defined are considered reserved for this specification unless otherwise stated. Other registers required for a processing element are defined in other applicable RapidIO specifications and by the requirements of the specific device and are beyond the scope of this specification. Read and write accesses to reserved register offsets shall terminate normally and not cause an error condition in the target device. Writes to CAR (read-only) space shall terminate normally and not cause an error condition in the target device.

Register bits defined as reserved are considered reserved for this specification only. Bits that are reserved in this specification may be defined in another RapidIO specification.

Table 4-1. I/O Register Map

Configuration Space Byte Offset	Register Name (Word 0)	Register Name (Word 1)
0x0	Device Identity CAR	Device Information CAR
0x8	Assembly Identity CAR	Assembly Information CAR
0x10	Processing Element Features CAR	Switch Port Information CAR
0x18	Source Operations CAR	Destination Operations CAR
0x20–38	Reserved	
0x40	Reserved	Write Port CSR
0x48	Reserved	Processing Element Logical Layer Control CSR
0x50	Reserved	
0x58	Local Configuration Space High Base Address CSR	Local Configuration Space Base Address CSR
0x60–F8	Reserved	
0x100–FFF8	Extended Features Space	
0x10000–FFFFF8	Implementation-defined Space	

4.2 Reserved Register and Bit Behavior

Table 4-2 describes the required behavior for accesses to reserved register bits and reserved registers for the RapidIO register space.

Table 4-2. Configuration Space Reserved Access Behavior

Byte Offset	Space Name	Item	Initiator behavior	Target behavior
0x0–3C	Capability Register Space (CAR Space - this space is read-only)	Reserved bit	read - ignore returned value[1]	read - return logic 0
			write -	write - ignored
		Implementation-defined bit	read - ignore returned value unless implementation-defined function understood	read - return implementation-defined value
			write -	write - ignored
		Reserved register	read - ignore returned value	read - return logic 0s
			write -	write - ignored
0x40–FC	Command and Status Register Space (CSR Space)	Reserved bit	read - ignore returned value	read - return logic 0
			write - preserve current value[2]	write - ignored
		Implementation-defined bit	read - ignore returned value unless implementation-defined function understood	read - return implementation-defined value
			write - preserve current value if implementation-defined function not understood	write - implementation-defined
		Reserved register	read - ignore returned value	read - return logic 0s
			write -	write - ignored
0x100–FFFC	Extended Features Space	Reserved bit	read - ignore returned value	read - return logic 0
			write - preserve current value	write - ignored
		Implementation-defined bit	read - ignore returned value unless implementation-defined function understood	read - return implementation-defined value
			write - preserve current value if implementation-defined function not understood	write - implementation-defined
		Reserved register	read - ignore returned value	read - return logic 0s
			write -	write - ignored
0x10000–FFFFFC	Implementation-defined Space	Reserved bit and register	All behavior implementation-defined	

1. Do not depend on reserved bits being a particular value; use appropriate masks to extract defined bits from the read value.
2. All register writes shall be in the form: read the register to obtain the values of all reserved bits, merge in the desired values for defined bits to be modified, and write the register, thus preserving the value of all reserved bits.

4.3 Extended Features Data Structure

The RapidIO capability and command and status registers implement an extended capability data structure. If the extended features bit (bit 28) in the processing element features register is set, the extended features pointer is valid and points to the first entry in the extended features data structure. This pointer is an offset into the standard 16 Mbyte capability register (CAR) and command and status register (CSR) space and is accessed with a maintenance read operation in the same way as when accessing CARs and CSRs.

The extended features data structure is a singly linked list of double-word structures. Each of these contains a pointer to the next structure (EF_PTR) and an extended feature type identifier (EF_ID). The end of the list is determined when the next extended feature pointer has a value of logic 0. All pointers and extended features blocks shall index completely into the extended features space of the CSR space, and all shall be aligned to a double-word boundary so the three least significant bits shall equal logic 0. Pointer values not in extended features space or improperly aligned are illegal and shall be treated as the end of the data structure. Figure 4-1 shows an example of an extended features data structure. It is required that the extended features bit is set to logic 1 in the processing element features register.

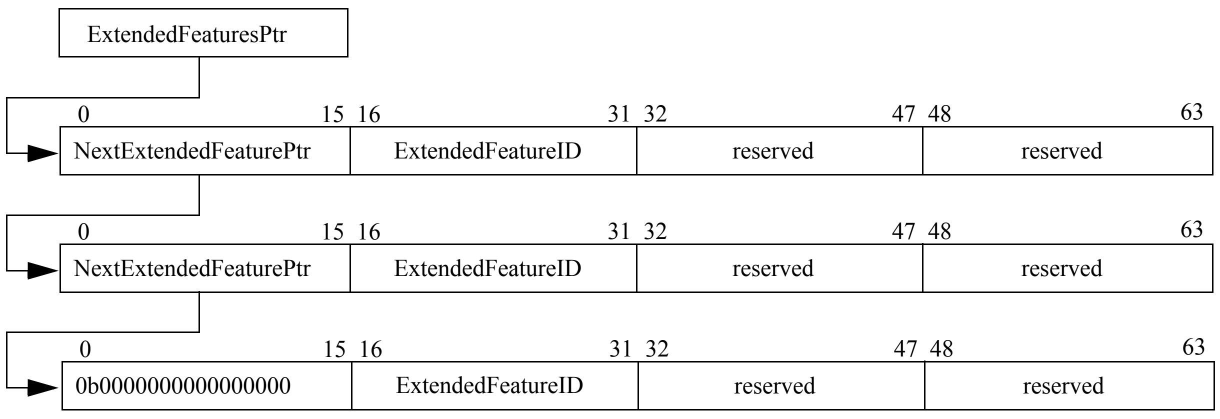


Figure 4-1. Example Extended Features Data Structure

4.4 Capability Registers (CARs)

Every processing element shall contain a set of registers that allows an external processing element to determine its capabilities through maintenance read operations. All registers are 32 bits wide and are organized and accessed in 32-bit (4 byte) quantities, although some processing elements may optionally allow larger accesses. CARs are read-only. Refer to Table 4-2 for the required behavior for accesses to reserved registers and register bits.

CARs are big-endian with bit 0 and Word 0 respectively the most significant bit and word.

4.4.1 Device Identity CAR (Offset 0x0 Word 0)

The DeviceVendorIdentity field identifies the vendor that manufactured the device containing the processing element. A value for the DeviceVendorIdentity field is uniquely assigned to a device vendor by the registration authority of the RapidIO Trade Association.

The DeviceIdentity field is intended to uniquely identify the type of device from the vendor specified by the DeviceVendorIdentity field. The values for the DeviceIdentity field are assigned and managed by the respective vendor. See Table 4-3.

Table 4-3. Bit Settings for Device Identity CAR

Bit	Field Name	Description
0–15	DeviceIdentity	Device identifier
16–31	DeviceVendorIdentity	Device vendor identifier

4.4.2 Device Information CAR (Offset 0x0 Word 1)

The DeviceRev field is intended to identify the revision level of the device. The value for the DeviceRev field is assigned and managed by the vendor specified by the DeviceVendorIdentity field. See Table 4-4

Table 4-4. Bit Settings for Device Information CAR

Bit	Field Name	Description
0-31	DeviceRev	Device revision level

4.4.3 Assembly Identity CAR (Offset 0x8 Word 0)

The AssyVendorIdentity field identifies the vendor that manufactured the assembly or subsystem containing the device. A value for the AssyVendorIdentity field is uniquely assigned to a assembly vendor by the registration authority of the RapidIO Trade Association.

The AssyIdentity field is intended to uniquely identify the type of assembly from the vendor specified by the AssyVendorIdentity field. The values for the AssyIdentity field are assigned and managed by the respective vendor. See Table 4-5.

Table 4-5. Bit Settings for Assembly Identity CAR

Bit	Field Name	Description
0–15	AssyIdentity	Assembly identifier
16–31	AssyVendorIdentity	Assembly vendor identifier

4.4.4 Assembly Information CAR (Offset 0x8 Word 1)

This register contains additional information about the assembly; see Table 4-6.

Table 4-6. Bit Settings for Assembly Information CAR

Bit	Field Name	Description
0–15	AssyRev	Assembly revision level
16–31	ExtendedFeaturesPtr	Pointer to the first entry in the extended features list

4.4.5 Processing Element Features CAR (Offset 0x10 Word 0)

This register identifies the major functionality provided by the processing element; see Table 4-7

Table 4-7. Bit Settings for Processing Element Features CAR

Bit	Field Name	Description
0	Bridge	PE can bridge to another interface. Examples are PCI, proprietary processor buses, DRAM, etc.
1	Memory	PE has physically addressable local address space and can be accessed as an end point through non-maintenance (i.e. non-coherent read and write) operations. This local address space may be limited to local configuration registers, or could be on-chip SRAM, etc.
2	Processor	PE physically contains a local processor or similar device that executes code. A device that bridges to an interface that connects to a processor does not count (see bit 0 above).
3	Switch	PE can bridge to another external RapidIO interface - an internal port to a local end point does not count as a switch port. For example, a device with two RapidIO ports and a local end point is a two port switch, not a three port switch, regardless of the internal architecture.
4–27	—	Reserved
28	Extended features	PE has extended features list; the extended features pointer is valid
29-31	Extended addressing support	Indicates the number address bits supported by the PE both as a source and target of an operation. All PEs shall at minimum support 34 bit addresses. 0b111 - PE supports 66, 50, and 34 bit addresses 0b101 - PE supports 66 and 34 bit addresses 0b011 - PE supports 50 and 34 bit addresses 0b001 - PE supports 34 bit addresses All other encodings reserved

4.4.6 Switch Port Information CAR (Offset 0x10 Word 1)

This register defines the switching capabilities of a processing element. This register is only valid if bit 3 is set in the processing element features CAR; see Table 4-8

Table 4-8. Bit Settings for Switch Port Information CAR

Bit	Field Name	Description
0–15	—	Reserved
16–23	PortTotal	The total number of RapidIO ports on the processing element 0b00000000 - Reserved 0b00000001 - 1 port 0b00000010 - 2 ports 0b00000011 - 3 ports 0b00000100 - 4 ports ... 0b11111111 - 255 ports
24–31	PortNumber	This is the port number from which the maintenance read operation accessed this register. Ports are numbered starting with 0x00.

4.4.7 Source Operations CAR (Offset 0x18 Word 0)

This register defines the set of RapidIO IO logical operations that can be issued by this processing element; see Table 4-9. It is assumed that a processing element can generate I/O logical maintenance read and write requests if it is required to access CARs and CSRs in other processing elements. The Source Operations CAR is applicable for end point devices only. RapidIO switches shall be able to route any packet.

Table 4-9. Bit Settings for Source Operations CAR

Bit	Field Name	Description
0–13	—	Reserved
14–15	Implementation Defined	Defined by the device implementation
16	Read	PE can support a read operation
17	Write	PE can support a write operation
18	Streaming-write	PE can support a streaming-write operation
19	Write-with-response	PE can support a write-with-response operation
20-22	—	Reserved
23	Atomic (test-and-swap)	PE can support an atomic test-and-swap operation
24	Atomic (increment)	PE can support an atomic increment operation
25	Atomic (decrement)	PE can support an atomic decrement operation
26	Atomic (set)	PE can support an atomic set operation
27	Atomic (clear)	PE can support an atomic clear operation
28	—	Reserved
29	Port-write	PE can support a port-write operation
30–31	Implementation Defined	Defined by the device implementation

4.4.8 Destination Operations CAR (Offset 0x18 Word 1)

This register defines the set of RapidIO I/O operations that can be supported by this processing element; see Table 4-10. It is required that all processing elements can respond to maintenance read and write requests in order to access these registers. The Destination Operations CAR is applicable for end point devices only. RapidIO switches shall be able to route any packet.

Table 4-10. Bit Settings for Destination Operations CAR

Bit	Field Name	Description
0-13	—	Reserved
14-15	Implementation Defined	Defined by the device implementation
16	Read	PE can support a read operation
17	Write	PE can support a write operation
18	Streaming-write	PE can support a streaming-write operation
19	Write-with-response	PE can support a write-with-response operation
20-22	—	Reserved
23	Atomic (test-and-swap)	PE can support an atomic test-and-swap operation
24	Atomic (increment)	PE can support an atomic increment operation
25	Atomic (decrement)	PE can support an atomic decrement operation
26	Atomic (set)	PE can support an atomic set operation
27	Atomic (clear)	PE can support an atomic clear operation
28	—	Reserved
29	Port-write	PE can support a port-write operation
30-31	Implementation Defined	Defined by the device implementation

4.5 Command and Status Registers (CSRs)

A processing element shall contain a set of command and status registers (CSRs) that allows an external processing element to control and determine the status of its internal hardware. All registers are 32 bits wide and are organized and accessed in the same way as the CARs. Refer to Table 4-2 for the required behavior for accesses to reserved registers and register bits.

4.5.1 Write Port CSR (Offset 0x40 Word 1)

The write port CSR is accessed if an external processing element wishes to determine the status of this processing element's write port hardware if the target processing element supports the port-write maintenance operation. It is not necessary to examine this register before sending a port-write transaction since the protocol will behave appropriately depending upon the status of the hardware. This register is read-only. See Table 4-11 for the bit settings for the write port CSR.

Table 4-11. Bit Settings for Write Port CSR

Bit	Field Name	Description
0–23	—	Reserved
24	Write Port Available	Write port hardware is initialized and ready to accept a port-write transaction. If not available, all incoming port-write transactions will be discarded.
25	Write Port Full	Write port hardware is full. All incoming port-write transactions will be discarded.
26	Write Port Empty	Write port hardware has no outstanding port-write transactions

Table 4-11. Bit Settings for Write Port CSR(Continued)

Bit	Field Name	Description
27	Write Port Busy	Write port hardware is busy queueing a port-write transaction. Incoming port-write transactions may or may not be discarded depending upon the implementation of the write port hardware in the PE.
28	Write Port Failed	Write port hardware has had an internal fault or error condition and is waiting for assistance. All incoming port-write transactions will be discarded.
29	Write Port Error	Write port hardware has encountered a port-write transaction that is found to be illegal for some reason. All incoming port-write transactions will be discarded.
30-31	—	Reserved

4.5.2 Processing Element Logical Layer Control CSR (Offset 0x48 Word 1)

The Processing Element Logical Layer Control CSR is used for general command and status information for the logical interface.

Table 4-12. Bit Settings for Processing Element Logical Layer Control CSR

Bit	Field Name	Description
0–28	—	Reserved
29-31	Extended addressing control	Controls the number of address bits generated by the PE as a source and processed by the PE as the target of an operation. 0b100 - PE supports 66 bit addresses 0b010 - PE supports 50 bit addresses 0b001 - PE supports 34 bit addresses (default) All other encodings reserved

4.5.3 Local Configuration Space High Base Address CSR (Offset 0x58 Word 0)

The local configuration space high base address register (LCSHBAR) specifies the most significant bytes of a local physical address offset for the processing element's configuration register space if the local address space is greater than 32 bits. See Section 4.5.4 below for a detailed description.

Table 4-13. Bit Settings for Local Configuration Space High Base Address CSR

Bit	Field Name	Description
0-31	LCSHBAR	Local Configuration Space High Base Address Register

4.5.4 Local Configuration Space Base Address CSR (Offset 0x58 Word 1)

The local configuration space base address register (LCSBAR) specifies the local physical address offset for the processing element's configuration register space, causing the configuration register space to be physically mapped in the processing element. This register allows configuration and maintenance of a processing element through regular read and write operations rather than maintenance configuration operations.

Table 4-14. Bit Settings for Local Configuration Space Low Base Address Register CSR

Bit	Field Name	Description
0-31	LCSBAR	Local Configuration Space Base Address Register

Partition II: Message Passing Logical Specification

II Partition II

Partition II is intended for users who need to understand the message passing architecture of the RapidIO interconnect.

II.1 Overview

The *Message Passing Logical Specification* is part of RapidIO's logical layer specifications that define the interconnect's overall protocol and packet formats. This layer contains the transaction protocols necessary for end points to process a transaction. Another RapidIO logical layer specification is explained in *Partition I: Input/Output Logical Specification.*

The logical specifications do not imply a specific transport or physical interface, therefore they are specified in a bit stream format. Necessary bits are added to the logical encoding for the transport and physical layers lower in the RapidIO three-layer hierarchy.

RapidIO is targeted toward memory mapped distributed memory systems. A message passing programming model is supported to enable distributed I/O processing.

II.2 Contents

Following are the contents of *Partition II: Message Passing Logical Specification:*

- Chapter 1, "System Models," introduces some possible devices that might participate in a RapidIO message passing system environment. The chapter also explains the message passing model, detailing the data and doorbell message types used in a RapidIO system. System issues such as the lack of transaction ordering and deadlock prevention are presented.
- Chapter 2, "Operation Descriptions," describes the set of operations and transactions supported by the RapidIO message passing protocols.
- Chapter 3, "Packet Format Descriptions," contains the packet format definitions for the message passing specification. The two basic types, request and response packets, and their fields and sub-fields are explained.
- Chapter 4, "Message Passing Registers," displays the RapidIO register map that allows an external processing element to determine the message passing capabilities, configuration, and status of a processing element using this logical specification. Only registers or register bits specific to the message passing logical specification are explained. Refer to the other RapidIO logical, transport, and physical specifications of interest to determine a complete list of registers and bit definitions.

1 Chapter 1 - System Models

This overview introduces some possible devices in a RapidIO system.

1.1 Processing Element Models

Figure 1-1describes a possible RapidIO-based system. The processing element is a computer device such as a processor attached to local memory and a RapidIO interconnect. The bridge part of the system provides I/O subsystem services such as high-speed PCI interfaces and Gbit ethernet ports, interrupt control, and other system support functions.

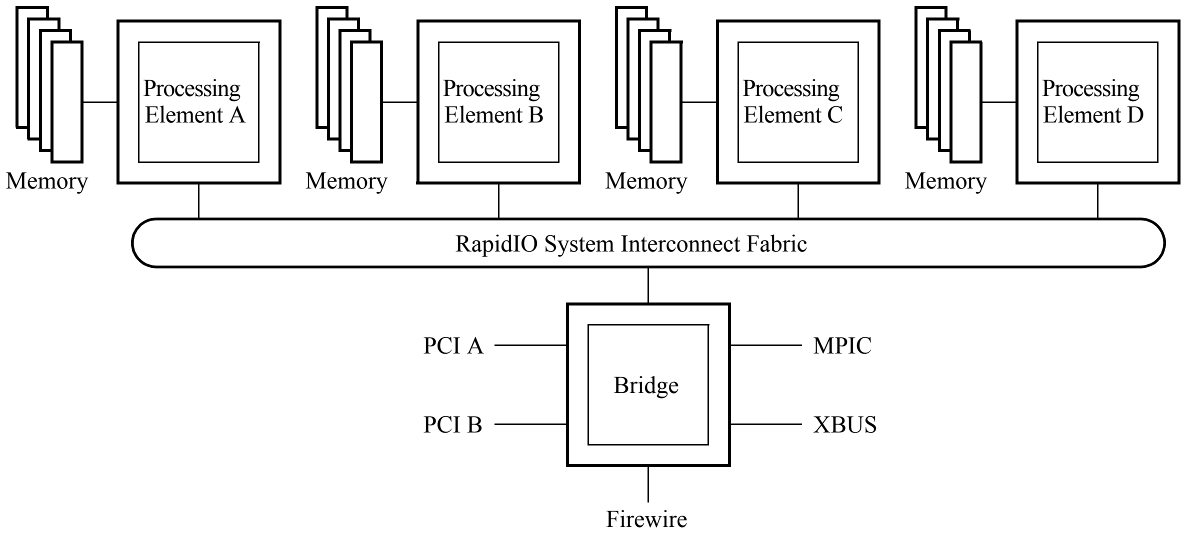


Figure 1-1. A Possible RapidIO-Based Computing System

The following sections describe several possible processing elements.

1.1.1 Processor-Memory Processing Element Model

Figure 1-2 shows an example of a processing element consisting of a processor connected to an agent device. The agent carries out several services on behalf of the processor. Most importantly, it provides access to local memory. It also provides an interface to the RapidIO interconnect to service message requests that are used for communications with other processing elements.

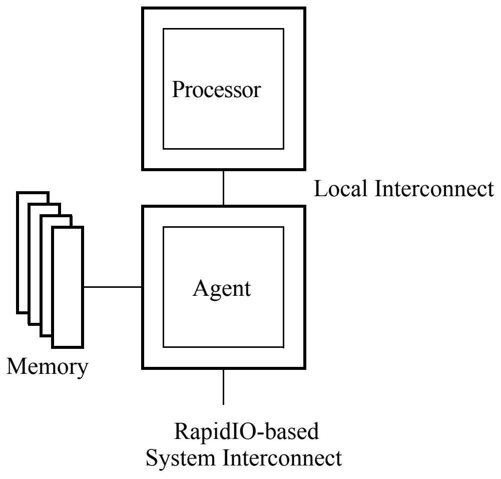


Figure 1-2. Processor-Memory Processing Element Example

1.1.2 Integrated Processor-Memory Processing Element Model

Another form of a processor-memory processing element is a fully integrated component that is designed specifically to connect to a RapidIO interconnect system, Figure 1-3. This type of device integrates a memory system and other-support logic with a processor on the same piece of silicon or within the same package.

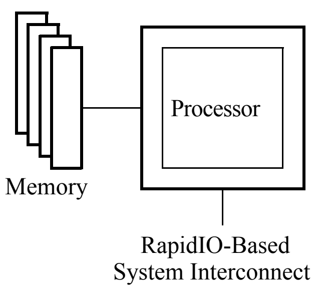


Figure 1-3. Integrated Processor-Memory Processing Element Example

1.1.3 Memory-Only Processing Element Model

A different processing element may not contain a processor at all, but may be a memory-only device as in Figure 1-4. This type of device is much simpler than a processor in that it is only responsible for responding to requests from the external system, not from local requests as in the processor-based model. As such, its memory is remote for all processors in the system.

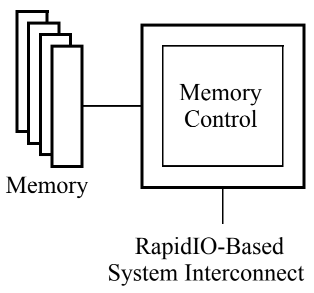


Figure 1-4. Memory-Only Processing Element Example

1.1.4 Processor-Only Processing Element

Similar to a memory-only element, a processor-only element has no local memory. A processor-only processing element is shown in Figure 1-5..

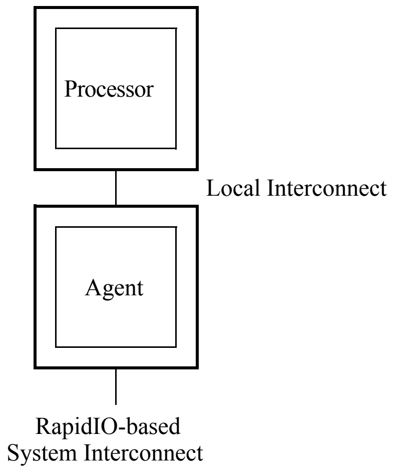


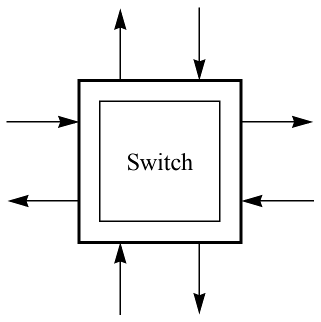
Figure 1-5. Processor-Only Processing Element Example

1.1.5 I/O Processing Element

This type of processing element is shown as the bridge in Figure 1-1. This device has distinctly different behavior than a processor or a memory. An I/O device only needs to move data into and out of local or remote memory.

1.1.6 Switch Processing Element

A switch processing element is a device that allows communication with other processing elements through the switch. A switch may be used to connect a variety of RapidIO-compliant processing elements. A possible switch is shown in Figure 1-6. Behavior of the switches, and the interconnect fabric in general, is addressed in the *RapidIO Common Transport Specification*.



Switch Processing Element Example

1.2 Message Passing System Model

RapidIO supports a message passing programming model. Message passing is a programming model commonly used in distributed memory system machines. In this model, processing elements are only allowed to access memory that is local to themselves, and communication between processing elements is handled through specialized hardware manipulated through application or OS software. For two processors to communicate, the sending processor writes to a local message passing device that reads a section of the sender's local memory and moves that information to the receiving processor's local message passing device. The recipient message passing device then stores that information in local memory and informs the recipient processor that a message has arrived, usually via an interrupt. The recipient processor then accesses its local memory to read the message.

For example, referring to Figure 1-1, processing element A can only access the memory attached to it, and cannot access the memory attached to processing elements B, C, or D. Correspondingly, processing element B can only access the memory attached to it and cannot access the memory attached to processing element A, C, or D, and so on. If processing element A needs to communicate with processing element B, the application software accesses special message passing hardware (also called mailbox hardware) through operating system calls or API libraries and configure it to assemble the message and send it to processing element B. The message passing hardware for processing element B receives the message and puts it into local memory at a predetermined address, then notifies processing element B.

Many times a message is required to be larger than a single packet allows, so the source needs to break up the message into multiple packets before transmitting it. At times it may also be useful to have more than one message being transmitted at a time. RapidIO has facilities for both of these features.

1.2.1 Data Message Operations

A source may generate a single message operation of up to 16 individual packets containing as much as 256 data bytes per packet. A variety of data payload sizes exist, allowing a source to choose a smaller size data payload if needed for an application. RapidIO defines all data message packets as containing the same amount of data with the exception of the last one, which can contain a smaller data payload if desired. The packets are formatted with three fields:

- One field specifies the size of the data payload for all except the last packet for the data message operation.
- The second field specifies the size of the data payload for that packet, and
- The third field contains the packet sequence order information.

The actual packet formats are shown in Chapter 3, "Packet Format Descriptions."

Because all packets except the last have the same data payload size, the receiver is able to calculate the local memory storage addresses if the packets are received out of order, allowing operation with an interconnect fabric that does not guarantee packet delivery ordering.

A letter field and a mailbox field allow a source to simultaneously have up to four data message operations (or "letters") in progress to each of four different mailboxes, allowing up to sixteen concurrent data message operations between a sender and a receiver. The mailbox field can be used to indicate the priority of a data message, allowing a higher priority message to interrupt a lower priority one at the sender, or it can be used as a simple mailbox identifier for a particular receiver if the receiver allows multiple mailbox addresses. If the mailbox number is used as a priority indicator, mailbox number 0 is the highest priority and mailbox 3 is the lowest.

The number of packets comprising a data message operation, the maximum data payload size, the number of concurrent letters, and the number of mailboxes that can be sent or received is determined by the implementation of a particular processing element. For example, a processing element could be designed to generate two concurrent letters of at most four packets with a maximum 64-byte data payload. That same processing element could also be designed to receive data messages in two mailboxes with two concurrent letters for each, all with the maximum data payload size and number of packets.

There is further discussion of the data message operation programming model and the necessary hardware support in Annex A, "Message Passing Interface".

1.2.2 Doorbell Message Operations

RapidIO supports a second message type, the doorbell message operation. The doorbell message operation sends a small amount of software-defined information to the receiver and the receiver controls all local memory addressing as with the data message operation. It is the responsibility of the processor receiving the doorbell message to determine the action to undertake by examining the ID of the sender and the received data. All information supplied in a doorbell message is embedded in the packet header so the doorbell message never has a data payload.

The generation, transmission, and receipt of a doorbell message packet is handled in a fashion similar to a data message packet. If processing element A wants to send a doorbell message to processing element B, the application software accesses special doorbell message hardware through operating system calls or API libraries and configures it to assemble the doorbell message and send it to processing element B. The doorbell message hardware for processing element B receives the doorbell message and puts it into local memory at a predetermined address, then notifies processing element B, again, usually via an interrupt.

There is further discussion of the doorbell message operation programming model and the necessary hardware support in Annex A, "Message Passing Interface".

1.3 System Issues

The following sections describe transaction ordering and system deadlock considerations in a RapidIO system.

1.3.1 Operation Ordering

The *RapidIO Message Passing Logical Specification* requires no special system operation ordering. Message operation completion is managed by the overlying system software.

It is important to recognize that systems may contain a mix of transactions that are maintained under the message passing model as well as under another model. As an example, I/O traffic may be interspersed with message traffic. In this case, the shared I/O traffic may require strong ordering rules to maintain coherency. This may set an operation ordering precedence for that implementation, especially in the case where the connection fabric cannot discern between one type of operation and another.

1.3.2 Transaction Delivery

There are two basic types of delivery schemes that can be built using RapidIO processing elements: unordered and ordered. The RapidIO logical protocols assume that all outstanding transactions to another processing element are delivered in an arbitrary order. In other words, the logical protocols do not rely on transaction interdependencies for operation. RapidIO also allows completely ordered delivery systems to be constructed. Each type of system puts different constraints on the implementation of the source and destination processing elements and any intervening hardware.

A message operation may consist of several transactions. It is possible for these transactions to arrive at a target mailbox in an arbitrary order. A message transaction contains explicit tagging information to allow the message to be reconstructed as it arrives at the target processing element.

1.3.3 Deadlock Considerations

A deadlock can occur if a dependency loop exists. A dependency loop is a situation where a loop of buffering devices is formed, in which forward progress at each device is dependent upon progress at the next device. If no device in the loop can make progress then the system is deadlocked.

The simplest solution to the deadlock problem is to discard a packet. This releases resources in the network and allows forward progress to be made. RapidIO is designed to be a reliable fabric for use in real time tightly coupled systems, therefore discarding packets is not an acceptable solution.

In order to produce a system with no chance of deadlock it is required that a deadlock free topology be provided for response-less operations. Dependency loops to single direction packets can exist in unconstrained switch topologies. Often the dependency loop can be avoided with simple routing rules. Topologies like hypercubes or three-dimensional meshes, physically contain loops. In both cases, routing is done in several dimensions (x,y,z). If routing is constrained to the x dimension, then y, then z (dimension ordered routing) then topology related dependency loops are avoided in these structures.

In addition, a processing element design must not form dependency links between its input and output port. A dependency link between input and output ports occurs if a processing element is unable to accept an input packet until a waiting packet can be issued from the output port.

RapidIO supports operations, such as read operations, that require responses to complete. These operations can lead to a dependency link between an processing element's input port and output port.

As an example of a input to output port dependency, consider a processing element where the output port queue is full. The processing element cannot accept a new request at its input port since there is no place to put the response in the output port queue. No more transactions can be accepted at the input port until the output port is able to free entries in the output queue by issuing packets to the system.

The method by which a RapidIO system maintains a deadlock free environment is described in the appropriate Physical Layer specification.

2 Chapter 2 - Operation Descriptions

This chapter describes the set of operations and transactions supported by the RapidIO message passing protocols. The opcodes and packet formats are described in Chapter 3, "Packet Format Descriptions".

The RapidIO operation protocols use request/response transaction pairs through the interconnect fabric. A processing element sends a request transaction to another processing element if it requires an activity to be carried out. The receiving processing element responds with a response transaction when the request has been completed or if an error condition is encountered. Each transaction is sent as a packet through the interconnect fabric. For example, a processing element that needs to send part of a message operation to another processing element sends a MESSAGE request packet to that processing element, which processes the message packet and returns a DONE response packet.

Three possible response transactions can be received by a requesting processing element:

- A DONE response indicates to the requestor that the desired transaction has completed.
- A RETRY response shall be generated for a message transaction that attempts to access a mailbox that is busy servicing another message operation, as can a doorbell transaction that encounters busy doorbell hardware. All transactions that are retried for any reason shall be retransmitted by the sender. This prevents a transaction from partially completing and then leaving the system in an unknown state.
- An ERROR response means that the target of the transaction encountered an unrecoverable error and could not complete the transaction.

Packets may contain additional information that is interpreted by the interconnect fabric to route the packets through the fabric from the source to the destination, such as a device number. These requirements are described in the appropriate RapidIO transport layer specification, and are beyond the scope of this specification.

Depending upon the interconnect fabric, other packets may be generated as part of the physical layer protocol to manage flow control, errors, etc. Flow control and other fabric-specific communication requirements are described in the appropriate RapidIO physical layer specification and are beyond the scope of this document.

Each request transaction sent into the system is marked with a transaction ID that is unique for each requestor and responder processing element pair. This transaction ID allows a response to be easily matched to the original request when it is returned to the requestor. An end point cannot reuse a transaction ID value to the same destination until the response from the original transaction has been received by the requestor. The number of outstanding transactions that may be supported is implementation dependent.

2.1 Message Passing Operations Cross Reference

Table 2.1 contains a cross-reference list of the message passing operations defined in this RapidIO specification and their system usage.

Table 2-1. Message Passing Operations Cross Reference

Operation	Transactions Used	Possible System Usage	Description	Packet Format
Doorbell	DOORBELL, RESPONSE		Section	Type 10 Section 3.1.4
Data Message	MESSAGE, RESPONSE		Section	Type 11 Section 3.1.5

2.2 Message Passing Operations

The two kinds of message passing transactions are described in this section and defined as follows:

- Doorbell
- Data Message

2.2.1 Doorbell Operations

The doorbell operation, consisting of the DOORBELL and RESPONSE transactions (typically a DONE response) as shown in Figure 2-1, is used by a processing element to send a very short message to another processing element through the interconnect fabric. The DOORBELL transaction contains the info field to hold information and does not have a data payload. This field is software-defined and can be used for any desired purpose; see Section 3.1.4, “Type 10 Packet Formats (Doorbell Class),” for information about the info field.

A processing element that receives a doorbell transaction takes the packet and puts it in a doorbell message queue within the processing element. This queue may be implemented in hardware or in local memory. This behavior is similar to that of typical message passing mailbox hardware. The local processor is expected to read the queue to determine the sending processing element and the info field and determine what action to take based on that information.

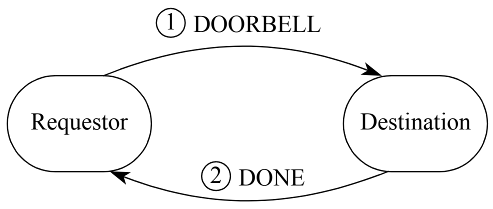


Figure 2-1. Doorbell Operation

2.2.2 Data Message Operations

The data message operation, consisting of the MESSAGE and RESPONSE transactions (typically a DONE response) as shown in Figure 2-2, is used by a processing element’s message passing support hardware to send a data message to other processing elements. Completing a data message operation can consist of up to 16 individual MESSAGE transactions. MESSAGE transaction data payloads are always multiples of doubleword quantities.

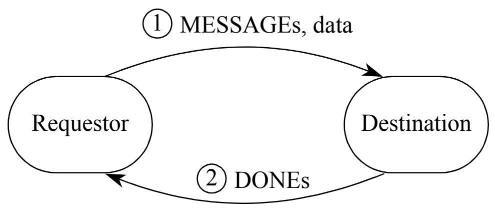


Figure 2-2. 22Message Operation

The processing element's message passing hardware that is the recipient of a data message operation examines a number of fields in order to place an individual MESSAGE packet data in local memory:

- Message length (msglen) field—Specifies the number of transactions that comprise the data message operation.
- Message segment (msgseg) field—Identifies which part of the data message operation is contained in this transaction. The message length and segment fields allow the individual packets of a data message to be sent or received out of order.
- Mailbox (mbox) field—Specifies which mailbox is the target of the data message.
- Letter (letter) field —Allows receipt of multiple concurrent data message operations from the same source to the same mailbox.
- Standard size (ssize) field—Specifies the data size of all of the transactions except (possibly) the last transaction in the data message.

From this information, the message passing hardware of the recipient processing element can calculate to which local memory address the transaction data should be placed.

For example, assume that the mailbox starting addresses for the recipient processing element are at addresses 0x1000 for mailbox 0, 0x2000 for mailbox 1, 0x3000 for mailbox 2, and 0x4000 for mailbox 3, and that the processing element receives a message transaction with the following fields:

- message length of 6 packets
- message segment is 3rd packet
- mailbox is mailbox 2
- letter is 1
- standard size is 32 bytes
- data payload is 32 bytes (it shall be 32 bytes since this is not the last transaction)

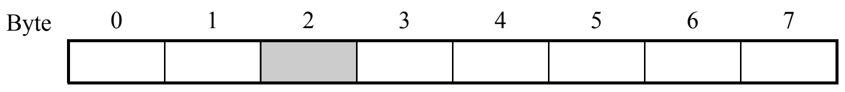
Using this information, the processing element's message passing hardware can determine that the 32 bytes contained in this part of the data message shall be put into local memory at address 0x3040.

The message passing hardware may also snoop the local processing element's caching hierarchy when writing local memory if the mailbox memory is defined as being cacheable by that processing element.

2.3 Endian, Byte Ordering, and Alignment

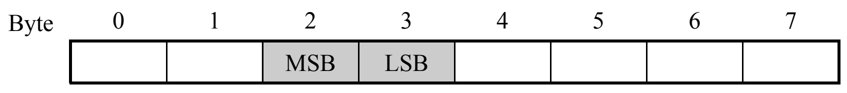
RapidIO has double-word (8-byte) aligned big-endian data payloads. This means that the RapidIO interface to devices that are little-endian shall perform the proper endian transformation at the output to format a data payload.

Operations that specify data quantities that are less than 8 bytes shall have the bytes aligned to their proper byte position within the big-endian double-word, as in the examples shown in Figure 2-3 through Figure 2-5.



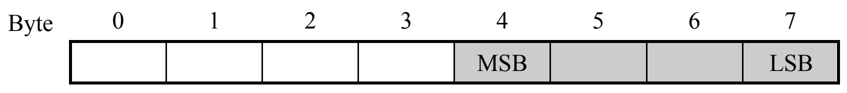
Byte address 0x0000_0002, the proper byte position is shaded.

Figure 2-3. Byte Alignment Example



Half-word address 0x0000_0002, the proper byte positions are shaded.

Figure 2-4. Half-Word Alignment Example



Word address 0x0000_0004, the proper byte positions are shaded.

Figure 2-5. Word Alignment Example

3 Chapter 3 - Packet Format Descriptions

This chapter contains the packet format definitions for the RapidIO Message Passing Logical Specification. There are four types of message passing packet formats:

- Request
- Response
- Implementation-defined
- Reserved

The packet formats are intended to be interconnect fabric independent so the system interconnect can be anything required for a particular application. Reserved formats, unless defined in another logical specification, shall not be used by a device.

3.1 Request Packet Formats

A request packet is issued by a processing element that needs a remote processing element to accomplish some activity on its behalf, such as a doorbell operation. The request packet format types and their transactions for the *RapidIO Message Passing Logical Specification* are shown in Table .3-1

Table 3-1. Request Packet Type to Transaction Type Cross Reference

Request Packet Format Type	Transaction Type	Definition	Document Section Number
Type 0	Implementation-defined	Defined by the device implementation	Section 3.1.2
Type 1–9	—	Reserved	Section 3.1.3
Type 10	DOORBELL	Send a short message	Section 3.1.4
Type 11	MESSAGE	Send a message	Section 3.1.5

3.1.1 Field Definitions for All Request Packet Formats

The field definitions in Table 3-2 apply to all of the request packet formats. Fields that are unique to type 10 and type 11 formats are defined in the sections that describe each type. Bit fields that are defined as “reserved” shall be assigned to logic 0s when generated and ignored when received. Bit field encodings that are defined as “reserved” shall not be assigned when the packet is generated. A received reserved encoding is regarded as an error if a meaningful encoding is required for the transaction and function, otherwise it is ignored. Implementation-defined fields shall be ignored unless the encoding is understood by the receiving device. All packets described are bit streams from the first bit to the last bit, represented in the figures from left to right respectively.

Table 3-2. General Field Definitions for All Request Packets

Field	Definition
ftype	Format type—Represented as a 4-bit value; is always the first four bits in the logical packet stream.
rsrv	Reserved

3.1.2 Type 0 Packet Format (Implementation-Defined)

The type 0 packet format is reserved for implementation-defined functions such as flow control.

3.1.3 Type 1–9 Packet Formats (Reserved)

The type 1–9 formats are reserved.

3.1.4 Type 10 Packet Formats (Doorbell Class)

The type 10 packet format is the DOORBELL transaction format. Type 10 packets never have data payloads. The field value 0b1010 in Figure specifies that the packet format is of type 10.

Definitions and encodings of fields specific to type 10 packets are provided in Table .3-3 Fields that are not specific to type 10 packets are described in Table .3-2.

Table 3-3. Specific Field Definitions for Type 10 Packets

Field	Encoding	Definition
info	—	Software-defined information field

Figure 3-1displays a type 10 packet with all its fields.

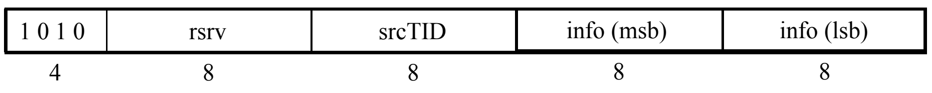


Figure 3-1. Type 10 Packet Bit Stream Format

3.1.5 Type 11 Packet Format (Message Class)

The type 11 packet is the MESSAGE transaction format. Type 11 packets always have a data payload. Sub-double-word messages are not specifiable and must be managed in software.

Definitions and encodings of fields specific to type 11 packets are provided in Table .3-4. Fields that are not specific to type 11 packets are described in Table .3-2.

Table 3-4. Specific Field Definitions and Encodings for Type 11 Packets

Field	Encoding	Definition
msglen	—	Total number of packets comprising this message operation. A value of 0 indicates a single-packet message. A value of 15 (0xF) indicates a 16-packet message, etc. See example in Section 2.2.2, “Data Message Operations”.
msgseg	—	Specifies the part of the message supplied by this packet. A value of 0 indicates that this is the first packet in the message. A value of 15 (0xF) indicates that this is the sixteenth packet in the message, etc. See example in Section 2.2.2, “Data Message Operations”.

Table 3-4. Specific Field Definitions and Encodings for Type 11 Packets(Continued)

Field	Encoding	Definition
ssize	—	Standard message packet data size. This field informs the receiver of a message the size of the data payload to expect for all of the packets for a single message operations except for the last packet in the message. This prevents the sender from having to pad the data field excessively for the last packet and allows the receiver to properly put the message in local memory. See example in Section 2.2.2, "Data Message Operations".
	0b0000–1000	Reserved
	0b1001	8 bytes
	0b1010	16 bytes
	0b1011	32 bytes
	0b1100	64 bytes
	0b1101	128 bytes
	0b1110	256 bytes
	0b1111	Reserved
mbox	—	Specifies the recipient mailbox in the target processing element
letter	—	Identifies a slot within a mailbox. This field allows a sending processing element to concurrently send up to four messages to the same mailbox on the same processing element.

Figure 3-2 displays a type 11 packet with all its fields. The value 0b1011 in Figure 3-2 specifies that the packet format is of type 11.

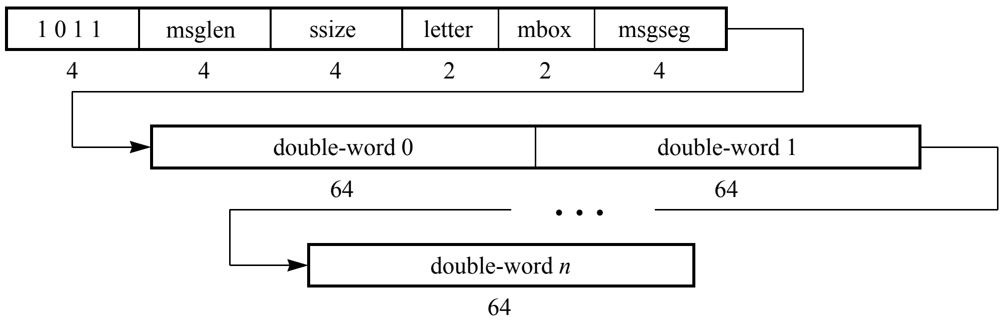


Figure 3-2. Type 11 Packet Bit Stream Format

The combination of the letter, mbox, and msgseg fields uniquely identifies the message packet in the system for each requestor and responder processing element pair in the same way as the transaction ID is used for other request types.

3.2 Response Packet Formats

A response transaction is issued by a processing element when it has completed a request made by a remote processing element. Response packets are always directed and are transmitted in the same way as request packets. Currently two response packet format types exist, as shown in 3-5..

Table 3-5. Response Packet Type to Transaction Type Cross Reference

Response Packet Format Type	Transaction Type	Definition	Document Section Number
Type 12	—	Reserved	Section 3.2.2
Type 13	RESPONSE	Issued by a processing element when it completes a request by a remote element.	Section 3.2.3
Type 14	—	Reserved	Section 3.2.4
Type 15	Implementation-defined	Defined by the device implementation	Section 3.2.5

3.2.1 Field Definitions for All Response Packet Formats

The field definitions in Table 3-6 apply to more than one of the response packet formats. Fields that are unique to the type 13 format are defined in Section , “3.2.3 Type 13 Packet Format (Response Class).”

Table 3-6. Field Definitions and Encodings for All Response Packets

Field	Encoding	Sub-Field	Definition
transaction	0b0000		RESPONSE transaction with no data payload
	0b0001		Message RESPONSE transaction
	0b0010–1111		Reserved
status	Type of status and encoding		
	0b0000	DONE	Requested transaction has been successfully completed
	0b0001–0010	—	Reserved
	0b0011	RETRY	Requested transaction is not accepted; must retry the request
	0b0100–0110	—	Reserved
	0b0111	ERROR	Unrecoverable error detected
	0b1000–1011	—	Reserved
	0b1100–1111	Implementation	Implementation defined—Can be used for additional information such as an error code

3.2.2 Type 12 Packet Format (Reserved)

The type 12 packet format is reserved.

3.2.3 Type 13 Packet Format (Response Class)

The type 13 packet format returns status and the requestor’s transaction ID or message segment and mailbox information. The type 13 format is used for response packets to all request packets. Responses to message and doorbell packets never contain data.

Definitions and encodings of fields specific to type 13 packets are provided in Table .3-7. Fields that are not specific to type 13 packets are described in Table .3-6.

Table 3-7. Specific Field Definitions for Type 13 Packets

Field	Sub-Field	Definition
target_info	As shown in Figure 3.3, when the response is the target_info field, these three sub-fields are used:	
	msgseg	Specifies the part of the message supplied by the corresponding message packet. A value of 0 indicates that this is the response for the first packet in the message. A value of 15 (0xF) indicates that this is the response for the sixteenth (and last) packet in the message, etc.
	mbox	Specifies the recipient mailbox from the corresponding message packet.
	letter	Identifies the slot within the target mailbox. This field allows a sending processing element to concurrently send up to four messages to the same mailbox on the same processing element.
targetTID	—	Transaction ID of the request that caused this response (except for message responses defined in Figure 3.3).

Figure 3-3 shows the format of the target_info field for message responses.

letter	mbox	msgseg
2	2	4

Figure 3.3. target_info Field for Message Responses

Figure 3-4 displays a type 13 packet with all its fields. The value 0b1101 in Figure specifies that the packet format is of type 13.

1 1 0 1	transaction	status	target_info/targetTID
4	4	4	8

Figure 3-4. Type 13 Packet Bit Stream Format

3.2.4 Type 14 Packet Format (Reserved)

The type 14 packet format is reserved.

3.2.5 Type 15 Packet Format (Implementation-Defined)

The type 15 packet format is reserved for implementation-defined functions such as flow control.

4 Chapter 4 - Message Passing Registers

This chapter describes the visible register set that allows an external processing element to determine the capabilities, configuration, and status of a processing element using this logical specification. This chapter only describes registers or register bits defined by this specification. Refer to the other RapidIO logical, transport, and physical specifications of interest to determine a complete list of registers and bit definitions. All registers are 32-bits and aligned to a 32-bit boundary.

4.1 Register Summary

Table 4-1 shows the register map for this RapidIO specification. These capability registers (CARs) and command and status registers (CSRs) can be accessed using *Partition I: Input/Output Logical Specification* maintenance operations. Any register offsets not defined are considered reserved for this specification unless otherwise stated. Other registers required for a processing element are defined in other applicable RapidIO specifications and by the requirements of the specific device and are beyond the scope of this specification. Read and write accesses to reserved register offsets shall terminate normally and not cause an error condition in the target device. Writes to CAR (read-only) space shall terminate normally and not cause an error condition in the target device.

Register bits defined as reserved are considered reserved for this specification only. Bits that are reserved in this specification may be defined in another RapidIO specification.

Table 4-1. Message Passing Register Map

Configuration Space Byte Offset	Register Name (Word 0)	Register Name (Word 1)
0x0-8	Reserved	
0x10	Processing Element Features CAR	Reserved
0x18	Source Operations CAR	Destination Operations CAR
0x20–38	Reserved	
0x40	Mailbox CSR	Doorbell CSR
0x48–F8	Reserved	
0x100–FFF8	Extended Features Space	
0x10000–FFFFF8	Implementation-defined Space	

4.2 Reserved Register and Bit Behavior

Table 4-2 describes the required behavior for accesses to reserved register bits and reserved registers for the RapidIO register space,

Table 4-2. Configuration Space Reserved Access Behavior

Byte Offset	Space Name	Item	Initiator behavior	Target behavior
0x0–3C	Capability Register Space (CAR Space - this space is read-only)	Reserved bit	read - ignore returned value[1]	read - return logic 0
			write -	write - ignored
		Implementation-defined bit	read - ignore returned value unless implementation-defined function understood	read - return implementation-defined value
			write -	write - ignored
		Reserved register	read - ignore returned value	read - return logic 0s
			write -	write - ignored
0x40–FC	Command and Status Register Space (CSR Space)	Reserved bit	read - ignore returned value	read - return logic 0
			write - preserve current value[2]	write - ignored
		Implementation-defined bit	read - ignore returned value unless implementation-defined function understood	read - return implementation-defined value
			write - preserve current value if implementation-defined function not understood	write - implementation-defined
		Reserved register	read - ignore returned value	read - return logic 0s
			write -	write - ignored

Table 4-2. Configuration Space Reserved Access Behavior(Continued)

Byte Offset	Space Name	Item	Initiator behavior	Target behavior
0x100–FFFC	Extended Features Space	Reserved bit	read - ignore returned value	read - return logic 0
			write - preserve current value	write - ignored
		Implementation-defined bit	read - ignore returned value unless implementation-defined function understood	read - return implementation-defined value
			write - preserve current value if implementation-defined function not understood	write - implementation-defined
		Reserved register	read - ignore returned value	read - return logic 0s
			write -	write - ignored
0x10000–FFFFFC	Implementation-defined Space	Reserved bit and register	All behavior implementation-defined	

1. Do not depend on reserved bits being a particular value; use appropriate masks to extract defined bits from the read value.
2. All register writes shall be in the form: read the register to obtain the values of all reserved bits, merge in the desired values for defined bits to be modified, and write the register, thus preserving the value of all reserved bits.

4.3 Capability Registers (CARs)

Every processing element shall contain a set of registers that allows an external processing element to determine its capabilities using the I/O logical maintenance read operation. All registers are 32 bits wide and are organized and accessed in 32-bit (4 byte) quantities, although some processing elements may optionally allow larger accesses. CARs are read-only. Refer to Table 4-2 for the required behavior for accesses to reserved registers and register bits.

CARs are big-endian with bit 0 and Word 0 respectively the most significant bit and word.

4.3.1 Processing Element Features CAR (Offset 0x10 Word 0)

This register identifies the major functionality provided by the processing element; see Table . 4-3.

Table 4-3. Bit Settings for Processing Element Features CAR

Bit	Field Name	Description
0–7	—	Reserved
8	Mailbox 0	PE supports inbound mailbox #0
9	Mailbox 1	PE supports inbound mailbox #1
10	Mailbox 2	PE supports inbound mailbox #2
11	Mailbox 3	PE supports inbound mailbox #3
12	Doorbell	PE supports inbound doorbells
13–31	—	Reserved

4.3.2 Source Operations CAR (Offset 0x18 Word 0)

This register defines the set of RapidIO message passing logical operations that can be issued by this processing element; see Table .4-4. It is assumed that a processing element can generate I/O logical maintenance read and write requests if it is required to access CARs and CSRs in other processing elements. The Source Operations CAR is applicable for end point devices only. RapidIO switches shall be able to route any packet.

Table 4-4. Bit Settings for Source Operations CAR

Bit	Field Name	Description
0–13	—	Reserved
14–15	Implementation Defined	Defined by the device implementation
16–19	—	Reserved
20	Data message	PE can support a data message operation
21	Doorbell	PE can support a doorbell operation
22–29	—	Reserved
30–31	Implementation Defined	Defined by the device implementation

4.3.3 Destination Operations CAR (Offset 0x18 Word 1)

This register defines the set of RapidIO message passing operations that can be supported by this processing element; see Table .4-5. It is required that all processing elements can respond to I/O logical maintenance read and write requests in order to access these registers. The Destination Operations CAR is applicable for end point devices only. RapidIO switches shall be able to route any packet.

Table 4-5. Bit Settings for Destination Operations CAR

Bit	Field Name	Description
0–13	—	Reserved
14–15	Implementation Defined	Defined by the device implementation
16–19	—	Reserved
20	Data message	PE can support a data message operation
21	Doorbell	PE can support a doorbell operation
22–29	—	Reserved
30–31	Implementation Defined	Defined by the device implementation

4.4 Command and Status Registers (CSRs)

A processing element shall contain a set of command and status registers (CSRs) that allows an external processing element to control and determine the status of its internal hardware. All registers are 32 bits wide and are organized and accessed in the same way as the CARs. Refer to Table for the required behavior for accesses to reserved registers and register bits.

4.4.1 Mailbox CSR (Offset 0x40 Word 0)

The mailbox command and status register is accessed if an external processing element wishes to determine the status of this processing elements’s mailbox hardware, if any is present. It is not necessary to examine this register before sending a message since the RapidIO protocol shall accept, retry, or send an error response message depending upon the status of the addressed mailbox. This register is read-only. Table 4-6 shows bit settings for the mailbox status register (CSR).

Table 4-6. Bit Settings for Mailbox CSR

Bit	Field Name	Description
0	Mailbox 0 Available	Mailbox 0 is initialized and ready to accept messages. If not available, all incoming message transactions return error responses.
1	Mailbox 0 Full	Mailbox 0 is full. All incoming message transactions return retry responses.
2	Mailbox 0 Empty	Mailbox 0 has no outstanding messages.

Table 4-6. Bit Settings for Mailbox CSR(Continued)

Bit	Field Name	Description
3	Mailbox 0 Busy	Mailbox 0 is busy receiving a message operation. New message operations return retry responses.
4	Mailbox 0 Failed	Mailbox 0 had an internal fault or error condition and is waiting for assistance. All incoming message transactions return error responses.
5	Mailbox 0 Error	Mailbox 0 encountered a message operation or transaction of an unacceptable size. All incoming message transactions return error responses.
6–7	—	Reserved
8	Mailbox 1 Available	Mailbox 1 is initialized and ready to accept messages. If not available, all incoming message transactions return error responses.
9	Mailbox 1 Full	Mailbox 1 is full. All incoming message transactions return retry responses.
10	Mailbox 1 Empty	Mailbox 1 has no outstanding messages.
11	Mailbox 1 Busy	Mailbox 1 is busy receiving a message operation. New message operations return retry responses.
12	Mailbox 1 Failed	Mailbox 1 had an internal fault or error condition and is waiting for assistance. All incoming message transactions return error responses.
13	Mailbox 1 Error	Mailbox 1 encountered a message operation or transaction of an unacceptable size. All incoming message transactions return error responses.
14-15	—	Reserved
16	Mailbox 2 Available	Mailbox 2 is initialized and ready to accept messages. If not available, all incoming message transactions return error responses.
17	Mailbox 2 Full	Mailbox 2 is full. All incoming message transactions return retry responses.
18	Mailbox 2 Empty	Mailbox 2 has no outstanding messages.
19	Mailbox 2 Busy	Mailbox 2 is busy receiving a message operation. New message operations return retry responses.
20	Mailbox 2 Failed	Mailbox 2 had an internal fault or error condition and is waiting for assistance. All incoming message transactions return error responses.
21	Mailbox 2 Error	Mailbox 2 encountered a message operation or transaction of an unacceptable size. All incoming message transactions return error responses.
22-23	—	Reserved
24	Mailbox 3 Available	Mailbox 3 is initialized and ready to accept messages. If not available, all incoming message transactions return error responses.
25	Mailbox 3 Full	Mailbox 3 is full. All incoming message transactions return retry responses.
26	Mailbox 3 Empty	Mailbox 3 has no outstanding messages.
27	Mailbox 3 Busy	Mailbox 3 is busy receiving a message operation. New message operations return retry responses.

Table 4-6. Bit Settings for Mailbox CSR(Continued)

Bit	Field Name	Description
28	Mailbox 3 Failed	Mailbox 3 had an internal fault or error condition and is waiting for assistance. All incoming message transactions return error responses.
29	Mailbox 3 Error	Mailbox 3 encountered a message operation or transaction of an unacceptable size. All incoming message transactions return error responses.
30-31	—	Reserved

4.4.2 Doorbell CSR (Offset 0x40 Word 1)

The doorbell CSR is accessed if an external processing element wishes to determine the status of this processing element's doorbell hardware if the target processing element supports these operations. It is not necessary to examine this register before sending a doorbell message since the protocol shall behave appropriately depending upon the status of the hardware. This register is read-only. See 4-7 for the bit settings for the doorbell status register.

Table 4-7. Bit Settings for Doorbell CSR

Bit	Field Name	Description
0	Doorbell Available	Doorbell hardware is initialized and ready to accept doorbell messages. If not available, all incoming doorbell transactions return error responses.
1	Doorbell Full	Doorbell hardware is full. All incoming doorbell transactions return retry responses.
2	Doorbell Empty	Doorbell hardware has no outstanding doorbell messages
3	Doorbell Busy	Doorbell hardware is busy queueing a doorbell message. Incoming doorbell transactions may or may not return a retry response depending upon the implementation of the doorbell hardware in the PE.
4	Doorbell Failed	Doorbell hardware has had an internal fault or error condition and is waiting for assistance. All incoming doorbell transactions return error responses.
5	Doorbell Error	Doorbell hardware has encountered an Doorbell transaction that is found to be illegal for some reason. All incoming doorbell transactions return error responses.
6–31	—	Reserved

Annex A

Interface Management (Informative)

This annex contains state machine descriptions that illustrate a number of behaviors that are described in the *RapidIO Physical Layer 8/16 LP-LVDS Specification*. They are included as examples and are believed to be correct, however, actual implementations should not use the examples directly.

A.1 Link Initialization and Maintenance Mechanism

This section contains the link training and initialization state machine referred to in Section 2.6.1.1, “Sampling Window Alignment.” Training takes place in two circumstances; when coming out of reset and after the loss of reliable input port sampling during system operation.

Link initialization and maintenance actually requires two inter-dependent state machines in order to operate, one associated with the input port and the other with the output port. The two state machines work together to complete the link training. The state machines are intended for a device with an 8-bit port or a device with a 16-bit port. The port can only transition from the “Port uninitialized” state to the “Port ready” state when both halves of the state machine are in their “ready” state.

A.1.1 Input port training state machine

Figure A-1 illustrates the input port training state machine. Error conditions are only detectable while in the “ready” states (ready and ready_maint_trn). The optional ready_maint_trn state, shaded in Figure A-1, is used to adjust the device input port sampling circuitry during system operation.

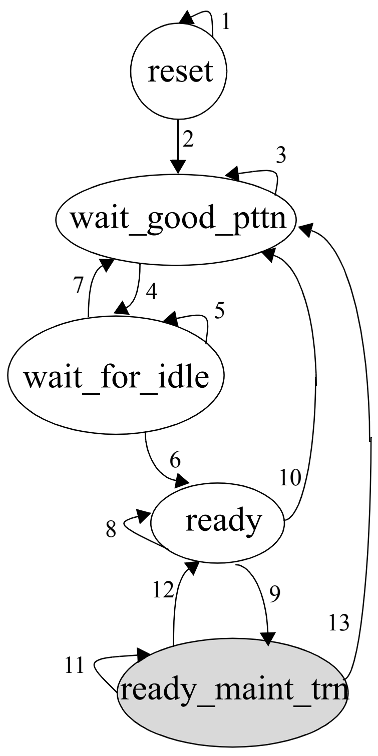


Figure A-1. Input port training state machine

Table A-1 describes the state transition arcs for Figure A-1.

Table A-1. Input port training state machine transition table

Arc	Current State	Next state	cause	Comments
1	reset	reset	Start training condition not met.	Remain in the reset state until the start training condition is met. Typically, this is after reset has been applied to the device and all other necessary initialization activity has completed.
2	reset	wait_good_pttn	Start training condition met.	This state is entered after all initialization activity has completed for the device.
3	wait_good_pttn	wait_good_pttn	The defined training pattern has not been detected yet. Wait for the sampling circuitry to indicate that the defined training pattern has been received and the sampling circuitry is calibrated.	Remain in this state until the defined training pattern is detected.
4	wait_good_pttn	wait_for_idle	Sampling circuitry is calibrated and the defined training pattern has been received.	Upon recognizing the defined training pattern, a 16-bit port can decide whether it’s output port needs to be downgraded to drive in 8-bit mode. Request the output port to start sending idle control symbols.
5	wait_for_idle	wait_for_idle	Remain in this state until an exit condition occurs.	In this state, only training patterns and the idle and link-request/ send-training control symbols are legal.
6	wait_for_idle	ready	Idle control symbol has been received and the output port is in the “send_idles” state.	This transition indicates that the input port is ready to start receiving packets and other control symbols.
7	wait_for_idle	wait_good_pttn	The input port receives something besides a training pattern, idle, or link-request/send-training control symbol, or the sampling circuitry is no longer calibrated.	Receiving something unexpected or when the sampling circuitry is no longer able to reliably sample the device pins causes both the input port and output port to start restart the training sequence.
8	ready	ready	Sampling circuitry remains calibrated and is not drifting.	This is a functional state in which packets and control symbols can be accepted. Errors are also reported in this state.
9	ready	ready_maint_trn	Sampling circuitry drift.	This transition takes place when the sampling circuitry can still reliably sample the device pins, but adjustment is required to prevent eventual loss of calibration.

Table A-1. Input port training state machine transition table(Continued)

Arc	Current State	Next state	cause	Comments
10	ready	wait_good_pttn	Sampling circuitry is no longer calibrated.	Both the input port and output port restart the training sequence when the sampling circuitry is no longer able to reliably sample the device pins. This error invokes the error recovery algorithm when the ready state is re-entered to attempt to recover possible lost data.
11	ready_maint_trn	ready_maint_trn	The complete sequence of 256 training patterns has not been received, and the sampling circuitry is still calibrated.	This is a functional state in which packets and control symbols can be accepted. Errors are also reported in this state. In this state, the device adjusts the sampling circuitry when the training patterns are received.
12	ready_maint_trn	ready	The complete sequence of 256 training patterns has been received and the sampling circuitry is still calibrated.	Sampling circuitry has been adjusted.
13	ready_maint_trn	wait_good_pttn	Sampling circuitry is no longer calibrated.	Both the input port and output port restart the training sequence when the sampling circuitry is no longer able to reliably sample the device pins. This error invokes the error recovery algorithm when the ready state is re-entered to attempt to recover possible lost data.

A.1.2 Output port training state machine

Figure A-2 illustrates the output port training state machine. Packets can only be transmitted when both the input port and output port are in their “ready” states (ready and ready_maint_trn for the input port, and ready, ready_trn_req and ready_send_pttn for the output port). The 8-bit mode adjustment state for a 16-bit port is heavily shaded in igure A-2 and is not required for 8-bit ports. The optional ready_maint_trn state, lightly shaded in Figure A-2, is used to adjust the device input port sampling circuitry during system operation, and is associated with the ready_maint_trn state in the input port state machine.

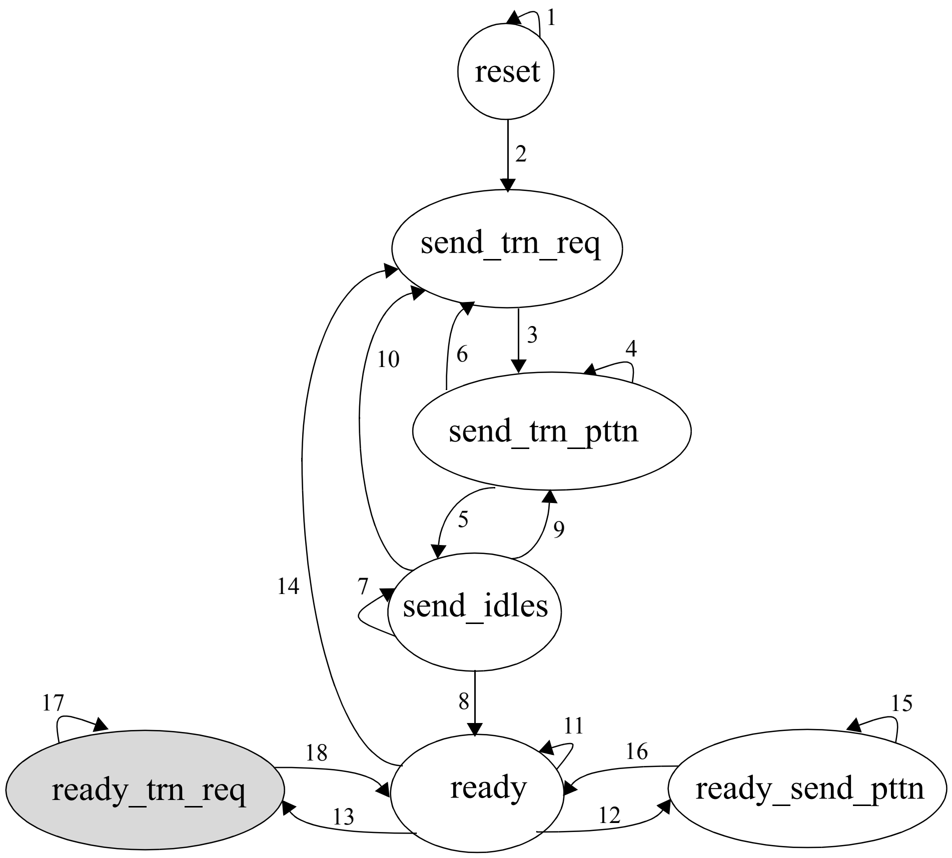


Figure A-2. Output port training state machine

Table A-2 describes the state transition arcs for Figure A-2.

Table A-2. Output port training state machine transition table

Arc	Current State	Next state	cause	Comments
1	reset	reset	Start training condition not met.	Remain in the reset state until the start training condition is met. Typically, this is after reset has been applied to the device and all other necessary initialization activity has completed.
2	reset	send_trn_req	Start training condition met.	This state is entered after all initialization activity has completed for the device. The output port will send a link-request/send-training control symbol
3	send_trn_req	send_trn_pttn	Unconditional transition.	The output port will send 256 iterations of the training pattern
4	send_trn_pttn	send_trn_pttn	The 256 iterations of the training pattern is not completed.	The input port is waiting to calibrate and receive the defined training pattern. The output port is sending training patterns.

Table A-2. Output port training state machine transition table(Continued)

Arc	Current State	Next state	cause	Comments
5	send_trn_pttn	send_idles	The 256 iterations of the training pattern is completed and the input port has requested to send idle control symbols.	The input port sampling circuitry is calibrated and the input port has received the defined training pattern. In the send_idles state, idle control symbols are sent out on the output port.
6	send_trn_pttn	send_trn_req	The 256 iterations of the training pattern are completed but the input port has not requested to send idle control symbols.	Remain in the send_trn_req - send_trn_pttn loop until the input port sampling circuitry is calibrated and the input port recognizes the defined training pattern and then requests to send idle control symbols. A link-request/send-training control symbol is sent out in state send_trn_req.
7	send_idles	send_idles	Remain in this state until an exit condition occurs.	A reset to the beginning of the training sequence occurs on input port transitions into the wait_good_pttn state. Idle control symbols are sent on the output port in this state.
8	send_idles	ready	The input port is in state wait_for_idle and has received an idle control symbol.	Ready to start sending packets and any control symbol.
9	send_idles	send_trn_pttn	A link-request/send-training is received on the input port.	The output port will send 256 iterations of the of the training pattern as requested.
10	send_idles	send_trn_req	The input port asks for a reset back to the beginning of the training sequence.	Transition to send_trn_req and start over.
11	ready	ready	A link-request/send-training is not received on the input port and the input port does not ask for a reset to the beginning of the training sequence.	This is a functional state in which packets and control symbols are transmitted. Errors are detected and reported in this state.
12	ready	ready_send_pttn	link-request/send-training is received on the input port.	This transition occurs when in the ready state and a training request is received from the attached device.
13	ready	ready_trn_req	The input port wants the attached device to send 256 iterations of the training pattern.	This transition occurs when in the ready state and input port sampling circuitry needs to be adjusted, and is associated with the optional input port ready_maint_trn state.

Table A-2. Output port training state machine transition table(Continued)

Arc	Current State	Next state	cause	Comments
14	ready	send_trn_req	The input port asks for a reset to the beginning of the training sequence.	Transition to send_trn_req and start over. This occurs when the sampling circuitry is no longer able to reliably sample the device pins.
15	ready_send_pttn	ready_send_pttn	The 256 iterations of the training pattern is not completed.	The output port is sending training patterns. Errors are detected and reported in this state. Must send at least one idle control symbol after the 256 iterations.
16	ready_send_pttn	ready	The 256 iterations of the training pattern are completed and followed by at least one idle control symbol.	This is a normal operating case where the attached device requested that we send training patterns.
17	ready_trn_req	ready_trn_req	Waiting to send the link-request/send-training	Might have to wait for the end of the current packet because link-request control symbols can not be embedded. Errors are detected and reported in this state.
18	ready_trn_req	ready	link-request/send-training sent out on the output port as requested by the input port.	Input port is requesting training patterns from the other end to adjust its sampling circuitry.

A.2 Packet Retry Mechanism

This section contains the example packet retry mechanism state machine referred to in Section 1.2.3, “Transaction and Packet Delivery”.

Packet retry recovery actually requires two inter-dependent state machines in order to operate, one associated with the input port and the other with the output port on the two connected devices. The two state machines work together to attempt recovery from a retry condition.

A.2.1 Input port retry recovery state machine

If a packet cannot be accepted by a receiver for reasons other than error conditions, such as a full input buffer, the receiver follows the state sequence shown in Figure A-3.

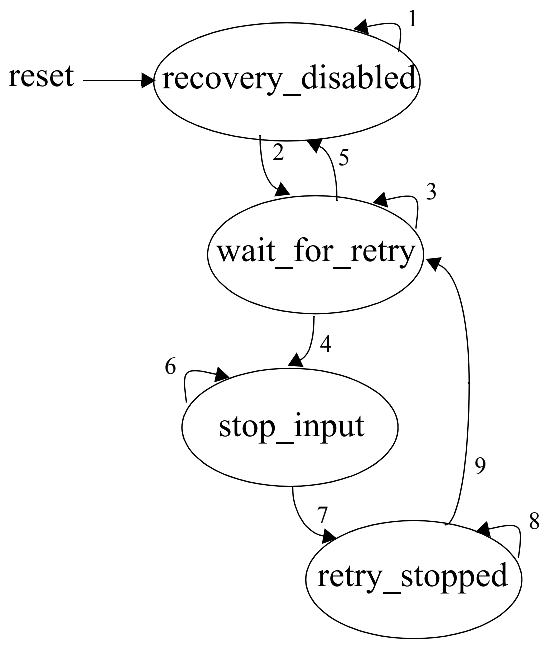


Figure A-3. Input port retry recovery state machine

Table A-3 describes the state transition arcs for Figure A-3. The states referenced in the comments in quotes are the RapidIO 8/16 LP-LVDS defined status states, not states in this state machine.

Table A-3. Input port retry recovery state machine transition table

Arc	Current State	Next state	cause	Comments
1	recovery_disabled	recovery_disabled	Remain in this state until the input port is enabled to receive packets.	This is the initial state after reset. The input port can't be enabled before the training sequence has been completed, and may be controlled through other mechanisms as well, such as a software enable bit.
2	recovery_disabled	wait_for_retry	Input port is enabled.	
3	wait_for_retry	wait_for_retry	Remain in this state until a packet retry situation has been detected.	
4	wait_for_retry	stop_input	A packet retry situation has been detected.	Usually this is due to an internal resource problem such as not having packet buffers available for low priority packets.
5	wait_for_retry	recovery_disabled	Input port is disabled.	
6	stop_input	stop_input	Remain in this state until described input port stop activity is completed.	Send a packet-retry control symbol with the expected ackID, discard the packet, and don't change the expected ackID. This will force the attached device to initiate recovery starting at the expected ackID. Clear the "Port ready" state and set the "Input Retry-stopped" state.

Table A-3. Input port retry recovery state machine transition table(Continued)

Arc	Current State	Next state	cause	Comments
7	stop_input	retry_stopped	Input port stop activity is complete.	
8	retry_stopped	retry_stopped	Remain in this state until a restart-from-retry or restart-from-error control symbol is received or an input port error is encountered.	The “Input Retry-stopped” state causes the input port to silently discard all incoming packets and not change the expected ackID value.
9	retry_stopped	wait_for_retry	Received a restart-from-retry or a restart-from-error control symbol or an input port error is encountered.	The restart-from-error control symbol is a link-request/input-status control symbol. Clear the “Input Retry-stopped” state and set the “Port ready” state. An input port error shall cause a clean transition between the retry recovery state machine and the error recovery state machine.

A.2.2 Output port retry recovery state machine

On receipt of an error-free packet-retry acknowledge control symbol, the attached output port follows the behavior shown in Figure A-4. The states referenced in the comments in quotes are the RapidIO 8/16 LP-LVDS defined status states, not states in this state machine.

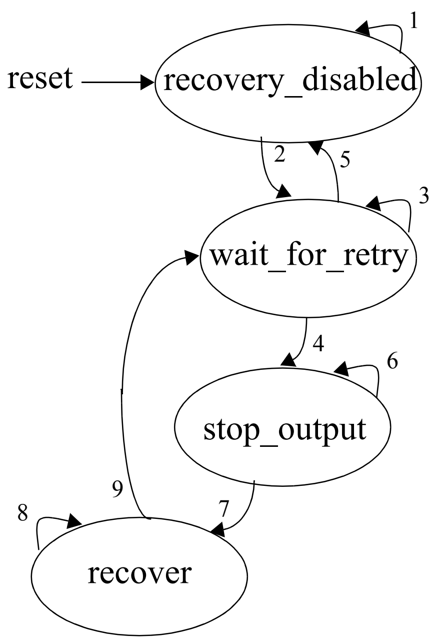


Figure A-4. Output port retry recovery state machine

Table A-4 describes the state transition arcs for Figure A-4.

Table A-4. Output port retry recovery state machine transition table

Arc	Current State	Next state	cause	Comments
1	recovery_disabled	recovery_disabled	Remain in this state until the output port is enabled to receive packets.	This is the initial state after reset. The output port can't be enabled before the training sequence has been completed, and may be controlled through other mechanisms as well, such as a software enable bit.
2	recovery_disabled	wait_for_retry	Output port is enabled.	
3	wait_for_retry	wait_for_retry	Remain in this state until a packet-retry control symbol is received.	The packet-retry control symbol shall be error free.
4	wait_for_retry	stop_output	A packet-retry control symbol has been received.	Start the output port stop procedure.
5	wait_for_retry	recovery_disabled	Output port is disabled.	
6	stop_output	stop_output	Remain in this state until the output port stop procedure is completed.	Clear the "Port ready" state, set the "Output Retry-stopped" state, and stop transmitting new packets.
7	stop_output	recover	Output port stop procedure is complete.	
8	recover	recover	Remain in this state until the internal recovery procedure is completed.	The packet sent with the ackID value returned in the packet-retry control symbol and all subsequent packets shall be re-transmitted. Output port state machines and the outstanding ackID scoreboard shall be updated with this information, then clear the "Output Retry-stopped" state and set the "Port ready" state to restart the output port. Receipt of a packet-not-accepted control symbol or other output port error during this procedure shall cause a clean transition between the retry recovery state machine and the error recovery state machine.
9	recover	wait_for_retry	Internal recovery procedure is complete.	Re-transmission has started, so return to the wait_for_retry state to wait for the next packet-retry control symbol.

A.3 Error Recovery

This section contains the error recovery state machine referred to in Section 1.3.5, "Link Behavior Under Error."

Error recovery actually requires two inter-dependent state machines in order to operate, one associated with the input port and the other with the output port on the two connected devices. The two state machines work together to attempt recovery.

A.3.1 Input port error recovery state machine

There are a variety of recoverable error types described in detail in Section 1.3.5, “Link Behavior Under Error”. The first group of errors are associated with the input port, and consists mostly of corrupt packet and control symbols. An example of a corrupt packet is a packet with an incorrect CRC. An example of a corrupt control symbol is a control symbol where the second 16 bits are not an inversion of the first 16 bits. The recovery state machine for the input port of a RapidIO link is shown inFigure A-5.

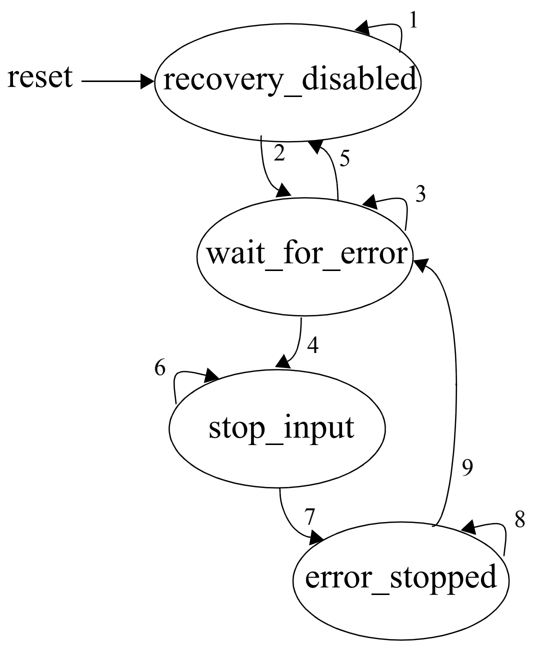


Figure A-5. Input port error recovery state machine

Table A-5 describes the state transition arcs for Figure A-5. The states referenced in the comments in quotes are the RapidIO 8/16 LP-LVDS defined status states, not states in this state machine.

Table A-5. Input port error recovery state machine transition table

Arc	Current State	Next state	cause	Comments
1	recovery_disabled	recovery_disabled	Remain in this state until error recovery is enabled.	This is the initial state after reset. Error recovery can’t be enabled before the training sequence has been completed, and may be controlled through other mechanisms as well, such as a software enable bit.
2	recovery_disabled	wait_for_error	Error recovery is enabled.	
3	wait_for_error	wait_for_error	Remain in this state until a recoverable error is detected.	Detected errors and the level of coverage is implementation dependent.
4	wait_for_error	stop_input	A recoverable error has been detected.	An output port associated error will not cause this transition, only an input port associated error.
5	wait_for_error	recovery_disabled	Error recovery is disabled.	

Table A-5. Input port error recovery state machine transition table(Continued)

Arc	Current State	Next state	cause	Comments
6	stop_input	stop_input	Remain in this state until described input port stop activity is completed.	Send a packet-not-accepted control symbol and, if the error was on a packet, discard the packet and don’t change the expected ackID value. This will force the attached device to initiate recovery. Clear the “Port ready” state and set the “Input Error-stopped” state.
7	stop_input	error_stopped	Input port stop activity is complete.	
8	error_stopped	error_stopped	Remain in this state until a restart-from-error control symbol is received.	The “Input Error-stopped” state causes the input port to silently discard all subsequent incoming packets and ignore all subsequent input port errors.
9	error_stopped	wait_for_error	Received a restart-from-error control symbol.	The restart-from-error control symbol is a link-request/input-status control symbol. Clear the “Input Error-stopped” state and set the “Port ready” state, which will put the input port back in normal operation.

A.3.2 Output port error recovery state machine

The second recoverable group of errors described in Section 1.3.5, “Link Behavior Under Error” is associated with the output port, and is comprised of control symbols that are error-free and indicate that the attached input port has detected a transmission error or some other unusual situation has occurred. An example of this situation is indicated by the receipt of a packet-not-accepted control symbol. Another example is the receipt of a link-request/send-training control symbol, which should cause the error recovery procedure to be followed after responding to the request. The state machine for the output port is shown in Figure A-6.

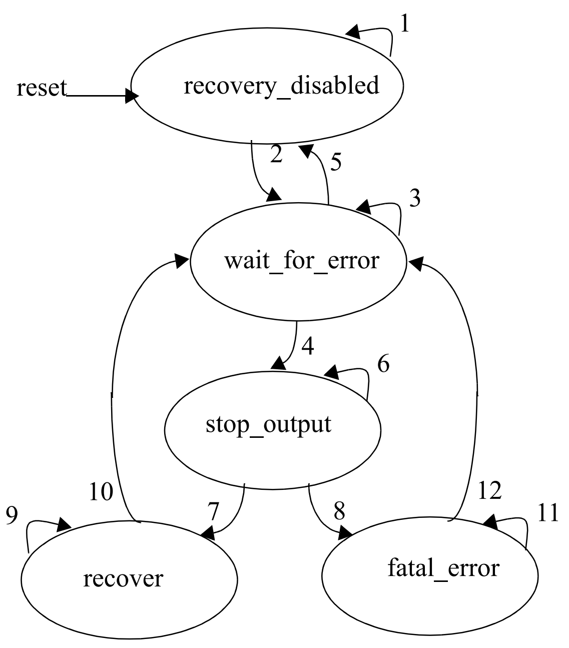


Figure A-6. Output port error recovery state machine

Table A-6 describes the state transition arcs for Figure A-6. The states referenced in the comments in quotes are the RapidIO 8/16 LP-LVDS defined status states, not states in this state machine.

Table A-6. Output port error recovery state machine transition table

Arc	Current State	Next state	cause	Comments
1	recovery_disabled	recovery_disabled	Remain in this state until error recovery is enabled.	This is the initial state after reset. Error recovery can’t be enabled before the training sequence has been completed, and may be controlled through other mechanisms as well, such as a software enable bit.
2	recovery_disabled	wait_for_error	Error recovery is enabled.	
3	wait_for_error	wait_for_error	Remain in this state until a recoverable error is detected.	Detected errors and the level of coverage is implementation dependent.
4	wait_for_error	stop_output	A recoverable error has been detected.	An input port associated error will not cause this transition, only an output port associated error.
5	wait_for_error	recovery_disabled	Error recovery is disabled.	

Table A-6. Output port error recovery state machine transition table(Continued)

Arc	Current State	Next state	cause	Comments
6	stop_output	stop_output	Remain in this state until an exit condition occurs.	Clear the “Port ready” state, set the “Output Error-stopped” state, stop transmitting new packets, and send a link-request/input-status control symbol. Ignore all subsequent output port errors. The input on the attached device is in the “Input Error-stopped” state and is waiting for a link-request/input-status in order to be re-enabled to receive packets. An implementation may wish to time-out several times before regarding a time-out as fatal using a threshold counter or some other mechanism.
7	stop_output	recover	The link-response is received and returned an outstanding ackID value	An outstanding ackID is a value sent out on a packet that has not been acknowledged yet. In the case where no ackIDs are outstanding the returned ackID value shall match the next expected/next assigned ackID value, indicating that the devices are synchronized. Recovery is possible, so follow recovery procedure.
8	stop_output	fatal_error	The link-response is received and returned an ackID value that is not outstanding, or timed out waiting for the link-response.	Recovery is not possible, so start error shutdown procedure.
9	recover	recover	Remain in this state until the internal recovery procedure is completed.	The packet sent with the ackID value returned in the link-response and all subsequent packets shall be re-transmitted. All packets transmitted with ackID values preceding the returned value were received by the attached device, so they are treated as if packet-accepted control symbols have been received for them. Output port state machines and the outstanding ackID scoreboard shall be updated with this information, then clear the “Output Error-stopped” state and set the ‘Port ready” state to restart the output port.

Table A-6. Output port error recovery state machine transition table(Continued)

Arc	Current State	Next state	cause	Comments
10	recover	wait_for_error	The internal recovery procedure is complete.	Re-transmission (if any was necessary) has started, so return to the wait_for_error state to wait for the next error.
11	fatal_error	fatal_error	Remain in this state until error shutdown procedure is completed.	Clear the “Output Error-stopped” state, set the “Port Error” state, and signal a system error.
12	fatal_error	wait_for_error	Error shutdown procedure is complete.	Return to the wait_for_error state even though the output port is shut off.

Partition III: Common Transport Specification

III Partition III

Partition III is intended for users who need to understand the common transport architecture of the RapidIO interconnect.

III.1 Overview

The *Common Transport Specification* defines a standard transport mechanism. In doing so, it specifies the header information added to a RapidIO logical packet and the way the header information is interpreted by a switching fabric. The RapidIO interconnect defines this mechanism independent of a physical implementation. The physical features of an implementation using RapidIO are defined by the requirements of the implementation, such as I/O signaling levels, interconnect topology, physical layer protocol, and error detection. These requirements are specified in the appropriate RapidIO physical layer specification.

This transport specification is also independent of any RapidIO logical layer specification.

III.2 Contents

Partition III: Common Transport Specification contains two chapters:

- Chapter 1, “Transport Format Description,” describes the routing methods used in RapidIO for sending packets across the systems of switches described in this chapter.
- Chapter 2, “Common Transport Registers,” describes the visible register set that allows an external processing element to determine the capabilities, configuration, and status of a processing element using this RapidIO transport layer definition.

1 Chapter 1 - Transport Format Description

This chapter contains the transport format definition for the RapidIO Common Transport Specification. Three transport fields are added to the packet formats described in the RapidIO logical specifications. The transport formats are intended to be fabric independent so the system interconnect can be anything required for a particular application; therefore all descriptions of the transport fields and their relationship with the logical packets are shown as bit streams.

1.1 System Topology

RapidIO is intended to be interconnect fabric independent. This section describes several of the possible system topologies and routing methodologies allowed by the processing element models described in the Models chapters of the different Logical Specifications.

1.1.1 Switch-Based Systems

A RapidIO system can be organized around the concept of switches. Figure 1-1 shows a small system in which five processing elements are interconnected through two switches. A logical packet sent from one processing element to another is routed through the interconnect fabric by the switches by interpreting the transport fields. Because a request usually requires a response, the transport fields must somehow indicate the return path from the requestor to the responder.

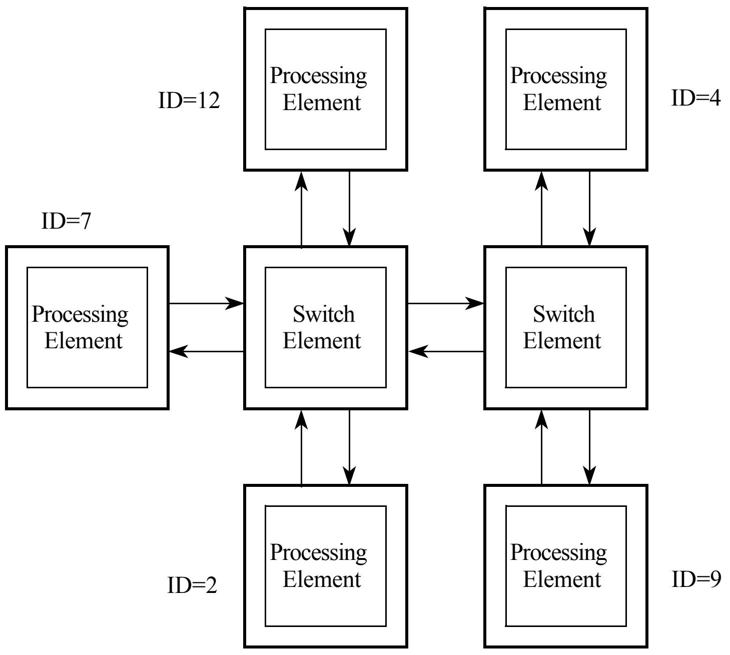


Figure 1-1. A Small Switch-Based System

1.1.2 Ring-Based Systems

A simplification of the switch structure is a ring as shown in Figure .1-2. A ring is a point-to-point version of a common bus; therefore, it is required to have a unique identifier for each processing element in the system. A packet put onto the ring contains the source and destination identifier in the transport fields. Each packet issued is examined by the downstream processing element. If that processing element's identifier matches that of the destination, it removes the packet from the ring for processing. If the destination identifier does not match the packet, it is passed to the next processing element in the ring.

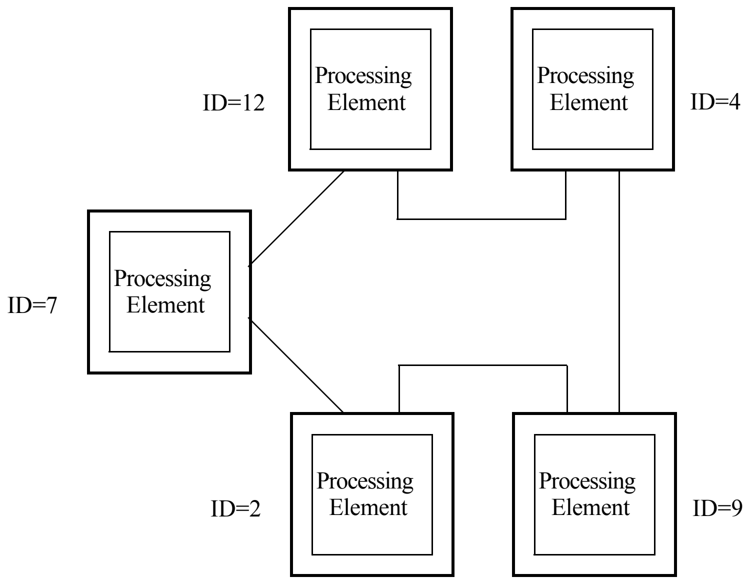


Figure 1-2. A Small Ring-Based System

1.2 System Packet Routing

There are many algorithms that can be used for routing through a system. The *RapidIO Common Transport Specification* requires device identifier based packet routing. Each directly addressable device in the system shall have one or more unique device identifiers. When a packet is generated, the device ID of the destination of the packet is put in the packet header. The device ID of the source of the packet is also put in the packet header for use by the destination when generating response packets. When the destination of a request packet generates a response packet, it swaps the source and destination fields from the request, making the original source the new destination and itself the new source. Packets are routed through the fabric based on the destination device ID.

One method of routing packets in a switch fabric using device ID information incorporates routing tables. Each switch in the interconnect fabric contains a table that tells the switch how to route every destination ID from an input port to the proper output port. The simplest form of this method allows only a single path from every processing element to every other processing element. More complex forms of this method may allow adaptive routing for redundancy and congestion relief. However, the actual method by which packets are routed between the input of a switch and the output of a switch is implementation dependent.

1.3 Field Alignment and Definition

The *RapidIO Common Transport Specification* adds a transport type (tt) field to the logical specification packet that allows four different transport packet types to be specified. The tt field indicates which type of additional transport fields are added to the packet.

The three fields (tt, destinationID, and sourceID) added to the logical packets allow for two different sizes of the device ID fields, a large (16-bit), and a small (8-bit), as shown in Table 1-1. The two sizes of device ID fields allow two different system scalability points to optimize packet header overhead, and only affix additional transport field overhead if the additional addressing is required. The small device ID fields allow a maximum of 256 devices to be attached to the fabric. The large device ID fields allow systems with up to 65,536 devices.

Table 1-1. tt Field Definition

tt	Definition
0b00	8-bit deviceID fields
0b01	16-bit deviceID fields
0b10	Reserved
0b11	Reserved

Figure 1-3 shows the transport header definition bit stream. The shaded fields are the bits associated with the logical packet definition that are related to the transport bits. Specifically, the field labeled “Logical ftype” is the format type field defined in the logical specifications. This field comprises the first four bits of the logical packet. The second logical field shown (“Remainder of logical packet”) is the remainder of the logical packet of a size determined by the logical specifications, not including the logical ftype field which has already been included in the combined bit stream. The unshaded fields (tt=0b00 or tt=0b01 and destinationID and sourceID fields) are the transport fields added to the logical packet by the *RapidIO Common Transport Specification*.

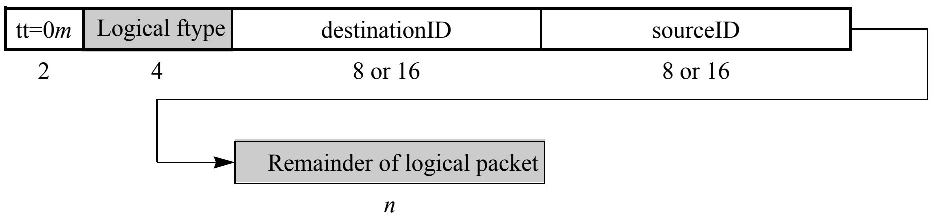


Figure 1-3. Destination-Source Transport Bit Stream

1.3.1 Routing Maintenance Packets

Routing maintenance packets in a switch-based network may be difficult because a switch processing element may not have its own device ID. An alternative method of addressing for maintenance packets for these devices uses an additional hop_count field in the packet to specify the number of switches (or hops) into the network from the issuing processing element that is being addressed. Whenever a switch processing element that does not have as associated device ID receives a maintenance packet it examines the hop_count field. If the received hop_count is zero, the access is for that switch. If the hop_count is not zero, it is decremented and the packet is sent out of the switch according to the destinationID field. This method allows easy access to any intervening switches in the path between two addressable processing elements. However, since maintenance response packets are always targeted at an end point, the hop_count field shall always be assigned a value of 0xFF by the source of the packets to prevent them from being inadvertently accepted by an intervening device. Figure 1-4 shows the transport layer fields added to a maintenance logical packet. Maintenance logical packets can be found in the *Partition I: Input/Output Logical Specification*.

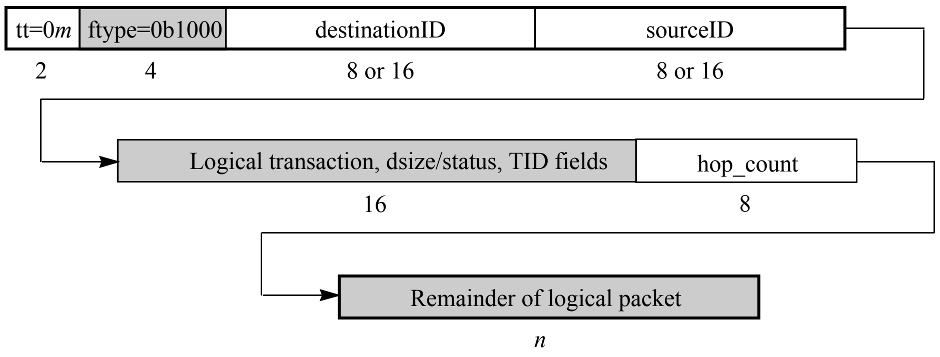

tt=0m
ftype=0b1000
destinationID
sourceID
2
4
8 or 16
8 or 16
Logical transaction, dsize/status, TID fields
hop_count
16
8
Remainder of logical packet
n

Figure 1-4. Maintenance Packet Transport Bit Stream

2 Chapter 2 - Common Transport Registers

This chapter describes the visible register set that allows an external processing element to determine the capabilities, configuration, and status of a processing element using this transport layer definition. This chapter only describes registers or register bits defined by this specification. Refer to the other RapidIO logical, transport, and physical specifications of interest to determine a complete list of registers and bit definitions. All registers are 32-bits and aligned to a 32-bit boundary.

2.1 Register Summary

Table 2-1 shows the register address map for this RapidIO specification. These capability registers (CARs) and command and status registers (CSRs) can be accessed using *Partition I: Input/Output Logical Specification* maintenance operations. Any register offsets not defined are considered reserved for this specification unless otherwise stated. Other registers required for a processing element are defined in other applicable RapidIO specifications and by the requirements of the specific device and are beyond the scope of this specification. Read and write accesses to reserved register offsets shall terminate normally and not cause an error condition in the target device. Writes to CAR (read-only) space shall terminate normally and not cause an error condition in the target device.

Register bits defined as reserved are considered reserved for this specification only. Bits that are reserved in this specification may be defined in another RapidIO specification.

Table 2-1. Common Transport Register Map

Configuration Space Byte Offset	Register Name (Word 0)	Register Name (Word 1)
0x0-8	Reserved	
0x10	Processing Element Features CAR	Reserved
0x18–58	Reserved	
0x60	Base Device ID CSR	Reserved
0x68	Host Base Device ID Lock CSR	Component Tag CSR
0x70–F8	Reserved	

Table 2-1. Common Transport Register Map(Continued)

Configuration Space Byte Offset	Register Name (Word 0)	Register Name (Word 1)
0x100–FFF8	Extended Features Space	
0x10000–FFFFF8	Implementation-defined Space	

2.2 Reserved Register and Bit Behavior

Table 2-2 describes the required behavior for accesses to reserved register bits and reserved registers for the RapidIO register space,

Table 2-2. Configuration Space Reserved Access Behavior

Byte Offset	Space Name	Item	Initiator behavior	Target behavior
0x0–3C	Capability Register Space (CAR Space - this space is read-only)	Reserved bit	read - ignore returned value[1]	read - return logic 0
			write -	write - ignored
		Implementation-defined bit	read - ignore returned value unless implementation-defined function understood	read - return implementation-defined value
			write -	write - ignored
		Reserved register	read - ignore returned value	read - return logic 0s
			write -	write - ignored
0x40–FC	Command and Status Register Space (CSR Space)	Reserved bit	read - ignore returned value	read - return logic 0
			write - preserve current value[2]	write - ignored
		Implementation-defined bit	read - ignore returned value unless implementation-defined function understood	read - return implementation-defined value
			write - preserve current value if implementation-defined function not understood	write - implementation-defined
		Reserved register	read - ignore returned value	read - return logic 0s
			write -	write - ignored
0x100–FFFC	Extended Features Space	Reserved bit	read - ignore returned value	read - return logic 0
			write - preserve current value	write - ignored
		Implementation-defined bit	read - ignore returned value unless implementation-defined function understood	read - return implementation-defined value
			write - preserve current value if implementation-defined function not understood	write - implementation-defined
		Reserved register	read - ignore returned value	read - return logic 0s
			write -	write - ignored
0x10000–FFFFFC	Implementation-defined Space	Reserved bit and register	All behavior implementation-defined	

1. Do not depend on reserved bits being a particular value; use appropriate masks to extract defined bits from the read value.
2. All register writes shall be in the form: read the register to obtain the values of all reserved bits, merge in the desired values for defined bits to be modified, and write the register, thus preserving the value of all reserved bits.

2.3 Capability Registers (CARs)

Every processing element shall contain a set of registers that allows an external processing element to determine its capabilities using the I/O logical maintenance read operation. All registers are 32 bits wide and are organized and accessed in 32-bit (4 byte) quantities, although some processing elements may optionally allow larger accesses. CARs are read-only. Refer to Table 2-2 for the required behavior for accesses to reserved registers and register bits.

CARs are big-endian with bit 0 and Word 0 respectively the most significant bit and word.

2.3.1 Processing Element Features CAR (Offset 0x10 Word 0)

The processing element features CAR identifies the major functionality provided by the processing element. The bit settings are shown in Table .2-3.

Table 2-3. Bit Settings for Processing Element Features CAR

Bits	Name	Description
0–26	—	Reserved
27	Common transport large system support	0b0 - PE does not support common transport large systems 0b1 - PE supports common transport large systems
28–31	—	Reserved

2.4 Command and Status Registers (CSRs)

A processing element shall contain a set of registers that allows an external processing element to control and determine status of its internal hardware. All registers are 32 bits wide and are organized and accessed in the same way as the CARs. Refer to Table 2-2 for the required behavior for accesses to reserved registers and register bits.

2.4.1 Base Device ID CSR (Offset 0x60 Word 0)

The base device ID CSR contains the base device ID values for the processing element. A device may have multiple device ID values, but these are not defined in a standard CSR. The bit settings are shown in Table 2-4.

Table 2-4. Bit Settings for Base Device ID CSR

Bits	Name	Reset Value	Description
0-7	—		Reserved
8-15	Base_deviceID	see footnote 1	This is the base ID of the device in a small common transport system (end point devices only)
16–31	Large_base_deviceID	see footnote 2	This is the base ID of the device in a large common transport system (only valid for end point device and if bit 27 of the Processing Element Features CAR is set)

1. The Base_deviceID reset value is implementation dependent
2. The Large_base_deviceID reset value is implementation dependent

2.4.2 Host Base Device ID Lock CSR (Offset 0x68 Word 0)

The host base device ID lock CSR contains the base device ID value for the processing element in the system that is responsible for initializing this processing element. The Host_base_deviceID field is a write-once/reset-able field which provides a lock function. Once the Host_base_deviceID field is written, all subsequent writes to the field are ignored, except in the case that the value written matches the value contained in the field. In this case, the register is re-initialized to 0xFFFF. After writing the Host_base_deviceID field a processing element must then read the Host Base

Device ID Lock CSR to verify that it owns the lock before attempting to initialize this processing element. The bit settings are shown in Table 2-5.

Table 2-5. Bit Settings for Host Base Device ID Lock CSR

Bits	Name	Reset Value	Description
0-15	—		Reserved
16–31	Host_base_deviceID	0xFFFF	This is the base device ID for the PE that is initializing this PE.

2.4.3 Component Tag CSR (Offset 0x68 Word 1)

The component tag CSR contains a component tag value for the processing element and can be assigned by software when the device is initialized. It is especially useful for labeling and identifying devices that are not end points and do not have device ID registers. The bit settings are shown in Table .2-6.

Table 2-6. Bit Settings for Component ID CSR

Bits	Name	Reset Value	Description
0–31	component_tag	All 0s	This is a component tag for the PE.

Partition IV
Physical Layer 8/16 LP-LVDS Specification

IV Partition IV

Partition IV is intended for users who need to understand the physical 8/16 LP-LVDS architecture of the RapidIO interconnect.

IV.1 Overview

The *Physical Layer 8/16 LP-LVDS Specification* is RapidIO's physical layer specification that addresses the physical layer requirements for a RapidIO device.

This specification defines a full duplex interface with 8-bit or 16-bit unidirectional low voltage differential signaling (LVDS) differential ports, assuming that the interface (called a link) only communicates directly with one other device on that interface. Systems are built out of switch processing elements as described in *Partition III: Common Transport Specification,* and packets are sent between electrically connected devices through the network one link at a time. The flow control, error management, and signal acknowledgement protocols between linked devices are handled through control symbols.

RapidIO is targeted toward memory-mapped distributed memory systems. A message passing programming model is supported to enable distributed I/O processing and is described in *Partition II: Message Passing Logical Specification. Partition I: Input/Output Logical Specification* defines the basic input/output system architecture of RapidIO.

IV.2 Contents

Following are the contents of *Partition IV: Physical Layer 8/16 LP-LVDS Specification:*

- Chapter 1, "Physical Layer Protocol," describes the physical layer protocol for packet delivery to the RapidIO fabric, including packet transmission, flow control, error management, and link maintenance protocols.
- Chapter 2, "Packet and Control Symbol Transmission," defines packet and control symbol delineation and alignment on the physical port and mechanisms to control the pacing of a packet.
- Chapter 3, "Control Symbol Formats," explains the physical layer control formats that manage the packet delivery protocols mentioned in Chapter 2.
- Chapter 4, "8/16 LP-LVDS Registers," describes the register set that allows an external processing element to determine the physical capabilities and status of an 8/16 LP-LVDS RapidIO implementation.
- Chapter 5, "System Clocking Considerations," discusses the RapidIO synchronous clock and how it is distributed in a typical switch configuration.
- Chapter 6, "Board Routing Guidelines," explains board layout guidelines and application environment considerations for the RapidIO architecture.
- Chapter 7, "Signal Descriptions," contains the signal pin descriptions for a typical RapidIO end point device.
- Chapter 8, "Electrical Specifications," describes the LVDS electrical specifications of the RapidIO 8/16 LP-LVDS device.
- Annex A, "Interface Management (Informative)," contains information pertinent to interface management in a RapidIO system, including SECDED error tables, error recovery flowcharts, and link initialization and packet retry mechanisms.

1 Chapter 1 - Physical Layer Protocol

This chapter describes the physical layer protocol for packet delivery to the interconnect fabric including packet transmission, flow control, error management, and other system functions. See the user's manual or implementation specification for specific implementation details of a device.

1.1 Packet Exchange Protocol

This physical layer 8/16 LP-LVDS specification defines an exchange of packet and acknowledgment control symbols in which a destination or intermediate processing element (such as a switch) acknowledges receipt of a request or response packet from a source.

If a packet cannot be accepted for any reason, an acknowledgment control symbol indicates that the original packet and any already transmitted subsequent packets should be resent. This behavior provides a flow control and transaction ordering mechanism between processing elements. Figure 1-1 shows an example of transporting a request and response packet pair across an interconnect fabric with acknowledgments between the link transmitter/receiver pairs along the way. This allows flow control and error handling to be managed between each electrically connected device pair rather than between the original source and final target of the transaction. A device shall transmit an acknowledge control symbol for a request before the response transaction corresponding to that request.

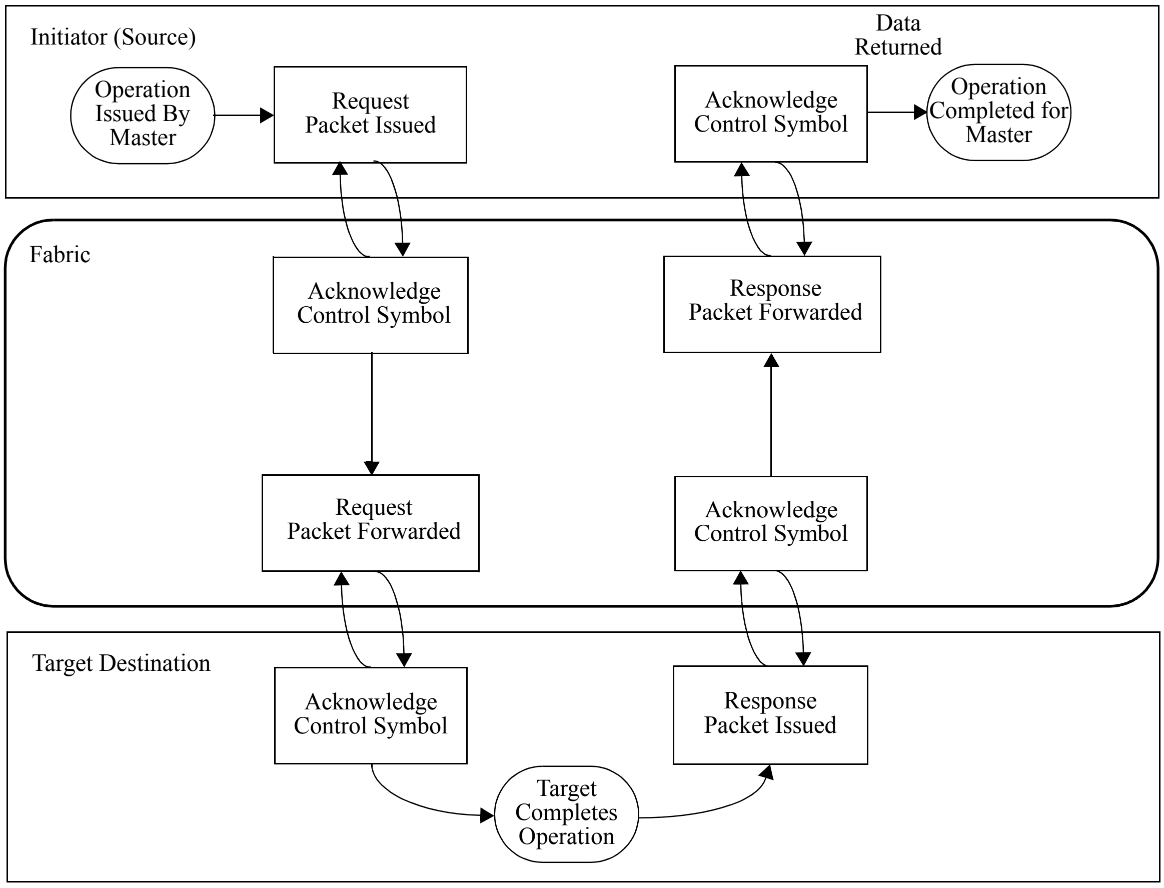

Initiator (Source)
Data Returned
Operation Issued By Master
Request Packet Issued
Acknowledge Control Symbol
Operation Completed for Master
Fabric
Acknowledge Control Symbol
Response Packet Forwarded
Request Packet Forwarded
Acknowledge Control Symbol
Target Destination
Acknowledge Control Symbol
Response Packet Issued
Target Completes Operation

Figure 1-1. Example Transaction with Acknowledge

1.1.1 Packet and Control Alignment

All packets defined by the combination of this specification and the appropriate logical and transport specifications are aligned to 16-bit boundaries, however, all packets and control symbols sent over the 8-bit and 16-bit ports are aligned to 32-bit boundaries. This alignment allows devices to work on packets using a larger internal width thus requiring lower core operating frequencies. Packets that are not naturally aligned to a 32-bit boundary are padded.
See Figure 1-11 and Figure 1-12 for examples of padded packets. Control symbols are nominally 16-bit quantities, but are defined as a 16-bit control symbol followed by a bit-wise inverted copy of itself to align it to the 32-bit boundary. This, in turn, adds error detection capability to the interface. These 32-bit quantities are referred to as aligned control symbols.

The 16-bit wide port is compatible with an 8-bit wide port. If an 8-bit wide port is properly connected to a 16-bit wide port, the port will function as an 8-bit interface between the devices. Port width connections are described in Chapter 7, “Signal Descriptions”.

1.1.2 Acknowledge Identification

A packet requires an identifier to uniquely identify its acknowledgment. This identifier, known as the acknowledge ID (or ackID), is three bits, allowing for a range of one to eight outstanding unacknowledged request or response packets between adjacent processing elements, however only up to seven outstanding unacknowledged packets are allowed at any one time. The ackIDs are assigned sequentially (in increasing order, wrapping back to 0 on overflow) to indicate the order of the packet transmission. The acknowledgments themselves are a number of aligned control symbols defined in Chapter 3, “Control Symbol Formats.”

1.2 Field Placement and Definition

This section contains the 8/16 LP-LVDS specification for the additional physical layer bit fields and control symbols required to implement the flow control, error management, and other specified system functions.

1.2.1 Flow Control Fields Format

The fields used to control packet flow in the system are described in Table 1-1.

Table 1-1. Fields that Control Packet Flow

Field	Description
S	0b0 - RapidIO request or response packet 0b1 - Physical layer control symbol
$\overline{S}$	Inverse of S-bit for redundancy (odd parity bit)
ackID	Acknowledge ID is the packet identifier for acknowledgments back to the packet sender—see Section
prio	Sets packet priority: 0b00 - lowest priority 0b01 - medium priority 0b10 - high priority 0b11 - highest priority See Section for an explanation of prioritizing packets
buf_status	Specifies the number of available packet buffers in the receiving device. See Section and Table 0-1.
stype	Control symbol type—see Chapter 3, “Control Symbol Formats” for definition.
rsrv	Reserved

Table 0-1. buf_status Field Definition

buf_status Encoding Value	Description
0b0000	Specifies the number of maximum length packets that the port can accept without issuing a retry due to a lack of resources. The value of buf_status in a control symbol is the number of maximum packets that can be accepted, inclusive of the effect of the packet being accepted or retried. **Value 0-13**: The encoding value specifies the number of new maximum sized packets the receiving device can receive. The value 0, for example, signifies that the downstream device has no available packet buffers (thus is not able to hold any new packets). **Value 14**: The value 14 signifies that the downstream device can receive 14 or more new maximum sized packets. **Value 15**: The downstream device can receive an undefined number of maximum sized packets, and relies on the retry protocol for flow control.
0b0001	
0b0010	
0b0011	
0b0100	
0b0101	
0b0110	
0b0111	
0b1000	
0b1001	
0b1010	
0b1011	
0b1100	
0b1101	
0b1110	
0b1111	

Figure 1-2 shows the format for the physical layer fields for packets. In order to pad packets to the 16-bit boundary there are three reserved bits in a packet's physical layer fields. These bits are assigned to logic 0 when generated and ignored when received.

S=0	ackID	rsrv=0	$\overline{S}$=1	rsrv=00	prio
1	3	1	1	2	2

Figure 1-2. Packet Physical Layer Fields Format

Figure 1-3 shows the basic format for the physical layer fields for control symbols. In order to pad the control symbol to the 16-bit boundary there are four reserved bits in the control symbol. These bits are assigned to logic 0 when generated and ignored when received. The field formats for all control symbols are defined in Chapter 3, "Control Symbol Formats."

S=1	ackID	rsrv=0	$\overline{S}$=0	rsrv=000	buf_status	stype
1	3	1	1	3	4	3

Figure 1-3. Control Symbol Physical Layer Fields Format

Figure 1-4 shows how the physical layer fields are prefixed to the combined transport and logical layer packet.

S=0	ackID	rsrv=0	$\overline{S}$=1	rsrv=00	prio	tt	ftype	Remainder of transport & logical fields
1	3	1	1	2	2	2	4	*n*

Figure 1-4. Flow Control Fields Bit Stream

The unshaded fields are the physical layer fields defined by this physical specification. The shaded fields are the bits associated with the combined transport and logical transaction definitions. The first transport and logical field shown is the two bit tt field specified in the *RapidIO Common Transport Specification*. The second field is the four bit format type (ftype) defined in the logical specifications. The third combined field is the remainder of the transport and logical packet of a size determined by those specifications.

1.2.2 Packet Priority and Transaction Request Flows

Each packet has a priority that is assigned by the end point processing element that is the source of (initiates) the packet. The priority is carried in the prio field of the packet and has four possible values, 0, 1, 2 or 3. Packet priority increases with the priority value with 0 being the lowest priority and 3 being the highest. Packet priority is used in Rapid IO for several purposes which include transaction ordering and deadlock prevention.

When a transaction is encapsulated in a packet for transmission, the transaction request flow indicator (flowID) of the transaction is mapped into the prio field of the packet. Transaction request flows A and B are mapped to priorities 0 and 1 respectively and transaction request flows C and above are mapped to priority 2 as specified in Table 1-3.

The mapping of transaction request flow onto packet priority (prio) allows a Rapid IO transport fabric to maintain transaction request flow ordering without the fabric having any knowledge of transaction types or their interdependencies. This allows a Rapid IO fabric to be forward compatible as the types and functions of transactions evolve. A fabric can maintain transaction request flow ordering by simply maintaining the order of packets with the same priority for each path through the fabric and can maintaining transaction request flow priority by never allowing a lower priority packet to pass a higher priority packet taking the same path through the fabric.

Table 1-3. Transaction Request Flow to Priority Mapping

Flow	System Priority	Request Packet Priority	Response Packet Priority
C or higher	Highest	2	3
B	Next	1	2 or 3
A	Lowest	0	1, 2, or 3

1.2.3 Transaction and Packet Delivery

Certain physical layer fields and a number of control symbols are used for handling flow control. One physical layer field contains the ackID field (Table), which is assigned by the sending processing element, and expected by the receiving processing element, in a sequential fashion.

Packets shall be accepted by the receiving processing element only when ackID values of successive packets occur in the specified sequence. The receiving processing element signals the acceptance of a packet by returning a packet-accepted control symbol to the sender. This order allows a device to detect when a packet has been lost and also provides a mechanism to maintain ordering.

A device that retries a packet (by returning a packet-retry control symbol to the sender) due to some temporary internal condition shall silently discard all new incoming packets until it receives a restart-from-retry control symbol from the sender. The sender then retransmits all packets starting from the retried ackID, reestablishing the proper ordering between the devices. The packet sent with the retried ackID may be the original retried packet or a higher priority packet, if one is available, allowing higher priority packets to bypass lower priority packets across the link. This behavior is shown in an example state machine in Section A.2, “Packet Retry Mechanism.”

Similarly, if a receiving processing element encounters an error condition, it shall return a packet-not-accepted control symbol, indicating an error condition, to the sender. It shall also silently discard all new incoming packets. If the error condition is due to a transmission error the sender may able to recover from the effects of the error condition. The error recovery mechanism is described in Section 1.3.5.

A retried transaction shall eventually be retransmitted by the sending device.

1.2.3.1 Transaction and Packet Delivery Ordering Rules

The rules specified in this section are required for the physical layer to support the transaction ordering rules specified in the logical layer specifications.

Transaction Delivery Ordering Rules:

1. The physical layer of an end point processing element port shall encapsulate in packets and forwarded to the RapidIO fabric transactions comprising a given transaction request flow in the same order that the transactions were received from the transport layer of the processing element.
2. The physical layer of an end point processing element port shall ensure that a higher priority request transaction that it receives from the transport layer of the processing element before a lower priority request transaction with the same sourceID and the same destinationID is forwarded to the fabric before the lower priority transaction.
3. The physical layer of an end point processing element port shall deliver transactions to the transport layer of the processing element in the same order that the packetized transactions were received by the port.

Packet Delivery Ordering Rules:

1. A packet initiated by a processing element shall not be considered committed to the RapidIO fabric and does not participate in the packet delivery ordering rules until the packet has been accepted by the device at the other end of the link. (RapidIO does not have the concept of delayed or deferred transactions. Once a packet is accepted into the fabric, it is committed.)
2. A switch shall not alter the priority of a packet.
3. Packet forwarding decisions made by a switch processing element shall provide a consistent output port selection which is based solely on the value of the destinationID field carried in the packet.
4. A switch processing element shall not change the order of packets comprising a transaction request flow (packets with the same sourceID, the same destinationID, the same priority and ftype != 8) as the packets pass through the switch.
5. A switch processing element shall not allow lower priority non-maintenance packets (ftype != 8) to pass higher priority non-maintenance packets with the same sourceID and destinationID as the packets pass through the switch.
6. A switch processing element shall not allow a priority N maintenance packet (ftype = 8) to pass another maintenance packet of priority N or greater that takes the same path through the switch (same switch input port and same switch output port).

1.2.4 Resource Allocation

This section defines RapidIO LP-LVDS link level flow control. The flow control operates between each pair of ports connected by an LP-LVDS link. The purpose of link level flow control is to prevent the loss of packets due to a lack of buffer space in a link receiver.

The LP-LVDS protocol defines two methods or modes of flow control. These are named receiver-controlled flow control and transmitter-controlled flow control. Every RapidIO LP-LVDS port shall support receiver-controlled flow control. LP-LVDS ports may optionally support transmitter-controlled flow control.

1.2.4.1 Receiver-Controlled Flow Control

Receiver-controlled flow control is the simplest and most basic method of flow control. In this method, the input side of a port controls the flow of packets from its link partner by accepting or rejecting (retrying) packets on a packet by packet basis. The receiving port provides no information to its link partner about the amount of buffer space it has available for packet reception.

As a result, its link partner transmits packets with no *a priori* expectation as to whether a given packet will be accepted or rejected. A port signals its link partner that it is operating in receiver-controlled flow control mode by setting the buf_status field to all 1's in every control symbol containing the field that the port transmits. This method is named receiver-controlled flow control because the receiver makes all of the decisions about how buffers in the receiver are allocated for packet reception.

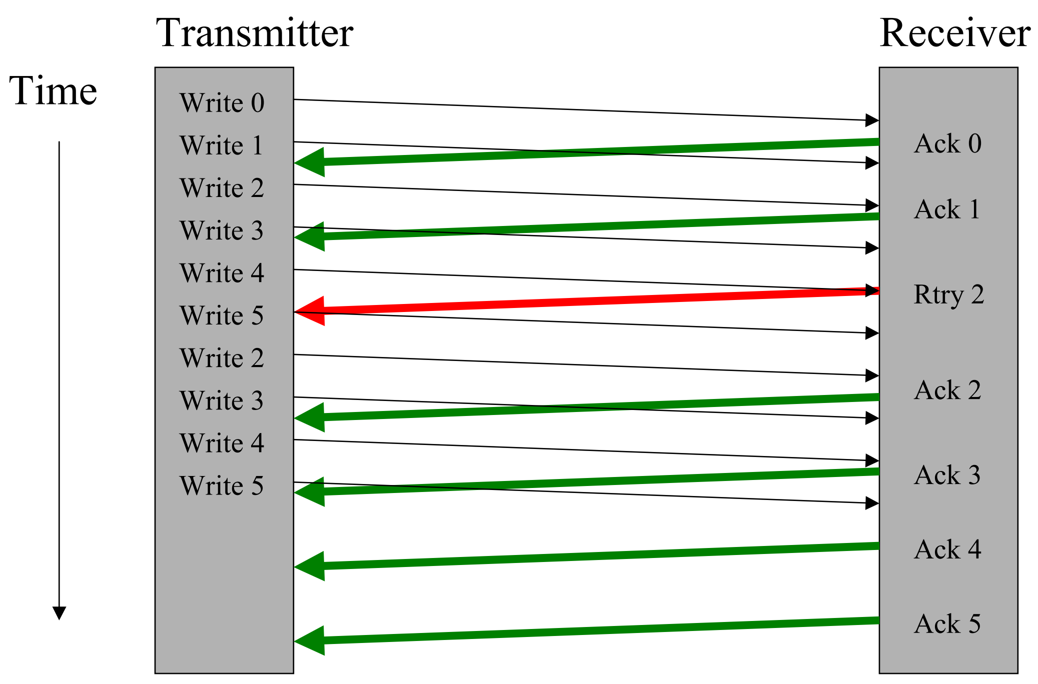
A port operating in receiver-controlled flow control mode accepts or rejects each inbound error-free packet based on whether the receiving port has enough buffer space available at the priority level of the packet. If there is enough buffer space available, the port accepts the packet and transmits a packet-accepted control symbol to its link partner that contains the ackID of the accepted packet in its packet_ackID field. This informs the port's link partner that the packet has been received without detected errors and that it has been accepted by the port. On receiving the packet-accepted control symbol, the link partner discards its copy of the accepted packet freeing buffer space in the partner.

If buffer space is not available, the port rejects the packet. When a port rejects (retries) an error-free packet, it behaves as described in Section 1.2.3, "Transaction and Packet Delivery". As part of the recovery process, the port sends a packet-retry control symbol to its link partner indicating that the packet whose ackID is in the packet_ackID field of the control symbol and all packets subsequently transmitted by the port have been discarded by the link partner and must all be retransmitted. The control symbol also indicates that the link partner is temporarily out of buffers for packets of priority less than or equal to the priority of the retried packet.

A port that receives a packet-retry control symbol also behaves as described in Section 1.2.3. As part of the recovery process, the port receiving the packet-retry control symbol sends a restart-from-retry control symbol which causes its link partner to resume packet reception. The ackID assigned to that first packet transmitted after the restart-from-retry control symbol is the ackID of the packet that was retried.

Figure 1.5 shows an example of receiver-controlled flow control operation. In this example the transmitter is capable of sending packets faster than the receiver is able to absorb them. Once the transmitter has received a retry for a packet, the transmitter may elect to cancel any packet that is presently being transmitted since it will be discarded anyway. This makes bandwidth available for any higher priority packets that may be pending transmission.

Figure 1-5. Receiver-Controlled Flow Control



1.2.4.2 Transmitter-Controlled Flow Control

In transmitter-controlled flow control, the receiving port provides information to its link partner about the amount of buffer space it has available for packet reception. With this information, the sending port can allocate the use of the receiving port's receive buffers according to the number and priority of packets that the sending port has waiting for transmission without concern that one or more of the packets shall be forced to retry.

A port signals its link partner that it is operating in transmitter-controlled flow control mode by setting the buf_status field to a value different from all 1's in every control symbol containing the field that the port transmits. This method is named transmitter-controlled flow control because the transmitter makes almost all of the decisions about how the buffers in the receiver are allocated for packet reception.

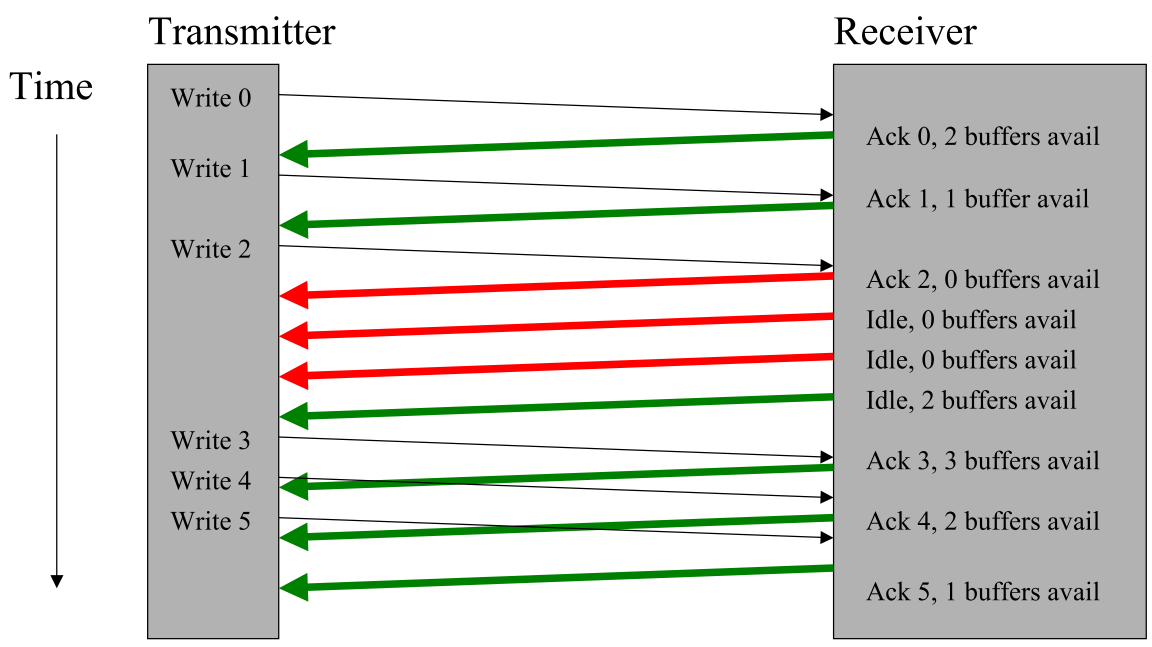
The number of free buffers that a port has available for packet reception is conveyed to its link partner by the value of the buf_status field in control symbols that the port transmits. The value conveyed by the buf_status field is the number of maximum length packet buffers currently available for packet reception up to the limit that can reported in the field. If a port has more buffers available than the maximum value that can be reported in the buf_status field, the port sets the field to that maximum value. A port may report a smaller number of buffers than it actually has available, but it shall not report a greater number.

A port informs its link partner when the number of free buffers available for packet reception changes. The new value of buf_status is conveyed in the buf_status field in every control symbol containing the field that the port transmits. Each change in the number of free buffers a port has available for packet reception need not be conveyed to the link partner.

A port whose link partner is operating in transmitter-control flow control mode should never receive a packet-retry control symbol from its link partner unless the port has transmitted more packets than its link partner has receive buffers, violated the rules that all input buffer may not be filled with low priority packets or there is some fault condition. If a port whose link partner is operating in transmitter-control flow control mode receives a packet-retry control symbol, the output side of the port behaves as described in Section 1.2.3.

A simple example of transmitter-controlled flow control is shown in Figure 1-6.

Figure 1-6. Transmitter-Controlled Flow Control



1.2.4.3 Receive Buffer Management

In transmitter-controlled flow control, the transmitter manages the packet receive buffers in the receiver. This may be done in a number of ways, but the selected method shall not violate the rules in Section 1.2.2, “Packet Priority and Transaction Request Flows” concerning the acceptance of packets by ports.

One possible implementation to organize the buffers is establish watermarks and use them to progressively limit the packet priorities that can be transmitted as the effective number of free buffers in the receiver decreases. For example, RapidIO LP-LVDS has four priority levels. Three non-zero watermarks are needed to progressively limit the packet priorities that may be transmitted as the effective number of free buffers decreases. Designate the three watermarks as WM0, WM1, and WM2 where WM0 > WM1 > WM2 > 0 and employ the following rules.

If free_buffer_count >= WM0, all priority packets may be transmitted.

If WM0 > free_buffer_count >= WM1, only priority 1, 2, and 3 packets may be transmitted.

If WM1 > free_buffer_count >= WM2, only priority 2 and 3 packets may be transmitted.

If WM2 > free_buffer_count, only priority 3 packets may be transmitted.

If this method is implemented, the initial values of the watermarks may be set by the hardware at reset as follows.

WM0 = 4

WM1 = 3

WM2 = 2

These initial values may be modified by hardware or software. The modified watermark values shall be based on the number of free buffers reported in the buf_status field of idle control symbols received by the port following link initialization and before the start of packet transmission.

The three watermark values and the number of free buffers reported in the buf_status field of idle control symbols received by the port following link initialization and before the start of packet transmission may be stored in a CSR. Since the maximum value of each of these four items is 14, each will fit in an 8-bit field and all four will fit in a single 32-bit CSR. If the watermarks are software setable, the three watermark fields in the CSR should be writable. For the greatest flexibility, a watermark register should be provided for each port on a device.

1.2.4.4 Effective Number of Free Receive Buffers

The number of buffers available in a port's link partner for packet reception is typically less than the value of the buf_status field most recently received from the link partner. The value in the buf_status field does not account for packets that have been transmitted by the port but not acknowledged by its link partner. The variable free_buffer_count is defined to be the effective number of free buffers available in the link partner for packet reception. The value of free_buffer_count shall be determined according to the following rules.

The port shall maintain a count of the packets that it has transmitted but that have not been acknowledged by its link partner. This count is named the outstanding_packet_count.

After link initialization and before the start of packet transmission,

```
If (received_buf_status < 15) {
flow_control_mode = transmitter;
free_buffer_count = received_buf_status;
outstanding_packet_count = 0;
}
else
flow_control_mode = receiver;
```

When a packet is transmitted by the port,

```
outstanding_packet_count =
outstanding_packet_count + 1;
```

When a control symbol containing a buf_status field is received by the port,

```
free_buffer_count = received_buf_status -
outstanding_packet_count;
```

When a packet-accepted control symbol is received by the port indicating that a packet has been accepted by the link partner,

```
Outstanding_packet_count =
Outstanding_packet_count - 1;
free_buffer_count = received_buf_status -
outstanding_packet_count;
```

When a packet-retry control symbol is received by the port indicating that a packet has been forced by the link partner to retry,

Outstanding_packet_count = 0;

free_buffer_count = received_buf_status;

When a packet-not-accepted control symbol is received by the port indicating that a packet has been rejected by the link partner because of one or more detected errors,

Outstanding_packet_count = 0;

free_buffer_count = 0;

The port then transmits a link-request/input-status (for input-status) control symbol and waits for the link partner to respond with a link-response control symbol. When the link-response control symbol is received,

free_buffer_count = received_buf_status;

1.2.4.5 Speculative Packet Transmission

A port whose link partner is operating in transmitter-controlled flow control mode may send more packets than the number of free buffers indicated by the link partner. Packets transmitted in excess of the free_buffer_count are transmitted on a speculative basis and are subject to retry by the link partner. The link partner accepts or rejects these packets on a packet by packet basis in exactly the same way it would if operating in receiver-controlled flow control mode. A port may use such speculative transmission in an attempt to maximize the utilization of the link. However, speculative transmission that results in a significant number of retries and discarded packets can reduce the effective bandwidth of the link.

1.2.5 Flow Control Mode Negotiation

Immediately following the initialization of a link, each port begins sending idle control symbols to its link partner. The value of the buf_status field in these control symbols indicates to the link partner the flow control mode supported by the sending port.

The flow control mode negotiation rule is as follows:

If the port and its link partner both support transmitter-controlled flow control, then both ports shall use transmitter-controlled flow control. Otherwise, both ports shall use receiver-controlled flow control.

1.3 Error Detection and Recovery

Error detection and recovery is becoming a more important issue for many systems as operational frequencies increase and system electrical margins are reduced. The 8/16 LP-LVDS specification provides extensive error detection and recovery by combining retry protocols, cyclic redundancy codes, and single and multiple error detect capabilities, thereby tolerating all single-bit errors and many multiple bit errors. One goal of the error protection strategy is to keep the interconnect fabric from having to regenerate a CRC value as the packet moves through the fabric. All RapidIO ports require error checking.

1.3.1 Control Symbol Protection

The control symbols defined in this specification are protected in two ways:

- The S bit, distinguishing a control symbol from a packet header, has an odd parity bit to protect a control symbol from being interpreted as a packet.
- The entire aligned control symbol is protected by the bit-wise inversion of the control symbol used to align it to the 32-bit boundary described in Sectin 1.1.1. This allows extensive error detection.

A transmission error in the buf_status field, regardless of the control symbol type, may optionally not be treated as an error condition because it is always a reserved or an information only field that is not critical for proper system behavior. For example, if a corrupt value of buf_status is used, a low value may temporarily prevent a packet from being issued, or a high value may result in a packet being issued when it should not have been, causing a retry. In either case the problems are temporary and will properly resolve themselves through the existing protocol.

1.3.2 Packet Protection

The packets specified in the *RapidIO Common Transport Specification* and the *RapidIO Logical Specification* are protected with a CRC code that also covers the two bit priority field of this specification. The S bit is duplicated as in the control symbols to protect the packet from being interpreted as a control symbol, and the packet is also protected by protocol as described below.

Figure 1.7 shows the error coverage for the first 16 bits of a packet header. CRC protects the prio, tt, and ftype fields and two of the reserved bits as well as the remainder of the transport and logical fields. Since a new packet has an expected value for the ackID field at the receiver, bit errors on this field are easily detected and the packet is not accepted due to the unexpected value. An error on the S bit is detected with the redundant inverted S parity bit.

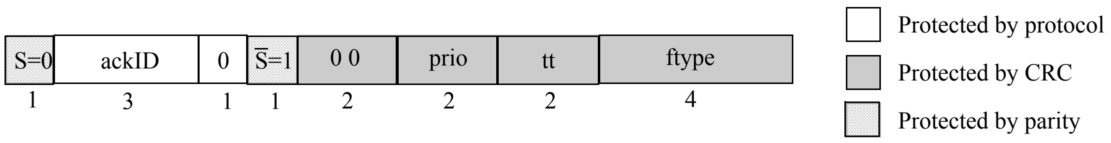


Figure 1-7. Error Coverage of First 16 Bits of Packet Header

This structure does not require that a packet's CRC value be regenerated when the uncovered physical fields are assigned in the fabric.

NOTE

All packets defined in the combination of this specification and the RapidIO interconnect logical and common transport specifications are now evenly divisible by 16 bits, or the complete packets are now naturally 16-bit aligned. This is illustrated in Figure 1.8. The leading 16 bits of the packet are referred to as the first symbol of the packet. The first symbol of a packet shall always land on the most significant half of the 32-bit boundary. Other aligned 16-bit packet quantities are also referred to as symbols.

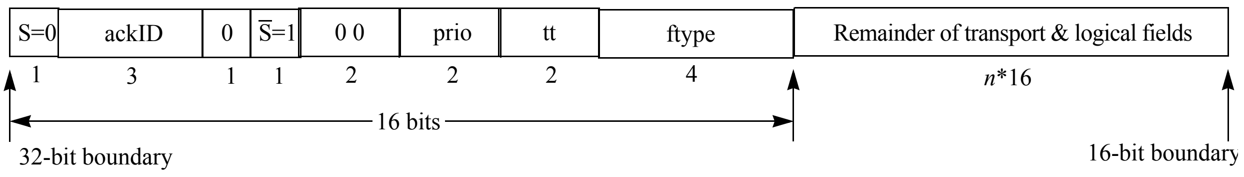


Figure 1-8. Naturally Aligned Packet Bit Stream

1.3.3 Lost Packet Detection

Some types of errors, such as a lost request or response packet or a lost acknowledgment, result in a system with hung resources. To detect this type of error there shall be time-out counters that expire when sufficient time has elapsed without receiving the expected response from the system. Because the expiration of one of these timers should indicate to the system that there is a problem, this time interval should be set long enough so that a false time-out is not signaled. The response to this error condition is implementation dependent.

The RapidIO specifications assume an implementation has time-out counters for the physical layer, the port link time-out counters, and counters for the logical layer, the port response time-out counters. The logical layer timers are discussed here in the physical layer specification because the packet delivery mechanism is an artifact of the physical layer. The values for these counters are specified in the physical layer registers in Chapter 4, "8/16 LP-LVDS Registers," on page 55. The interpretation of the values is implementation dependent, based on a number of factors including link clock rate, the internal clock rate of the device, and the desired system behavior.

The physical layer time-out occurs between the transmission of a packet and the receipt of an acknowledgment control symbol. This time-out interval is likely to be comparatively short because the packet and acknowledgment pair must only traverse a single link. For the purpose of error recovery, a port link time-out should be treated as an unexpected acknowledge control symbol.

The logical layer time-out occurs between the issuance of a request packet that requires a response packet and the receipt of that response packet. This time-out is counted from the time that the logical layer issues the packet to the physical layer to the time that the associated response packet is delivered from the physical layer to the logical layer. Should the physical layer fail to complete the delivery of the packet, the logical layer time-out will occur. This time-out interval is likely to be comparatively long because the packet and response pair have to traverse the fabric at least twice and be processed by the target. Error handling for a response time-out is implementation dependent.

Certain GSM operations may require two response transactions, and both must be received for the operation to be considered complete. In the case of a device implementation with multiple links, one response packet may be returned on the same link where the operation was initiated and the other response packet may be returned on a different link. If this is behavior is supported by the issuing processing element, the port response time-out implementation must look for both responses, regardless of which links they are returned on.

1.3.4 Implementation Note: Transactional Boundaries

A system address map usually contains memory boundaries that separate one type of memory space from another. Memory spaces are typically allocated with a preset minimum granularity. These spaces are often called page boundaries. Page boundaries allow the operating system to manage the entire address space through a standard mechanism. These boundaries are often used to mark the start and end of read-only space, peripheral register space, data space, and so forth.

RapidIO allows DMA streaming of data between two processing elements. Typically, in system interfaces that allow streaming, the targeted device of the transaction has a way to disconnect from the master once a transactional boundary has been crossed. The RapidIO specifications do not define a page boundary, nor a mechanism by which a target can disconnect part way through a transaction. Therefore, it is up to the system software and/or hardware implementation to guarantee that a transaction can complete gracefully to the address space requested.

As an example, a RapidIO write transaction does not necessarily have a size associated with it. Given a bus error condition whereby a packet delimiting control symbol is missed, the target hardware could continue writing data beyond the intended address space, thus possibly corrupting memory. Hardware implementations should set up page boundaries so this condition does not occur. In such an implementation, should a transaction cross the boundary, an error should be indicated and the transaction discarded.

1.3.5 Link Behavior Under Error

Transmission error detection is done at the input port, and all transmission error recovery is also initiated at the input port. Error detection can be done in a number of ways and at differing levels of complexity depending upon the requirements and implementation of a device.

1.3.5.1 Recoverable Errors

Four basic types of errors are detected by a port: an error on a packet, an error on a control symbol, an indeterminate error (an S bit parity failure), and a time-out waiting for a control symbol. A detailed state machine description of the behavior described in the sections below is included in Section A.2, "Error Recovery". The error recovery mechanism requires that a copy of each transmitted data packet be retained by the sending port so that the packet can be retransmitted if it is not accepted by the receiving port. The copy is retained until the sending port either receives a packet-accepted control symbol for the packet or determines that the packet has encountered an unrecoverable error condition.

When a sending port detects that the receiving port has not accepted a packet because one or more of the errors listed above has occurred (or the port has received a retry control symbol), the sending port resets the link time-out counters for the affected packet and all subsequently transmitted data packets. This prevents the generation of spurious time-out errors.

Any awaiting higher priority data packets are transmitted and all unaccepted data packets are retransmitted by the sending port. The number of times a data packet is retransmitted due to a recoverable error before the sending port declares an unrecoverable error condition exists is implementation dependent.

1.3.5.1.1 Packet Errors

Three types of packet errors exist: a packet with an unexpected ackID value, a corrupted packet indicated by a bad CRC value, and a packet that overruns some defined boundary such as the maximum data payload or a transactional boundary as described in Section 1.3.4. A processing element that detects a packet error immediately transitions into an “Input Error-stopped” state and silently discards all new packets until it receives a restart-from-error control symbol from the sender. The device also sends a packet-not-accepted control symbol with the received ackID value back to the sender. The sender then initiates recovery as described in Section 1.3.5.1.2 for unexpected control symbols.

1.3.5.1.2 Control Symbol Errors

The two types of control symbol errors are uncorrupted protocol violating control symbols, such as a packet-accepted, packet-retry, or packet-not-accepted control symbol which is either unsolicited or has an unexpected ackID value, or a corrupt control symbol. A corrupt control symbol is detected as a mismatch between the true and complement 16-bit halves of the aligned control symbol. A time-out on an acknowledge control symbol for a packet is treated like an acknowledge control symbol with an unexpected ackID value. An example of the first case, an uncorrupted protocol violating acknowledgment control symbol which has an unexpected ackID value, causes the receiving device to enter an “Output Error-stopped” state, immediately stop transmitting new packets, and issue a restart-from-error control symbol. The restart-from-error control symbol receives a response containing interesting receiver internal state, including the expected ackID. This expected ackID indicates to the sender where to begin re-transmission because the interface may have gotten out of sequence. The sender then backs up to the appropriate unaccepted packet and begins re-transmission.

For example, the sender transmits packets labeled ackID 2, 3, 4, and 5. It receives acknowledgments for packets 2, 4 and 5, indicating a probable error associated with ackID 3. The sender then stops transmitting new packets and sends a restart-from-error control symbol to the receiver. The receiver then returns a response control symbol indicating which packets it has received properly. These are the possible responses and the sender’s resulting behavior:

- expecting ackID = 3 - sender must re-transmit packets 3, 4, and 5
- expecting ackID = 4 - sender must re-transmit packets 4 and 5
- expecting ackID = 5 - sender must re-transmit packet 5
- expecting ackID = 6 - receiver got all packets, resume operation
- expecting ackID = anything else - fatal (non-recoverable) error

The second case, a corrupt control symbol, causes the receiver to enter the “Input Error-stopped” state and send a packet-not-accepted control symbol with an undefined ackID value to the sender. This informs the sending device that a transmission error has occurred and it will enter the recovery process described in the first control symbol error case described above.

1.3.5.1.3 Indeterminate Errors

An indeterminate error is an S bit parity error in which it is unclear whether the information being received is for a packet or a control symbol. These errors shall be handled as a corrupt con1trol symbols.

1.3.6 CRC Operation

A 16-bit CRC is selected as the method of error detection for the 8/16 LP-LVDS physical layer. This CRC is generated over all of a packet header, and all of the data payload except the first 6 bits of the added physical layer fields as shown in Figure 1-7. This checksum is appended to a packet in one of two ways. For a packet that has up to 80 bytes of header (including all logical, transport, and 8/16 LP-LVDS fields) and logical data payload, a single CRC value is appended to the packet. For packets with greater than 80 bytes of header and logical data payload, a CRC value is inserted after the first 80 bytes, aligning it to the first half of the 32-bit alignment boundary, and a second CRC value is appended at the end of the packet. The second CRC value is a continuation of the first and included in the running calculation, meaning that the running CRC value is not re-initialized after it is inserted after the first 80 bytes of the packet. This allows intervening devices to regard the embedded CRC value as 2 bytes of packet payload for CRC checking purposes.

NOTE
The embedded CRC value is itself used in the running CRC. As a result, from the CRC generator’s point of view the running CRC value is guaranteed to be all logic 0’s because the running CRC is XORed with itself. This fact may be useful in an implementation.

The early CRC value can be used by the receiving processing element to validate the header of a large packet and start processing the data before the entire packet has been received, freeing up resources earlier and reducing transaction completion latency. If the final appended CRC value does not cause the total packet to align to the 32-bit boundary, a 2 byte pad of all logic 0s is postpended to the packet. The pad of logic 0s allows the CRC check to always be done at the 32-bit boundary.

NOTE
While the embedded CRC value can be used by a processing element to start processing the data within a packet before receiving the entire packet, it is possible that upon reception of the end of the packet the final CRC value for the packet is incorrect. This would result in a processing element that has processed data that may have been corrupted. Outside of the error recovery mechanism described in Section 1.3.5, the RapidIO Interconnect Specification does not address the occurrence of such situations nor does it suggest a means by which a processing element would handle such situations. Instead, the mechanism for handling this situation is left to be addressed by the device manufacturers for devices

that implement the functionality of early processing of packet data.
Switch devices shall maintain the packet error coverage internally in order to preserve the integrity of the packets though the fabric. This will prevent undetected device internal errors such as SRAM bit errors from silently corrupting the system. The simplest method for preserving error coverage is to pass the CRC values through the switch as part of the packet. This works well for all non-maintenance packets whose CRC does not change as the packets are transported from source to destination thought the fabric. Maintaining error detection coverage is more complicated for maintenance packets as their hop_count and CRC change every time they pass through a switch.

Figure 1-9 is an example of a naturally 32-bit aligned packet of less than or equal to 80 bytes.

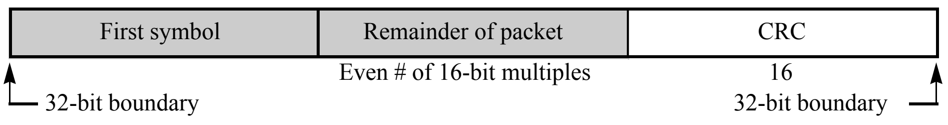


Figure 1-9. Naturally Aligned Packet Bit Stream Example 1

Figure 1-10 is an example of a naturally 32-bit aligned packet of greater than 80 bytes.

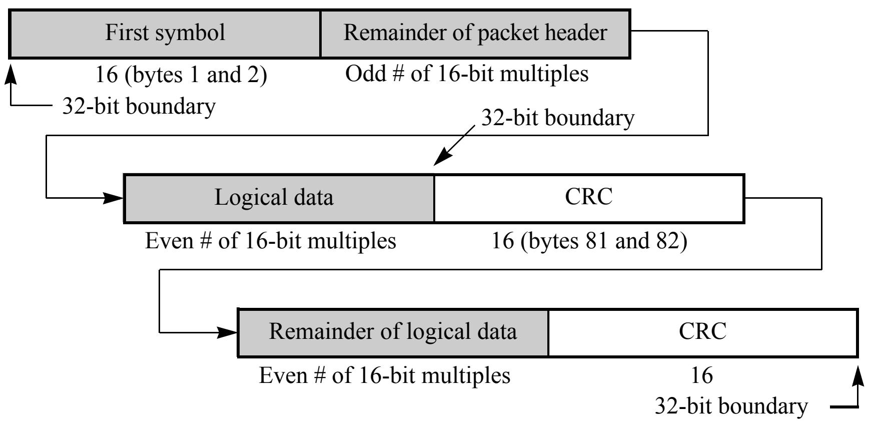

First symbol
Remainder of packet header
16 (bytes 1 and 2)
Odd # of 16-bit multiples
32-bit boundary
32-bit boundary
Logical data
CRC
Even # of 16-bit multiples
16 (bytes 81 and 82)
Remainder of logical data
CRC
Even # of 16-bit multiples
16
32-bit boundary

Figure 1-10. Naturally Aligned Packet Bit Stream Example 2

Figure 1-11 is an example of a padded 32-bit aligned packet of less than or equal to 80 bytes.

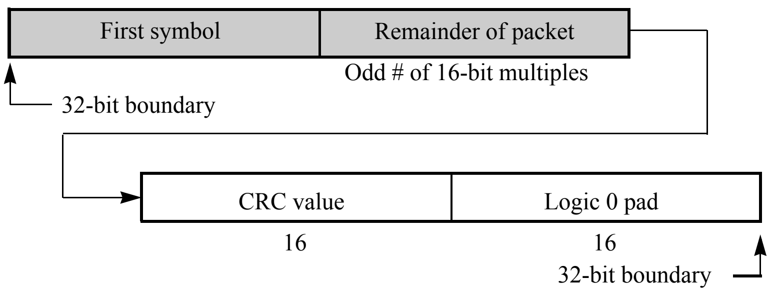


Figure 1-11. added Aligned Packet Bit Stream Example 1

Figure 1-12 is an example of a padded 32-bit aligned packet of greater than 80 bytes.

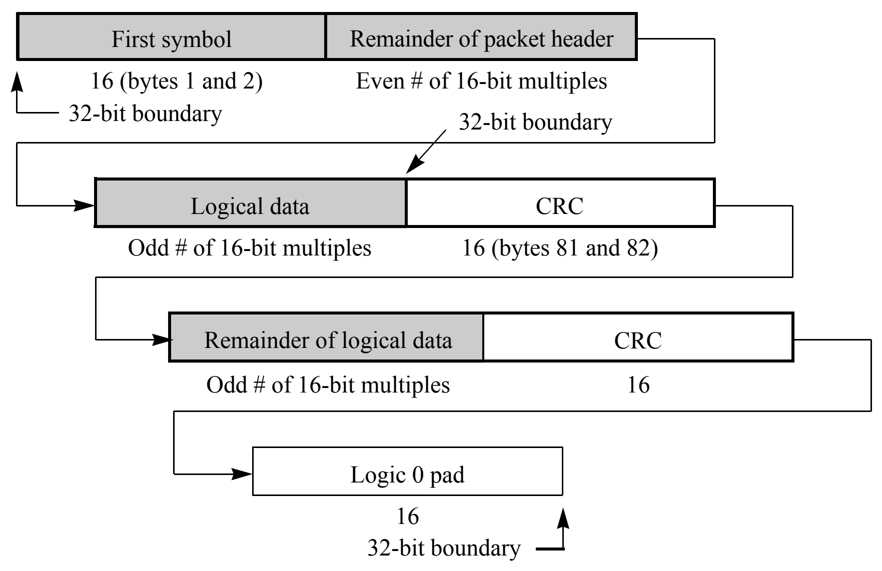


Figure 1-12. Padded Aligned Packet Bit Stream Example 2

1.3.7 CRC Code

The CCITT polynomial $X^{16}+X^{12}+X^{5}+1$ is a popular CRC code. The initial value of the CRC is 0xFFFF (all logic 1s). For the CRC calculation, the uncovered 6 bits are treated as logic 0s. As an example, a 16-bit wide parallel calculation is described in the equations in Table 1-4. Equivalent implementations of other widths can be employed.

Table 1-4. Parallel CRC Intermediate Value Equations

Check Bit	**e00**	**e01**	**e02**	**e03**	**e04**	**e05**	**e06**	**e07**	**e08**	**e09**	**e10**	**e11**	**e12**	**e13**	**e14**	**e15**
C00					x	x			x				x			
C01						x	x			x				x		
C02							x	x			x				x	
C03	x							x	x			x				x
C04	x	x			x	x				x						
C05		x	x			x	x				x					
C06	x		x	x			x	x				x				
C07	x	x		x	x			x	x				x			
C08	x	x	x		x	x			x	x				x		
C09		x	x	x		x	x			x	x				x	
C10			x	x	x		x	x			x	x				x
C11	x			x				x				x				
C12	x	x			x				x				x			
C13		x	x			x				x				x		
C14			x	x			x				x				x	
C15				x	x			x				x				x

where:

C00–C15 contents of the new check symbol

e00–e15 contents of the intermediate value symbol
e00 = d00 XOR c00
e01 = d01 XOR c01
through
e15 = d15 XOR c15

d00–d15 contents of the next 16 bits of the packet

c00–c15 contents of the previous check symbol

assuming the pipeline described in Figure 1-13.

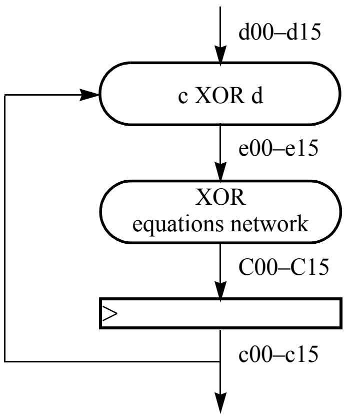


Figure 1-13. CRC Generation Pipeline

1.3.8 Maximum Packet Size

The maximum packet size permitted by the 8/16 LP-LVDS specification is 276 bytes. This includes all packet logical, transport, and physical layer header information, data payload, and required CRC bytes.

1.4 Link Maintenance Protocol

To initialize, explore, and recover from errors it is necessary to have a secondary mechanism to communicate between connected system devices. This mechanism is used to establish communications between connected devices (described in Section 2.6.1, “Link Initialization”), attempt automatic error recovery as described above in Section 1.3.5, “Link Behavior Under Error”, and allows software-managed link maintenance operations.

This protocol involves a request and response pair between electrically connected (linked) devices in the system. For software management, the request is generated through ports in the configuration space of the sending device. An external processing element write of a command to the link-request register with a *Partition I: Input/Output Logical Specification* maintenance write transaction causes an aligned link-request control symbol to be issued onto the output port of the device, but only one link-request can be outstanding on a link at a time. The device that is linked to the sending device shall respond with an aligned link-response control symbol if the link-request command required it to do so. The external processing element retrieves the link-response by polling the link-response register with I/O logical maintenance read transactions. A device with multiple RapidIO interfaces has a link-request and a link-response register pair for each corresponding RapidIO interface.

The automatic recovery mechanism relies on the hardware generating link-request control symbols under the transmission error conditions described in Section 1.3.5.1 and using the corresponding link-response information to attempt recovery.

Automatic link initialization also depends upon hardware generation of the appropriate link-requests and link-responses.

1.4.1 Command Descriptions

Table 1-5 contains a summary of the link maintenance commands that use the link maintenance protocol described above. Three link request commands are defined currently. The input-status command generates a paired link-response control symbol; the reset and send-training commands do not.

Table 1-5. Secondary Link Maintenance Command Summary

Command	Description
Reset	Resets the device
Input-status	Returns input port status; functions as a restart-from-error control symbol under error conditions. Generates a paired link-response control symbol.
Send-training	Stops normal operation and transmits 256 training pattern iterations

1.4.1.1 Reset and Safety Lockouts

The reset command causes the receiving device to go through its hard reset or power up sequence. All state machines and the configuration registers reset to the original power on states. The reset command does not generate a link-response control symbol.

Due to the undefined reliability of system designs it is necessary to put a safety lockout on the reset function of the link request control symbol. A device receiving a reset command in a link-request control symbol shall not perform the reset function unless it has received four reset commands in a row without any other intervening packets or control symbols, except idle control symbols. This will prevent spurious reset commands inadvertently resetting a device.

1.4.1.2 Input-status

The input-status command requests the receiving device to return the ackID value it expects to next receive from the sender on its input port and the current input port operational status for informational purposes. This command causes the receiver to flush its output port of all control symbols generated by packets received before the input-status command. The receiver then responds with a link-response control symbol.

The input-status command is the command used by the hardware to recover from transmission errors. If the input port had stopped due to a transmission error that generated a packet-not-accepted control symbol back to the sender, this input-status command acts as a restart-from-error control symbol, and the receiver is re-enabled to receive new packets after generating the link-response control symbol. This restart-from-error control symbol may also be used to restart the receiving device if it is waiting for a restart-from-retry control symbol after retrying a packet. This situation can occur if transmission errors are encountered while trying to re-synchronize the sending and receiving devices after the retry.

1.4.1.3 Send-training

The send-training command causes the recipient device to suspend normal operation and begin transmitting a special training pattern. The receiving device transmits a total of 256 iterations of the training pattern followed by at least one idle control symbol and then resumes operation. The usage of this command is described in Section 2.6.1.1, “Sampling Window Alignment.” The send-training command does not generate a link-response control symbol.

1.4.2 Status Descriptions

The input-status request generates two pieces of information that are returned in the link-response:

- link status
- ackID usage

The first type of data is the current operational status of the interface. These status indicators are described in Table 1-6.

Table 1-6. Link Status Indicators

Status Indicator	Description
OK	The port is working properly.
Error	The port has encountered an unrecoverable error and has shut down.
Retry-stopped[1]	The port has been stopped due to a retry.
Error-stopped[1]	The port has been stopped due to a transmission error.

1. Valid only with the Stopped indicator

The retry-stopped state indicates that the port has retried a packet and is waiting to be restarted. This state is cleared when a restart-from-retry (or a link-request/input-status) control symbol is received. The error-stopped state indicates that the port has encountered a transmission error and is waiting to be restarted. This state is cleared when a link-request/input-status control symbol is received.

The second field returned in the link-response control symbol is state information about the acknowledge identifier usage. The input port returns a value indicating the next ackID expected to be received by the port. The automatic error recovery mechanism uses this information to determine where to begin packet re-transmission after a transmission error condition has been encountered.

2 Chapter 2 - Packet and Control Symbol Transmission

This RapidIO chapter defines packet and control symbol delineation and alignment on the physical port and mechanisms to control the pacing of a packet. Each input and output port is either one or two bytes wide. All 8/16 LP-LVDS defined protocols are used identically for both the 8- and 16-bit wide versions of the physical interface. The only difference is the number of pins used to transmit the packets and aligned control symbols.

2.1 Packet Start and Control Symbol Delineation

The control framing signal used to delineate the start of a packet or a control symbol on the physical port is a no-return-to-zero, or NRZ signal. This frame signal is toggled for the first symbol (see the Note in Section 1.3.2, “Packet Protection”) of each packet and for the first control symbol of each aligned control symbol. Therefore, if a 16-bit symbol contains a RapidIO logical packet format type (the ftype field in the RapidIO logical specifications) or a control symbol (ttype) field, the frame signal shall toggle. In order for the receiving processing element to sample the data and frame signals, a data reference signal is supplied that toggles on all possible transitions of the interface pins. This type of data reference signal is also known as a double-data-rate clock. These received clocks on devices with multiple RapidIO ports have no required frequency or phase relationship.

The framing signal is not toggled for other symbols such as those containing remaining packet header and data bytes. However, it is toggled for all idle control symbols between packets. This means that the maximum toggle rate of the control framing signal is every 4 bytes, and the framing signal is only allowed to toggle on every fourth byte. Therefore, the framing signal is aligned to a 32-bit boundary as are all of the packets and aligned control symbols. Additionally, the data reference signal shall transition from low to high on this same boundary. Examples of these constraints are shown in Figure 2-1 and Figure 2-3 for an 8-bit port and Figure 2-2 and Figure 2-4 for a 16-bit port.

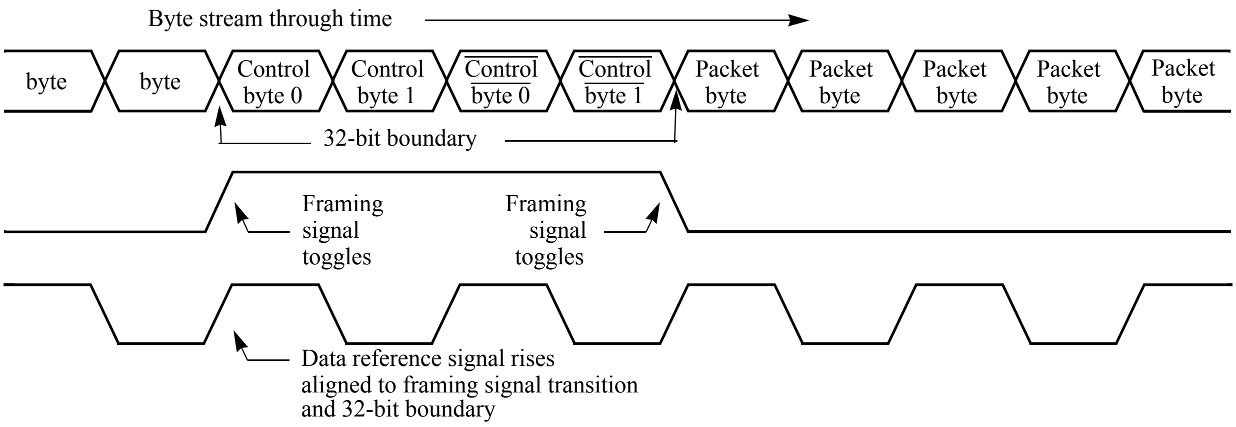


Figure 2-1. Framing Signal Maximum Toggle Rate for 8-bit Port

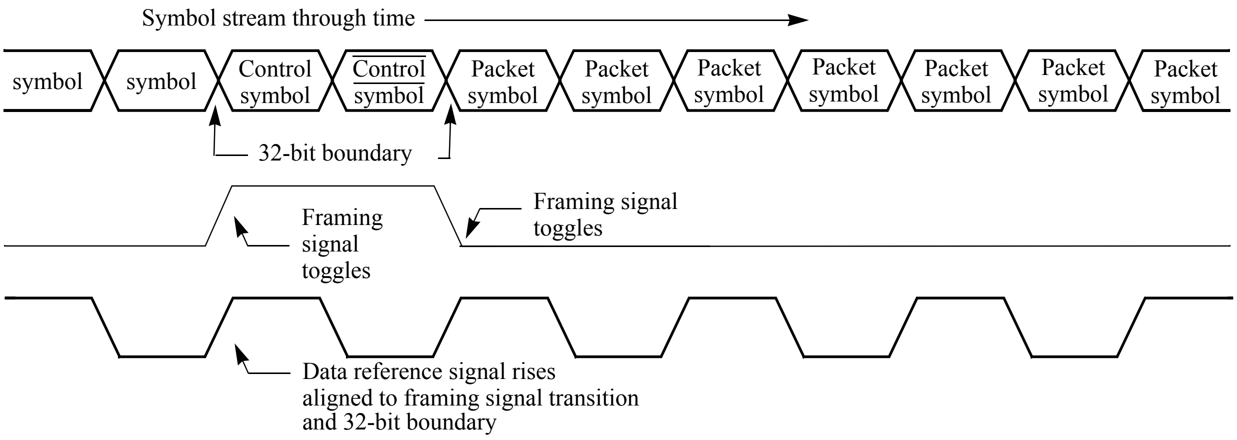


Figure 2-2. Framing Signal Maximum Toggle Rate for 16-bit Port

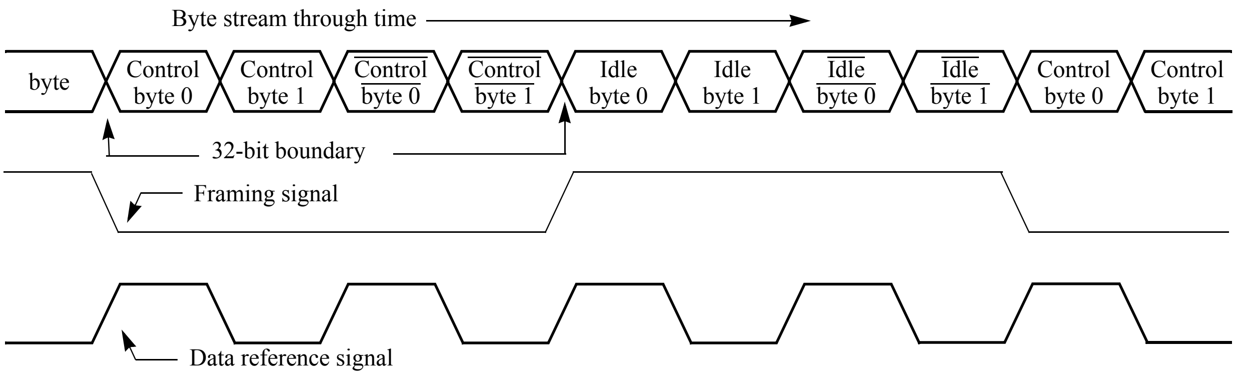


Figure 2-3. Control Symbol Delineation Example for 8-bit Port

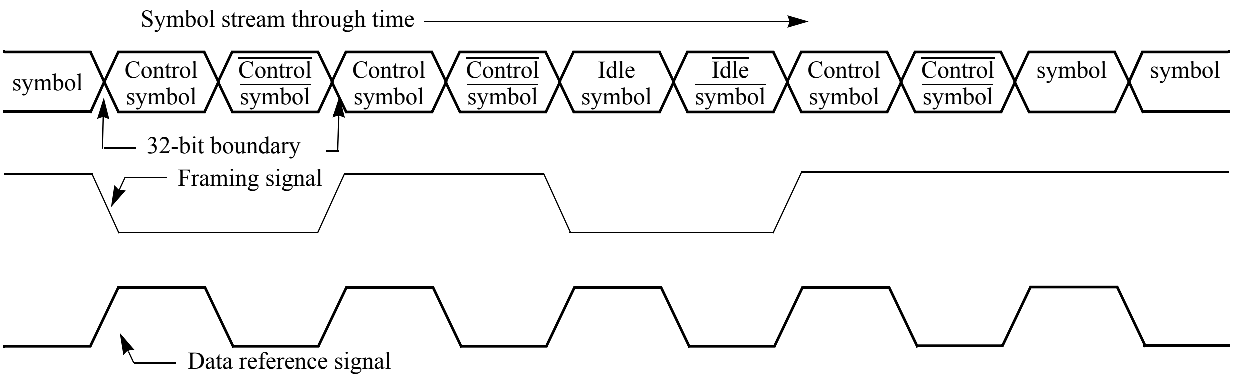


Figure 2-4. Control Symbol Delineation Example for 16-bit Port

Errors on the framing and data reference signals can be detected either directly by verifying that the signals transition only when they are allowed and expected to transition, or indirectly by depending upon detection of packet header or CRC or control symbol corruption, etc. if these signals behave improperly. Either method of error detection on the framing and data reference signals allows error recovery by following the mechanisms described in Section 1.3.5.1, "Recoverable Errors" and Section A.3, "Error Recovery."

For simplicity, the data reference signal will not be included in any additional figures in this document. It is always rising on the 32-bit boundary when it is legal for the frame signal to toggle as shown in Figure 2-1 through Figure 2-4.

2.2 Packet Termination

A packet is terminated in one of two ways:

- The beginning of a new packet marks the end of a previous packet.
- The end of a packet may be marked with one of the following: an aligned end-of-packet (eop), restart-from-retry, link-request, or stomp control symbol.

The stomp control symbol is used if a transmitting processing element detects a problem with the transmission of a packet. It may choose to cancel the packet by sending the stomp control symbol instead of terminating it in a different, possibly system fatal, fashion like corrupting the CRC value.

The restart-from-retry control symbol can cancel the current packet as well as be transmitted on an idle link. This control symbol is used to enable the receiver to start accepting packets after the receiver has retried a packet.

The link-request control symbol can cancel the current packet as well as be transmitted on an idle link and has several applications. It can be used by software for system observation and maintenance, and it can be used by software or hardware to enable the receiver to start accepting packets after the receiver has refused a packet due to a transmission error as described in Section 1.3, "Error Detection and Recovery."

A receiver shall drop a canceled packet without generating any errors and shall then respond with a packet-retry acknowledgment control symbol unless an acknowledgment has already been sent for that packet or the receiver is stopped due to an earlier retry or error. If the receiver is not already stopped it shall follow the packet retry mechanism if the packet was canceled with a control symbol other than a restart-from-retry or a link-request/input-status control symbol.

Figure 2-5 is an example of a new packet marking the end of a packet.

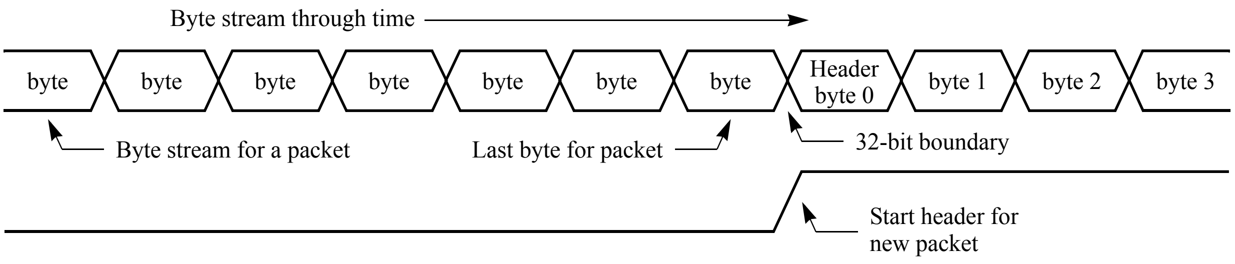


Figure 2-5. Header Marked End of Packet (8-bit Port)

Figure 2-6 is an example of an aligned end-of-packet control symbol marking the end of a packet. The stomp, link-request, and restart-from-retry control symbol cases look similar.

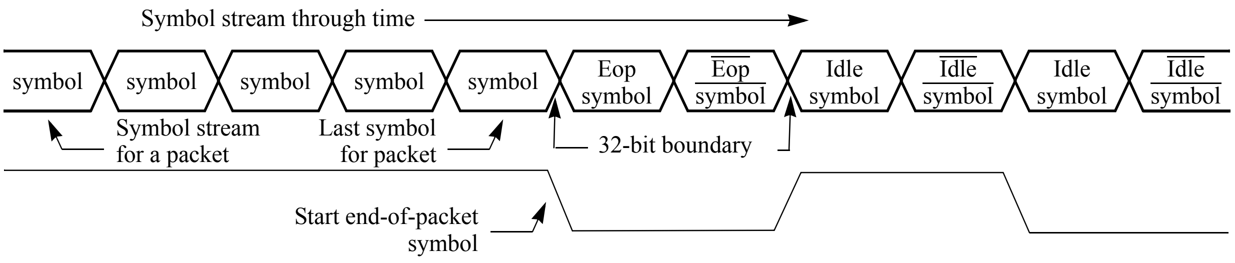

Symbol stream through time
symbol
symbol
symbol
symbol
symbol
Eop symbol
$\overline{\text{Eop}}$ symbol
Idle symbol
$\overline{\text{Idle}}$ symbol
Idle symbol
$\overline{\text{Idle}}$ symbol
Symbol stream for a packet
Last symbol for packet
32-bit boundary
Start end-of-packet symbol

Figure 2-6. End-Of-Packet Control Symbol Marked End of Packet (16-bit Port)

2.3 Packet Pacing

If a device cannot transmit a packet as a contiguous stream of control symbols, it may force wait states by inserting idle control symbols called pacing idles. As with the other control symbols, the pacing idle control symbols are always followed by a bit-wise inverted copy and are then called aligned pacing idle control symbols. Any number of aligned pacing idle control symbols can be inserted, up to some implementation defined limit, at which point the sender should instead send a stomp control symbol and cancel the packet in order to attempt to transmit a different packet. figure 2-7 shows an example of packet pacing. These idle control symbols are ignored by the receiving device, and more data is sent when it becomes available. Pacing idle control symbols can be embedded anywhere in a packet where they can be legally delineated.

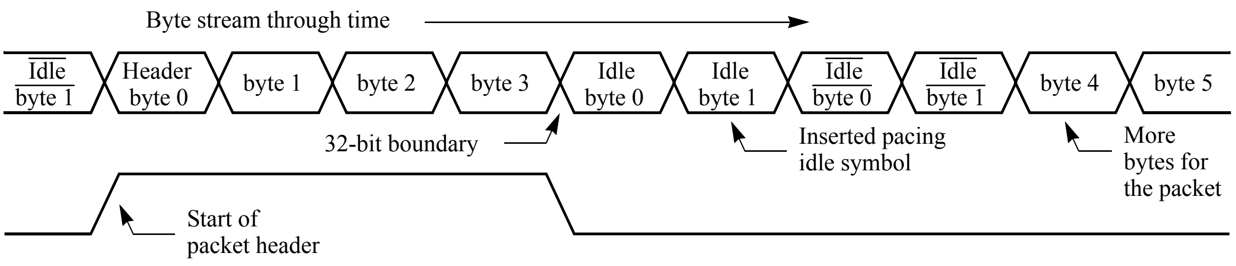

Byte stream through time
$\overline{\text{Idle}}$ byte 1
Header byte 0
byte 1
byte 2
byte 3
Idle byte 0
Idle byte 1
$\overline{\text{Idle}}$ byte 0
$\overline{\text{Idle}}$ byte 1
byte 4
byte 5
32-bit boundary
Inserted pacing idle symbol
More bytes for the packet
Start of packet header

Figure 2-7. Pacing Idle Insertion in Packet (8-bit Port)

The receiver of a packet may request that the sender insert pacing idle control symbols on its behalf by sending a throttle control symbol specifying the number of aligned pacing idle control symbols to delay. The packet sender then inserts that number of aligned pacing idles into the packet stream. If additional delay is needed, the receiver can send another throttle control symbol.

If the receiver requests too many aligned pacing idles indicating an excessive delay, determined by some implementation defined limit, it should terminate the packet transmission by issuing a packet-retry acknowledge control symbol. Alternatively, the sender may issue a stomp control symbol to cancel the packet if too many aligned pacing idle control symbols are requested by the receiver. The throttle control symbol shall be honored because it is used to force insertion of idle control symbols for clock re-synchronization in the receiver as described in Chapter 5, “System Clocking Considerations.”

The maximum allowed response time from the receipt of the last byte of an aligned throttle control symbol at the input pins to the appearance of the first byte of an aligned pacing idle control symbol on the output pins is 40 interface clocks (80 data ticks).

Note that for CRC values for a packet, the aligned pacing idle control symbols are not included in the calculation.

2.4 Embedded Control Symbols

Control symbols can be embedded anywhere in a packet in the same fashion as pacing idle control symbols, as long as all delineation and alignment rules are followed.

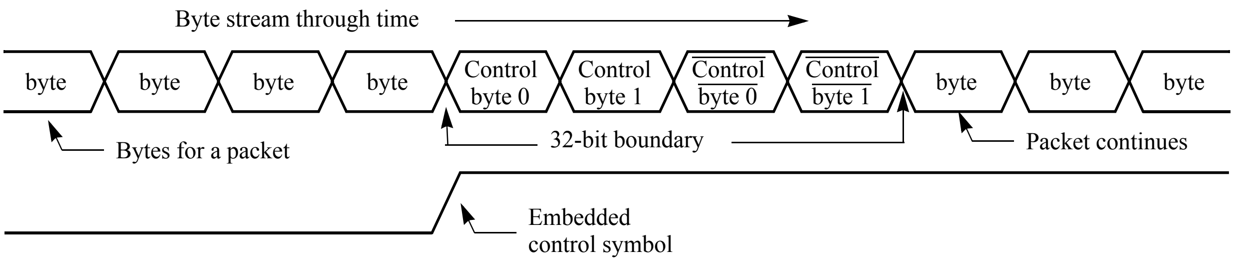


Figure 2-8. Embedded Control Symbols for 8-bit Port

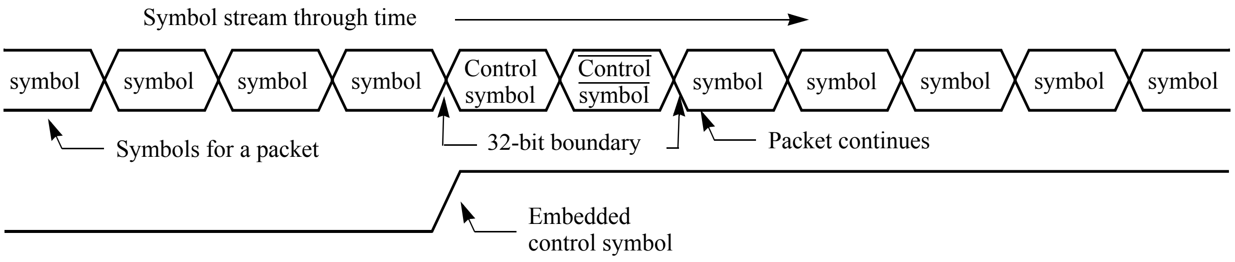


Figure 2-9. Embedded Control Symbols for 16-bit Port

As with the pacing idle control symbols, the embedded aligned control symbols are not included in the CRC value calculation for the packet.

A special error case exists when a corrupt embedded control symbol is detected. In this case a packet-not-accepted control symbol shall be generated and the embedding packet is discarded.

2.5 Packet to Port Alignment

This section shows examples of packet transmission over the 8-bit and 16-bit interfaces. The corresponding control symbol alignment is shown in Section 3.6, “Control Symbol to Port Alignment.”

Figure 2-10 shows the byte transmission ordering on a port through time using a small transport format ftype 2 packet from the *RapidIO Input/Output Logical Specification* and *RapidIO Common Transport Specification*. Note that for this example the two bytes following the CRC would indicate some form of packet termination such as a new packet or an eop.

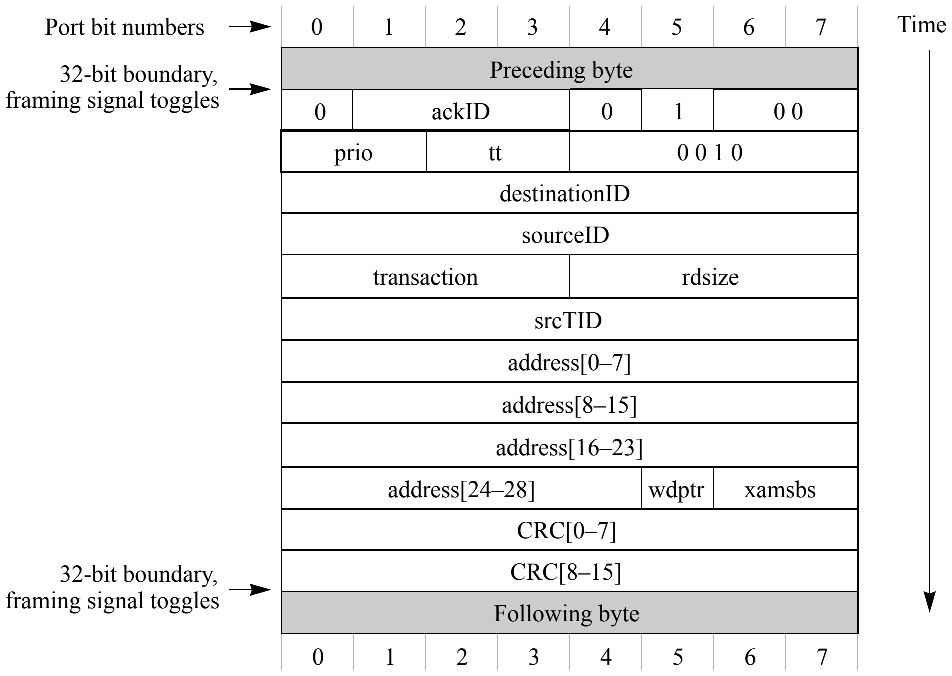


Figure 2-10. Request Packet Transmission Example 1

Figure 2-11 shows the same packet transmitted over a 16-bit port.

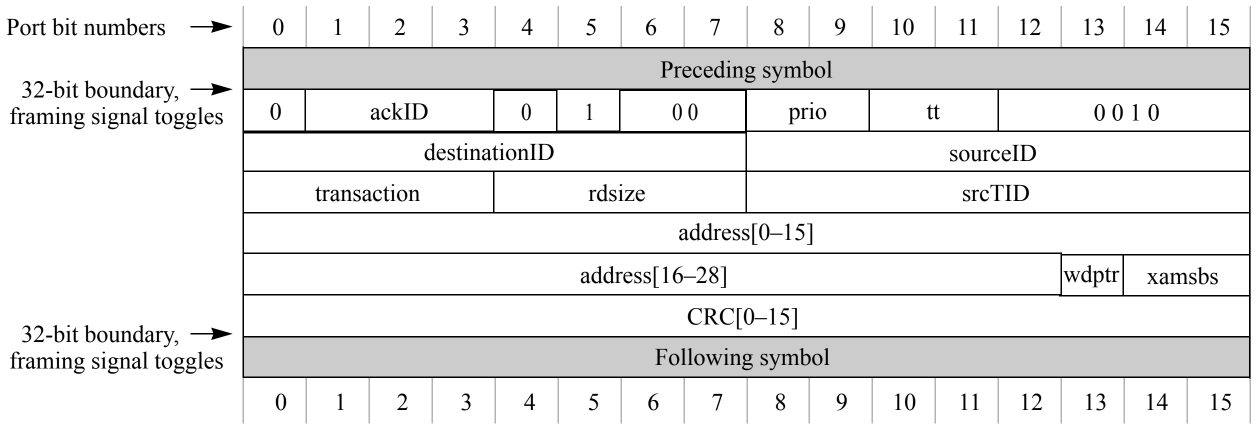


Figure 2-11. Request Packet Transmission Example 2

Figure 2-12 shows the same example again but with the large transport format over the 8-bit port. Note that for this example the two bytes following the CRC of the packet are all logic 0 pads.

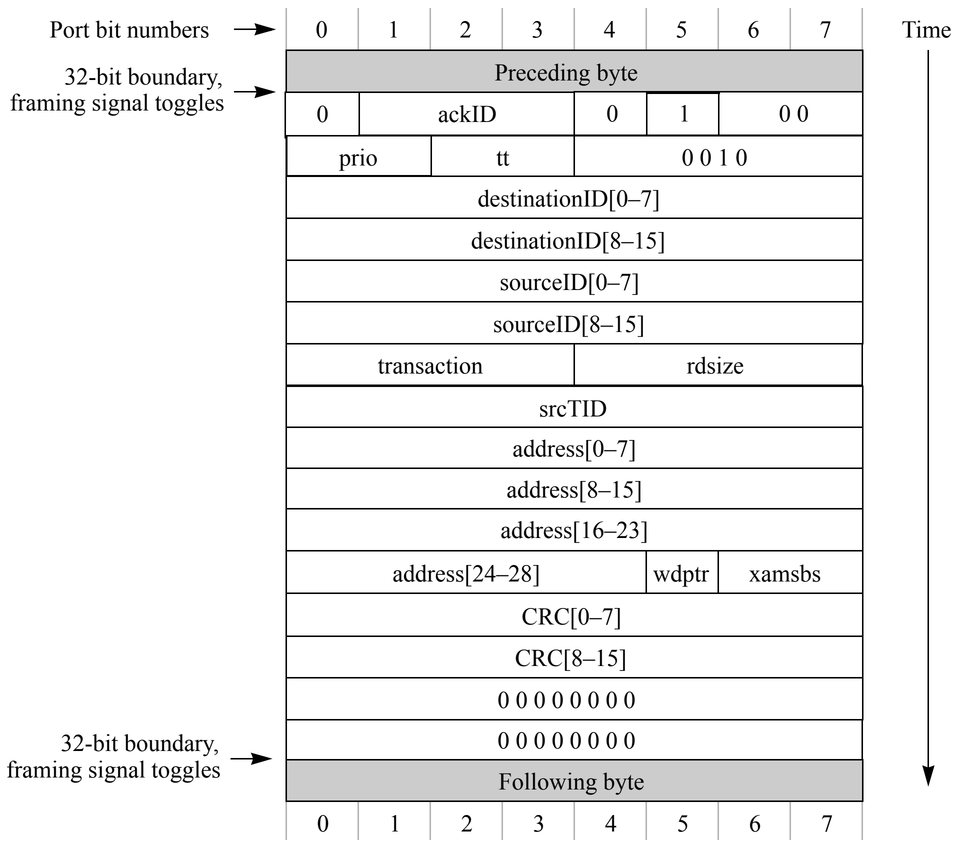


Figure 2-12. Request Packet Transmission Example 3

Figure 2-13 is the same packet as for Figure but over the 16-bit port.

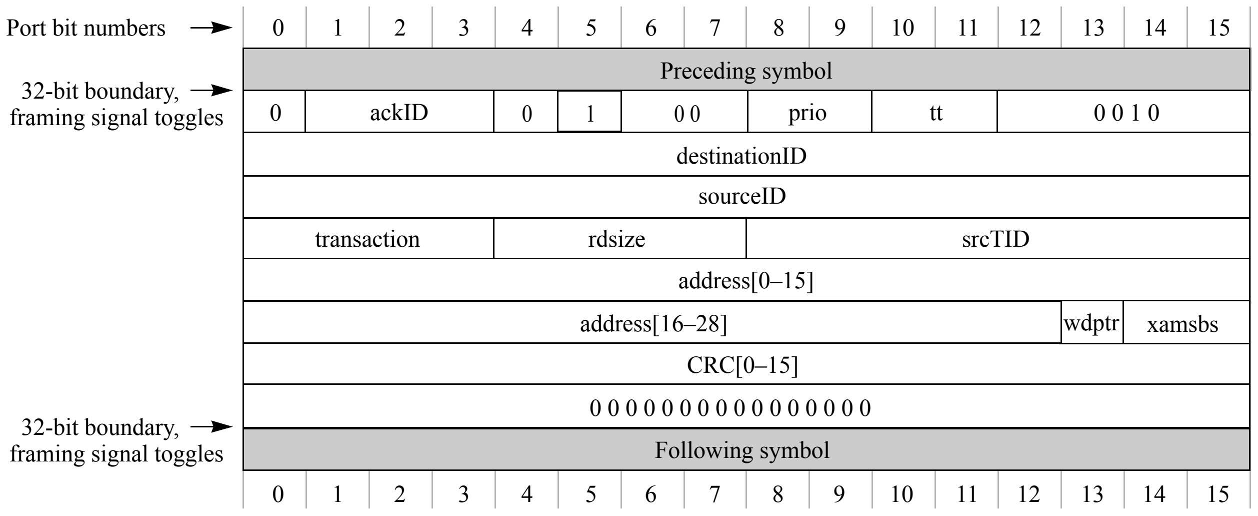


Figure 2-13. Request Packet Transmission Example 4

Figure 2-14 and Figure 2-15 show the ftype 13 response packet for request example—the small transport format packet.

Note that the two bytes following the packet CRC may be logic 0 pads depending on the size of the packet.

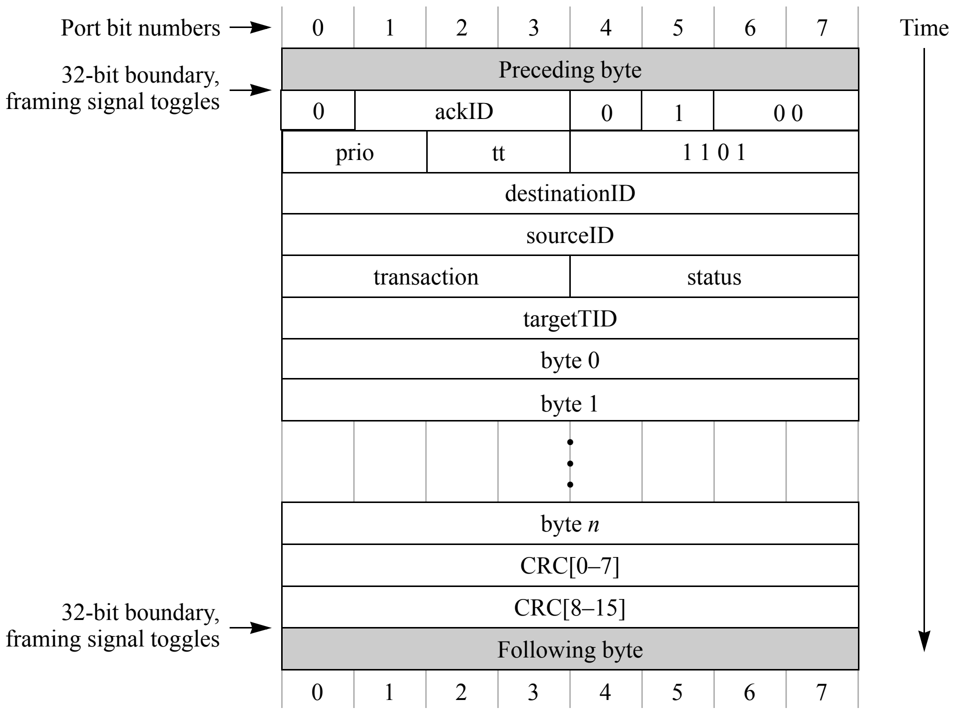


Figure 2-14. Response Packet Transmission Example 1

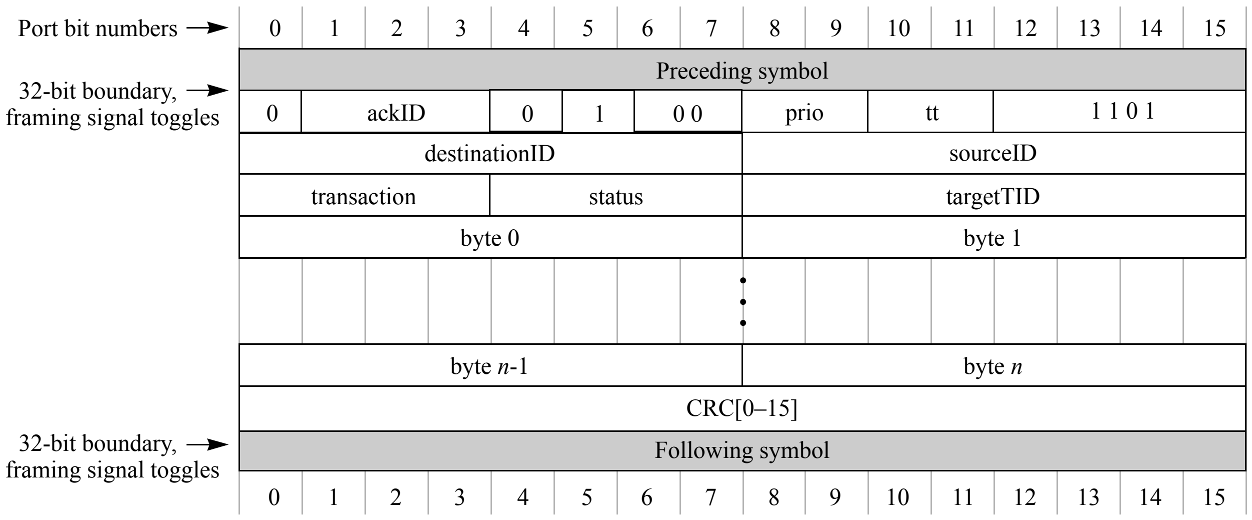


Figure 2-15. Response Packet Transmission Example 2

2.6 System Maintenance

A necessary part of any system are methods for initializing, configuring, and maintaining the system during operation.

2.6.1 Link Initialization

Because the RapidIO 8/16 LP-LVDS interface is source synchronous, it is necessary to initialize the input ports so that packets and control symbols can be accurately received.

There are two procedures needed for initializing an 8/16 LP-LVDS input port:

- Aligning the sampling window of the input clock and data signals for reliable sampling of incoming data
- Aligning the input to the 32-bit boundary for proper packet and control symbol extraction

These two procedures can be done in parallel by the receiver.

2.6.1.1 Sampling Window Alignment

Depending upon the device implementation, the data sampling window of the receiver may need to be adjusted to accomplish reliable data sampling. Adjusting the sampling window for an input port requires that a special pre-defined signal pattern, or training pattern, be applied on the input pins during initialization. Such a training pattern allows the receiver to align the input signals to properly sample the data and frame signals. The 8/16 LP-LVDS training pattern is defined in Section 3.5, “Training Pattern Format.” It is aligned to the 32-bit boundary, and it is easily distinguishable from control symbols and packet headers.

The initialization procedure described here applies for system power-up and normal operation, such as system reset and error recovery. Sampling window alignment is needed for a device when it is reset or when it has lost previously established alignment due to events such as excessive system noise or power fluctuations. The reception of an unsolicited training pattern by a port is a link protocol violation. It causes the port to enter the “Output Error-stopped” state and indicates that the attached port has lost input data sampling window alignment and has most likely lost some previously sent packets and control symbols. The port shall execute the “Output Error-stopped” recovery sequence specified in Section 1.3.5.1.2 after communication with the attached port is re-established. Link initialization for other scenarios (such as hot swap) are not specifically addressed. The actual method implemented by a device to adjust its internal sampling window is beyond the scope of this specification.

Following are the events of an alignment sequence coming out of reset or upon losing synchronization:

- The port begins the alignment sequence by transmitting the link-request/send-training control symbol followed by transmitting the training pattern 256 times to the attached port.
- At the same time, the port tries to detect and align its input sampling window to the training pattern that is (or will eventually be) sent from the attached port. If the port has completed transmitting the 256 iterations of the training pattern but has not yet successfully adjusted its input sampling window, it again sends a link-request/send-training control symbol and another 256 iterations of the training pattern, and continues trying to align its input sampling window to the pattern coming from the attached port.
- Eventually, if the attached port is operating, the port will finish adjusting its input sampling window to the training pattern coming from the attached port. At this point, the port shall send one idle control symbol (instead of sending a link-request/send-training control symbol) between sending 256 iterations of its training pattern.
- The port shall continue to send one idle control symbol after sending 256 iterations of its training pattern until it has received an idle control symbol.

The port signals that it no longer requires the training pattern by replacing the transmission of link-request/send-training control symbol with one idle control symbol. The link is regarded as operational when the port is transmitting one idle control symbol between sending the 256 iterations of the training pattern and is successfully receiving one idle control symbol in between the 256 iterations of the training pattern, and the port can transition into normal operation (“OK”) mode.

A port that does not require sampling window adjustment does not follow this sequence out of reset and instead shall begin to transmit idle control symbols immediately upon leaving reset. If a port that does not require sampling window adjustment is connected to a port that does require adjustment, then the port that requires training shall begin the alignment sequence by transmitting a link-request/send-training control symbol followed by 256 iterations of the training pattern to indicate that the alignment sequence is required. The port that does not require training shall respond with 256 iterations of the training pattern followed by one idle control symbol and continue the alignment sequence until it has received one idle control symbol.

Periodically a port may need to adjust its sampling window to maintain proper window alignment. In such a case, a port shall issue a link-request/send-training control symbol to indicate to the attached port that the port requires maintenance training. Since this is a request for maintenance training, the link-request/send-training control symbol is followed by normal link traffic, not by the training pattern. When maintenance training is requested, the output of the port receiving the request shall end the transmission of packets and control symbols as quickly as possible without violating the link protocol and then transmit 256 iterations of the training pattern followed by at least one idle control symbol and resume normal operation.

Because an 8-bit wide port can be connected to a 16-bit wide port, the training pattern is also used to detect the usable width of the 16-bit interface. If the training pattern is discovered on data bits 0-7 of a 16-bit interface and not on data bits 8-15, it is assumed that the connected port is an 8-bit port. The operation of the corresponding 16-bit output port shall then degrade to 8-bit mode, with the port treating bits 8-15 of its output port as a replication of bits 0-7. Operation of an 8-bit interface connected to the data bits 8-15 of a 16-bit interface is undefined.

The example state machine in Section A.1, “Link Initialization and Maintenance Mechanism” shows how the required behavior may be implemented.

2.6.1.2 32-Bit Boundary Alignment

The input port shall be aligned to the 32-bit boundary of the connected output port. To accomplish this alignment, all control symbols are delineated on the 32-bit boundary, thereby providing a steady stream of properly aligned frame signal transitions.

2.6.2 Multicast-Event

The Multicast-Event control symbol provides a mechanism through which notice that some system defined event has occurred, can be selectively multicast throughout the system. Refer to Section 3.2 for the format of the multicast-event control symbol.

When a switch processing element receives a Multicast-Event control symbol, the switch shall forward the Multicast-Event by issuing a Multicast-Event control symbol from each port that is designated in the port's CSR as a Multicast-Event output port. A switch port shall never forward a Multicast-Event control symbol back to the device from which it received a Multicast-Event control symbol regardless of whether the port is designated a Multicast-Event output or not.

It is intended that at any given time, Multicast-Event control symbols will be sourced by a single device. However, the source device can change (in case of failover, for example). In the event that two or more Multicast-Event control symbols are received by a switch processing element close enough in time that more than one is present in the switch at the same time, at least one of the Multicast-Event control symbols shall be forwarded. The others may be forwarded or discarded (device dependent).

The system defined event whose occurrence Multicast-Event gives notice of has no required temporal characteristics. It may occur randomly, periodically, or anything in between. For instance, Multicast-Event may be used for a heartbeat function or for a clock synchronization function in a multiprocessor system.

In an application such as clock synchronization in a multiprocessor system, both the propagation time of the notification through the system and the variation in propagation time from Multicast-Event to Multicast-Event are of concern. For these reasons and the need to multicast, control symbols are used to convey Multicast-Events as control symbols have the highest priority for transmission on a link and can be embedded in packets.

While this specification places no limits on Multicast-Event forwarding delay or forwarding delay variation, switch functions should be designed to minimize these characteristics. In addition, switch functions shall include in their specifications the maximum value of Multicast-Event forwarding delay (the maximum value of Multicast-Event forwarding delay through the switch) and the maximum value of Multicast-Event forwarding delay variation (the maximum value of Multicast-Event forwarding delay through the switch minus the minimum value of Multicast-Event forwarding delay through the switch).

2.7 Power Management

Power management is currently beyond the scope of this specification and is implementation dependent. A device that supports power management features can make these features accessible to the rest of the system in the device’s local configuration registers.

3 Chapter 3 - Control Symbol Formats

This chapter defines the RapidIO physical layer control symbols described in Chapter 1, “Physical Layer Protocol.” Note that the S bit defined in Section 1.2.1 is always set to logic 1 and the $\overline{S}$ bit (also defined in Section 1.2.1) is always set to logic 0 for the physical layer control symbols. All control symbols are aligned to 32 bits with the last 16 bits as a bit-wise inverse of the first 16. A device receiving an undefined control symbol shall treat the control symbol as an idle control symbol for forward compatibility.

3.1 Acknowledgment Control Symbol Formats

An acknowledgment control symbol is a transmission status indicator issued by a processing element when it has received a packet from another processing element to which it is electrically connected. Acknowledgment control symbols are used for flow control and resource de-allocation between adjacent devices. The following are the different acknowledgment control symbols that can be transmitted back to sending elements from receiving elements:

- Packet-accepted
- Packet-retry
- Packet-not-accepted

Because receipt of an acknowledgment control symbol does not imply the end of a packet, a control symbol can be embedded in a packet, as well as sent when the interconnect is idle. Embedded control symbols are discussed in Section 2.4, “Embedded Control Symbols.”

Field definitions for the acknowledgment control symbols are shown in Table 3-1.

Table 3-1. Field Definitions for Acknowledgment Control Symbols

Field	Definition
packet_ackID	Acknowledgment ID is the packet identifier for acknowledgments back to the request or response packet sender.
buf_status	buf_status field indicates the number of maximally sized packets that can be received, described in Section 1.2.1
cause	cause field indicates the type of error encountered by an input port, defined in Table 3-2

3.1.1 Packet-Accepted Control Symbol

The packet-accepted acknowledgment control symbol indicates that the adjacent device in the interconnect fabric has taken responsibility for sending the packet to its final destination and that resources allocated by the sending device can be released. This control symbol shall be generated only after the entire packet has been received and found to be free of detectable errors. This control symbol format is displayed in Figure 3-1.

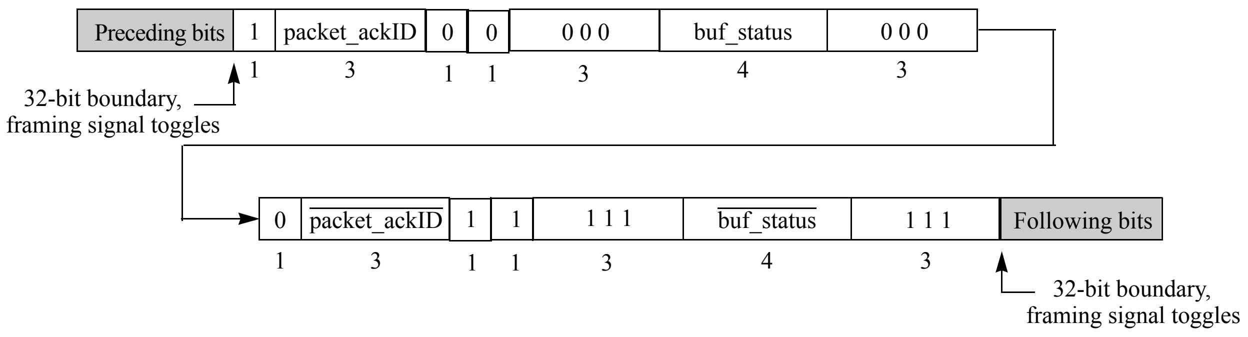


Figure 3-1. Type 0 Packet-Accepted Control Symbol Format

3.1.2 Packet-Retry Control Symbol

A packet-retry acknowledgment control symbol indicates that the adjacent device in the interconnect fabric was not able to accept the packet due to some temporary resource conflict such as insufficient buffering and the source should retransmit the packet. This control symbol can be generated at any time after the start of a packet, which allows the sender to cancel the packet and try sending a packet with a different priority or destination. This will avoid wasting bandwidth by transmitting all of the rejected packet. This control symbol format is displayed in Figure 3-2.

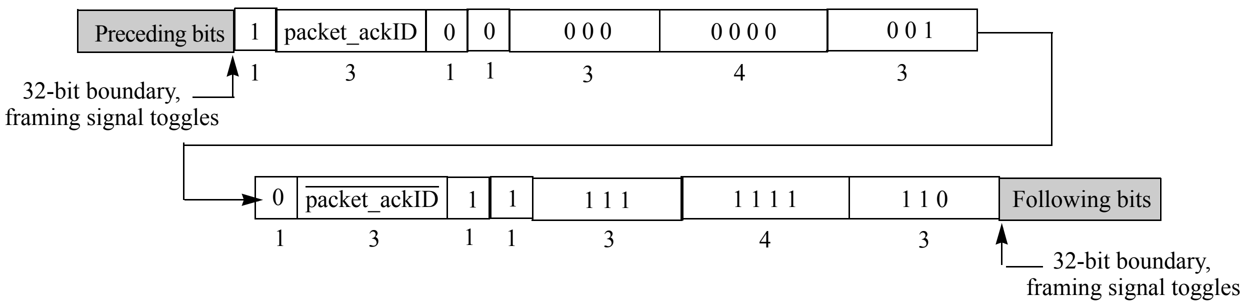


Figure 3-2. Type 1 Packet-Retry Control Symbol Format

3.1.3 Packet-Not-Accepted Control Symbol

A packet-not-accepted acknowledgment control symbol means that the receiving device could not accept the packet due to an error condition, and that the source should retransmit the packet. This control symbol can be generated at any time after the start of a packet, which allows the sender to cancel the packet and try sending a packet with a different priority or destination. Generating this control symbol at any point in packet transmission avoids wasting bandwidth by transmitting all of the rejected packet. The packet-not-accepted control symbol contains a field describing the cause of the error condition, shown in If the receiving device is not able to specify the cause for some reason, or the cause is not one of defined options, the general error encoding shall be used. This control symbol format is displayed in figure 3-3.

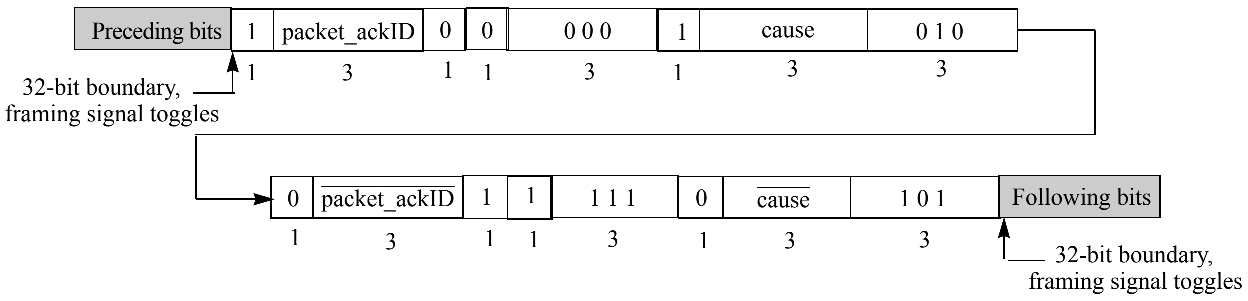


Figure 3-3. Type 2 Packet-Not-Accepted Control Symbol Format

The cause field shall be used to display informational fields useful for debug. Table 3-2 displays the reasons a packet may not be accepted, indicated by the cause field.

Table 3-2. Cause Field Definition

Encoding	Definition
0b000	Encountered internal error
0b001	Received unexpected ackID on packet
0b010	Received error on control symbol
0b011	Non-maintenance packet reception is stopped
0b100	Received bad CRC on packet
0b101	Received S bit parity error on packet/control symbol
0b110	Reserved
0b111	General error

3.1.4 Canceling Packets

A packet-retry or packet-not-accepted acknowledgment control symbol that is received for a packet that is still being transmitted may result with the sender canceling the packet.

The sending device can use the stomp (see Chapter 2, “Packet and Control Symbol Transmission”), restart-from-retry (in response to a packet-retry control symbol), or link-request (in response to a packet-not-accepted control symbol) control symbol to cancel the packet. Because the receiver has already rejected the packet, it will not detect any induced error. Alternatively, the sending device can choose to complete transmission of the packet normally.

3.2 Packet Control Symbol Formats

Packet control symbols are used for packet delineation, transmission, pacing, and other link interface control functions as described in Chapter 2, “Packet and Control Symbol Transmission.”

The packet control symbols are the throttle, stomp, restart-from-retry control symbols, idle, end-of-packet (eop), and multicast-event control symbols, which are specified in the sub_type field of the type 4 control symbol format. The packet control symbols also have a contents field, which has a different meaning depending upon the particular control symbol. Of these control symbols, all control symbols that are not defined as terminating a packet may be embedded within a packet.

This control symbol format is displayed in Figure 3-4.

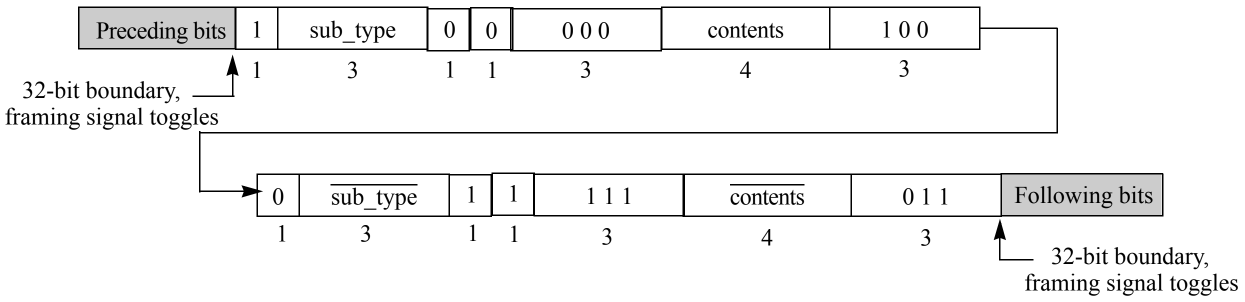


Figure 3-4. Type 4 Packet Control Symbol Format

Table 3-3 shows how sub_type values function with values of the contents field. For the idle, eop, and multicast-event control symbols the contents field is used as the buf_status field described in Section 1.2.1, whose encodings are specified in Table 1-2. For a throttle control symbol, the contents field specifies the number of aligned pacing idle control symbols that the sender should insert in the packet. One of the specified encodings indicates to the sender that it can immediately begin to resume packet transmission, as can be seen in Table 3-4. For the stomp and restart-from-retry control symbols, the contents field is unused and shall be tied to all logic 0’s and ignored by the receiving device.

Table 3-3. sub_type and contents Field Definitions

sub_type Field Definition	sub_type Encoding	contents Field Definition
idle	0b000	Used as a buf_status field that indicates the number of maximum-sized packets that can be received. Described in Section 1.2.1, encodings are defined in Table 1-2.
stomp	0b001	Unused, contents=0b0000
eop	0b010	Used as a buf_status field that indicates the number of maximum-sized packets that can be received. Described in Section 1.2.1, encodings are defined in Table 1-2.
restart-from-retry	0b011	Unused, contents=0b0000
throttle	0b100	Specifies the number of aligned pacing idles that the sender inserts in a packet. The encodings are defined in Table 9-4.

Table 3-3. sub_type and contents Field Definitions(Continued)

sub_type Field Definition	sub_type Encoding	contents Field Definition
Multicast-event	0b101	Used as a buf_status field that indicates the number of maximally sized packets that can be received. Described in Section 1.2.1, encodings are defined in Table 1-2.
Reserved	0b110-111	

The pacing idle count content field for a throttle control symbol is defined in Table 3-4

Table 3-4. Throttle Control Symbol contents Field Definition

Encoding	Definition
0b0000	1 aligned pacing idle control symbol
0b0001	2 aligned pacing idle control symbols
0b0010	4 aligned pacing idle control symbols
0b0011	8 aligned pacing idle control symbols
0b0100	16 aligned pacing idle control symbols
0b0101	32 aligned pacing idle control symbols
0b0110	64 aligned pacing idle control symbols
0b0111	128 aligned pacing idle control symbols
0b1000	256 aligned pacing idle control symbols
0b1001	512 aligned pacing idle control symbols
0b1010	1024 aligned pacing idle control symbols
0b1011-1101	Reserved
0b1110	1 aligned pacing idle control symbol for oscillator drift compensation
0b1111	Stop transmitting pacing idles, can immediately resume packet transmission

3.3 Link Maintenance Control Symbol Formats

Maintenance of a link is controlled by link-request/link-response control symbol pairs as described in the link maintenance protocol of Section 1.4. Each of the control symbols is described below:

- A link-request control symbol issues a command to or requests status from the device that is electrically connected, or linked, to the issuing device. The link-request control symbol is followed by a complemented version of itself as with the other control symbols. A link-request control symbol cannot be embedded in a packet, but can be used to cancel the packet. Under error conditions a link-request/input-status control symbol acts as a restart-from-error control symbol as described in Section 1.3.5.1, “Recoverable Errors.” This control symbol format is displayed in Figure 3-5.

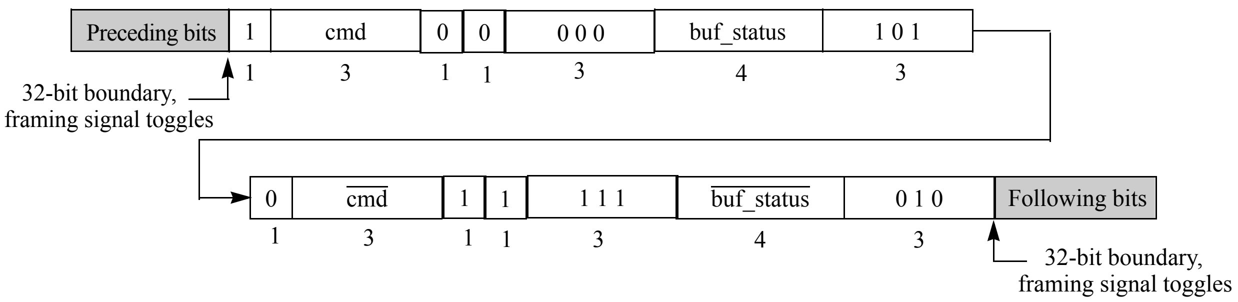


Figure 3-5. Type 5 Link-Request Control Symbol Format

The cmd, or command, field of the link-request control symbol format is defined in Table 3-5.

Table 3-5. cmd Field Definition

cmd Encoding	Command Name	Description
0b000	Send-training	Send 256 iterations of the training pattern
0b001-010		Reserved
0b011	Reset	Reset the receiving device
0b100	Input-status	Return input port status; functions as a restart-from-error control symbol under error conditions
0b101-111		Reserved

- The link-response control symbol is used by a device to respond to a link-request control symbol as described in the link maintenance protocol described in Section 1.4. The link-response control symbol is the same as all other control symbols in that the second 16 bits are a bit-wise inversion of the first 16 bits. A link-response control symbol can be embedded in a packet. This control symbol format is displayed in Figure 3-6.

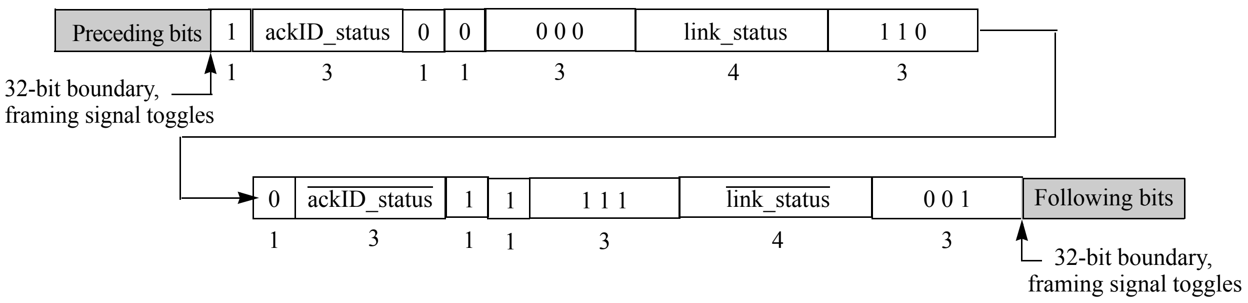


Figure 3-6. Type 6 Link-Response Control Symbol Format

The ackID_status field of the link-response format is defined in table 3-6.

Table 3-6. ackID_status Field Definition

Encoding	Description
0b000	Expecting ackID 0
0b001	Expecting ackID 1
0b010	Expecting ackID 2
0b011	Expecting ackID 3
0b100	Expecting ackID 4

Table 3-6. ackID_status Field Definition(Continued)

Encoding	Description
0b101	Expecting ackID 5
0b110	Expecting ackID 6
0b111	Expecting ackID 7

The link_status field is defined in Table 3-7. Note that the ackID information is included in both fields for additional error coverage if the receiver is working properly (encodings 8-15).

Table 3-7. link_status Field Definition

link_status Encoding	Port Status	Description
0b0000 - 0b0001	Reserved	
0b0010	Error	Unrecoverable error encountered.
0b0011	Reserved	
0b0100	Retry-stopped	The port has been stopped due to a retry.
0b0101	Error-stopped	The port has been stopped due to a transmission error; this state is cleared after the link-request/ input-status command is completed.
0b0110 - 0b0111	Reserved	
0b1000	OK, ackID0	Working properly, expecting ackID 0.
0b1001	OK, ackID1	Working properly, expecting ackID 1.
0b1010	OK, ackID2	Working properly, expecting ackID 2.
0b1011	OK, ackID3	Working properly, expecting ackID 3.
0b1100	OK, ackID4	Working properly, expecting ackID 4.
0b1101	OK, ackID5	Working properly, expecting ackID 5.
0b1110	OK, ackID6	Working properly, expecting ackID 6.
0b1111	OK, ackID7	Working properly, expecting ackID 7.

3.4 Reserved Symbol Formats

The control symbols corresponding to stypes 0b011 and 0b111 are reserved.

3.5 Training Pattern Format

A training pattern is needed in order to properly set up the input port to sample information coming in off of the wires. The training pattern is not delineated in the same way as are control symbols and packets, but is a special bit pattern that can be easily recognized by the input port logic and is ignored by the input once the device can reliably sample information from its input port.

A training pattern can not be embedded in a packet or used to terminate a packet. All ongoing activity shall be stopped gracefully before training patterns can be issued.

Notice that the training pattern is a 64-bit pattern for 8-bit ports and a 128-bit pattern for 16-bit ports, with the frame signal switching at the same time as the data bits. This format provides 4 beats of logic 1 alternating with 4 beats of logic 0 for both the data bits and the frame signal for both port widths. The frame signal does not have to transition high to low or low to high in phase with the data bits. The behavior of a device connected to an 8/16 LP-LVDS port with regards to the training pattern is described in Section 2.6.1.1, “Sampling Window Alignment.”

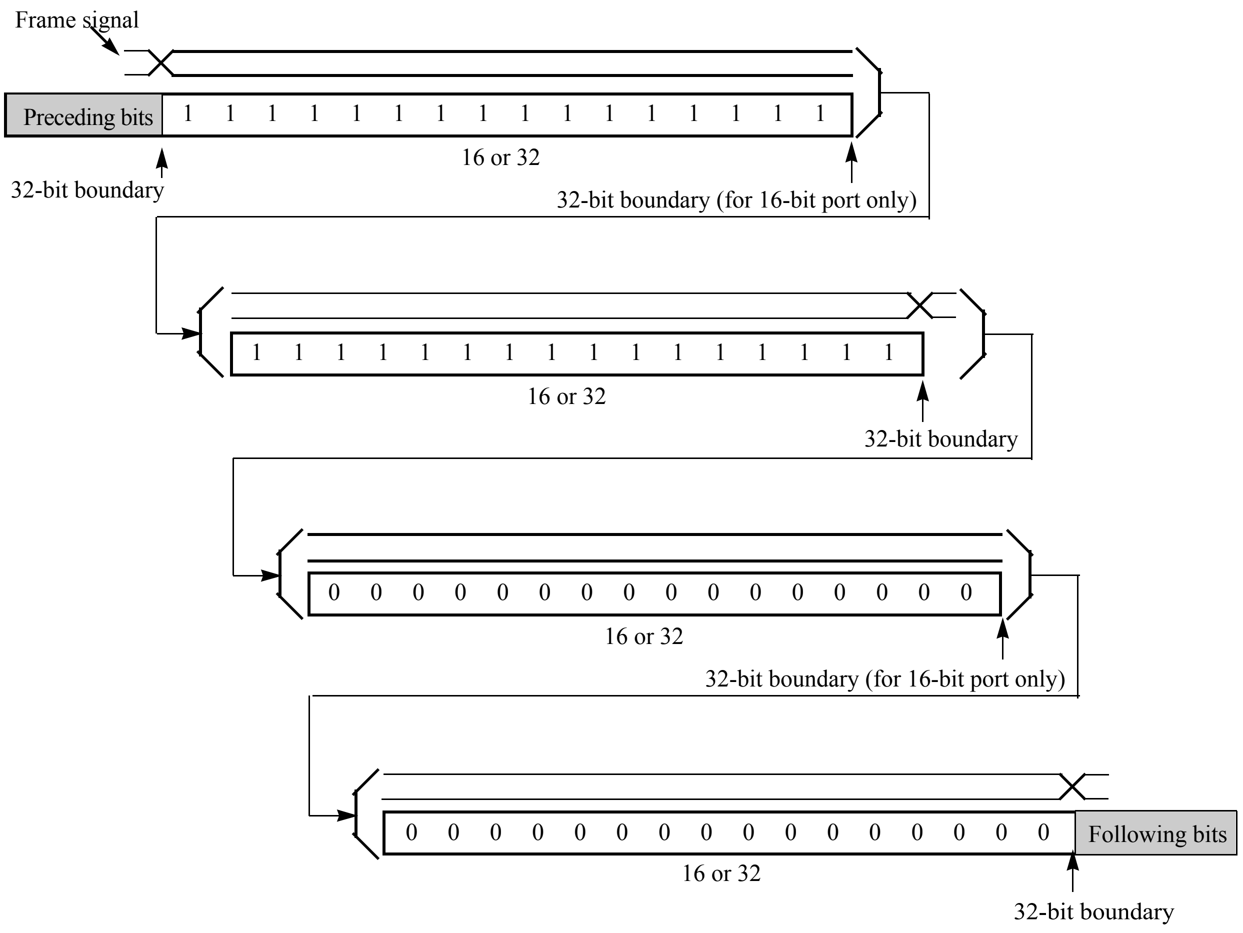


Figure 3-7. Type 7 TRAINING Pattern Format

3.6 Control Symbol to Port Alignment

This section shows examples of control symbol transmission over the 8-bit and 16-bit interfaces. The corresponding packet transmission alignment is shown in Section 2.5, “Packet to Port Alignment.”

Figure 3-8 shows the byte transmission ordering on an 8-bit port through time using an aligned packet-accepted control symbol as an example.

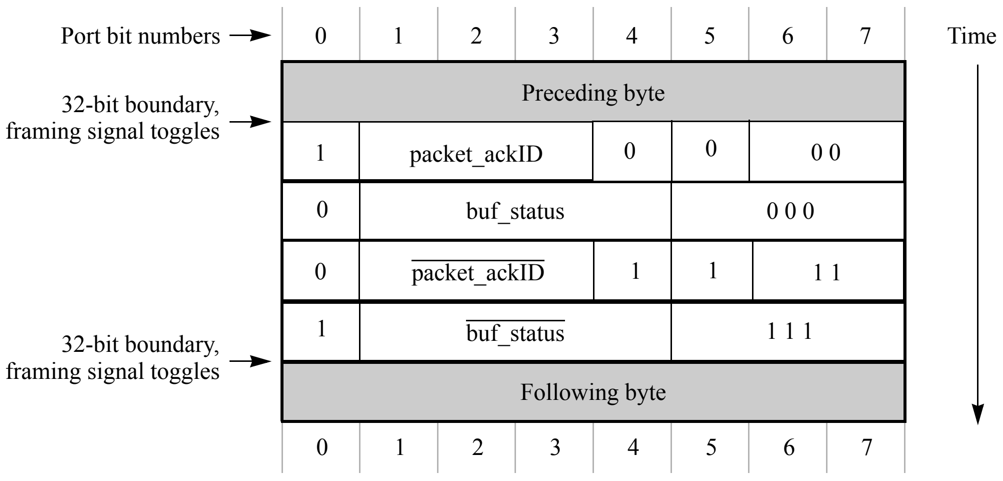

Port bit numbers
0
1
2
3
4
5
6
7
Time
32-bit boundary, framing signal toggles
Preceding byte
1
packet_ackID
0
0
0 0
0
buf_status
0 0 0
0
$\overline{\text{packet\_ackID}}$
1
1
1 1
1
$\overline{\text{buf\_status}}$
1 1 1
32-bit boundary, framing signal toggles
Following byte

Figure 3-8. Control Symbol Transmission Example 1

Figure 3-9 shows the same control symbol over the 16-bit interface.

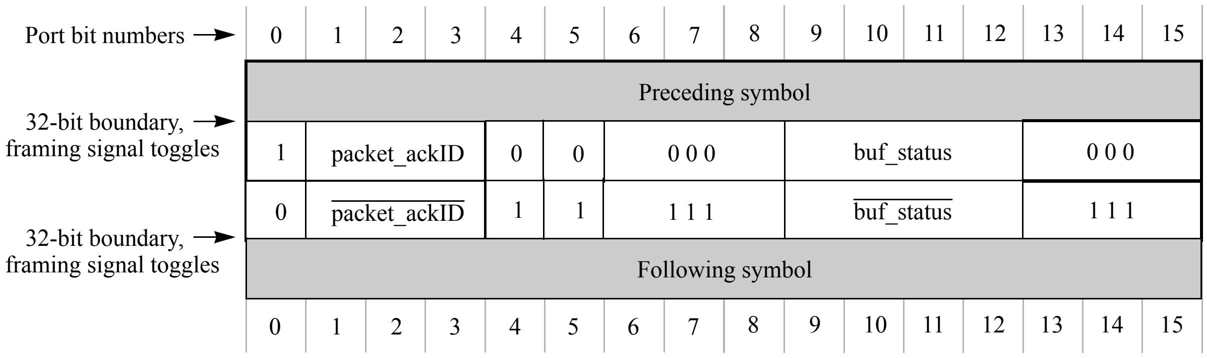

Port bit numbers
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
32-bit boundary, framing signal toggles
Preceding symbol
1
packet_ackID
0
0
0 0 0
buf_status
0 0 0
0
$\overline{\text{packet\_ackID}}$
1
1
1 1 1
$\overline{\text{buf\_status}}$
1 1 1
32-bit boundary, framing signal toggles
Following symbol

Figure 3-9. Control Symbol Transmission Example 2

4 Chapter 4 - 8/16 LP-LVDS Registers

This chapter describes the Command and Status Register (CSR) set that allows an external processing element to determine the capabilities, configuration, and status of a processing element using this physical layer specification. This chapter only describes registers or register bits defined by this specification. Refer to the other RapidIO logical, transport, and physical specifications of interest to determine a complete list of registers and bit definitions. All registers are 32-bits and aligned to a 32-bit boundary.

These registers utilize the Extended Features blocks and can be accessed using *Partition I: Input/Output Logical Specification* maintenance operations. Any register offsets not defined are considered reserved for this specification unless otherwise stated. Other registers required for a processing element are defined in other applicable RapidIO specifications and by the requirements of the specific device and are beyond the scope of this specification. Read and write accesses to reserved register offsets shall terminate normally and not cause an error condition in the target device. The Extended Features pointer (EF_PTR) defined in the RapidIO logical specifications contains the offset of the first Extended Features block in the Extended Features data structure for a device. The 8/16 LP-LVDS physical features block shall exist in any position in the Extended Features data structure and shall exist in any portion of the Extended Features Space in the register address map for the device.

Table 4-1 describes the required behavior for accesses to reserved register bits and reserved registers for the RapidIO

Extended Features register space,

Table 4-1. Extended Feature Space Reserved Access Behavior

Byte Offset	Space Name	Item	Initiator behavior	Target behavior
0x100–FFFC	Extended Features Space	Reserved bit	read - ignore returned value[1]	read - return logic 0
			write - preserve current value[2]	write - ignored
		Implementation-defined bit	read - ignore returned value unless implementation-defined function understood	read - return implementation-defined value
			write - preserve current value if implementation-defined function not understood	write - implementation-defined
		Reserved register	read - ignore returned value	read - return logic 0s
			write -	write - ignored

1. Do not depend on reserved bits being a particular value; use appropriate masks to extract defined bits from the read value.
2. All register writes shall be in the form: read the register to obtain the values of all reserved bits, merge in the desired values for defined bits to be modified, and write the register, thus preserving the value of all reserved bits.

This chapter is divided up into three sections, each addressing a different type of RapidIO device.

4.1 Generic End Point Devices

This section describes the 8/16 LP-LVDS registers for a general end point device. This Extended Features register block is assigned Extended Features block ID=0x0001.

4.1.1 Register Map

Table 4-2 shows the register map for generic RapidIO 8/16 LP-LVDS end point devices. The Block Offset is the offset based on the Extended Features pointer (EF_PTR) to this block. This register map is currently only defined for devices with up to 16 RapidIO ports, but can be extended or shortened if more or less port definitions are required for a device. For example, a device with four RapidIO ports is only required to use register map space corresponding to offsets [EF_PTR + 0x00] through [EF_PTR + 0x98]. Register map offset [EF_PTR + 0xA0] can be used for another Extended Features block.

Table 4-2. Physical 8/16 LP-LVDS Register Map

	Block Byte Offset	Register Name (Word 0)	Register Name (Word 1)
	0x0	8/16 LP-LVDS Port Maintenance Block Header	
	0x8–18	Reserved	
General	0x20	Port Link Time-Out Control CSR	Port Response Time-Out Control CSR
	0x28	Reserved	
	0x30	Reserved	
	0x38	Reserved	Port General Control CSR
Port 0	0x40	Reserved	
	0x48	Reserved	
	0x50	Reserved	
	0x58	Port 0 Error and Status CSR	Port 0 Control CSR

Table 4-2. Physical 8/16 LP-LVDS Register Map(Continued)

	Block Byte Offset	Register Name (Word 0)	Register Name (Word 1)
Port 1	0x60	Reserved	
	0x68	Reserved	
	0x70	Reserved	
	0x78	Port 1 Error and Status CSR	Port 1 Control CSR
Port 2-14	0x80–218	Assigned to Port 2-14 CSRs	
Port 15	0x220	Reserved	
	0x228	Reserved	
	0x230	Reserved	
	0x238	Port 15 Error and Status CSR	Port 15 Control CSR

4.1.2 Command and Status Registers (CSRs)

Refer to table 4-1 for the required behavior for accesses to reserved registers and register bits.

4.1.2.1 Port Maintenance Block Header 0 (Block Offset 0x0 Word 0)

The port maintenance block header 0 register contains the EF_PTR to the next EF_BLK and the EF_ID that identifies this as the generic end point port maintenance block header.

Table 4-3. Bit Settings for Port Maintenance Block Header 0

Bit	Name	Reset Value	Description
0-15	EF_PTR		Hard wired pointer to the next block in the data structure, if one exists
16-31	EF_ID	0x0001	Hard wired Extended Features ID

4.1.2.2 Port Maintenance Block Header 1 (Block Offset 0x0 Word 1)

The port maintenance block header 1 register is reserved.

Table 4-4. Bit Settings for Port Maintenance Block Header 1

Bit	Name	Reset Value	Description
0-31	—		Reserved

4.1.2.3 Port Link Time-out Control CSR (Block Offset 0x20 Word 0)

The port link time-out control register contains the time-out timer value for all ports on a device. This time-out is for link events such as sending a packet to receiving the corresponding acknowledge, and sending a link-request to receiving the corresponding link-response. The reset value is the maximum time-out interval, and represents between 3 and 5 seconds.

Table 4-5. Bit Settings for Port Link Time-out Control CSR

Bit	Name	Reset Value	Description
0–23	time-out_value	All 1s	time-out interval value
24-31	—		Reserved

4.1.2.4 Port Response Time-out Control CSR (Block Offset 0x20 Word 1)

The port response time-out control register contains the time-out timer count for all ports on a device. This time-out is for sending a request packet to receiving the corresponding response packet.The reset value is the maximum time-out interval, and represents between 3 and 5 seconds.

Table 4-6. Bit Settings for Port Response Time-out Control CSR

Bit	Name	Reset Value	Description
0–23	time-out_value	All 1s	time-out interval value
24-31	—		Reserved

4.1.2.5 Port General Control CSR (Block Offset 0x38 Word 1)

The port general control register contains control register bits applicable to all ports on a processing element.

Table 4-7. Bit Settings for Port General Control CSRs

Bit	Name	Reset Value	Description
0	Host	see footnote[1]	A Host device is a device that is responsible for system exploration, initialization, and maintenance. Agent or slave devices are typically initialized by Host devices. 0b0 - agent or slave device 0b1 - host device
1	Master Enable	see footnote[2]	The Master Enable bit controls whether or not a device is allowed to issue requests into the system. If the Master Enable is not set, the device may only respond to requests. 0b0 - processing element cannot issue requests 0b1 - processing element can issue requests
2	Discovered	see footnote[3]	This device has been located by the processing element responsible for system configuration 0b0 - The device has not been previously discovered 0b1 - The device has been discovered by another processing element
3-31	—		Reserved

1. The Host reset value is implementation dependent
2. The Master Enable reset value is implementation dependent
3. The Discovered reset value is implementation dependent

4.1.2.6 Port *n* Error and Status CSRs (Block Offsets 0x58, 78, ..., 238 Word 0)

These registers are accessed when a local processor or an external device wishes to examine the port error and status information.

Table 4-8. Bit Settings for Port *n* Error and Status CSRs

Bit	Name	Reset Value	Description
0-10	—		Reserved
11	Output Retry-encountered	0b0	Output port has encountered a retry condition. This bit is set when bit 13 is set. Once set remains set until written with a logic 1 to clear.
12	Output Retried	0b0	Output port has received a packet-retry control symbol and can not make forward progress. This bit is set when bit 13 is set and is cleared when a packet-accepted or a packet-not-accepted control symbol is received (read-only).
13	Output Retry-stopped	0b0	Output port has received a packet-retry control symbol and is in the "output retry-stopped" state (read-only).
14	Output Error-encountered	0b0	Output port has encountered (and possibly recovered from) a transmission error. This bit is set when bit 15 is set. Once set remains set until written with a logic 1 to clear.
15	Output Error-stopped	0b0	Output port is in the "output error-stopped" state (read-only).
16-20	—		Reserved
21	Input Retry-stopped	0b0	Input port is in the "input retry-stopped" state (read-only).
22	Input Error-encountered	0b0	Input port has encountered (and possibly recovered from) a transmission error. This bit is set when bit 23 is set. Once set remains set until written with a logic 1 to clear.
23	Input Error-stopped	0b0	Input port is in the "input error-stopped" state (read-only).
24-26	—		Reserved
27	Port-write Pending	0b0	Port has encountered a condition which required it to initiate a Maintenance Port-write operation.This bit is only valid if the device is capable of issuing a maintenance port-write transaction. Once set remains set until written with a logic 1 to clear.
28	Port Present	0b0	The port is receiving the free-running clock on the input port.
29	Port Error	0b0	Input or output port has encountered an error from which hardware was unable to recover. Once set remains set until written with a logic 1 to clear.
30	Port OK	0b0	Input and output ports are initialized and can communicate with the adjacent device. This bit and bit 31 are mutually exclusive (read-only).
31	Port Uninitialized	0b1	Input and output ports are not initialized and is in training mode. This bit and bit 30 are mutually exclusive (read-only).

4.1.2.7 Port *n* Control CSR (Block Offsets 0x58, 78, ..., 238 Word 1)

The port *n* control registers contain control register bits for individual ports on a processing element.

Table 4-9. Bit Settings for Port *n* Control CSRs (continued)

Bit	Name	Reset Value	Description
0	Output Port Width	see footnote[1]	Operating width of the port (read-only): 0b0 - 8-bit port 0b1 - 16-bit port
1	Output Port Enable	see footnote[2]	Output port transmit enable: 0b0 - port is stopped and not enabled to issue any packets except to route or respond to I/O logical MAINTENANCE packets, depending upon the functionality of the processing element. Control symbols are not affected and are sent normally. 0b1 - port is enabled to issue any packets
2	Output Port Driver Disable	0b0	Output port driver disable: 0b0 - output port drivers are turned on and will drive the pins normally 0b1 - output port drivers are turned off and will not drive the pins This is useful for power management.
3	—		Reserved
4	Input Port Width	see footnote[3]	Operating width of the port (read-only): 0b0 - 8-bit port 0b1 - 16-bit port
5	Input Port Enable	see footnote[4]	Input port receive enable: 0b0 - port is stopped and only enabled to route or respond I/O logical MAINTENANCE packets, depending upon the functionality of the processing element. Other packets generate packet-not-accepted control symbols to force an error condition to be signaled by the sending device. Control symbols are not affected and are received and handled normally. 0b1 - port is enabled to respond to any packet
6	Input Port Receiver Disable	0b0	Input port receiver enable: 0b0 - input port receivers are enabled 0b1 - input port receivers are disabled and are unable to receive to any packets or control symbols
7	—		Reserved
8	Error Checking Disable	0b0	This bit disables all RapidIO transmission error checking 0b0 - Error checking and recovery is enabled 0b1 - Error checking and recovery is disabled Device behavior when error checking and recovery is disabled and an error condition occurs is undefined
9	Multicast-event Participant	see footnote[5]	Send incoming multicast-event control symbols to this port (multiple port devices only)
10-30	—		Reserved
31	Port Type	0b0	This indicates the port type, parallel or serial (read only) 0b0 - Parallel port 0b1 - Serial port

1. The output port width reset value is implementation dependent
2. The output port enable reset value is implementation dependent
3. The input port width reset value is implementation dependent
4. The Input port enable reset value is implementation dependent
5. The multicast-event participant reset value is implementation dependent

4.2 Generic End Point Devices, software assisted error recovery option

This section describes the 8/16 LP-LVDS registers for a general end point device that supports software assisted error recovery. This is most useful for devices that for whatever reason do not want to implement error recovery in hardware and to allow software to generate link request control symbols and see the results of the responses. This Extended Features register block is assigned Extended Features block ID=0x0002.

4.2.1 Register Map

Table 4.10 shows the register map for generic RapidIO 8/16 LP-LVDS end point devices with software assisted error recovery. The Block Offset is the offset based on the Extended Features pointer (EF_PTR) to this block. This register map is currently only defined for devices with up to 16 RapidIO ports, but can be extended or shortened if more or less port definitions are required for a device. For example, a device with four RapidIO ports is only required to use register map space corresponding to offsets [EF_PTR + 0x00] through [EF_PTR + 0x98]. Register map offset [EF_PTR + 0xA0] can be used for another Extended Features block.

Table 4-10. Physical 8/16 LP-LVDS Register Map

	Block Byte Offset	**Register Name (Word 0)**	**Register Name (Word 1)**
	0x0	8/16 LP-LVDS Port Maintenance Block Header	
	0x8–18	Reserved	
General	0x20	Port Link Time-Out Control CSR	Port Response Time-Out Control CSR
General	0x28	Reserved	
General	0x30	Reserved	
General	0x38	Reserved	Port General Control CSR
Port 0	0x40	Port 0 Link Maintenance Request CSR	Port 0 Link Maintenance Response CSR
Port 0	0x48	Port 0 Local ackID Status CSR	Reserved
Port 0	0x50	Reserved	
Port 0	0x58	Port 0 Error and Status CSR	Port 0 Control CSR
Port 1	0x60	Port 1 Link Maintenance Request CSR	Port 1Link Maintenance Response CSR
Port 1	0x68	Port 1 Local ackID Status CSR	Reserved
Port 1	0x70	Reserved	
Port 1	0x78	Port 1 Error and Status CSR	Port 1 Control CSR
Port 2-14	0x80–218	Assigned to Port 2-14 CSRs	

Table 4-10. Physical 8/16 LP-LVDS Register Map(Continued)

	Block Byte Offset	**Register Name (Word 0)**	**Register Name (Word 1)**
Port 15	0x220	Port 15 Link Maintenance Request CSR	Port 15 Link Maintenance Response CSR
	0x228	Port 15 Local ackID Status CSR	Reserved
	0x230	Reserved	
	0x238	Port 15 Error and Status CSR	Port 15 Control CSR

4.2.2 Command and Status Registers (CSRs)

Refer to Table 4-1 for the required behavior for accesses to reserved registers and register bits.

4.2.2.1 Port Maintenance Block Header 0 (Block Offset 0x0 Word 0)

The port maintenance block header 0 register contains the EF_PTR to the next EF_BLK and the EF_ID that identifies this as the generic end point port maintenance block header.

Table 4-11. Bit Settings for Port Maintenance Block Header 0

Bit	Name	Reset Value	Description
0-15	EF_PTR		Hard wired pointer to the next block in the data structure, if one exists
16-31	EF_ID	0x0002	Hard wired Extended Features ID

4.2.2.2 Port Maintenance Block Header 1 (Block Offset 0x0 Word 1)

The port maintenance block header 1 register is reserved.

Table 4-12. Bit Settings for Port Maintenance Block Header 1

Bit	Name	Reset Value	Description
0-31	—		Reserved

4.2.2.3 Port Link Time-out Control CSR (Block Offset 0x20 Word 0)

The port link time-out control register contains the time-out timer value for all ports on a device. This time-out is for link events such as sending a packet to receiving the corresponding acknowledge and sending a link-request to receiving the corresponding link-response. The reset value is the maximum time-out interval, and represents between 3 and 5 seconds.

Table 4-13. Bit Settings for Port Link Time-out Control CSR

Bit	Name	Reset Value	Description
0–23	time-out_value	All 1s	time-out interval value
24-31	—		Reserved

4.2.2.4 Port Response Time-out Control CSR (Block Offset 0x20 Word 1)

The port response time-out control register contains the time-out timer count for all ports on a device. This time-out is for sending a request packet to receiving the corresponding response packet.The reset value is the maximum time-out interval, and represents between 3 and 5 seconds.

Table 4-14. Bit Settings for Port Response Time-out Control CSR

Bit	Name	Reset Value	Description
0–23	time-out_value	All 1s	time-out interval value
24-31	—		Reserved

4.2.2.5 Port General Control CSR (Block Offset 0x38 Word 1)

The port general control register contains control register bits applicable to all ports on a processing element.

Table 4-15. Bit Settings for Port General Control CSRs

Bit	Name	Reset Value	Description
0	Host	see footnote[1]	A Host device is a device that is responsible for system exploration, initialization, and maintenance. Agent or slave devices are initialized by Host devices. 0b0 - agent or slave device 0b1 - host device
1	Master Enable	see footnote[2]	The Master Enable bit controls whether or not a device is allowed to issue requests into the system. If the Master Enable is not set, the device may only respond to requests. 0b0 - processing element cannot issue requests 0b1 - processing element can issue requests
2	Discovered	see footnote[3]	This device has been located by the processing element responsible for system configuration 0b0 - The device has not been previously discovered 0b1 - The device has been discovered by another processing element
3-31	—		Reserved

1. The Host reset value is implementation dependent
2. The Master Enable reset value is implementation dependent
3. The Discovered reset value is implementation dependent

4.2.2.6 Port *n* Link Maintenance Request CSRs (Block Offsets 0x40, 60, ..., 220 Word 0)

The port link maintenance request registers are accessible both by a local processor and an external device. A write to one of these registers generates a link-request control symbol on the corresponding RapidIO port interface.

Table 4-16. Bit Settings for Port *n* Link Maintenance Request CSRs

Bit	Name	Reset Value	Description
0–28	—		Reserved
29-31	Command	0b000	Command to be sent in the link-request control symbol. If read, this field returns the last written value.

4.2.2.7 Port *n* Link Maintenance Response CSRs (Block Offsets 0x40, 60, ..., 220 Word 1)

The port link maintenance response registers are accessible both by a local processor and an external device. A read to this register returns the status received in a link-response control symbol. The link_status and ackID_status fields are defined in Section 3.3, “Link Maintenance Control Symbol Formats.” This register is read-only.

Table 4-17. Bit Settings for Port *n* Link Maintenance Response CSRs

Bit	Name	Reset Value	Description
0	response_valid	0b0	If the link-request causes a link-response, this bit indicates that the link-response has been received and the status fields are valid. If the link-request does not cause a link-response, this bit indicates that the link-request has been transmitted. This bit automatically clears on read.
1-24	—		Reserved
25-27	ackID_status	0b000	ackID status field from the link-response control symbol
28-31	link_status	0b0000	link status field from the link-response control symbol

4.2.2.8 Port *n* Local ackID Status CSRs (Block Offsets 0x48, 68, ..., 228 Word 0)

The port link local ackID status registers are accessible both by a local processor and an external device. A read to this register returns the local ackID status for both the out and input ports of the device.

Table 4-18. Bit Settings for Port *n* Local ackID Status CSRs

Bit	Name	Reset Value	Description
0-4	—		Reserved
5-7	Inbound_ackID	0b000	Input port next expected ackID value
8-15	—		Reserved
16-23	Outstanding_ackID	0x00	Output port unacknowledged ackID status. A set bit indicates that the corresponding ackID value has been used to send a packet to an attached device but a corresponding acknowledge control symbol has not been received. 0b1xxx_xxxx indicates ackID 0, 0bx1xx_xxxx indicates ackID 1, 0bxx1x_xxxx indicates ackID 2, etc. This field is read-only.
24-28	—		Reserved
29-31	Outbound_ackID	0b000	Output port next transmitted ackID value. Software writing this value can force re-transmission of outstanding unacknowledged packets in order to manually implement error recovery.

4.2.2.9 Port *n* Error and Status CSRs (Block Offsets 0x58, 78, ..., 238 Word 0)

These registers are accessed when a local processor or an external device wishes to examine the port error and status information.

Table 4-19. Bit Settings for Port *n* Error and Status CSRs

Bit	Name	Reset Value	Description
0-10	—		Reserved
11	Output Retry-encountered	0b0	Output port has encountered a retry condition. This bit is set when bit 13 is set. Once set remains set until written with a logic 1 to clear.
12	Output Retried	0b0	Output port has received a packet-retry control symbol and can not make forward progress. This bit is set when bit 13 is set and is cleared when a packet-accepted or a packet-not-accepted control symbol is received (read-only).
13	Output Retry-stopped	0b0	Output port has received a packet-retry control symbol and is in the “output retry-stopped” state (read-only).
14	Output Error-encountered	0b0	Output port has encountered (and possibly recovered from) a transmission error. This bit is set when bit 15 is set. Once set remains set until written with a logic 1 to clear.
15	Output Error-stopped	0b0	Output port is in the “output error-stopped” state (read-only).
16-20	—		Reserved
21	Input Retry-stopped	0b0	Input port is in the “input retry-stopped” state (read-only).
22	Input Error-encountered	0b0	Input port has encountered (and possibly recovered from) a transmission error. This bit is set when bit 23 is set. Once set remains set until written with a logic 1 to clear.

Table 4-19. Bit Settings for Port *n* Error and Status CSRs(Continued)

Bit	Name	Reset Value	Description
23	Input Error-stopped	0b0	Input port is in the “input error-stopped” state (read-only).
24-26	—		Reserved
27	Port-write Pending	0b0	Port has encountered a condition which required it to initiate a Maintenance Port-write operation.This bit is only valid if the device is capable of issuing a maintenance port-write transaction. Once set remains set until written with a logic 1 to clear.
28	Port Present	0b0	The port is receiving the free-running clock on the input port.
29	Port Error	0b0	Input or output port has encountered an error from which hardware was unable to recover. Once set remains set until written with a logic 1 to clear.
30	Port OK	0b0	Input and output ports are initialized and can communicate with the adjacent device. This bit and bit 31 are mutually exclusive (read-only).
31	Port Uninitialized	0b1	Input and output ports are not initialized and is in training mode. This bit and bit 30 are mutually exclusive (read-only).

4.2.2.10 Port *n* Control CSR (Block Offsets 0x58, 78, ..., 238 Word 1)

The port *n* control registers contain control register bits for individual ports on a processing element.

Table 4-20. Bit Settings for Port *n* Control CSRs

Bit	Name	Reset Value	Description
0	Output Port Width	see footnote[1]	Operating width of the port (read-only): 0b0 - 8-bit port 0b1 - 16-bit port
1	Output Port Enable	see footnote[2]	Output port transmit enable: 0b0 - port is stopped and not enabled to issue any packets except to route or respond to I/O logical MAINTENANCE packets, depending upon the functionality of the processing element. Control symbols are not affected and are sent normally. 0b1 - port is enabled to issue any packets
2	Output Port Driver Disable	0b0	Output port driver disable: 0b0 - output port drivers are turned on and will drive the pins normally 0b1 - output port drivers are turned off and will not drive the pins This is useful for power management.
3	—		Reserved
4	Input Port Width	see footnote[3]	Operating width of the port (read-only): 0b0 - 8-bit port 0b1 - 16-bit port

Table 4-20. Bit Settings for Port *n* Control CSRs(Continued)

Bit	Name	Reset Value	Description
5	Input Port Enable	see footnote[4]	Input port receive enable: 0b0 - port is stopped and only enabled to route or respond I/O logical MAINTENANCE packets, depending upon the functionality of the processing element. Other packets generate packet-not-accepted control symbols to force an error condition to be signaled by the sending device. Control symbols are not affected and are received and handled normally. 0b1 - port is enabled to respond to any packet
6	Input Port Receiver Disable	0b0	Input port receiver enable: 0b0 - input port receivers are enabled 0b1 - input port receivers are disabled and are unable to receive to any packets or control symbols
7	—		Reserved
8	Error Checking Disable	0b0	This bit disables all RapidIO transmission error checking 0b0 - Error checking and recovery is enabled 0b1 - Error checking and recovery is disabled Device behavior when error checking and recovery is disabled and an error condition occurs is undefined
9	Multicast-event Participant	see footnote[5]	Send incoming multicast-event control symbols to this port (multiple port devices only)
10-30	—		Reserved
31	Port Type	0b0	This indicates the port type, parallel or serial (read only) 0b0 - Parallel port 0b1 - Serial port

1. The output port width reset value is implementation dependent
2. The output port enable reset value is implementation dependent
3. The input port width reset value is implementation dependent
4. The Input port enable reset value is implementation dependent
5. The multicast-event participant reset value is implementation dependent

4.3 Generic End Point Free Devices

This section describes the 8/16 LP-LVDS registers for a general devices that do not contain end point functionality. Typically these devices are switches. This Extended Features register block uses extended features block ID=0x0003.

4.3.1 Register Map

Table 4-21 shows the register map for generic RapidIO 8/16 LP-LVDS end point-free devices. The Block Offset is the offset based on the Extended Features pointer (EF_PTR) to this block. This register map is currently only defined for devices with up to 16 RapidIO ports, but can be extended or shortened if more or less port definitions are required for a device. For example, a device with four RapidIO ports is only required to use register map space corresponding to offsets [EF_PTR + 0x00] through [EF_PTR + 0x98]. Register map offset [EF_PTR + 0xA0] can be used for another Extended Features block.

Table 4-21. Physical 8/16 LP-LVDS Register Map

	Block Byte Offset	Register Name (Word 0)	Register Name (Word 1)
	0x0	8/16 LP-LVDS Port Maintenance Block Header	
	0x8–18	Reserved	
General	0x20	Port Link Time-Out Control CSR	Reserved
General	0x28	Reserved	
General	0x30	Reserved	
General	0x38	Reserved	Port General Control CSR
Port 0	0x40	Reserved	
Port 0	0x48	Reserved	
Port 0	0x50	Reserved	
Port 0	0x58	Port 0 Error and Status CSR	Port 0 Control CSR
Port 1	0x60	Reserved	
Port 1	0x68	Reserved	
Port 1	0x70	Reserved	
Port 1	0x78	Port 1 Error and Status CSR	Port 1 Control CSR
Port 2-14	0x80–218	Assigned to Port 2-14 CSRs	
Port 15	0x220	Reserved	
Port 15	0x228	Reserved	
Port 15	0x230	Reserved	
Port 15	0x238	Port 15 Error and Status CSR	Port 15 Control CSR

4.3.2 Command and Status Registers (CSRs)

Refer to Table 4-1 for the required behavior for accesses to reserved registers and register bits.

4.3.2.1 Port Maintenance Block Header 0 (Block Offset 0x0 Word 0)

The port maintenance block header 0 register contains the EF_PTR to the next EF_BLK and the EF_ID that identifies this as the generic end point port maintenance block header.

Table 4-22. Bit Settings for Port Maintenance Block Header 0

Bit	Name	Reset Value	Description
0-15	EF_PTR		Hard wired pointer to the next block in the data structure, if one exists
16-31	EF_ID	0x0003	Hard wired Extended Features ID

4.3.2.2 Port Maintenance Block Header 1 (Block Offset 0x0 Word 1)

The port maintenance block header 1 register is reserved.

Table 4-23. Bit Settings for Port Maintenance Block Header 1

Bit	Name	Reset Value	Description
0-31	—		Reserved

4.3.2.3 Port Link Time-out Control CSR (Block Offset 0x20 Word 0)

The port link time-out control register contains the time-out timer value for all ports on a device. This time-out is for link events such as sending a packet to receiving the corresponding acknowledge and sending a link-request to receiving the corresponding link-response. The reset value is the maximum time-out interval, and represents between 3 and 5 seconds.

Table 4-24. Bit Settings for Port Link Time-out Control CSR

Bit	Name	Reset Value	Description
0–23	time-out_value	All 1s	time-out interval value
24-31	—		Reserved

4.3.2.4 Port General Control CSR (Block Offset 0x38 Word 1)

The port general control register contains control register bits applicable to all ports on a processing element.

Table 4-25. Bit Settings for Port General Control CSRs

Bit	Name	Reset Value	Description
0-31	—		Reserved

4.3.2.5 Port *n* Error and Status CSRs (Block Offsets 0x58, 78, ..., 238 Word 0)

These registers are accessed when a local processor or an external device wishes to examine the port error and status information.

Table 4-26. Bit Settings for Port *n* Error and Status CSRs

Bit	Name	Reset Value	Description
0-10	—		Reserved
11	Output Retry-encountered	0b0	Output port has encountered a retry condition. This bit is set when bit 13 is set. Once set remains set until written with a logic 1 to clear.
12	Output Retried	0b0	Output port has received a packet-retry control symbol and can not make forward progress. This bit is set when bit 13 is set and is cleared when a packet-accepted or a packet-not-accepted control symbol is received (read-only).
13	Output Retry-stopped	0b0	Output port has received a packet-retry control symbol and is in the “output retry-stopped” state (read-only).
14	Output Error-encountered	0b0	Output port has encountered (and possibly recovered from) a transmission error. This bit is set when bit 15 is set. Once set remains set until written with a logic 1 to clear.
15	Output Error-stopped	0b0	Output port is in the “output error-stopped” state (read-only).

Table 4-26. Bit Settings for Port *n* Error and Status CSRs(Continued)

Bit	Name	Reset Value	Description
16-20	—		Reserved
21	Input Retry-stopped	0b0	Input port is in the “input retry-stopped” state (read-only).
22	Input Error-encountered	0b0	Input port has encountered (and possibly recovered from) a transmission error. This bit is set when bit 23 is set. Once set remains set until written with a logic 1 to clear.
23	Input Error-stopped	0b0	Input port is in the “input error-stopped” state (read-only).
24-26	—		Reserved
27	Port-write Pending	0b0	Port has encountered a condition which required it to initiate a Maintenance Port-write operation.This bit is only valid if the device is capable of issuing a maintenance port-write transaction. Once set remains set until written with a logic 1 to clear.
28	Port Present	0b0	The port is receiving the free-running clock on the input port.
29	Port Error	0b0	Input or output port has encountered an error from which hardware was unable to recover. Once set remains set until written with a logic 1 to clear.
30	Port OK	0b0	Input and output ports are initialized and can communicate with the adjacent device. This bit and bit 31 are mutually exclusive (read-only).
31	Port Uninitialized	0b1	Input and output ports are not initialized and is in training mode. This bit and bit 30 are mutually exclusive (read-only).

4.3.2.6 Port *n* Control CSR (Block Offsets 0x58, 78, ..., 238 Word 1)

The port *n* control registers contain control register bits for individual ports on a processing element.

Table 4-27. Bit Settings for Port *n* Control CSRs

Bit	Name	Reset Value	Description
0	Output Port Width	see footnote[1]	Operating width of the port (read-only): 0b0 - 8-bit port 0b1 - 16-bit port
1	Output Port Enable	see footnote[2]	Output port transmit enable: 0b0 - port is stopped and not enabled to issue any packets except to route or respond to I/O logical MAINTENANCE packets, depending upon the functionality of the processing element. Control symbols are not affected and are sent normally. 0b1 - port is enabled to issue any packets
2	Output Port Driver Disable	0b0	Output port driver disable: 0b0 - output port drivers are turned on and will drive the pins normally 0b1 - output port drivers are turned off and will not drive the pins This is useful for power management.
3	—		Reserved

Table 4-27. Bit Settings for Port *n* Control CSRs(Continued)

Bit	Name	Reset Value	Description
4	Input Port Width	see footnote[3]	Operating width of the port (read-only): 0b0 - 8-bit port 0b1 - 16-bit port
5	Input Port Enable	see footnote[4]	Input port receive enable: 0b0 - port is stopped and only enabled to route or respond I/O logical MAINTENANCE packets, depending upon the functionality of the processing element. Other packets generate packet-not-accepted control symbols to force an error condition to be signaled by the sending device. Control symbols are not affected and are received and handled normally. 0b1 - port is enabled to respond to any packet
6	Input Port Receiver Disable	0b0	Input port receiver enable: 0b0 - input port receivers are enabled 0b1 - input port receivers are disabled and are unable to receive to any packets or control symbols
7-8	—		Reserved
9	Multicast-event Participant	see footnote[5]	Send incoming multicast-event control symbols to this output port (multiple port devices only)
10-30	—		Reserved
31	Port Type	0b0	This indicates the port type, parallel or serial (read only) 0b0 - Parallel port 0b1 - Serial port

1. The output port width reset value is implementation dependent
2. The output port enable reset value is implementation dependent
3. The input port width reset value is implementation dependent
4. The Input port enable reset value is implementation dependent
5. The multicast-event participant reset value is implementation dependent

5 Chapter 5 - System Clocking Considerations

The RapidIO parallel physical interface can be deployed in a variety of system configurations. A fundamental aspect to the successful deployment of RapidIO is clock distribution. This section is provided to point out the issues of distributing clocks in a system.

5.1 Example Clock Distribution

Clock distribution in a small system is straightforward. It is assumed that clocking is provided from a single clock source-Figure 5-1).-

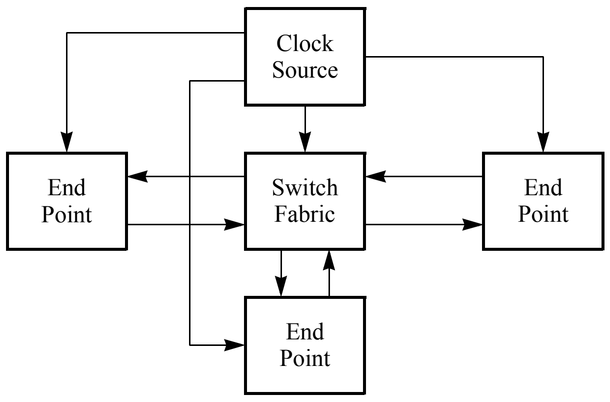


Figure 5-1. Clock Distribution in a Small System

In this case the timing budget must account for any skew and jitter component between each point. Skew and jitter are introduced owing to the end point clock regeneration circuitry (PLL or DLL) and to transmission line effects.

Distributing a clock from a central source may not be practical in larger or more robust systems. In these cases it may be desirable to have multiple clock sources or to distribute the clock through the interconnect. Figure 5-2 displays the clock distribution in a larger system.

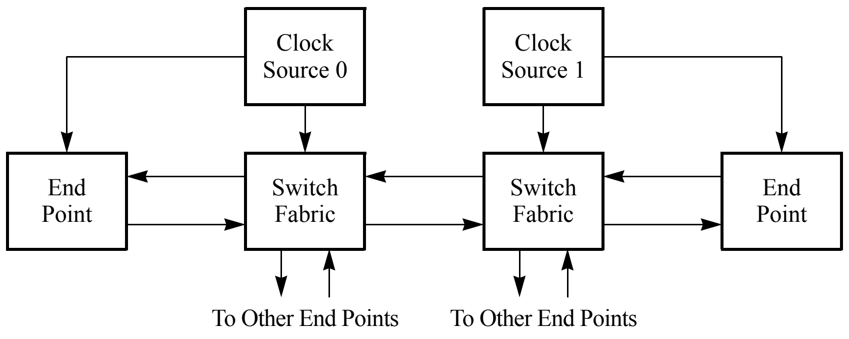


Figure 5-2. Clock Distribution in a Larger System

In such a system the clock sources may be of the same relative frequency; however, they are not guaranteed to be always at exact frequency. Clock sources will drift in phase relationship with each other over time. This adds an additional component because it is possible that one device may be slightly faster than its companion device. This requires a packet elasticity mechanism. If the clock is transported through the interconnect as shown in Figure 5-3, then additive clock jitter must be taken into account.

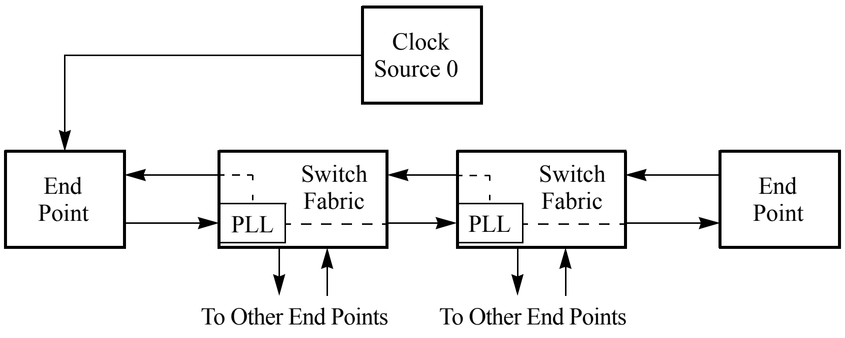


Figure 5-3. Clock Distribution Through the Interconnect

Assuming that each device gets a clock that was regenerated by its predecessor, and each device adds a certain jitter component to the clock, the resulting clock at the end point may be greatly unstable. This factor must be added to the timing budget.

5.2 Elasticity Mechanism

In systems with multiple clock sources, clocks may be of the same relative frequency but not exact. Their phase will drift over time. An elasticity mechanism is therefore required to keep devices from missing data beats. For example, if the received clock is faster than the internal clock, then it may be necessary to delete an inbound symbol. If the received clock is slower than the internal clock, then it may be necessary to insert an inbound symbol.

This RapidIO 8/16 LP-LVDS interface is source synchronous; therefore, it is guaranteed that a data element will have an associated clock strobe with which to synchronize. A clock boundary is crossed in the receive logic of the end point as the inbound data is synchronized to the internal clock. It must be guaranteed in the end point that a drift between the two clock sources does not cause a setup hold violation resulting in metastability in capturing the data.

To ensure that data is not missed, an end point implements an elasticity buffer. RapidIO uses idle control symbols as the elasticity mechanism. If a receiver needs to skip a symbol during receipt of a large packet, it can issue a throttle control symbol to cause the sender to insert an aligned pacing idle control symbol in the byte stream.

A data beat is clocked into the elasticity buffer with the external clock. The data beat is pulled out of the elasticity buffer using the internal clock delayed by a number of clocks behind the external clock event. This allows the data to become stable before it is synchronized to the internal clock. If the two clock events drift too close together then it is necessary for the synchronization logic to reset the tap and essentially skip a symbol. By guaranteeing a periodic idle control symbol, it is possible for the receive logic to skip a data beat and not miss a critical symbol element.

6 Chapter 6 - Board Routing Guidelines

This chapter contains board design guidelines for RapidIO based systems. The information here is presented as a guide for implementing a RapidIO board design. It is noted that the board designer may have constraints such as standard design practices, vendor selection criteria, and design methodology that must be followed. Therefore appropriate diligence must be applied by the designer.

RapidIO is a source-synchronous differential point-to-point interconnect, so routing considerations are minimal. The very high clock rate places a premium on minimizing skew and discontinuities, such as vias and bends. Generally, layouts should be as straight and free of vias as possible using controlled impedance differential pairs.

6.1 Impedance

Interconnect design should follow standard practice for differential pairs. To minimize reflections from the receiver’s 100 Ohm termination, the differential pair should have an differential impedance of 50 Ohms. The two signals forming the differential pair should be tightly coupled. The differential pairs should be widely spaced, consistent with skew control and quality routing, so that the crosstalk noise is common mode.

6.2 Skew

To minimize the skew on a RapidIO channel the total electrical length for each trace within each unidirectional channel should be equal. Several layouts are suggested in Figure 6-1.

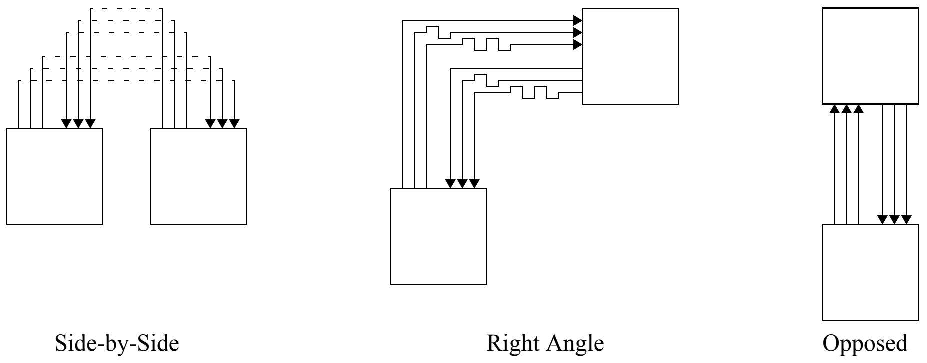


Figure 6-1. Routing for Equalized Skew for Several Placements

Because the RapidIO model is source synchronous, the total length is not critical. Best signal integrity is achieved using a clean layout between opposed parts due to routing on a single layer.

The side-by-side layout requires two routing layers and has reduced signal integrity due to the vias between layers. To keep the total electrical length equal, both layers must have the same phase velocity.

Finally, right angle routing requires meandering to equalize delay, and meandered sections reduce signal integrity while increasing radiation. It may be necessary to place meandered sections on a second routing layer to keep the routing clean.

All skew calculations should be taken to the edge of the package. The package layout and PCB breakout are co-designed to minimize skew, and a recommended PCB breakout is provided.

6.3 PCB Stackup

PCB stackup has a significant effect on EMI generated by the high frequency of operation of a RapidIO channel, so EMI control must be planned from the start. Several stackups are shown in Figure 6-2.

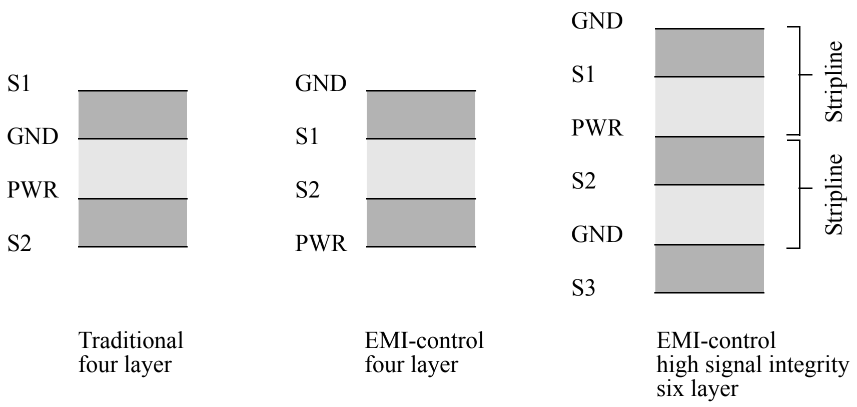


Figure 6-2. Potential PCB Stackups

The traditional four-layer stackup provides equal phase velocities on the two routing layers, but the placement of routing on the outside layers allows for easier radiation. This stackup is suitable for very short interconnects or for applications using an add-on shield.

The four-layer stackup can be rearranged to help with EMI control by placing the power and ground layers on the outside. Each routing layer still has equal phase velocities, but orthogonal routing can degrade signal integrity at very high speeds. The power distribution inductance is approximately tripled due to the larger spacing between the power and ground planes, so applications using this stackup should plan on using more and higher quality bypass capacitance.

The six-layer stackup shows one of many possible stackups. High-speed routing is on S1 and S2 in stripline, so signal quality is excellent with EMI control. S3 is for low-speed signals. Both S1 and S2 have equal phase velocities, good impedance control, and excellent isolation. Power distribution inductance is comparable to the four-layer stackup since the extra GND plane makes up for the extra (2X) spacing between PWR and GND. This example stackup is not balanced with respect to metal loading.

6.4 Termination

RapidIO is source terminated within the driver and differentially terminated within the receiver. No additional termination is needed.

6.5 Additional Considerations

The application environment for a RapidIO channel may place additional constraints on the PCB design.

6.5.1 Single Board Environments

A RapidIO channel completely constructed onto a single board offers the highest performance in terms of clock rate and signal integrity. The primary issues are clean routing with minimal skew. Higher clock rates put greater emphasis on the use of quality sockets (in terms of electrical performance) or on eliminating sockets altogether.

6.5.2 Single Connector Environments

The high clock rate of the 8/16 LP-LVDS physical layer requires the use of an impedance-controlled edge connector. The number of pins dedicated to power should equal the number dedicated to ground, and the distribution of power and ground pins should be comparable. If ground pins greatly outnumber power pins, then bypass capacitors along the length of each side of the connector should be provided. Place the connector as close to one end of the RapidIO interconnect as possible.

6.5.3 Backplane Environments

With two connectors, the design considerations from the single connector environment apply but with greater urgency. The two connectors should either be located as close together or as far apart as possible.

6.6 Recommended pin escape ordering

Given the source-synchronous nature of the 8/16 LP-LVDS physical layer and the clock to data pin skew concern for maximum operating frequency, the recommended bit escape ordering (assuming the device and port orientation shown in Figure 6-1) is shown graphically in Figure 6-3 and Figure 6-4. The figures assume that the device is being viewed from the top. For BGA-style packaged devices the recommended bit escape wire route should be supplied to the board designer. The signal names are defined in Chapter 7, “Signal Descriptions”.

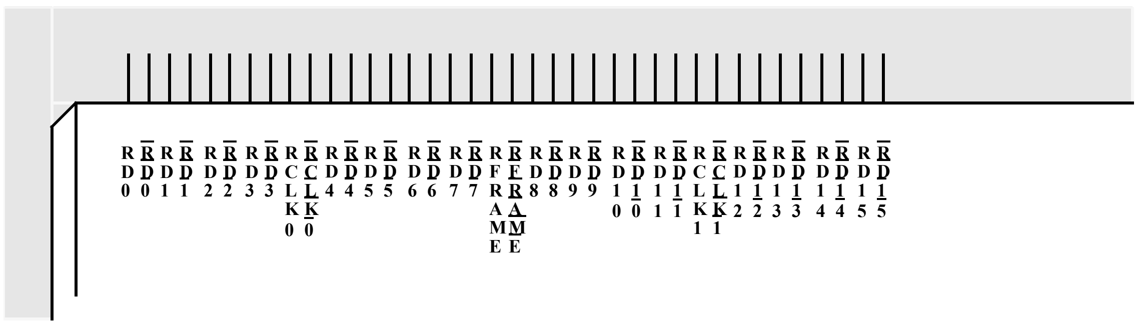


Figure 6-3. Recommended device pin escape, input port, top view of device

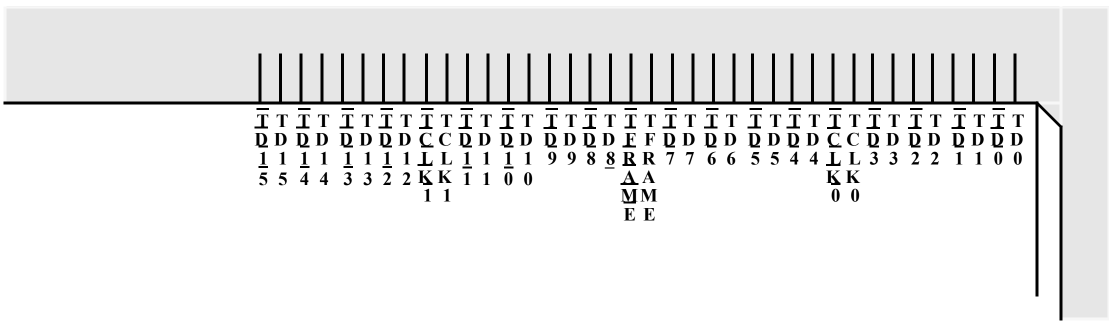


Figure 6-4. Recommended device pin escape, output port, top view of device

These pin escapes allow clean board routes that provide maximum performance connections between two devices as can be seen in the example in Figure 6-5 below.

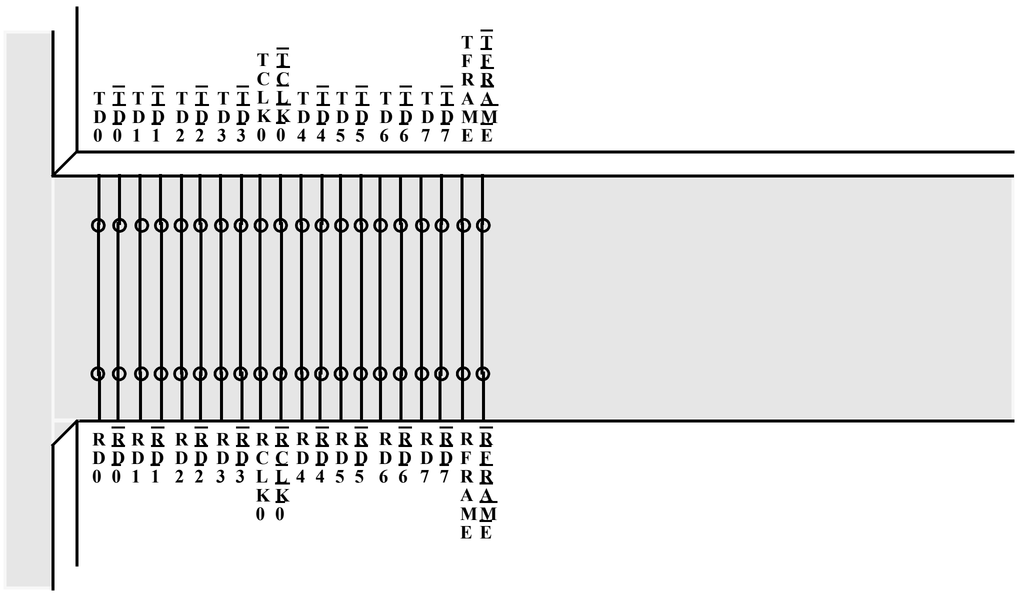


Figure 6-5. Opposed orientation, same side of board

If the attached devices are mounted with certain device orientations the bit wires become crossed. An example of this situation is shown in Figure 6-6. It is permissible for a device to also allow a bit-reversing option on the output (or input) port to support these orientations, as shown in Figure 6-6 and Figure 6-7.

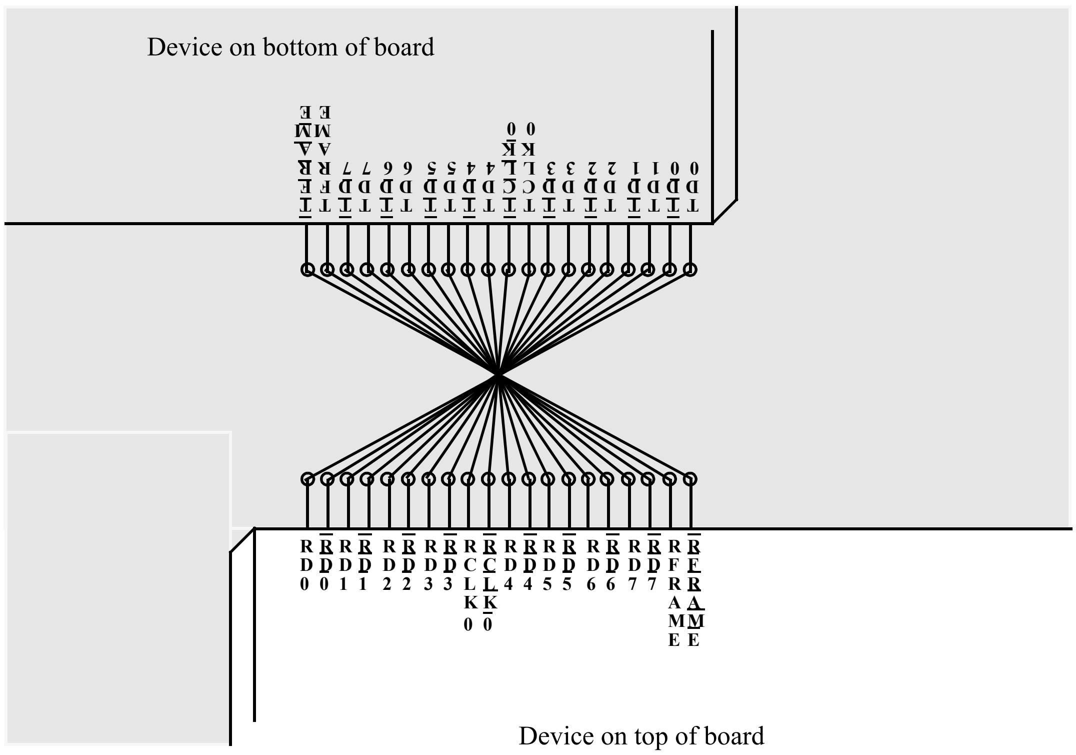


Figure 6-6. Opposed orientation, opposite sides of board

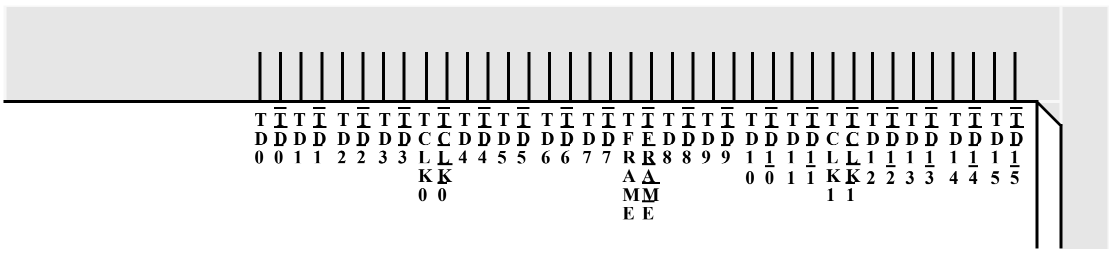


Figure 6-7. Recommended device pin escape, output port reversed, top view of device

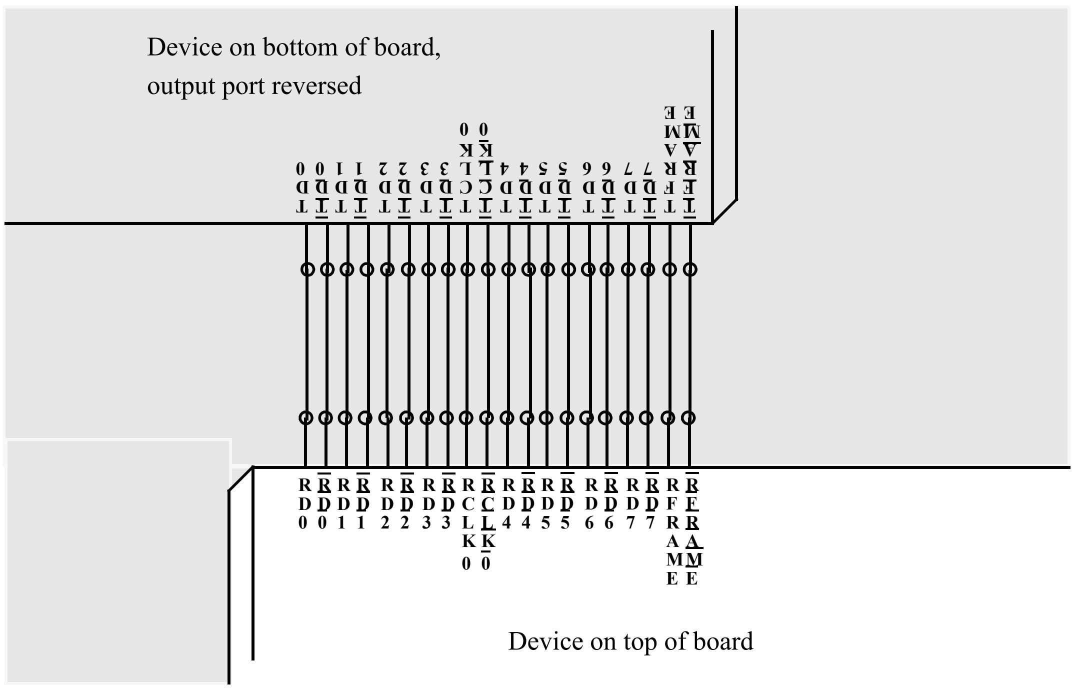

Device on bottom of board,
output port reversed
TD0 TD0 TD1 TD1 TD2 TD2 TD3 TD3 TCLK0 TCLK0 TD4 TD4 TD5 TD5 TD6 TD6 TD7 TD7 TFRAME TFRAME
RD0 RD0 RD1 RD1 RD2 RD2 RD3 RD3 RCLK0 RCLK0 RD4 RD4 RD5 RD5 RD6 RD6 RD7 RD7 RFRAME RFRAME
Device on top of board

Figure 6-8. Opposed orientation, output port reversed, opposite sides of board

7 Chapter 7 - Signal Descriptions

This chapter contains the signal pin descriptions for a RapidIO 8/16 LP-LVDS port. The interface is defined as a parallel 10 bit full duplex point-to-point interface using differential LVDS signaling. The LVDS electrical details are described in Chapter 8, "Electrical Specifications."

7.1 Signal Definitions

Table 7-1 provides a summary of the RapidIO signal pins as well as a short description of their functionality.

Table 7-1. Memory Interface Signal Description

Signal Name	I/O	Signal Meaning	Timing Comments
TCLK0	O	Transmit Clock—Free-running clock for the 8-bit port and the most significant half of the 16-bit port. TCLK0 connects to RCLK0 of the receiving device.	
TCLK0	O	Transmit Clock complement—This signal is the differential pair of the TCLK0 signal.	
TD[0-7]	O	Transmit Data—The transmit data is a unidirectional point to point bus designed to transmit the packet information along with the associated TCLK0 and TFRAME. The TD bus of one device is connected to the RD bus of the receiving device.	Assertion of TD[0-7] is always done with a fixed relationship to TCLK0 as defined in the AC section
TD[0-7]	O	Transmit Data complement—This vector is the differential pair of TD[0-7].	Same as TD

Table 7-1. Memory Interface Signal Description(Continued)

Signal Name	I/O	Signal Meaning	Timing Comments
TFRAME	O	Transmit framing signal—When issued as active this signal indicates a packet control event. TFRAME is connected to RFRAME of the receiving device.	Assertion of TFRAME is always done with a fixed relationship to TCLK0 as defined in the AC section
TFRAME	O	Transmit frame complement—This signal is the differential pair of the TFRAME signal.	Same as TFRAME
TCLK1	O	Transmit Clock—Free-running clock for the least significant half of the 16-bit port (TD[8-15]). TCLK1 connects to RCLK1 of the receiving device. This signal is not used when connected to an 8-bit device.	
TCLK1	O	Transmit Clock complement—This signal is the differential pair of the TCLK1 signal.	
TD[8-15]	O	Transmit Data—least significant half of the 16-bit port. These signals are not used when connected to an 8-bit device.	Assertion of TD[8-15] is always done with a fixed relationship to TCLK0 and TCLK1 as defined in the AC section
TD[8-15]	O	Transmit Data complement—This vector is the differential pair of TD[8-15]	Same as TD[8-15]
RCLK0	I	Receive Clock—Free-running input clock for the 8-bit port and the most significant half of the 16-bit port. RCLK0 connects to TCLK0 of the transmitting device.	
RCLK0	I	Receive Clock complement—This signal is the differential pair of the RCLK signal. $\overline{\text{RCLK0}}$ connects to $\overline{\text{TCLK0}}$ of the transmitting device.	
RD[0-7]	I	Receive Data—The Receive data is a unidirectional packet data input bus. It is connected to the TD bus of the transmitting device.	
RD[0-7]	I	Receive Data complement—This vector is the differential pair of the RD vector.	
RFRAME	I	Receive Frame—This control signal indicates a special packet framing event on the RD pins.	RFRAME is sampled with respect to RCLK0
RFRAME	I	Receive Frame complement—This signal is the differential pair of the RFRAME signal.	Same as RFRAME
RCLK1	I	Receive Clock—Free-running input clock for the least significant half of the 16-bit port (RD[8-15]). RCLK1 connects to TCLK1 of the transmitting device. This signal is not used when connected to an 8-bit device.	
RCLK1	I	Receive Clock complement—This signal is the differential pair of the RCLK1 signal.	

Table 7-1. Memory Interface Signal Description(Continued)

Signal Name	I/O	Signal Meaning	Timing Comments
RD[8-15]	I	Receive Data—Least significant half of the 16-bit port. These signals are not used when connected to an 8-bit device.	
RD[8-15]	I	Receive Data complement—This vector is the differential pair of the RD[8-15] vector.	

7.2 RapidIO Interface Diagrams

Figure 7-1 shows the signal interface diagram connecting two 8-bit devices together with the RapidIO 8/16 LP-LVDS interconnect.

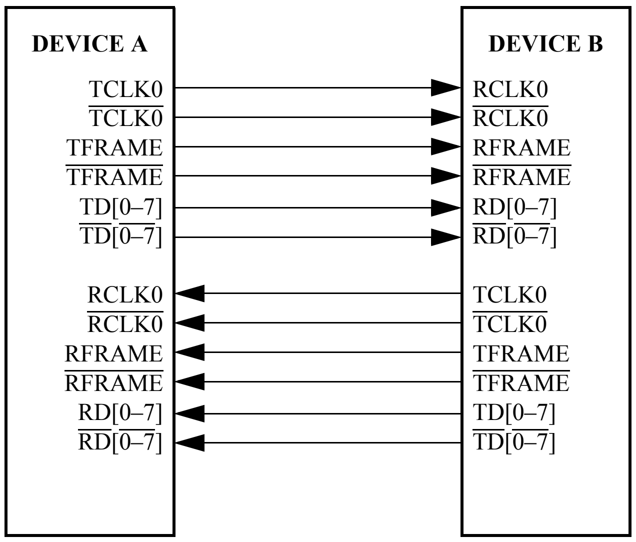


Figure 7-1 RapidIO 8-bit Device to 8-bit Device Interface Diagram

Figure 7-2 shows the connections between an 8-bit wide 8/16 LP-LVDS device and a 16-bit wide device.

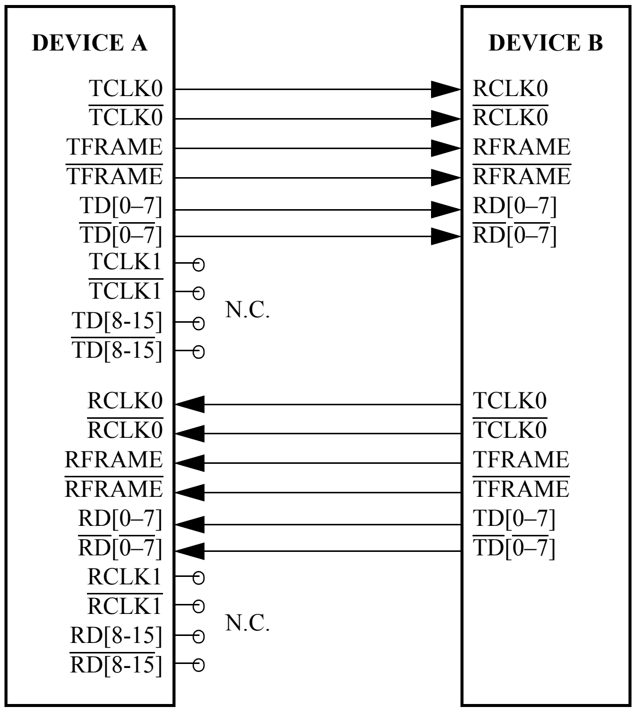

DEVICE A
DEVICE B
TCLK0
$\overline{\text{TCLK0}}$
TFRAME
$\overline{\text{TFRAME}}$
TD[0–7]
$\overline{\text{TD[0–7]}}$
TCLK1
$\overline{\text{TCLK1}}$
TD[8-15]
$\overline{\text{TD[8-15]}}$
N.C.
RCLK0
$\overline{\text{RCLK0}}$
RFRAME
$\overline{\text{RFRAME}}$
RD[0–7]
$\overline{\text{RD[0–7]}}$
RCLK1
$\overline{\text{RCLK1}}$
RD[8-15]
$\overline{\text{RD[8-15]}}$
N.C.

Figure 7-2. RapidIO 8-bit Device to 16-bit Device Interface Diagram

Figure 7-3 shows the connections between two 16-bit wide 8/16 LP-LVDS devices.

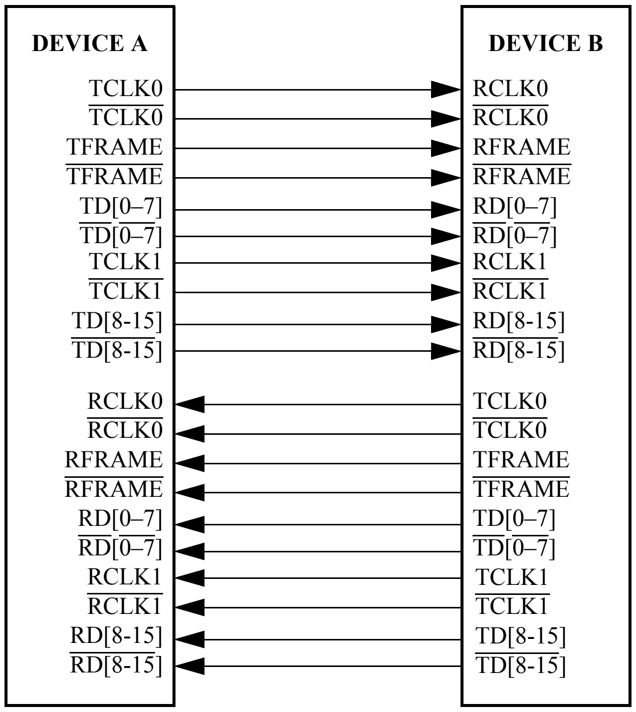

DEVICE A
DEVICE B
TCLK0
$\overline{\text{TCLK0}}$
TFRAME
$\overline{\text{TFRAME}}$
TD[0–7]
$\overline{\text{TD[0–7]}}$
TCLK1
$\overline{\text{TCLK1}}$
TD[8-15]
$\overline{\text{TD[8-15]}}$
RCLK0
$\overline{\text{RCLK0}}$
RFRAME
$\overline{\text{RFRAME}}$
RD[0–7]
$\overline{\text{RD[0–7]}}$
RCLK1
$\overline{\text{RCLK1}}$
RD[8-15]
$\overline{\text{RD[8-15]}}$

Figure 7-3. RapidIO 16-bit Device to 16-bit Device Interface Diagram

8 Chapter 8 - Electrical Specifications

This chapter contains the driver and receiver AC and DC electrical specifications for a RapidIO 8/16 LP-LVDS device. The interface defined is a parallel differential low-power high-speed signal interface.

8.1 Overview

To allow more general compatibility with a variety of silicon solutions, the RapidIO parallel interface builds on the low voltage differential signaling (LVDS) standard. For reference refer to ANSI/TIA/EIA-644-A, *Electrical Characteristics of Low Voltage Differential Signaling (LVDS) Interface Circuits*. The goal of the interface is to allow two devices to communicate with each other within a monolithic system, and key factors in choosing an interface are electrical performance, power consumption (both at the end point and in the switch fabric), signal robustness, circuit complexity, pin count, future scalability, and industry acceptance. LVDS satisfies these requirements.

Although differential signaling requires twice as many signals as single-ended signaling, the total pin count including power and ground pins for high-speed differential and single-ended interfaces are more comparable. Single-ended interfaces require large numbers of power and ground pins to provide a low-impedance AC return path. Since LVDS uses constant-current drivers, a low-impedance AC return path is not needed, allowing for a dramatic reduction in the number of power and ground pins dedicated to the interface. The constant-current drivers also generate very small switching transients leading to lower noise and lower EMI. Differential signaling is also not as susceptible to imperfections in transmission lines and connectors.

LVDS provides for a low-voltage swing (less than 1 Volt), process independent, point-to-point differential interface. The intent of this signaling specification is for device-to-device and board-to-board applications, but it may not be suitable for cable applications owing to the stringent signal-to-signal skew requirements.

LVDS is an end point self-terminated interface. It is assumed that each receiver provides its own termination resistors. LVDS can tolerate ground potential differences between transmitter and receiver of +/- 1V.

8.2 DC Specifications

RapidIO driver and receiver DC specifications are displayed in Table 8-1 and Table 8-2 Power variation is +/- 5%. Resistor tolerances are +/- 1%.

Table 8-1. RapidIO 8/16 LP-LVDS Driver Specifications (DC)

Characteristic	Symbol	Min	Max	Unit	Notes
Differential output high voltage	V_{OHD}	247	454	mV	Bridged 100Ω load See Figure 2-7(a)
Differential output low voltage	V_{OLD}	-454	-247	mV	Bridged 100Ω load See Figure 2-7(a)
Differential offset voltage	ΔV_{OD}		50	mV	Bridged 100Ω load $\lvert V_{OHD}+V_{OLD}\rvert$. See Figure 2-7(b)
Output high common mode voltage	V_{OSH}	1.125	1.375	V	Bridged 100Ω load
Output low common mode voltage	V_{OSL}	1.125	1.375	V	Bridged 100Ω load
Common mode offset voltage	ΔV_{OS}		50	mV	Bridged 100Ω load $\lvert V_{OSH}-V_{OSL}\rvert$. See Figure 2-7(c)
Differential termination	R_{TERM}	90	220	W	
Short circuit current (either output)	$\lvert I_{SS}\rvert$		24	mA	Outputs shorted to V_{DD} or V_{SS}
Bridged short circuit current	$\lvert I_{SB}\rvert$		12	mA	Outputs shorted together

Table 8-2. RapidIO 8/16 LP-LVDS Receiver Specifications (DC)

Characteristic	Symbol	Min	Max	Unit	Notes
Voltage at either input	V_I	0	2.4	V	
Differential input high voltage	V_{IHD}	100	600	mV	Over the common mode range
Differential input low voltage	V_{ILD}	-600	-100	mV	Over the common mode range
Common mode input range (referenced to receiver ground)	V_{IS}	0.050	2.350	V	Limited by V_I
Input differential resistance	R_{IN}	90	110	W	

DC driver signal levels are displayed in Figure 8-1.

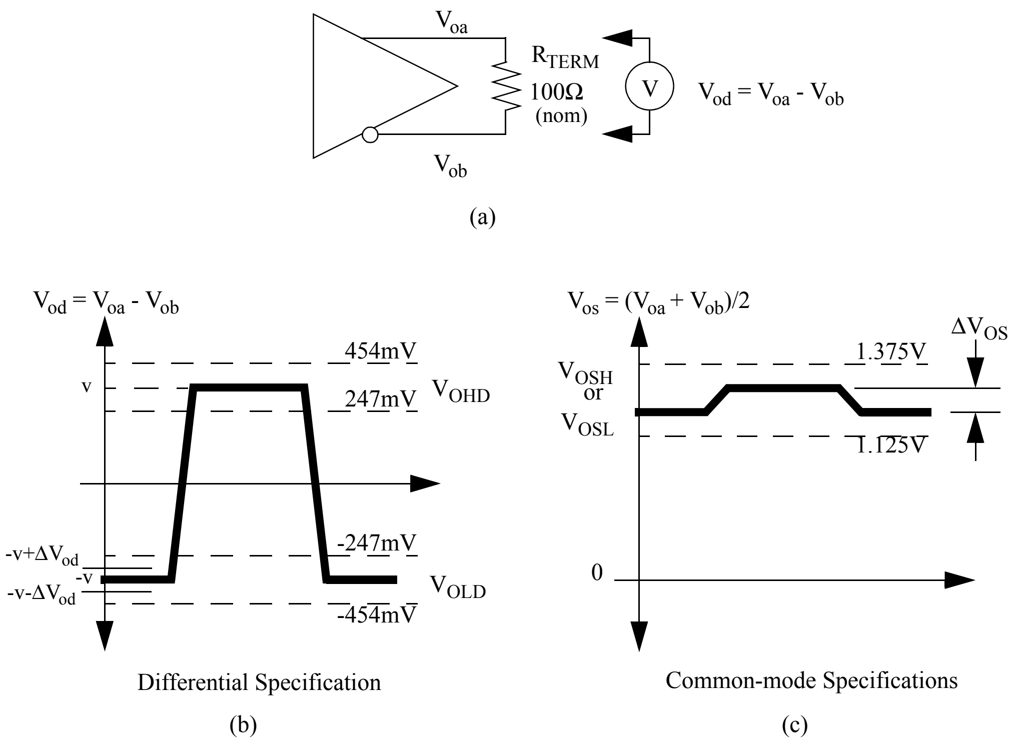


Figure 8-1. DC driver signal levels

8.3 AC Specifications

This section contains the AC electrical specifications for a RapidIO 8/16 LP-LVDS interface. The interface defined is a parallel differential low-power high-speed signal interface. RapidIO specifies operation at specific nominal frequencies only. Correct operation at other frequencies is not implied, even if the frequency is lower than the specified frequency.

8.3.1 Concepts and Definitions

This section specifies signals using differential voltages. Figure shows how the signals are defined. The figure shows waveforms for either a transmitter output (TD and $\overline{TD}$) or a receiver input (RD and $\overline{RD}$). Each signal swings between A volts and B volts where A > B. Using these waveforms, the definitions are as follows:

1) The transmitter output and receiver input signals TD, $\overline{TD}$, RD and $\overline{RD}$ each have a peak-to-peak swing of A-B Volts.

2) The differential output signal of the transmitter, V_{OD}, is defined as V_{TD}-$V_{\overline{TD}}$.

3) The differential input signal of the receiver, V_{ID}, is defined as V_{RD}-$V_{\overline{RD}}$.

4) The differential output signal of the transmitter, or input signal of the receiver, ranges from A - B Volts to -(A - B) Volts.

5) The peak differential signal of the transmitter output, or receiver input, is A - B Volts.

6) The peak to peak differential signal of the transmitter output, or receiver input, is 2*(A - B) Volts.

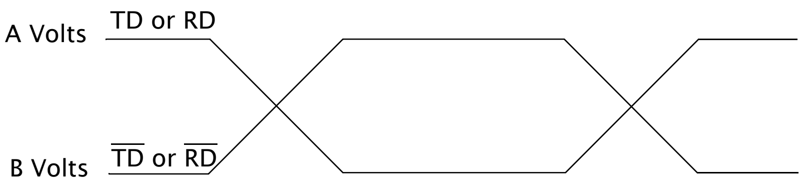


Figure 8-2. Differential Peak-Peak Voltage of Transmitter or Receiver

To illustrate these definitions using numerical values, consider the case where a LVDS transmitter has a common mode voltage of 1.2V and each signal has a swing that goes between 1.4V and 1.0V. Using these values, the peak-to-peak voltage swing of the signals TD, $\overline{TD}$, RD and $\overline{RD}$ is 400 mV. The differential signal ranges between 400mV and -400mV. The peak differential signal is 400mV, and the peak to peak differential signal is 800mV.

A timing edge is the zero-crossing of a differential signal. Each skew timing parameter on a parallel bus is synchronously measured on two signals relative to each other in the same cycle, such as data to data, data to clock, or clock to clock. A skew timing parameter may be relative to the edge of a signal or to the middle of two sequential edges.

Static skew represents the timing difference between signals that does not vary over time regardless of system activity or data pattern. Path length differences are a primary source of static skew.

Dynamic skew represents the amount of timing difference between signals that is dependent on the activity of other signals and varies over time. Crosstalk between signals is a source of dynamic skew.

Eye diagrams and compliance masks are a useful way to visualize and specify driver and receiver performance. This technique is used in several serial bus specifications. An example compliance mask is shown in Figure 8-3. The key difference in the application of this technique for a parallel bus is that the data is source synchronous to its bus clock while serial data is referenced to its embedded clock. Eye diagrams reveal the quality (“cleanness”, “openness”, “goodness”) of a driver output or receiver input. An advantage of using an eye diagram and a compliance mask is that it allows specifying the quality of a signal without requiring separate specifications for effects such as rise time, duty cycle distortion, data dependent dynamic skew, random dynamic skew, etc. This allows the individual semiconductor manufacturer maximum flexibility to trade off various performance criteria while keeping the system performance constant.

In using the eye pattern and compliance mask approach, the quality of the signal is specified by the compliance mask. The mask specifies the maximum permissible magnitude of the signal and the minimum permissible eye opening. The eye diagram for the signal under test is generated according to the specification. Compliance is determined by whether the compliance mask can be positioned over the eye diagram such that the eye pattern falls entirely within the unshaded portion of the mask.

Serial specifications have clock encoded with the data, but the LP-LVDS physical layer defined by RapidIO is a source synchronous parallel port so additional specifications to include effects that are not found in serial links are required. Specifications for the effect of bit to bit timing differences caused by static skew have been added and the eye diagrams specified are measured relative to the associated clock in order to include clock to data effects. With the transmit output (or receiver input) eye diagram, the user can determine if the transmitter output (or receiver input) is compliant with an oscilloscope with the appropriate software.

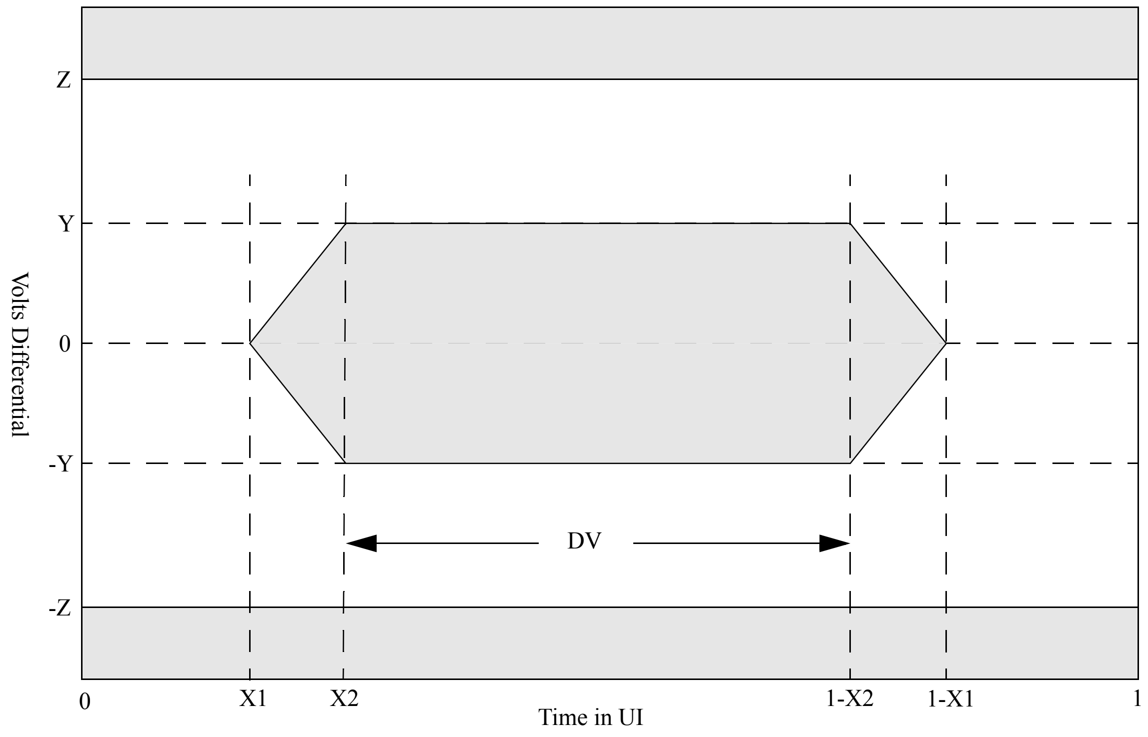


Figure 8-3. Example Compliance Mask

Y = Minimum data valid amplitude

Z = Maximum amplitude

1 UI = 1 Unit Interval = 1/Baud rate

X1 = End of zero crossing region

X2 = Beginning of Data Valid window

DV = Data Valid window = 1 - 2*X2

The waveform of the signal under test must fall within the unshaded area of the mask to be compliant. Different masks are used for the driver output and the receiver input allowing each to be separately specified.

8.3.2 Driver Specifications

Driver AC timing specifications are given in Table 8-3 through Table 8-7 below. A driver shall comply with the specifications for each data rate/frequency for which operation of the driver is specified. Unless otherwise specified, these specifications are subject to the following conditions.

The specifications apply over the supply voltage and ambient temperature ranges specified by the device vendor.

The specifications apply for any combination of data patterns on the data signals.

The output of a driver shall be connected to a 100 Ohm, +/- 1%, differential (bridged) resistive load.

Clock specifications apply only to clock signals (CLK0 and, if present, CLK1).

Data specifications apply only to data signals (FRAME, D[0-7], and, if present, D[8-15]).

FRAME and D[0-7] are the data signals associated with CLK0, D[8-5] are the data signals associated with CLK1.

Table 8-3. Driver AC Timing Specifications - 500Mbps Data Rate/250MHz Clock Rate

Characteristic	Symbol	Range		Unit	Notes
		Min	Max		
Differential output high voltage	V_{OHD}	200	540	mV	See Figure 8-4
Differential output low voltage	V_{OLD}	-540	-200	mV	See Figure 8-4
Unit interval	UI	2000	2000	ps	Requires +/-100ppm long term frequency stability
Duty cycle of the clock output	DC	48	52	%	Measured at V_{OD}=0V
V_{OD} fall time, 20-80% of the peak to peak differential signal swing	t_{FALL}	.1		UI	
V_{OD} rise time, 20-80%of the peak to peak differential signal swing	t_{RISE}	.1		UI	
Data Valid	DV	.63		UI	Measured using the RapidIO Transmit Mask shown in Figure 8-4
Allowable static skew between any two data outputs within a 8 bit/9 bit group	t_{DPAIR}		.09	UI	See Figure 8-10
Allowable static skew of data outputs to associated clock	$t_{SKEW,PAIR}$	-.09	.09	UI	See Figure , Figure 8-10
Clock to clock static skew	$t_{CSKEW, PAIR}$		.09	UI	See Figure 8-9
Clock to clock dynamic skew	$t_{CSKEW, PAIRD}$		.2	UI	See Figure 8-9

Table 8-4. Driver AC Timing Specifications - 750Mbps Data Rate/375MHz Clock Rate (continued)

Characteristic	Symbol	Range		Unit	Notes
		Min	Max		
Differential output high voltage	V_{OHD}	200	540	mV	See Figure 8-4
Differential output low voltage	V_{OLD}	-540	-200	mV	See Figure 8-4
Unit interval		1333	1333	ps	Requires +/-100ppm long term frequency stability
Duty cycle of the clock output	DC	48	52	%	Measured at V_{OD}=0V
V_{OD} fall time, 20-80% of the peak to peak differential signal swing	t_{FALL}	.1		UI	
V_{OD} rise time, 20-80%of the peak to peak differential signal swing	t_{RISE}	.1		UI	

Table 8-4. Driver AC Timing Specifications - 750Mbps Data Rate/375MHz Clock Rate (continued)

Characteristic	Symbol	Range		Unit	Notes
		Min	Max		
Data Valid	DV	.6		UI	Measured using the RapidIO Transmit Mask shown in Figure 8-4
Allowable static skew between any two data outputs within a 8 bit/9 bit group	t_{DPAIR}		.1	UI	See Figure 8-10
Allowable static skew of data outputs to associated clock	$t_{SKEW,PAIR}$	-.1	.1	UI	See Figure 8-8, Figure 8-10
Clock to clock static skew	$t_{CSKEW, PAIR}$		.15	UI	See Figure 8-9
Clock to clock dynamic skew	$t_{CSKEW, PAIRD}$		.2	UI	See Figure 8-9

Table 8-5. Driver AC Timing Specifications - 1000Mbps Data Rate/500MHz Clock Rate

Characteristic	Symbol	Range		Unit	Notes
		Min	Max		
Differential output high voltage	V_{OHD}	200	540	mV	See Figure 8-4
Differential output low voltage	V_{OLD}	-540	-200	mV	See Figure 8-4
Unit interval		1000	1000	ps	Requires +/-100ppm long term frequency stability
Duty cycle of the clock output	DC	48	52	%	Measured at V_{OD}=0V
V_{OD} fall time, 20-80% of the peak to peak differential signal swing	t_{FALL}	.1		UI	
V_{OD} rise time, 20-80%of the peak to peak differential signal swing	t_{RISE}	.1		UI	
Data Valid	DV	.575		UI	Measured using the RapidIO Transmit Mask shown in Figure 8-4
Allowable static skew between any two data outputs within a 8 bit/9 bit group	t_{DPAIR}		.1	UI	See Figure 8-10
Allowable static skew of data outputs to associated clock	$t_{SKEW,PAIR}$	-.1	.1	UI	See Figure ,8-8 Figure 8-10
Clock to clock static skew	$t_{CSKEW, PAIR}$		.15	UI	See Figure 8-9
Clock to clock dynamic skew	$t_{CSKEW, PAIRD}$		.2	UI	See Figure 8-9

Table 8-6. Driver AC Timing Specifications - 1500Mbps Data Rate/750MHz Clock Rate

Characteristic	Symbol	Range		Unit	Notes
		Min	Max		
Differential output high voltage	V_{OHD}	200	540	mV	See Figure 8-4
Differential output low voltage	V_{OLD}	-540	-200	mV	See Figure 8-4
Unit interval		667	667	ps	Requires +/-100ppm long term frequency stability
Duty cycle of the clock output	DC	48	52	%	Measured at V_{OD}=0V
V_{OD} fall time, 20-80% of the peak to peak differential signal swing	t_{FALL}	.1		UI	
V_{OD} rise time, 20-80% of the peak to peak differential signal swing	t_{RISE}	.1		UI	
Data Valid	DV	.525		UI	Measured using the RapidIO Transmit Mask shown in Figure 8-4
Allowable static skew between any two data outputs within a 8 bit/9 bit group	t_{DPAIR}		.2	UI	See Figure 8-10
Allowable static skew of data outputs to associated clock	$t_{SKEW,PAIR}$	-.2	.2	UI	See Figure 8-8, Figure 8-10
Clock to clock static skew	$t_{CSKEW, PAIR}$		.15	UI	See Figure 8-9
Clock to clock dynamic skew	$t_{CSKEW, PAIRD}$		.2	UI	See Figure 8-9

Table 8-7. Driver AC Timing Specifications - 2000Mbps Data Rate/1000MHz Clock Rate (continued)

Characteristic	Symbol	Range		Unit	Notes
		Min	Max		
Differential output high voltage	V_{OHD}	200	540	mV	See Figure 8-4
Differential output low voltage	V_{OLD}	-540	-200	mV	See Figure 8-4
Unit interval		500	500	ps	Requires +/-100ppm long term frequency stability
Duty cycle of the clock output	DC	48	52	%	Measured at V_{OD}=0V
V_{OD} fall time, 20-80% of the peak to peak differential signal swing	t_{FALL}	.1		UI	
V_{OD} rise time, 20-80% of the peak to peak differential signal swing	t_{RISE}	.1		UI	

Table 8-7. Driver AC Timing Specifications - 2000Mbps Data Rate/1000MHz Clock Rate (continued)

Characteristic	Symbol	Range		Unit	Notes
		Min	Max		
Data Valid	DV	.5		UI	Measured using the RapidIO Transmit Mask shown in Figure 8-4
Allowable static skew between any two data outputs within a 8 bit/9 bit group	t_{DPAIR}		.2	UI	See Figure 8-10
Allowable static skew of data outputs to associated clock	$t_{SKEW,PAIR}$	-.2	.2	UI	See Figure 8-8, Figure 8-10
Clock to clock static skew	$t_{CSKEW, PAIR}$		.2	UI	See Figure 8-9
Clock to clock dynamic skew	$t_{CSKEW, PAIRD}$		.2	UI	See Figure 8-9

The compliance of driver output signals TD[0-15] and TFRAME with their minimum Data Valid window (DV) specification shall be determined by generating an eye pattern for each of the data signals and comparing the eye pattern of each data signal with the RapidIO Transmit Mask shown in Figure 8-4. The value of X2 used to construct the mask shall be (1 - DV_{min})/2. A signal is compliant with the Data Valid window specification if and only if the Transmit Mask can be positioned on the signal's eye pattern such that the eye pattern falls entirely within the unshaded portion of the mask.

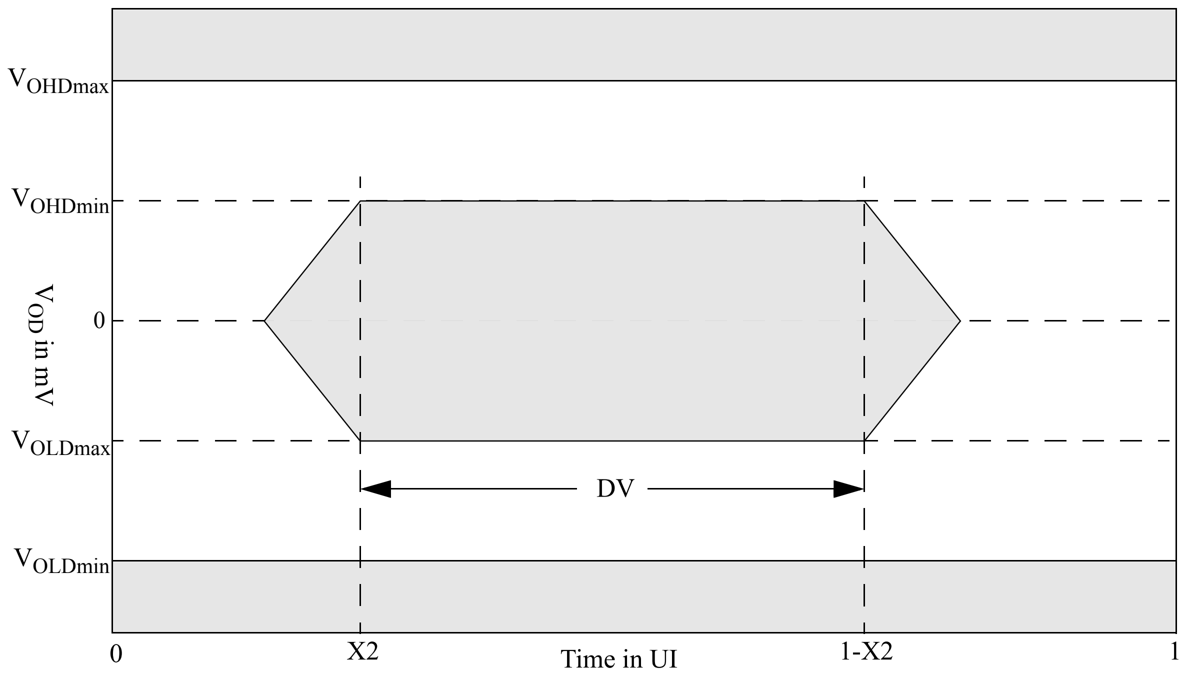


Figure 8-4. RapidIO Transmit Mask

The eye pattern for a data signal is generated by making a large number of recordings of the signal and then overlaying the recordings. The number of recordings used to generate the eye shall be large enough that further increasing the number of recordings used does not cause the resulting eye pattern to change from one that complies with the RapidIO Transmit Mask to one that does not. Each data signal in the interface shall be carrying random or pseudo-random data when the recordings are made. If pseudo-random data is used, the length of the pseudo-random sequence (repeat length) shall be long enough that increasing the length of the sequence does not cause the resulting eye pattern to change from one that complies with the RapidIO Transmit Mask to one that does not comply with the mask. The data carried by any given data signal in the interface may not be correlated with the data carried by any other data signal in the interface. The zero-crossings of the clock associated with a data signal shall be used as the timing reference for aligning the multiple recordings of the data signal when the recordings are overlaid.

While the method used to make the recordings and overlay them to form the eye pattern is not specified, the method used shall be demonstrably equivalent to the following method. The signal under test is repeatedly recorded with a digital oscilloscope in infinite persistence mode. Each recording is triggered by a zero-crossing of the clock associated with the data signal under test. Roughly half of the recordings are triggered by positive-going clock zero-crossings and roughly half are triggered by negative-going clock zero-crossings. Each recording is at least 1.9 UI in length (to ensure that at least one complete eye is formed) and begins 0.5 UI before the trigger point (0.5 UI before the associated clock zero-crossing). Depending on the length of the individual recordings used to generate the eye pattern, one or more complete eyes will be formed. Regardless of the number of eyes, the eye whose center is immediately to the right of the trigger point is the eye used for compliance testing.

An example of an eye pattern generated using the above method with recordings 3 UI in length is shown in Figure 8-5. In this example, there is no skew between the signal under test and the associated clock used to trigger the recordings. If skew was present, the eye pattern would be shifted to the left or right relative to the oscilloscope trigger point.

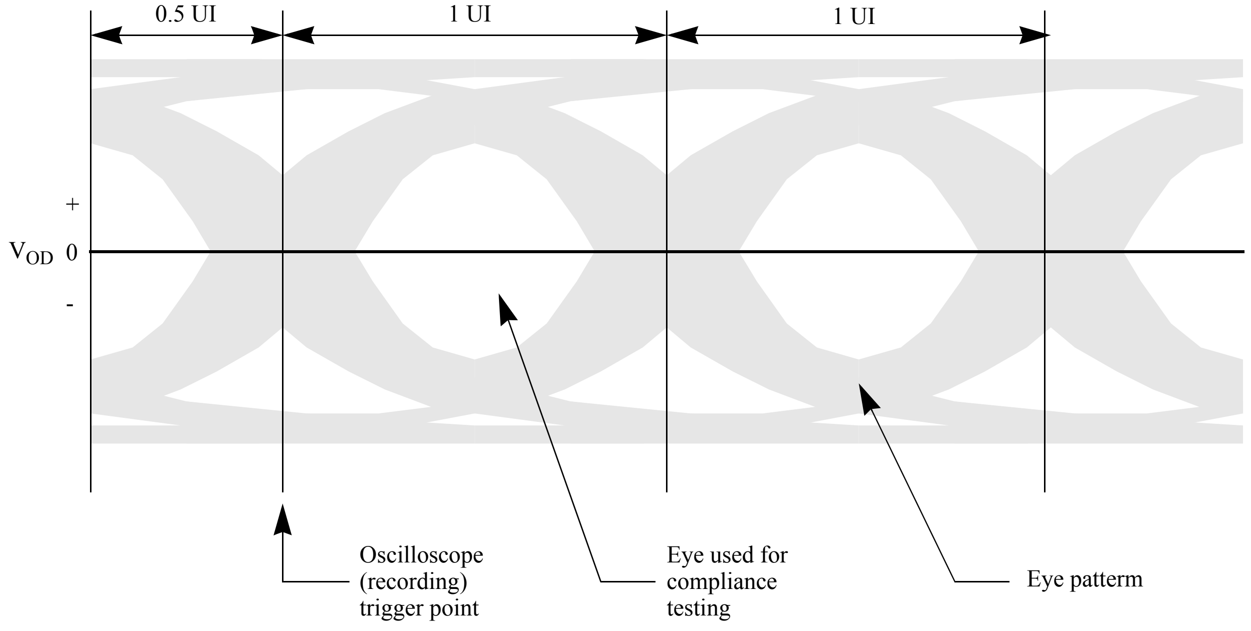

0.5 UI
1 UI
1 UI
+
V_{OD} 0
-
Oscilloscope (recording) trigger point
Eye used for compliance testing
Eye patterm

Figure 8-5. Example Driver Output Eye Pattern

8.3.3 Receiver Specifications

Receiver AC timing specifications are given in Table 8-1 through Table 8-5 below. A receiver shall comply with the specifications for each data rate/frequency for which operation of the receiver is specified. Unless otherwise specified, these specifications are subject to the following conditions.

The specifications apply over the supply voltage and ambient temperature ranges specified by the device vendor.

The specifications apply for any combination of data patterns on the data signals.

The specifications apply over the receiver common mode and differential input voltage ranges.

Clock specifications apply only to clock signals (CLK0 and, if present, CLK1).

Data specifications apply only to data signals (FRAME, D[0-7], and, if present, D[8-15]).

FRAME and D[0-7] are the data signals associated with CLK0, D[8-5] are the data signals associated with CLK1.

Table 8-1. Receiver AC Timing Specifications - 500Mbps Data Rate/250MHz Clock Rate

Characteristic	Symbol	Range		Unit	Notes
		Min	Max		
Duty cycle of the clock input	DC	47	53	%	Measured at V_{ID}=0V
Data Valid	DV	.54		UI	Measured using the RapidIO Receive Mask shown in Figure 8-6
Allowable static skew between any two data inputs within a 8 bit/9 bit group	t_{DPAIR}		.19	UI	See Figure 8-10
Allowable static skew of data inputs to associated clock	$t_{SKEW,PAIR}$	-.15	.15	UI	See Figure 8-8, Figure 8-10
Clock to clock static skew	$t_{CSKEW, PAIR}$		.14	UI	See Figure 8-9
Clock to clock dynamic skew	$t_{CSKEW, PAIRD}$		.3	UI	See Figure 8-9

Table 8-2. Receiver AC Timing Specifications - 750Mbps Data Rate/375MHz Clock Rate

Characteristic	Symbol	Range		Unit	Notes
		Min	Max		
Duty cycle of the clock input	DC	47	53	%	Measured at V_{ID}=0V
Data Valid	DV	.45		UI	Measured using the RapidIO Receive Mask shown in Figure 8-6
Allowable static skew between any two data inputs within a 8 bit/9 bit group	t_{DPAIR}		.3	UI	See Figure 8-10
Allowable static skew of data inputs to associated clock	$t_{SKEW,PAIR}$	-.2	.2	UI	See Figure 8-8, Figure 8-10
Clock to clock static skew	$t_{CSKEW, PAIR}$		.2	UI	See Figure 8-9
Clock to clock dynamic skew	$t_{CSKEW, PAIRD}$		.3	UI	See Figure 8-9

Table 8-3. Receiver AC Timing Specifications - 1000Mbps Data Rate/500MHz Clock Rate (continued)

Characteristic	Symbol	Range		Unit	Notes
		Min	Max		
Duty cycle of the clock input	DC	47	53	%	Measured at V_{ID}=0V
Data Valid	DV	.425		UI	Measured using the RapidIO Receive Mask shown in Figure 8-6

Table 8-3. Receiver AC Timing Specifications - 1000Mbps Data Rate/500MHz Clock Rate (continued)

Characteristic	Symbol	Range		Unit	Notes
		Min	Max		
Allowable static skew between any two data inputs within a 8 bit/9 bit group	t_{DPAIR}		.3	UI	See Figure 8-10
Allowable static skew of data inputs to associated clock	$t_{SKEW,PAIR}$	-.2	.2	UI	See Figure 8-8, Figure 8-10
Clock to clock static skew	$t_{CSKEW, PAIR}$		.2	UI	See Figure 8-9
Clock to clock dynamic skew	$t_{CSKEW, PAIRD}$		.3	UI	See Figure 8-9

Table 8-4. Receiver AC Timing Specifications - 1500Mbps Data Rate/750MHz Clock Rate

Characteristic	Symbol	Range		Unit	Notes
		Min	Max		
Duty cycle of the clock input	DC	47	53	%	Measured at V_{ID}=0V
Data Valid	DV	.375		UI	Measured using the RapidIO Receive Mask shown in Figure 8-6
Allowable static skew between any two data inputs within a 8 bit/9 bit group	t_{DPAIR}		.4	UI	See Figure 8-10
Allowable static skew of data inputs to associated clock	$t_{SKEW,PAIR}$	-.25	.25	UI	See Figure 8-8, Figure 8-10
Clock to clock static skew	$t_{CSKEW, PAIR}$		.3	UI	See Figure 8-9
Clock to clock dynamic skew	$t_{CSKEW, PAIRD}$		.3	UI	See Figure 8-9

Table 8-5. Receiver AC Timing Specifications - 2000Mbps Data Rate/1000MHz Clock Rate (continued)

Characteristic	Symbol	Range		Unit	Notes
		Min	Max		
Duty cycle of the clock input	DC	47	53	%	Measured at V_{ID}=0V
Data Valid	DV	.35		UI	Measured using the RapidIO Receive Mask shown in Figure 8-6
Allowable static skew between any two data inputs within a 8 bit/9 bit group	t_{DPAIR}		.4	UI	See Figure 8-10
Allowable static skew of data inputs to associated clock	$t_{SKEW,PAIR}$	-.25	.25	UI	See Figure 8-8, Figure 8-10

Table 8-5. Receiver AC Timing Specifications - 2000Mbps Data Rate/1000MHz Clock Rate (continued)

Characteristic	Symbol	Range		Unit	Notes
		Min	Max		
Clock to clock static skew	$t_{CSKEW, PAIR}$		.3	UI	See Figure 8-9
Clock to clock dynamic skew	$t_{CSKEW, PAIRD}$		.3	UI	See Figure 8-9

The compliance of receiver input signals RD[0-15] and RFRAME with their minimum Data Valid window (DV) specification shall be determined by generating an eye pattern for each of the data signals and comparing the eye pattern of each data signal with the RapidIO Receive Mask shown in Figure . The value of X2 used to construct the mask shall be $(1 - DV_{min})/2$. The +/- 100mV minimum data valid and +/- 600mV maximum input voltage values are from the DC specification. A signal is compliant with the Data Valid window specification if and only if the Receive Mask can be positioned on the signal's eye pattern such that the eye pattern falls entirely within the unshaded portion of the mask.

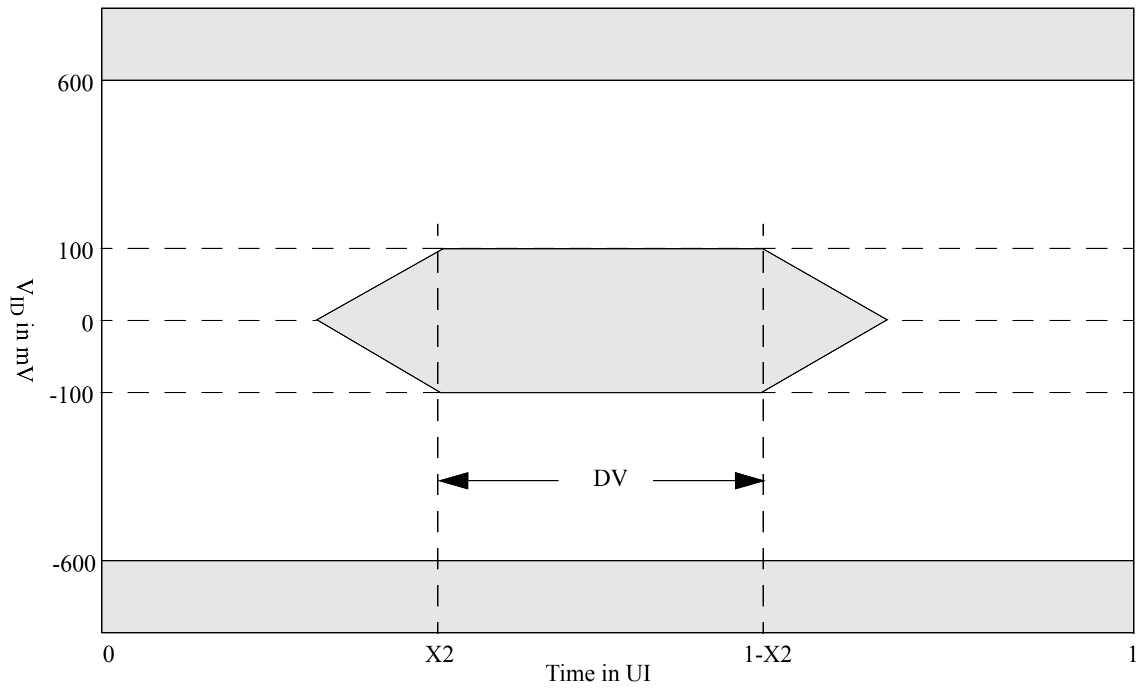


Figure 8-6. RapidIO Receive Mask

The eye pattern for a data signal is generated by making a large number of recordings of the signal and then overlaying the recordings. The number of recordings used to generate the eye shall be large enough that further increasing the number of recordings used does not cause the resulting eye pattern to change from one that complies with the RapidIO Receive Mask to one that does not. Each data signal in the interface shall be carrying random or pseudo-random data when the recordings are made. If pseudo-random data is used, the length of the pseudo-random sequence (repeat length) shall be long enough that increasing the length of the sequence does not cause the resulting eye pattern to change from one that complies with the RapidIO Receive Mask to one that does not comply with the mask. The data carried by any given data signal in the interface may not be correlated with the data carried by any other data signal in the interface. The zero-crossings of the clock associated with a data signal shall be used as the timing reference for aligning the multiple recordings of the data signal when the recordings are overlaid.

While the method used to make the recordings and overlay them to form the eye pattern is not specified, the method used shall be demonstrably equivalent to the following method. The signal under test is repeatedly recorded with a digital oscilloscope in infinite persistence mode. Each recording is triggered by a zero-crossing of the clock associated with the data signal under test. Roughly half of the recordings are triggered by positive-going clock zero-crossings and roughly half are triggered by negative-going clock zero-crossings. Each recording is at least 1.9 UI in length (to ensure that at least one complete eye is formed) and begins 0.5 UI before the trigger point (0.5 UI before the associated clock zero-crossing). Depending on the length of the individual recordings used to generate the eye pattern, one or more complete eyes will be formed. Regardless of the number of eyes, the eye whose center is immediately to the right of the trigger point is the eye used for compliance testing.

An example of an eye pattern generated using the above method with recordings 3 UI in length is shown in Figure 8-7. In this example, there is no skew between the signal under test and the associated clock used to trigger the recordings. If skew was present, the eye pattern would be shifted to the left or right relative to the oscilloscope trigger point.

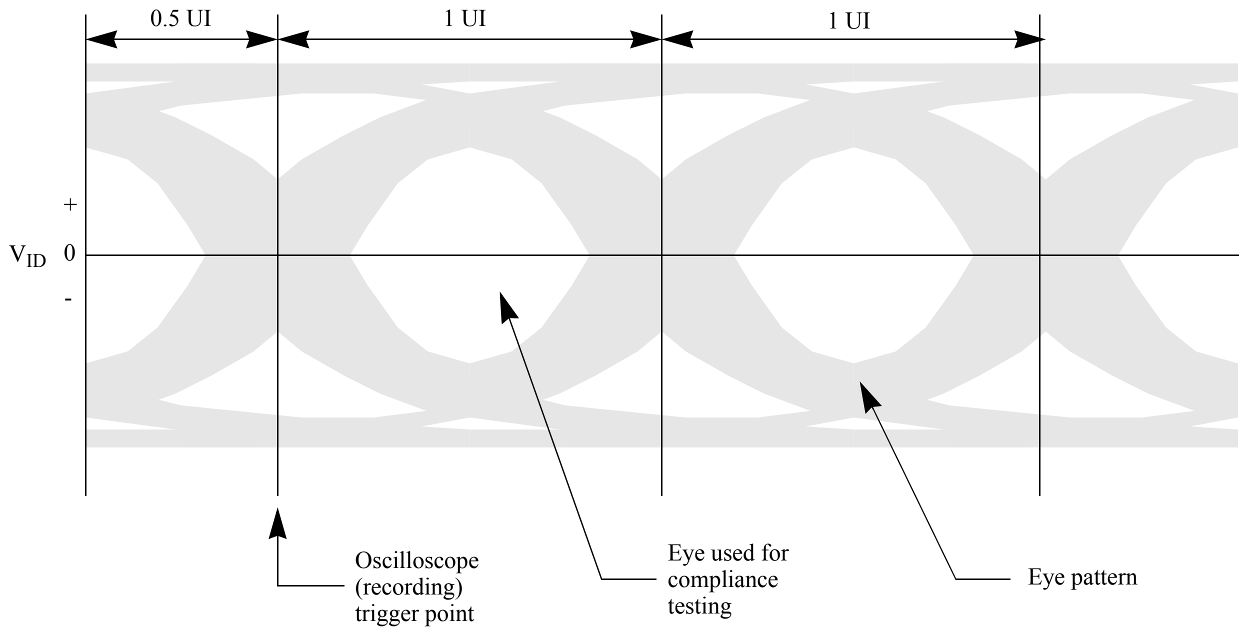

0.5 UI
1 UI
1 UI
+
V_{ID} 0
-
Oscilloscope (recording) trigger point
Eye used for compliance testing
Eye pattern

Figure 8-7. Example Receiver Input Eye Pattern

Figure 8-8 shows the definitions of the data to clock static skew parameter $t_{SKEW,PAIR}$ and the Data Valid window parameter DV. The data and frame bits are those that are associated with the clock. The figure applies for all zero-crossings of the clock. All of the signals are differential signals. V_D represents V_{OD} for the transmitter and V_{ID} for the receiver. The center of the eye is defined as the midpoint of the region in which the magnitude of the signal voltage is greater than or equal to the minimum DV voltage.

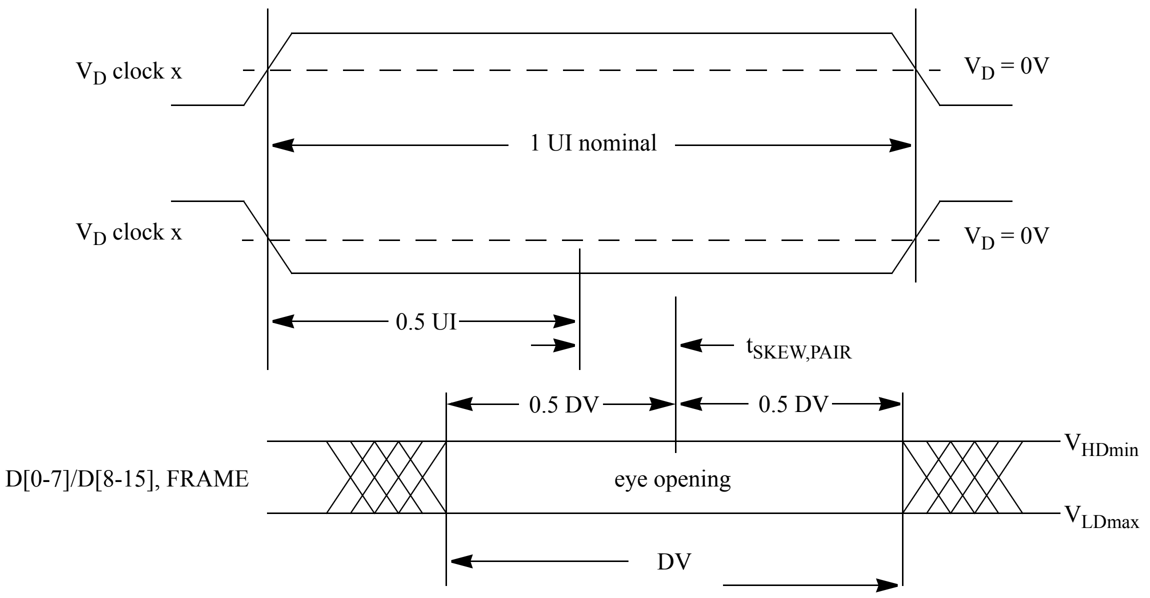


Figure 8-8. Data to Clock Skew

Figure 8-10 shows the definitions of the clock to clock static skew parameter $t_{CSKEW,\ PAIR}$ and the clock to clock dynamic skew parameter $t_{CSKEW,\ PAIRD}$. All of the signals shown are differential signals. V_D represents V_{OD} for the transmitter and V_{ID} for the receiver. These two parameters, $t_{CSKEW,\ PAIR}$ and $t_{CSKEW,\ PAIRD}$, only apply to 16 bit interfaces.

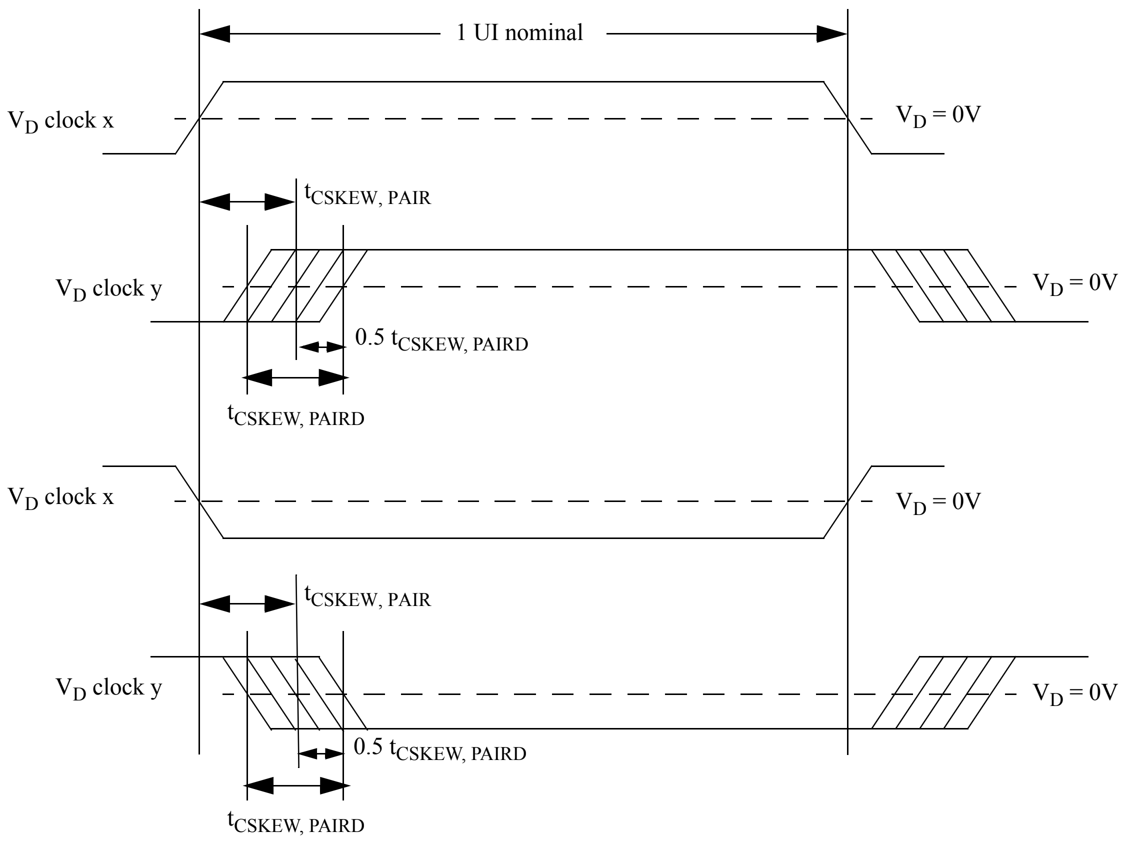


Figure 8-9. Clock to Clock Skew

Figure 8-10 shows the definition of the data to data static skew parameter t_{DPAIR} and how the skew parameters are applied.

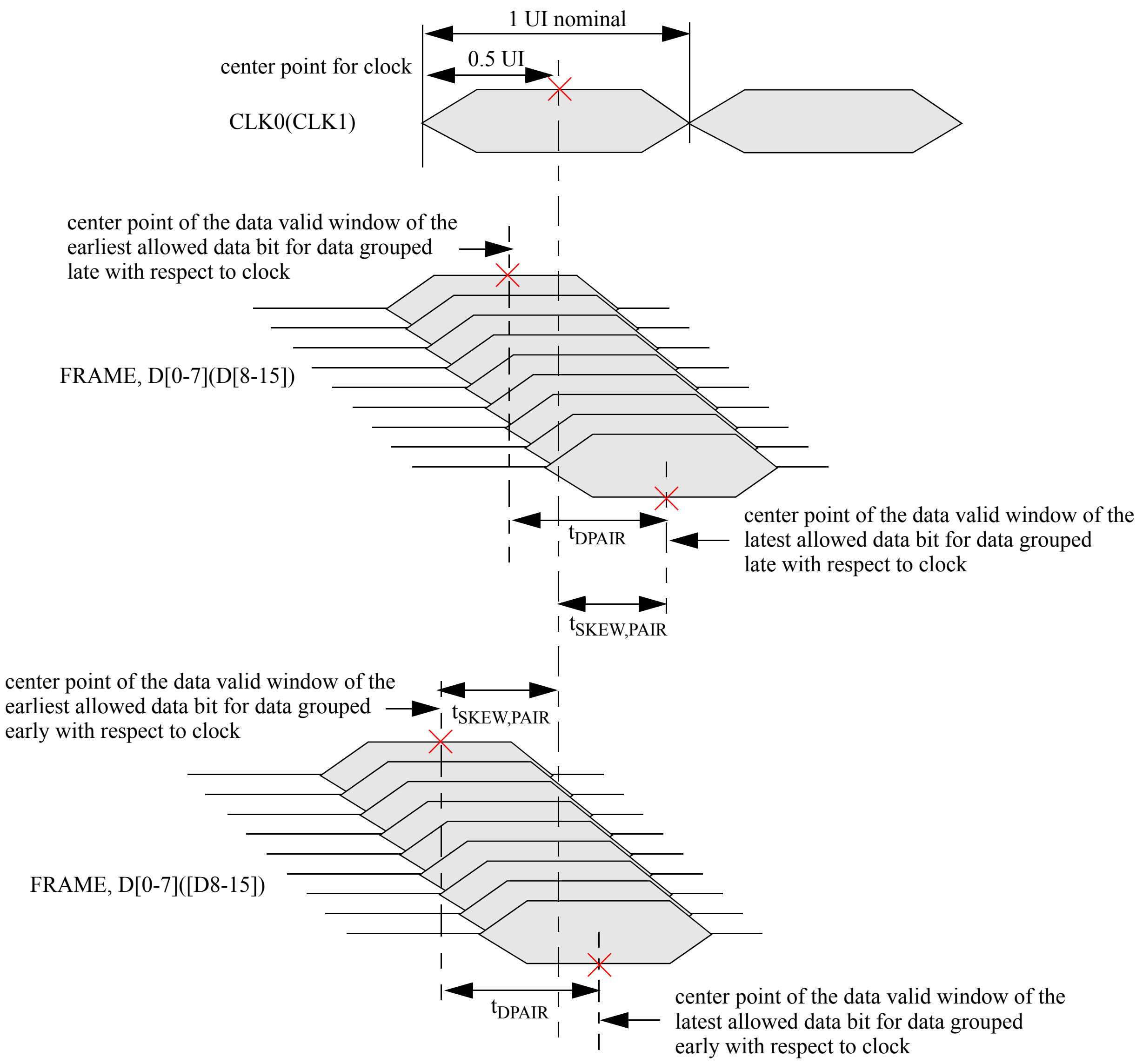

1 UI nominal
0.5 UI
center point for clock
CLK0(CLK1)
center point of the data valid window of the earliest allowed data bit for data grouped late with respect to clock
FRAME, D[0-7](D[8-15])
center point of the data valid window of the latest allowed data bit for data grouped late with respect to clock
tDPAIR
tSKEW,PAIR
center point of the data valid window of the earliest allowed data bit for data grouped early with respect to clock
tSKEW,PAIR
FRAME, D[0-7]([D8-15])
tDPAIR
center point of the data valid window of the latest allowed data bit for data grouped early with respect to clock

Figure 8-10. Static Skew Diagram

Annex A

Message Passing Interface

The RapidIO Message Passing Logical Specification defines several packet formats that are useful for sending messages from a source device to a destination. These formats do not describe a specific programming model but are instantiated as an example packetizing mechanism. Because the actual programming models for message passing can vary greatly in both capability and complexity, they have been deemed beyond the scope of the RapidIO Logical Message Passing Specification. This annex is provided as a reference model for message passing and is not intended to be all encompassing.

A.1 Definitions and Goals

A system may be made up of several processors and distributed memory elements. These processors may be tightly coupled and operating under a monolithic operating system in certain applications. When this is true the operating system is tasked with managing the pool of processors and memory to solve a set of tasks. In most of these cases, it is most efficient for the processors to work out of a common hardware-maintained coherent memory space. This allows processors to communicate initialization and completion of tasks through the use of semaphores, spin locks, and inter-process interrupts. Memory is managed centrally by the operating system with a paging protection scheme.

In other such distributed systems, processors and memory may be more loosely coupled. Several operating systems or kernels may be coexistent in the system, each kernel being responsible for a small part of the entire system. It is necessary to have a communication mechanism whereby kernels can communicate with other kernels in a system of this nature. Since this is a shared nothing environment, it is also desirable to have a common hardware and software interface mechanism to accomplish this communication. This model is typically called message passing.

In these message passing systems, two mechanisms typically are used to move data from one portion of memory space to another. The first mechanism is called direct memory access (DMA), the second is messaging. The primary difference between the two models is that DMA transactions are steered by the source whereas messages are steered by the target. This means that a DMA source not only requires access to a target but must also have visibility into the target’s address space. The message source only requires access to the target and does not need visibility into the target’s address space. In distributed systems it is common to find a mix of DMA and messaging deployed.

The RapidIO architecture contains a packet transport mechanism that can aid in the distributed shared nothing environment. The RapidIO message passing model meets several goals:

- A message is constructed of one or more transactions that can be sent and received through a possibly unordered interconnect
- A sender can have a number of outstanding messages queued for sending
- A sender can send a higher priority message before a lower priority message and can also preempt a lower priority message to send a higher priority one and have the lower priority message resume when the higher is complete (prioritized concurrency)
- A sender requires no knowledge of the receiver’s internal structure or memory map
- A receiver of a message has complete control over it’s local address space
- A receiver can have a number of outstanding messages queued for servicing if desired
- A receiver can receive a number of concurrent multiple-transaction messages if desired

A.2 Message Operations

The *RapidIO Message Passing Logical Specification* defines the type 11 packet as the MESSAGE transaction format. The transaction may be used in a number of different ways dependent on the specific system architecture. The transaction header contains the following field definitions:

For a detailed description of the message packet format, refer to Section 3.1.5, "Type 11 Packet Format (Message Class)."

mbox	Specifies the recipient mailbox in the target processing element. RapidIO allows up to four mailbox ports in each target device. This can be useful for defining blocks of different message frame sizes or different local delivery priority levels.
letter	A RapidIO message operation may be made up of several transactions. It may be desirable in some systems to have more than one multi-transaction message concurrently in transit to the target mailbox. The letter identifies the specific message within the mailbox. This field allows a sending of up to four messages to the same mailbox in the same target device.
multi-transaction fields	In cases where message operations are made up of multiple transactions, the following fields allow reconstruction of a message transported through an unordered interconnect fabric:
msglen	Specifies the total number of transactions comprising this message. A value of 0 indicates a single transaction message. A value of 15 (0xF) indicates a 16 transaction message, and so forth.
msgseg	Specifies the part of the message operation supplied by this transaction. A value of 0 indicates that this is the first transaction in the message. A value of 15 (0xF) indicates that this is the sixteenth transaction in the message, and so on.
ssize	Standard message transaction data size. This field tells the receiver to expect a message the size of the data field for all of the transactions except the last one. This prevents the sender from having to pad the data field excessively for the last transaction and allows the receiver to properly put the message in local memory; otherwise, if the last transaction is the first one received, the address calculations will be in error when writing the transaction to memory.

The second type of message packet is the type 10 doorbell transaction packet. The doorbell transaction is a lightweight transaction that contains only a 16-bit information field that is completely software defined. The doorbell is intended to be an in-band mechanism to send interrupts between processors. In this usage the information field would be used to convey interrupt level and target information to the recipient. For a more detailed description of the doorbell packet format, refer to Section 3.1.4, "Type 10 Packet Formats (Doorbell Class)."

There are two transaction format models described in this annex, a simple model and an extended model. The simple model is recommended for both the type 10 (doorbell) and type 11 (message) packet format messages. The extended model is only recommended for the type 11 (message) packet format messages.

A.3 Inbound Mailbox Structure

RapidIO provides two message transaction packet formats. By nature of having such formats it is possible for one device to pass a message to another device without a specific memory mapped transaction. The transaction allows for the concept of a memory map independent port. As mentioned earlier, how the transactions are generated and what is done with them at the destination is beyond the scope of the *RapidIO Message Passing Logical Specification*. There are, however, a few examples as to how they could be deployed. First, look at the destination of the message.

A.3.1 Simple Inbox

Probably the most simple inbound mailbox structure is that of a single-register port or direct map into local memory space (see figure A-1).

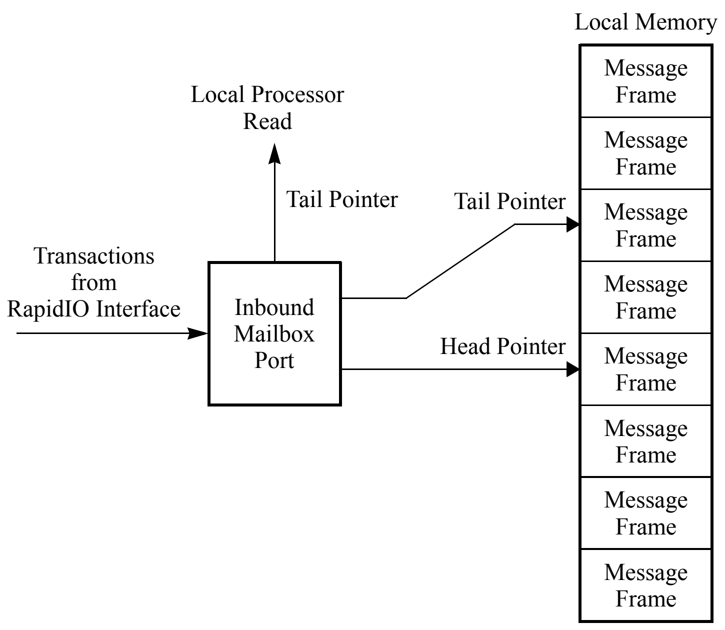


Figure A-1. Simple Inbound Mailbox Port Structure

In this structure, the inbound single transaction message is posted to either a register, set of registers, or circular queue in local memory. In the case of the circular queue, hardware maintains a head and tail pointer that points at a fixed window of pre-partitioned message frames in memory. Whenever the head pointer equals the tail pointer, no more messages can be accepted and they are retried on the RapidIO interface. When messages are posted, the local processor is interrupted. The interrupt service routine reads the mailbox port that contains the message located at the tail pointer. The message frame is equal to the largest message operation that can be received.

The RapidIO MESSAGE transaction allows up to four such inbound mailbox ports per target address. The DOORBELL transaction is defined as a single mailbox port.

A.3.2 Extended Inbox

A second more extensible structure similar to that used in the intelligent I/O (I_2O) specification, but managed differently, also works for the receiver (see figure A-2).

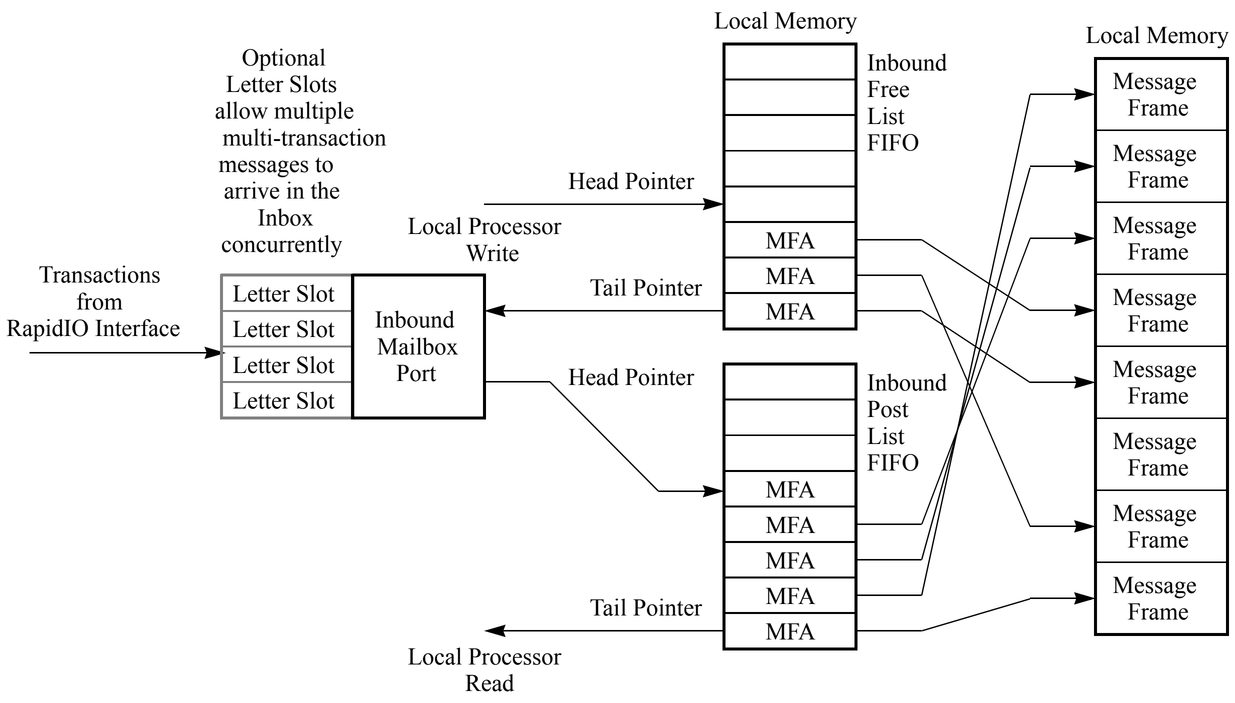


Figure A-2 Inbound Mailbox Structure

One of these structures is required for each priority level supported in an implementation. There are inbound post and free list FIFOs which function as circular queues of a fixed size. The message frames are of a size equal to the maximum message size that can be accepted by the receiver. Smaller messages can be accepted if allowed by the overlaying software. The sender only specifies the mailbox and does not request the frame pointer and perform direct memory access as with I_2O, although the I_2O model can be supported in software with this structure. All pointers are managed by the inbound hardware and the local processor. Message priority and letter number are managed by software.

The advantage of the extended structure is that it allows local software to service message frames in any order. It also allows memory regions to be moved in and out of the message structure instead of forcing software to copy the message to a different memory location.

A.3.3 Received Messages

When a message transaction is received, the inbound mailbox port takes the message frame address (MFA) pointed at by the inbound free list tail pointer and increments that pointer (this may cause a memory read to prefetch the next MFA), effectively taking the MFA from the free list. Subsequent message transactions from a different sender or with a different letter number are now retried until all of the transactions for this message operation have been received, unless there is additional hardware to handle multiple concurrent message operations for the same mailbox, differentiated by the letter slots.

The inbound mailbox port uses the MFA to write the transaction data into local memory at that base address with the exact address calculated as described in Section 1.2.1, “Data Message Operations” and Section 2.2.2, “Data Message Operations.” When the entire message is received and written into memory, the inbound post list pointer is incremented and the MFA is written into that location. If the queue was previously empty, an interrupt is generated to the local processor to indicate that there is a new message pending. This causes a window where the letter hardware is busy and cannot service a new operation between the receipt of the final transaction and the MFA being committed to the local memory.

When the local processor services a received message, it reads the MFA indicated by the inbound post FIFO tail pointer and increments the tail pointer. When the message has been processed (or possibly deferred), it puts a new MFA in the memory address indicated by the inbound free list head pointer and increments that pointer, adding the new MFA to the free list for use by the inbound message hardware.

If the free list head and tail pointer are the same, the FIFO is empty and there are no more MFAs available and all new messages are retried. If the post list head and tail pointers are the same, there are no outstanding messages awaiting service from the local processor. Underflow conditions are fatal since they indicate improper system behavior. This information can be part of an associated status register.

A.4 Outbound Message Queue Structure

Queuing messages in RapidIO is accomplished either through a simple or a more extended outbox.

A.4.1 Simple Outbox

Generation of a message can be as simple as writing to a memory-mapped descriptor structure either in local registers or memory. The outbound message queue (see Figure A-3) looks similar to the inbox.

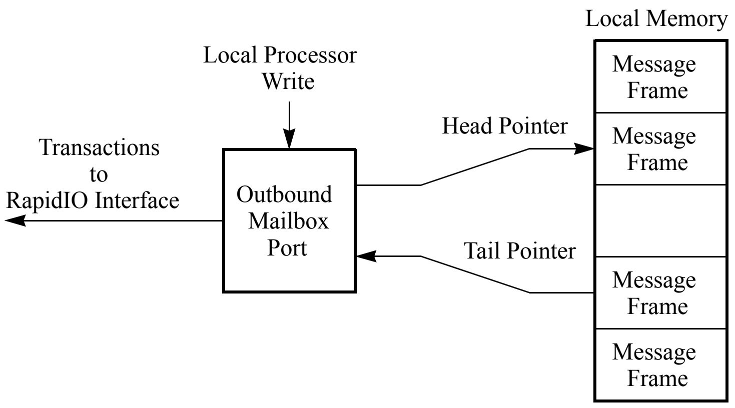


Figure A-3. Outbound Message Queue

The local processor reads a port in the outbound mailbox to obtain the position of a head pointer in local memory. If the read results in a pre-determined pattern the message queue is full. The processor then writes a descriptor structure and message to that location. When it is done, it writes the message port to advance the head point and mark the message as queued. The outbound mailbox hardware then reads the messages pointed to by the tail pointer and transfers them to the target device pointed at by the message descriptor.

One of these structures is required for each priority level of outbound messages supported.

A.4.2 Extended Outbox

A more extensible method of queueing messages is again a two-level approach (see Figure A-4). Multiple structures are required if concurrent operation is desired in an implementation. The FIFO is a circular queue of some fixed size. The message frames are of a size that is equal to the maximum message operation size that can be accepted by the receivers in the system. Smaller message operations can be sent if allowed by the hardware and the overlaying software. As with the receive side, the outbound slots can be virtual and any letter number can be handled by an arbitrary letter slot.

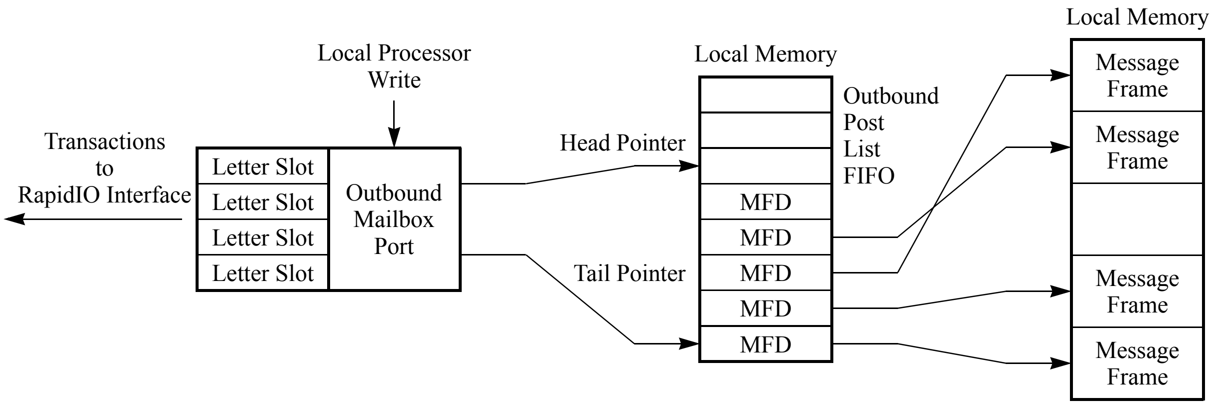


Figure A-4. Extended Outbound Message Queue

When the local processor wishes to send a message, it stores the message in local memory, writes the message frame descriptor (MFD) to the outbound mailbox port (which in-turn writes it to the location indicated by the outbound post FIFO head pointer), and increments the head pointer.

The advantage of this method is that software can have pre-set messages stored in local memory. Whenever it needs to communicate an event to a specific end point it writes the address of the message frame to the outbound mailbox, and the outbound mailbox generates the message transactions and completes the operation.

If the outbound post list FIFO head and tail pointers are not equal, there is a message waiting to be sent. This causes the outbound mailbox port to read the MFD pointed to by the outbound post list tail pointer and then decrement the pointer (this may cause a memory read to prefetch the next MFD). The hardware then uses the information stored in the MFD to read the message frame, packetize it, and transmit it to the receiver. Multiple messages can be transmitted concurrently if there is hardware to support them, differentiated by the letter slots in Figure A-4.

If the free list head and tail pointer are the same, the FIFO is empty and there are no more MFDs to be processed. Underflow conditions are fatal because they indicate improper system behavior. This information can also be part of a status register.

Because the outbound and inbound hardware are independent entities, it is possible for more complex outbound mailboxes to communicate with less complex inboxes by simply reducing the complexity of the message descriptor to match. Likewise simple outboxes can communicate with complex inboxes. Software can determine the capabilities of a device during initial system setup. The capabilities of a devices message hardware are stored in the port configuration registers.

Partition V: Globally Shared Memory Logical Specification

V Partition V

1 Chapter 1 - Globally Shared Memory

This chapter provides an overview of the *RapidIO™ Interconnect Globally Shared Memory Logical Specification*, including a description of the relationship between the GSM specifications and the other specifications of the RapidIO interconnect.

1.1 Overview

Although RapidIO is targeted toward the message passing programming model, it supports a globally shared distributed memory (GSM) model as defined by this specification. The globally shared memory programming model is the preferred programming model for modern general-purpose multiprocessing computer systems, which requires cache coherency support in hardware. This addition of GSM enables both distributed I/O processing and general purpose multiprocessing to co-exist under the same protocol.

The *RapidIO Interconnect Globally Shared Memory Logical Specification* is one of the RapidIO logical layer specifications that define the interconnect's overall protocol and packet formats. This layer contains the information necessary for end points to process a transaction. Other RapidIO logical layer specifications include *Partition I: Input/Output Logical Specification* and *Partition II: Message Passing Logical Specification* of the *RapidIO Interconnect Specification.*

The logical specifications do not imply a specific transport or physical interface, therefore they are specified in a bit stream format. Necessary bits are added to the logical encodings for the transport and physical layers lower in the specification hierarchy.

RapidIO is a definition of a system interconnect. System concepts such as processor programming models, memory coherency models and caching are beyond the scope of the RapidIO architecture. The support of memory coherency models, through caches, memory directories (or equivalent, to hold state and speed up remote memory access) is the responsibility of the end points (processors, memory, and possibly I/O devices), using RapidIO operations. RapidIO provides the operations to construct a wide variety of systems, based on programming models that range from strong consistency through total store ordering to weak ordering. Inter-operability between end points supporting different coherency/caching/directory models is not guaranteed. However, groups of end-points with conforming models can be linked to others conforming to different models on the same RapidIO fabric. These different groups can communicate through RapidIO messaging or I/O operations. Any reference to these areas within the RapidIO architecture specification are for illustration only.

The *RapidIO Interconnect Globally Shared Memory Logical Specification* assumes that the reader is familiar with the concepts and terminology of cache coherent systems in general and with CC-NUMA systems in specific. Further information on shared memory concepts can be found in:

Daniel E. Lenoski and Wolf-Dietrich Weber, “Scalable Shared-Memory Multiprocessing”, Morgan Kaufmann, 1995.

and

David Culler, Jaswinder Pal Singh, and Anoop Gupta: “Parallel Computer Architecture: A Hardware/Software Approach”, Morgan Kaufmann, 1998

1.1.1 Memory System

Under the globally shared distributed memory programming model, memory may be physically located in different places in the machine yet may be shared amongst different processing elements. Typically, mainstream system architectures have addressed shared memory using transaction broadcasts sometimes known as bus-based snoopy protocols. These are usually implemented through a centralized memory controller for which all devices have equal or uniform access. Figure 1-1 shows a typical bus-based shared memory system.

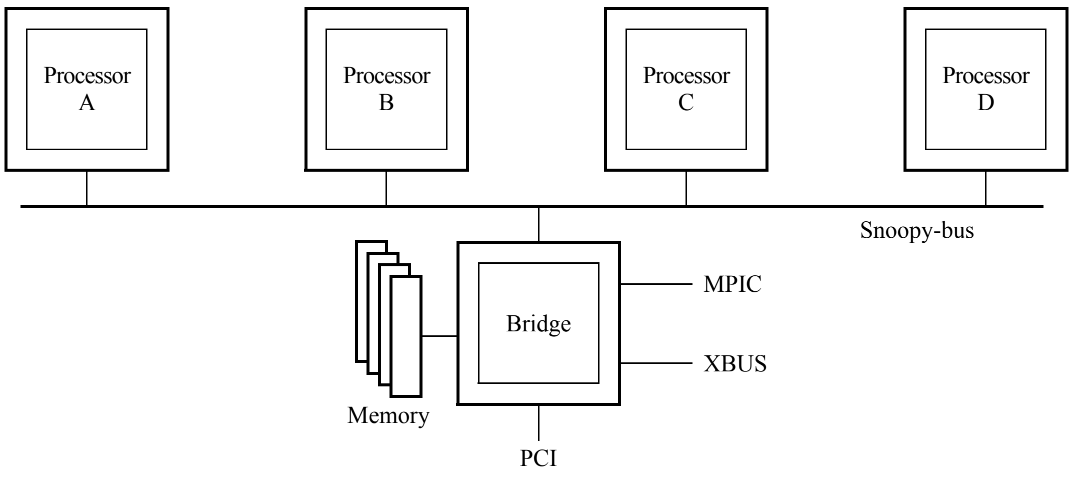


Figure 1-1. A Snoopy Bus-Based System

Super computers, massively parallel, and clustered machines that have distributed memory systems must use a different technique from broadcasting for maintaining memory coherency. Because a broadcast snoopy protocol in these machines is not efficient given the number of devices that must participate and the latency and transaction overhead involved, coherency mechanisms such as memory directories or distributed linked lists are required to keep track of where the most current copy of data resides. These schemes are often referred to as cache coherent non-uniform memory access (CC-NUMA) protocols. A typical distributed memory system architecture is shown in Figure .1-2

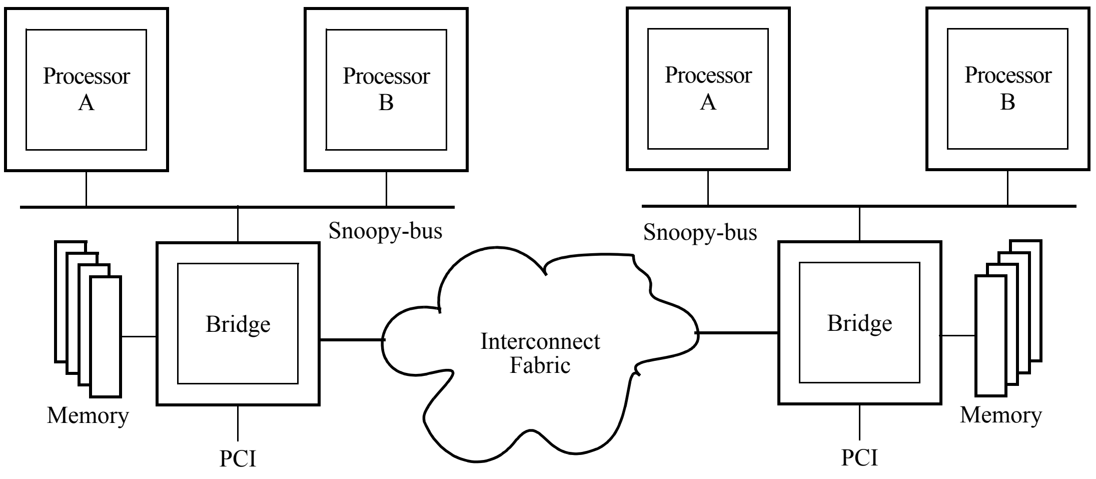


Figure 1-2. A Distributed Memory System

For RapidIO, a relatively simple directory-based coherency scheme is chosen. For this method each memory controller is responsible for tracking where the most current copy of each data element resides in the system. RapidIO furnishes a variety of ISA specific cache control and operating system support operations such as block flushes and TLB synchronization mechanisms.

To reduce the directory overhead required, the architecture is optimized around small clusters of 16 processors known as coherency domains. With the concept of domains, it is possible for multiple coherence groupings to coexist in the interconnect as tightly coupled processing clusters.

1.2 Features of the Globally Shared Memory Specification

The following are features of the RapidIO GSM specification designed to satisfy the needs of various applications and systems:

1.2.1 Functional Features

- A cache coherent non-uniform memory access (CC-NUMA) system architecture is supported to provide a globally shared memory model because physics is forcing component interfaces in many high-speed designs to be point-to-point instead of traditional bus-based.
- The size of processor memory requests are either in the cache coherence granularity, or smaller. The coherence granule size may be different for different processor families or implementations.
- Instruction sets in RapidIO support a variety of cache control and other operations such as block flushes. These functions are supported to run legacy applications and operating systems.

1.2.2 Physical Features

- RapidIO packet definition is independent of the width of the physical interface to other devices on the interconnect fabric.
- The protocols and packet formats are independent of the physical interconnect topology. The protocols work whether the physical fabric is a point-to-point ring, a bus, a switched multi-dimensional network, a duplex serial connection, and so forth.
- RapidIO is not dependent on the bandwidth or latency of the physical fabric.
- The protocols handle out-of-order packet transmission and reception.
- Certain devices have bandwidth and latency requirements for proper operation. RapidIO does not preclude an implementation from imposing these constraints within the system.

1.2.3 Performance Features

- Packet headers must be as small as possible to minimize the control overhead and be organized for fast, efficient assembly and disassembly.
- 48- and 64-bit addresses are required in the future, and must be supported initially.
- An interventionist (non-memory owner, direct-to-requestor data transfer, analogous to a cache-to-cache transfer) protocol saves a large amount of latency for memory accesses that cause another processing element to provide the requested data.
- Multiple transactions must be allowed concurrently in the system, otherwise a majority of the potential system throughput is wasted.

1.3 Contents

Following are the contents of the *RapidIO Interconnect Globally Shared Memory Logical Specification:*

- Chapter 1, “Overview,” describes the set of operations and transactions supported by the RapidIO globally shared memory protocols.
- Chapter 2, “System Models,” introduces some possible devices that could participate in a RapidIO GSM system environment. The chapter explains the memory directory-based mechanism that tracks memory accesses and maintains cache coherence. Transaction ordering and deadlock prevention are also covered.
- Chapter 3, “Operation Descriptions,” describes the set of operations and transactions supported by the RapidIO globally-shared memory (GSM) protocols.
- Chapter 4, “Packet Format Descriptions,” contains the packet format definitions for the GSM specification. The two basic types, request and response packets, with their sub-types and fields are defined. The chapter explains how memory read latency is handled by RapidIO.
- Chapter 5, “Globally Shared Memory Registers,” describes the visible register set that allows an external processing element to determine the globally shared memory capabilities, configuration, and status of a processing element using this logical specification. Only registers or register bits specific to the GSM logical specification are explained. Refer to the other RapidIO logical, transport, and physical specifications of interest to determine a complete list of registers and bit definitions.
- Chapter 6, “Communication Protocols,” contains the communications protocol definitions for this GSM specification.
- Chapter 7, “Address Collision Resolution Tables,” explains the actions necessary under the RapidIO GSM model to resolve address collisions.

1.4 Terminology

Refer to the Glossary at the back of this document.

1.5 Conventions

||Concatenation, used to indicate that two fields are physically associated as consecutive bits

ACTIVE_HIGHNames of active high signals are shown in uppercase text with no overbar. Active-high signals are asserted when high and not asserted when low.

$\overline{\text{ACTIVE_LOW}}$Names of active low signals are shown in uppercase text with an overbar. Active low signals are asserted when low and not asserted when high.

*italics*Book titles in text are set in italics.

REG[FIELD]Abbreviations or acronyms for registers are shown in uppercase text. Specific bits, fields, or ranges appear in brackets.

TRANSACTIONTransaction types are expressed in all caps.

operationDevice operation types are expressed in plain text.

nA decimal value.

[n-m]Used to express a numerical range from n to m.

0bnnA binary value, the number of bits is determined by the number of digits.

0x*nn*A hexadecimal value, the number of bits is determined by the number of digits or from the surrounding context; for example, 0x*nn* may be a 5, 6, 7, or 8 bit value

2 Chapter 2 - System Models

This overview introduces some possible devices in a RapidIO system.

2.1Processing Element Models

Figure 2-1 describes a possible RapidIO-based computing system. The processing element is a computer device such as a processor attached to a local memory and also attached to a RapidIO system interconnect. The bridge part of the system provides I/O subsystem services such as high-speed PCI interfaces and gigabit ethernet ports, interrupt control, and other system support functions. Multiple processing elements require cache coherence support in the RapidIO protocol to preserve the traditional globally shared memory programming model (discussed in Section, “2.2.1 Globally Shared Memory System Model”).

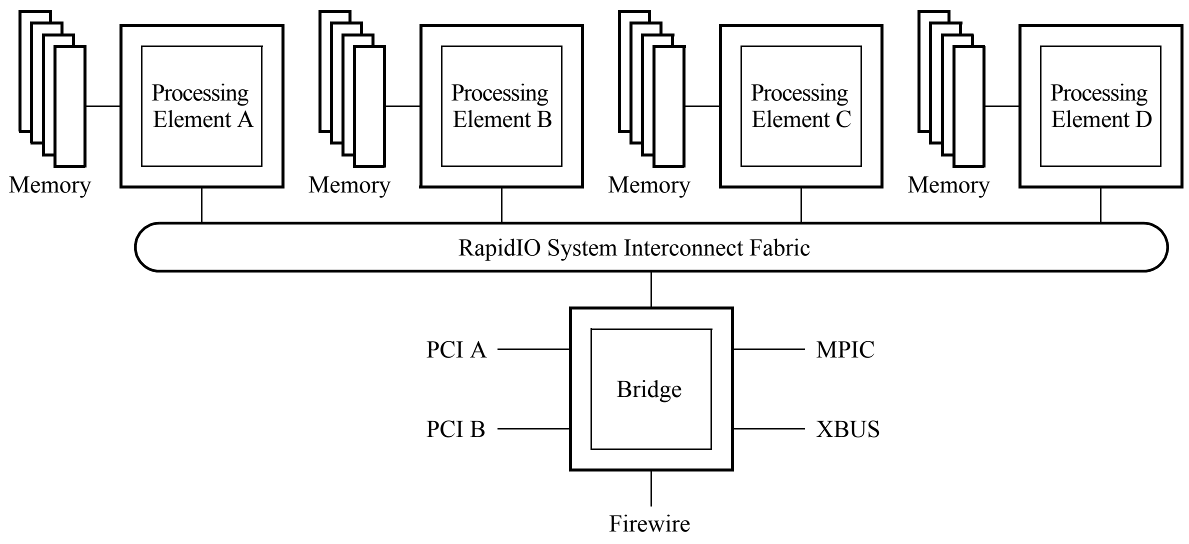


Figure 2-1. A Possible RapidIO-Based Computing System

A processing element containing a processor typically has associated with it a caching hierarchy to improve system performance. The RapidIO protocol supports a set of operations sufficient to fulfill the requirements of a processor with a caching hierarchy and associated support logic such as a processing element.

RapidIO is defined so that many types of devices can be designed for specific applications and connected to the system interconnect. These devices may participate in the cache coherency protocol, act as a DMA device, utilize the message passing facilities to communicate with other devices on the interconnect, and so forth. A bridge could be designed, for example, to use the message passing facility to pass ATM packets to and from a processing element for route processing. The following sections describe several possible processing elements.

2.1.1 Processor-Memory Processing Element Model

Figure 2-2 shows an example of a processing element consisting of a processor connected to an agent device. The agent carries out several services on behalf of the processor. Most importantly, it provides access to a local memory that has much lower latency than memory that is local to another processing element (remote memory accesses). It also provides an interface to the RapidIO interconnect to service those remote memory accesses.

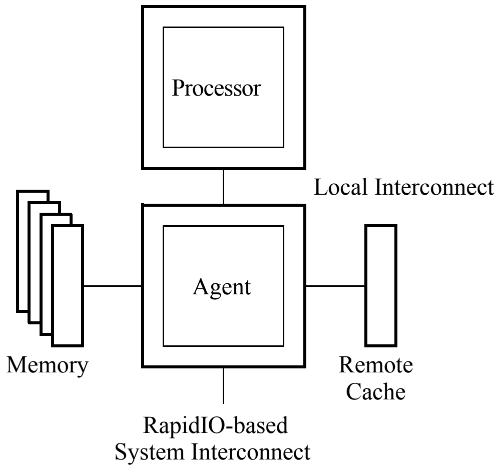


Figure 2-2 Processor-Memory Processing Element Example

In support of the remote accesses, the agent maintains a cache of remote accesses that includes all remote data currently residing in and owned by the local processor. This cache may be either external or internal to the agent device.

Agent caching is necessary due to the construction of the RapidIO cache coherence protocol combined with the cache hierarchy behavior in modern processors. Many modern processors have multiple level non-inclusive caching structures that are maintained independently. This implies that when a coherence granule is cast out of the processor, it may or may not be returning ownership of the granule to the memory system. The RapidIO protocol requires that ownership of a coherence granule be guaranteed to be returned to the system on demand and without ambiguous cache state changes as with the castout behavior. The remote cache can guarantee that a coherence granule requested by the system is owned locally and can be returned to the home memory (the physical memory containing the coherence granule) on demand. A processing element that is fully integrated would also need to support this behavior.

2.1.2 Integrated Processor-Memory Processing Element Model

Another form of a processor-memory processing element is a fully integrated component that is designed specifically to connect to a RapidIO interconnect system as shown in Figure .2-3. This type of device integrates a memory system and other support logic with a processor on the same piece of silicon or within the same package. Because such a device is designed specifically for RapidIO, a remote cache is not required because the proper support can be designed into the processor and its associated logic rather than requiring an agent to compensate for a stand alone processor's behavior.

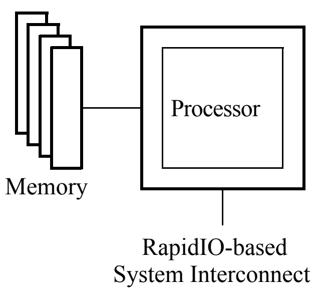


Figure 2-3. Integrated Processor-Memory Processing Element Example

2.1.3 Memory-Only Processing Element Model

A different processing element may not contain a processor at all, but may be a memory-only device as in Figure 2-4. This type of device is much simpler than a processor as it is only responsible for responding to requests from the external system, not from local requests as in the processor-based model. As such, its memory is remote for all processors in the system.

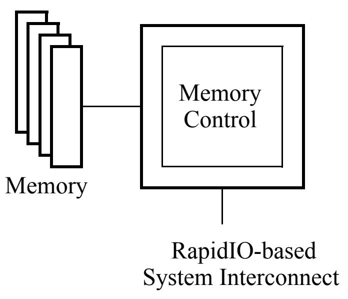


Figure 2-4. Memory-Only Processing Element Example

2.1.4 Processor-Only Processing Element

Similar to a memory-only element, a processor-only element has no local memory. A processor-only processing element is shown in Figure .2-5.

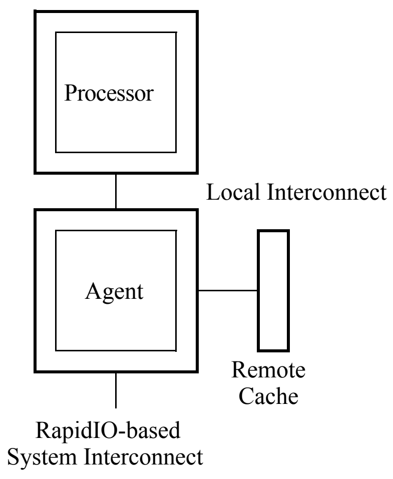


Figure 2-5. Processor-Only Processing Element Example

2.1.5 I/O Processing Element

This type of processing element is shown as the bridge in Figure 0-1. This device has distinctly different behavior than a processor or a memory. An I/O device only needs to move data into and out of local or remote memory in a cache coherent fashion. This means that if the I/O device needs to read from memory, it only needs to obtain a known good copy of the data to write to the external device (such as a disk drive or video display). If the I/O device needs to write to memory, it only needs to get ownership of the coherence granule returned to the home memory and not take ownership for itself. Both of these operations have special support in the RapidIO protocol.

2.1.6 Switch Processing Element

A switch processing element is a device that allows communication with other processing elements through the switch. A switch may be used to connect a variety of RapidIO compliant processing elements. A possible switch is shown in Figure . Behavior of the switches, and the interconnect fabric 0.1in general, is addressed in the *RapidIO Common Transport Specification*.

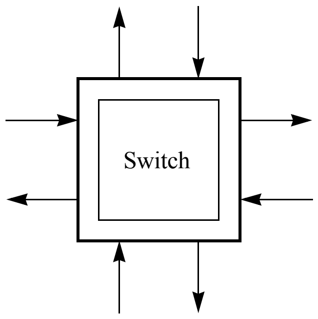


Figure 2-6. Switch Processing Element ExFigure 0.1. ample

2.2 Programming Models

RapidIO supports applications developed under globally shared memory and software-managed cache coherence programming models.

2.2.1 Globally Shared Memory System Model

The preferred programming model for modern computer systems provides memory that is accessible from all processors in a cache coherent fashion. This model is also known as GSM, or globally shared memory. For traditional bus-based computer systems this is not a difficult technical problem to solve because all participants in the cache coherence mechanism see all memory activity simultaneously, meaning that communication between processors is very fast and handled without explicit software control. However, in a non-uniform memory access system, this simultaneous memory access visibility is not the case.

With a distributed memory system, cache coherence needs to be maintained through some tracking mechanism that keeps records of memory access activity and explicitly notifies specific cache coherence participant processing elements when a cache coherence hazard is detected. For example, if a processing element wishes to write to a memory address, all participant processing elements that have accessed that coherence granule are notified to invalidate that address in their caches. Only when all of the participant processing elements have completed the invalidate operation and replied back to the tracking mechanism is the write allowed to proceed.

The tracking mechanism preferred for the RapidIO protocol is the memory directory based system model. This system model allows efficient, moderate scalability with a reasonable amount of information storage required for the tracking mechanism.

Cache coherence is defined around the concept of domains. The RapidIO protocol assumes a memory directory based cache coherence mechanism. Because the storage requirements for the directory can be high, the protocol was optimized assuming a 16-participant domain size as a reasonable coherence scalability limit. With this limit in mind, a moderately scalable system of 16 participants can be designed, possibly using a multicast mechanism in the transport layer for better efficiency. This size does not limit a system designer from defining a larger or a smaller coherent system such as the four processing element system in Figure 2.1 since the number of domains and the number of participants is flexible. The total number of coherence domains and the scalability limit are determined by the number of transport bits allowed by the appropriate transport layer specification.

Table 2.1 describes an example of the directory states assumed for the RapidIO protocol for a small four-processing element cache coherent system (the table assumes that processor 0 is the local processor). Every coherence granule that is accessible by a remote processing element has this 4-bit field associated with it, so some state storage is required for each globally shared granule. The least significant bit (the right most, bit 3) indicates that a processing element has taken ownership of a coherence granule. The remaining three bits indicate that processing elements have accessed that coherence granule, or the current owner if the granule has been modified, with bit 0 corresponding to processor 3, bit 1 corresponding to processor 2, and bit 2 corresponding to processor 1. These bits are also known as the sharing mask or sharing list.

Owing to the encoding of the bits, the local processing element is always assumed to have accessed the granule even if it has not. This definition allows us to know exactly which processing elements have participated in the cache coherency protocol for each shared coherence granule at all times. Other state definitions can be implemented as long as they encompass the MSL (modified, shared, local) state functionality described here.

Table 2-1. RapidIO Memory Directory Definition

State	Description
0000	Processor 0 (local) shared
0001	Processor 0 (local) modified
0010	Processor 1, 0 shared
0011	Processor 1 modified
0100	Processor 2, 0 shared
0101	Processor 2 modified
0110	Processor 2, 1, 0 shared
0111	Illegal
1000	Processor 3, 0 shared
1001	Processor 3 modified
1010	Processor 3, 1, 0 shared
1011	Illegal
1100	Processor 3, 2, 0 shared
1101	Illegal
1110	Processor 3, 2, 1, 0 shared
1111	Illegal

When a coherence granule is referenced, the corresponding 4-bit coherence state is examined by the memory controller to determine if the access can be handled in memory, or if data must be obtained from the current owner (a shared granule is owned by the home memory). Coherence activity in the system is started using the cache coherence protocol, if it is necessary to do so, to complete the memory operation.

2.2.2 Software-Managed Cache Coherence Programming Model

The software-managed cache coherence programming model depends upon the application programmer to guarantee that the same coherence granule is not resident in more than one cache in the system simultaneously if it is possible for that coherence granule to be written by one of the processors. The application software allows sharing of written data by using cache manipulation instructions to flush these coherence granules to memory before they are read by another processor. This programming model is useful in transaction and distributed processing types of systems.

2.3 System Issues

The following sections describe transaction ordering and system deadlock considerations in a RapidIO GSM system.

2.3.1 Operation Ordering

Operation completion ordering in a globally shared memory system is managed by the completion units of the processing elements participating in the coherence protocol and by the coherence protocol itself.

2.3.2 Transaction Delivery

There are two basic types of delivery schemes that can be built using RapidIO processing elements: unordered and ordered. The RapidIO logical protocols assume that all outstanding transactions to another processing element are delivered in an arbitrary order. In other words, the logical protocols do not rely on transaction interdependencies for operation. RapidIO also allows completely ordered delivery systems to be constructed. Each type of system puts different constraints on the implementation of the source and destination processing elements and any intervening hardware. The specific mechanisms and definitions of how RapidIO enforces transaction ordering are discussed in the appropriate physical layer specification.

2.3.3 Deadlock Considerations

A deadlock can occur if a dependency loop exists. A dependency loop is a situation where a loop of buffering devices is formed, in which forward progress at each device is dependent upon progress at the next device. If no device in the loop can make progress then the system is deadlocked.

The simplest solution to the deadlock problem is to discard a packet. This releases resources in the network and allows forward progress to be made. RapidIO is designed to be a reliable fabric for use in real time tightly coupled systems, therefore, discarding packets is not an acceptable solution.

In order to produce a system with no chance of deadlock it is required that a deadlock free topology be provided for response-less operations. Dependency loops to single direction packets can exist in unconstrained switch topologies. Often the dependency loop can be avoided with simple routing rules. Topologies like hypercubes or three-dimensional meshes, physically contain loops. In both cases, routing is done in several dimensions (x,y,z). If routing is constrained to the x dimension, then y, then z (dimension ordered routing), then topology related dependency loops are avoided in these structures.

In addition, a processing element design shall not form dependency links between its input and output port. A dependency link between input and output ports occurs if a processing element is unable to accept an input packet until a waiting packet can be issued from the output port.

RapidIO supports operations, such as coherent read-for-ownership operations, that require responses to complete. These operations can lead to a dependency link between an processing element’s input port and output port.

As an example of an input to output port dependency, consider a processing element where the output port queue is full. The processing element can not accept a new request at its input port since there is no place to put the response in the output port queue. No more transactions can be accepted at the input port until the output port is able to free entries in the output queue by issuing packets to the system.

A further consideration is that of the read-for-ownership operation colliding with a castout of the requested memory address by another processing element. In order for the read-for-ownership operation to complete the underlying castout operation must complete. Therefore the castout must be given higher preference in the system in order to move ahead of other operations in order to break up the dependency.

The method by which a RapidIO system maintains a deadlock free environment is described in the appropriate Physical Layer specification.

3 Chapter 3 - Operation Descriptions

This chapter describes the set of operations and transactions supported by the RapidIO globally-shared memory (GSM) protocols. The opcodes and packet formats are described in Chapter 4, “Packet Format Descriptions.” The complete protocols are described in Chapter 6, “Communication Protocols.”

The RapidIO operation protocols use request/response transaction pairs through the interconnect fabric. A processing element sends a request transaction to another processing element if it requires an activity to be carried out. The receiving processing element responds with a response transaction when the request has been completed or if an error condition is encountered. Each transaction is sent as a packet through the interconnect fabric. For example, a processing element that requires data from home memory in another processing element sends a READ_HOME transaction in a request packet. The receiving element then reads its local memory at the requested address and returns the data in a DONE transaction via a response packet. Note that not all requests require responses; some requests assume that the desired activity will complete properly.

A number of possible response transactions can be received by a requesting processing element:

- A DONE response indicates to the requestor that the desired transaction has completed and also returns data for read-type transactions as described above.
- The INTERVENTION, DONE_INTERVENTION, and DATA_ONLY responses are generated as part of the processing element-to-processing element (as opposed to processing element-to-home memory) data transfer mechanism defined by the cache coherence protocol. The INTERVENTION and DONE_INTERVENTION responses are abbreviated as INTERV and DONE_INTERV in this chapter.
- The NOT_OWNER and RETRY responses are received when there are address conflicts within the system that need resolution.
- An ERROR response means that the target of the transaction encountered an unrecoverable error and could not complete the transaction.

Packets may contain additional information that is interpreted by the interconnect fabric to route the packets through the fabric from the source to the destination, such as a device number. These requirements are described in the appropriate RapidIO transport layer specification and are beyond the scope of this specification.

Depending upon the interconnect fabric, other packets may be generated as part of the physical layer protocol to manage flow control, errors, etc. Flow control and other fabric-specific communication requirements are described in the appropriate RapidIO physical layer specification and are beyond the scope of this document.

Each request transaction sent into the system is marked with a transaction ID that is unique for each requestor and responder processing element pair. This transaction ID allows a response to be easily matched to the original request when it is returned to the requestor. An end point cannot reuse a transaction ID value to the same destination until the response from the original transaction has been received by the requestor. The number of outstanding transactions that may be supported is implementation dependent.

The transaction behaviors are also described as state machine behavior in Chapter 6, “Communication Protocols”.

3.1 GSM Operations Cross Reference

Table 3-1contains a cross reference of the GSM operations defined in this RapidIO specification and their system usage.

Table 3-1. GSM Operations Cross Reference

Operation	Transactions Used	Possible System Usage	Description	Packet Format	Protocol
Read	READ_HOME, READ_OWNER, RESPONSE	CC-NUMA operation	Section 6.3	Types 1 and 2: Section 4.1.5 and Section 4.1.6	Section 6.3
Instruction read	IREAD_HOME, READ_OWNER, RESPONSE	Combination of CC-NUMA and software-maintained coherence of instruction caches	Section 6.3	Type 2: Section 4.1.6	Section 6.3
Read-for-ownership	READ_TO_OWN_HOME, READ_TO_OWN_OWNER, DKILL_SHARER RESPONSE	CC-NUMA operation	Section 3.2.3	Types 1 and 2: Section 4.1.5 and Section 4.1.6	Section 6.5
Data cache invalidate	DKILL_HOME, DKILL_SHARER, RESPONSE	CC-NUMA operation; software-maintained coherence operation	Section 3.2.4	Type 2: Section 4.1.6	Section 6.6
Castout	CASTOUT, RESPONSE	CC-NUMA operation	Section 3.2.5	Type 5: Section 4.1.8	Section 6.7

Table 3-1. GSM Operations Cross Reference(Continued)

Operation	Transactions Used	Possible System Usage	Description	Packet Format	Protocol
TLB invalidate-entry	TLBIE, RESPONSE	Software-maintained coherence of page table entries	Section 3.2.6	Type 2: Section 4.1.6	Section 6.8
TLB invalidate-entry synchronize	TLBSYNC, RESPONSE	Software-maintained coherence of page table entries	Section 3.2.7	Type 2: Section 4.1.6	Section 6.8
Instruction cache invalidate	IKILL_HOME, IKILL_SHARER, RESPONSE,	Software-maintained coherence of instruction caches	Section 3.2.8	Type 2: Section 4.1.6	Section 6.6
Data cache flush	FLUSH, DKILL_SHARER, READ_TO_OWN_OWNER, RESPONSE	CC-NUMA flush instructions; CC-NUMA write-through cache support; CC-NUMA DMA I/O device support; software-maintained coherence operation.	Section 3.2.9	Types 2 and 5: Section 4.1.6 and Section 4.1.8	Section 6.9
I/O read	IO_READ_HOME, IO_READ_OWNER, INTERV, RESPONSE	CC-NUMA DMA, I/O DMA device support	Section 3.2.10	Types 1 and 2: Section 4.1.5 and Section 4.1.6	Section 6.10

3.2 GSM Operations

A set of transactions are used to support GSM (cache coherence) operations to cacheable memory space. The following descriptions assume that all requests are to system memory rather than to some other type of device.

GSM operations occur based on the size of the coherence granule. Changes in the coherence granule for a system do not change any of the operation protocols, only the data payload size. The only exception to this is a flush operation, which may have a double-word or sub-double-word data payload to support coherent I/O and write-through caches and no data payload to support cache manipulation instructions.

Some transactions are sent to multiple recipients in the process of completing an operation. These transactions can be sent either as a number of directed transactions or as a single transaction if the transport layer has multicast capability. Multicast capability and operation is defined in the appropriate RapidIO transport layer specification.

3.2.1 Read Operations

The READ_HOME, READ_OWNER, and RESPONSE transactions are used during a read operation by a processing element that needs a shared copy of cache-coherent data from the memory system. A read operation always returns one coherence granule-sized data payload.

The READ_HOME transaction is used by a processing element that needs to read a shared copy of a coherence granule from a remote home memory on another processing element.

The READ_OWNER transaction is used by a home memory processing element that needs to read a shared copy of a coherence granule that is owned by a remote processing element.

The following types of read operations are possible:

If the requested data exists in the memory directory as shared, the data can be returned immediately from memory with a DONE RESPONSE transaction and the requesting processing element's device ID is added to the sharing mask as shown in Figure 3-1.

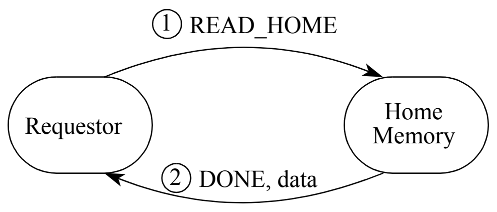


Figure 3-1. Read Operation to Remote Shared CoherencFie Granule

- If the requested data exists in the memory directory as modified, the up-to-date (current) data must be obtained from the owner. The home memory then sends a READ_OWNER request to the processing element that owns the coherence granule. The owner passes a copy of the data to the original requestor and to memory, memory is updated, and the directory state is changed from modified and owner to shared by the previous owner and the requesting processing element's device ID as shown in Figure . 3-2.

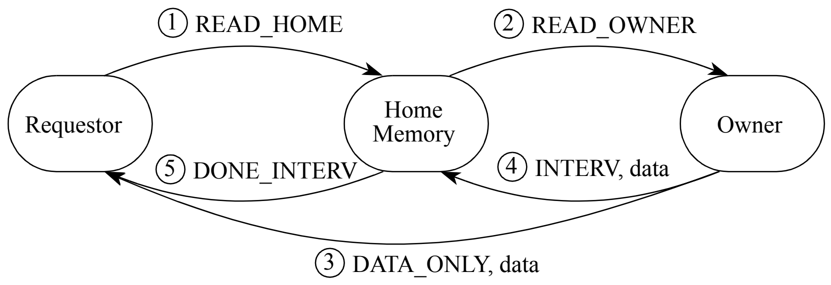


Figure 3-2. Read Operation to Remote Modified Coherence Granule

- If the processing element requesting a modified coherence granule happens to be the home for the memory, some of the transactions can be eliminated as shown in Figure .3-3.

Figure 0.1

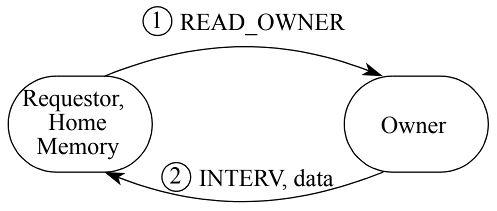


Figure 3-3. Read Operation to Local Modified Coherence Granule

3.2.2 Instruction Read Operations

Some processors have instruction caches that do not participate in the system cache coherence mechanism. Additionally, the instruction cache load may also load a shared instruction and data cache lower in the cache hierarchy. This can lead to a situation where the instruction cache issues a shared read operation to the system for a coherence granule that is owned by that processor's data cache, resulting in a cache coherence paradox to the home memory directory.

Due to this situation, an instruction read operation must behave like a coherent shared read relative to the memory directory and as a non-coherent operation relative to the requestor. Therefore, the behavior of the instruction read operation is nearly identical to a data read operation with the only difference being the way that the apparent coherence paradox is managed.

The IREAD_HOME and RESPONSE transactions are used during an instruction read operation by a processing element that needs a copy of sharable instructions from the memory system. An instruction read operation always returns one coherence granule-sized data payload. Use of the IREAD_HOME transaction rather than the READ_HOME transaction allows the memory directory to properly handle the paradox case without sacrificing coherence error detection in the system. The IREAD_HOME transaction participates in address collision detection at the home memory but does not participate in address collision detection at the requestor.

The following types of instruction read operations are possible:

- If the requested instructions exists in the memory directory as shared, the instructions can be returned immediately from memory and the requesting processing element's device ID is added to the sharing mask as shown in Figure . 3-4.

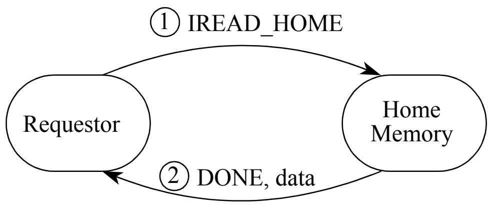


Figure 3-4. Instruction Read Operation to Remote Shared Coherence Granule

- If the requested data exists in the memory directory as modified, the up-to-date (current) data must be obtained from the owner. The home memory then sends a READ_OWNER request to the processing element that owns the coherence granule. The owner passes a copy of the data to the original requestor and to memory, memory is updated, and the directory state is changed from modified and owner to shared by the previous owner and the requesting processing element's device ID as shown in Figure .3-5.

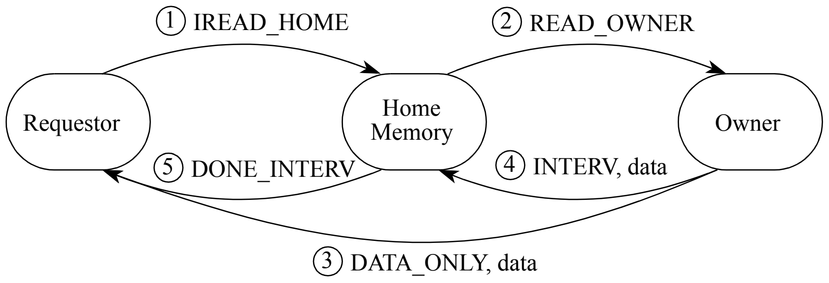


Figure 3-5. Instruction Read Operation to Remote Modified Coherence Granule

- If the processing element requesting a modified coherence granule happens to be the home for the memory the READ_OWNER transaction is used to obtain the coherence granule as shown in Figure .3-6.

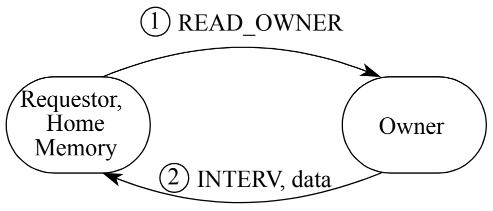


Figure 3-6. Instruction Read Operation to Local Modified Coherence Granule

- The apparent paradox case is if the requesting processing element is the owner of the coherence granule as shown in Figure 3.7. The home memory sends a READ_OWNER transaction back to the requesting processing element with the source and secondary ID set to the home memory ID, which indicates that the response behavior should be an INTERVENTION transaction rather than an INTERVENTION and a DATA_ONLY transaction as shown in Figure .3.5.

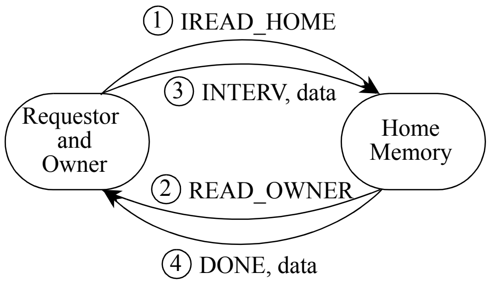


Figure 3-7. Instruction Read Operation Paradox Case

3.2.3 Read-for-Ownership Operations

The READ_TO_OWN_HOME, READ_TO_OWN_OWNER, DKILL_SHARER, and RESPONSE transactions are used during read-for-ownership operations by a processing element that needs to write to a coherence granule that does not exist in its caching hierarchy. A read-for-ownership operation always returns one coherence granule-sized data payload. These transactions are used as follows:

The READ_TO_OWN_HOME transaction is used by a processing element that needs to read a writable copy of a coherence granule from a remote home memory on another processing element. This transaction causes a copy of the data to be returned to the requestor, from memory if the data is shared, or from the owner if it is modified.

- The READ_TO_OWN_OWNER transaction is used by a home memory processing element that needs to read a writable copy of a coherence granule that is owned by a remote processing element.
- The DKILL_SHARER transaction is used by the home memory processing element to invalidate shared copies of the coherence granule in remote processing elements.
- Following are descriptions of the read-for-ownership operations:
- If the coherence granule is shared, DKILL_SHARER transactions are sent to the participants indicated in the sharing mask, which results in a cache invalidate operation for the recipients as shown in Figure .3-8.

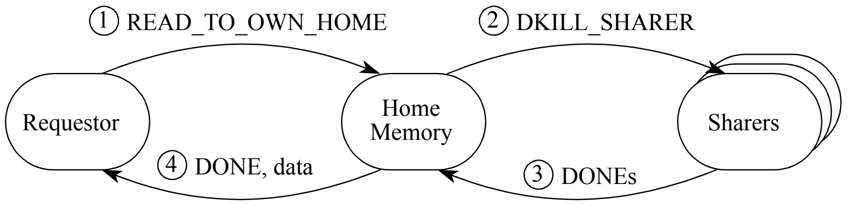


Figure 3-8. Read-for-Ownership Operation to Remote Shared Coherence Granule

- If the coherence granule is modified, a READ_TO_OWN_OWNER transaction is sent to the owner, who sends a copy of the data to the requestor (intervention) and marks the address as invalid as shown in Figure .3-9. The final memory directory state shows that the coherence granule is modified and owned by the requestor's device ID.
- Because the coherence granule in the memory directory was marked as modified, home memory does not necessarily need to be updated. However, the RapidIO protocol requires that a processing element return the modified data and update the memory, allowing some attempt for data recovery if a coherence problem occurs.

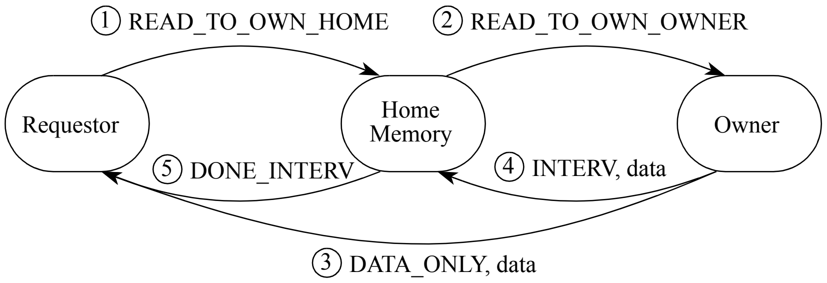


Figure 3-9. Read-for-Ownership Operation to Remote Modified Coherence Granule

If the requestor is on the same processing element as the home memory and the coherence granule is shared, a DKILL_SHARER transaction is sent to all sharing processing elements (see Figure 3-10). The final directory state is marked as modified and owned by the local requestor.

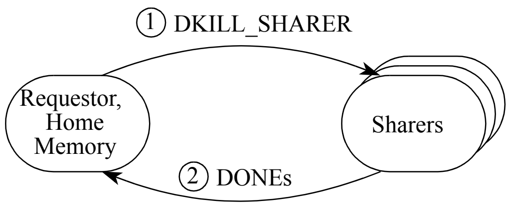


Figure 3-10. Read-for-Ownership Operation to Local Shared Coherence Granule

If the requestor is on the same processing element as the home memory and the coherence granule is owned by a remote processing element, a READ_TO_OWN_OWNER transaction is sent to the owner (see Figure 3-11). The final directory state is marked as modified and owned by the local requestor.

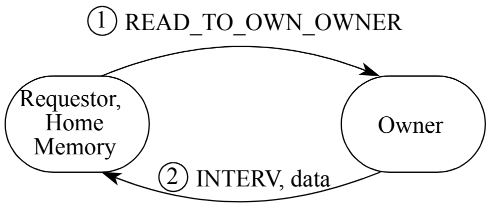


Figure 3-11. Read-for-Ownership Operation to Local Modified Coherence Granule

3.2.4 Data Cache Invalidate Operations

The DKILL_HOME, DKILL_SHARER, and RESPONSE transactions are requests to invalidate a coherence granule in all of the participants in the coherence domain as follows:

- The DKILL_HOME transaction is used by a processing element to invalidate a data coherence granule that has home memory in a remote processing element.
- The DKILL_SHARER transaction is used by the home memory processing element to invalidate shared copies of the data coherence granule in remote processing elements.

Data cache invalidate operations are also useful for systems that implement software-maintained cache coherence. In this case, a requestor may send DKILL_HOME and DKILL_SHARER transactions directly to other processing elements without going through home memory as in a CC-NUMA system. The transactions used for the data cache invalidate operation depend on whether the requestor is on the same processing element as the home memory of the coherence granule as follows:

- If the requestor is not on the same processing element as the home memory of the coherence granule, a DKILL_HOME transaction is sent to the remote home memory processing element. This causes the home memory for the shared coherence granule to send a DKILL_SHARER to all processing elements marked as sharing the granule in the memory directory state except for the requestor (see Figure 3-12). The final memory state shows that the coherence granule is modified and owned by the requesting processing element's device ID.

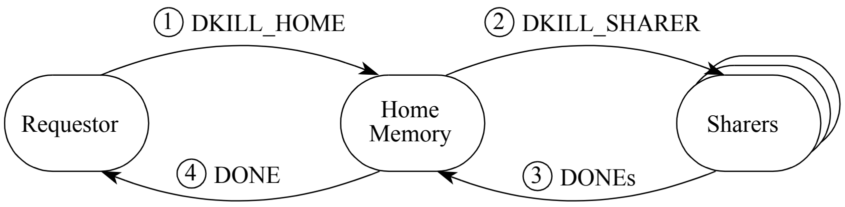


Figure 3-12. Data Cache Invalidate Operation to Remote Shared Coherence Granule

- If the requestor is on the same processing element as the home memory of the coherence granule, the home memory sends a DKILL_SHARER transaction to all processing elements marked as sharing the coherence granule in the memory directory. The final memory state shows the coherence granule modified and owned by the local processor (see Figure 3-13)..

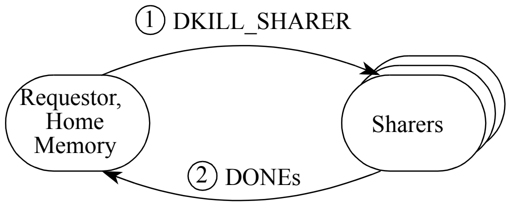


Figure 3-13. Data Cache Invalidate Operation to Local Shared Coherence Granule

3.2.5 Castout Operations

The CASTOUT and RESPONSE transactions are used in a castout operation by a processing element to relinquish its ownership of a coherence granule and return it to the home memory. The CASTOUT can be treated as a low-priority transaction unless there is an address collision with an incoming request, at which time it must become a high-priority transaction. The CASTOUT causes the home memory to be updated with the most recent data and changes the directory state to owned by home memory and shared (or owned, depending upon the default directory state) by the local processing element (see Figure 3-14).

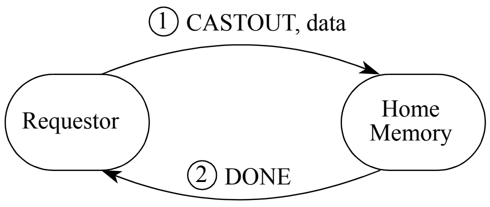


Figure 3-14. Castout Operation on Remote Modified Coherence Granule

A CASTOUT transaction does not participate in address collision detection at the home memory to prevent deadlocks or cache paradoxes caused by packet-to-packet timing in the interconnect fabric. For example, consider a case where processing element A is performing a CASTOUT that collides with an incoming READ_OWNER transaction. If the CASTOUT is not allowed to complete at the home memory, the system will deadlock. If the read operation that caused the READ_OWNER completes (through intervention) before the CASTOUT transaction is received at the home memory, the CASTOUT will appear to be illegal because the directory state will have changed.

3.2.6 TLB Invalidate-Entry Operations

The TLBIE and RESPONSE transactions are used for TLB invalidate-entry operations. If the processor TLBs do not participate in the cache coherence protocol, the TLB invalidate-entry operation is used when page table translation entries need to be modified. The TLBIE transaction is sent to all participants in the coherence domain except for the original requestor. A TLBIE transaction has no effect on the memory directory state for the specified address and does not participate in address collisions (see Figure 3-15)..

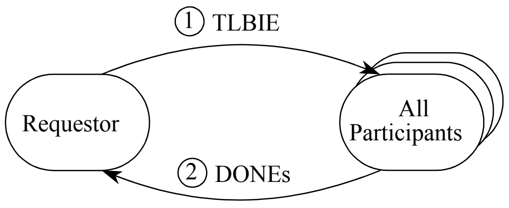


Figure 3-15. TLB Invalidate-Entry Operation

3.2.7 TLB Invalidate-Entry Synchronization Operations

The TLBSYNC and RESPONSE transactions are used for TLB invalidate-entry synchronization operations. It is used to force the completion of outstanding TLBIE transactions at the participants. The DONE response for a TLBSYNC transaction is only sent when all preceding TLBIE transactions have completed. This operation is necessary due to possible indeterminate completion of individual TLBIE transactions when multiple TLBIE transactions are being executed simultaneously. The TLBSYNC transaction is sent to all participants in the coherence domain except for the original requestor. The transaction has no effect on the memory directory state for the specified address and does not participate in address collisions (see Figure 3-16).

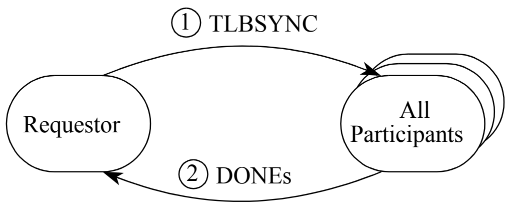


Figure 3-16. TLB Invalidate-Entry Synchronization Operation

3.2.8 Instruction Cache Invalidate Operations

The IKILL_HOME, IKILL_SHARER, and RESPONSE transactions are used during instruction cache invalidate operations to invalidate shared copies of an instruction coherence granule in remote processing elements. Instruction cache invalidate operations are needed if the processor instruction caches do not participate in the cache coherence protocol, requiring instruction cache coherence to be maintained by software.

An instruction cache invalidate operation has no effect on the memory directory state for the specified address and does not participate in address collisions. Following are descriptions of the instruction cache invalidate operations:

- If the requestor is not on the same processing element as the home memory of the coherence granule, an IKILL_HOME transaction is sent to the remote home memory processing element. This causes the home memory for the shared coherence granule to send an IKILL_SHARER to all processing element participants in the coherence domain because the memory directory state only properly tracks data, not instruction, accesses. (See Figure .3-17).

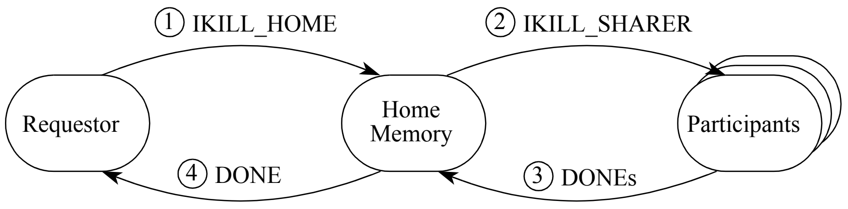


Figure 3-17. Instruction Cache Invalidate Operation to Remote Sharable Coherence Granule

- If the requestor is on the same processing element as the home memory of the coherence granule, the home memory sends an IKILL_SHARER transaction to all processing element participants in the coherence domain as shown in Figure . 3-18.

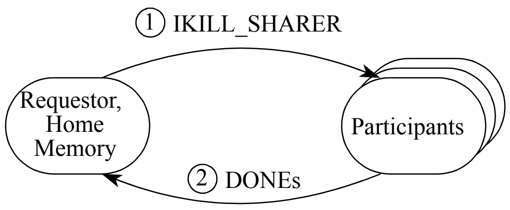


Figure 3-18. Instruction Cache Invalidate Operation to Local Sharable Coherence Granule

3.2.9 Data Cache Flush Operations

The FLUSH, DKILL_SHARER, READ_TO_OWN_OWNER, and RESPONSE transactions are used for data cache flush operations, which return ownership of a coherence granule back to the home memory if it is modified and invalidate all copies if the granule is shared. A flush operation with associated data can be used to implement an I/O system write operation and to implement processor write-through and cache manipulation operations. These transactions are used as follows:

- The FLUSH transaction is used by a processing element to return the ownership and current data of a coherence granule to home memory. The data payload for the FLUSH transaction is typically the size of the coherence granule for the system but may be multiple double-words or one double-word or less. FLUSH transactions without a data payload are used to support cache manipulation operations. The memory directory state is changed to owned by home memory and shared (or modified, depending upon the processing element's normal default state) by the local processing element.
- The DKILL_SHARER transaction is used by the home memory processing element to invalidate shared copies of the data coherence granule in remote processing elements.
- The READ_TO_OWN_OWNER transaction is used by a home memory processing element that needs to retrieve ownership of a coherence granule that is owned by a remote processing element.

The FLUSH transaction is able to specify multiple double-word and sub-double-word data payloads; however, they must be aligned to byte, half-word, word, or double-word boundaries. Multiple double-word FLUSH transactions cannot exceed the number of double-words in the coherence granule. The write size and alignment for the FLUSH transaction are specified in Table 4-8. Unaligned and non-contiguous operations are not supported and must be broken into multiple FLUSH transactions by the sending processing element.

A flush operation internal to a processing element that would cause a FLUSH transaction for a remote coherence granule owned by that processing element (for example, attempting a cache write-through operation to a locally owned remote coherence granule) must generate a CASTOUT rather than a FLUSH transaction to properly implement the RapidIO protocol. Issuing a FLUSH under these circumstances generates a memory directory state paradox error in the home memory processing element.

Following are descriptions of the flush operations:

- If a flush operation is to a remote shared coherence granule, the FLUSH transaction is sent to the home memory, which sends a DKILL_SHARER transaction to all of the processing elements marked in the sharing list except for the requesting processing element. The processing elements that receive the DKILL_SHARER transaction invalidate the specified address if it is found shared in their caching hierarchy (see Figure 3-19).

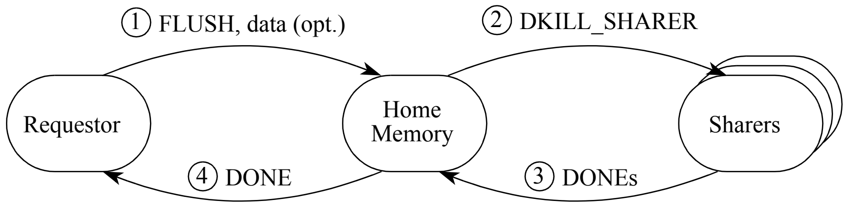


Figure 3-19. Flush Operation to Remote Shared Coherence Granule

- If the coherence granule is owned by a remote processing element, the home memory sends a READ_TO_OWN_OWNER transaction to it with the secondary (intervention) ID set to the home memory ID instead of the requestor ID. The owner then invalidates the coherence granule in its caching hierarchy and returns the coherence granule data (see Figure 3-20).

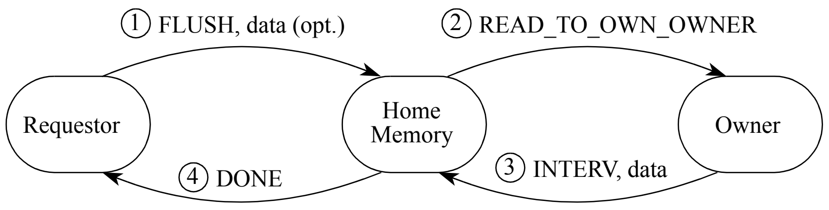


Figure 3-20. Flush Operation to Remote Modified Coherence Granule

If the requestor and the home memory for the coherence granule are in the same processing element, DKILL_SHARER transactions are sent to all participants marked in the sharing list (see Figure 3-21).

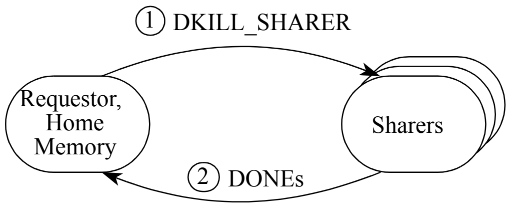


Figure 3-21. Flush Operation to Local Shared Coherence Granule

- If the requestor and the home memory for the coherence granule are in the same processing element but the coherence granule is owned by a remote processing element, a READ_TO_OWN_OWNER transaction is sent to the owner (see Figure 3-22).

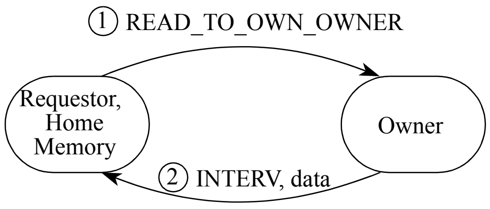


Figure 3-22. Flush Operation to Local Modified Coherence Granule

3.2.10 I/O Read Operations

The IO_READ_HOME, IO_READ_OWNER, and RESPONSE transactions are used during I/O read operations by a processing element that needs a current copy of cache-coherent data from the memory system, but does not need to be added to the sharing list in the memory directory state. The I/O read operation is most useful for DMA I/O devices. An I/O read operation always returns one coherence granule-sized data payload. These transactions are used as follows:

- The IO_READ_HOME transaction is used by a requestor that is not in the same processing element as the home memory for the coherence granule.
- The IO_READ_OWNER transaction is used by a home memory processing element that needs to read a copy of a coherence granule owned by a remote processing element.

Following are descriptions of the I/O operations:

- If the requested data exists in the memory directory as shared, the data can be returned immediately from memory and the sharing mask is not modified (see Figure3-23).

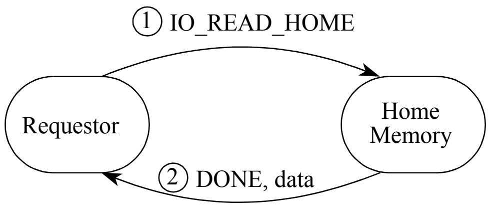


Figure 3-23. I/O Read Operation to Remote Shared Coherence Granule

- If the requested data exists in the memory directory as modified, the home memory sends an IO_READ_OWNER transaction to the processing element that owns the coherence granule. The owner passes a copy of the data to the requesting processing element (intervention) but retains ownership of and responsibility for the coherence granule (see Figure 3-24 and Figure 3-25).

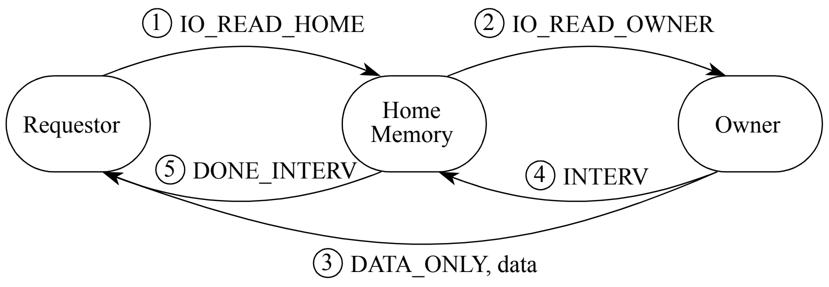


Figure 3-24. I/O Read Operation to Remote Modified Coherence Granule

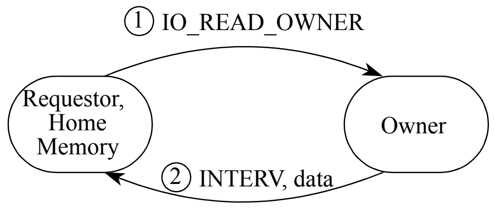
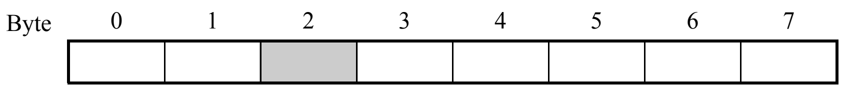


Figure 3-25. I/O Read Operation to Local Modified Coherence Granule

3.3 Endian, Byte Ordering, and Alignment

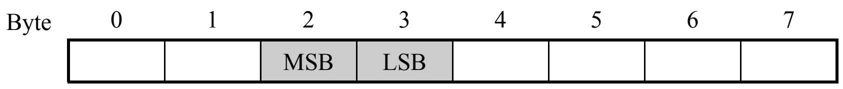
RapidIO has double-word (8-byte) aligned big-endian data payloads. This means that the RapidIO interface to devices that are little-endian shall perform the proper endian transformation to format a data payload.

Operations that specify data quantities that are less than 8 bytes shall have the bytes aligned to their proper byte position within the big-endian double-word, as in the examples shown in Figure 3-26 through Figure 3-27.



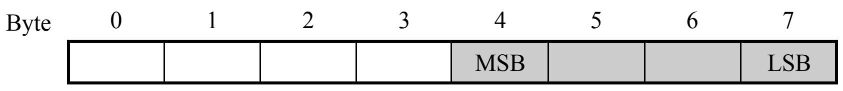
Byte address 0x0000_0002, the proper byte position is shaded.

Figure 3-26. Byte Alignment Example



Half-word address 0x0000_0002, the proper byte positions are shaded.

Figure 3-27. Half-Word Alignment Example



Word address 0x0000_0004, the proper byte positions are shaded.

Figure 3-28. Word Alignment Example

For write operations, a processing element shall properly align data transfers to a double-word boundary for transmission to the destination. This alignment may require breaking up a data stream into multiple transactions if the data is not naturally aligned. A number of data payload sizes and double-word alignments are defined to minimize this burden. Figure shows a 48-byte data stream that a processing element wishes to write to another processing element through the interconnect fabric. The data displayed in the figure is big-endian and double-word aligned with the bytes to be written shaded in grey. Because the start of the stream and the end of the stream are not aligned to a double-word boundary, the sending processing element shall break the stream into three transactions as shown in the figure.

The first transaction sends the first three bytes (in byte lanes 5, 6, and 7) and indicates a byte lane 5, 6, and 7 three-byte write. The second transaction sends all of the remaining data except for the final sub-double-word. The third transaction sends the final 5 bytes in byte lanes 0, 1, 2, 3, and 4 indicating a five-byte write in byte lanes 0, 1, 2, 3, and 4.

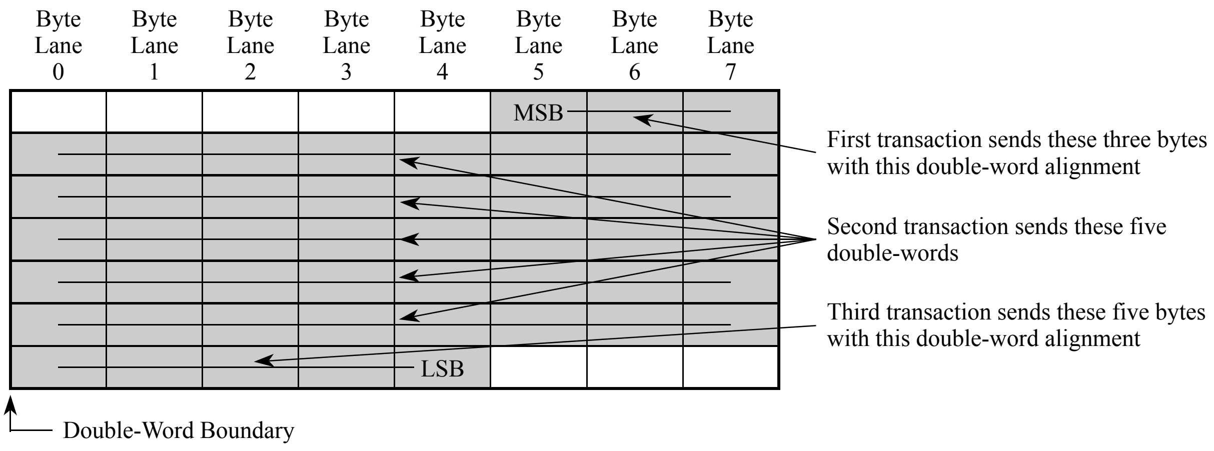

Byte Lane 0
Byte Lane 1
Byte Lane 2
Byte Lane 3
Byte Lane 4
Byte Lane 5
Byte Lane 6
Byte Lane 7
MSB
LSB
First transaction sends these three bytes with this double-word alignment
Second transaction sends these five double-words
Third transaction sends these five bytes with this double-word alignment
Double-Word Boundary

Figure 3-29. Data Alignment Example

4 Chapter 4 - Packet Format Descriptions

This chapter contains the packet format definitions for the *RapidIO Interconnect Globally Shared Memory Logical Specification*. There are four types of globally shared memory packet formats:

- Request
- Response
- Implementation-defined
- Reserved

The packet formats are intended to be interconnect fabric independent, so the system interconnect can be anything required for a particular application. Reserved formats, unless defined in another logical specification, shall not be used by a device.

4.1 Request Packet Formats

A request packet is issued by a processing element that needs a remote processing element to accomplish some activity on its behalf, such as a memory read operation. The request packet format types and their transactions for the *RapidIO Interconnect Globally Shared Memory Logical Specification* are shown in Table .

Table 4-1. Request Packet Type to Transaction Type Cross Reference

Request Packet Format Type	Transaction Type	Definition	Document Section No.
Type 0	Implementation-defined	Defined by the device implementation	Section 4.1.4
Type 1	READ_OWNER	Read shared copy of remotely owned coherence granule	Section 4.1.5
	READ_TO_OWN_OWNER	Read for store of remotely owned coherence granule	
	IO_READ_OWNER	Read for I/O of remotely owned coherence granule	

Table 4-1. Request Packet Type to Transaction Type Cross Reference(Continued)

Request Packet Format Type	Transaction Type	Definition	Document Section No.
Type 2	READ_TO_OWN_HOME	Read for store of home memory for coherence granule	Section 4.1.6
	READ_HOME	Read shared copy of home memory for coherence granule	
	IO_READ_HOME	Read for I/O of home memory for coherence granule	
	DKILL_HOME	Invalidate to home memory of coherence granule	
	IKILL_HOME	Invalidate to home memory of coherence granule	
	TLBIE	Invalidate TLB entry	
	TLBSYNC	Synchronize TLB invalidates	
	IREAD_HOME	Read shared copy of home memory for instruction cache	
	FLUSH	Force return of ownership of coherence granule to home memory, no update to coherence granule	
	IKILL_SHARER	Invalidate cached copy of coherence granule	
	DKILL_SHARER	Invalidate cached copy of coherence granule	
Type 3–4	—	Reserved	4.1.7
Type 5	CASTOUT	Return ownership of coherence granule to home memory	4.1.8
	FLUSH (with data)	Force return of ownership of coherence granule to home memory, update returned coherence granule	
Type 6–11	—	Reserved	4.1.9

4.4.1 Addressing and Alignment

The size of the address is defined as a system-wide parameter; thus the packet formats do not support mixed local physical address fields simultaneously. The least three significant bits of all addresses are not specified and are assumed to be logic 0.

The coherence-granule-sized cache-coherent write requests and read responses are aligned to a double-word boundary within the coherence granule, with the specified data payload size matching that of the coherence granule. Sub-double-word data payloads must be padded and properly aligned within the 8-byte boundary. Non-contiguous or unaligned transactions that would ordinarily require a byte mask are not supported. A sending device that requires this behavior must break the operation into multiple request transactions. An example of this is shown in Section 3.3, "Endian, Byte Ordering, and Alignment."

4.1.2 Data Payloads

Cache coherent systems are very sensitive to memory read latency. One way of reducing the latency is by returning the requested, or critical, double-word first upon a read request. Subsequent double-words are then returned in a sequential fashion. Table 4-2 and Table 4-3 show the return ordering for 32- and 64-byte coherence granules.

Table 4-2. Coherent 32-Byte Read Data Return Ordering

Requested Double-word	Double-word Return Ordering
0	0, 1, 2, 3
1	1, 2, 3, 0
2	2, 3, 0, 1
3	3, 0, 1, 2

Table 4-3. Coherent 64-Byte Read Data Return Ordering

Requested Double-word	Double-word Return Ordering
0	0, 1, 2, 3, 4, 5, 6, 7
1	1, 2, 3, 0, 4, 5, 6, 7
2	2, 3, 0, 1, 4, 5, 6, 7
3	3, 0, 1, 2, 4, 5, 6, 7
4	4, 5, 6, 7, 0, 1, 2, 3
5	5, 6, 7, 4, 0, 1, 2, 3
6	6, 7, 4, 5, 0, 1, 2, 3
7	7, 4, 5, 6, 0, 1, 2, 3

Data payloads for cache coherent write-type transactions are always linear starting with the specified address at the first double-word to be written, (including flush transactions that are not the size of the coherence granule). Data payloads that cross the coherence granule boundary can not be specified. This implies that all castout transactions start with the first double-word in the coherence granule. Table 4-4 and Table 4-5 show the cache-coherent write-data ordering for 32- and 64-byte coherence granules, respectively.

4.1.3 Field Definitions for All Request Packet Formats

Fields that are unique to type 1, type 2, and type 5 formats are defined in their sections. Bit fields that are defined as “reserved” shall be assigned to logic 0s when generated and ignored when received. Bit field encodings that are defined as “reserved” shall not be assigned when the packet is generated. A received reserved encoding is regarded as an error if a meaningful encoding is required for the transaction and function, otherwise it is ignored. Implementation-defined fields shall be ignored unless the encoding is understood by the receiving device. All packets described are bit streams from the first bit to the last bit, represented in the figures from left to right respectively.

The following field definitions in Table apply to all of the request packet formats.

Table 4-6. General Field Definitions for All Request Packets

Field	Definition
ftype	Format type, represented as a 4-bit value; is always the first four bits in the logical packet stream.
wdptr	Word pointer, used in conjunction with the data size (rdsize and wrsize) fields—see Table , Table , and Section 3.3: "Endian, Byte Ordering, and Alignment".
rdsize	Data size for read transactions, used in conjunction with the word pointer (wdptr) bit—see Table and Section 3.3: "Endian, Byte Ordering, and Alignment".
wrsize	Write data size for sub-double-word transactions, used in conjunction with the word pointer (wdptr) bit—see Table and Section 3.3: "Endian, Byte Ordering, and Alignment". For writes greater than one double-word, the size is the maximum payload.
rsrv	Reserved

Table 4-6. General Field Definitions for All Request Packets(Continued)

Field	Definition
srcTID	The packet's transaction ID.
transaction	The specific transaction within the format class to be performed by the recipient; also called type or ttype.
extended address	Optional. Specifies the most significant 16 bits of a 50-bit physical address or 32 bits of a 66-bit physical address.
xamsbs	Extended address most significant bits. Further extends the address specified by the address and extended address fields by 2 bits. This field provides 34-, 50-, and 66-bit addresses to be specified in a packet with the xamsbs as the most significant bits in the address.
address	Least significant 29 bits (bits [0-28] of byte address [0-31]) of the double-word physical address

Table 4-7. Read Size (rdsize) Definitions

wdptr	rdsize	Number of Bytes	Byte Lanes
0b0-1	0b0000-1011	Reserved	
0b0	0b1100	32	
0b1	0b1100	64	
0b0-1	0b1101-1111	Reserved	

Table 4-8. Write Size (wrsize) Definitions

wdptr	wrsize	Number of Bytes	Byte Lanes
0b0	0b0000	1	0b10000000
0b0	0b0001	1	0b01000000
0b0	0b0010	1	0b00100000
0b0	0b0011	1	0b00010000
0b1	0b0000	1	0b00001000
0b1	0b0001	1	0b00000100
0b1	0b0010	1	0b00000010
0b1	0b0011	1	0b00000001
0b0	0b0100	2	0b11000000
0b0	0b0101	3	0b11100000
0b0	0b0110	2	0b00110000
0b0	0b0111	5	0b11111000
0b1	0b0100	2	0b00001100
0b1	0b0101	3	0b00000111
0b1	0b0110	2	0b00000011
0b1	0b0111	5	0b00011111
0b0	0b1000	4	0b11110000

Table 4-8. Write Size (wrsize) Definitions(Continued)

wdptr	wrsize	Number of Bytes	Byte Lanes
0b1	0b1000	4	0b00001111
0b0	0b1001	6	0b11111100
0b1	0b1001	6	0b00111111
0b0	0b1010	7	0b11111110
0b1	0b1010	7	0b01111111
0b0	0b1011	8	0b11111111
0b1	0b1011	16 maximum	
0b0	0b1100	32 maximum	
0b1	0b1100	64 maximum	
0b0-1	0b1101-1111	Reserved	

4.1.4 Type 0 Packet Format (Implementation-Defined)

The type 0 packet format is reserved for implementation-defined functions such as flow control.

4.1.5 Type 1 Packet Format (Intervention-Request Class)

Type 1 request packets never include data. They are the only request types that can cause an intervention, so the secondary domain, secondary ID, and secondary transaction ID fields are required. The total number of bits available for the secondary domain and secondary ID fields (shown in Figure 4-1) is determined by the size of the transport field defined in the appropriate transport layer specification, so the size (labeled *m* and *n*, respectively) of these fields are not specified. The division of the bits between the logical coherence domain and device ID fields is determined by the specific application. For example, an 8 bit transport field allows 16 coherence domains of 16 participants.

The type 1 packet format is used for the READ_OWNER, READ_TO_OWN_OWNER, and IO_READ_OWNER transactions that are specified in the transaction sub-field column defined in Table .4-9. Type 1 packets are issued only by a home memory controller to allow the third party intervention data transfer.

Definitions and encodings of fields specific to type 1 packets are displayed in Table 4-9. Fields that are not specific to type 1 packets are described in Table 4-6.

Table 4-9. Specific Field Definitions and Encodings for Type 1 Packets

Field	Encoding	Sub-Field	Definition
secID	—		Original requestor’s, or secondary, ID for intervention
secTID	—		Original requestor’s, or secondary, transaction ID for intervention
sec_domain	—		Original requestor’s, or secondary, domain for intervention
transaction	0b0000	READ_OWNER	
	0b0001	READ_TO_OWN_OWNER	
	0b0010	IO_READ_OWNER	
	0b0011–1111	Reserved	

Figure 4-1 displays a type 1 packet with all its fields. The field value 0b0001 in Figure 4-1 specifies that the packet format is of type 1.

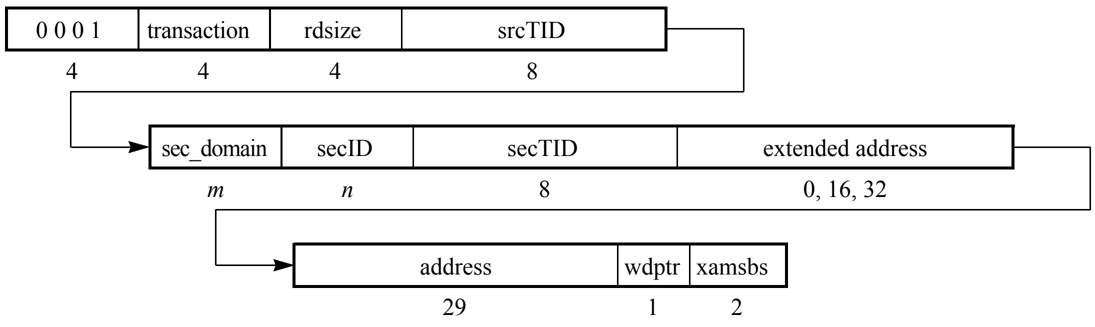


Figure 4-1. Type 1 Packet Bit Stream Format

4.1.6 Type 2 Packet Format (Request Class)

Type 2 request packets never include data. They cannot cause an intervention so the secondary domain and ID fields specified in the intervention-request format are not required. This format is used for the READ_HOME, IREAD_HOME, READ_TO_OWN_HOME, IO_READ_HOME, DKILL_HOME, DKILL_SHARER, IKILL_HOME, IKILL_SHARER, TLBIE, and TLBSYNC transactions as specified in the transaction field defined in Table . Type 2 packets for READ_HOME, IREAD_HOME, READ_TO_OWN_HOME, IO_READ_HOME, FLUSH without data, DKILL_HOME, and IKILL_HOME transactions are issued to home memory by a processing element. DKILL_SHARER and IKILL_SHARER transactions are issued by a home memory to the sharers of a coherence granule. DKILL_HOME, DKILL_SHARER, IKILL_HOME, IKILL_SHARER, FLUSH without data, and TLBIE are address-only transactions so the rdsize and wdptr fields are ignored and shall be set to logic 0. TLBSYNC is a transaction-type-only transaction so both the address, xamsbs, rdsize, and wdptr fields shall be set to logic 0.

The transaction field encodings for type 2 packets are displayed in Table 4-10. Fields that are not specific to type 2 packets are described in Table .

Transaction Field Encodings for Type 2 Packets

Encoding	Transaction Field
0b0000	READ_HOME
0b0001	READ_TO_OWN_HOME
0b0010	IO_READ_HOME
0b0011	DKILL_HOME
0b0100	Reserved
0b0101	IKILL_HOME
0b0110	TLBIE
0b0111	TLBSYNC
0b1000	IREAD_HOME
0b1001	FLUSH without data
0b1010	IKILL_SHARER
0b1011	DKILL_SHARER
0b1100–1111	Reserved

Figure 4-2 displays a type 2 packet with all its fields. The field value 0b0010 in Figure specifies that the packet format is of type 2.

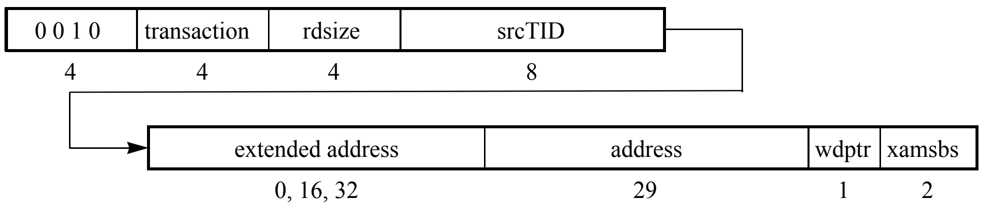


Figure 4-2. Type 2 Packet Bit Stream Format

4.1.7 Type 3–4 Packet Formats (Reserved)

The type 3–4 packet formats are reserved.

4.1.8 Type 5 Packet Format (Write Class)

Type 5 packets always contain data. A data payload that consists of a single double-word or less has sizing information as defined in Table 4-8. The wrsize field specifies the maximum size of the data payload for multiple double-word transactions. The FLUSH with data and CASTOUT transactions use type 5 packets as defined in Table 4-11. Note that type 5 transactions always contain data.

Fields that are not specific to type 5 packets are described in Table 4-6..

Table 4-11. Transaction Field Encodings for Type 5 Packets

Encoding	Transaction Field
0b0000	CASTOUT
0b0001	FLUSH with data
0b0010–1111	Reserved

Figure 4-3 displays a type 5 packet with all its fields. The field value 0b0101 in Figure 4.3 specifies that the packet format is of type 5.

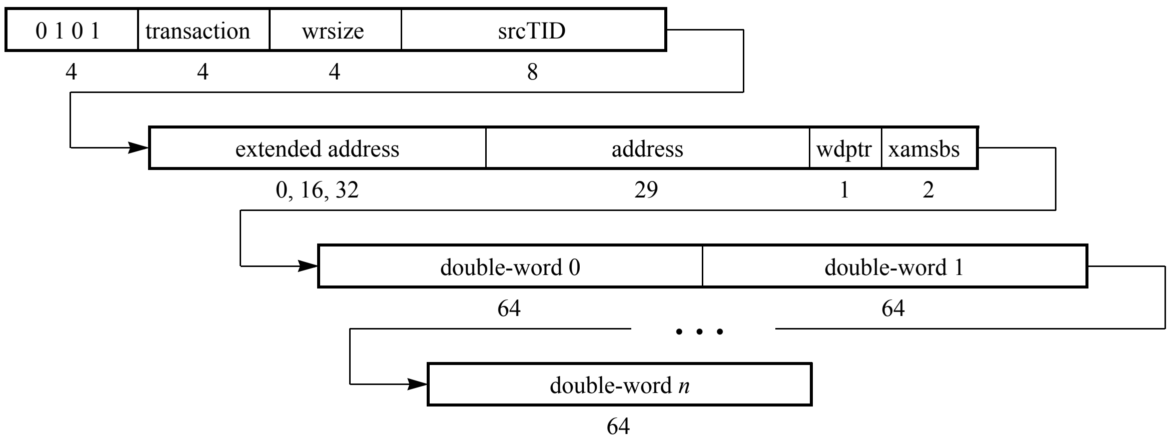


Figure 4-3. Type 5 Packet Bit Stream Format

4.1.9 Type 6–11 Packet Formats (Reserved)

The type 6–11 packet formats are reserved.

4.2 Response Packet Formats

A response transaction is issued by a processing element when it has completed a request made by a remote processing element. Response packets are always directed and are transmitted in the same way as request packets. Currently two response packet format types exist, as shown in Table .

Table 4-12. Request Packet Type to Transaction Type Cross Reference

Request Packet Format Type	Transaction Type	Definition	Document Section No.
Type 12	—	Reserved	Section 4.2.2
Type 13	RESPONSE	Issued by a processing element when it completes a request by a remote element.	Section 4.2.3
Type 14	—	Reserved	Section 4.2.4
Type 15	Implementation-defined	Defined by the device implementation	Section 4.2.5

4.2.1 Field Definitions for All Response Packet Formats

The field definitions in Table 4-13 apply to more than one of the response packet formats.

Table 4-13. Field Definitions and Encodings for All Response Packets

Field	Encoding	Sub-Field	Definition
transaction	0b0000		RESPONSE transaction with no data payload
	0b0001–0111		Reserved
	0b1000		RESPONSE transaction with data payload
	0b1001–1111		Reserved
targetTID	—		The corresponding request packet’s transaction ID
status	Type of status and encoding		
	0b0000	DONE	Requested transaction has been successfully completed
	0b0001	DATA_ONLY	This is a data only response
	0b0010	NOT_OWNER	Not owner of requested coherence granule
	0b0011	RETRY	Requested transaction is not accepted; must retry the request
	0b0100	INTERVENTION	Update home memory with intervention data
	0b0101	DONE_INTERVENTION	Done for a transaction that resulted in an intervention
	0b0110	—	Reserved
	0b0111	ERROR	Unrecoverable error detected
	0b1000–1011	—	Reserved
	0b1100–1111	Implementation	Implementation defined—Can be used for additional information such as an error code

4.2.2 Type 12 Packet Format (Reserved)

The type 12 packet format is reserved.

4.2.3 Type 13 Packet Format (Response Class)

The type 13 packet format returns status, data (if required), and the requestor’s transaction ID. A RESPONSE packet with an “ERROR” status or a response that is not expected to have a data payload never has a data payload. The type 13 format is used for response packets to all request transactions.

Note that type 13 packets do not have any special fields.

Figure 4-4 illustrates the format and fields of type 13 packets. The field value 0b1101 in Figure 4-4 specifies that the packet format is of type 13.

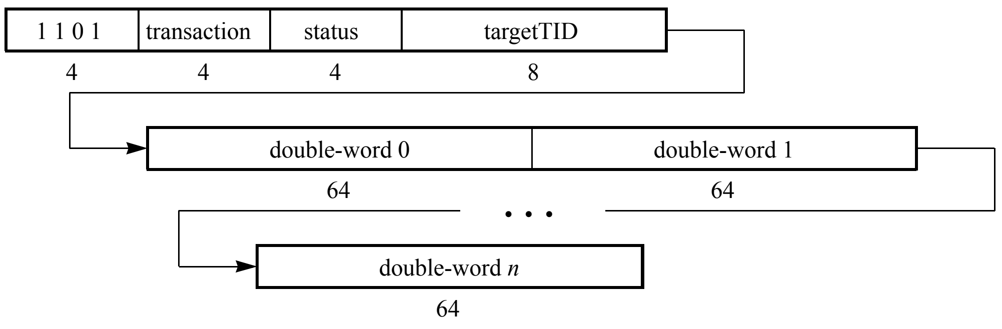


Figure 4-4. Type 13 Packet Bit Stream Format

4-2-4 Type 14 Packet Format (Reserved)

The type 14 packet format is reserved.

4-2-5 Type 15 Packet Format (Implementation-Defined)

The type 15 packet format is reserved for implementation-defined functions such as flow control.

5 Chapter 5 - Globally Shared Memory Registers

This chapter describes the visible register set that allows an external processing element to determine the capabilities, configuration, and status of a processing element using this logical specification. This chapter only describes registers or register bits defined by this specification. Refer to the other RapidIO logical, transport, and physical specifications of interest to determine a complete list of registers and bit definitions. All registers are 32-bits and aligned to a 32-bit boundary.

5.1 Register Summary

Table 5-1 shows the register map for this RapidIO specification. These capability registers (CARs) and command and status registers (CSRs) can be accessed using the *Partition I: Input/Output Logical Specification* maintenance operations. Any register offsets not defined are considered reserved for this specification unless otherwise stated. Other registers required for a processing element are defined in other applicable RapidIO specifications and by the requirements of the specific device and are beyond the scope of this specification. Read and write accesses to reserved register offsets shall terminate normally and not cause an error condition in the target device. Writes to CAR (read-only) space shall terminate normally and not cause an error condition in the target device.

Register bits defined as reserved are considered reserved for this specification only. Bits that are reserved in this specification may be defined in another RapidIO specification.

Table 5-1. GSM Register Map

Configuration Space Byte Offset	**Register Name (Word 0)**	**Register Name (Word 1)**
0x0-10	Reserved	
0x18	Source Operations CAR	Destination Operations CAR
0x20-F8	Reserved	
0x100-FFF8	Extended Features Space	
0x10000-FFFFF8	Implementation-defined Space	

5.2 Reserved Register and Bit Behavior

Table 5.2 describes the required behavior for accesses to reserved register bits and reserved registers for the RapidIO register space,

Table 5-2. Configuration Space Reserved Access Behavior

Byte Offset	Space Name	Item	Initiator behavior	Target behavior
0x0-3C	Capability Register Space (CAR Space - this space is read-only)	Reserved bit	read - ignore returned value[1]	read - return logic 0
			write -	write - ignored
		Implementation-defined bit	read - ignore returned value unless implementation-defined function understood	read - return implementation-defined value
			write -	write - ignored
		Reserved register	read - ignore returned value	read - return logic 0s
			write -	write - ignored
0x40-FC	Command and Status Register Space (CSR Space)	Reserved bit	read - ignore returned value	read - return logic 0
			write - preserve current value[2]	write - ignored
		Implementation-defined bit	read - ignore returned value unless implementation-defined function understood	read - return implementation-defined value
			write - preserve current value if implementation-defined function not understood	write - implementation-defined
		Reserved register	read - ignore returned value	read - return logic 0s
			write -	write - ignored
0x100-FFFC	Extended Features Space	Reserved bit	read - ignore returned value	read - return logic 0
			write - preserve current value	write - ignored
		Implementation-defined bit	read - ignore returned value unless implementation-defined function understood	read - return implementation-defined value
			write - preserve current value if implementation-defined function not understood	write - implementation-defined
		Reserved register	read - ignore returned value	read - return logic 0s
			write -	write - ignored
0x10000-FFFFFC	Implementation-defined Space	Reserved bit and register	All behavior implementation-defined	

1. Do not depend on reserved bits being a particular value; use appropriate masks to extract defined bits from the read value.

2. All register writes shall be in the form: read the register to obtain the values of all reserved bits, merge in the desired values for defined bits to be modified, and write the register, thus preserving the value of all reserved bits.

5.3 Capability Registers (CARs)

Every processing element shall contain a set of registers that allows an external processing element to determine its capabilities using the I/O logical maintenance read operation. All registers are 32 bits wide and are organized and accessed in 32-bit (4 byte) quantities, although some processing elements may optionally allow larger accesses. CARs are read-only. Refer to Table 5-2 for the required behavior for accesses to reserved registers and register bits.

CARs are big-endian with bit 0 and Word 0 respectively the most significant bit and word.

5.3.1 Source Operations CAR (Offset 0x18 Word 0)

This register defines the set of RapidIO GSM logical operations that can be issued by this processing element; see Table 5-3. It is assumed that a processing element can generate I/O logical maintenance read and write requests if it is required to access CARs and CSRs in other processing elements. The Source Operations CAR is applicable for end point devices only. RapidIO switches shall be able to route any packet.

Table 5-3. Bit Settings for Source Operations CAR

Bit	Field Name	Description
0	Read	PE can support a read operation
1	Instruction read	PE can support an instruction read operation
2	Read-for-ownership	PE can support a read-for-ownership operation
3	Data cache invalidate	PE can support a data cache invalidate operation
4	Castout	PE can support a castout operation
5	Data cache flush	PE can support a data cache flush operation
6	I/O read	PE can support an I/O read operation
7	Instruction cache invalidate	PE can support an instruction cache invalidate operation
8	TLB invalidate-entry	PE can support a TLB invalidate-entry operation
9	TLB invalidate-entry sync	PE can support a TLB invalidate-entry sync operation
10–13	—	Reserved
14–15	Implementation Defined	Defined by the device implementation
16–29	—	Reserved
30–31	Implementation Defined	Defined by the device implementation

5.3.2 Destination Operations CAR (Offset 0x18 Word 1)

This register defines the set of RapidIO GSM operations that can be supported by this processing element; see Table 5-4. It is required that all processing elements can respond to I/O logical maintenance read and write requests in order to access these registers. The Destination Operations CAR is applicable for end point devices only. RapidIO switches shall be able to route any packet.

Table 5-4. Bit Settings for Destination Operations CAR

Bit	Field Name	Description
0	Read	PE can support a read operation
1	Instruction read	PE can support an instruction read operation
2	Read-for-ownership	PE can support a read-for-ownership operation
3	Data cache invalidate	PE can support a data cache invalidate operation
4	Castout	PE can support a castout operation
5	Data cache flush	PE can support a flush operation
6	I/O read	PE can support an I/O read operation
7	Instruction cache invalidate	PE can support an instruction cache invalidate operation
8	TLB invalidate-entry	PE can support a TLB invalidate-entry operation

Table 5-4. Bit Settings for Destination Operations CAR(Continued)

Bit	Field Name	Description
9	TLB invalidate-entry sync	PE can support a TLB invalidate-entry sync operation
10–13	—	Reserved
14-15	Implementation Defined	Defined by the device implementation
16-29	—	Reserved
30-31	Implementation Defined	Defined by the device implementation

6 Chapter 6 - Communication Protocols

This chapter contains the RapidIO globally shared memory (GSM) communications protocol definitions. Three state machines are required for a processing element on the RapidIO interface: one for local system accesses to local and remote space, one for remote accesses to local space, and one for handling responses made by the remote system to requests from the local system. The protocols are documented as pseudo-code partitioned by operation type. The RapidIO protocols as defined here assume a directory state definition that uses a modified bit with the local processor always sharing as described in Chapter 2, "System Models." The protocols can be easily modified to use an alternate directory scheme that allows breaking the SHARED state into a REMOTE_SHARED and a REMOTE_AND_LOCAL_SHARED state pair.

Similarly, it may be desirable for an implementation to have an UNOWNED state instead of defaulting to LOCAL_SHARED or LOCAL_MODIFIED. These optimizations only affect the RapidIO transaction issuing behavior within a processing element, not the globally shared memory protocol itself. This flexibility allows a variety of local processor cache state coherence definitions such as MSI or MESI.

Some designs may not have a source of local system requests, for example, the memory only processing element described in Section 2.1.3, "Memory-Only Processing Element Model". The protocols for these devices are much less complicated, only requiring the external request state machine and a portion of the response state machine. Similarly, a design may not have a local memory controller, which is also a much less complicated device, requiring only a portion of the internal request and response state machines. The protocols assume a processor element and memory processing element as described in Figure 2-2.

6.1 Definitions

The general definitions of Section 6.1.1 apply throughout the protocol, and the requests and responses of state machines are defined in Section 6.2.1 "Request and Response Definitions".

6.1.1 General Definitions

address_collision An address match between the new request and an address currently being serviced by the state machines or some other address-based internal hazard. This frequently causes a retry of the new request.

assign_entry() Assign resources (such as a queue entry) to service a request, mark the address as able to participate in address collision detection (if appropriate), and assign a transaction ID

dataAny data associated with the transaction; this field is frequently null

directory_state The memory directory state for the address being serviced

error() Signal an error (usually through an interrupt structure) to software, usually to indicate a coherence violation problem

free_entry() Release all resources assigned to this transaction, remove it from address collision detection, and deallocate the transaction ID

localMemory local to the processing element

local_request(m,n,...) A local request to a local processor caused by an incoming external request that requires a snoop of the processor's caches local_response(m,n,.) A local response to a local request; usually indicates the cache state for the requesting processor to mark the requested data

LOCAL_RTYPEThis is the response from the local agent to the local processor in response to a local request.

LOCAL_TTYPEThis is the transaction type for a request passed from the RapidIO interconnect to a local device.

(mask <= (mask ~= received_srcid))
"Assign the mask field to the old mask field with the received ID bit cleared." This result is generated when a response to a multicast is received and it is not the last one expected.

((mask ~= (my_id OR received_id)) == 0)
"The mask field not including my ID or the received ID equals 0." This result indicates that we have received all of the expected responses to a multicast request.

(mask ~= my_id)"The sharing mask not including my ID." This result is used for multicast operations where the requestor is in the sharing list but does not need to be included in the multicast transaction because it is the source of the transaction.

(mask <= (participant_list ~= my_id))
"The sharing mask includes all participants except my ID." This result is used for the IKILL operation, which does not use the memory directory information.

(mask <= (participant_list ~= (received_srcid AND my_id)))
"The sharing mask includes all participants except the requestor's and my IDs." This result is used for the IKILL operation, which does not use the memory directory information.

(mask == received_srcid)
"The sharing mask only includes the requestor's ID." This result is used for the DKILL operation to detect a write-hit-on-shared case where the requestor has the only remote copy of the coherence granule.

original_srcid The ID of the initial requestor for a transaction, saved in the state associated with the transaction ID

received_data The response contained data

received_data_only_message
Flag set by set_received_data_only_message()

received_done_message
Flag set by set_received_done_message()

remote_request (m,n,...)
Make a request to the interconnect fabric

remote_response (m,n,...)
Send a response to the interconnect fabric

RESPONSE_TTYPE
This is the RapidIO transaction type for a response to a request

return_data() Return data to the local requesting processor, either from memory or from a interconnect fabric buffer; the source can be determined from the context

secondary_id The third party identifier for intervention responses; the processing element ID concatenated with the processing element domain.

set_received_data_only_message()
Remember that a DATA_ONLY response was received for this transaction ID

set_received_done_message()
Remember that a DONE response was received for this transaction ID

source_idThe source device identifier; the processing element ID concatenated with the processing element domain

target_idThe destination device identifier; the processing element ID concatenated with the processing element domain

TRANSACTIONThe RapidIO transaction type code for the request

update_memory()Write memory with data received from a response

update_state(m,n,...)Modify the memory directory state to reflect the new system status

6.2.1 Request and Response Definitions

Following are the formats used in the pseudocode to describe request and response transactions sent between processing elements and the formats of local requests and responses between the cache coherence controller and the local cache hierarchy and memory controllers.

6.1.2.1 System Request

The system request format is:

emote_request(TRANSACTION, target_id, source_id, secondary_id, data)

which describes the necessary RapidIO request to implement the protocol.

6.1.2.2 Local Request

The local request format is:

local_request(LOCAL_TTYPE)

that is the necessary local processor request to implement the protocol; the pseudocode assumes a generic local bus. A local request also examines the remote cache as part of the processing element's caching hierarchy. The local transactions are defined as:

DKILL Causes the processor to transition the coherence granule to invalid regardless of the current state; data is not pushed if current state is modified

IKILL Causes the processor to invalidate the coherence granule in the instruction cache

READ Causes the processor to transition the coherence granule to shared and push data if necessary

READ_LATEST Causes the processor to push data if modified but not transition the cache state

READ_TO_OWN Causes the processor to transition the coherence granule to invalid and push data

TLBIE Causes the processor to invalidate the specified translation look-aside buffer entry

TLBSYNC Causes the processor to indicate when all outstanding TLBIEs have completed

6.1.2.3 System Response

The system response format is:

remote_response(RESPONSE_TTYPE, target_id, source_id, data (opt.))

which is the proper response to implement the protocol.

6.1.2.4 Local Response

The local response format is:

local_response(LOCAL_RTYPE)

In general, a transaction ID (TID) is associated with each device ID in order to uniquely identify a request. This TID is frequently a queue index in the source processing element. These TIDs are not explicitly called out in the pseudocode below. The local responses are defined as:

EXCLUSIVEThe processor has exclusive access to the coherence granule

OK The transaction requested by the processor has or will complete properly

RETRY Causes the processor to re-issue the transaction; this response may cause a local bus spin loop until the protocol allows a different response

SHARED The processor has a shared copy of the coherence granule

6.2 Operation to Protocol Cross Reference

Table 6-1 contains a cross reference of the operations defined in the *RapidIO Interconnect Globally Shared Memory Log-*

ical Specification and their system usage.

Table 6-1. Operation to Protocol Cross Reference

Operations	Protocol
Read	Section : "6.3 Read Operations" 6.3
Instruction read	Section : "6.3 Read Operations" 6.3
Read for ownership	Section 6.5
Data cache invalidate	Section 6.6
Instruction cache invalidate	Section 6.6
Castout	Section 6.7
TLB invalidate entry	Section 6.8
TLB invalidate entry synchronize	Section 6.8
Data cache flush	Section 6.9
I/O read	Section 6.10

6.3 Read Operations

This operation is a coherent data cache read; refer to the description in Section 3.2.1: "Read Operations".

6.3.1 Internal Request State Machine

This state machine handles requests to both local and remote memory from the local processor.

```
if (address_collision)   // this is due to an external request
// in progress or a cache
local_response(RETRY);  // index hazard from a previous request
elseif (local)  // our local memory
switch (directory_state)
case LOCAL_MODIFIED:   // local modified is OK if we default
//local memory to owned
local_response(EXCLUSIVE);
return_data();
case LOCAL_SHARED,// local, owned by memory
case SHARED:  // shared local and remote
local_response(SHARED);
return_data();// keep directory state
// the way it was
case REMOTE_MODIFIED:
local_response(SHARED);
assign_entry();// this means to assign
// a transaction ID,
// usually a queue entry
remote_request(READ_OWNER, mask_id, my_id, my_id);
default:
error();
else // remote - we've got to go
// to another processing element
assign_entry();
local_response(RETRY);// can't guarantee data before a
// snoop yet
remote_request(READ_HOME, mem_id, my_id);
endif;
```

6.3.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of the local system or a third party.

```
if (my_id == mem_id == original_srcid)// original requestor is home memory
switch(remote_response)// matches my_id only for
// REMOTE_MODIFIED case
case INTERVENTION:
update_memory();
update_state(SHARED, original_srcid);
return_data();
free_entry();
case NOT_OWNER,// due to address collision or
case RETRY:// passing requests
switch (directory_state)
case LOCAL_MODIFIED:
local_response(EXCLUSIVE);
// when processor re-requests
return_data();
free_entry();
case LOCAL_SHARED:
local_response(SHARED);
// when processor re-requests
return_data();
free_entry();
case REMOTE_MODIFIED: // mask_id must match received_srcid

//or error; spin or wait for castout
remote_request(READ_OWNER, received_srcid,
my_id, my_id);
default:
error();
default
error();
elseif(my_id == mem_id ~== original_id// i'm home memory working for
//a third party
switch(remote_response)
case INTERVENTION:
update_memory();
update_state(SHARED, original_srcid);
remote_response(DONE_INTERVENTION, original_srcid, my_id);
free_entry();
case NOT_OWNER,// data comes from memory,
// mimic intervention
case RETRY:
switch(directory_state)
case LOCAL_SHARED:
update_state(SHARED, original_srcid);
remote_response(DATA_ONLY, original_srcid,
my_id, data);
emote_response(DONE_INTERVENTION, original_srcid,
my_id);
free_entry();
case LOCAL_MODIFIED:
update_state(SHARED, original_srcid);
remote_response(DATA_ONLY, original_srcid,
my_id, data);
```

```
remote_response(DONE_INTERVENTION, original_srcid,
my_id);
free_entry();
case REMOTE_MODIFIED:// spin or wait for castout
remote_request(READ_OWNER, received_srcid,
my_id, my_id);
default:
error();
default:
error();
else// my_id ~= mem_id - I'm
// requesting a remote
// memory location
switch(remote_response)
case DONE:
local_response(SHARED);  // when processor re-requests
return_data();
free_entry();
case DONE_INTERVENTION:// must be from third party
set_received_done_message();
if (received_data_only_message)
free_entry();
else
// wait for a DATA_ONLY
endif;
case DATA_ONLY: // this is due to an intervention, a
// DONE_INTERVENTION should come
// separately
local_response(SHARED);
set_received_data_only_message();
if (received_done_message)
return_data();
free_entry();
else
return_data(); // OK for weak ordering
endif;
case RETRY:
remote_request(READ_HOME, received_srcid, my_id);
default
error();
endif;
```

6.3.3 External Request State Machine

This state machine handles requests from the system to the local memory or the local system. This may require making further external requests.

```
if (address_collision)// use collision tables in
// Chapter 7, "Address Collision Resolution Tables"
elseif (READ_HOME)// remote request to our local memory
assign_entry();
switch (directory_state)
case LOCAL_MODIFIED:
local_request(READ);
update_state(SHARED, received_srcid);
//after possible push completes
remote_response(DONE, received_srcid, my_id, data);
free_entry();
```

```
case LOCAL_SHARED,
case SHARED:
update_state(SHARED, received_srcid);
remote_response(DONE, received_srcid, my_id, data);
free_entry();
case REMOTE_MODIFIED:
if (mask_id ~= received_srcid)
// intervention case
remote_request(READ_OWNER, mask_id,
my_id, received_srcid);
else
error();// he already owned it;
// cache paradox (or I-fetch after d-
// store if not fixed elsewhere)
endif;
default:
error();
else// READ_OWNER request to our caches
assign_entry();
local_request(READ);// spin until a valid response
// from caches
switch (local_response)
case MODIFIED: // processor indicated a push;
// wait for it
cache_state(SHARED or INVALID);
//surrender ownership
if (received_srcid == received_secid)
//original requestor is also home
remote_response(INTERVENTION, received_srcid,
my_id, data);
else
remote_response(DATA_ONLY, received_secid,
my_id, data);
remote_response(INTERVENTION, received_srcid,
my_id, data);
endif;
case INVALID: // must have cast it out
remote_response(NOT_OWNER, received_srcid, my_id);
default;
error();
free_entry();
endif;
```

6.4 Instruction Read Operations

This operation is a partially coherent instruction cache read; refer to the description in Section 3.2.2: "Instruction Read Operations".

6.4.1 Internal Request State Machine

This state machine handles requests to both local and remote memory from the local processor.

```
if (address_collision)// this is due to an external
// request in progress or a cache
local_response(RETRY); // index hazard from a previous request
elseif (local)// our local memory
switch (directory_state)
case LOCAL_MODIFIED: // local modified is OK if we default
// local memory to owned
```

```
local_response(EXCLUSIVE);
return_data();
case LOCAL_SHARED,// local, owned by memory
case SHARED: // shared local and remote
local_response(SHARED);
return_data();// keep directory state the way it was
case REMOTE_MODIFIED:
local_response(SHARED);
assign_entry();// this means to assign a transaction
// ID, usually a queue entry
remote_request(READ_OWNER, mask_id, my_id, my_id);
default:
error();
else// remote - we've got to go
//to another processing element
assign_entry();
local_response(RETRY);
// can't guarantee data before a
// snoop yet
remote_request(IREAD_HOME, mem_id, my_id);
endif;
```

6.4.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of the local system or a third party.

```
if (my_id == mem_id == original_srcid)// original requestor is home memory
error();
elseif(my_id == mem_id ~== original_id)// i'm home memory working for a
// third party
switch(remote_response)
case INTERVENTION:
update_memory();
update_state(SHARED, original_srcid);
remote_response(DONE, original_srcid, my_id);
free_entry();
case NOT_OWNER,// data comes from memory,
// mimic intervention
case RETRY:
switch(directory_state)
case LOCAL_SHARED:
update_state(SHARED, original_srcid);
remote_response(DONE, original_srcid, my_id);
free_entry();
case LOCAL_MODIFIED:
update_state(SHARED, original_srcid);
remote_response(DONE, original_srcid, my_id);
free_entry();
case REMOTE_MODIFIED:// spin or wait for castout
remote_request(READ_OWNER, received_srcid,
my_id, my_id);
default:
error();
default:
error();
else // my_id ~= mem_id - I'm requesting
// a remote memory location
```

```
switch(remote_response)
case DONE:
local_response(SHARED);// when processor re-requests
return_data();
free_entry();
case DONE_INTERVENTION:// must be from third party
set_received_done_message();
if (received_data_only_message)
free_entry();
else
// wait for a DATA_ONLY
endif;
case DATA_ONLY:// this is due to an intervention; a
// DONE_INTERVENTION should come
// separately
local_response(SHARED);
set_received_data_only_message();
if (received_done_message)
return_data();
free_entry();
else
return_data(); // OK for weak ordering
endif;
case RETRY:
remote_request(IREAD_HOME, received_srcid, my_id);
default
error();
endif;
```

6.4.3 External Request State Machine

This state machine handles requests from the system to the local memory or the local system. This may require making further external requests.

```
if (address_collision)// use collision tables in
// Chapter 7, "Address Collision Resolution Tables"
elseif(IREAD_HOME)// remote request to our local memory
assign_entry();
switch (directory_state)
case LOCAL_MODIFIED:
local_request(READ);
update_state(SHARED, received_srcid);
// after possible push completes
remote_response(DONE, received_srcid, my_id, data);
free_entry();
case LOCAL_SHARED,
case SHARED:
update_state(SHARED, received_srcid);
remote_response(DONE, received_srcid, my_id, data);
free_entry();
case REMOTE_MODIFIED:
if (mask_id ~= received_srcid)
// intervention case
remote_request(READ_OWNER, mask_id,
my_id, received_srcid);
else// he already owned it in his
//data cache; cache paradox case
remote_request(READ_OWNER, mask_id, my_id, my_id);
```

```
endif;
default:
error();
endif;
```

6.5 Read for Ownership Operations

This is the coherent cache store miss operation.

6.5.1 Internal Request State Machine

This state machine handles requests to both local and remote memory from the local processor.

```
if (address_collision) // this is due to an external request
// in progress or a cache index
local_response(RETRY);// hazard from a previous request
elseif (local)// our local memory
switch (directory_state
case LOCAL_MODIFIED, // local modified is OK if we
// default memory to owned locally
case LOCAL_SHARED:
local_response(EXCLUSIVE);  // give ownership to processor
return_data();
if (directory_state == LOCAL_SHARED)
update_state(LOCAL_MODIFIED)
endif;
case REMOTE_MODIFIED:// owned by another, get a copy
// and ownership
assign_entry();
local_response(RETRY);// retry
remote_request(READ_TO_OWN_OWNER, mask_id, my_id, my_id);
case SHARED:// invalidate the sharing list
assign_entry();
local_response(RETRY);   // retry
remote_request(DKILL_SHARER, (mask ~= my_id), my_id, my_id);
default:
error();
else // remote - we've got to go to another
// processing element
assign_entry();
local_response(RETRY);
remote_request(READ_TO_OWN_HOME, mem_id, my_id);
endif;
```

6.5.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of the local system or a third party.

```
if (my_id == mem_id == original_srcid)// original requestor is home memory
switch (received_response)
case DONE:// SHARED, so invalidate case
if ((mask ~= (my_id OR received_id)) == 0)
// this is the last DONE
local_response(EXCLUSIVE);
return_data();
update_state(LOCAL_MODIFIED);
free_entry();
else
mask <= (mask ~= received_srcid);
```

```
// flip the responder's shared
// bit and wait for next DONE
endif;
case NOT_OWNER:// due to address collision with
// CASTOUT or FLUSH
switch(directory_state)
case LOCAL_MODIFIED,:
local_response(EXCLUSIVE);
return_data();
free_entry();
case LOCAL_SHARED:
local_response(EXCLUSIVE);
return_data();
update_state(LOCAL_MODIFIED);
free_entry();
case REMOTE_MODIFIED:
// spin or wait for castout
remote_request(READ_TO_OWN_OWNER, mask_id,
my_id, my_id);
default:
error();
case INTERVENTION:// remotely owned
local_response(EXCLUSIVE);
return_data();
update_state(LOCAL_MODIFIED);
free_entry();
case RETRY:
switch (directory_state)
case LOCAL_MODIFIED:
local_response(EXCLUSIVE);
return_data();
free_entry();
case LOCAL_SHARED:
local_response(EXCLUSIVE);
return_data();
update_state(LOCAL_MODIFIED);
free_entry();
case REMOTE_MODIFIED: //mask_id must match received_srcid
//or error condition
remote_request(READ_TO_OWN_OWNER, received_srcid,
my_id, my_id);
case SHARED:
remote_request(DKILL_SHARER, received_srcid, my_id,
my_id);
default:
error();
default:
error();
elseif (my_id == mem_id ~= original_srcid)
// i'm home memory working
// for a third party
switch(received_response)
case DONE: // invalidates for shared
// directory states
if ((mask ~= (my_id OR received_id)) == 0)
// this is the last DONE
```

```
update_state(REMOTE_MODIFIED, original_srcid);
remote_response(DONE, original_srcid, my_id, data);
free_entry();
else
mask <= (mask ~= received_srcid);
// flip the responder's shared bit
endif;// and wait for next DONE
case INTERVENTION:
// remote_modified case
update_memory();// for possible coherence error
// recovery
update_state(REMOTE_MODIFIED, original_id);
remote_response(DONE_INTERVENTION, original_id, my_id);
free_entry();
case NOT_OWNER:// data comes from memory, mimic
// intervention
switch(directory_state)
case LOCAL_SHARED:
case LOCAL_MODIFIED:
update_state(REMOTE_MODIFIED, original_srcid);
remote_response(DATA_ONLY, original_srcid, my_id,
data);
remote_response(DONE, original_srcid, my_id);
free_entry();
case REMOTE_MODIFIED:
remote_request(READ_TO_OWN_OWNER, received_srcid,
my_id, original_srcid);
default:
error();
case RETRY:
switch (directory_state)
case LOCAL_MODIFIED,
case LOCAL_SHARED:
update_state(REMOTE_MODIFIED, original_srcid);
remote_response(DATA_ONLY, original_srcid, my_id,
data);
remote_response(DONE, original_srcid, my_id);
free_entry();
case REMOTE_MODIFIED:// mask_id must match received_srcid
// or error condition
remote_request(READ_TO_OWN_OWNER, received_srcid,
my_id, my_id);
case SHARED:
remote_request(DKILL_SHARER, received_srcid, my_id,
my_id);
default:
error();
default:
error();
else // my_id ~= mem_id - I'm requesting
// a remote memory location
switch (received_response)
case DONE:
local_response(EXCLUSIVE);
return_data();
free_entry();
```

```
case DONE_INTERVENTION:
set_received_done_message();
if (received_data_message)
free_entry();
else
// wait for DATA_ONLY
endif;
case DATA_ONLY:
set_received_data_message();
local_response(EXCLUSIVE);
if (received_done_message)
return_data();
free_entry();
else
return_data();// OK for weak ordering
endif;// and wait for a DONE
case RETRY:// lost at remote memory so retry
remote_request(READ_TO_OWN_HOME, mem_id, my_id);
default:
error();
endif;
```

6.5.3 External Request State Machine

This state machine handles requests from the interconnect to the local memory or the local system. This may require making further external requests.

```
if (address_collision)// use collision tables
// in Chapter 7, "Address Collision Resolution Tables"
elseif (READ_TO_OWN_HOME) // remote request to our local memory
assign_entry();
switch (directory_state)
case LOCAL_MODIFIED,
case LOCAL_SHARED:
local_request(READ_TO_OWN);
remote_response(DONE, received_srcid, my_id, data);
// after possible push
update_state(REMOTE_MODIFIED, received_srcid);
free_entry();
case REMOTE_MODIFIED:
if (mask_id ~= received_srcid)
// intervention case
remote_request(READ_TO_OWN_OWNER, mask_id, my_id,
received_srcid);
else
error();// he already owned it!
endif;
case SHARED:
local_request(READ_TO_OWN);
if (mask == received_srcid)
//requestor is only remote sharer
update_state(REMOTE_MODIFIED, received_srcid);
remote_response(DONE, received_srcid, my_id, data);
// from memory
free_entry();
else //there are other remote sharers
remote_request(DKILL_SHARER, (mask ~= received_srcid),
my_id, my_id);
```

```
endif;
default:
error();
elseif(READ_TO_OWN_OWNER) // request to our caches
assign_entry();
local_request(READ_TO_OWN);// spin until a valid response from
// the caches
switch (local_response)
case MODIFIED:// processor indicated a push
cache_state(INVALID);
//surrender ownership
if(received_srcid == received_secid)
// the original request is from the home
remote_response(INTERVENTION, received_srcid, my_id,
data);
else // the original request is from a
// third party
remote_response(DATA_ONLY, received_secid, my_id,
data);
remote_response(INTERVENTION, received_srcid, my_id,
data);
endif;
free_entry();
case INVALID:// castout address collision
remote_response(NOT_OWNER, received_srcid, my_id);
default:
error();
endif;
```

6.6 Data Cache and Instruction Cache Invalidate Operations

This operation is used with coherent cache store-hit-on-shared, cache operations; refer to the description in Section 3.2.4: "Data Cache Invalidate Operations".

6.6.1 Internal Request State Machine

This state machine handles requests to both local and remote memory from the local processor.

```
if (address_collision)// this is due to an external request in
// progress or a cache index
local_response(RETRY);// hazard from a previous request
elseif (local)// our local memory and we won
if (DKILL)// DKILL checks the directory
switch (directory_state)
case LOCAL_MODIFIED, // local modified is OK if we default
// memory to owned locally
case LOCAL_SHARED:
local_response(EXCLUSIVE);
if (LOCAL_SHARED)
update_state(LOCAL_MODIFIED, my_id);
endif;
case REMOTE_MODIFIED:// cache paradox; DKILL is
// write-hit-on-shared
error();
case SHARED:
local_response(RETRY);
assign_entry();// Multicast if possible otherwise
// issue direct to each sharer
remote_request(DKILL_SHARER, (mask ~= my_id), my_id);
```

```
default:
error();
else // IKILL always goes to everyone
remote_request(IKILL_SHARER,
mask <= (participant_list ~= my_id)), my_id);
endif;
else // remote - we've got to go to another
// processing element
assign_entry();
local_response(RETRY);
remote_request({DKILL_HOME, IKILL_HOME}, mem_id, my_id);
endif;
```

6.6.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of the local system or a third party.

```
if (my_id == mem_id == original_srcid)// original requestor is home memory
switch (received_response)
case DONE:// shared cases
if ((mask ~= (my_id OR received_id)) == 0)
// this is the last DONE
update_state(LOCAL_MODIFIED);
free_entry();
else
mask <= (mask ~= received_srcid);
// flip the responder's shared bit and
endif; // wait for next DONE
case RETRY:
remote_request({DKILL_SHARER, IKILL_SHARER}, received_srcid,
my_id); // retry the transaction
default:
error();
elseif (my_id == mem_id ~= original_srcid)
// i'm home memory working
// for a third party
switch(received_response)
case DONE // invalidates for shared
// directory states
if ((mask ~= (my_id OR received_id)) == 0)
// this is the last DONE
update_state(REMOTE_MODIFIED, original_srcid);
remote_response(DONE, original_srcid, my_id);
free_entry();
else
mask <= (mask ~= received_srcid);
// flip the responder's shared bit
endif;// and wait for next DONE
case RETRY:
remote_request({DKILL_SHARER, IKILL_SHARER}, received_srcid,
my_id);// retry
default:
error();
else // my_id ~= mem_id - I'm requesting
// a remote memory location
switch (received_response)
case DONE:
```

```
local_response(EXCLUSIVE);
free_entry();
case RETRY:
remote_request({DKILL_HOME, IKILL_HOME}, received_srcid,
my_id);// retry the transaction
default:
error();
endif;
```

6.6.3 External Request State Machine

This state machine handles requests from the system to the local memory or the local system. This may require making further external requests.

```
if (address_collision)   use collision tables in
// Chapter 7, “Address Collision Resolution Tables”
elseif (DKILL_HOME || IKILL_HOME)// remote request to our local memory
assign_entry();
if (DKILL_HOME)
switch (directory_state)
case LOCAL_MODIFIED,// cache paradoxes; DKILL is
// write-hit-on-shared
case LOCAL_SHARED,
case REMOTE_MODIFIED:
error();
case SHARED: // this is the right case, send
// invalidates to the sharing list
local_request(DKILL);
if (mask == received_srcid
// requestor is only remote sharer
update_state(REMOTE_MODIFIED, received_srcid);
remote_response(DONE, received_srcid, my_id);
free_entry();
else// there are other remote sharers
remote_request(DKILL_SHARER,
mask ~= received_srcid), my_id, NULL);
endif;
default:
error();
else  // KILL goes to everyone except the
// requestor
remote_request(IKILL_SHARER,
mask <= (participant_list ~=
received_srcid AND my_id), my_id);
else  // DKILL_SHARER or IKILL_SHARER to our caches
assign_entry();
local_request({READ_TO_OWN, IKILL});
// spin until a valid response from the
// caches
switch (local_response)
case SHARED,
case INVALID: // invalidating for shared cases
cache_state(INVALID);// surrender copy
remote_response(DONE, received_srcid, my_id);
free_entry();
default:
error();
endif;
```

6.7 Castout Operations

This operation is used to return ownership of a coherence granule to home memory, leaving it invalid in the cache; refer to the description in Section 3.2.5: "Castout Operations".

6.7.1 Internal Request State Machine

A castout is always done to remote memory space. A castout may require local activity to flush all caches in the hierarchy.

```
if (local) // our local memory
switch (directory_state)
case LOCAL_MODIFIED: // if the processor is doing a castout
// this is the only legal state
local_response(OK);
update_memory();
update_state(LOCAL_SHARED);
default:
error();
else // remote - we've got to go to another
// processing element
assign_entry();
local_response(OK);
remote_request(CASTOUT, mem_id, my_id, data);
endif;
```

6.7.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of the local system or a third party.

```
switch (received_response)
case DONE:
free_entry();
default:
error();
```

6.7.3 External Request State Machine

This state machine handles requests from the system to the local memory or the local system. This may require making further external requests.

```
assign_entry();
update_memory();
state_update(LOCAL_SHARED, my_id);// may be LOCAL_MODIFIED if the
//default is owned locally
remote_response(DONE, received_srcid, my_id);
free_entry();
```

6.8 TLB Invalidate Entry, TLB Invalidate Entry Synchronize Operations

These operations are used for software coherence management of the TLBs; refer to the descriptions in Section 3.2.6: "TLB Invalidate-Entry Operations" and Section 3.2.7: "TLB Invalidate-Entry Synchronization Operations".

6.8.1 Internal Request State Machine

The TLBIE and TLBSYNC transactions are always sent to all domain participants except the sender and are always to the processor not home memory.

```
assign_entry();
remote_request({TLBIE, TLBSYNC}, participant_id, my_id);
endif;
```

6.8.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of the local system. The responses are always from a coherence participant, not a home memory.

```
switch (received_response)
case DONE:
if ((mask ~= (my_id OR received_id)) == 0)
// this is the last DONE
free_entry();
else
mask <= (mask ~= received_srcid);
// flip the responder’s participant
// bit and wait for next DONE
endif;
case RETRY:
remote_request({TLBIE, TLBSYNC}, received_srcid, my_id, my_id);
default
error();
```

6.8.3 External Request State Machine

This state machine handles requests from the system to the local memory or the local system. The requests are always to the local caching hierarchy.

```
assign_entry();
local_request({TLBIE, TLBSYNC});// spin until a valid response
// from the caches
remote_response(DONE, received_srcid, my_id);
free_entry();
```

6.9 Data Cache Flush Operations

This operation returns ownership of a coherence granule to home memory and performs a coherent write; refer to the description in Section 3.2.9: "Data Cache Flush Operations".

6.9.1 Internal Request State Machine

This state machine handles requests to both local and remote memory from the local processor.

```
if (address_collision)// this is due to an external
// request in progress or a cache index
local_response(RETRY);// hazard from a previous request
else if (local) // our local memory
switch (directory_state)
case LOCAL_MODIFIED,
case LOCAL_SHARED:
local_response(OK);
update_memory();
case REMOTE_MODIFIED:
assign_entry();
remote_request(READ_TO_OWN_OWNER, mask_id, my_id, my_id);
case SHARED:
assign_entry();
remote_request(DKILL_SHARER, (mask ~= my_id), my_id);
default:
error();
else // remote - we’ve got to go to
// another processing element
assign_entry();
remote_request(FLUSH, mem_id, my_id, data);
```

```
// data is optional
endif;
```

6.9.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of the local system or a third party.

```
if (my_id == mem_id == original_srcid)// original requestor is home memory
switch (received_response)
case DONE:
if ((mask ~= (my_id OR received_id)) == 0)
// this is the last DONE
if (received_data)
// with local request or response
update_memory();
endif;
update_state(LOCAL_SHARED);// or LOCAL_MODIFIED
local_response(OK);
free_entry();
else
mask <= (mask ~= received_srcid);
// flip responder's shared bit
endif;// and wait for next DONE
case NOT_OWNER:
switch(directory_state)
case LOCAL_SHARED,
case LOCAL_MODIFIED:
if (received_data)
// with local request from memory
update_memory();
endif;
update_state(LOCAL_SHARED);// or LOCAL_MODIFIED
local_response(OK);
free_entry();
case REMOTE_MODIFIED:
remote_request(READ_TO_OWN_OWNER, mask_id, my_id, my_id);
default:
error();
case RETRY:
switch (directory_state)
case LOCAL_MODIFIED,
case LOCAL_SHARED:
if (received_data)
// with local request
update_memory();
// if there was some write data
endif;
update_state(LOCAL_SHARED);// or LOCAL_MODIFIED
local_response(OK);
free_entry();
case REMOTE_MODIFIED: // mask_id must match
// received_srcid or error
remote_request(READ_TO_OWN_OWNER, received_srcid,
my_id, my_id);
case SHARED:
remote_request(DKILL_SHARER, received_srcid, my_id,
my_id);
```

```
default:
error();
default:
error();
elseif (my_id == mem_id ~= original_srcid)
// i'm home memory working for a third
// party
switch(received_response)
case DONE:// invalidates for shared directory
// states
if ((mask ~= (my_id OR received_id)) == 0)
// this is the last DONE
remote_response(DONE, original_srcid, my_id, my_id);
if (received_data)
// with original request or response
update_memory();
endif;
update_state(LOCAL_SHARED);// or LOCAL_MODIFIED
free_entry();
else
mask <= (mask ~= received_srcid);
// flip responder's shared bit
endif; //and wait for next DONE
case NOT_OWNER:
switch(directory_state)
case LOCAL_SHARED,
case LOCAL_MODIFIED:
remote_response(DONE, original_srcid, my_id);
if (received_data)
// with original request
update_memory();
endif;
free_entry();
case REMOTE_MODIFIED:
remote_request(READ_TO_OWN_OWNER, received_srcid,
my_id, my_id);
default:
error();
case RETRY:
switch(directory_state)
case LOCAL_SHARED,
case LOCAL_MODIFIED:
remote_response(DONE, original_srcid, my_id);
if (received_data)
// with original request
update_memory();
endif;
free_entry();
case REMOTE_MODIFIED:
remote_request(READ_TO_OWN_OWNER, received_srcid,
my_id, my_id);
case SHARED:
remote_request(DKILL_SHARER, received_srcid, my_id);
default:
error();
default:
```

```
error();
else   // my_id ~= mem_id - I'm requesting
// a remote memory location
switch (received_response)
case DONE:
local_response(OK);
free_entry();
case RETRY:
remote_request(FLUSH, received_srcid, my_id, data);
// data is optional
default:
error();
endif;
```

6.9.3 External Request State Machine

This state machine handles requests from the system to the local memory or the local system. This may require making further external requests.

```
if (address_collision)// use collision table in
// Chapter 7, "Address Collision Resolution Tables"
elseif (FLUSH) // remote request to our local memory
assign_entry();
switch (directory_state)
case LOCAL_MODIFIED,
case LOCAL_SHARED:
local_request(READ_TO_OWN);
remote_response(DONE, received_srcid, my_id);
// after snoop completes
if (received_data)// from request or local response
update_memory();
endif;
update_state(LOCAL_SHARED, my_id);
// or LOCAL_MODIFIED
free_entry();
case REMOTE_MODIFIED:
if  (mask_id ~= received_srcid) // owned elsewhere
remote_request(READ_TO_OWN_OWNER, mask_id, my_id,
my_id); // secondary TID is a don't care since data is
// not forwarded to original requestor
else // requestor owned it; shouldn't
// generate a flush
error();
endif;
case SHARED:
local_request(READ_TO_OWN);
if (mask == received_srcid)  // requestor is only remote sharer
remote_response(DONE, received_srcid, my_id);
// after snoop completes
if (received_data)// from request or response
update_memory();
endif;
update_state(LOCAL_SHARED, my_id); // or LOCAL_MODIFIED
free_entry();
else//there are other remote sharers
remote_request(DKILL_SHARER, (mask ~= received_srcid), my_id,
my_id);
endif;
```

```
default:
error();
endif;
```

6.10 I/O Read Operations

This operation is used for I/O reads of globally shared memory space; refer to the description in Section 3.2.10: "I/O Read Operations".

6.10.1 Internal Request State Machine

This state machine handles requests to both local and remote memory from the local processor.

```
if (address_collision)   // this is due to an external request
// in progress or a cache index hazard
local_response(RETRY);   // from a previous request
elseif (local)   // our local memory
local_response(OK);
switch (directory_state)
case LOCAL_MODIFIED: // local modified is OK if we default
// local memory to owned
local_request(READ_LATEST);
return_data()// after possible push
case LOCAL_SHARED,
case SHARED:
return_data(); // keep directory state the way it was
case REMOTE_MODIFIED:
assign_entry();
remote_request(IO_READ_OWNER, mask_id, my_id, my_id);
default:
error();
else   // remote - we've got to go to
 // another processing element
assign_entry();
local_response(OK);
remote_request(IO_READ_HOME, mem_id, my_id);
endif;
```

6.10.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of the local system or a third party.

```
if (my_id == mem_id == original_srcid)
// original requestor is home memory
switch(remote_response)// matches my_id only for
// REMOTE_MODIFIED case
case INTERVENTION:
return_data();
free_entry();
case NOT_OWNER,// due to address collision or
// passing requests
case RETRY:
switch (directory_state)
case LOCAL_MODIFIED:
case LOCAL_SHARED
return_data();
free_entry();
case REMOTE_MODIFIED: // mask_id must match received_srcid or
// error; spin or wait for castout
```

```
remote_request(IO_READ_OWNER, received_srcid, my_id,
my_id);
default:
error();
default
error();
elseif(my_id == mem_id ~== original_id)  // i’m home memory working for a third
  // party
switch(remote_response)
case INTERVENTION:
update_memory();
remote_response(DONE_INTERVENTION, original_srcid, my_id);
free_entry();
case NOT_OWNER,  // data comes from memory, mimic
  // intervention
case RETRY:
switch(directory_state)
case LOCAL_MODIFIED,
case LOCAL_SHARED:
remote_response(DATA_ONLY, original_srcid, my_id,
data);
remote_response(DONE_INTERVENTION, original_srcid,
my_id);
free_entry();
case REMOTE_MODIFIED:// spin or wait for castout
remote_request(IO_READ_OWNER, received_srcid, my_id,
my_id);
default:
error();
default:
error();
else // my_id ~= mem_id - I’m requesting a
// remote memory location
switch(remote_response)
case DONE:
return_data();
free_entry();
case DONE_INTERVENTION:// must be from third party
set_received_done_message();
if (received_data_only_message)
free_entry();
else
// wait for a DATA_ONLY
endif;
case DATA_ONLY:// this is due to an intervention, a
// DONE_INTERVENTION should come
// separately
set_received_data_only_message();
if (received_done_message)
return_data();
free_entry();
else
return_data(); // OK for weak ordering
endif;
case RETRY:
remote_request(IO_READ_HOME, received_srcid, my_id);
```

```
default
error();
endif;
```

6.10.3 External Request State Machine

This state machine handles requests from the system to the local memory or the local system. This may require making further external requests.

```
if (address_collision) // use collision tables in
// Chapter 7, “Address Collision Resolution Tables”
elseif (IO_READ_HOME)  // remote request to our local memory
assign_entry();
switch (directory_state)
case LOCAL_MODIFIED:
local_request(READ_LATEST);
remote_response(DONE, received_srcid, my_id, data);
// after push completes
free_entry();
case LOCAL_SHARED:
remote_response(DONE, received_srcid, my_id, data);
free_entry();
case REMOTE_MODIFIED:
remote_request(IO_READ_OWNER, mask_id, my_id, received_srcid);
case SHARED:
remote_response(DONE, received_srcid, my_id, data);
free_entry();
default:
error();
else  // IO_READ_OWNER request to our caches
assign_entry();
local_request(READ_LATEST);// spin until a valid response from
// the caches
switch (local_response)
case MODIFIED:// processor indicated a push;
// wait for it
if (received_srcid == received_secid)
// original requestor is also home
// memory
remote_response(INTERVENTION, received_srcid, my_id,
data);
else
remote_response(DATA_ONLY, received_secid, my_id,
data);
remote_response(INTERVENTION, received_srcid, my_id);
endif;
case INVALID:   // must have cast it out during
// an address collision
remote_response(NOT_OWNER, received_srcid, my_id);
default:
error();
free_entry();
endif;
```

7 Chapter 7 - Address Collision Resolution Tables

Address collisions are conflicts between incoming cache coherence requests to a processing element and outstanding cache coherence requests within it. A collision is usually due to a match between the associated addresses, but also may be because of a conflict for some internal resource such as a cache index. Within a processing element, actions taken in response to an address collision vary depending upon the outstanding request and the incoming request. These actions are described in Table 7-1 through Table .7-17. Non-cache coherent transactions (transactions specified in other RapidIO logical specifications) do not cause address collisions.

Some of the table entries specify that an outstanding request should be canceled at the local processor and that the incoming transaction then be issued immediately to the processor. This choosing between transactions is necessary to prevent deadlock conditions between multiple processing elements vying for ownership of a coherence granule.

7.1 Resolving an Outstanding READ_HOME Transaction

Table 7-1 describes the address collision resolution for an incoming transaction that collides with an outstanding READ_HOME transaction.

Table 7-1. Address Collision Resolution for READ_HOME

Outstanding Request	Incoming Request	Resolution
READ_HOME	READ_HOME	Generate “ERROR” response
READ_HOME	IREAD_HOME	Generate “ERROR” response
READ_HOME	READ_OWNER	Generate “NOT_OWNER” response
READ_HOME	READ_TO_OWN_HOME	Generate “ERROR” response
READ_HOME	READ_TO_OWN_OWNER	Generate “NOT_OWNER” response
READ_HOME	DKILL_HOME	Generate “ERROR” response
READ_HOME	DKILL_SHARER	If outstanding request, wait for all expected responses. If final response is “DONE”, return data if necessary and forward DKILL_SHARER to processor then generate a “DONE” response. If final response is “RETRY”, cancel the read at the processor and forward DKILL_SHARED to processor then generate a “DONE” response If no outstanding request, cancel the read at the processor and forward DKILL_SHARER to processor then generate a “DONE” response (this case should be very rare).
READ_HOME	CASTOUT	Generate “ERROR” response
READ_HOME	TLBIE	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
READ_HOME	TLBSYNC	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
READ_HOME	IKILL_HOME	Generate “ERROR” response
READ_HOME	IKILL_SHARER	No collision, forward to processor then generate “DONE” response (software must maintain instruction cache coherence)
READ_HOME	FLUSH	Generate “ERROR” response

Table 7-1. Address Collision Resolution for READ_HOME(Continued)

Outstanding Request	Incoming Request	Resolution
READ_HOME	IO_READ_HOME	Generate "ERROR" response
READ_HOME	IO_READ_OWNER	Generate "NOT_OWNER" response

7.2 Resolving an Outstanding IREAD_HOME Transaction

Table 7-2 describes the address collision resolution for an incoming transaction that collides with an outstanding IREAD_HOME transaction.

Table 7-2. Address Collision Resolution for IREAD_HOME

Outstanding Request	Incoming Request	Resolution
IREAD_HOME	READ_HOME	Generate "ERROR" response
IREAD_HOME	IREAD_HOME	Generate "ERROR" response
IREAD_HOME	READ_OWNER	No collision, forward to processor then generate "DONE" response (software must maintain instruction cache coherence)
IREAD_HOME	READ_TO_OWN_HOME	Generate "ERROR" response
IREAD_HOME	READ_TO_OWN_OWNER	No collision, forward to processor then generate "DONE" response (software must maintain instruction cache coherence)
IREAD_HOME	DKILL_HOME	Generate "ERROR" response
IREAD_HOME	DKILL_SHARER	No collision, forward to processor then generate "DONE" response (software must maintain instruction cache coherence)
IREAD_HOME	CASTOUT	Generate "ERROR" response
IREAD_HOME	TLBIE	No collision, forward to processor then generate "DONE" response (software must maintain TLB entry coherence)
IREAD_HOME	TLBSYNC	No collision, forward to processor then generate "DONE" response (software must maintain TLB entry coherence)
IREAD_HOME	IKILL_HOME	Generate "ERROR" response
IREAD_HOME	IKILL_SHARER	No collision, forward to processor then generate "DONE" response (software must maintain instruction cache coherence)
IREAD_HOME	FLUSH	Generate "ERROR" response
IREAD_HOME	IO_READ_HOME	Generate "ERROR" response
IREAD_HOME	IO_READ_OWNER	No collision, forward to processor then generate "DONE" response (software must maintain instruction cache coherence)

7.3 Resolving an Outstanding READ_OWNER Transaction

Table 7.3 describes the address collision resolution for an incoming transaction that collides with an outstanding

READ_OWNER transaction.

Table 7-3. Address Collision Resolution for READ_OWNER

Outstanding Request	Incoming Request	Resolution
READ_OWNER	READ_HOME	Generate “RETRY” response
READ_OWNER	IREAD_HOME	Generate “RETRY” response
READ_OWNER	READ_OWNER	Generate “ERROR” response
READ_OWNER	READ_TO_OWN_HOME	Generate “RETRY” response
READ_OWNER	READ_TO_OWN_OWNER	Generate “ERROR” response
READ_OWNER	DKILL_HOME	Generate “RETRY” response
READ_OWNER	DKILL_SHARER	Generate “ERROR” response
READ_OWNER	CASTOUT	No collision, update directory state, generate “DONE” response (CASTOUT bypasses address collision detection)
READ_OWNER	TLBIE	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
READ_OWNER	TLBSYNC	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
READ_OWNER	IKILL_HOME	No collision, forward to processor, send IKILL_SHARER to all participants except requestor (software must maintain instruction cache coherence)
READ_OWNER	IKILL_SHARER	Generate “ERROR” response
READ_OWNER	FLUSH	Generate “RETRY” response
READ_OWNER	IO_READ_HOME	Generate “RETRY” response
READ_OWNER	IO_READ_OWNER	Generate “ERROR” response

7.4 Resolving an Outstanding READ_TO_OWN_HOME Transaction

Table 7.4 describes the address collision resolution for an incoming transaction that collides with an outstanding READ_TO_OWN_HOME transaction.

Table 7.4. Address Collision Resolution for READ_TO_OWN_HOME

Outstanding Request	Incoming Request	Resolution
READ_TO_OWN_HOME	READ_HOME	Generate “ERROR” response
READ_TO_OWN_HOME	IREAD_HOME	Generate “ERROR” response

Table 7.4. Address Collision Resolution for READ_TO_OWN_HOME(Continued)

Outstanding Request	Incoming Request	Resolution
READ_TO_OWN_HOME	READ_OWNER	If outstanding request, wait for all expected responses. If final response is “DONE”, return data if necessary and forward READ_OWNER to processor and generate an “DONE_INTERVENTION” with data response and a “DATA_ONLY” to originator as in Section 3.2.1: "Read Operations". If final response is “RETRY” generate an “ERROR” response If no outstanding request generate an “NOT_OWNER” response.
READ_TO_OWN_HOME	READ_TO_OWN_HOME	Generate “ERROR” response
READ_TO_OWN_HOME	READ_TO_OWN_OWNER	If outstanding request, wait for all expected responses. If final response is “DONE”, return data if necessary and forward READ_TO_OWN_OWNER to processor and generate an “DONE_INTERVENTION” with data response and a “DATA_ONLY” to originator as in Section 3.2.3: "Read-for-Ownership Operations". If final response is “RETRY” generate an “ERROR” response
READ_TO_OWN_HOME	DKILL_HOME	Generate “ERROR” response
READ_TO_OWN_HOME	DKILL_SHARER	If outstanding request, wait for all expected responses. If final response is “DONE” generate an “ERROR” response (we own the coherence granule and should never see a DKILL). If final response is “RETRY” generate a “DONE” response and continue the READ_TO_OWN_HOME. If no outstanding request generate a “DONE” response.
READ_TO_OWN_HOME	CASTOUT	Generate “ERROR” response
READ_TO_OWN_HOME	TLBIE	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
READ_TO_OWN_HOME	TLBSYNC	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
READ_TO_OWN_HOME	IKILL_HOME	Generate “ERROR” response
READ_TO_OWN_HOME	IKILL_SHARER	No collision, forward to processor then generate “DONE” response (software must maintain instruction cache coherence)

Table 7.4. Address Collision Resolution for READ_TO_OWN_HOME(Continued)

Outstanding Request	Incoming Request	Resolution
READ_TO_OWN_HOME	FLUSH	If outstanding request, wait for all expected responses. If final response is “DONE”, return data if necessary and forward FLUSH to processor and generate a “DONE” with data response as in Section 3.2.9: "Data Cache Flush Operations". If final response is “RETRY” generate an “ERROR” response (we didn’t own the data and we lost at home memory) If no outstanding request generate an “ERROR” response (we didn’t own the data).
READ_TO_OWN_HOME	IO_READ_HOME	Generate “ERROR” response
READ_TO_OWN_HOME	IO_READ_OWNER	If outstanding request, wait for all expected responses. If final response is “DONE”, return data if necessary and forward IO_READ_OWNER to processor then generate a “DONE” with data response, etc. as in Section 3.2.10: "I/O Read Operations". If final response is “RETRY” generate an “ERROR” response (we didn’t own the data and we lost at home memory) If no outstanding request generate an “NOT_OWNER” response.

7-5 Resolving an Outstanding READ_TO_OWN_OWNER Transaction

Table 7-5 describes the address collision resolution for an incoming transaction that collides with an outstanding READ_TO_OWN_OWNER transaction.

Table 7-5. Address Collision Resolution for READ_TO_OWN_OWNER

Outstanding Request	Incoming Request	Resolution
READ_TO_OWN_OWNER	READ_HOME	Generate “RETRY” response
READ_TO_OWN_OWNER	IREAD_HOME	Generate “RETRY” response
READ_TO_OWN_OWNER	READ_OWNER	Generate “ERROR” response
READ_TO_OWN_OWNER	READ_TO_OWN_HOME	Generate “RETRY” response
READ_TO_OWN_OWNER	READ_TO_OWN_OWNER	Generate “ERROR” response
READ_TO_OWN_OWNER	DKILL_HOME	Generate “RETRY” response
READ_TO_OWN_OWNER	DKILL_SHARER	Generate “ERROR” response
READ_TO_OWN_OWNER	CASTOUT	No collision, update directory state, generate “DONE” response (CASTOUT bypasses address collision detection)
READ_TO_OWN_OWNER	TLBIE	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
READ_TO_OWN_OWNER	TLBSYNC	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)

Table 7-5. Address Collision Resolution for READ_TO_OWN_OWNER(Continued)

Outstanding Request	Incoming Request	Resolution
READ_TO_OWN_OWNER	IKILL_HOME	No collision, forward to processor, send IKILL_SHARER to all participants except requestor (software must maintain instruction cache coherence)
READ_TO_OWN_OWNER	IKILL_SHARER	Generate “ERROR” response
READ_TO_OWN_OWNER	FLUSH	Generate “RETRY” response
READ_TO_OWN_OWNER	IO_READ_HOME	Generate “RETRY” response
READ_TO_OWN_OWNER	IO_READ_OWNER	Generate “ERROR” response

7.6 Resolving an Outstanding DKILL_HOME Transaction

Table 7-6 describes the address collision resolution for an incoming transaction that collides with an outstanding DKILL_HOME transaction.

Table 7-6. Address Collision Resolution for DKILL_HOME

Outstanding Request	Incoming Request	Resolution
DKILL_HOME	READ_HOME	Generate “ERROR” response
DKILL_HOME	IREAD_HOME	Generate “ERROR” response
DKILL_HOME	READ_OWNER	If outstanding request, wait for all expected responses. If final response is “DONE”, return data if necessary and forward READ_OWNER to processor and generate a “DONE_INTERVENTION” with data response and a “DATA_ONLY” to originator as in Section 3.2.1: "Read Operations". If final response is “RETRY” generate an “ERROR” response (we didn’t own the data and we lost at home memory) If no outstanding request generate an “ERROR” response (we didn’t own the data).
DKILL_HOME	READ_TO_OWN_HOME	Generate “ERROR” response
DKILL_HOME	READ_TO_OWN_OWNER	If outstanding request, wait for all expected responses. If final response is “DONE” forward READ_TO_OWN_OWNER to processor and generate a “DONE_INTERVENTION” with data response and a “DATA_ONLY” to originator as in Section 3.2.3: "Read-for-Ownership Operations". If final response is “RETRY” generate an “ERROR” response (we didn’t own the data and we lost at home memory) If no outstanding request generate an “ERROR” response (we didn’t own the data).
DKILL_HOME	DKILL_HOME	Generate “ERROR” response

Table 7-6. Address Collision Resolution for DKILL_HOME(Continued)

Outstanding Request	Incoming Request	Resolution
DKILL_HOME	DKILL_SHARER	If outstanding request, wait for all expected responses. If final response is “DONE” generate an “ERROR” response (we should never see a DKILL_SHARER if we own the coherence granule). If final response is “RETRY” cancel the data cache invalidate at the processor and forward DKILL_SHARER to processor then generate a “DONE” response If no outstanding request, cancel the data cache invalidate at the processor and forward DKILL_SHARER to processor then generate a “DONE” response.
DKILL_HOME	CASTOUT	Generate “ERROR” response (cache paradox, can’t have a SHARED granule also MODIFIED in another processing element)
DKILL_HOME	TLBIE	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
DKILL_HOME	TLBSYNC	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
DKILL_HOME	IKILL_HOME	Generate “ERROR” response
DKILL_HOME	IKILL_SHARER	No collision, forward to processor then generate “DONE” response (software must maintain instruction cache coherence)
DKILL_HOME	FLUSH	Generate “ERROR” response
DKILL_HOME	IO_READ_HOME	Generate “ERROR” response
DKILL_HOME	IO_READ_OWNER	If outstanding request, wait for all expected responses. If final response is “DONE” forward IO_READ_OWNER to processor then generate a “DONE” with data response, etc. as in Section 3.2.10: "I/O Read Operations". If final response is “RETRY” generate an “ERROR” response (we didn’t own the data and we lost at home memory) If no outstanding request generate an “ERROR” response (we didn’t own the data).

7.7 Resolving an Outstanding DKILL_SHARER Transaction

Table 7-7 describes the address collision resolution for an incoming transaction that collides with an outstanding DKILL_SHARER transaction.

Table 7-7Address Collision Resolution for DKILL_SHARER

Outstanding Request	Incoming Request	Resolution
DKILL_SHARER	READ_HOME	Generate “RETRY” response
DKILL_SHARER	IREAD_HOME	Generate “RETRY” response
DKILL_SHARER	READ_OWNER	Generate “ERROR” response

Table 7-7Address Collision Resolution for DKILL_SHARER(Continued)

Outstanding Request	Incoming Request	Resolution
DKILL_SHARER	READ_TO_OWN_HOME	Generate “RETRY” response
DKILL_SHARER	READ_TO_OWN_OWNER	Generate “ERROR” response
DKILL_SHARER	DKILL_HOME	Generate “RETRY” response
DKILL_SHARER	DKILL_SHARER	Generate “ERROR” response
DKILL_SHARER	CASTOUT	Generate “ERROR” response (cache paradox, can’t have a SHARED granule also MODIFIED in another processing element)
DKILL_SHARER	TLBIE	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
DKILL_SHARER	TLBSYNC	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
DKILL_SHARER	IKILL_HOME	No collision, forward to processor, send IKILL_SHARER to all participants except requestor (software must maintain instruction cache coherence)
DKILL_SHARER	IKILL_SHARER	Generate “ERROR” response
DKILL_SHARER	FLUSH	Generate “RETRY” response
DKILL_SHARER	IO_READ_HOME	If processing element is HOME: generate a “RETRY” response If processing element is not HOME: If outstanding request, wait for all expected responses. If final response is “DONE” forward IO_READ to processor then generate a “DONE” with data response, etc. as in Section 3.2.10: "I/O Read Operations". If final response is “RETRY” generate an “ERROR” response (we didn’t own the data and we lost at home memory) If no outstanding request generate an “ERROR” response (we didn’t own the data).
DKILL_SHARER	IO_READ_OWNER	Generate “ERROR” response

7.8 Resolving an Outstanding IKILL_HOME Transaction

Table 7-8 describes the address collision resolution for an incoming transaction that collides with an outstanding IKILL_HOME transaction.

Table 7-8. Address Collision Resolution for IKILL_HOME

Outstanding Request	Incoming Request	Resolution
IKILL_HOME	READ_HOME	Generate “ERROR” response
IKILL_HOME	IREAD_HOME	Generate “ERROR” response
IKILL_HOME	READ_OWNER	No collision, process normally
IKILL_HOME	READ_TO_OWN_HOME	Generate “ERROR” response
IKILL_HOME	READ_TO_OWN_OWNER	No collision, process normally

Table 7-8. Address Collision Resolution for IKILL_HOME(Continued)

Outstanding Request	Incoming Request	Resolution
IKILL_HOME	DKILL_HOME	Generate “ERROR” response
IKILL_HOME	DKILL_SHARER	No collision, process normally
IKILL_HOME	CASTOUT	No collision, process normally
IKILL_HOME	TLBIE	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
IKILL_HOME	TLBSYNC	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
IKILL_HOME	IKILL_HOME	Generate “ERROR” response
IKILL_HOME	IKILL_SHARER	No collision, forward to processor then generate “DONE” response (software must maintain instruction cache coherence)
IKILL_HOME	FLUSH	Generate “ERROR” response
IKILL_HOME	IO_READ_HOME	Generate “ERROR” response
IKILL_HOME	IO_READ_OWNER	No collision, process normally

7.9 Resolving an Outstanding IKILL_SHARER Transaction

Table 7-9 describes the address collision resolution for an incoming transaction that collides with an outstanding IKILL_SHARER transaction.

Table 7-9. Address Collision Resolution for IKILL_SHARER

Outstanding Request	Incoming Request	Resolution
IKILL_SHARER	READ_HOME	No collision, process normally
IKILL_SHARER	IREAD_HOME	No collision, process normally
IKILL_SHARER	READ_OWNER	Generate “ERROR” response
IKILL_SHARER	READ_TO_OWN_HOME	No collision, process normally
IKILL_SHARER	READ_TO_OWN_OWNER	Generate “ERROR” response
IKILL_SHARER	DKILL_HOME	No collision, process normally
IKILL_SHARER	DKILL_SHARER	Generate “ERROR” response
IKILL_SHARER	CASTOUT	No collision, process normally
IKILL_SHARER	TLBIE	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
IKILL_SHARER	TLBSYNC	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
IKILL_SHARER	IKILL_HOME	No collision, forward to processor, send IKILL_SHARER to all participants except requestor (software must maintain instruction cache coherence)
IKILL_SHARER	IKILL_SHARER	Generate “ERROR” response

Table 7-9. Address Collision Resolution for IKILL_SHARER(Continued)

Outstanding Request	Incoming Request	Resolution
IKILL_SHARER	FLUSH	No collision, process normally
IKILL_SHARER	IO_READ_HOME	If processing element is HOME: generate a “RETRY” response If processing element is not HOME: If outstanding request, wait for all expected responses. If final response is “DONE” forward IO_READ to processor then generate a “DONE” with data response, etc. as in Section 3.2.10: "I/O Read Operations". If final response is “RETRY” generate an “ERROR” response (we didn’t own the data and we lost at home memory) If no outstanding request generate an “ERROR” response (we didn’t own the data).
IKILL_SHARER	IO_READ_OWNER	Generate “ERROR” response

7.10 Resolving an Outstanding CASTOUT Transaction

Table 7-10 describes the address collision resolution for an incoming transaction that collides with an outstanding CASTOUT transaction.

Table 7-10. Address Collision Resolution for CASTOUT

Outstanding Request	Incoming Request	Resolution
CASTOUT	READ_HOME	Generate “ERROR” response
CASTOUT	IREAD_HOME	Generate “ERROR” response
CASTOUT	READ_OWNER	Generate “RETRY” response; the CASTOUT will bypass address collision at home memory and modify the directory state
CASTOUT	READ_TO_OWN_HOME	Generate “ERROR” response
CASTOUT	READ_TO_OWN_OWNER	Generate “RETRY” response; the CASTOUT will bypass address collision at home memory and modify the directory state
CASTOUT	DKILL_HOME	Generate “ERROR” response
CASTOUT	DKILL_SHARER	Generate “ERROR” response
CASTOUT	CASTOUT	Generate “ERROR” response
CASTOUT	TLBIE	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
CASTOUT	TLBSYNC	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
CASTOUT	IKILL_HOME	Generate “ERROR” response
CASTOUT	IKILL_SHARER	No collision, forward to processor then generate “DONE” response (software must maintain instruction cache coherence)
CASTOUT	FLUSH	Generate “ERROR” response

Table 7-10. Address Collision Resolution for CASTOUT(Continued)

Outstanding Request	Incoming Request	Resolution
CASTOUT	IO_READ_HOME	Generate “ERROR” response
CASTOUT	IO_READ_OWNER	Generate “RETRY” response; the CASTOUT will bypass address collision at home memory and modify the directory state

7.11 Resolving an Outstanding TLBIE or TLBSYNC Transaction

Table 7.11 describes the address collision resolution for an incoming transaction that collides with an outstanding TLBIE or TLBSYNC transaction.

Table 7-11. Address Collision Resolution for Software Coherence Operations

Outstanding Request	Incoming Request	Resolution
TLBIE, TLBSYNC	ANY	No collision, process request as described in Chapter 6, “Communication Protocols”

7.12 Resolving an Outstanding FLUSH Transaction

The flush operation has two distinct versions. The first is for processing elements that participate in the coherence protocol such as a processor and it’s associated agent, which may also have a local I/O device. The second is for processing elements that do not participate in the coherence protocols such as a pure I/O device that does not have a corresponding bit in the directory sharing mask. Table 7-12 describes the address collision resolution for an incoming transaction that collides with an outstanding participant FLUSH transaction.

Table 7-12. Address Collision Resolution for Participant FLUSH

Outstanding Request	Incoming Request	Resolution
FLUSH	READ_HOME	Generate “ERROR” response
FLUSH	IREAD_HOME	Generate “ERROR” response
FLUSH	READ_OWNER	Generate “NOT_OWNER” response (we are not allowed to issue FLUSH to an owned coherence granule - should be a CASTOUT)
FLUSH	READ_TO_OWN_HOME	Generate “ERROR” response
FLUSH	READ_TO_OWN_OWNER	Generate “NOT_OWNER” response (we are not allowed to issue FLUSH to an owned coherence granule - should be a CASTOUT)
FLUSH	DKILL_HOME	Generate “ERROR” response
FLUSH	DKILL_SHARER	If outstanding request, wait for all expected responses. If final response is “DONE” generate an “ERROR” response (we should never see a DKILL_SHARER if we own the coherence granule). If final response is “RETRY” cancel the flush at the processor and forward DKILL_SHARER to processor then generate a “DONE” response If no outstanding request, cancel the data cache invalidate at the processor and forward DKILL_SHARER to processor then generate a “DONE” response.
FLUSH	CASTOUT	Generate “ERROR” response

Table 7-12. Address Collision Resolution for Participant FLUSH(Continued)

Outstanding Request	Incoming Request	Resolution
FLUSH	TLBIE	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
FLUSH	TLBSYNC	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
FLUSH	IKILL_HOME	Generate “ERROR” response
FLUSH	IKILL_SHARER	No collision, forward to processor then generate “DONE” response (software must maintain instruction cache coherence)
FLUSH	FLUSH	Generate “ERROR” response
FLUSH	IO_READ_HOME	Generate “ERROR” response
FLUSH	IO_READ_OWNER	Generate “NOT_OWNER” response (we are not allowed to issue FLUSH to an owned coherence granule - should be a CASTOUT)

Table 7-13 describes the address collision resolution for an incoming transaction that collides with an outstanding non-participant FLUSH transaction.

Table 7-13. Address Collision Resolution for Non-participant FLUSH

Outstanding Request	Incoming Request	Resolution
FLUSH	READ_HOME	Generate “ERROR” response (should never receive coherent operation)
FLUSH	IREAD_HOME	Generate “ERROR” response (should never receive coherent operation)
FLUSH	READ_OWNER	Generate “ERROR” response (should never receive coherent operation)
FLUSH	READ_TO_OWN_HOME	Generate “ERROR” response (should never receive coherent operation)
FLUSH	READ_TO_OWN_OWNER	Generate “ERROR” response (should never receive coherent operation)
FLUSH	DKILL_HOME	Generate “ERROR” response (should never receive coherent operation)
FLUSH	DKILL_SHARER	Generate “ERROR” response (should never receive coherent operation)
FLUSH	CASTOUT	Generate “ERROR” response (should never receive coherent operation)
FLUSH	TLBIE	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence) - non-participant may have page table hardware.

Table 7-13. Address Collision Resolution for Non-participant FLUSH(Continued)

Outstanding Request	Incoming Request	Resolution
FLUSH	TLBSYNC	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence) - non-participant may have page table hardware.
FLUSH	IKILL_HOME	Generate “ERROR” response
FLUSH	IKILL_SHARER	No collision, forward to processor then generate “DONE” response (software must maintain instruction cache coherence) - non-participant may have software coherence.
FLUSH	FLUSH	Generate “ERROR” response (should never receive coherent operation)
FLUSH	IO_READ_HOME	Generate “ERROR” response (should never receive coherent operation)
FLUSH	IO_READ_OWNER	Generate “ERROR” response (should never receive coherent operation)

7.13 Resolving an Outstanding IO_READ_HOME Transaction

The I/O read operation is used by processing elements that do not want to participate in the coherence protocol but do want to get current copies of cached data. There are two versions of this operation, one for processing elements that have both processors and I/O devices, the second for pure I/O devices that do not have a corresponding bit in the directory sharing mask. Table 7-14 describes the address collision resolution for an incoming transaction that collides with an outstanding participant IO_READ_HOME transaction.

Table 7-14. Address Collision Resolution for Participant IO_READ_HOME

Outstanding Request	Incoming Request	Resolution
IO_READ_HOME	READ_HOME	Generate “ERROR” response
IO_READ_HOME	IREAD_HOME	Generate “ERROR” response
IO_READ_HOME	READ_OWNER	Generate “NOT_OWNER” response (we don’t own the data otherwise we could have obtained a copy locally)
IO_READ_HOME	READ_TO_OWN_HOME	Generate “ERROR” response
IO_READ_HOME	READ_TO_OWN_OWNER	Generate “NOT_OWNER” response (we don’t own the data otherwise we could have obtained a copy locally)
IO_READ_HOME	DKILL_HOME	Generate “ERROR” response
IO_READ_HOME	DKILL_SHARER	If outstanding request, wait for all expected responses. If final response is “DONE”, return data if necessary and forward DKILL_SHARER to processor then generate a “DONE” response. If final response is “RETRY” forward DKILL_SHARED to processor then generate a “DONE” response If no outstanding request forward DKILL_SHARER to processor then generate a “DONE” response

Table 7-14. Address Collision Resolution for Participant IO_READ_HOME(Continued)

Outstanding Request	Incoming Request	Resolution
IO_READ_HOME	CASTOUT	Generate “ERROR” response
IO_READ_HOME	TLBIE	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
IO_READ_HOME	TLBSYNC	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
IO_READ_HOME	IKILL_HOME	Generate “ERROR” response
IO_READ_HOME	IKILL_SHARER	No collision, forward to processor then generate “DONE” response (software must maintain instruction cache coherence)
IO_READ_HOME	FLUSH	Generate “ERROR” response
IO_READ_HOME	IO_READ_HOME	Generate “ERROR” response
IO_READ_HOME	IO_READ_OWNER	Generate “NOT_OWNER” response (we don’t own the data otherwise we could have obtained a copy locally)

Table 7-15 describes the address collision resolution for an incoming transaction that collides with an outstanding non-participant IO_READ_HOME transaction.

Table 7-15. Address Collision Resolution for Non-participant IO_READ_HOME

Outstanding Request	Incoming Request	Resolution
IO_READ_HOME	READ_HOME	Generate “ERROR” response (should never receive coherent operation)
IO_READ_HOME	IREAD_HOME	Generate “ERROR” response (should never receive coherent operation)
IO_READ_HOME	READ_OWNER	Generate “ERROR” response (should never receive coherent operation)
IO_READ_HOME	READ_TO_OWN_HOME	Generate “ERROR” response (should never receive coherent operation)
IO_READ_HOME	READ_TO_OWN_OWNER	Generate “ERROR” response (should never receive coherent operation)
IO_READ_HOME	DKILL_HOME	Generate “ERROR” response (should never receive coherent operation)
IO_READ_HOME	DKILL_SHARER	Generate “ERROR” response (should never receive coherent operation)
IO_READ_HOME	CASTOUT	Generate “ERROR” response (should never receive coherent operation)
IO_READ_HOME	TLBIE	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence) - broadcast operation and non-participant may have page table hardware.

Table 7-15. Address Collision Resolution for Non-participant IO_READ_HOME(Continued)

Outstanding Request	Incoming Request	Resolution
IO_READ_HOME	TLBSYNC	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence) - broadcast operation and non-participant may have page table hardware.
IO_READ_HOME	IKILL_HOME	Generate “ERROR” response
IO_READ_HOME	IKILL_SHARER	No collision, forward to processor then generate “DONE” response (software must maintain instruction cache coherence) - broadcast operation and non-participant may have software coherence.
IO_READ_HOME	FLUSH	Generate “ERROR” response (should never receive coherent operation)
IO_READ_HOME	IO_READ_HOME	Generate “ERROR” response (should never receive coherent operation)
IO_READ_HOME	IO_READ_OWNER	Generate “ERROR” response (should never receive coherent operation)

7.14 Resolving an Outstanding IO_READ_OWNER Transaction

The I/O read operation is used by processing elements that do not want to participate in the coherence protocol but do want to get current copies of cached data. There are two versions of this operation, one for processing elements that have both processors and I/O devices, the second for pure I/O devices that do not have a corresponding bit in the directory sharing mask. Table 7-16 describes the address collision resolution for an incoming transaction that collides with an outstanding IO_READ_OWNER transaction.

Table 7-16. Address Collision Resolution for Participant IO_READ_OWNER

Outstanding Request	Incoming Request	Resolution
IO_READ_OWNER	READ_HOME	Generate “RETRY” response
IO_READ_OWNER	IREAD_HOME	Generate “RETRY” response
IO_READ_OWNER	READ_OWNER	Generate “ERROR” response
IO_READ_OWNER	READ_TO_OWN_HOME	Generate “RETRY” response
IO_READ_OWNER	READ_TO_OWN_OWNER	Generate “ERROR” response
IO_READ_OWNER	DKILL_HOME	Generate “RETRY” response
IO_READ_OWNER	DKILL_SHARER	Generate “ERROR” response
IO_READ_OWNER	CASTOUT	No collision, update directory state and memory, generate DONE response (CASTOUT bypasses address collision detection)
IO_READ_OWNER	TLBIE	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
IO_READ_OWNER	TLBSYNC	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)

Table 7-16. Address Collision Resolution for Participant IO_READ_OWNER(Continued)

Outstanding Request	Incoming Request	Resolution
IO_READ_OWNER	IKILL_HOME	No collision, forward to processor, send IKILL_SHARER to all participants except requestor (software must maintain instruction cache coherence)
IO_READ_OWNER	IKILL_SHARER	Generate “ERROR” response
IO_READ_OWNER	FLUSH	Generate “RETRY” response
IO_READ_OWNER	IO_READ_HOME	Generate “RETRY” response
IO_READ_OWNER	IO_READ_OWNER	Generate “ERROR” response (we don’t own the data otherwise we could have obtained a copy locally)

Table 7-17 describes the address collision resolution for an incoming transaction that collides with an outstanding non-participant IO_READ_OWNER transaction.

Table 7-17. Address Collision Resolution for Non-participant IO_READ_OWNER

Outstanding Request	Incoming Request	Resolution
IO_READ_OWNER	READ_HOME	Generate “ERROR” response (should never receive coherent operation)
IO_READ_OWNER	IREAD_HOME	Generate “ERROR” response (should never receive coherent operation)
IO_READ_OWNER	READ_OWNER	Generate “ERROR” response (should never receive coherent operation)
IO_READ_OWNER	READ_TO_OWN_HOME	Generate “ERROR” response (should never receive coherent operation)
IO_READ_OWNER	READ_TO_OWN_OWNER	Generate “ERROR” response (should never receive coherent operation)
IO_READ_OWNER	DKILL_HOME	Generate “ERROR” response (should never receive coherent operation)
IO_READ_OWNER	DKILL_SHARER	Generate “ERROR” response (should never receive coherent operation)
IO_READ_OWNER	CASTOUT	Generate “ERROR” response (should never receive coherent operation)
IO_READ_OWNER	TLBIE	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence) - non-participant may have page table hardware.
IO_READ_OWNER	TLBSYNC	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence) - non-participant may have page table hardware.

Table 7-17. Address Collision Resolution for Non-participant IO_READ_OWNER(Continued)

Outstanding Request	Incoming Request	Resolution
IO_READ_OWNER	IKILL_HOME	No collision, forward to processor, send IKILL_SHARER to all participants except requestor (software must maintain instruction cache coherence)
IO_READ_OWNER	IKILL_SHARER	Generate “ERROR” response
IO_READ_OWNER	FLUSH	Generate “ERROR” response (should never receive coherent operation)
IO_READ_OWNER	IO_READ_HOME	Generate “ERROR” response (should never receive coherent operation)
IO_READ_OWNER	IO_READ_OWNER	Generate “ERROR” response (should never receive coherent operation)

Glossary of Terms and Abbreviations

The glossary contains an alphabetical list of terms, phrases, and abbreviations used in this book.

A

Address collision An address based conflict between two or more cache coherence operations when referencing the same coherence granule.

Agent. A processing element that provides services to a processor.

Asychronous transfer mode (**ATM**). A standard networking protocol which dynamically allocates bandwidth using a fixed-size packet.

B

Big-endian. A byte-ordering method in memory where the address n of a word corresponds to the most significant byte. In an addressed memory word, the bytes are ordered (left to right) 0, 1, 2, 3, with 0 being the most significant byte.

Block flush. An operation that returns the latest copy of a block of data from caches within the system to memory.

Bridge. A processing element that connects one computer bus to another, allowing a processing element on one bus to access an processing element on the other.

Broadcast. The concept of sending a packet to all processing elements in a system.

Bus-based snoopy protocol. A broadcast cache coherence protocol that assumes that all caches in the system are on a common bus.

C

Cache. High-speed memory containing recently accessed data and/or instructions (subset of main memory) associated with a processor.

Cache coherence. Caches are coherent if a processor performing a read from its cache is supplied with data corresponding to the most recent value written to memory or to another processor’s cache. In other words, a write operation to an address in the system is visible to all other caches in the system. Also referred to as memory coherence.

Cache coherent-non uniform memory access (**CC-NUMA**). A cache coherent system in which memory accesses have different latencies depending upon the physical location of the accessed address.

Cache paradox. A circumstance in which the caches in a system have an undefined or disallowed state for a coherence granule, for example, two caches have the same coherence granule marked “modified”.

Capability registers (CARs). A set of read-only registers that allows a processing element to determine another processing element’s capabilities.

Castout operation. An operation used by a processing element to relinquish its ownership of a coherence granule and return it to home memory.

Coherence domain. A logically associated group of processing elements that participate in the globally shared memory protocol and are able to maintain cache coherence among themselves.

Coherence granule. A contiguous block of data associated with an address for the purpose of guaranteeing cache coherence.

Command and status registers (CSRs). A set of registers that allows a processing element to control and determine the status of another processing element's internal hardware.

D

Deadlock. A situation in which two processing elements that are sharing resources prevent each other from accessing the resources, resulting in a halt of system operation.

Destination. The termination point of a packet on the RapidIO interconnect, also referred to as a target.

Device. A generic participant on the RapidIO interconnect that sends or receives RapidIO transactions, also called a processing element.

Device ID. The identifier of an end point processing element connected to the RapidIO interconnect.

Direct Memory Access (**DMA**). The process of accessing memory in a device by specifying the memory address directly.

Distributed memory. System memory that is distributed throughout the system, as opposed to being centrally located.

Domain. A logically associated group of processing elements.

Double-word. An eight byte quantity, aligned on eight byte boundaries.

E

End point. A processing element which is the source or destination of transactions through a RapidIO fabric.

Ethernet. A common local area network (LAN) technology.

Exclusive. A processing element has the only cached copy of a sharable coherence granule. The exclusive state allows the processing element to modify the coherence granule without notifying the rest of the system.

F

Field or Field name. A sub-unit of a register, where bits in the register are named and defined.

Flush operation. An operation used by a processing element to return the ownership and current data of a coherence granule to home memory.

G

Globally shared memory (GSM). Cache coherent system memory that can be shared between multiple processors in a system.

H

Half-word. A two byte or 16 bit quantity, aligned on two byte boundaries.

Home memory. The physical memory corresponding to the physical address of a coherence granule.

I

Initiator. The origin of a packet on the RapidIO interconnect, also referred to as a source.

Instruction cache. High-speed memory containing recently accessed instructions (subset of main memory) associated with a processor.

Instruction cache invalidate operation. An operation that is used if the instruction cache coherence must be maintained by software.

Instruction read operation. An operation used to obtain a globally shared copy of a coherence granule specifically for an instruction cache.

Instruction set architecture (ISA). The instruction set for a certain processor or family of processors.

Intervention. A data transfer between two processing elements that does not go through the coherence granule's home memory, but directly between the requestor of the coherence granule and the current owner.

Invalidate operation. An operation used to remove a coherence granule from caches within the coherence domain.

I/O. Input-output.

I/O read operation. An operation used by an I/O processing element to obtain a globally shared copy of a coherence granule without disturbing the coherence state of the granule.

L

Little-endian. A byte-ordering method in memory where the address n of a word corresponds to the least significant byte. In an addressed memory word, the bytes are ordered (left to right) 3, 2, 1, 0, with 3 being the most significant byte.

Local memory. Memory associated with the processing element in question.

LSB. Least significant byte.

M

Memory coherence. Memory is coherent if a processor performing a read from its cache is supplied with data corresponding to the most recent value written to memory or to another processor's cache. In other words, a write operation to an address in the system is visible to all other caches in the system. Also referred to as cache coherence.

Memory controller. The point through which home memory is accessed.

Memory directory. A table of information associated with home memory that is used to track the location and state of coherence granules cached by coherence domain participants.

Message passing. An application programming model that allows processing elements to communicate via messages to mailboxes instead of via DMA or GSM. Message senders do not write to a memory address in the receiver.

Modified. A processing element has written to a locally cached coherence granule and so has the only valid copy of the coherence granule in the system.

Modified exclusive shared invalid (**MESI**). A standard 4 state cache coherence definition.

Modified shared invalid (**MSI**). A standard 3 state cache coherence definition.

Modified shared local (**MSL**). A standard 3 state cache coherence definition.

MSB. Most significant byte.

Multicast. The concept of sending a packet to more than one processing elements in a system.

N

Non-coherent. A transaction that does not participate in any system globally shared memory cache coherence mechanism.

O

Operation. A set of transactions between end point devices in a RapidIO system (requests and associated responses) such as a read or a write.

Ownership. A processing element has the only valid copy of a coherence granule and is responsible for returning it to home memory.

P

Packet. A set of information transmitted between devices in a RapidIO system.

Peripheral component interface (PCI). A bus commonly used for connecting I/O devices in a system.

Priority. The relative importance of a packet; in most systems a higher priority packet will be serviced or transmitted before one of lower priority.

Processing Element (**PE**). A generic participant on the RapidIO interconnect that sends or receives RapidIO transactions, also called a device.

Processor. The logic circuitry that responds to and processes the basic instructions that drive a computer.

R

Read operation. An operation used to obtain a globally shared copy of a coherence granule.

Read-for-ownership operation. An operation used to obtain ownership of a coherence granule for the purposes of performing a write operation.

Remote access. An access by a processing element to memory located in another processing element.

Remote memory. Memory associated with a processing element other than the processing element in question.

S

Shared. A processing element has a cached copy of a coherence granule that may be cached by other processing elements and is consistent with the copy in home memory.

Sharing mask. The state associated with a coherence granule in the memory directory that tracks the processing elements that are sharing the coherence granule.

Source. The origin of a packet on the RapidIO interconnect, also referred to as an initiator.

Sub-double-word. Aligned on eight byte boundaries.

Switch. A multiple port processing element that directs a packet received on one of its input ports to one of its output ports.

T

Target. The termination point of a packet on the RapidIO interconnect, also referred to as a destination.

Transaction. A specific request or response packet transmitted between end point devices in a RapidIO system.

Translation look-aside buffer (TLB). Part of a processor’s memory management unit; a TLB contains a set of virtual to physical page address translations, along with a set of attributes that describe access behavior for that portion of physical memory.

W

Write-through. A cache policy that passes all write operations through the caching hierarchy directly to home memory.

Word. A four byte or 32 bit quantity, aligned on four byte boundaries.

Partition VI:

Physical Layer 1x/4x LP-Serial Specification

VI Partition VI - Physical Layer 1x/4x LP-Serial Specification

1 Chapter 1 - Overview

The RapidIO^{TM} 1x/4x LP-Serial (Link Protocol - Serial) architecture was developed to address the need for a high-performance, low pin-count, and low-power serial packet-switched system level interconnect to be used in a variety of applications as an open standard. The architecture is targeted toward networking, telecom, and high performance embedded applications. It is intended primarily as an intra-system interface, allowing chip-to-chip and boar-to-board communications at Gigabytes per second performance levels.

The RapidIO *Physical Layer 1x/4x LP-Serial Specification* addresses the physical layer requirements for devices utilizing an electrical serial connection medium. This specification defines a full duplex serial physical layer interface (link) between devices using unidirectional differential signals in each direction. Further, it allows ganging of four serial links for applications requiring higher link performance. It also defines a protocol for link management and packet transport over a link.

RapidIO systems are comprised of end point processing elements and switch processing elements. The RapidIO interconnect architecture is partitioned into a layered hierarchy of specifications which includes the Logical, Common Transport, and Physical layers. The Logical layer specifications define the operations and associated transactions by which end point processing elements communicate with each other. The Common Transport layer defines how transactions are routed from one end point processing element to another through switch processing elements. The Physical Layer defines how adjacent processing elements electrically connect to each other. RapidIO packets are formed through the combination of bit fields defined in the Logical, Common Transport, and Physical Layer specifications.

The RapidIO Physical Layer 1x/4x LP-Serial specification defines a protocol for packet delivery between serial RapidIO devices including packet and control symbol transmission, flow control, error management, and other device to device functions. A particular device may not implement all of the mode selectable features found in this document. See the appropriate user's manual or implementation specification for specific implementation details of a device.

The 1x/4x LP-Serial physical layer specification has the following properties:

- Embeds the transmission clock with data using an 8B/10B encoding scheme.
- Supports one serial differential pair, referred to as one lane, or four ganged serial differential pairs, referred to as four lanes, in each direction.
- Allows switching packets between RapidIO 1x/4x LP-Serial Ports and RapidIO Physical Layer 8/16 LP-LVDS ports without requiring packet manipulation.
- Employs similar retry and error recovery protocols as the RapidIO Physical Layer 8/16 LP-LVDS specification.
- Supports transmission rates of 1.25, 2.5, and 3.125 Gbaud (data rates of 1.0, 2.0, and 2.5 Gbps) per lane.

This specification first defines the individual elements that make up the link protocol such as packets, control symbols, and the serial bit encoding scheme. This is followed by a description of the link protocol. Finally, the control and status registers, signal descriptions, and electrical specifications are specified.

1.1 Packets

Chapter 2, "Packets" defines how a RapidIO 1x/4x LP-Serial packet is formed by prefixing a 10-bit physical layer header to the combined RapidIO transport and logical layer bit fields followed by an appended 16-bit CRC field.

This chapter shows the packet header format, the packet field definitions, the CRC error detection mechanism, and the packet alignment rules necessary to form LP-Serial packets.

1.2 Control Symbols

Chapter 3, "Control Symbols" defines the format of the two classes of control symbols (stype0 and stype1) used for packet acknowledgment, link utility functions, link maintenance, and packet delineation. A control symbol is a 24-bit entity (including a 5-bit CRC code). The control symbol is used for packet delineation and may also be embedded within a packet as well as sent when the link is idle.

Acknowledgment control symbols are used by processing elements to indicate packet transmission status. Utility control symbols are used to communicate buffer status and link recovery synchronization. Link maintenance control symbols are used by adjacent devices to communicate physical layer status, synchronization requests, and device reset.

1.3 PCS and PMA Layers

Chapter 4, "PCS and PMA Layers" describes the Physical Coding Sublayer (PCS) functionality as well as the Physical Media Attachment (PMA) functionality. The PCS layer functionality includes 8B/10B encoding scheme for embedding clock with data. It also gives transmission rules for the 1x and 4x interfaces and defines the link initialization sequence for clock synchronization.

The PMA (Physical Medium Attachment) function is responsible for serializing the 10-bit code-groups to and from the serial bitstream(s).

1.4 LP-Serial Protocol

Chapter 5, "LP-Serial Protocol" describes in detail how packets, control symbols, and the PCS/PMA layers are used to implement the physical layer protocol. This includes topics such as link initialization, link maintenance, error detection and recovery, flow control, and transaction delivery ordering.

1.5 LP-Serial Registers

Chapter 6, "LP-Serial Registers" describes the physical layer control and status register set. By accessing these registers a processing element may query the capabilities and status and configure another 1x/4x LP-Serial RapidIO processing element.

These registers utilize the Extended Features blocks and are accessed using *Part I: Input/Output Logical Specification* Maintenance operations. Three types of RapidIO devices are defined in this section as follows:

- Generic End Point Processing Elements
- Generic End Point Processing Elements with software assisted error recovery
- Generic End Point Free Processing Elements (typically switch processing elements)

1.6 Signal Descriptions

Chapter 7, "Signal Descriptions" contains the signal pin descriptions for a RapidIO LP-Serial end point device and shows connectivity between processing elements with 1x ports and processing elements with 4x ports.

1.7 AC Electrical Specifications

Chapter 8, "AC Electrical Specifications" describes the electrical specifications for the RapidIO 1x/4x LP-Serial device. This section defines two transmission types; short run and long run, as well as three speed grades (1.25 GHz, 2.5 GHz, and 3.125 GHz). This section also shows the required receiver eye diagrams for each link speed.

1.8 Interface Management

Appendix A, "Interface Management (Informative)" contains information pertinent to interface management in a RapidIO system, including error recovery, link initialization, and packet retry state machines.

	A decimal value.
[n-m]	Used to express a numerical range from n to m.
0bnn	A binary value, the number of bits is determined by the number of digits.
0xnn	A hexadecimal value, the number of bits is determined by the number of digits or from the surrounding context; for example, 0xnn may be a 5, 6, 7, or 8 bit value.2

2 Chapter 2 - Packets

This chapter specifies the LP-Serial packet format and the fields that are added by LP-Serial physical layer. These packets are fed into the PCS function explained in Chapter 4, "PCS and PMA Layers".

2.1 Packet Field Definitions

This section specifies the bit fields added to a packet by the LP-Serial physical layer. These fields are required to implement the flow control, error management, and other specified system functions of the LP-Serial specification. The fields are specified in Table .

Table 2.1 - Packet Field Definitions

Field	Description
ackID[0-4]	Acknowledge ID is the packet identifier for acknowledgments back to the packet sender—see Section 5.3.2: "Acknowledgment Identifier" for details concerning ackID functionality.
rsvd[0-2]	The reserved bits are set to logic 0 when the packet is generated and ignored when a packet is received.
prio[0-1]	Sets packet priority: 0b00 - lowest priority 0b01 - medium priority 0b10 - high priority 0b11 - highest priority See Section 5.3.3: "Packet Priority and Transaction Request Flows" for an explanation of prioritizing packets
crc[0-15]	16-bit code used to detect transmission errors in the packet. See Section : "2.3.1 Packet CRC Operation" for details on the CRC error detection scheme.

2.2 Packet Format

This section specifies the format of a LP-Serial packets. Figure shows the format of the LP-Serial packet and how the physical layer ackID, rsvd, and prio fields are prefixed at the beginning of the packet and the 16-bit CRC field is appended to the end of the packet.

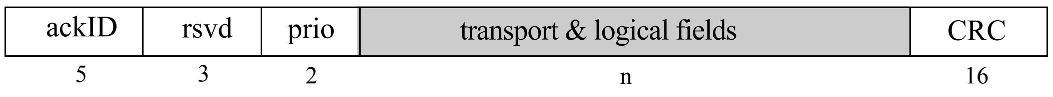


Figure 2-1. Packet Format

The unshaded fields are the fields added by the physical layer. The shaded field is the combined logical and transport layer bits and fields that are passed to the physical layer. The 3-bit rsvd field is required to make the packet length an integer multiple of 16 bits.

LP-Serial packets shall have a length that is an integer multiple of 32 bits. This sizing simplifies the design of port logic whose internal data paths are an integer multiple of 32 bits in width. Packets, as defined in this specification and the appropriate logical and transport layer specifications, have a length that is an integer multiple of 16 bits. This is illustrated in Figure . If the length of a packet defined by the above combination of specifications is an odd multiple of 16 bits, a 16-bit pad whose value is 0 (0x0000) shall be appended at the end of the packet such that the resulting padded packet is an integer multiple of 32 bits in length.

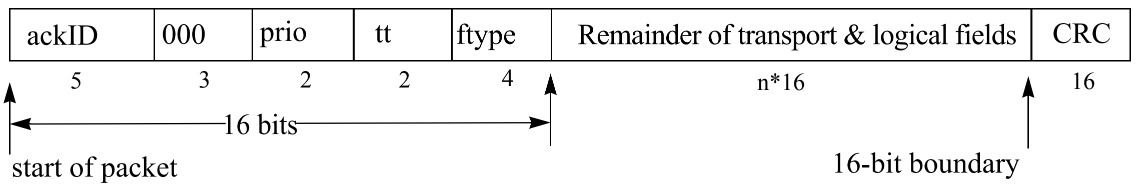


Figure 2.2. Packet Alignment

2.3 Packet Protection

A 16-bit CRC code is added to each packet by the LP-Serial physical layer to provide error detection. The code covers the entire packet except for the ackID field and one bit of the rsvd field, which are considered to be zero for the CRC calculations. Figure 2-3 shows the CRC coverage for the first 16 bits of the packet which contain the bits not covered by the code.

This structure allows the ackID to be changed on a link-by-link basis as the packet is transported across the fabric without requiring that the CRC be recomputed for each link. Since ackIDs on each link are assigned sequentially for each subsequent transmitted packet, an error in the ackID field is easily detected.

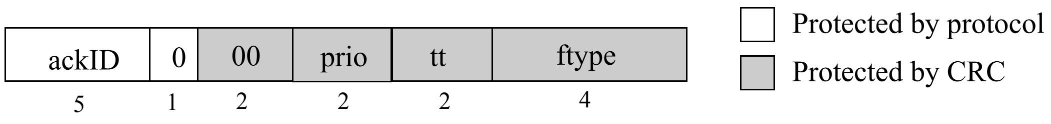


Figure 2.3. Error Coverage of First 16 Bits of Packet Header

2.3.1 Packet CRC Operation

The CRC is appended to a packet in one of two ways. For a packet whose length, exclusive of CRC, is 80 bytes or less, a single CRC is appended at the end of the logical fields. For packets whose length, exclusive of CRC, is greater than 80 bytes, a CRC is added after the first 80 bytes and a second CRC is appended at the end of the logical layer fields.

The second CRC value is a continuation of the first. The first CRC is included in the running calculation, meaning that the running CRC value is not reinitialized after it is inserted after the first 80 bytes of the packet. This allows intervening devices to regard the embedded CRC value as two bytes of packet payload for CRC checking purposes. If the CRC appended to the end of the logical layer fields does not cause the end of the resulting packet to align to a 32-bit boundary, a two byte pad of all logic 0s is postpended to the packet. The pad of logic 0s allows the CRC check to always be done at the 32-bit boundary.

The early CRC value can be used by the receiving processing element to validate the header of a large packet and start processing the data before the entire packet has been received, freeing up resources earlier and reducing transaction completion latency.

NOTE:

While the embedded CRC value can be used by a processing element to start processing the data within a packet before receiving the entire packet, it is possible that upon reception of the end of the packet the final CRC value for the packet is incorrect. This would result in a processing element that has processed data that may have been corrupted. Outside of the error recovery mechanism described in Section 5.10.2: "Link Behavior Under Error", the RapidIO Interconnect Specification does not address the occurrence of such situations nor does it suggest a means by which a processing element would handle such situations. Instead, the mechanism for handling this situation is left to be addressed by the device manufacturers for devices that implement the functionality of early processing of packet data.

Figure 2-4 is an example of an unpadded packet of length less than or equal to 80 bytes.

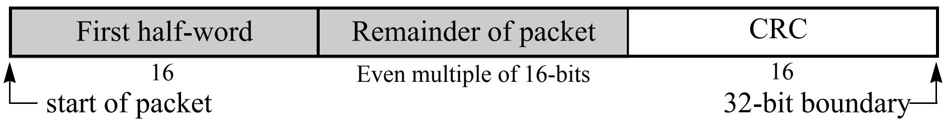


Figure 2-4. Unpadded Packet of Length 80 Bytes or Less

Figure 2-5 is an example of a padded packet of length less than or equal to 80 bytes.

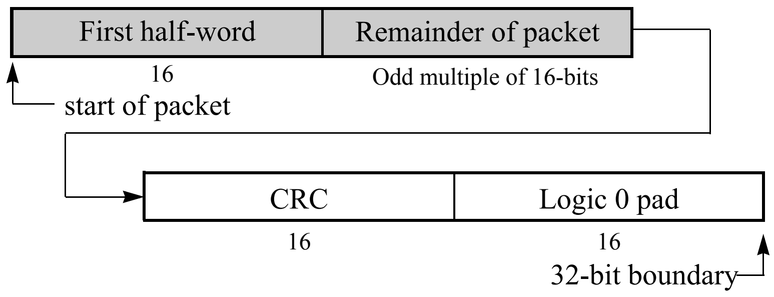


Figure 2-5. Padded Packet of Length 80 Bytes or Less

Figure 2-6 is an example of an unpadded packet of length greater than 80 bytes.

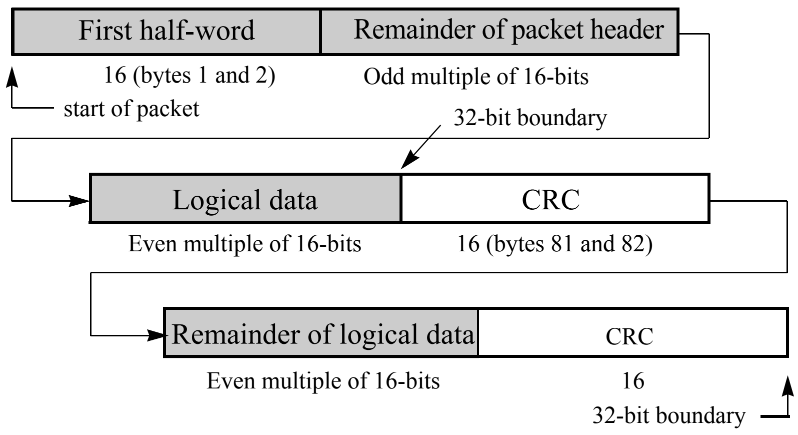


Figure 2-6. Unpadded Packet of Length Greater than 80 Bytes

Figure 2-7 is an example of a padded packet of length greater than 80 bytes.

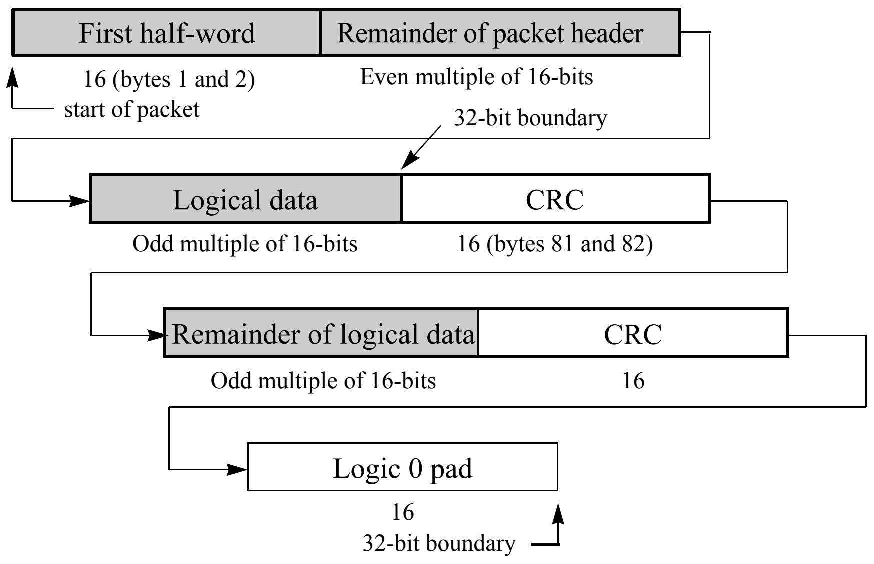


Figure 2-7. Padded Packet of Length Greater than 80 Bytes

2.3.2 16-Bit Packet CRC Code

The ITU polynomial $X^{16}+X^{12}+X^{5}+1$ shall be used to generate the 16-bit CRC for packets. The value of the CRC shall be initialized to 0xFFFF (all logic 1s) at the beginning of each packet. For the CRC calculation, the uncovered six bits are treated as logic 0s. As an example, a 16-bit wide parallel calculation is described in the equations in Table 2-2. Equivalent implementations of other widths can be employed.

Table 2-2. Parallel CRC Intermediate Value Equations

Check Bit	**e00**	**e01**	**e02**	**e03**	**e04**	**e05**	**e06**	**e07**	**e08**	**e09**	**e10**	**e11**	**e12**	**e13**	**e14**	**e15**
C00					x	x			x				x			
C01						x	x			x				x		
C02							x	x			x				x	
C03	x							x	x			x				x
C04	x	x			x	x				x						
C05		x	x			x	x				x					
C06	x		x	x			x	x				x				
C07	x	x		x	x			x	x				x			
C08	x	x	x		x	x			x	x				x		
C09		x	x	x		x	x			x	x				x	
C10			x	x	x		x	x			x	x				x
C11	x			x				x				x				
C12	x	x			x				x				x			
C13		x	x			x				x				x		
C14			x	x			x				x				x	
C15				x	x			x				x				x

The following nomenclature is used in the diagram below:

C00–C15 contents of the new check half-word

e00–e15 contents of the intermediate value half-word
e00 = d00 XOR c00
e01 = d01 XOR c01
through
e15 = d15 XOR c15

d00–d15 contents of the next 16 bits of the packet

c00–c15 contents of the previous check half-word

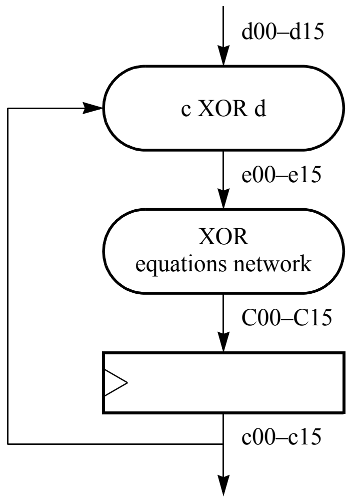


Figure 2-8. CRC Generation Pipeline

2.4 Maximum Packet Size

The maximum packet size permitted by the LP-Serial specification is 276 bytes. This includes all packet logical, transport, and physical layer header information, data payload, and required CRC bytes.

The maximum packet size of 276 bytes is achieved as shown below:

Table 2-3. Maximum Packet Size

Field	Size (bytes)	Layer	Notes
Header	2	Physical, Transport, Logical	
Source ID	2	Transport	
Destination ID	2	Transport	
Trans/wrsize	1	Logical	
srcTID	1	Logical	
Address	8	Logical	Includes Extended_address, Address, Wdptr, and Xambs
Payload	256	Logical	
CRC	4	Physical	Extra two CRC bytes for packets greater than 80 bytes
Total	276		

3 Chapter 3 - Control Symbols

This chapter specifies RapidIO physical layer control symbols. Control symbols are the message elements used by ports connected by an LP-Serial link to manage all aspects of LP-Serial link operation. They are used for link maintenance, packet delimiting, packet acknowledgment, error reporting, and error recovery.

3.1 Control Symbol Field Definitions

Table 3-1. Control Symbol Field Definitions

Field	Description
stype0 [0-2]	Encoding for control symbols that make use of parameter0 and parameter1. Eight encodings are defined in Table .
parameter0 [0-4]	Used in conjunction with stype0 encodings. Reference Table for the description of parameter0 encodings.
parameter1 [0-4]	Used in conjunction with stype0 encodings. Reference Table for the description of parameter1 encodings.
stype1 [0-2]	Encoding for control symbols which make use of the cmd field. The eight encodings are defined in Table .
cmd [0-2]	Used in conjunction with the stype1 field to define the link maintenance commands. Refer to Table for the cmd field descriptions.
CRC [0-4]	5-bit code used to detect transmission errors in control symbols. See Section : "3.5 Control Symbol Protection" for details on the CRC error detection scheme.

3.2 Control Symbol Format

This section describes the general format of the LP-Serial control symbols. Figure shows the control symbol format.

0 2	3 7	8 12	13 15	16 18	19 23
stype0 [0-2]	parameter0 [0-4]	parameter1 [0-4]	stype1 [0-2]	cmd [0-2]	CRC [0-4]

Figure 3-1. Control Symbol Format

All control symbols follow the 24-bit control symbol format as detailed above. The fields parameter0 and parameter1 are used by the functions encoded in the stype0 field. The cmd field is a modifier for the functions encoded in the stype1 field.

Control symbols can carry two functions, one encoded in the stype0 field and one encoded in the stype1 field. The functions encoded in stype0 are “status” functions that convey some type of status about the port transmitting the control symbol. The functions encoded in stype1 are requests to the receiving port or transmission delimiters.

A control symbol carrying one function is referred to using the name of the function it carries. A control symbol carrying two functions may be referred to using the name of either function that it carries. For example, a control symbol with stype0 set to packet-accepted and stype1 set to NOP is referred to a packet-accepted control symbol. A control symbol with stype0 set to packet-accepted and stype1 set to restart-from-retry is referred to as either a packet-accepted control symbol or a restart-from-retry control symbol depending on which name is appropriate for the context.

Control symbols are specified with the ability to carry two functions so that a packet acknowledgment and a packet delimiter can be carried in the same control symbol. Packet acknowledgment and packet delimiter control symbols constitute the vast majority of control symbol traffic on a busy link. Carrying an acknowledgment (or status) and a packet delimiter whenever possible in a single control symbol allows a significant reduction in link overhead traffic and an increase in the link bandwidth available for packet transmission.

3.3 Stype0 Control Symbols

The encoding and function of stype0 and the information carried in parameter0 and parameter1 for each stype0 encoding shall be as specified in Table 3-2.

Table 3-2. Stype0 Control Symbol Encoding

stype0 [0-2]	Function	Contents of		Reference
		Parameter0	Parameter1	
0b000	Packet-accepted	packet_ackID	buf_status	Section : "3.3.1 Packet-Accepted Control Symbol"
0b001	Packet-retry	packet_ackID	buf_status	Section : "3.3.2 Packet-Retry Control Symbol"
0b010	Packet-not-accepted	packet_ackID	cause	Section : "3.3.3 Packet-Not-Accepted Control Symbol"
0b011	Reserved	-	-	-
0b100	Status	ackID_status	buf_status	Section : "3.3.4 Status Control Symbol"
0b101	Reserved	-	-	-
0b110	Link-response	ackID_status	port_status	Section : "3.3.5 Link-Response Control Symbol"
0b111	Reserved	-	-	-

The status control symbol is the default stype0 encoding and is used when the control symbol does not convey another stype0 function. The following table defines the parameters valid for stype0 control symbols.

Table 3-3. Stype0 Parameter Definitions

Parameter	Definition
packet_ackID [0-4]	The ackID of the packet being acknowledged by an acknowledgment control symbol.
ackID_status [0-4]	The value of ackID expected in the next packet the port receives. For example, a value of 0b00001 indicates the device is expecting to receive ackID 1.
buf_status [0-4]	Specifies the number of maximum length packets that the port can accept without issuing a retry due to a lack of resources. The value of buf_status in a packet-accepted, packet-retry, or status control symbol is the number of maximum packets that can be accepted, inclusive of the effect of the packet being accepted or retried. **Value 0-29**: The encoding value specifies the number of new maximum sized packets the receiving device can receive. The value 0, for example, signifies that the downstream device has no available packet buffers (thus is not able to hold any new packets). **Value 30**: The value 30 signifies that the downstream device can receive 30 or more new maximum sized packets. **Value 31**: The downstream device can receive an undefined number of maximum sized packets, and relies on the retry protocol for flow control.

NOTE:

The following sections depict various control symbols. Since control symbols can contain one or two functions, shading in the figures is used to indicate which fields are applicable to that specific control symbol function.

3.3.1 Packet-Accepted Control Symbol

The packet-accepted control symbol indicates that the receiving device has taken responsibility for sending the packet to its final destination and that resources allocated by the sending device can be released. This control symbol shall be generated only after the entire packet has been received and found to be free of detectable errors. The packet-accepted control symbol format is displayed in Figure 3-2.

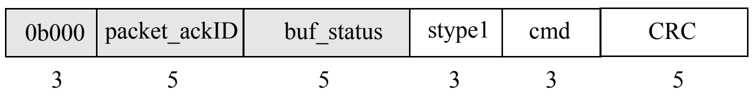


Figure 3-2. Packet-Accepted Control Symbol Format

3.3.2 Packet-Retry Control Symbol

A packet-retry control symbol indicates that the receiving device was not able to accept the packet due to some temporary resource conflict such as insufficient buffering and the sender should retransmit the packet. This control symbol format is displayed in Figure 3-3.

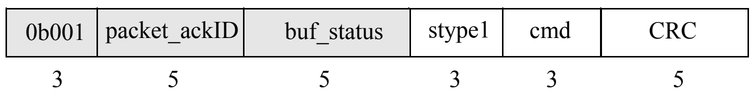


Figure 3-3. Packet-Retry Control Symbol Format

3.3.3 Packet-Not-Accepted Control Symbol

The packet-not-accepted control symbol is used to indicate to the sender of a packet why the packet was not accepted by the receiving port. As shown in Figure 3-4, the control symbol contains a cause field that indicates the reason for not accepting the packet and a packet_ackID field. If the receiving device is not able to specify the cause, or the cause is not one of defined options, the general error encoding shall be used.

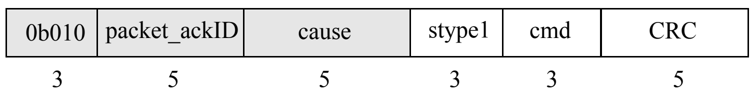


Figure 3-4. Packet-Not-Accepted Control Symbol Format

The cause field shall be used to display informational fields useful for debug. Table displays the reasons a packet may not be accepted, indicated by the cause field.

Table 3-4. Cause Field Definition (continued)

Cause [0-4]	Definition
0b00000	Reserved
0b00001	Received unexpected ackID on packet
0b00010	Received a control symbol with bad CRC
0b00011	Non-maintenance packet reception is stopped
0b00100	Received packet with bad CRC
0b00101	Received invalid character, or valid but illegal character

Table 3-4. Cause Field Definition (continued)

Cause [0-4]	Definition
0b00110 - 0b11110	Reserved
0b11111	General error

3.3.4 Status Control Symbol

The status control symbol is the default stype0 encoding and is used when the control symbol does not convey another stype0 function. The status control symbol contains the ackID_status and the buf_status fields. The buf_status field indicates to the receiving port the number of maximum length packet buffers the sending port had available for packet reception at the time the control symbol was generated. The ackID_status field allows the receiving port to determine if it and the sending port are in sync with respect to the next ackID value the sending port expects to receive. The status control symbol format is shown in Figure 3-5 below.

0b100	ackID_status	buf_status	stype1	cmd	CRC
3	5	5	3	3	5

Figure 3-5. Status Control Symbol Format

3.3.5 Link-Response Control Symbol

The link-response control symbol is used by a device to respond to a link-request control symbol as described in the link maintenance protocol described in Section 5.4: "Link Maintenance Protocol". The status reported in the status field is the status of the port at the time the associated input-status link-request control symbol was received.

0b110	ackID_status	port_status	stype1	cmd	CRC
3	5	5	3	3	5

Figure 3-6. Link-Response Control Symbol Format

The port_status field of the link-response control symbol is defined in Table 3-5.

Table 3-5. Port_status Field Definitions

Port_status [0-4]	Status	Description
0b00000		Reserved
0b00001		Reserved
0b00010	Error	The port has encountered an unrecoverable error and is unable to accept packets.
0b00011		Reserved
0b00100	Retry-stopped	The port has retried a packet and is waiting in the input retry-stopped state to be restarted.
0b00101	Error-stopped	The port has encountered a transmission error and is waiting in the input error-stopped state to be restarted.
0b00110 - 0b01111		Reserved
0b10000	OK	The port is accepting packets
0b10001 - 0b11111		Reserved

3.4 Stype1 Control Symbols

The encoding of stype1 and the function of the cmd field are defined in Table 3-6.

Table 3-6. Stype1 Control Symbol Encoding

stype1 [0-2]	stype1 Function	cmd [0-2]	cmd Function	Packet Delimiter	Reference
0b000	Start-of-packet	0b000	Reserved	yes	Section : "3.4.1 Start-of-Packet Control Symbol"
0b001	Stomp	0b000	Reserved	yes	Section : "3.4.2 Stomp Control Symbol"
0b010	End-of-packet	0b000	Reserved	yes	Section : "3.4.3 End-of-Packet Control Symbol"
0b011	Restart-from-retry	0b000	Reserved	*	Section : "3.4.4 Restart-From-Retry Control Symbol"
0b100	Link-request	0b000 - 0b010	-	*	-
		0b011	Reset-device		Section : "3.4.5.1 Reset-Device Command"
		0b100	Input-status		Section : "3.4.5.2 Input-Status Command"
		0b101- 0b111	-		-
0b101	Multicast-event	0b000	Reserved	No	Section : "3.4.6 Multicast-Event Control Symbol"
0b110	Reserved	0b000	Reserved	No	-
0b111	NOP (Ignore) **	0b000	Reserved	No	-

NOTE:

** denotes that restart-from-retry and link-request control symbols may only be packet delimiters if a packet is in progress.*

*** NOP (Ignore) is not defined as a control symbol, but is the default value when the control symbol does not convey another stype1 function.*

The following sections depict various control symbols. Since control symbols can contain one or two functions, shading in the figures is used to indicate which fields are applicable to that specific control symbol function.

3.4.1 Start-of-Packet Control Symbol

The start-of-packet control symbol format is shown in Figure below.

stype0	parameter0	parameter1	0b000	0b000	CRC
3	5	5	3	3	5

Figure 3-7. Start-of-Packet Control Symbol Format

3.4.2 Stomp Control Symbol

The stomp control symbol is used to cancel a partially transmitted packet. The protocol for packet cancellation is specified in Section 5.7: "Canceling Packets". The stomp control symbol format is shown in Figure 3-8 below.

stype0	parameter0	parameter1	0b001	0b000	CRC
3	5	5	3	3	5

Figure 3-8. Stomp Control Symbol Format

3.4.3 End-of-Packet Control Symbol

The end-of-packet control symbol format is shown in Figure 3-9 below.

stype0	parameter0	parameter1	0b010	0b000	CRC
3	5	5	3	3	5

Figure 3-9. End-of-Packet Control Symbol Format

3.4.4 Restart-From-Retry Control Symbol

The restart-from-retry control symbol cancels a current packet and may also be transmitted on an idle link. This control symbol is used to mark the beginning of packet retransmission, so that the receiver knows when to start accepting packets after the receiver has requested a packet to be retried. The control symbol format is shown in Figure 3-10 below.

stype0	parameter0	parameter1	0b011	0b000	CRC
3	5	5	3	3	5

Figure 3-10. Restart-From-Retry Control Symbol Format

3.4.5 Link-Request Control Symbol

A link-request control symbol is used by a device to either issue a command to the connected device or request its input port status. A link-request control symbol cancels a current packet and can be sent between packets. Under error conditions, a link-request/input-status control symbol acts as a link-request/restart-from-error control symbol as described in Section 5.10.2.1, “Recoverable Errors.” This control symbol format is displayed in Figure 3-11.

stype0	parameter0	parameter1	0b100	cmd	CRC
3	5	5	3	3	5

Figure 3-11. Link-Request Control Symbol Format

The cmd, or command, field of the link-request control symbol format is defined in Table 3-7 below.

Table 3-7. Cmd Field Definitions

cmd[0-2] Encoding	Command Name	Description	Reference
0b000-0b010	-	Reserved	
0b011	Reset-device	Reset the receiving device	Section : "3.4.5.1 Reset-Device Command"

cmd[0-2] Encoding	Command Name	Description	Reference
0b100	Input-status	Return input port status; functions as a link request (restart-from-error) control symbol under error conditions	Section : "3.4.5.2 Input-Status Command"
0b101-0b111	-	Reserved	

3.4.5.1 Reset-Device Command

The reset-device command causes the receiving device to go through its reset or power-up sequence. All state machines and the configuration registers reset to the original power on states. The reset-device command does not generate a link-response control symbol.

Due to the undefined reliability of system designs it is necessary to put a safety lockout on the reset function of the link-request control symbol. A device receiving a reset-device command in a link-request control symbol shall not perform the reset function unless it has received four reset-device commands in a row without any other intervening packets or control symbols, except status control symbols. This will prevent spurious reset commands from inadvertently resetting a device.

3.4.5.2 Input-Status Command

The input-status command requests the receiving device to return the ackID value it expects to next receive from the sender on its input port and the current input port operational status for informational purposes. This command causes the receiver to flush its output port of all control symbols generated by packets received before the input-status command. Flushing the output port is implementation dependent and may result in either discarding the contents of the receive buffers or sending the control symbols on the link. The receiver then responds with a link-response control symbol.

3.4.6 Multicast-Event Control Symbol

The multicast-event control symbol differs from other control symbols in that it carries information not related to the link carrying the control symbol. The multicast-event control symbol allows the occurrence of a user-defined system event to be multicast throughout a system. Refer to Section 5.2.4: "Multicast-Event Control Symbols" for more details on Multicast-Events.

The multicast-event control symbol format is shown in Figure 3-12 below.

stype0	parameter0	parameter1	0b101	0b000	CRC
3	5	5	3	3	5

Figure 3-12. Multicast-Event Control Symbol Format

3.5 Control Symbol Protection

The 5-bit CRC shall be computed over control symbol bits 0 through 18 and provides 5-bit burst error detection for the entire 24-bit control symbol.

3.5.1 CRC-5 Code

The ITU polynomial $X^5+X^4+X^2+1$ shall be used to generate the 5-bit CRC for control symbols. The CRC check bits c0, c1, c2, c3, and c4 occupy the last 5 bits of a control symbol. It should be noted that the 5-bit CRC must be generated by each transmitter and verified by each receiver. Before the 5-bit CRC is computed, the CRC should be set to all 1's or 0b11111. In order to provide maximum implementation flexibility for all types of designs, a 20th bit has been added. For all computations, the 20th bit shall be the last bit applied and shall be set to a logic 0 (0b0).

3.5.2 CRC-5 Parallel Code Generation

Since it is often more efficient to implement a parallel CRC algorithm rather than a serial, examples of the equations for a complete, 19-bit single-stage parallel implementation is shown in shown in Table . Since only a single stage is used, the effect of both setting the initial CRC to all 1's (0b11111) and a 20th bit set to logic 0 (0b0) have been included in the equations.

In Table , an "x" means that the data input should be an input to the Exclusive-OR necessary to compute that particular bit of the CRC. A "!x", means that bit 18 being applied to the CRC circuit must be inverted. Figure shows the 19-bits that the CRC covers and how they should be applied to the circuit. As seen in Figure , bits are labeled with 0 on the left and 18 on the right. Bit 0, from the stype0 field, would apply to D0 in Table and bit 18, from the cmd field, would apply to D18 in Table . Once completed, the 5-bit CRC is appended to the control symbol.

CRC Checksum	**Control Symbol Data For CRC**																		
Bits	D 0	D 1	D 2	D 3	D 4	D 5	D 6	D 7	D 8	D 9	D 10	D 11	D 12	D 13	D 14	D 15	D 16	D 17	D 18
4	x		x	x	x					x		x			x	x		x	x
3		x	x	x					x		x			x	x		x	x	!x
2		x		x	x			x				x	x	x	x		x		!x
1	x		x	x			x				x	x	x	x		x		x	!x
0	x	x		x	x	x					x		x			x	x		x

Parallel CRC Equations

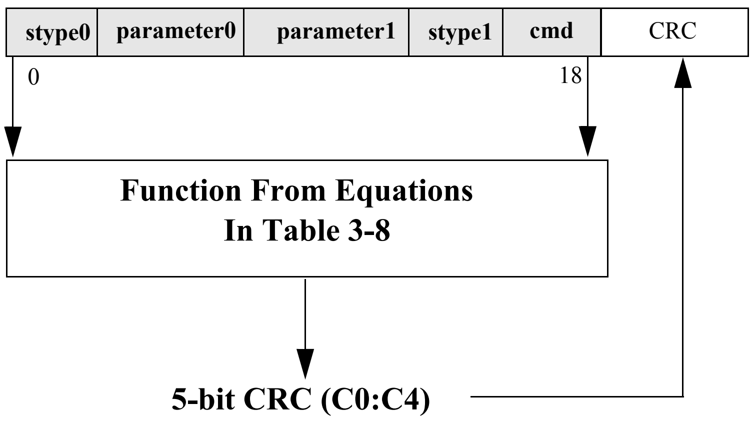


Figure 3-13. 5-bit CRC Implementation4Chapter 4 - PCS and PMA Layers

This chapter specifies the functions provided by the Physical Coding Sublayer (PCS) and Physical Media Attachment (PMA) sublayer. (The PCS and PMA terminology is adopted from IEEE 802.3). The topics include 8B/10B encoding, character representation, serialization of the data stream, code-groups, columns, link transmission rules, idle sequences, and link initialization.

The concept of lanes is used to describe the width of a LP-Serial link. A lane is defined as one unidirectional differential pair in each direction. RapidIO LP-Serial defines two link widths. The 1x LP-Serial link is a one-lane link and the 4x LP-Serial link is a 4-lane link. Wider links are possible, but are left for future work.

4 Chapter 4 - Layer Functions

4.1 PCS Layer Functions

The Physical Coding Sublayer (PCS) function is responsible for idle sequence generation, lane striping, and encoding for transmission and decoding, lane alignment, and destriping on reception. The PCS uses an 8B/10B encoding for transmission over the link. See reference #5 of the bibliography section on page 243 for the source of the 8B/10B encoding scheme.

The PCS layer also provides mechanisms for determining the operational mode of the port as 4-lane or 1-lane operation, and means to detect link states. It provides for clock difference tolerance between the sender and receiver without requiring flow control.

The PCS layer performs the following transmit functions:

- Dequeues packets and delimited control symbols awaiting transmission as a character stream.
- Stripes the transmit character stream across the available lanes.
- Generates the idle sequence and inserts it into the transmit character stream for each lane when no packets or delimited control symbols are available for transmission.
- Encodes the character stream of each lane independently into 10-bit parallel code-groups.
- Passes the resulting 10-bit parallel code-groups to the PMA.

The PCS layer performs the following receive functions:

- Decodes the received stream of 10-bit parallel code-groups for each lane independently into characters.
- Marks characters decoded from invalid code-groups as invalid.
- If the link is using more than one lane, aligns the character streams to eliminate the skew between the lanes and reassembles (destripes) the character stream from each lane into a single character stream.
- Delivers the decoded character stream of packets and delimited control symbols to the higher layers.

4.2 PMA Layer Functions

The PMA (Physical Medium Attachment) function is responsible for serializing 10-bit parallel code-groups to/from a serial bitstream on a lane-by-lane basis. Upon receiving data, the PMA function provides alignment of the received bitstream to 10-bit code-group boundaries, independently on a lane-by-lane basis. It then provides a continuous stream of 10-bit code-groups to the PCS, one stream for each lane. The 10-bit code-groups are not observable by layers higher than the PCS.

4.3 Definitions

Definitions of terms used in this specification are provided below.

Byte: An 8-bit unit of information. Each bit of a byte has the value 0 or 1.

Character: A 9-bit entity comprised of an information byte and a control bit that indicates whether the information byte contains data or control information. The control bit has the value D or K indicating that the information byte contains respectively data or control information.

D-character: A character whose control bit has the value “D”.

K-character: A character whose control bit has the value “K”. Also referred to as a special character.

Code-group: A 10-bit entity that is the result of 8B/10B encoding a character.

Column: A group of four characters that are transmitted simultaneously on a 4x (4 lane) link.

Comma: A 7-bit pattern, unique to certain 8B/10B special code-groups, that is used by a receiver to determine code-group boundaries. See more in “Section , 4.4.7.4 Sync (/K/)” on page 270 and Table , “Table 4-2. Special Character Encodings,” on page 268.

Idle sequence: The sequence of characters (code-groups after encoding) that is transmitted when a packet or control symbol is not being transmitted. The idle sequence allows the receiver to maintain bit synchronization and code-group alignment in between packets and control symbols.

Lane Alignment: The process of eliminating the skew between the lanes of a 4-lane LP-Serial link such that the characters transmitted as a column by the sender are output by the alignment process of receiver as a column. Without lane alignment, the characters transmitted as a column might be scattered across several columns output by the receiver. The alignment process uses the columns of "A" special characters transmitted as part of the idle sequence.

Striping: The method used on a 4x link to send data across four lanes simultaneously. The character stream is *striped* across the lanes, on a character-by-character basis, starting with lane 0, to lane 1, to lane 2, to lane3, and wrapping back with the 5th character to lane 0.

4.4 8B/10B Transmission Code

The 8B/10B transmission code used by the PCS encodes 9-bit characters (8 bits of information and a control bit) into 10-bit code-groups for transmission and reverses the process on reception. Encodings are defined for 256 data characters and 12 special (control) characters.

The code-groups used by the code have either an equal number of ones and zeros (balanced) or the number of ones differs from the number of zeros by two (unbalanced). This selection of code-groups guarantees a minimum of two transitions, 0 to 1 or 1 to 0, within each code-group and it also eases the task of maintaining balance. Characters are encoded into either a single balanced code-group or a pair of unbalanced code-groups. The members of each code-group pair are the logical complement of each other. This allows the encoder, when selecting an unbalanced code-group, to select a code-group unbalanced toward ones or unbalanced toward zeros, depending on which is required to maintain the 0/1 balance of the encoder output code-group stream.

The 8B/10B code has the following properties.

- Sufficient bit transition density (3 to 8 transitions per code-group) to allow clock recovery by the receiver.
- Special code-groups that are used for establishing the receiver synchronization to the 10-bit code-group boundaries, delimiting control symbols and maintaining receiver bit and code-group boundary synchronization.
- Balanced. (can be AC coupled)
- Detection of single and some multiple-bit errors.

4.4.1 Character and Code-Group Notation

The description of 8B/10B encoding and decoding uses the following notation for characters, code-group and their bits.

The information bits ([0-7]) of an unencoded character are denoted with the letters "A" through "H" where the letter "H" denotes the most significant information bit (RapidIO bit 0) and the letter "A" denotes the least significant information bit (RapidIO bit 7). This is shown in Figure .Each data character has a representation of the form Dx.y where x is the decimal value of the least significant 5 information bits EDCBA, and y is the decimal value of the most significant 3 information bits HGF as shown in Figure 4-1. Each special character has a similar representation of theform Kx.y.

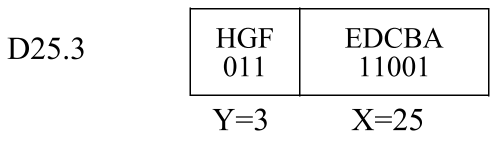


Figure 4-1. Character Notation Example (D25.3)

The output of the 8B/10B encoding process is a 10-bit code-group. The bits of a code-group are denoted with the letters "a" through "j". The bits of a code-group are all of equal significance, there is no most significant or least significant bit. The ordering of the code-group bits is shown in Figure 4-2.

The code-groups corresponding to the data character Dx.y is denoted by /Dx.y/. The code-groups corresponding to the special character Kx.y is denoted by /Kx.y/.

	abcdei	fghj
/D25.3/	100110	1100

Figure 4-2. Code-Group Notation Example (/D25.3/)

4.4.2 Running Disparity

The 8B/10B encoding and decoding functions use a binary variable called running disparity. The variable can have a value of either positive (RD+) or negative (RD-). The encoder and decoder each have a running disparity variable for each lane which are all independent of each other.

The primary use of running disparity in the encoding process is to keep track of whether the decoder has output more ones or more zeros. The current value of encoder running disparity is used to select the which unbalanced code-group will be used when the encoding for a character requires a choice between two unbalanced code-groups.

The primary use of running disparity in the decoding process is to detect errors. Given a value of decoder running disparity, only (256 + 12) = 268 of the 1024 possible code-group values have defined decodings. The remaining 756 possible code-group values have no defined decoding and represent errors, either in that code-group or in an earlier code-group.

4.4.3 Running Disparity Rules

After power-up and before the port is operational, both the transmitter (encoder) and receiver (decoder) must establish current values of running disparity.

The transmitter shall use a negative value as the initial value for the running disparity for each lane.

The receiver may use either a negative or positive initial value of running disparity for each lane.

The following algorithm shall be used for calculating the running disparity for each lane. In the encoder, the algorithm operates on the code-group that has just been generated by the encoder. In the receiver, the algorithm operates on the received code-group that has just been decoded by the decoder.

Each code-group is divided to two sub-blocks as shown in Figure , where the first six bits (abcdei) form one sub-block (6-bit sub-block) and the second four bits (fghj) form a second sub-block (4-bit sub-block). Running disparity at the beginning of the 6-bit sub-block is the running disparity at the end of the previous code-group. Running disparity at the beginning of the 4-bit sub-block is the running disparity at the end of the 6-bit sub-block. Running disparity at the end of the code-group is the running disparity at the end of the 4-bit sub-block.

The sub-block running disparity shall be calculated as follows:

1. The running disparity is positive at the end of any sub-block if the sub-block contains more 1s than 0s. It is also positive at the end of a 4-bit sub-block if the sub-block has the value 0b0011 and at the end of a 6-bit sub-block if the sub-block has the value 0b000111.
2. The running disparity is negative at the end of any sub-block if the sub-block contains more 0s than 1s. It is also negative at the end of a 4-bit sub-block if the sub-block has the value 0b1100 and at the end of a 6-bit sub-block if the sub-block has the value 0b111000.
3. In all other cases, the value of the running disparity at the end of the sub-block is running disparity at the beginning of the sub-block (the running disparity is unchanged).

4.4.4 8B/10B Encoding

The 8B/10B encoding function encodes 9-bit characters into 10-bit code-groups.

The encodings for the 256 data characters (Dx.y) are specified in Table . The encodings for the 12 special characters (Kx.y) are specified in Table . Both tables have two columns of encodings, one marked RD- and one marked RD+. When encoding a character, the code-group in the RD- column is selected if the current value of encoder running disparity is negative and the code-group in the RD+ column is selected if the current value of encoder running disparity is positive.

Data characters (Dx.y) shall be encoded according to Table and the current value of encoder running disparity. Special characters (Kx.y) shall be encoded according to Table and the current value of encoder running disparity. After each character is encoded, the resulting code-group shall be used by the encoder to update the running disparity according to the rules in Section , “4.4.3 Running Disparity Rules.

4.4.5 Transmission Order

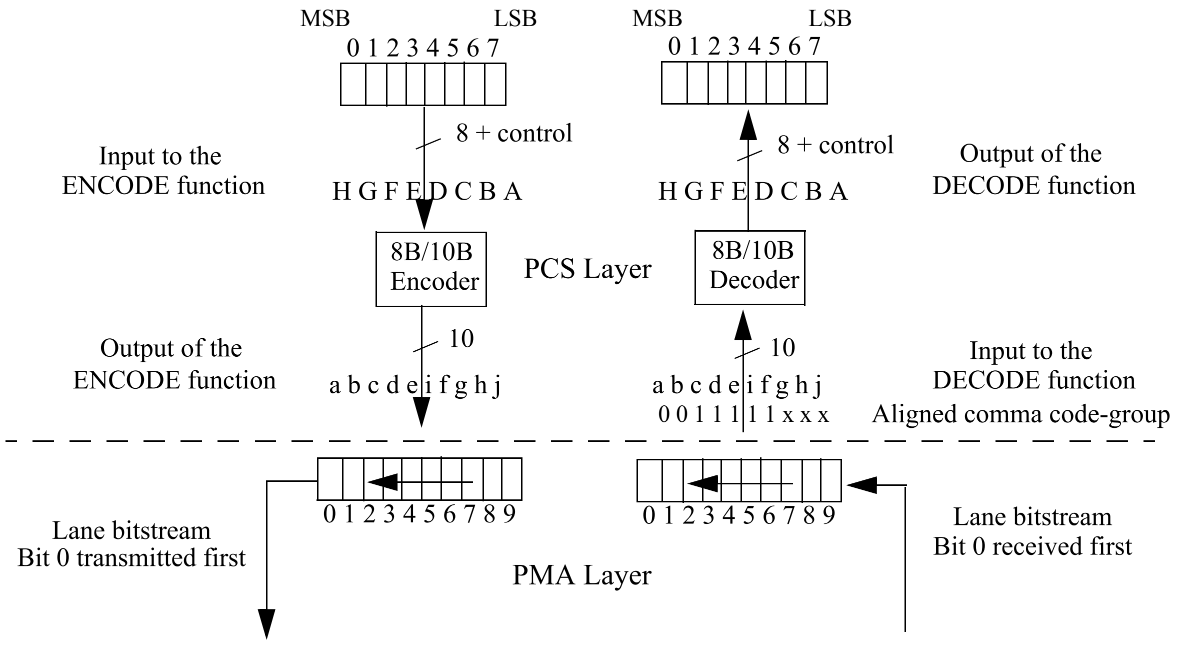
The parallel 10-bit code-group output of the encoder shall be serialized and transmitted with bit “a” transmitted first and a bit ordering of “abcdeifghj”. This is shown in Figure 4-3.

Figure 4-3 gives an overview of a character passing through the encoding, serializing, transmission, deserializing, and decoding processes. The left side of the figure shows the transmit process of encoding a character stream using 8B/10B encoding and the 10-bit serialization. The right side shows the reverse process of the receiver deserializing and using 8B/10B decoding on the received code-groups.

The dotted line shows the functional separation between the PCS layer, that provides 10-bit code-groups, and the PMA layer that serializes the code-groups.

The drawing also shows on the receive side the bits of a special character containing the comma pattern that is used by the receiver to establish 10-bit code-boundary synchronization.

Figure 4-3. Lane Encoding, Serialization, Deserialization, and Decoding Process



4.4.6 8B/10B Decoding

The 8B/10B decoding function decodes received 10-bit code-groups into 9-bit characters, detects received code-groups that have no defined decoding and marks the resulting characters in the output stream of the decode as invalid character (INVALID).

The decoding function uses Table , Table and the current value of the decoder running disparity. To decode a received code-group, the decoder shall select the RD- column of Table and Table if the current value of the decoder running disparity is negative or shall select the RD+ column if the value is positive. The decoder shall then compare the received code-group with the code-groups in the selected column of both tables. If a match is found, the code-group is decoded to the associated character. If no match is found, the code-group is decoded to a character that is flagged in some manner as invalid. After each code-group is decoded, the decoded code-group shall be used by the decoder to update the decoder running disparity according to the rules in Section , “4.4.3 Running Disparity Rules.

Table 4-1. Data Character Encodings

Character Name	Character Value (hex)	Character Bits HGF EDCBA	Current RD –	Current RD +
			abcdei fghj	abcdei fghj
D0.0	00	000 00000	100111 0100	011000 1011
D1.0	01	000 00001	011101 0100	100010 1011
D2.0	02	000 00010	101101 0100	010010 1011
D3.0	03	000 00011	110001 1011	110001 0100
D4.0	04	000 00100	110101 0100	001010 1011
D5.0	05	000 00101	101001 1011	101001 0100
D6.0	06	000 00110	011001 1011	011001 0100
D7.0	07	000 00111	111000 1011	000111 0100
D8.0	08	000 01000	111001 0100	000110 1011
D9.0	09	000 01001	100101 1011	100101 0100
D10.0	0A	000 01010	010101 1011	010101 0100
D11.0	0B	000 01011	110100 1011	110100 0100
D12.0	0C	000 01100	001101 1011	001101 0100
D13.0	0D	000 01101	101100 1011	101100 0100
D14.0	0E	000 01110	011100 1011	011100 0100
D15.0	0F	000 01111	010111 0100	101000 1011
D16.0	10	000 10000	011011 0100	100100 1011
D17.0	11	000 10001	100011 1011	100011 0100
D18.0	12	000 10010	010011 1011	010011 0100
D19.0	13	000 10011	110010 1011	110010 0100
D20.0	14	000 10100	001011 1011	001011 0100
D21.0	15	000 10101	101010 1011	101010 0100
D22.0	16	000 10110	011010 1011	011010 0100
D23.0	17	000 10111	111010 0100	000101 1011
D24.0	18	000 11000	110011 0100	001100 1011
D25.0	19	000 11001	100110 1011	100110 0100
D26.0	1A	000 11010	010110 1011	010110 0100
D27.0	1B	000 11011	110110 0100	001001 1011
D28.0	1C	000 11100	001110 1011	001110 0100
D29.0	1D	000 11101	101110 0100	010001 1011

Table 4-1. Data Character Encodings

Character Name	Character Value (hex)	Character Bits HGF EDCBA	Current RD –	Current RD +
			abcdei fghj	abcdei fghj
D30.0	1E	000 11110	011110 0100	100001 1011
D31.0	1F	000 11111	101011 0100	010100 1011
D0.1	20	001 00000	100111 1001	011000 1001
D1.1	21	001 00001	011101 1001	100010 1001
D2.1	22	001 00010	101101 1001	010010 1001
D3.1	23	001 00011	110001 1001	110001 1001
D4.1	24	001 00100	110101 1001	001010 1001
D5.1	25	001 00101	101001 1001	101001 1001
D6.1	26	001 00110	011001 1001	011001 1001
D7.1	27	001 00111	111000 1001	000111 1001
D8.1	28	001 01000	111001 1001	000110 1001
D9.1	29	001 01001	100101 1001	100101 1001
D10.1	2A	001 01010	010101 1001	010101 1001
D11.1	2B	001 01011	110100 1001	110100 1001
D12.1	2C	001 01100	001101 1001	001101 1001
D13.1	2D	001 01101	101100 1001	101100 1001
D14.1	2E	001 01110	011100 1001	011100 1001
D15.1	2F	001 01111	010111 1001	101000 1001
D16.1	30	001 10000	011011 1001	100100 1001
D17.1	31	001 10001	100011 1001	100011 1001
D18.1	32	001 10010	010011 1001	010011 1001
D19.1	33	001 10011	110010 1001	110010 1001
D20.1	34	001 10100	001011 1001	001011 1001
D21.1	35	001 10101	101010 1001	101010 1001
D22.1	36	001 10110	011010 1001	011010 1001
D23.1	37	001 10111	111010 1001	000101 1001
D24.1	38	001 11000	110011 1001	001100 1001
D25.1	39	001 11001	100110 1001	100110 1001
D26.1	3A	001 11010	010110 1001	010110 1001
D27.1	3B	001 11011	110110 1001	001001 1001

Table 4-1. Data Character Encodings

Character Name	Character Value (hex)	Character Bits HGF EDCBA	Current RD –	Current RD +
			abcdei fghj	abcdei fghj
D28.1	3C	001 11100	001110 1001	001110 1001
D29.1	3D	001 11101	101110 1001	010001 1001
D30.1	3E	001 11110	011110 1001	100001 1001
D31.1	3F	001 11111	101011 1001	010100 1001
D0.2	40	010 00000	100111 0101	011000 0101
D1.2	41	010 00001	011101 0101	100010 0101
D2.2	42	010 00010	101101 0101	010010 0101
D3.2	43	010 00011	110001 0101	110001 0101
D4.2	44	010 00100	110101 0101	001010 0101
D5.2	45	010 00101	101001 0101	101001 0101
D6.2	46	010 00110	011001 0101	011001 0101
D7.2	47	010 00111	111000 0101	000111 0101
D8.2	48	010 01000	111001 0101	000110 0101
D9.2	49	010 01001	100101 0101	100101 0101
D10.2	4A	010 01010	010101 0101	010101 0101
D11.2	4B	010 01011	110100 0101	110100 0101
D12.2	4C	010 01100	001101 0101	001101 0101
D13.2	4D	010 01101	101100 0101	101100 0101
D14.2	4E	010 01110	011100 0101	011100 0101
D15.2	4F	010 01111	010111 0101	101000 0101
D16.2	50	010 10000	011011 0101	100100 0101
D17.2	51	010 10001	100011 0101	100011 0101
D18.2	52	010 10010	010011 0101	010011 0101
D19.2	53	010 10011	110010 0101	110010 0101
D20.2	54	010 10100	001011 0101	001011 0101
D21.2	55	010 10101	101010 0101	101010 0101
D22.2	56	010 10110	011010 0101	011010 0101
D23.2	57	010 10111	111010 0101	000101 0101
D24.2	58	010 11000	110011 0101	001100 0101
D25.2	59	010 11001	100110 0101	100110 0101

Table 4-1. Data Character Encodings

Character Name	Character Value (hex)	Character Bits HGF EDCBA	Current RD –	Current RD +
			abcdei fghj	abcdei fghj
D26.2	5A	010 11010	010110 0101	010110 0101
D27.2	5B	010 11011	110110 0101	001001 0101
D28.2	5C	010 11100	001110 0101	001110 0101
D29.2	5D	010 11101	101110 0101	010001 0101
D30.2	5E	010 11110	011110 0101	100001 0101
D31.2	5F	010 11111	101011 0101	010100 0101
D0.3	60	011 00000	100111 0011	011000 1100
D1.3	61	011 00001	011101 0011	100010 1100
D2.3	62	011 00010	101101 0011	010010 1100
D3.3	63	011 00011	110001 1100	110001 0011
D4.3	64	011 00100	110101 0011	001010 1100
D5.3	65	011 00101	101001 1100	101001 0011
D6.3	66	011 00110	011001 1100	011001 0011
D7.3	67	011 00111	111000 1100	000111 0011
D8.3	68	011 01000	111001 0011	000110 1100
D9.3	69	011 01001	100101 1100	100101 0011
D10.3	6A	011 01010	010101 1100	010101 0011
D11.3	6B	011 01011	110100 1100	110100 0011
D12.3	6C	011 01100	001101 1100	001101 0011
D13.3	6D	011 01101	101100 1100	101100 0011
D14.3	6E	011 01110	011100 1100	011100 0011
D15.3	6F	011 01111	010111 0011	101000 1100
D16.3	70	011 10000	011011 0011	100100 1100
D17.3	71	011 10001	100011 1100	100011 0011
D18.3	72	011 10010	010011 1100	010011 0011
D19.3	73	011 10011	110010 1100	110010 0011
D20.3	74	011 10100	001011 1100	001011 0011
D21.3	75	011 10101	101010 1100	101010 0011
D22.3	76	011 10110	011010 1100	011010 0011
D23.3	77	011 10111	111010 0011	000101 1100

Table 4-1. Data Character Encodings

Character Name	Character Value (hex)	Character Bits HGF EDCBA	Current RD –	Current RD +
			abcdei fghj	abcdei fghj
D24.3	78	011 11000	110011 0011	001100 1100
D25.3	79	011 11001	100110 1100	100110 0011
D26.3	7A	011 11010	010110 1100	010110 0011
D27.3	7B	011 11011	110110 0011	001001 1100
D28.3	7C	011 11100	001110 1100	001110 0011
D29.3	7D	011 11101	101110 0011	010001 1100
D30.3	7E	011 11110	011110 0011	100001 1100
D31.3	7F	011 11111	101011 0011	010100 1100
D0.4	80	100 00000	100111 0010	011000 1101
D1.4	81	100 00001	011101 0010	100010 1101
D2.4	82	100 00010	101101 0010	010010 1101
D3.4	83	100 00011	110001 1101	110001 0010
D4.4	84	100 00100	110101 0010	001010 1101
D5.4	85	100 00101	101001 1101	101001 0010
D6.4	86	100 00110	011001 1101	011001 0010
D7.4	87	100 00111	111000 1101	000111 0010
D8.4	88	100 01000	111001 0010	000110 1101
D9.4	89	100 01001	100101 1101	100101 0010
D10.4	8A	100 01010	010101 1101	010101 0010
D11.4	8B	100 01011	110100 1101	110100 0010
D12.4	8C	100 01100	001101 1101	001101 0010
D13.4	8D	100 01101	101100 1101	101100 0010
D14.4	8E	100 01110	011100 1101	011100 0010
D15.4	8F	100 01111	010111 0010	101000 1101
D16.4	90	100 10000	011011 0010	100100 1101
D17.4	91	100 10001	100011 1101	100011 0010
D18.4	92	100 10010	010011 1101	010011 0010
D19.4	93	100 10011	110010 1101	110010 0010
D20.4	94	100 10100	001011 1101	001011 0010
D21.4	95	100 10101	101010 1101	101010 0010

Table 4-1. Data Character Encodings

Character Name	Character Value (hex)	Character Bits HGF EDCBA	Current RD –	Current RD +
			abcdei fghj	abcdei fghj
D22.4	96	100 10110	011010 1101	011010 0010
D23.4	97	100 10111	111010 0010	000101 1101
D24.4	98	100 11000	110011 0010	001100 1101
D25.4	99	100 11001	100110 1101	100110 0010
D26.4	9A	100 11010	010110 1101	010110 0010
D27.4	9B	100 11011	110110 0010	001001 1101
D28.4	9C	100 11100	001110 1101	001110 0010
D29.4	9D	100 11101	101110 0010	010001 1101
D30.4	9E	100 11110	011110 0010	100001 1101
D31.4	9F	100 11111	101011 0010	010100 1101
D0.5	A0	101 00000	100111 1010	011000 1010
D1.5	A1	101 00001	011101 1010	100010 1010
D2.5	A2	101 00010	101101 1010	010010 1010
D3.5	A3	101 00011	110001 1010	110001 1010
D4.5	A4	101 00100	110101 1010	001010 1010
D5.5	A5	101 00101	101001 1010	101001 1010
D6.5	A6	101 00110	011001 1010	011001 1010
D7.5	A7	101 00111	111000 1010	000111 1010
D8.5	A8	101 01000	111001 1010	000110 1010
D9.5	A9	101 01001	100101 1010	100101 1010
D10.5	AA	101 01010	010101 1010	010101 1010
D11.5	AB	101 01011	110100 1010	110100 1010
D12.5	AC	101 01100	001101 1010	001101 1010
D13.5	AD	101 01101	101100 1010	101100 1010
D14.5	AE	101 01110	011100 1010	011100 1010
D15.5	AF	101 01111	010111 1010	101000 1010
D16.5	B0	101 10000	011011 1010	100100 1010
D17.5	B1	101 10001	100011 1010	100011 1010
D18.5	B2	101 10010	010011 1010	010011 1010
D19.5	B3	101 10011	110010 1010	110010 1010

Table 4-1. Data Character Encodings

Character Name	Character Value (hex)	Character Bits HGF EDCBA	Current RD –	Current RD +
			abcdei fghj	abcdei fghj
D20.5	B4	101 10100	001011 1010	001011 1010
D21.5	B5	101 10101	101010 1010	101010 1010
D22.5	B6	101 10110	011010 1010	011010 1010
D23.5	B7	101 10111	111010 1010	000101 1010
D24.5	B8	101 11000	110011 1010	001100 1010
D25.5	B9	101 11001	100110 1010	100110 1010
D26.5	BA	101 11010	010110 1010	010110 1010
D27.5	BB	101 11011	110110 1010	001001 1010
D28.5	BC	101 11100	001110 1010	001110 1010
D29.5	BD	101 11101	101110 1010	010001 1010
D30.5	BE	101 11110	011110 1010	100001 1010
D31.5	BF	101 11111	101011 1010	010100 1010
D0.6	C0	110 00000	100111 0110	011000 0110
D1.6	C1	110 00001	011101 0110	100010 0110
D2.6	C2	110 00010	101101 0110	010010 0110
D3.6	C3	110 00011	110001 0110	110001 0110
D4.6	C4	110 00100	110101 0110	001010 0110
D5.6	C5	110 00101	101001 0110	101001 0110
D6.6	C6	110 00110	011001 0110	011001 0110
D7.6	C7	110 00111	111000 0110	000111 0110
D8.6	C8	110 01000	111001 0110	000110 0110
D9.6	C9	110 01001	100101 0110	100101 0110
D10.6	CA	110 01010	010101 0110	010101 0110
D11.6	CB	110 01011	110100 0110	110100 0110
D12.6	CC	110 01100	001101 0110	001101 0110
D13.6	CD	110 01101	101100 0110	101100 0110
D14.6	CE	110 01110	011100 0110	011100 0110
D15.6	CF	110 01111	010111 0110	101000 0110
D16.6	D0	110 10000	011011 0110	100100 0110
D17.6	D1	110 10001	100011 0110	100011 0110

Table 4-1. Data Character Encodings

Character Name	Character Value (hex)	Character Bits HGF EDCBA	Current RD –	Current RD +
			abcdei fghj	abcdei fghj
D18.6	D2	110 10010	010011 0110	010011 0110
D19.6	D3	110 10011	110010 0110	110010 0110
D20.6	D4	110 10100	001011 0110	001011 0110
D21.6	D5	110 10101	101010 0110	101010 0110
D22.6	D6	110 10110	011010 0110	011010 0110
D23.6	D7	110 10111	111010 0110	000101 0110
D24.6	D8	110 11000	110011 0110	001100 0110
D25.6	D9	110 11001	100110 0110	100110 0110
D26.6	DA	110 11010	010110 0110	010110 0110
D27.6	DB	110 11011	110110 0110	001001 0110
D28.6	DC	110 11100	001110 0110	001110 0110
D29.6	DD	110 11101	101110 0110	010001 0110
D30.6	DE	110 11110	011110 0110	100001 0110
D31.6	DF	110 11111	101011 0110	010100 0110
D0.7	E0	111 00000	100111 0001	011000 1110
D1.7	E1	111 00001	011101 0001	100010 1110
D2.7	E2	111 00010	101101 0001	010010 1110
D3.7	E3	111 00011	110001 1110	110001 0001
D4.7	E4	111 00100	110101 0001	001010 1110
D5.7	E5	111 00101	101001 1110	101001 0001
D6.7	E6	111 00110	011001 1110	011001 0001
D7.7	E7	111 00111	111000 1110	000111 0001
D8.7	E8	111 01000	111001 0001	000110 1110
D9.7	E9	111 01001	100101 1110	100101 0001
D10.7	EA	111 01010	010101 1110	010101 0001
D11.7	EB	111 01011	110100 1110	110100 1000
D12.7	EC	111 01100	001101 1110	001101 0001
D13.7	ED	111 01101	101100 1110	101100 1000
D14.7	EE	111 01110	011100 1110	011100 1000
D15.7	EF	111 01111	010111 0001	101000 1110

Table 4-1. Data Character Encodings

Character Name	Character Value (hex)	Character Bits HGF EDCBA	Current RD –	Current RD +
			abcdei fghj	abcdei fghj
D16.7	F0	111 10000	011011 0001	100100 1110
D17.7	F1	111 10001	100011 0111	100011 0001
D18.7	F2	111 10010	010011 0111	010011 0001
D19.7	F3	111 10011	110010 1110	110010 0001
D20.7	F4	111 10100	001011 0111	001011 0001
D21.7	F5	111 10101	101010 1110	101010 0001
D22.7	F6	111 10110	011010 1110	011010 0001
D23.7	F7	111 10111	111010 0001	000101 1110
D24.7	F8	111 11000	110011 0001	001100 1110
D25.7	F9	111 11001	100110 1110	100110 0001
D26.7	FA	111 11010	010110 1110	010110 0001
D27.7	FB	111 11011	110110 0001	001001 1110
D28.7	FC	111 11100	001110 1110	001110 0001
D29.7	FD	111 11101	101110 0001	010001 1110
D30.7	FE	111 11110	011110 0001	100001 1110
D31.7	FF	111 11111	101011 0001	010100 1110

Table 4-2. Special Character Encodings

Character Name	Character Value (hex)	Character Bits HGF EDCBA	Current RD –	Current RD +	Notes
			abcdei fghj	abcdei fghj	
K28.0	1C	000 11100	001111 0100	110000 1011	
K28.1	3C	001 11100	001111 1001	110000 0110	1,2
K28.2	5C	010 11100	001111 0101	110000 1010	1
K28.3	7C	011 11100	001111 0011	110000 1100	
K28.4	9C	100 11100	001111 0010	110000 1101	1
K28.5	BC	101 11100	001111 1010	110000 0101	2
K28.6	DC	110 11100	001111 0110	110000 1001	1
K28.7	FC	111 11100	001111 1000	110000 0111	1,2
K23.7	F7	111 10111	111010 1000	000101 0111	1

Table 4-2. Special Character Encodings

Character Name	Character Value (hex)	Character Bits HGF EDCBA	Current RD –	Current RD +	Notes
			abcdei fghj	abcdei fghj	
K27.7	FB	111 11011	110110 1000	001001 0111	
K29.7	FD	111 11101	101110 1000	010001 0111	
K30.7	FE	111 11110	011110 1000	100001 0111	1

1 - Reserved code-groups.
2 - The code-groups /K28.5/, /K28.7/, and /K28.1/ contain a comma.

4.4.7 Special Characters and Columns

Table 4-3 defines the special characters and columns of special characters used by 1x and 4x LP-Serial links. Special characters are used for the following functions:

1. Alignment to code-group (10-bit) boundaries on lane-by-lane basis.
2. Alignment of the receive data stream across four lanes.
3. Clock rate compensation between receiver and transmitter.
4. Control symbol delimiting.

Table 4-3. Special Characters and Columns

Code-Group/Column Designation	Code-Group/Column Use	Number of Code-groups	Encoding
/PD/	Packet_Delimiter Control Symbol	1	/K28.3/
/SC/	Start_of_Control_Symbol	1	/K28.0/
/**I**/	Idle		
/K/	1x Sync	1	/K28.5/
/R/	1x Skip	1	/K29.7/
/A/	1x Align	1	/K27.7/
||**I**||	Idle column		
||K||	4x Sync column	4	/K28.5/K28.5/K28.5/K28.5/
||R||	4x Skip column	4	/K29.7/K29.7/K29.7/K29.7/
||A||	4x Align column	4	/K27.7/K27.7/K27.7/K27.7/

4.4.7.1 Packet Delimiter Control Symbol (/PD/)

PD and /PD/ are aliases for respectively the K28.3 character and the /K28.3/ code-group which are used to delimit the beginning of a control symbol that contains a packet delimiter.

4.4.7.2 Start of Control Symbol (/SC/)

SC and /SC/ are aliases for respectively the K28.0 character and the /K28.0/ code-group which are used to delimit the beginning of a control symbol that does not contain a packet delimiter.

4.4.7.3 Idle (/I/)

I and /I/ are aliases for respectively any of the idle sequence characters (A, K, or R) and idle sequence code-groups (/A/, /K/, or /R/).

4.4.7.4 Sync (/K/)

K and /K/ are aliases for respectively the K28.5 character and the /K28.5/ code-group which is used in the idle sequence to provide the receiver with the information its requires to achieve and maintain bit and 10-bit code-group boundary synchronization. The /K28.5/ code-group was selected as the Sync character for the following reasons:

1. It contains the comma pattern in bits **abcdeif** which can be easily found in the code-group bit stream and marks the code-group boundary.
2. The bits **ghj** provide the maximum number of transitions (i.e. 101 or 010).

A comma is a 7-bit string defined as either b'0011111' (comma+) or b'1100000' (comma-). Within the code-group set it is a singular bit pattern, which, in the absence of transmission errors, cannot appear in any other location of a code-group and cannot be generated across the boundaries of any two adjacent code-groups with the following exception:

NOTE:

The /K28.7/ special code-group when followed by any of the data code-groups /D3.y/, /D11.y/, /D12.y/, /D19.y/,

/D20.y/, /D28.y/, or /K28.y/, where y is an integer in the range 0 through 7, may (depending on the value of running disparity) cause a comma to be generated across the boundary of the two code-groups. A comma that is generated across the boundary between two adjacent code-groups may cause the receiver to change the 10-bit code-group alignment. As a result, the /K28.7/ special code-group may be used for test and diagnostic purposes only.

4.4.7.5 Skip (/R/)

R and /R/ are aliases for respectively the K29.7 character and the /29.7/ code-group which are used in the idle sequence and are also used in the clock compensation sequence.

4.4.7.6 Align (/A/)

A and /A/ are aliases for respectively the K27.7 character and the /27.7/ code-group which are used in the idle sequence and are used for lane alignment on 4x links.

4.4.8 Effect of Single Bit Code-Group Errors

Single bit code-group errors will be the dominant code-group error by many orders of magnitude. It is therefore useful to know the variety of code-group corruptions that can be caused by a single bit error.

Table lists all possible code-group corruptions that can be caused by a single-bit error. The notation /X/ => /Y/ means that the code-group for the character X has been corrupted by a single-bit error into the code-group for the character Y. If the corruption results in a code-group that is invalid for the current receiver running disparity, the notation /X/ => /INVALID/ is used. The table provides the information required to deterministically detect all isolated single bit transmission errors.

Table 4-4. Code-Group Corruption Caused by Single Bit Errors

Corruption	Detection
/SC/ => /INVALID/	Detectable as an error when decoding the code-group. When this error occurs within a packet, it is indistinguishable from a /Dx.y/ => /INVALID/. When this error occurs outside of a packet, the type of error can be inferred from whether the /INVALID/ is followed by the three /Dx.y/ that comprise the control symbol data.
/PD/ => /INVALID/	Detectable as an error when decoding the code-group. When this error occurs within a packet, it is indistinguishable from a /Dx.y/ => /INVALID/. When this error occurs outside of a packet, the type of error can be inferred from whether the /INVALID/ is followed by the three /Dx.y/ that comprise the control symbol data.
/A/, /K/ or /R/ => /Dx.y/	Detectable as an error as /Dx.y/ is illegal outside of a packet or control symbol and /A/, /K/ and /R/ are illegal within a packet or control symbol.
/A/, /K/ or /R/ => /INVALID/	Detectable as an error when decoding the code-group.
/Dx.y/ => /A/, /K/ or /R/	Detectable as an error as /A/, /K/ and /R/ are illegal within a packet or control symbol and /Dx.y/ is illegal outside of a packet or control symbol.
/Dx.y/ => /INVALID/	Detectable as an error when decoding the code-group.
/Dx.y/ => /Du.v/	Detectable as an error by the packet or control symbol CRC. The error will also result in a subsequent unerrored code-group being decoded as INVALID, but that resulting INVALID code-group may occur an arbitrary number of code-groups after the errored code-group.

4.4.9 Idle Sequence

The idle sequence is a sequence of code-groups that shall be transmitted continuously over each lane of an LP-Serial link whenever neither packets nor control symbols are not being transmitted. An idle sequence may not be inserted in a packet or delimited control symbol. The idle sequence is transmitted over each lane as part of the port initialization process as required in Section : "4.6.3.5 1x Mode Initialization State Machine" and Section : "4.6.3.6 1x/4x Mode Initialization State Machine". An idle sequence containing a special clock compensation sequence shall be transmitted at least once every 5000 code-groups even when there are packets or control symbols available to transmit to allow clock rate compensation.

The 1x idle sequence consists of a sequence of the code-groups /K/, /A/, and /R/ (the idle code-groups) and shall be used by ports in operating is 1x mode. The 4x idle sequence consists of a sequence of the columns ||K||, ||A||, ||R|| (the idle columns) and shall be used by ports operating in 4x mode. Both sequences shall comply with the following requirements:

1. The first code-group (column) of an idle sequence generated by a port operating in 1x mode (4x mode) shall be a /K/ (||K||). The first code-group (column) shall be transmitted immediately following the last code-group (column) of a packet or delimited control symbol.
2. At least once every 5000 code-groups (columns) transmitted by a port operating in 1x mode (4x mode), an idle sequence containing the /K/R/R/R/ code-group sequence (||K||R||R||R|| column sequence) shall be transmitted by

the port. This sequence is referred to as the “compensation sequence”.

3. When not transmitting the compensation sequence, all code-groups (columns) following the first code-group (column) of an idle sequence generated by a port operating in 1x mode (4x mode) shall be a pseudo-randomly selected sequence of /A/, /K/, and /R/ (||A||, ||K||, and ||R||) based on a pseudo-random sequence generator of 7th order or greater and subject to the minimum and maximum /A/ (||A||) spacing requirements.

4. The number of non /A/ code-groups (non ||A|| columns) between /A/ code-groups (||A|| columns) in the idle sequence of a port operating in 1x mode (4x mode) shall be no less than 16 and no more than 32. The number shall be pseudo-randomly selected, uniformly distributed across the range and based on a pseudo-random sequence generator of 7th order or greater.

There are no requirements on the length of an idle sequence. An idle sequence may be of any length.

The idle sequence is transmitted on each lane of a link when neither packets nor delimited control symbols are being transmitted to allow each lane receiver to maintain bit and 10-bit code-group boundary synchronization.

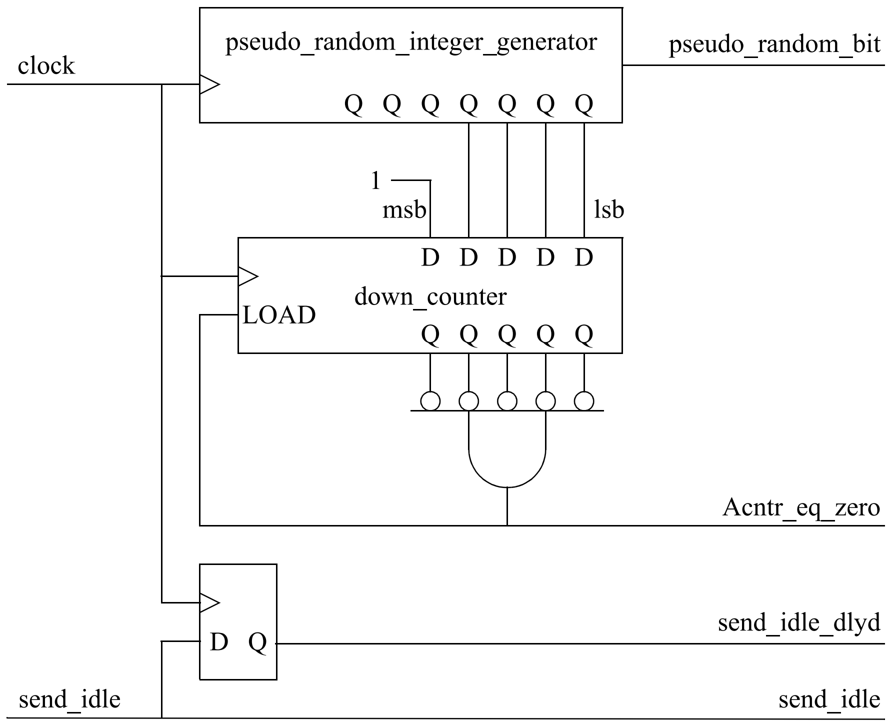
The pseudo-random selection of code-groups in the idle sequence results in an idle sequence whose spectrum has no discrete lines which minimizes the EMI of long idle sequences.

The compensation sequence allows retiming repeaters (discussed in Section : "4.5 Retimers and Repeaters") to compensate for up to a +/- 200 ppm difference between input bit rate and output bit rate, each of which have a +/-100 ppm tolerance. It may also be used to allow the input side of a port to compensate for up to a +/-200 ppm difference between the input bit rate and the bit rate of the device core which may be running off a different clock. This is done by dropping or adding an /R/ or ||R|| as needed to avoid overrun/underrun. Since a packet or delimited control symbol may not be interrupted by an idle sequence, designers must be careful to guarantee that no more than 5000 code-groups are transmitted between compensation sequences.

4.4.9.1 Idle Sequence Generation

The 7th order polynomial $x^7+x^6+x^1$ is recommended as the generating polynomial for the pseudo-random sequence that is used in the generation of the idle sequence. The pseudo-random sequence generator is clocked (generates a new pseudo-random sequence value) once per idle sequence code-group (column). Four of the pseudo-random sequence generator state bits may be selected as the pseudo-random value for /A/ (||A||) spacing and any other state bit or logical function of state bits may be selected as the /K/ vs. /R/ selector.

Figure shows an example circuit illustrating how this may be done. The clock ticks whenever a code-group or column is transmitted. Send_idle is asserted whenever an idle sequence begins. The equations indicate the states in which to transmit the indicated idle code-group, except when the compensation sequence is being transmitted. Any equivalent method is acceptable.



send_K = send_idle & (!send_idle_dlyd | send_idle_dlyd & !Acntr_eq_zero & pseudo_random_bit)
send_A = send_idle & send_idle_dlyd & Acntr_eq_zero
send_R = send_idle & send_idle_dlyd & !Acntr_eq_zero & !pseudo_random_bit

Figure 4-4. Example of a Pseudo-Random Idle Code-Group Generator

4.4.10 1x Link Transmission Rules

A 1x LP-Serial link has a single lane (differential pair) in each direction. A 1x LP-Serial port shall be encoded and transmit the character stream of delimited control symbols and packets received from the upper layers over the differential pair in the order the characters were received from the upper layers. When neither control symbols nor packets are available from the upper layers for transmission, the 1x idle sequence shall be fed to the input of the encoder for encoding and transmission. On reception, the code-group stream is decoded and passed to the upper layers.

When a 4x port is operating in 1x mode, the character stream from the upper layers is not striped across the lanes before encoding as is done when operating in 4x mode. The entire character stream from the upper layers is fed in parallel to both lanes 0 and 2. A 4x LP-Serial port operating in 1x mode shall encoded and transmit the character stream of delimited control symbols and packets received from the upper layers over both lanes 0 and 2 in the order the characters were received from the upper layers.

When neither delimited control symbols nor packets are available from the upper layers for transmission, the 1x idle sequence shall be fed in parallel to the input of the lane 0 and lane 2 encoders for encoding and transmission on lanes 0 and 2. On reception, the code-group stream from either lane 0 or 2 is selected according to the state of the 1x/4x_Initialization state machine (Section , “4.6.3.6 1x/4x Mode Initialization State Machine), decoded and passed to the upper layers.

Figure shows the encoding and transmission order for a control symbol transmitted over a 1x LP-Serial link.

Figure 4-5. 1x Mode Control Symbol Encoding and Transmission Order

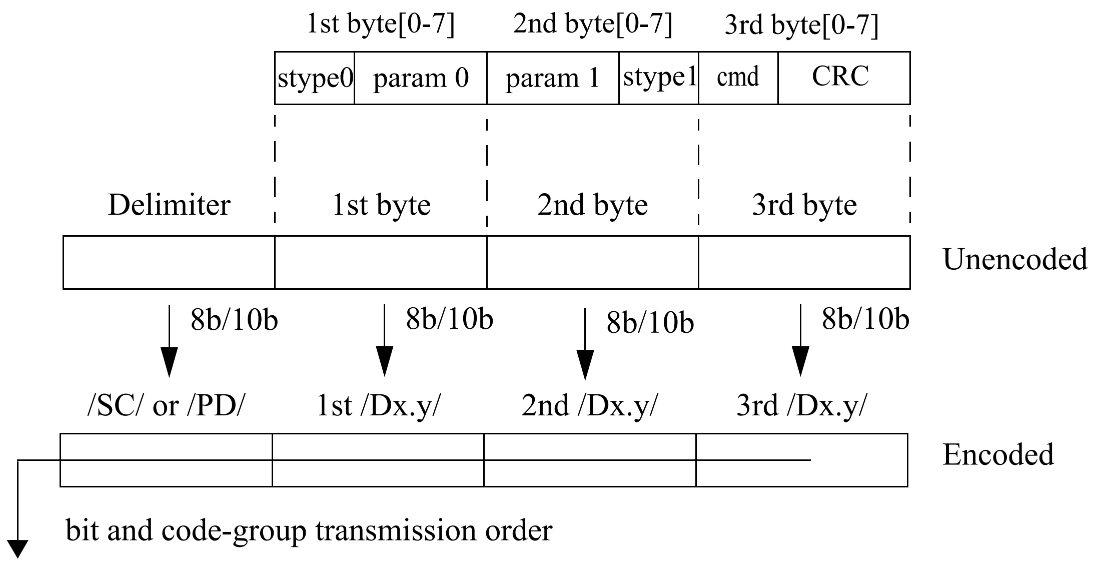


Figure shows the encoding and transmission order for a packet transmitted over a 1x LP-Serial link.

Figure 4-6. 1x Mode Packet Encoding and Transmission Order

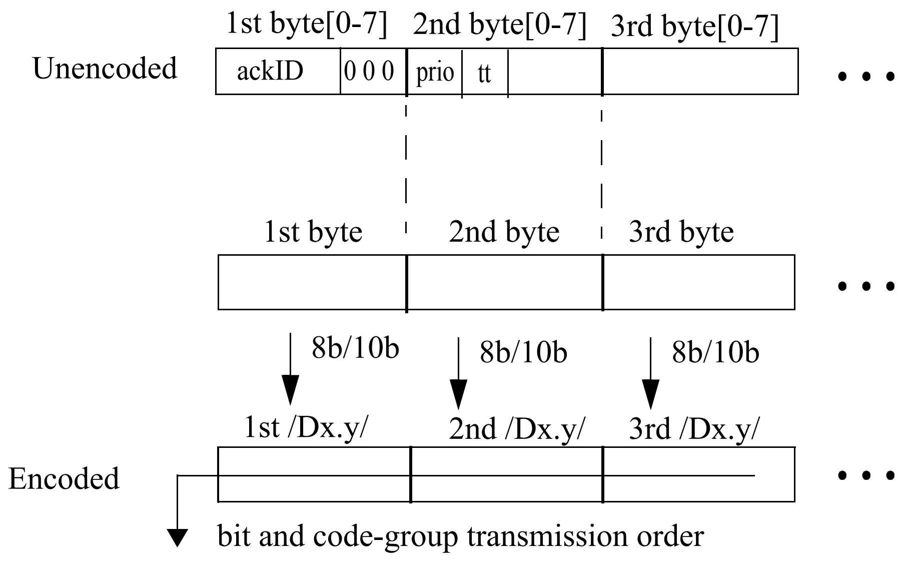


Figure 4-7 shows an example of control symbol, packet, and idle sequence transmission on a 1x LP-Serial link.

Figure 4-7. 1x Typical Data Flow

/SC/	char-0
Cdata-0	char-1
Cdata-1	char-2
Cdata-2	char-3
/I/	char-0
/I/	char-1
/I/	char-2
/I/	char-3
/PD/	char-0
Control	char-1
Symbol	char-2
(start-pkt)	char-3
Data-0	char-0
Data-1	char-1
Data-2	char-2
Data-3	char-3
Data-4	char-0
Data-5	char-1
Data-6	char-2
Data-7	char-3

/PD/	char-0
Control	char-1
Symbol	char-2
(start pkt)	char-3
Data-0	char-0
Data-1	char-1
Data-2	char-2
Data-3	char-3
Data-4	char-0
Data-5	char-1
Data-6	char-2
Data-7	char-3
Data-8	char-0
Data-9	char-1
Data-10	char-2
Data-11	char-3
/SC/	char-0
Cdata-0	char-1
Cdata-1	char-2
Cdata-2	char-3

/PD/	char-0
Control	char-1
Symbol	char-2
(end pkt)	char-3
/PD/	char-0
Control	char-1
Symbol	char-2
(start pkt)	char-3
Data-0	char-0
Data-1	char-1
Data-2	char-2
Data-3	char-3
Data-4	char-0
Data-5	char-1
Data-6	char-2
Data-7	char-3
/SC/	char-0
Cdata-0	char-1
Cdata-1	char-2
Cdata-2	char-3

Data-8	char-0
Data-9	char-1
Data-10	char-2
Data-11	char-3
/SC/	char-0
Cdata-0	char-1
Cdata-1	char-2
Cdata-2	char-3
Data-12	char-0
Data-13	char-1
Data-14	char-2
Data-15	char-3
/PD/	char-0
Restart	char-1
from	char-2
Retry	char-3
/PD/	char-0
Control	char-1
Symbol	char-2
(start pkt)	char-3

/SC/	char-0
Cdata-0	char-1
Cdata-1	char-2
Cdata-2	char-3
Data-0	char-0
Data-1	char-1
Data-2	char-2
Data-3	char-3
Data-4	char-0
Data-5	char-1
Data-6	char-2
Data-7	char-3
/PD/	char-0
Control	char-1
Symbol	char-2
(end pkt)	char-3
/I/	char-0
/I/	char-1
/I/	char-2
/I/	char-3

4.4.11 4x Link Striping and Transmission Rules

A LP-Serial port operating in 4x mode shall stripe the character stream of delimited control symbols and packets that it receives from the upper layers across the four lanes before 8B/10B encoding as follows:

> Packets and delimited control symbols shall be striped across the four lanes beginning with lane 0. The first character of each packet, or delimited control symbol, is placed in lane 0, the second character is placed in lane1, the third character is placed in lane 2, and the fourth character is placed in lane 3. The fifth character and subsequent characters wrap around and continue beginning in lane 0.

As a result of their defined lengths, delimited control symbols will form a column after striping and a packet will form an integer number of contiguous columns.

After striping, each of the 4 streams of characters shall be independently 8B/10B encoded and transmitted.

When neither delimited control symbols nor packets are available from the upper layers for transmission, the 4x idle sequence shall be transmitted. This can be achieved by feeding the 1x idle sequence in parallel to the inputs of the encoders for all four lanes for encoding and transmission on the four lanes. Feeding the 1x idle sequence in parallel to the input of all four lane encoders converts each 1x idle sequence character in to a 4x idle sequence column. The 1x sequence is not striped across the four lanes.

On reception, each lane shall be 8B/10B decoded. After decoding, the four lanes shall be aligned. The ||A|| columns transmitted as part of the 4x idle sequence provide the information needed to perform alignment. After alignment, the columns are destriped into a single character stream and passed to the upper layers.

The lane alignment process eliminates the skew between lanes so that after destriping, the ordering of characters in the received character stream is the same as the ordering of characters before striping and transmission. Since the minimum number of non ||A|| columns between ||A|| columns is 16, the maximum lane skew that can be unambiguously corrected is the time it takes for to transmit 7 code-groups on a lane.

Figure 4-8. shows an example of idle sequence, packet, and delimited control symbol transmission on a 4x link.

Figure 4-8. Typical 4x Data Flow

Lane-0	Lane-1	Lane-2	Lane-3
/SC/	Cdata-0	Cdata-1	Cdata-2
/I/	/I/	/I/	/I/
/PD/	Control Symbol (Start-of-packet)		
Data-0	Data-1	Data-2	Data-3
Data-4	Data-5	Data-6	Data-7
/PD/	Control Symbol (Start-of-packet)		
Data-0	Data-1	Data-2	Data-3
Data-4	Data-5	Data-6	Data-7
Data-8	Data-9	Data-10	Data-11
/SC/	Cdata-0	Cdata-1	Cdata-2
/PD/	Control Symbol (End-of-packet)		
/PD/	Control Symbol (Start-of-packet)		
Data-0	Data-1	Data-2	Data-3
Data-4	Data-5	Data-6	Data-7
/SC/	Cdata-0	Cdata-1	Cdata-2
Data-8	Data-9	Data-10	Data-11
/SC/	Cdata-0	Cdata-1	Cdata-2

Figure 4-8. Typical 4x Data Flow

Lane-0	Lane-1	Lane-2	Lane-3
Data-12	Data-13	Data-14	Data-15
/PD/	Control Symbol (Restart-from-retry)		
/PD/	Control Symbol (Start-of-packet)		
/SC/	Cdata-0	Cdata-1	Cdata-2
Data-0	Data-1	Data-2	Data-3
Data-4	Data-5	Data-6	Data-7
/PD/	Control Symbol (End-of-packet)		
/I/	/I/	/I/	/I/

4.5 Retimers and Repeaters

The LP-Serial Specification allows “retimers” and “repeaters”. Retimers amplify a weakened signal, but do not transfer jitter to the next segment. Repeaters also amplify a weakened signal, but transfer jitter to the next segment. Retimers allow greater distances between end points at the cost of additional latency. Repeaters support less distance between end points than retimers, only add a small amount of latency.

4.5.1 Retimers

A retimer shall comply with all AC specifications found in Chapter 8, “AC Electrical Specifications”. This includes resetting the jitter budget thus extending the transmission distance for the link. The retimer repeats the received code-groups after performing code-group synchronization and serializes the bitstream again on transmission, based on a local clock reference. Up to two retimers are allowed between two end nodes.

A retimer is not RapidIO protocol-aware or addressable in any way. The only awareness a retimer has is to the synchronization on the /K/ code-group and the function of /R/ insertion and removal. A retimer may insert up to one /R/ code-group immediately following a /K/ code-group sequence, or remove one /R/ code-group that immediately follows a /K/ code-group sequence. Note that the /R/ code-group is disparity neutral and therefore its insertion or deletion does not affect the running disparity.

A 4-lane retimer must perform lane synchronization and deskew, in exactly the same way a RapidIO device implementing this physical layer does when synchronizing inputs during initialization and startup. A 4-lane retimer will synchronize and align all lanes that are driven to it. Therefore, such a retimer allows for the degradation of an input 4x link to a 1x link on either lane 0 or 2. If any link drops out, the retimer must merely continue to pass the active links, monitoring for the compensation sequence and otherwise passing through whatever code-groups appear on its inputs. A retimer may optionally not drive any outputs whose corresponding inputs are not active.

Any insertion or removal of a /R/ code-groups in a 4-lane retimer must be done on a full column. A retimer may retime links operating at the same width only (i.e. cannot connect a link operating at 1x to a link operating at 4x). A retimer may connect a 1x link to a 4x link that is operating at 1x. Retimers perform clock tolerance compensation between the receive and transmit clock. The transmit clock is usually derived from a local reference.

Retimers do not check for code violations. Code-groups received on one port are transmitted on the other regardless of code violations or running disparity errors.

4.5.2 Repeaters

A repeater is used to amplify the signal, but does not retime the signal, and therefore can add additional jitter to the signal. It does not compensate for clock rate variation. The repeater repeats the received code-groups as the bits are received by sampling the incoming bits with a clock derived from the bit stream, and then retransmitting them based on that clock. Repeaters may be used with 4x links but lane-to-lane skew may be amplified. Repeaters do not interpret or alter the bit stream in any way.

4.6 Port Initialization

This section specifies the port initialization process. The process includes detecting the presence of a partner at the other end of the link (a link partner), establishing bit synchronization and code-group boundary alignment and if the port is capable of supporting both 1x and 4x modes (a 1x/4x port), discovering whether the link partner is capable of 4x mode (4-lane) operation, selecting 1x or 4x mode operation and if 1x mode is selected, selecting lane 0 or lane 2 for link reception.

The initialization process is controlled by several state machines. The number and type of state machines depends on whether the port supports only 1x mode (a 1x port) or supports both 1x and 4x modes (a 1x/4x port). In either case, there is a primary state machine and one or more secondary state machines. The use of multiple state machines results in a simpler overall design. As might be expected, the initialization process for a 1x port is simpler than and a subset of the initialization process for a 1x/4x port.

The initialization process for 1x and 1x/4x ports is both described in text and specified with state machine diagrams. **In the case of conflict between the text and a state machine diagram, the state machine diagram takes precedence.**

4.6.1 1x Mode Initialization

The initialization process for ports that support only 1x mode shall be controlled by two state machines, 1x_Initialization and Lane_Synchronization. 1x_Initialization is the primary state machine and Lane_Synchronization is the secondary state machine. The operation of these state machines is described and specified in Section : "4.6.3.5 1x Mode Initialization State Machine" and Section : "4.6.3.3 Lane Synchronization State Machine" respectively.

After a 1x port has been initialized, the port is required to receive seven error-free status control symbols without intervening detected errors to verify link operation before transmitting any packets.

4.6.2 1x/4x Mode Initialization

The initialization process for ports that support both 1x and 4x modes shall be controlled by six state machines, 1x/4x_Initialization, Lane_Synchronization[0-3] (one for each of the four lanes) and Lane_Alignment. 1x/4x_Initialization is the primary state machine. Lane_Synchronization[0-3] and Lane_Alignment are the secondary state machines. The operation of these state machines is described and specified in Section : "4.6.3.6 1x/4x Mode Initialization State Machine", Section : "4.6.3.3 Lane Synchronization State Machine" and Section 0.2.3.4: "Lane Alignment State Machine" respectively.

After a 4x port has been initialized, the port is required to receive seven error-free status control symbols without intervening detected errors to verify link operation before transmitting any packets.

4.6.3 State Machines

4.6.3.1 State Machine Conventions

A state machine state is persistent until an exit condition occurs.

A state machine variable that is listed in the body of a state but is not part of an assignment statement is asserted for the duration of that state only.

A state machine variable that is assigned a value in the body of a state retains that value until assigned a new value in another state.

A state machine function that is listed in the body of a state is executed once during the state.

4.6.3.2 State Machine Variables and Functions

A state machine variable is asserted when its value is 1 and deasserted when it value is 0.

The functions used in the state machines are defined as follows.

change( )

Asserted when the variable on which it operates changes state.

next_code_group( )

Gets the next 10 bit code-group for the lane when it becomes available.

next_column( )

Gets the next column of 4 code-groups when it becomes available.

The variables used in the state machines are defined as follows.

4x-mode

Asserted when the port is operating in 4x mode

||A||

Asserted when the current column contains all /A/s.

Acounter

A counter used in the Lane Alignment state machine to count received alignment columns (||A||s).

align_error

Asserted when the current column contains at least one /A/, but not all /A/s.

all_lane_sync

Asserted when all four lanes of a 4x mode receiver are in bit synchronization and code-group boundary alignment. (all_lane_sync = lane_sync[0] & lane_sync[1] & lane_sync[2] & lane_sync[3])

discovery_timer_done

Asserted when discovery_timer_en has been continuously asserted for 12 +/- 4 msec.

discovery_timer_en

When asserted, the discovery_timer runs. When deasserted, the discovery_timer is reset to and maintains its initial value.

force_1x_mode

When asserted, forces the 1x/4x Initialization state machine to use 1x mode.

force_lane2

When asserted in 1x mode, forces the 1x/4x Initialization state machine to use lane 2 for reception.

Icounter

Counter used in the Lane_Synchronization state machine to count INVALID received code-groups. There is one Icounter for each lane in a 4x mode receiver.

/K28.5/

Asserted when the current code-group is /K28.5/

Kcounter

Counter used in the Lane_Synchronization state machine to count received /K.28.5/ code-groups. There is one Kcounter for each lane in a 4x mode receiver.

lane02_drvr_oe

The output enable for the lane 0 and the lane 2 output drivers of a 1x/4x mode port.

lane13_drvr_oe

The output enable for the lane 1 and the lane 3output drivers of a 1x/4x mode port.

lanes_aligned

Asserted by the Lane_Alignment state machine when it determines that all four lanes are in sync and aligned.

lane_sync

Asserted by the Lane_Synchronization state machine when it determines that the lane it is monitoring is in bit synchronization and code-group boundary alignment. The is a lane_sync signal for each lane in a 4x mode receiver.

link_drvr_oe

When asserted, the output link driver of a 1x port is enabled.

Mcounter

Mcounter is used in the Lane_Alignment state machine to count columns received that contain at least one /A/, but not all /A/s.

Vcounter

Vcounter is used in the Lane_Synchronization state machine to count VALID received code-groups. There is one Vcounter for each lane in a 4x mode receiver.

port_initialized

When asserted, port_initialized indicates that the link is initialized and is available for transmitting control symbols and packets. When deasserted, the link is not initialized and is not available for transmitting control symbols and packets.

receive_lane2

In a 1x/4x capable port that is initialized and is operating in 1x mode (4x_mode deasserted), receive_lane2 indicates which lane the port has selected for input. When asserted, the port input is from lane 2. When deasserted the port input is from lane 0. When the port is operating in 4x mode (4x_mode asserted), receive_lane2 is undefined and shall be ignored.

signal_detect

Asserted when a lane receiver is enabled and a signal meeting an implementation defined criteria is present at the input of the receiver. The use of signal_detect is implementation dependent. It may be continuously asserted or it may be used to require that some implementation defined additional condition be met before the Lane_Synchronization state machine is allowed to exit the NO_SYNC state. Signal_detect might for example be used to ensure that the input signal to a lane receiver meet some minimum AC input power requirement so that the receiver can not lock up on crosstalk from an output of the same port.

silence_timer_done

Asserted when silence_timer_en has been continuously asserted for 120 +/- 40 µs.

silence_timer_en

When asserted, the silence_timer runs. When deasserted, the silence_timer is reset to and maintains its initial value.

/VALID/

When asserted, /VALID/ indicates that the current code-group is a valid code-group given the current running disparity.

/INVALID/

When asserted, /INVALID/ indicates that the current code-group is an invalid code-group given the current running disparity.

4.6.3.3 Lane Synchronization State Machine

The Lane_Synchronization state machine monitors the bit synchronization and code-group boundary alignment for a lane receiver. A port that supports only 1x mode has one Lane_Synchronization state machine. A port that supports both 1x and 4x modes has four Lane_Synchronization state machines, one for each lane (Lane_Synchronization[0] through Lane_Synchronization[3]).

The state machine determines the bit synchronization and code-group boundary alignment state of a lane receiver by monitoring the received code-groups and looking for /K28.5/s, other valid code-groups and invalid code-groups. The code-group /K28.5/ contains the “comma” bit sequence that is used to establish code-group boundary alignment. When a lane is error free, the “comma” pattern occurs only in the /K28.5/ code-group. Several counters are used to provide hysteresis so that occasional bit errors do not cause spurious lane_sync state changes.

The state machine does not specify how bit synchronization and code-group boundary alignment is to be achieved. The methods used by a lane receiver to achieve bit synchronization and code-group boundary alignment are implementation dependent. However, isolated single bit errors shall not cause the code-group boundary alignment mechanism to change alignment. For example, a isolated single bit error that results in a "comma" pattern across a code-group boundary may not cause the code-group boundary alignment mechanism to change alignment.

The state machine starts in the NO_SYNC state and sets the variables Kcounter[0] and lane_sync[n] to 0 (lane n is out of code-group boundary sync). It then looks for a /K28.5/ code-group. When it finds one and the signal signal_detect[n] is asserted, the machine moves to the NO_SYNC_1 state.

The NO_SYNC_1 state in combination with the NO_SYNC_2 state looks for 127 /K28.5/ code-groups without any intervening /INVALID/ code-groups. When this condition achieves, machine goes to state SYNC. If an intervening /INVALID/ code-group is detected, the machine goes back to the NO_SYNC state. There is no magic is the number 127. It was selected to be large enough that it would be highly unlikely that SYNC would be falsely achieved and it fits in a 7-bit counter. Since the /K28.5/ code-group comprises slightly less than half of the code-groups in the idle sequence, something more than 256 code-groups must be received after the first /K28.5/ to achieve the 128 /K28.5/ code-group criteria to transition to the SYNC state.

In the SYNC state, the machine sets the variable lane_sync[n] to 1 (lane n is in code-group boundary sync), set the variable Icounter[n] to 0 and begins looking for /INVALID/ code-groups. If an /INVALID/ code-group is detected, the machine goes to state SYNC_1.

The SYNC_1 state in combination with the SYNC_2, SYNC_3, and SYNC_4 states look for 255 consecutive /VALID/ code-groups without any /INVALID/ code-groups. If this achieves, the machine returns to the SYNC state. If it does not, the machine goes to the NO_SYNC state and starts over. This algorithm tolerates isolated single bit errors in that an isoled single bit error will not cause the machine to change the variable lane_snc[n] FROM 1 TO 0 (in sync to out of sync). Two errors within 256 code-groups will result in an out-of-sync indication.

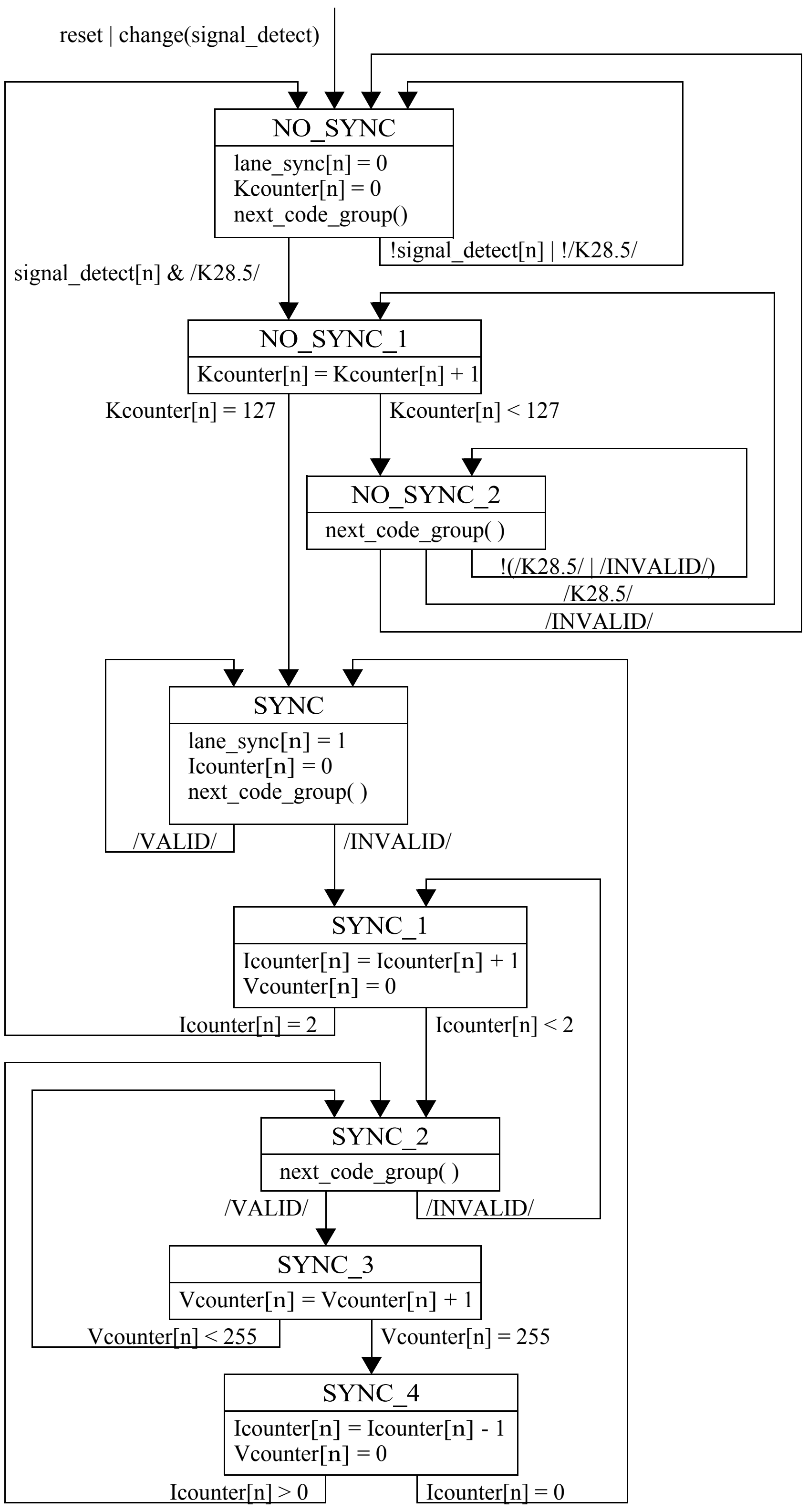


Figure 4-9. Lane_Synchronization State

4.6.3.4 Lane Alignment State Machine

The Lane_Alignment state machine monitors the alignment of the output of the four lane receivers in a port operating in 4x mode. A port supporting 4x mode has one Lane_alignment state machine, a port supporting only 1x mode does not have a Lane_Alignment state machine. (Lane alignment is required in a 4x port receiver to compensate for unequal propagation delays through the four lanes.)

The state machine determines the alignment state by monitoring the fours lanes for columns containing all /A/s (||A||), columns containing at least one but not all /A/s and columns containing no /A/s. Several counters are used to provide hysteresis so that occasional bit errors do not cause spurious lanes_aligned state changes.

The state machine does not specify how lane alignment is to be achieved. The methods used by a 4x port receiver to achieve lane alignment are implementation dependent. However, isolated single bit errors shall not cause the lane alignment mechanism to change lane alignment. For example, a isolated single bit error that results in a column that contains at least one /A/ but not all /A/s may not cause the lane alignment mechanism to change the lane alignment.

The state machine starts in the NOT_ALIGNED state where the variables Acounter and lanes_aligned are set to 0 (all lanes are not aligned). The machine then waits for all (four) lanes to achieve code-group boundary alignment (all_lanes_sync asserted) and the reception of an ||A|| (a column of all /A/s). When this obtains, the machine goes to NOT_ALIGNED_1 state.

The NOT_ALIGNED_1 state in combination with the NOT_ALIGNED_2 state looks for the reception of four ||A||s without the intervening reception of a misaligned column (a column with at least one /A/ but not all /A/s which causes the signal align_error to be asserted). When this obtains, the machine goes to the ALIGNED state. If an intervening misaligned column is received, the machine goes back to the NOT_ALIGNED state.

In the ALIGNED state, the machine sets the variable lanes_aligned to 1 (all lanes are aligned) and the variable Mcounter to 0 and looks for a misaligned column (align_error asserted). If a misaligned column is detected, the machine goes to the ALIGNED_1 state.

The ALIGNED_1 stte in combination with the ALIGNED_2 and ALIGNED_3 states look for the reception of four ||A||s without the intervening reception of misaligned column. If this condition obtains, the machine returns to the ALIGNED state. If an intervening misaligned column occurs, the machine goes to the NOT_ALIGNED state and starts over. This algorithm tolerates isolated single bit errors in that an isolated single bit error will not cause the machine to change the variable lanes_aligned from 1 to 0 (in lane alignment to out of lane alignment). The Lane_Alignment state machine is specified in Figure 4-10.

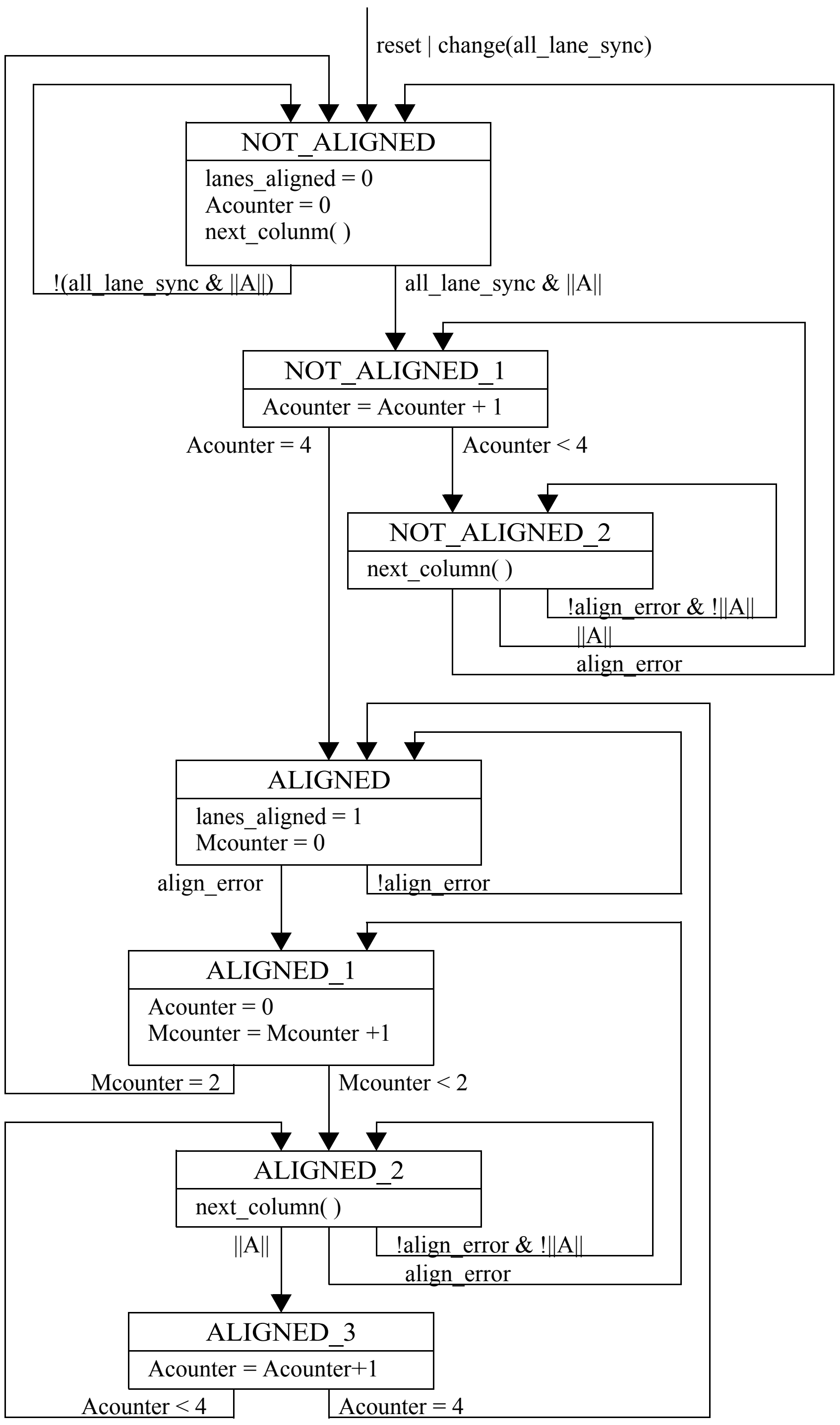


Figure 4-10. Lane_Alignment State Machine

4.6.3.5 1x Mode Initialization State Machine

The 1x_Initialization state machine shall be used by ports that support only 1x mode (1x ports).

The machine starts in the SILENT state. The link output driver is disabled to force the link partner to initialize regardless of its current state. The duration of the SILENT state is controlled by the silence_timer. The duration must be long enough to ensure that the link partner detects the silence (as a loss of lane_sync) and is forced to initialize but short enough that it is readily distinguished from a link break. When the silent interval is complete, the SEEK state is entered.

In the SEEK state, the link output driver is enabled, the idle sequence is transmitted, and the port waits for lane_sync to be asserted indicating the presence of a link partner. While lane_sync as defined indicates the bit and code-group boundary alignment synchronization state of the link receiver, it may also be thought of as indicating the presence of a link partner. When lane-sync is asserted, the 1X_MODE state is entered.

The input signal force_reinit allows the port to force link initialization at any time.

The variable port_initialized is asserted only in the 1X_MODE state. When port_initialized is deasserted, the port shall transmit a continuous idle sequence uninterrupted by control symbols or packets. When port_initialized is asserted, the port shall transmit control symbols and packets when they are offered for transmission. To maintain receiver bit synchronization and code-group alignment, the port shall transmit the idle sequence when no control symbol or packet is being transmitted.

The 1x_Initialization state machine is specified in Figure .

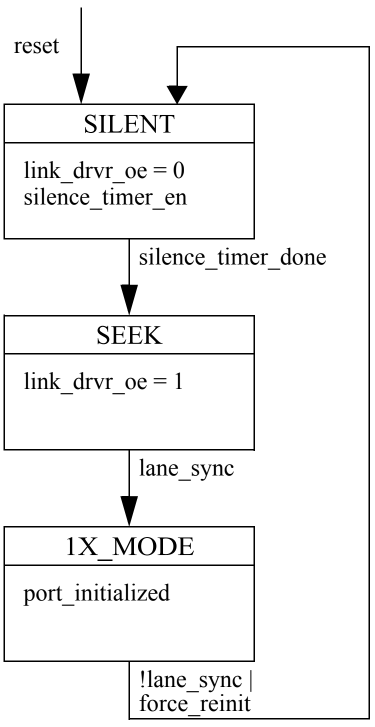


Figure 4-11. 1x_Initialization State Machine

4.6.3.6 1x/4x Mode Initialization State Machine

The 1x/4x_Initialization state machine shall be used by ports that support both 1x and 4x mode (1x/4x ports). In addition to determining when the link is initialized, the state machine controls whether the port receiver operates in 1x or 4x mode and in 1x mode whether lane 0 or lane 2 is selected as the inbound lane.

The machine starts in SILENT state. All four lane output drivers are disabled to force the link partner to initialize regardless of its current state. The duration of the SILENT state is controlled by the silence_timer. The duration must be long enough to ensure that the link partner detects the silence (as a loss of lane_sync) and is forced to initialize but short enough that it is readily distinguished from a link break. When the silent interval is complete, the SEEK state is entered.

In the SEEK state, a 1x/4x port transmits the idle sequence on lanes 0 and 2 (the lane 1 and lane 3 output drivers remain disabled to save power) and waits for an indication that a link partner is present. While lane_sync as defined indicates the bit and code-group boundary alignment synchronization state of a lane receiver, it is also used to indicate the presence of a link partner. A link partner is declared to be present when either lane_sync[0] or lane_sync[2] is asserted which causes the state machine to enter the DISCOVERY state.

In the DISCOVERY state, the port enables the output drivers for lanes 1 and 3 and transmits the idle sequence on all four lanes. The discovery_timer is also started. The discovery_timer allows time for the link partner to enter its DISCOVERY state and if the link partner is supporting 4x mode, for all four local lane receivers to acquire bit synchronization and code-group boundary alignment and for all four lanes to be aligned.

If four lane alignment is achieved (lane_aligned asserted) and the variable force_1x_mode is not asserted, the machine enters the 4X_MODE state. It remains in this state until lane alignment or at least one lane_sync is lost (lane_aligned deasserted) or reinitialization is forced (force_reinit is asserted).

When the time allowed for discovery is over (discovery_timer_done is asserted), if four- lane alignment has not been achieved (lane_aligned not asserted) or if the variable force_1x_mode is asserted, the machine enters one of the 1x mode states. If code-group alignment has been achieved on lane 0 (lane0_sync asserted) and the variable force_lane2 is not asserted, the machine enters 1X_MODE_LANE0 and remains in that state until code-group boundary alignment is lost (lane0_sync deasserted) or reinitialization is forced (force_reinit is asserted). If the variable force_lane2 is asserted or if code-group alignment has not been achieved on lane 0 (lane0_sync not asserted) but has been achieved on lane 2 (lane2_sync asserted), the machine enters 1X_MODE_LANE2 and remains in that state until code-group boundary alignment is lost (lane2_sync deasserted) or reinitialization is forced (force_reinit is asserted).

The input signals force_1x_mode and force_lane2 allow the state of the machine to be forced during initialization into 1x mode, and in 1x mode to be forced to receive on lane 2. The input signal force_reinit allows the port to force link initialization at any time.

The variable port_initialized is asserted only in the 1X_MODE_LANE0, 1X_MODE_LANE2 and 4X_MODE states. When port_initialized is deasserted, the port shall transmit a continuous idle sequence uninterrupted by control symbols or packets. When port_initialized is asserted, the port shall transmit control symbols and packets when they are offered for transmission. To maintain receiver bit synchronization and code-group alignment, the port shall transmit the idle sequence when no control symbol or packet is being transmitted. The 1x/4x_Initialization state machine is specified in Figure 4-12.

.

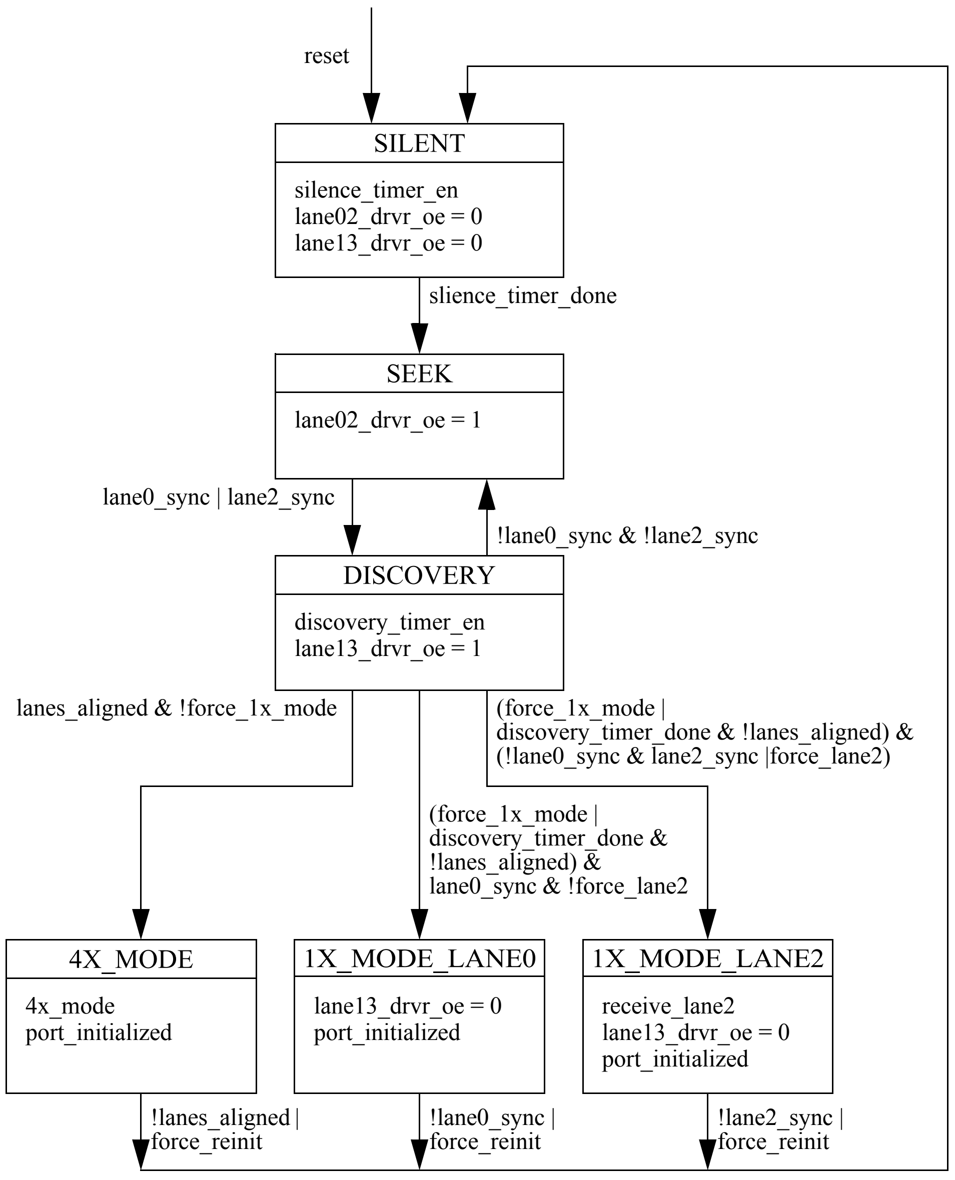

reset
SILENT
silence_timer_en
lane02_drvr_oe = 0
lane13_drvr_oe = 0
slience_timer_done
SEEK
lane02_drvr_oe = 1
lane0_sync | lane2_sync
!lane0_sync & !lane2_sync
DISCOVERY
discovery_timer_en
lane13_drvr_oe = 1
lanes_aligned & !force_1x_mode
(force_1x_mode | discovery_timer_done & !lanes_aligned) & (!lane0_sync & lane2_sync |force_lane2)
(force_1x_mode | discovery_timer_done & !lanes_aligned) & lane0_sync & !force_lane2
4X_MODE
4x_mode
port_initialized
1X_MODE_LANE0
lane13_drvr_oe = 0
port_initialized
1X_MODE_LANE2
receive_lane2
lane13_drvr_oe = 0
port_initialized
!lanes_aligned | force_reinit
!lane0_sync | force_reinit
!lane2_sync | force_reinit

Figure 4-12. 1x/4x_Initialization State Machine

5 Chapter 5 - LP-Serial Protocol

This chapter specifies the LP-serial protocol. The protocol provides the reliable delivery of packets between two RapidIO devices that are connected by an LP-Serial link.

Packet priority, the mapping of transaction request flows onto packet priority, buffer management, and the use of control symbols in managing the delivery of packets between two devices is explained in this chapter.

5.1 Packet Exchange Protocol

This physical layer LP-Serial specification defines a protocol for devices connected by a LP-Serial link in which each packet transmitted by one device is acknowledged by control symbols transmitted by the other device. If a packet cannot be accepted for any reason, an acknowledgment control symbol indicates the reason and that the original packet and any transmitted subsequent packets must be resent. This behavior provides a flow control and error control mechanism between connected processing elements.

Figure shows an example of transporting a request and response packet pair across an interconnect fabric with acknowledgments between the link transmitter/receiver pairs along the way. This allows flow control and error handling to be managed between each electrically connected device pair rather than between the original source and final target of the packet. A device shall transmit an acknowledgment control symbol for a request packet before transmitting the response packet corresponding to that request.

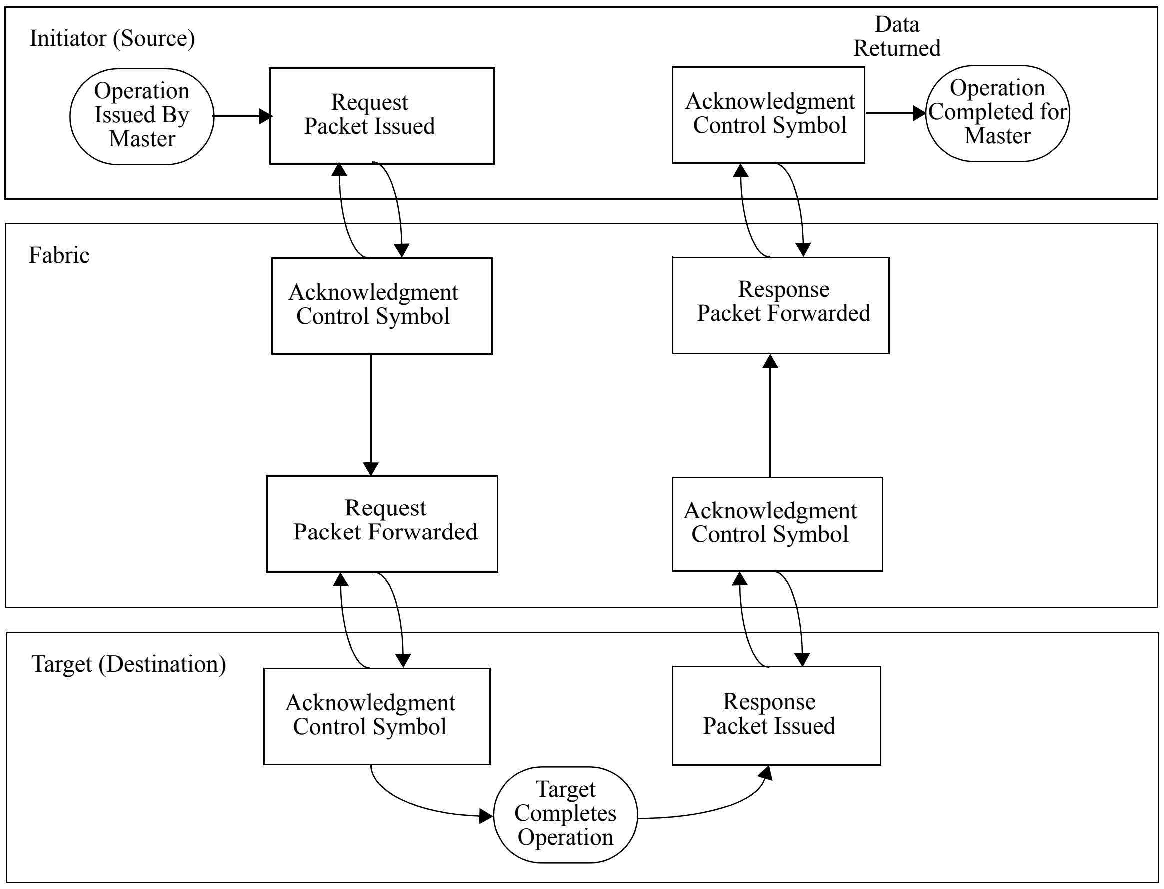

Initiator (Source)
Data Returned
Operation Issued By Master
Request Packet Issued
Acknowledgment Control Symbol
Operation Completed for Master
Fabric
Acknowledgment Control Symbol
Response Packet Forwarded
Request Packet Forwarded
Acknowledgment Control Symbol
Target (Destination)
Acknowledgment Control Symbol
Response Packet Issued
Target Completes Operation

Figure 5-1. Example Transaction with Acknowledgment

5.2 Control Symbols

Control Symbols are the message elements used by ports connected by an LP-Serial link to manage all aspects of LP-Serial link operation. They are used for link maintenance, packet delimiting, packet acknowledgment, error reporting, and error recovery.

5.2.1 Control Symbol Delimiting

LP-Serial control symbols are delimited for transmission by a single 8B/10B special (control) character. The control character marks the beginning of the control symbol and immediately precedes the first bit of the control symbol. The control symbol delimiting special character is added to the control symbol before the control symbol is passed to the PCS sublayer for lane striping (if applicable) and 8B/10B encoding. Since control symbol length is constant and known, control symbols do not need end delimiters. The combined delimiter and control symbol is referred to as a delimited control symbol.

One of two special characters is used to delimit a control symbol. If the control symbol contains a packet delimiter, the special character PD (K28.3) is used. If the control symbol does not contain a packet delimiter, the special character SC (K28.0) is used. This use of special characters provides the receiver with an “early warning” of the content of the control symbol.

5.2.2 Control Symbol Transmission

After an LP-Serial link is initialized, each port connected by the link shall transmit status control symbols at least once every 1024 code-groups whenever the port has nothing else (no control symbols and no packets) to transmit. The time required to transmit 1024 code-groups shall be computed based on the aggregate unidirectional baud rate of the link and is independent of the number of lanes. For example, a 4x link with each lane operating at 3.125 GB has an aggregate unidirectional baud rate of (4*3.125) = 12.5 GB. The time required to transmit 1024 code-groups is (1024*10/12.5*10^9) = 819.2 ns. Hence on such a link, a status control symbol must be transmitted by each port at least one every 819.2 ns if the port has nothing else to transmit.

After an LP-Serial link is initialized, each port connected by the link shall not begin packet transmission until the port has received 7 error-free status control symbols with no intervening detected errors. This provides a degree of link verification before packet transmission begins.

After an LP-Serial port begins packet transmission, it shall transmit a control symbol containing the buf_status field once every 1024 code-groups.

5.2.3 Embedded Control Symbols

Any control symbol that does not contain a packet delimiter may be embedded in a packet. An embedded control symbol may contain any defined encoding of stype0 and an stype1 encoding of “multicast-event” or “NOP”. Control symbols with stype1 encodings of start-of-packet, end-of-packet, stomp, restart-from-retry, or link-request/input-status cannot be embedded as they would terminate the packet.

The manner and degree to which control symbol embedding is used on a link impacts both link and system performance. For example, embedding multicast-event control symbols allows their propagation delay and delay variation through switch processing elements to be minimized and is highly desirable for some multicast-event applications. On the other hand, embedding all packet acknowledgment control symbols rather than combining as many of them as possible with packet delimiter control symbols reduces the link bandwidth available for packet transmission and may be undesirable.

5.2.4 Multicast-Event Control Symbols

The Multicast-Event control symbol provides a mechanism through which notice that some system defined event has occurred, can be selectively multicast throughout the system. Refer to Section 3.4.6: "Multicast-Event Control Symbol" for the format of the multicast-event control symbol.

When a switch processing element receives a Multicast-Event control symbol, the switch shall forward the Multicast-Event by issuing a Multicast-Event control symbol from each port that is designated in the port's CSR as a Multicast-Event output port. A switch port shall never forward a Multicast-Event control symbol back to the device from which it received a Multicast-Event control symbol regardless of whether the port is designated a Multicast-Event output or not.

It is intended that at any given time, Multicast-Event control symbols will be sourced by a single device. However, the source device can change (in case of failover, for example). In the event that two or more Multicast-Event control symbols are received by a switch processing element close enough in time that more than one is present in the switch at the same time, at least one of the Multicast-Event control symbols shall be forwarded. The others may be forwarded or discarded (device dependent).

The system defined event whose occurrence Multicast-Event gives notice of has no required temporal characteristics. It may occur randomly, periodically, or anything in between. For instance, Multicast-Event may be used for a heartbeat function or for a clock synchronization function in a multiprocessor system.

In an application such as clock synchronization in a multiprocessor system, both the propagation time of the notification through the system and the variation in propagation time from Multicast-Event to Multicast-Event are of concern. For these reasons and the need to multicast, control symbols are used to convey Multicast-Events as control symbols have the highest priority for transmission on a link and can be embedded in packets.

While this specification places no limits on Multicast-Event forwarding delay or forwarding delay variation, switch functions should be designed to minimize these characteristics. In addition, switch functions shall include in their specifications the maximum value of Multicast-Event forwarding delay (the maximum value of Multicast-Event forwarding delay through the switch) and the maximum value of Multicast-Event forwarding delay variation (the maximum value of Multicast-Event forwarding delay through the switch minus the minimum value of Multicast-Event forwarding delay through the switch).

5.3 Packets

5.3.1 Packet Delimiting

LP-Serial packets are delimited for transmission by control symbols. Since packet length is variable, both start-of-packet and end-of-packet delimiters are required. The control symbol marking the end of a packet (packet termination) follows the end of the packet or the end of an embedded control symbol.

The following control symbols are used to delimit packets.

- Start-of-packet
- End-of-packet
- Stomp
- Restart-from-retry
- Link-request/input-status

5.3.1.1 Packet Start

The beginning of a packet (packet start) shall be marked by a start-of-packet control symbol.

5.3.1.2 Packet Termination

A packet shall be terminated in one of the following three ways:

- The end of a packet is marked with an end-of-packet control symbol.
- The end of a packet is marked with a start-of-packet control symbol that also marks the beginning of a new packet.
- The packet is canceled by a restart-from-retry, link-request/input-status, or stomp control symbol

5.3.2 Acknowledgment Identifier

Each packet requires an identifier to uniquely identify its acknowledgment control symbol. This identifier, the acknowledge ID (ackID), is five bits long, allowing for a range of one to thirty two outstanding unacknowledged request or response packets between adjacent processing elements, however only up to thirty one outstanding unacknowledged packets are allowed at any one time. The first value of ackID assigned after a reset shall be 0b00000. Subsequent values of ackID shall be assigned sequentially (in increasing numerical order, wrapping back to 0 on overflow) to indicate the order of the packet transmission. The acknowledgments themselves are a number of control symbols defined in Chapter 3, “Control Symbols.

5.3.3 Packet Priority and Transaction Request Flows

Each packet has a priority that is assigned by the end point processing element that is the source of (initiates) the packet. The priority is carried in the prio field of the packet and has four possible values: 0, 1, 2, or 3. Packet priority increases with the priority value with 0 being the lowest priority and 3 being the highest. Packet priority is used in Rapid IO for several purposes which include transaction ordering and deadlock prevention.

When a transaction is encapsulated in a packet for transmission, the transaction request flow indicator (flowID) of the transaction is mapped into the prio field of the packet. Transaction request flows A and B are mapped to priorities 0 and 1 respectively and transaction request flows C and above are mapped to priority 2 as specified in Table .

The mapping of transaction request flow onto packet priority (prio) allows a Rapid IO transport fabric to maintain transaction request flow ordering without the fabric having any knowledge of transaction types or their interdependencies. This allows a Rapid IO fabric to be forward compatible as the types and functions of transactions evolve. A fabric can maintain transaction request flow ordering by simply maintaining the order of packets with the same priority for each path through the fabric and can maintaining transaction request flow priority by never allowing a lower priority packet to pass a higher priority packet taking the same path through the fabric.

Table 5-1. Transaction Request Flow to Priority Mapping

Flow	System Priority	Request Packet Priority	Response Packet Priority
C or higher	Highest	2	3
B	Next	1	2 or 3
A	Lowest	0	1, 2, or 3

5.4 Link Maintenance Protocol

The link maintenance protocol involves a request and response pair between ports connected by an LP-Serial link. For software management, the request is generated through ports in the configuration space of the sending device. An external host write of a command to the link-request register with a Partition I: Input/Output Logical Specification maintenance write transaction causes an link-request control symbol to be issued onto the output port of the device, but only one link-request can be outstanding on a link at a time.

The device that is linked to the sending device shall respond with an link-response control symbol if the link-request command required it to do so. The external host retrieves the link-response by polling the link-response register with I/O logical maintenance read transactions. A device with multiple RapidIO interfaces has a link-request and a link-response register pair for each corresponding RapidIO interface.

The automatic error recovery mechanism relies on the hardware generating link-request/input-status control symbols under the transmission error conditions described in Section , “5.10.2.1 Recoverable Errors and using the corresponding link-response information to attempt recovery.

Due to the undefined reliability of system designs, it is necessary to put a safety lockout on the reset function of the link-request/reset-device control symbol. A device receiving a link-request/reset-device control symbol shall not perform the reset function unless it has received four link-request/reset-device control symbols in a row without any intervening packets or other control symbols, except status control symbols. This will prevent spurious reset-device commands inadvertently resetting a device. The link-request/reset-device control symbol does not require a response.

The input-status command of the link-request/input-status control symbol is used by the hardware to recover from transmission errors. If the input port had stopped due to a transmission error that generated a packet-not-accepted control symbol back to the sender, the link-request/input-status control symbol acts as a link-request/restart-from-error control symbol, and the receiver is re-enabled to receive new packets after generating the link-response control symbol. The link-request/input-status control symbol may also be used to restart the receiving device if it is waiting for a restart-from-retry control symbol after retrying a packet. This situation can occur if transmission errors are encountered while trying to resynchronize the sending and receiving devices after the retry.

The link-request/input-status control symbol requires a response. A port receiving a link-request/input-status control symbol returns a link-response control symbol containing two pieces of information:

- port_status
- ackID_status

These status indicators are described in Table 3-5.

The retry-stopped state indicates that the port has retried a packet and is waiting to be restarted. This state is cleared when a restart-from-retry (or a link-request/input-status) control symbol is received. The error-stopped state indicates that the port has encountered a transmission error and is waiting to be restarted. This state is cleared when a link-request/input-status control symbol is received.

5.5 Packet Transmission Protocol

The LP-Serial protocol for packet transmission provides link level flow and error detection and recovery.

The LP-Serial link protocol uses control symbols to delimit packets when they are transmitted across an LP-Serial link as specified in Section , “5.3.1 Packet Delimiting.

The LP-Serial link protocol uses acknowledgment to monitor packet transmission. Each packet transmitted across an LP-Serial link packet shall be acknowledged by the receiving port with a packet acknowledgment control symbol. To associate packet acknowledgment control symbols with transmitted packets, each packet shall be assigned an ackID value that is carried in the ackID field of the packet and the packet_ackID field of the associated acknowledgment control symbol. AckID values are assigned to packets sequentially in increasing numerical order wrapping to 0 on overflow. The ackID value carried by packets indicates their order of transmission.

The LP-Serial link protocol uses retransmission to recover from packet transmission errors. To enable packet retransmission, a copy of each packet transmitted across an LP-Serial link shall kept by the sending port until either a packet-accepted packet acknowledgment control symbol is received for the packet from the receiving port indicating that the port has received the packet without detected error and has accepted responsibility for the packet or the port determines that the packet has an encountered an unrecoverable error condition.

The LP-Serial link protocol uses the ackID value carried in each packet to ensure that no packets are lost due to transmission errors. A port shall accept packets from an LP-Serial link only in sequential ackID order, i.e. if the ackID value of the last packet accepted was N, the ackID value of the next packet that is accepted must be (N+1) modulo32.

An LP-Serial port accepts or rejects each error-free packet it receives depending on whether the port has input buffer space available at the priority level of the packet. The use of the packet-accepted, packet-retry, and restart-from-retry control symbols and the buf_status field in packet acknowledgment control symbols to control the flow of packets across an LP-Serial link is cover in Section , “5.6 Flow Control.

The LP-Serial link protocol allows a packet that is being transmitted to be canceled at any point during its transmission. Packet cancellation is covered in Section , “If the port and its link partner both support transmitter-controlled flow control, then both ports shall use transmitter-controlled flow control. Otherwise, both ports shall use receiver-controlled flow control.Canceling Packets.

The LP-Serial link protocol provides detection and recovery processes for both transmission errors and protocol violations. The enumeration of detectable errors, the detection of errors and the associated error recovery processes are covered in Section , “As an example, suppose an end point processing element has a blocked input port because all available resources are being used for a response packet that the processing element is trying to send. If the response packet is retried by the downstream processing element, raising the priority of the response packet until it is accepted allows the processing element’s input port to unblock so the system can make forward progress..

In order to prevent internal switch processing element internal errors, such as SRAM soft bit errors, from silently corrupting a packet and the system, switch processing elements shall maintain packet error detection coverage while a packet is passing though the switch. The simplest method for maintain packet error detection coverage is pass the packet CRC through the switch as part of the packet. This works well for all non-maintenance packets whose CRC does not change as the packets are transported from source to destination through the fabric. Maintaining error detection coverage is more complicated for maintenance packets as their hop_count and CRC change every time they pass through a switch.

In order to support transaction ordering requirements of the I/O Logical Layer specification, the LP-Serial protocol imposes packet delivery ordering requirements within the physical layer and transaction delivery ordering requirements between the physical layer and the transport layer in end point processing elements. These requirements are covered in Section , “5.8 Transaction and Packet Delivery Ordering Rules.

In order to prevent deadlock, the LP-Serial protocol imposes a set of deadlock prevention rules. These rules are covered in Section , “5.9 Deadlock Avoidance.

The LP-Serial specification does not require the use of fair bandwidth allocation mechanisms within the transport fabric, therefore, it is possible that traffic associated with higher flow levels can starve traffic associated with lower flow levels. Any sort of starvation prevention, flow level bandwidth allocation, or fairness mechanisms are device and system dependent and are beyond the scope of this specification.

5.6 Flow Control

This section defines RapidIO LP-Serial link level flow control. The flow control operates between each pair of ports connected by an LP-Serial link. The purpose of link level flow control is to prevent the loss of packets due to a lack of buffer space in a link receiver.

The LP-Serial protocol defines two methods or modes of flow control. These are named receiver-controlled flow control and transmitter-controlled flow control. Every RapidIO LP-Serial port shall support receiver-controlled flow control. LP-Serial ports may optionally support transmitter-controlled flow control.

5.6.1 Receiver-Controlled Flow Control

Receiver-controlled flow control is the simplest and basic method of flow control. In this method, the input side of a port controls the flow of packets from its link partner by accepting or rejecting (retrying) packets on a packet by packet basis. The receiving port provides no information to its link partner about the amount of buffer space it has available for packet reception.

As a result, its link partner transmits packets with no *a priori* expectation as to whether a given packet will be accepted or rejected. A port signals its link partner that it is operating in receiver-controlled flow control mode by setting the buf_status field to all 1’s in every control symbol containing the field that the port transmits. This method is named receiver-controlled flow control because the receiver makes all of the decisions about how buffers in the receiver are allocated for packet reception.

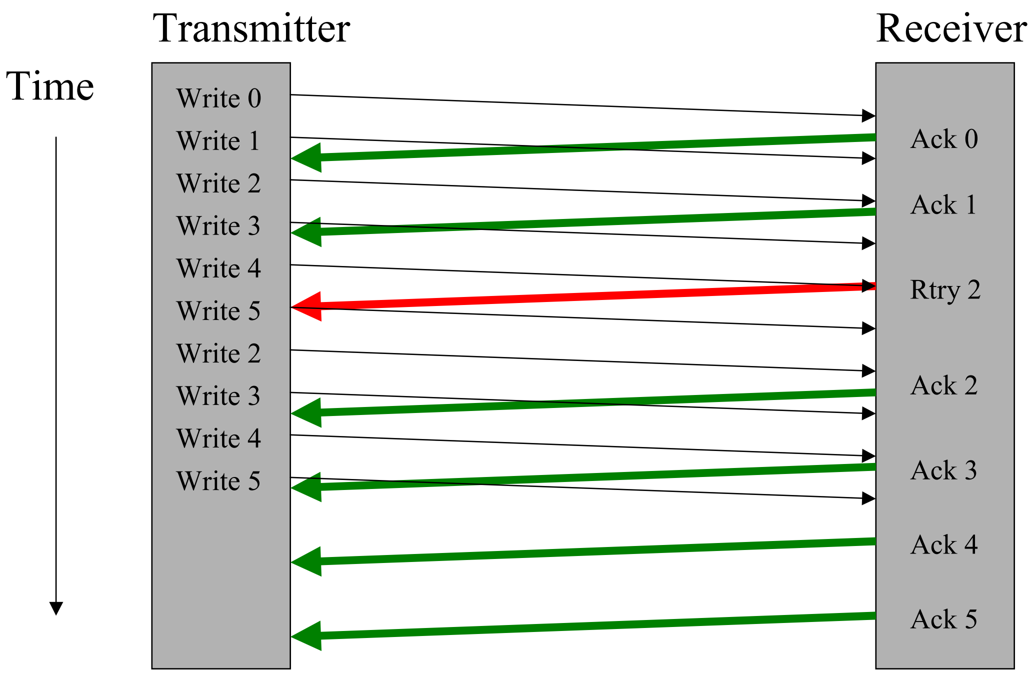
A port operating in receiver-controlled flow control mode accepts or rejects each inbound error-free packet based on whether the receiving port has enough buffer space available at the priority level of the packet. If there is enough buffer space available, the port accepts the packet and transmits a packet-accepted control symbol to its link partner that contains the ackID of the accepted packet in its packet_ackID field. This informs the port’s link partner that the packet has been received without detected errors and that it has been accepted by the port. On receiving the packet-accepted control symbol, the link partner discards its copy of the accepted packet freeing buffer space in the partner.

If buffer space is not available, the port rejects the packet. When a port rejects (retries) an error-free packet, it immediately enters the Input Retry-stopped state and follows the Input Retry-stopped recovery process specified in Section , “5.6.2.1 Input Retry-Stopped Recovery Process. As part of the Input Retry-stopped recovery process, the port sends a packet-retry control symbol to its link partner indicating that the packet whose ackID is in the packet_ackID field of the control symbol and all packets subsequently transmitted by the port have been discarded by the link partner and must all be retransmitted. The control symbol also indicates that the link partner is temporarily out of buffers for packets of priority less than or equal to the priority of the retried packet.

A port that receives a packet-retry control symbol immediately enters the Output Retry-stopped state and follows the Output Retry-stopped recovery process specified in Section , “5.6.2.2 Output Retry-Stopped Recovery Process. As part of the Output Retry-stopped recovery process, the port receiving the packet-retry control symbol sends a restart-from-retry control symbol which causes its link partner to exit the Input Retry-stopped state and resume packet reception. The ackID assigned to that first packet transmitted after the restart-from-retry control symbol is the ackID of the packet that was retried.

Figure shows an example of receiver-controlled flow control operation. In this example the transmitter is capable of sending packets faster than the receiver is able to absorb them. Once the transmitter has received a retry for a packet, the transmitter may elect to cancel any packet that is presently being transmitted since it will be discarded anyway. This makes bandwidth available for any higher priority packets that may be pending transmission.

Figure 5-2. Receiver-Controlled Flow Control



5.6.2 Transmitter-Controlled Flow Control

In transmitter-controlled flow control, the receiving port provides information to its link partner about the amount of buffer space it has available for packet reception. With this information, the sending port can allocate the use of the receiving port's receive buffers according to the number and priority of packets that the sending port has waiting for transmission without concern that one or more of the packets shall be forced to retry.

A port signals its link partner that it is operating in transmitter-controlled flow control mode by setting the buf_status field to a value different from all 1's in every control symbol containing the field that the port transmits. This method is named transmitter-controlled flow control because the transmitter makes almost all of the decisions about how the buffers in the receiver are allocated for packet reception.

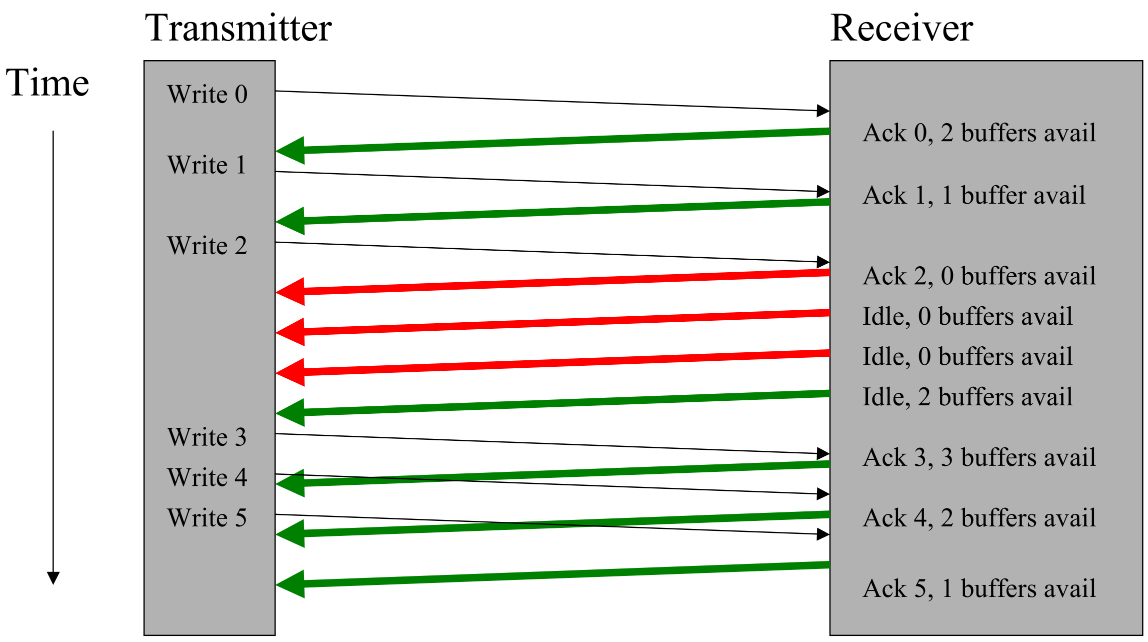
The number of free buffers that a port has available for packet reception is conveyed to its link partner by the value of the buf_status field in the control symbols that the port transmits. The value conveyed by the buf_status field is the number of maximum length packet buffers currently available for packet reception up to the limit that can reported in the field. If a port has more buffers available than the maximum value that can be reported in the buf_status field, the port sets the field to that maximum value. A port may report a smaller number of buffers than it actually has available, but it shall not report a greater number.

A port informs its link partner when the number of free buffers available for packet reception changes. The new value of buf_status is conveyed in the buf_status field of a packet-accepted, packet-retry, or status control symbol. Each change in the number of free buffers a port has available for packet reception need not be conveyed to the link partner. However, a port shall send a control symbol containing the buf_status field to its link partner no less often than the minimum rate specified in Section , "5.2.2 Control Symbol Transmission.

A port whose link partner is operating in transmitter-control flow control mode should never receive a packet-retry control symbol from its link partner unless the port has transmitted more packets than its link partner has receive buffers, violated the rules that all input buffer may not be filled with low priority packets or there is some fault condition. If a port whose link partner is operating in transmitter-control flow control mode receives a packet-retry control symbol, the output side of the port immediately enters the Output Retry-stopped state and follows the Output Retry-stopped recovery process specified in Section , “5.6.2.2 Output Retry-Stopped Recovery Process.

A simple example of transmitter-controlled flow control is shown in Figure .

Figure 5-3. Transmitter-Controlled Flow Control



5.6.2.1 Input Retry-Stopped Recovery Process

When the input side of a port retries a packet, it immediately enters the Input Retry-stopped state. To recover from this state, the input side of the port takes the following actions.

- Discards the rejected packet and ignores all subsequently received packets while the port is in the Input Retry-stopped state.

Causes the output side of the port to issue a packet-retry control symbol containing the ackID value of the retried packet in the packet_ackID field of the control symbol. (The packet-retry control symbol causes the output side of the link partner to enter the Output Retry-stopped state and send a restart-from-retry control symbol.)

- When a restart-from-retry control symbol is received, exit the Input Retry-stopped state and resume packet reception.

An example state machine with the behavior described in this section is included in Section A.1, “Packet Retry Mechanism".”

5.6.2.2 Output Retry-Stopped Recovery Process

To recover from the Output Retry-stopped state, the output side of a port takes the following actions.

- Immediately stops transmitting new packets. Resets the link packet acknowledgment timers for all transmitted but unacknowledged packets. (This prevents the generation of spurious time-out errors.)
- Transmits a restart-from-retry control symbol.
- Backs up to the first unaccepted packet (the retried packet) which is the packet whose ackID value is specified by the packet_ackID value contained in the packet-retry control symbol. (The packet_ackID value is also the value of ackID field the port retrying the packet expects in the first packet it receives after receiving the restart-from-retry control symbol.)
- Exits the Output Retry-stopped state and resumes transmission with either the retried packet or a higher priority packet which is assigned the ackID value contained in the packet_ackID field of the packet-retry control symbol.

An example state machine with the behavior described in this section is included in Section A.1, “Packet Retry Mechanism".”

5.6.2.3 Receive Buffer Management

In transmitter-controlled flow control, the transmitter manages the packet receive buffers in the receiver. This may be done in a number of ways, but the selected method shall not violate the rules in Section , “5.9 Deadlock Avoidance concerning the acceptance of packets by ports

One possible implementation to organize the buffers is establish watermarks and use them to progressively limit the packet priorities that can be transmitted as the effective number of free buffers in the receiver decreases. For example, RapidIO LP-Serial has four priority levels. Three non-zero watermarks are needed to progressively limit the packet priorities that may be transmitted as the effective number of free buffers decreases. Designate the three watermarks as WM0, WM1, and WM2 where WM0 > WM1 > WM2 > 0 and employ the following rules.

If free_buffer_count >= WM0, all priority packets may be transmitted.

If WM0 > free_buffer_count >= WM1, only priority 1, 2, and 3 packets may be transmitted.

If WM1 > free_buffer_count >= WM2, only priority 2 and 3 packets may be transmitted.

If WM2 > free_buffer_count, only priority 3 packets may be transmitted.

If this method is implemented, the initial values of the watermarks may be set by the hardware at reset as follows.

WM0 = 4

WM1 = 3

WM2 = 2

These initial values may be modified by hardware or software. The modified watermark values shall be based on the number of free buffers reported in the buf_status field of status control symbols received by the port following link initialization and before the start of packet transmission.

The three watermark values and the number of free buffers reported in the buf_status field of status control symbols received by the port following link initialization and before the start of packet transmission may be stored in a CSR. Since the maximum value of each of these four items is 30, each will fit in an 8-bit field and all four will fit in a single 32-bit CSR. If the watermarks are software setable, the three watermark fields in the CSR should be writable. For the greatest flexibility, a watermark register should be provided for each port on a device.

5.6.2.4 Effective Number of Free Receive Buffers

The number of buffers available in a port’s link partner for packet reception is typically less than the value of the buf_status field most recently received from the link partner. The value in the buf_status field does not account for packets that have been transmitted by the port but not acknowledged by its link partner. The variable free_buffer_count is defined to be the effective number of free buffers available in the link partner for packet reception. The value of free_buffer_count shall be determined according to the following rules.

The port shall maintain a count of the packets that it has transmitted but that have not been acknowledged by its link partner. This count is named the outstanding_packet_count.

After link initialization and before the start of packet transmission,

```
If (received_buf_status < 31) {
        flow_control_mode = transmitter;
        free_buffer_count = received_buf_status;
        outstanding_packet_count = 0;
}
else
        flow_control_mode = receiver;
```

When a packet is transmitted by the port,

```
outstanding_packet_count =
                                outstanding_packet_count + 1;
```

When a status control symbol is received by the port,

```
free_buffer_count = received_buf_status -
```

```
                                        outstanding_packet_count;
```

When a packet-accepted control symbol is received by the port indicating that a packet has been accepted by the link partner,

```
Outstanding_packet_count =
                                    Outstanding_packet_count - 1;
free_buffer_count = received_buf_status -
                                        outstanding_packet_count;
```

When a packet-retry control symbol is received by the port indicating that a packet has been forced by the link partner to retry,

```
Outstanding_packet_count = 0;
free_buffer_count = received_buf_status;
```

When a packet-not-accepted control symbol is received by the port indicating that a packet has been rejected by the link partner because of one or more detected errors,

```
Outstanding_packet_count = 0;
free_buffer_count = 0;
```

The port then transmits a link-request/input-status (for input-status) control symbol and waits for the link partner to respond with a link-response control symbol. When the link-response control symbol is received,

```
free_buffer_count = received_buf_status;
```

5.6.2.5 Speculative Packet Transmission

A port whose link partner is operating in transmitter-controlled flow control mode may send more packets than the number of free buffers indicated by the link partner. Packets transmitted in excess of the free_buffer_count are transmitted on a speculative basis and are subject to retry by the link partner. The link partner accepts or rejects these packets on a packet by packet basis in exactly the same way it would if operating in receiver-controlled flow control mode. A port may use such speculative transmission in an attempt to maximize the utilization of the link. However, speculative transmission that results in a significant number of retries and discarded packets can reduce the effective bandwidth of the link.

5.6.3 Flow Control Mode Negotiation

Immediately following the initialization of a link, each port begins sending status control symbols to its link partner. The value of the buf_status field in these control symbols indicates to the link partner the flow control mode supported by the sending port.

The flow control mode negotiation rule is as follows:

> If the port and its link partner both support transmitter-controlled flow control, then both ports shall use transmitter-controlled flow control.
> Otherwise, both ports shall use receiver-controlled flow control.Canceling Packets

5.7 Cancelling Packets

When a port becomes aware of some condition that will require the packet it is currently transmitting to be retransmitted, the port may cancel the packet. This allows the port to avoid wasting bandwidth by not completing the transmission of a packet that the port knows must be retransmitted. Alternatively, the sending port may choose to complete transmission of the packet normally.

A port may cancel a packet if the port detects a problem with the packet as it is being transmitted or if the port receives a packet-retry or packet-not-accepted control symbol for a packet that is still being transmitted or that was previously transmitted. A packet-retry or packet-not-accepted control symbol can be transmitted by a port for a packet at any time after the port begins receiving the packet.

The sending device shall use the stomp control symbol, the restart-from-retry control symbol (in response to a packet-retry control symbol), or link-request/input-status control symbol (in response to a packet-not-accepted control symbol) to cancel a packet.

A port receiving a cancelled packet shall drop the packet. The cancellation of a packet shall not result in the generation of any errors. If the packet was cancelled because the sender received a packet-not-accepted control symbol, the error that caused the packet-not-accepted to be sent shall be reported in the normal manner.

If a port receiving a cancelled packet has not previously acknowledged the packet and is not in an “Input Stopped” State (Retry-Stopped or Error-Stopped), the port shall immediately enter the Input Retry-stopped state and follow the Input Retry-stopped recovery process specified in Section , “5.6.2.1 Input Retry-Stopped Recovery Process if the packet was canceled with a control symbol other than a restart-from-retry or a link-request/input-status control symbol. As part of the Input Retry-stopped recovery process, the port sends a packet-retry control symbol to the sending port indicating that the stomped packet was not accepted.

5.8 Transaction and Packet Delivery Ordering Rules

The rules specified in this section are required for the physical layer to support the transaction ordering rules specified in the logical layer specifications.

Transaction Delivery Ordering Rules:

1. The physical layer of an end point processing element port shall encapsulate in packets and forwarded to the RapidIO fabric transactions comprising a given transaction request flow in the same order that the transactions were received from the transport layer of the processing element.
2. The physical layer of an end point processing element port shall ensure that a higher priority request transaction that it receives from the transport layer of the processing element before a lower priority request transaction with the same sourceID and the same destinationID is forwarded to the fabric before the lower priority transaction.
3. The physical layer of an end point processing element port shall deliver transactions to the transport layer of the processing element in the same order that the packetized transactions were received by the port.

Packet Delivery Ordering Rules:

1. A packet initiated by a processing element shall not be considered committed to the RapidIO fabric and does not participate in the packet delivery ordering rules until the packet has been accepted by the device at the other end of the link. (RapidIO does not have the concept of delayed or deferred transactions. Once a packet is accepted into the fabric, it is committed.)
2. A switch shall not alter the priority of a packet.
3. Packet forwarding decisions made by a switch processing element shall provide a consistent output port selection which is based solely on the value of the destinationID field carried in the packet.
4. A switch processing element shall not change the order of packets comprising a transaction request flow (packets with the same sourceID, the same destinationID, the same priority and ftype != 8) as the packets pass through the switch.
5. A switch processing element shall not allow lower priority non-maintenance packets (ftype != 8) to pass higher priority non-maintenance packets with the same sourceID and destinationID as the packets pass through the switch.
6. A switch processing element shall not allow a priority N maintenance packet (ftype = 8) to pass another maintenance packet of priority N or greater that takes the same path through the switch (same switch input port and same switch output port).

5.9 Deadlock Avoidance

To allow a RapidIO protocol to evolve without changing the switching fabric, switch processing elements are not required, with the sole exception of ftype 8 maintenance transactions, to discern between packet types, their functions or their interdependencies. Switches, for instance, are not required to discern between packets carrying request transactions and packets carrying response transactions. As a result, it is possible for two end points, A and B to each fill all of their output buffers, the fabric connecting them and the other end point’s input buffers with read requests. This would result in an input to output dependency loop in each end point in which there would be no buffer space to hold the responses necessary to complete any of the outstanding read requests.

To break input to output dependencies, end point processing elements must have the ability to issue outbound response packets even if outbound request packets awaiting transmission are congestion blocked by the connected device. Two techniques are provided to break input to output dependencies. First, a response packet (a packet carry a response transaction) is always assigned an initial priority one priority level greater than the priority of the associated request packet (the packet carrying the associated request transaction).

This requirement is specified in Table . It breaks the dependency cycle at the request flow level. Second, the end point processing element that is the source of the response packet may additionally raise the priority of the response packet to a priority higher than the minimum required by Table if necessary for the packet to be accepted by the connected device. This additional increase in response packet priority above the minimum required by Table is called promotion. An end point processing element may promote a response packet only to the degree necessary for the packet to be accepted by the connected device.

The following rules define the deadlock prevention mechanism:

Deadlock Prevention Rules:

1. A RapidIO fabric shall be dependency cycle free for all operations that do not require a response. (This rule is necessary as there are no mechanisms provided in the fabric to break dependency cycles for operations not requiring responses.)
2. A packet carrying a request transaction that requires a response shall not be issued at the highest priority. (This rule ensures that an end point processing element can issue a response packet at a priority higher then the priority of the associated request. This rule in combination with rule 3 are basis for the priority assignments in Table .)
3. A packet carrying a response shall have a priority at least one priority level higher than the priority of the associated request. (This rule in combination with rule 2 are basis for the priority assignments in Table .)
4. A switch processing element port shall accept an error-free packet of priority N if there is no packet of priority greater than or equal to N that was previously received by the port and is still waiting in the switch to be forwarded. (This rule has multiple implications which include but are not limited to the following. First, a switch processing element port must have at least as many maximum length packet input buffers as there are priority levels. Second, a minimum of one maximum length packet input buffer must be reserved for each priority level. A input buffer reserved for priority N might be restricted to only priority N packets or might be allowed to hold packets of priority greater than or equal to N, either approach complies with the rule.)
5. A switch processing element port that transmits a priority N packet that is forced to retry by the connected device shall select a packet of priority greater than N, if one is available, for transmission. (This guarantees that packets of a given priority will not block higher priority packets.)
6. An end point processing element port shall accept an error-free packet of priority N if the port has enough space for the packet in the input buffer space of the port allocated for packets of priority N. (Lack of input buffer space is the only reason an end point may retry a packet.)
7. The decision of an end point processing element to accept or retry an error-free packet of priority N shall not dependent on the ability of the end point to issue request packets of priority less than or equal to N from any of its ports. (This rule works in conjunction with rule 6. It prohibits a device’s inability to issue packets of priority less than or equal to N, due to congestion in the connected device, from resulting in a lack of buffers to receive inbound packets of priority greater than or equal to N which in turn would result in packets of priority greater than or equal to N being forced to retry. The implications and some ways of complying with this rule are presented in the following paragraphs.)

One implication of Rule 7 is that a port may not fill all of its buffers that can be used to hold packets awaiting transmission with packets carrying request transactions. If this situation was allowed to occur and the output was blocked due to congestion in the connected device, read transactions could not be processed (no place to put the response packet), input buffer space would become filled and all subsequent inbound request packets would be forced to retry violating Rule 7.

Another implication is that a port must have a way of preventing output blockage at priority less than or equal to N, due to congestion in the connected device, from resulting in a lack of input buffer space for inbound packets of priority greater than or equal to N. There are multiple ways of doing this.

One way is to provide a port with input buffer space for at least four maximum length packets and reserve input buffer space for higher priority packets in a manner similar to that required by Rule 4 for switches. In this case, output port blockage at priority less than or equal to N will not result is blocking inbound packets of priority greater than or equal to N as any responses packets they generate will be of priority greater than N which is not congestion blocked. The port must however have the ability to select packets of priority greater than N for transmission from the packets awaiting transmission. This approach does not require the use of response packet priority promotion.

Alternatively, a port that does not have enough input buffer space for at least four maximum length packets or that does not reserve space for higher priority packets can use the promotion mechanism to increase the priority of response packets until they are accepted by the connected device. This allows output buffer space containing response packets to be freed even though all request packets awaiting transmission are congestion blocked.

As an example, suppose an end point processing element has a blocked input port because all available resources are being used for a response packet that the processing element is trying to send. If the response packet is retried by the downstream processing element, raising the priority of the response packet until it is accepted allows the processing element's input port to unblock so the system can make forward progress.

5.10 Error Detection and Recovery

Error detection and recovery is becoming a more important issue for many systems. The LP-Serial specification provides extensive error detection and recovery by combining retry protocols with cyclic redundancy codes, the selection of delimiter control characters and response timers.

One feature of the error protection strategy is that with the sole exception of maintenance packets, the CRC value carried in a packet remains unchanged as the packet moves through the fabric. The CRC carried in a maintenance packet must be regenerated at each switch as the hop count changes.

5.10.3 Lost Packet Detection

Some types of errors, such as a lost request or response packet or a lost acknowledgment, result in a system with hung resources. To detect this type of error there shall be time-out counters that expire when sufficient time has elapsed without receiving the expected response from the system. Because the expiration of one of these timers should indicate to the system that there is a problem, this time interval should be set long enough so that a false time-out is not signaled. The response to this error condition is implementation dependent.

The RapidIO specifications require time-out counters for the physical layer, the port link time-out counters, and counters for the logical layer, the port response time-out counters. The interpretation of the counter values is implementation dependent, based on a number of factors including link clock rate, the internal clock rate of the device, and the desired system behavior.

The physical layer time-out occurs between the transmission of a packet and the receipt of an acknowledgment control symbol. This time-out interval is likely to be comparatively short because the packet and acknowledgment pair must only traverse a single link. For the purpose of error recovery, a port link time-out should be treated as an unexpected acknowledgment control symbol.

Certain GSM operations may require two response transactions, and both must be received for the operation to be considered complete. In the case of a device implementation with multiple links, one response packet may be returned on the same link where the operation was initiated and the other response packet may be returned on a different link. If this behavior is supported by the issuing processing element, the port response time-out implementation must look for both responses, regardless on which links they are returned.

5.10.2 Link Behavior Under Error

The LP-Serial link uses an error detection and retransmission protocol to protect against and recover from transmission errors. Transmission error detection is done at the input port, and all transmission error recovery is also initiated at the input port.

The protocol requires that each packet transmitted be acknowledged by the receiving port and that a copy of each transmitted packet be retained by the sender until the sender receives a packet-accepted control symbol acknowledgment for the packet or the sending port determines that the packet has encountered an unrecoverable error. If the receiving ports detects a transmission error in a packet, the port sends a packet-not-accepted control symbol acknowledgment back to the sender indicating that the packet was corrupted as received. After a link-request/input-status and link-response control symbol exchange, the sender begins retransmission with either the packet that was corrupted during transmission or a higher priority packet if one is awaiting transmission.

All packets corrupted in transmission are retransmitted. The number of times a packet may be retransmitted before the sending port determines that the packet has encountered an unrecoverable condition is implementation dependent.

5.10.2.1 Recoverable Errors

The following four basic types of errors are detected by an LP-Serial port:

- An idle sequence error
- A control symbol error
- A packet error
- A time-out waiting for an acknowledgment control symbol

5.10.2.2 Idle Sequence Errors

The idle sequence is comprised of A, K, and R (8B/10B special) characters. If an input port detects an invalid character or any valid character other then A, K, or R in an idle sequence, it shall enter the Input Error-stopped state and follow the Input Error-stopped recovery process specified in Section , “5.10.2.6 Input Error-Stopped Recovery Process.

To limit input port complexity, the port is no required to determine the specific error that resulted in an idle sequence error. Following are several examples of idle sequence errors.

- A single bit transmission error can change an /A/, /K/, or /R/ code-group into a /Dx.y/ (data) code-group which is illegal in an idle sequence.
- A single bit transmission error can change an /A/, /K/, or /R/ code-group into an invalid code-group.
- A single bit transmission error can change an /SP/ or /PD/ (control symbol delimiters) into an invalid code-group.

5.10.2.3 Control Symbol Errors

There are three types of detectable control symbol errors

- An uncorrupted control symbol with a reserved encoding of an stype field
- An uncorrupted control symbol that violates the link protocol
- A corrupted control symbol

5.10.2.3.1 Reserved Stype Field Encodings

A control symbol with no detected corruption but with a reserved encoding of stype0 or stype1 shall be ignored for forward compatibility.

5.10.2.3.2 Link Protocol Violations

The reception of a control symbol with no detected corruption but that violates the link protocol shall cause the receiving port to immediately enter the Output Error-stopped state and follow the Output Error-stopped recovery process specified in Section , “An example state machine with the behavior described in this section is included in Section A.2, “Error Recovery"”.Output Error-Stopped Recovery Process.

Link protocol violations include the following:

- Unexpected packet-accepted, packet-retry, or packet-not-accepted control symbol
- Packet acknowledgment control symbol with an unexpected packet_ackID value
- Link time-out while waiting for an acknowledgment control symbol

The following is an example of a link protocol violation and recovery. A sender transmits packets labeled ackID 2, 3, 4, and 5. It receives acknowledgments for packets 2, 4, and 5, indicating a probable error associated with ackID 3. The sender then stops transmitting new packets and sends a link-request/input-status (restart-from-error) control symbol to the receiver. The receiver then returns a link- response control symbol indicating which packets it has received properly. These are the possible responses and the sender’s resulting behavior:

- expecting ackID = 3 - sender must retransmit packets 3, 4, and 5
- expecting ackID = 4 - sender must retransmit packets 4 and 5
- expecting ackID = 5 - sender must retransmit packet 5
- expecting ackID = 6 - receiver got all packets, resume operation
- expecting ackID = anything else - fatal (non-recoverable) errorCorrupted Control symbols

The reception of a control symbol with detected corruption shall cause the receiving port to immediately enter the Input Error-stopped state and follow the Input Error-stopped recovery process specified in Section , “5.10.2.6 Input Error-Stopped Recovery Process. For this type of error, the packet-not-accepted control symbol sent by the output side of the port as part of the recovery process shall have an unexpected packet_ackID value.

Input ports detect the following types of control symbol corruption.

- A control symbol containing invalid characters or valid but non-data characters
- A control symbol with an incorrect CRC valuePacket Errors

The reception of a packet with detected corruption shall cause the receiving port to immediately enter the Input Error-stopped state and follow the Input Error-stopped recovery process specified in Section , "5.10.2.6 Input Error-Stopped Recovery Process.

Input ports detect the following types of packet corruption

- Packet with an unexpected ackID value
- Packet with a incorrect CRC value
- Packet containing invalid characters or valid non-data characters
- Packet that overruns some defined boundary such as the maximum data payload.

5.10.2.5 Link Time-Out

A link time-out while waiting for an acknowledgment control symbol is handled as link protocol violation as described in Section , "5.10.2.3.2 Link Protocol Violations

5.10.2.6 Input Error-Stopped Recovery Process

When the input side of a port detects a transmission error, it immediately enters the Input Error-stopped state. To recover from this state, the input side of the port takes the following actions.

- Record the error(s) that caused the port to enter the Input Error-stopped state.
- If the detected error(s) occurred in a control symbol or packet, discard the control symbol or packet.
- Ignore all subsequently received packets while the port is in the Input Error-stopped state.
- Cause the output side of the port to issue a packet-not-accepted control symbol. If the error occurred in a packet, the control symbol packet_ackID field contains the ackID value from the errored packet. Otherwise, the control symbol packet_ackID field contains an unexpected ackID value. (The packet-not-accepted control symbol causes the output side of the receiving port to enter the Output Error-stopped state and send a link-request/input-status control symbol.)
- When an link-request/input-status control symbol is received, cause the output side of the port to issue a link-response control symbol, exit the Input Error-stopped state and resume packet reception.

An example state machine with the behavior described in this section is included in Section A.2, "Error Recovery"".Output Error-Stopped Recovery Process

To recover from the Output Error-stopped state, the output side of a port takes the following actions.

- Immediately stops transmitting new packets.
- Resets the link packet acknowledgment timers for all transmitted but unacknowledged packets. (This prevents the generation of spurious time-out errors.)
- Transmits an input-status link-request/input-status (restart-from-error) control symbol. (The input status link-request/input-status control symbol causes the receiving port to transmit a link-response control symbol that contains the input_status and ackID_status of the input side of the port. The ackID_status is the ackID value that is expected in the next packet that the port receives.)
- When the link-response is received, the port backs up the first unaccepted packet, exits the Output Error-stopped state and resumes transmission with either the first unaccepted packet or a higher priority packet.

An example state machine with the behavior described in this section is included in Section A.2, "Error Recovery"".

5.11 Power Management

Power management is currently beyond the scope of this specification and is implementation dependent. A device that supports power management features can make these features accessible to the rest of the system using the device's local configuration registers.

6 Chapter 6 - LP-Serial Registers

This chapter specifies the 1x/4x LP-Serial Command and Status Register (CSR) set. All registers in the set are 32-bits long and aligned to a 32-bit boundary. These registers allow an external processing element to determine the capabilities, configuration, and status of a processing element using this 1x/4x LP-Serial physical layer. The registers can be accessed using the maintenance operations defined in *Part I: Input/Output Logical Specification*.

These registers are located in the 1x/4x LP-Serial physical features block which is an Extended Features block in the Extended Features Space. The block may exist in any position in the Extended Features data structure and may exist in any portion of the Extended Features address space implemented by a device. (The Extended Features Space is located at byte offsets 0x0100 through 0xFFFC of the device Configuration Space.)

Register offsets into the block that are not defined are reserved for this specification unless otherwise stated. Read and write accesses to reserved register offsets shall terminate normally and shall not cause an error condition in the target device.

This chapter specifies only the registers and register bits that comprise the 1x/4x LP-Serial Command and Status Register set. Refer to the other applicable RapidIO logical and transport specifications for the specification of the complete set of registers and register bits required for a given device.

Table describes the required behavior for accesses to reserved register bits and reserved registers for the RapidIO Extended Features register space.

Table 6-1. Extended Feature Space Reserved Access Behavior

Byte Offset	Space Name	Item	Initiator behavior	Target behavior
0x100–FFFC	Extended Features Space	Reserved bit	read - ignore returned value[1]	read - return logic 0
			write - preserve current value[2]	write - ignored
		Implementation-defined bit	read - ignore returned value unless implementation-defined function understood	read - return implementation-defined value
			write - preserve current value if implementation-defined function not understood	write - implementation-defined
		Reserved register	read - ignore returned value	read - return logic 0s
			write -	write - ignored

1. Do not depend on reserved bits being a particular value; use appropriate masks to extract defined bits from the read value.
2. All register writes shall be in the form: read the register to obtain the values of all reserved bits, merge in the desired values for defined bits to be modified, and write the register, thus preserving the value of all reserved bits.

This chapter is divided up into three sections, each addressing a different type of RapidIO device.

6.1 Generic End Point Devices

This section describes the 1x/4x LP-Serial registers for a general end point device. This Extended Features register block is assigned Extended Features block ID=0x0004.

6.1.1 Register Map

Table 6-2 shows the register map for generic RapidIO 1x/4x LP-Serial end point devices. The Block Offset is the offset relative to the 16-bit Extended Features Pointer (EF_PTR) that points to the beginning of the block.

The address of a byte in the block is calculated by adding the block byte offset to EP_PTR that points to the beginning of the block. This is denoted as [EF_PTR+xx] where xx is the block byte offset in hexadecimal.

This register map is currently only defined for devices with up to 16 RapidIO ports, but can be extended or shortened if more or less port definitions are required for a device. For example, a device with four RapidIO ports is only required to use register map space corresponding to offsets [EF_PTR + 0x00] through [EF_PTR + 0xB8]. Register map offset [EF_PTR + 0xC0] can be used for another Extended Features block.

Table 6-2. LP-Serial Register Map - Generic End Point Device

	Block Byte Offset	Register Name (Word 0)	Register Name (Word 1)
	0x0	1x/4x LP-Serial Port Maintenance Block Header	
	0x8–18	Reserved	
General	0x20	Port Link Time-Out Control CSR	Port Response Time-Out Control CSR
General	0x28	Reserved	
General	0x30	Reserved	
General	0x38	Reserved	Port General Control CSR
Port 0	0x40	Reserved	
Port 0	0x48	Reserved	
Port 0	0x50	Reserved	
Port 0	0x58	Port 0 Error and Status CSR	Port 0 Control CSR
Port 1	0x60	Reserved	
Port 1	0x68	Reserved	
Port 1	0x70	Reserved	
Port 1	0x78	Port 1 Error and Status CSR	Port 1 Control CSR
Port 2-14	0x80–218	Assigned to Port 2-14 CSRs	
Port 15	0x220	Reserved	
Port 15	0x228	Reserved	
Port 15	0x230	Reserved	
Port 15	0x238	Port 15 Error and Status CSR	Port 15 Control CSR

6.1.2 Command and Status Registers (CSRs)

Refer to Table 6-1 for the required behavior for accesses to reserved registers and register bits.

6.1.2.1 Port Maintenance Block Header 0 (Block Offset 0x0 Word 0)

The port maintenance block header 0 register contains the EF_PTR to the next EF_BLK and the EF_ID that identifies this as the generic end point port maintenance block header.

Table 6-3. Bit Settings for Port Maintenance Block Header 0

Bit	Name	Reset Value	Description
0-15	EF_PTR		Hard wired pointer to the next block in the data structure, if one exists
16-31	EF_ID	0x0004	Hard wired Extended Features ID

6.1.2.2 Port Maintenance Block Header 1 (Block Offset 0x0 Word 1)

The port maintenance block header 1 register is reserved.

Table 6-4. Bit Settings for Port Maintenance Block Header 1

Bit	Name	Reset Value	Description
0-31	—		Reserved

6.1.2.3 Port Link Time-out Control CSR (Block Offset 0x20 Word 0)

The port link time-out control register contains the time-out timer value for all ports on a device. This time-out is for link events such as sending a packet to receiving the corresponding acknowledge, and sending a link-request to receiving the corresponding link-response. The reset value is the maximum time-out interval, and represents between 3 and 6 seconds.

Table 6-5. Bit Settings for Port Link Time-out Control CSR

Bit	Name	Reset Value	Description
0–23	time-out value	All 1s	time-out interval value
24-31	—		Reserved

6.1.2.4 Port Response Time-out Control CSR (Block Offset 0x20 Word 1)

The port response time-out control register contains the time-out timer count for all ports on a device. This time-out is for sending a request packet to receiving the corresponding response packet.The reset value is the maximum time-out interval, and represents between 3 and 6 seconds.

Table 6-6. Bit Settings for Port Response Time-out Control CSR

Bit	Name	Reset Value	Description
0–23	time-out value	All 1s	time-out interval value
24-31	—		Reserved

6.1.2.5 Port General Control CSR (Block Offset 0x38 Word 1)

The port general control register contains control register bits applicable to all ports on a processing element.

Table 6-7. Bit Settings for Port General Control CSRs

Bit	Name	Reset Value	Description
0	Host	see footnote[1]	A Host device is a device that is responsible for system exploration, initialization, and maintenance. Agent or slave devices are typically initialized by Host devices. 0b0 - agent or slave device 0b1 - host device
1	Master Enable	see footnote[2]	The Master Enable bit controls whether or not a device is allowed to issue requests into the system. If the Master Enable is not set, the device may only respond to requests. 0b0 - processing element cannot issue requests 0b1 - processing element can issue requests
2	Discovered	see footnote[3]	This device has been located by the processing element responsible for system configuration 0b0 - The device has not been previously discovered 0b1 - The device has been discovered by another processing element
3-31	—		Reserved

1. The Host reset value is implementation dependent
2. The Master Enable reset value is implementation dependent
3. The Discovered reset value is implementation dependent

6.1.2.6 Port *n* Error and Status CSRs (Offsets 0x58, 78, ..., 238 Word 0)

These registers are accessed when a local processor or an external device wishes to examine the port error and status information.

Table 6-8. Bit Settings for Port *n* Error and Status CSRs

Bit	Name	Reset Value	Description
0-10	—		Reserved
11	Output Retry-encountered	0b0	Output port has encountered a retry condition.This bit is set when bit 13 is set. Once set, remains set until written with a logic 1 to clear.
12	Output Retried	0b0	Output port has received a packet-retry control symbol and can not make forward progress. This bit is set when bit 13 is set and is cleared when a packet-accepted or a packet-not-accepted control symbol is received (read-only).
13	Output Retry-stopped	0b0	Output port has received a packet-retry control symbol and is in the “output retry-stopped” state (read-only).
14	Output Error-encountered	0b0	Output port has encountered (and possibly recovered from) a transmission error. This bit is set when bit 15 is set. Once set, remains set until written with a logic 1 to clear.
15	Output Error-stopped	0b0	Output is in the “output error-stopped” state (read-only).
16-20	—		Reserved

Table 6-8. Bit Settings for Port *n* Error and Status CSRs

Bit	Name	Reset Value	Description
21	Input Retry-stopped	0b0	Input port is in the "input retry-stopped" state (read-only).
22	Input Error-encountered	0b0	Input port has encountered (and possibly recovered from) a transmission error. This bit is set when bit 23 is set. Once set, remains set until written with a logic 1 to clear.
23	Input Error-stopped	0b0	Input port is in the "input error-stopped" state (read-only).
24-26	—		Reserved
27	Port-write Pending	0b0	Port has encountered a condition which required it to initiate a Maintenance Port-write operation This bit is only valid if the device is capable of issuing a maintenance port-write transaction. Once set remains set until written with a logic 1 to clear.
28	—		Reserved
29	Port Error	0b0	Input or output port has encountered an error from which hardware was unable to recover. Once set, remains set until written with a logic 1 to clear.
30	Port OK	0b0	The input and output ports are initialized and the port is exchanging error-free control symbols with the attached device (read-only).
31	Port Uninitialized	0b1	Input and output ports are not initialized. This bit and bit 30 are mutually exclusive (read-only).

6.1.2.7 Port *n* Control CSR (Block Offsets 0x58, 78, ..., 238 Word 1)

The port *n* control registers contain control register bits for individual ports on a processing element.

Table 6-9. Bit Settings for Port *n* Control CSRs

Bit	Name	Reset Value	Description
0-1	Port Width	see footnote[1]	Hardware width of the port (read-only): 0b00 - Single-lane port 0b01 - Four-lane port 0b10 - 0b11 - Reserved
2-4	Initialized Port Width	see footnote[2]	Width of the ports after initialized (read only): 0b000 - Single-lane port, lane 0 0b001 - Single-lane port, lane 2 0b010 - Four-lane port 0b011 - 0b111 - Reserved
5-7	Port Width Override	0b000	Soft port configuration to override the hardware size: 0b000 - No override 0b001 - Reserved 0b010 - Force single lane, lane 0 0b011 - Force single lane, lane 2 0b100 - 0b111 - Reserved

Table 6-9. Bit Settings for Port *n* Control CSRs

Bit	Name	Reset Value	Description
8	Port Disable	0b0	Port disable: 0b0 - port receivers/drivers are enabled 0b1 - port receivers/drivers are disabled and are unable to receive/ transmit to any packets or control symbols
9	Output Port Enable	see footnote[3]	Output port transmit enable: 0b0 - port is stopped and not enabled to issue any packets except to route or respond to I/O logical MAINTENANCE packets, depending upon the functionality of the processing element. Control symbols are not affected and are sent normally. 0b1 - port is enabled to issue any packets
10	Input Port Enable	see footnote[4]	Input port receive enable: 0b0 - port is stopped and only enabled to route or respond I/O logical MAINTENANCE packets, depending upon the functionality of the processing element. Other packets generate packet-not-accepted control symbols to force an error condition to be signaled by the sending device. Control symbols are not affected and are received and handled normally. 0b1 - port is enabled to respond to any packet
11	Error Checking Disable	0b0	This bit disables all RapidIO transmission error checking 0b0 - Error checking and recovery is enabled 0b1 - Error checking and recovery is disabled Device behavior when error checking and recovery is disabled and an error condition occurs is undefined
12	Multicast-event Participant	see footnote[5]	Send incoming Multicast-event control symbols to this port (multiple port devices only)
13-30	—		Reserved
31	Port Type	0b1	This indicates the port type, parallel or serial (read only) 0b0 - Parallel port 0b1 - Serial port

1. The Port Width reset value is implementation dependent
2. The Initialized Port Width reset value is implementation dependent
3. The Output Port Enable reset value is implementation dependent
4. The Input Port Enable reset value is implementation dependent
5. The Multicast-event Participant reset value is implementation dependent

6.2 Generic End Point Devices, software assisted error recovery option

This section describes the 1x/4x LP-Serial registers for a general end point device that supports software assisted error recovery. This is most useful for devices that for whatever reason do not want to implement error recovery in hardware and to allow software to generate link-request control symbols and see the results of the responses. This Extended Features register block is assigned Extended Features block ID=0x0005.

6.2.1 Register Map

Table 6-10 shows the register map for generic RapidIO 1x/4x LP-Serial end point devices with software assisted error recovery. The Block Offset is the offset based on the Extended Features pointer (EF_PTR) to this block. This register map is currently only defined for devices with up to 16 RapidIO ports, but can be extended or shortened if more or less port definitions are required for a device. For example, a device with four RapidIO ports is only required to use register map space corresponding to offsets [EF_PTR + 0x00] through [EF_PTR + 0xB8]. Register map offset [EF_PTR + 0xC0] can be used for another Extended Features block.

Table 6-10. LP-Serial Register Map - Generic End Point Devices (SW assisted)

	Block Byte Offset	Register Name (Word 0)	Register Name (Word 1)
	0x0	1x/4x LP-Serial Port Maintenance Block Header	
	0x8–18	Reserved	
General	0x20	Port Link Time-Out Control CSR	Port Response Time-Out Control CSR
	0x28	Reserved	
	0x30	Reserved	
	0x38	Reserved	Port General Control CSR
Port 0	0x40	Port 0 Link Maintenance Request CSR	Port 0 Link Maintenance Response CSR
	0x48	Port 0 Local ackID Status CSR	Reserved
	0x50	Reserved	
	0x58	Port 0 Error and Status CSR	Port 0 Control CSR
Port 1	0x60	Port 1 Link Maintenance Request CSR	Port 1Link Maintenance Response CSR
	0x68	Port 1 Local ackID Status CSR	Reserved
	0x70	Reserved	
	0x78	Port 1 Error and Status CSR	Port 1 Control CSR
Port 2-14	0x80–218	Assigned to Port 2-14 CSRs	
Port 15	0x220	Port 15 Link Maintenance Request CSR	Port 15 Link Maintenance Response CSR
	0x228	Port 15 Local ackID Status CSR	Reserved
	0x230	Reserved	
	0x238	Port 15 Error and Status CSR	Port 15 Control CSR

6.2.2 Command and Status Registers (CSRs)

Refer to Table 6-1 for the required behavior for accesses to reserved registers and register bits.

6.2.2.1 Port Maintenance Block Header 0 (Block Offset 0x0 Word 0)

The port maintenance block header 0 register contains the EF_PTR to the next EF_BLK and the EF_ID that identifies this as the generic end point port maintenance block header.

Table 6-11. Bit Settings for Port Maintenance Block Header 0

Bit	Name	Reset Value	Description
0-15	EF_PTR		Hard wired pointer to the next block in the data structure, if one exists
16-31	EF_ID	0x0005	Hard wired Extended Features ID

6.2.2.2 Port Maintenance Block Header 1 (Block Offset 0x0 Word 1)

The port maintenance block header 1 register is reserved.

Table 6-12. Bit Settings for Port Maintenance Block Header 1

Bit	Name	Reset Value	Description
0-31	—		Reserved

6.2.2.3 Port Link Time-out Control CSR (Block Offset 0x20 Word 0)

The port link time-out control register contains the time-out timer value for all ports on a device. This time-out is for link events such as sending a packet to receiving the corresponding acknowledge and sending a link-request to receiving the corresponding link-response. The reset value is the maximum time-out interval, and represents between 3 and 6 seconds.

Table 6-13. Bit Settings for Port Link Time-out Control CSR

Bit	Name	Reset Value	Description
0–23	time-out value	All 1s	time-out interval value
24-31	—		Reserved

6.2.2.4 Port Response Time-out Control CSR (Block Offset 0x20 Word 1)

The port response time-out control register contains the time-out timer count for all ports on a device. This time-out is for sending a request packet to receiving the corresponding response packet.The reset value is the maximum time-out interval, and represents between 3 and 6 seconds.

Table 6-14. Bit Settings for Port Response Time-out Control CSR

Bit	Name	Reset Value	Description
0–23	time-out value	All 1s	time-out interval value
24-31	—		Reserved

6.2.2.5 Port General Control CSR (Block Offset 0x38 Word 1)

The port general control register contains control register bits applicable to all ports on a processing element.

Table 6-15. Bit Settings for Port General Control CSRs

Bit	Name	Reset Value	Description
0	Host	see footnote[1]	A Host device is a device that is responsible for system exploration, initialization, and maintenance. Agent or slave devices are initialized by Host devices. 0b0 - agent or slave device 0b1 - host device
1	Master Enable	see footnote[2]	The Master Enable bit controls whether or not a device is allowed to issue requests into the system. If the Master Enable is not set, the device may only respond to requests. 0b0 - processing element cannot issue requests 0b1 - processing element can issue requests

Table 6-15. Bit Settings for Port General Control CSRs

Bit	Name	Reset Value	Description
2	Discovered	see footnote[3]	This device has been located by the processing element responsible for system configuration 0b0 - The device has not been previously discovered 0b1 - The device has been discovered by another processing element
3-31	—		Reserved

1. The Host reset value is implementation dependent
2. The Master Enable reset value is implementation dependent
3. The Discovered reset value is implementation dependent

6.2.2.6 Port *n* Link Maint. Request CSRs (Offsets 0x40, 60, ..., 220 Word 0)

The port link maintenance request registers are accessible both by a local processor and an external device. A write to one of these registers generates a link-request control symbol on the corresponding RapidIO port interface.

Table 6-6. Bit Settings for Port *n* Link Maintenance Request CSRs

Bit	Name	Reset Value	Description
0–28	—		Reserved
29-31	Command	0b000	Command to be sent in the link-request control symbol. If read, this field returns the last written value.

6.2.2.7 Port *n* Link Maintenance Response CSRs (0x40, 60, ..., 220 Word 1)

The port link maintenance response registers are accessible both by a local processor and an external device. A read to this register returns the status received in a link-response control symbol. The ackID_status and port_status fields are defined in Table 3-3 and Table 3-5. This register is read-only.

Table 6-17. Bit Settings for Port *n* Link Maintenance Response CSRs

Bit	Name	Reset Value	Description
0	response_valid	0b0	If the link-request causes a link-response, this bit indicates that the link-response has been received and the status fields are valid. If the link-request does not cause a link-response, this bit indicates that the link-request has been transmitted. This bit automatically clears on read.
1-21	—		Reserved
22-26	ackID_status	0b00000	ackID status field from the link-response control symbol
27-31	link_status	0b00000	link status field from the link-response control symbol

6.2.2.8 Port *n* Local ackID CSRs (Block Offsets 0x48, 68, ..., 228 Word 0)

The port link local ackID status registers are accessible both by a local processor and an external device. A read to this register returns the local ackID status for both the out and input ports of the device.

Bit Settings for Port *n* Local ackID Status CSRs

Bit	Name	Reset Value	Description
0-2	—		Reserved
3-7	Inbound_ackID	0b0000 0	Input port next expected ackID value
8-15	—		Reserved
19-23	Outstanding_ackID	0x0000 0	Output port unacknowledged ackID status. Next expected acknowledge control symbol ackID field that indicates the ackID value expected in the next received acknowledge control symbol.
24-26	—		Reserved
27-31	Outbound_ackID	0b0000 0	Output port next transmitted ackID value. Software writing this value can force retransmission of outstanding unacknowledged packets in order to manually implement error recovery.

6.2.2.9 Port *n* Error and Status CSRs (Block Offset 0x58, 78, ..., 238 Word 0)

These registers are accessed when a local processor or an external device wishes to examine the port error and status information.

Table 6-19. Bit Settings for Port *n* Error and Status CSRs

Bit	Name	Reset Value	Description
0-10	—		Reserved
11	Output Retry-encountered	0b0	Output port has encountered a retry condition.This bit is set when bit 13 is set. Once set, remains set until written with a logic 1 to clear.
12	Output Retried	0b0	Output port has received a packet-retry control symbol and can not make forward progress. This bit is set when bit 13 is set and is cleared when a packet-accepted or a packet-not-accepted control symbol is received (read-only).
13	Output Retry-stopped	0b0	Output port has received a packet-retry control symbol and is in the “output retry-stopped” state (read-only).
14	Output Error-encountered	0b0	Output port has encountered (and possibly recovered from) a transmission error. This bit is set when bit 15 is set. Once set, remains set until written with a logic 1 to clear.
15	Output Error-stopped	0b0	Output is in the “output error-stopped” state (read-only).
16-20	—		Reserved
21	Input Retry-stopped	0b0	Input port is in the “input retry-stopped” state (read-only).
22	Input Error-encountered	0b0	Input port has encountered (and possibly recovered from) a transmission error. This bit is set when bit 23 is set. Once set, remains set until written with a logic 1 to clear.
23	Input Error-stopped	0b0	Input port is in the “input error-stopped” state (read-only).
24-26	—		Reserved

Table 6-19. Bit Settings for Port *n* Error and Status CSRs

Bit	Name	Reset Value	Description
27	Port-write Pending	0b0	Port has encountered a condition which required it to initiate a Maintenance Port-write operation This bit is only valid if the device is capable of issuing a maintenance port-write transaction. Once set remains set until written with a logic 1 to clear.
28	—		Reserved
29	Port Error	0b0	Input or output port has encountered an error from which hardware was unable to recover. Once set, remains set until written with a logic 1 to clear.
30	Port OK	0b0	The input and output ports are initialized and the port is exchanging error-free control symbols with the attached device (read-only).
31	Port Uninitialized	0b1	Input and output ports are not initialized. This bit and bit 30 are mutually exclusive (read-only).

6.2.2.10 Port *n* Control CSR (Block Offsets 0x58, 78, ..., 238 Word 1)

The port *n* control registers contain control register bits for individual ports on a processing element.

Table 6-20. Bit Settings for Port *n* Control CSRs

Bit	Name	Reset Value	Description
0-1	Port Width	see footnote[1]	Hardware width of the port (read-only): 0b00 - Single-lane port 0b01 - Four-lane port 0b10 - 0b11 - Reserved
2-4	Initialized Port Width	see footnote[2]	Width of the ports after initialized (read only): 0b000 - Single-lane port, lane 0 0b001 - Single-lane port, lane 2 0b010 - Four-lane port 0b011 - 0b111 - Reserved
5-7	Port Width Override	0b000	Soft port configuration to override the hardware size: 0b000 - No override 0b001 - Reserved 0b010 - Force single lane, lane 0 0b011 - Force single lane, lane 2 0b100 - 0b111 - Reserved
8	Port Disable	0b0	Port disable: 0b0 - port receivers/drivers are enabled 0b1 - port receivers/drivers are disabled and are unable to receive/ transmit to any packets or control symbols

Bit	Name	Reset Value	Description
9	Output Port Enable	see footnote[3]	Output port transmit enable: 0b0 - port is stopped and not enabled to issue any packets except to route or respond to I/O logical MAINTENANCE packets, depending upon the functionality of the processing element. Control symbols are not affected and are sent normally. 0b1 - port is enabled to issue any packets
10	Input Port Enable	see footnote[4]	Input port receive enable: 0b0 - port is stopped and only enabled to route or respond I/O logical MAINTENANCE packets, depending upon the functionality of the processing element. Other packets generate packet-not-accepted control symbols to force an error condition to be signaled by the sending device. Control symbols are not affected and are received and handled normally. 0b1 - port is enabled to respond to any packet
11	Error Checking Disable	0b0	This bit disables all RapidIO transmission error checking 0b0 - Error checking and recovery is enabled 0b1 - Error checking and recovery is disabled Device behavior when error checking and recovery is disabled and an error condition occurs is undefined
12	Multicast-event Participant	see footnote[5]	Send incoming Multicast-event control symbols to this port (multiple port devices only)
13-30	—		Reserved
31	Port Type	0b1	This indicates the port type, parallel or serial (read only) 0b0 - Parallel port 0b1 - Serial port

1. The Port Width reset value is implementation dependent
2. The Initialized Port Width reset value is implementation dependent
3. The Output Port Enable reset value is implementation dependent
4. The Input Port Enable reset value is implementation dependent
5. The Multicast-Event Participant reset value is implementation dependent

6.3 Generic End Point Free Devices

This section describes the 1x/4x LP-Serial registers for a general devices that do not contain end point functionality (i.e. switches). This Extended Features register block uses extended features block ID=0x0006.

6.3.1 Register Map

Table 6-21 shows the register map for generic RapidIO 1x/4x LP-Serial end point-free devices. The Block Offset is the offset based on the Extended Features pointer (EF_PTR) to this block. This register map is currently only defined for devices with up to 16 RapidIO ports, but can be extended or shortened if more or less port definitions are required for a device. For example, a device with four RapidIO ports is only required to use register map space corresponding to offsets [EF_PTR + 0x00] through [EF_PTR + 0xB8]. Register map offset [EF_PTR + 0xC0] can be used for another Extended Features block.

Table 6-21. LP-Serial Register Map - Generic End Point Free

	Block Byte Offset	Register Name (Word 0)	Register Name (Word 1)
	0x0	1x/4x LP-Serial Port Maintenance Block Header	
	0x8–18	Reserved	
General	0x20	Port Link Time-Out Control CSR	Reserved
	0x28	Reserved	
	0x30	Reserved	
	0x38	Reserved	Port General Control CSR
Port 0	0x40	Reserved	
	0x48	Reserved	
	0x50	Reserved	
	0x58	Port 0 Error and Status CSR	Port 0 Control CSR
Port 1	0x60	Reserved	
	0x68	Reserved	
	0x70	Reserved	
	0x78	Port 1 Error and Status CSR	Port 1 Control CSR
Port 2-14	0x80–218	Assigned to Port 2-14 CSRs	
Port 15	0x220	Reserved	
	0x228	Reserved	
	0x230	Reserved	
	0x238	Port 15 Error and Status CSR	Port 15 Control CSR

6.3.2 Command and Status Registers (CSRs)

Refer to Table 6-1 for the required behavior for accesses to reserved registers and register bits.

6.3.2.1 Port Maintenance Block Header 0 (Block Offset 0x0 Word 0)

The port maintenance block header 0 register contains the EF_PTR to the next EF_BLK and the EF_ID that identifies this as the generic end point port maintenance block header.

Table 6-22. Bit Settings for Port Maintenance Block Header 0

Bit	Name	Reset Value	Description
0-15	EF_PTR		Hard wired pointer to the next block in the data structure, if one exists
16-31	EF_ID	0x0006	Hard wired Extended Features ID

6.3.2.2 Port Maintenance Block Header 1 (Block Offset 0x0 Word 1)

The port maintenance block header 1 register is reserved.

Table 6-23. Bit Settings for Port Maintenance Block Header 1

Bit	Name	Reset Value	Description
0-31	—		Reserved

6.2.2.3 Port Link Time-out Control CSR (Block Offset 0x20 Word 0)

The port link time-out control register contains the time-out timer value for all ports on a device. This time-out is for link events such as sending a packet to receiving the corresponding acknowledge and sending a link-request to receiving the corresponding link-response. The reset value is the maximum time-out interval, and represents between 3 and 6 seconds.

Table 6-24. Bit Settings for Port Link Time-out Control CSR

Bit	Name	Reset Value	Description
0–23	time-out value	All 1s	time-out interval value
24-31	—		Reserved

6.3.2.4 Port General Control CSR (Block Offset 0x38 Word 1)

The port general control register contains control register bits applicable to all ports on a processing element.

Table 6-25. Bit Settings for Port General Control CSRs

Bit	Name	Reset Value	Description
0-31	—		Reserved

6.3.2.5 Port *n* Error and Status CSRs (Block Offsets 0x58, 78, .., 238 Word 0)

These registers are accessed when a local processor or an external device wishes to examine the port error and status information.

Table 6-26. Bit Settings for Port *n* Error and Status CSRs

Bit	Name	Reset Value	Description
0-10	—		Reserved
11	Output Retry-encountered	0b0	Output port has encountered a retry condition.This bit is set when bit 13 is set. Once set, remains set until written with a logic 1 to clear.
12	Output Retried	0b0	Output port has received a packet-retry control symbol and can not make forward progress. This bit is set when bit 13 is set and is cleared when a packet-accepted or a packet-not-accepted control symbol is received (read-only).
13	Output Retry-stopped	0b0	Output port has received a packet-retry control symbol and is in the "output retry-stopped" state (read-only).
14	Output Error-encountered	0b0	Output port has encountered (and possibly recovered from) a transmission error. This bit is set when bit 15 is set. Once set, remains set until written with a logic 1 to clear.
15	Output Error-stopped	0b0	Output is in the "output error-stopped" state (read-only).

Table 6-26. Bit Settings for Port *n* Error and Status CSRs

Bit	Name	Reset Value	Description
16-20	—		Reserved
21	Input Retry-stopped	0b0	Input port is in the “input retry-stopped” state (read-only).
22	Input Error-encountered	0b0	Input port has encountered (and possibly recovered from) a transmission error. This bit is set when bit 23 is set. Once set, remains set until written with a logic 1 to clear.
23	Input Error-stopped	0b0	Input port is in the “input error-stopped” state (read-only).
24-26	—		Reserved
27	Port-write Pending	0b0	Port has encountered a condition which required it to initiate a Maintenance Port-write operation This bit is only valid if the device is capable of issuing a maintenance port-write transaction. Once set remains set until written with a logic 1 to clear.
28	—		Reserved
29	Port Error	0b0	Input or output port has encountered an error from which hardware was unable to recover. Once set, remains set until written with a logic 1 to clear.
30	Port OK	0b0	The input and output ports are initialized and the port is exchanging error-free control symbols with the attached device (read-only).
31	Port Uninitialized	0b1	Input and output ports are not initialized. This bit and bit 30 are mutually exclusive (read-only).

6.3.2.6 Port *n* Control CSR (Block Offsets 0x58, 78, ..., 238 Word 1)

The port *n* control registers contain control register bits for individual ports on a processing element.

Table 6-27. Bit Settings for Port *n* Control CSRs

Bit	Name	Reset Value	Description
0-1	Port Width	see footnote[1]	Hardware width of the port (read-only): 0b00 - Single-lane port 0b01 - Four-lane port 0b10 - 0b11 - Reserved
2-4	Initialized Port Width	see footnote[2]	Width of the ports after initialized (read only): 0b000 - Single-lane port, lane 0 0b001 - Single-lane port, lane 2 0b010 - Four-lane port 0b011 - 0b111 - Reserved
5-7	Port Width Override	0b000	Soft port configuration to override the hardware size: 0b000 - No override 0b001 - Reserved 0b010 - Force single lane, lane 0 0b011 - Force single lane, lane 2 0b100 - 0b111 - Reserved

Bit	Name	Reset Value	Description
8	Port Disable	0b0	Port disable: 0b0 - port receivers/drivers are enabled 0b1 - port receivers/drivers are disabled and are unable to receive/ transmit to any packets or control symbols
9	Output Port Enable	see footnote[3]	Output port transmit enable: 0b0 - port is stopped and not enabled to issue any packets except to route or respond to I/O logical MAINTENANCE packets, depending upon the functionality of the processing element. Control symbols are not affected and are sent normally. 0b1 - port is enabled to issue any packets
10	Input Port Enable	see footnote[4]	Input port receive enable: 0b0 - port is stopped and only enabled to route or respond I/O logical MAINTENANCE packets, depending upon the functionality of the processing element. Other packets generate packet-not-accepted control symbols to force an error condition to be signaled by the sending device. Control symbols are not affected and are received and handled normally. 0b1 - port is enabled to respond to any packet
11	Error Checking Disable	0b0	This bit disables all RapidIO transmission error checking 0b0 - Error checking and recovery is enabled 0b1 - Error checking and recovery is disabled Device behavior when error checking and recovery is disabled and an error condition occurs is undefined
12	Multicast Event Participant	see footnote[5]	Send incoming Multicast-event control symbols to this port (multiple port devices only)
13-30	—		Reserved
31	Port Type	0b1	This indicates the port type, parallel or serial (read only) 0b0 - Parallel port 0b1 - Serial port

1. The Port Width reset value is implementation dependent
2. The Initialized Port Width reset value is implementation dependent
3. The Output Port Enable reset value is implementation dependent
4. The Input Port Enable reset value is implementation dependent
5. The Multicast-event Participant reset value is implementation dependent

7 Chapter 7 - Signal Descriptions

This chapter contains the signal pin descriptions for a RapidIO 1x/4x LP-Serial port. The interface is defined either as a single- or four-lane, full duplex, point-to-point interface using differential signaling. A single-lane implementation consists of 4 wires and a four-lane implementation consists of 16 wires. The electrical details are described in Chapter 8, "AC Electrical Specifications."

7.1 Signal Definitions

Table 7-1 provides a summary of the RapidIO 1x/4x LP-Serial signal pins as well as a short description of their functionality.

Table 7-1. 1x/4x LP-Serial Signal Description

Signal Name	I/O	Signal Meaning	Timing Comments
TD[0-3]	O	Transmit Data - The transmit data is a unidirectional point to point bus designed to transmit the packet information. The TD bus of one device is connected to the RD bus of the receiving device. TD[0] is used in 1x mode.	Clocking is embedded in data using 8B/10B encoding.
TD[0-3]	O	Transmit Data complement—These signals are the differential pairs of the TD signals.	
RD[0-3]	I	Receive Data - The receive data is a unidirectional point to point bus designed to receive the packet information. The RD bus of one device is connected to the TD bus of the receiving device. RD[0] is used in 1x mode.	
RD[0-3]	I	Receive Data complement—These signals are the differential pairs of the RD signals.	

7.2 Serial RapidIO Interface Diagrams

Figure shows the signal interface diagram connecting two 1x devices together with the RapidIO 1x/4x LP-Serial interconnect.

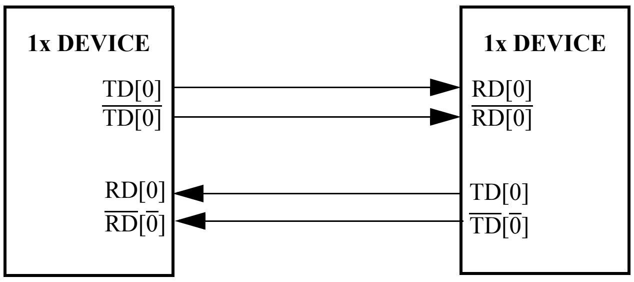


Figure 7-1. RapidIO 1x Device to 1x Device Interface Diagram

Figure 7-2 shows the signal interface diagram connecting two 4x devices together with the RapidIO 1x/4x LP-Serial interconnect.

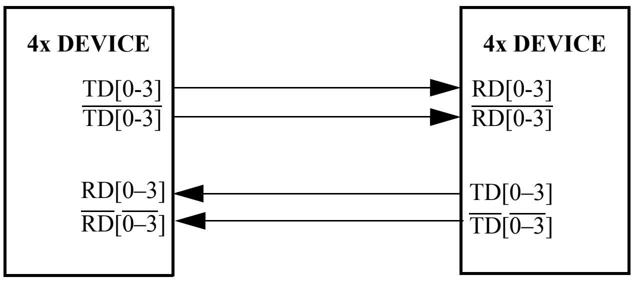


Figure 7-2. RapidIO 4x Device to 4x Device Interface Diagram

Figure 7-3 shows the connections between a 4x LP-Serial device and a 1x LP-Serial device.

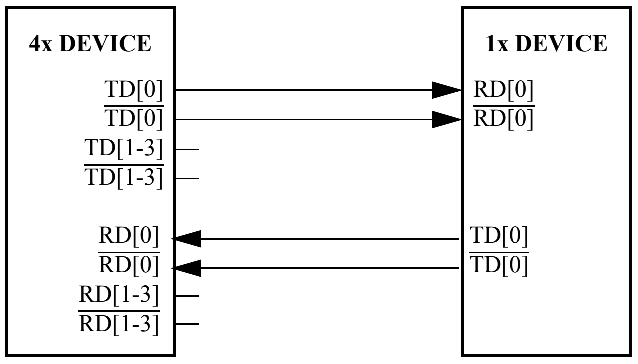

4x DEVICE
TD[0]
$\overline{\text{TD[0]}}$
TD[1-3]
$\overline{\text{TD[1-3]}}$
RD[0]
$\overline{\text{RD[0]}}$
RD[1-3]
$\overline{\text{RD[1-3]}}$
1x DEVICE
RD[0]
$\overline{\text{RD[0]}}$
TD[0]
$\overline{\text{TD[0]}}$

Figure 7-3. RapidIO 4x Device to 1x Device Interface Diagram

8 Chapter 8 - AC Electrical Specifications

8.1 Overview

The AC specifications covers both single and multiple links. The specifications define two types of transmitters and a single receiver. Baud rates of 1.25, 2.5, and 3.125 Gbps are specified. This chapter specifies differential signaling in quantities that represent the voltage difference between the true and complement signals. This known as the peak-peak voltage. The peak-peak voltage is twice that of the peak voltage of either the true or the complement signal. Specific definitions are given in Section : "8.2 Signal Definition".

Two transmitter specifications allow for solutions ranging from simple board-to-board interconnect to driving two connectors across a backplane. A single receiver specification is given that will accept signals from both the short run and long run transmitter specifications.

The short run transmitter should be used mainly for chip-to-chip connections on either the same printed circuit board or across a single connector. This covers the case where connections are made to a mezzanine (daughter) card. The minimum swings of the short run specification reduce the overall power used by the transceivers. A user can further reduce the power by lowering the termination voltages.

The long run transmitter specifications use larger voltage swings that are capable of driving signals across backplanes. This allow a user to drive signals across two connectors and a backplane. The specifications allow a distance of at least 50 cm at all frequencies.

All unit intervals are specified at +/- 100 ppm. The worst case frequency difference between any transmit and receive clock shall be 200 ppm.

To ensure interoperability between drivers and receivers of different vendors and technologies, AC coupling at the receiver input must be used.

8.2 Signal Definition

LP-serial protocol using differential signaling between ports. This section specifies signals using peak-to-peak differential voltages. Figure shows how the signals are defined. The figures shows waveforms for both a transmitter (TD and $\overline{\text{TD}}$) or a receiver (RD and $\overline{\text{RD}}$). Each signals swings between A volts and B volts. Using these waveforms, the definitions are as follows:

(1) The transmitter, or receiver, has a peak-to-peak range of A - B

(2) The differential signal of the transmitter, or receiver, ranges from +|A - B| to -|A - B|

(3) The peak differential signal of the transmitter, or receiver, is A - B

(4) The differential peak-to-peak signal of the transmitter, or receiver, is 2 * (A - B)

The term PEAK-TO-PEAK always means the difference between the most positive and the most negative readings of a particular signal. In this case we have (A - B) - (-(A - B)) = 2 * (A - B).

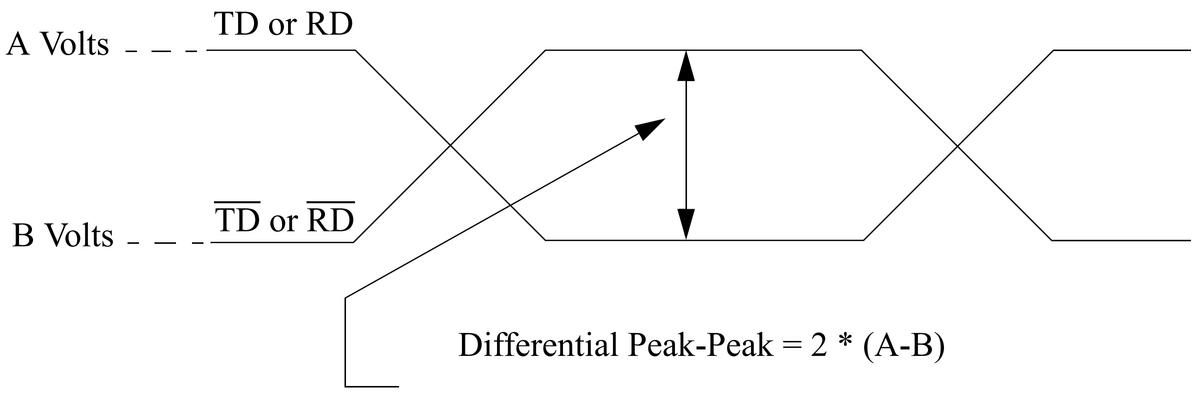

A Volts
TD or RD
B Volts
$\overline{\text{TD}}$ or $\overline{\text{RD}}$
Differential Peak-Peak = 2 * (A-B)

Figure 8-1. Differential Peak-Peak Voltage of Transmitter or Receiver

To illustrate this concept using real values, consider the case where a CML (Current Mode Logic) transmitter has a termination voltage of 2.5 V and has a swing that goes between 2.5V and 2.0V. Using these values, the peak-to-peak range is 500 mV. The differential signal ranges between 500 mV and -500 mV. The peak differential signal is 500 mV. The differential peak-to-peak signal is 1000 mV.Equalization

With the use of high speed serial links, the interconnect media will cause degradation of the signal at the receiver. Effects such as Inter-Symbol Interference (ISI) or data dependent jitter are produced. This loss can be large enough to degrade the eye opening at the receiver beyond what is allowed in the specification. To negate a portion of these effects, equalization can be used. The most common equalization techniques that can be used are:

- A passive high pass filter network placed at the receiver. This is often referred to as passive equalization.
- The use of active circuits in the receiver. This is often referred to as adaptive equalization.

8.4 Transmitter Specifications

Driver AC timing specifications are displayed in the tables below.

Table 8-1. Short Run Transmitter AC Timing Specifications - 1.25 Gbps Baud Rate

Characteristic	Symbol	Range		Unit	Notes
		Min	Max		
Differential Output Voltage	V_{DIFFPP}	500	2000	mV, pp	
Rise/Fall time (20% to 80%)	T_{RF}	40		ps	At driver output
Deterministic Jitter	J_D		0.12	UI	
Total Jitter	J_T		0.24	UI	
Output skew	S_O		25	ps	Skew at a transmitter output between the two signals comprising a differential pair
Multiple output skew	S_{MO}		1000	ps	Skew at the transmitter output between lanes of a multilane link
Unit Interval	UI	800	800	ps	+/- 100 ppm

1. AC coupling is required for interoperability between vendors.

Table 8-2. Short Run Transmitter AC Timing Specifications - 2.5 Gbps Baud Rate

Characteristic	Symbol	Range		Unit	Notes
		Min	Max		
Differential Output Voltage	V_{DIFFPP}	500	1000	mV, pp	
Rise/Fall time (20% to 80%)	T_{RF}	40		ps	At driver output
Deterministic Jitter	J_D		0.17	UI	
Total Jitter	J_T		0.35	UI	
Output skew	S_O		20	ps	Skew at a transmitter output between the two signals comprising a differential pair
Multiple Output skew	S_{MO}		1000	ps	Skew at the transmitter output between lanes of a multilane link
Unit Interval	UI	400	400	ps	+/- 100 ppm

1. AC coupling is required to guarantee interoperability between vendors.

Table 8-3. Short Run Transmitter AC Timing Specifications - 3.125 Gbps Baud Rate

Characteristic	Symbol	Range		Unit	Notes
		Min	Max		
Differential Output Voltage	V_{DIFFPP}	500	1000	mV, pp	
Rise/Fall time (20% to 80%)	T_{RF}	40		ps	At driver output
Deterministic Jitter	J_D		0.17	UI	
Total Jitter	J_T		0.35	UI	
Output Skew	S_O		15	ps	Skew at a transmitter output between the two signals comprising a differential pair
Multiple output skew	S_{MO}		1000	ps	Skew at the transmitter output between lanes of a multilane link
Unit Interval	UI	320	320	ps	+/- 100 ppm

1. AC coupling is required for interoperability between vendors.

Table 8-4. Long Run Transmitter AC Timing Specifications - 1.25 Gbps Baud Rate

Characteristic	Symbol	Range		Unit	Notes
		Min	Max		
Differential Output Voltage	V_{DIFFPP}	800	2000	mV, pp	
Rise/Fall time (20% to 80%)	T_{RF}	40		ps	At driver output
Deterministic Jitter	J_D		0.12	UI	
Total Jitter	J_T		0.24	UI	
Output skew	S_O		25	ps	Skew at a transmitter output between the two signals comprising a differential pair
Multiple output skew	S_{MO}		1000	ps	Skew at the transmitter output between lanes of a multilane link
Unit Interval	UI	800	800	ps	+/- 100 ppm

1. AC coupling is required to guarantee interoperability between vendors.

Table 8-5. Long Run Transmitter AC Timing Specifications - 2.5 Gbps Baud Rate

Characteristic	Symbol	Range		Unit	Notes
		Min	Max		
Differential Output Voltage	V_{DIFFPP}	1000	1600	mV, pp	
Rise/Fall time (20% to 80%)	T_{RF}	40		ps	At driver output
Deterministic Jitter	J_D		0.17	UI	
Total Jitter	J_T		0.35	UI	
Output skew	S_O		20	ps	Skew at a transmitter output between the two signals comprising a differential pair
Multiple output skew	S_{MO}		1000	ps	Skew at the transmitter output between lanes of a multilane link
Unit Interval	UI	400	400	ps	+/- 100 ppm

1. AC coupling is required for interoperability between vendors.

Table 8-6. Long Run Transmitter AC Timing Specifications - 3.125 Gbps Baud Rate

Characteristic	Symbol	Range		Unit	Notes
		Min	Max		
Differential Output Voltage	V_{DIFFPP}	1000	1600	mV, pp	
Rise/Fall time (20% to 80%)	T_{RF}	40		ps	At driver output
Deterministic Jitter	J_D		0.17	UI	
Total Jitter	J_T		0.35	UI	
Output skew	S_O		15	ps	Skew at a transmitter output between the two signals comprising a differential pair
Multiple output skew	S_{MO}		1000	ps	Skew at the transmitter output between lanes of a multilane link
Unit Interval	UI	320	320	ps	+/- 100 ppm

1. AC coupling is required for interoperability between vendors.

The output eye pattern of a LP-Serial transmitter shall fall entirely within the unshaded portion of the Transmitter Output Compliance Mask shown in igure 8-2 with the parameters specified in Table when measured at the output pins of the device.

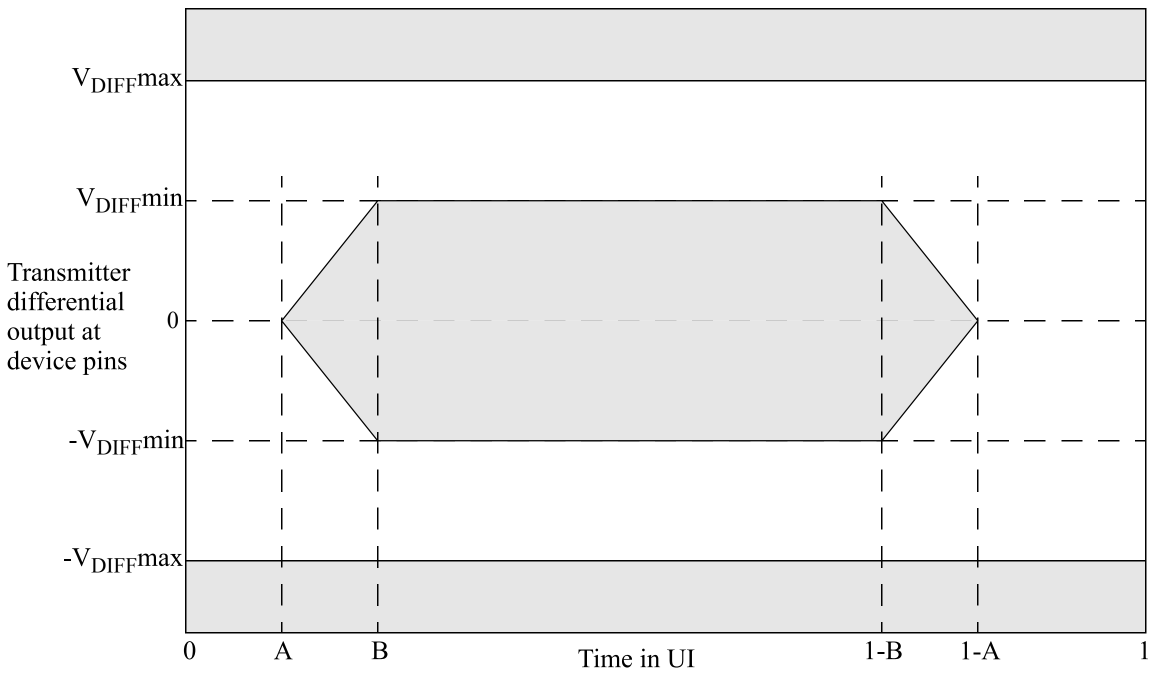


Figure 8-2. Transmitter Output Compliance Mask

Table 8-7. Transmitter Differential Output Eye Diagram Parameters

Transmitter Type	V_{DIFF}min (mV)	V_{DIFF}max (mV)	A (UI)	B (UI)
1.25 GBaud short range	250	1000	0.12	0.34
1.25 GBaud long range	400	1000	0.12	0.34
2.5 GBaud short range	250	500	0.175	0.39
2.5 GBaud long range	500	800	0.175	0.39
3.125 GBaud short range	250	500	0.175	0.39
3.125 GBaud long range	500	800	0.175	0.39

NOTES:

The values for B for 1.25 GBaud were taken from the values at TP2 for 10000BASE-CX in IEEE 802.3z. The parameter for B for 1.25 GBuad was previously unspecified by the LP-Serial Specification.

The values for B for 2.5 and 3.125 GBuad were taken from the values for output of a XAUI transmitter in IEEE 802.3ae. The parameter B for 2.50 and 3.125 GBuad was previously unspecified by the LP_Serial Specification.

8.5 Receiver Specifications

Receiver AC timing specifications are displayed in the tables below.

Table 8-8. Receiver AC Timing Specifications - 1.25 Gbps Baud Rate

Characteristic	Symbol	Range		Unit	Notes
		Min	Max		
Differential Input Voltage	V_{IN}	175	2000	mV	Peak-peak differential input voltage
Deterministic Jitter	J_D		0.45	UI	Measured at receiver
Total Jitter	J_T		0.71	UI	Measured at receiver
Input Skew	S_I		75	ps	Skew at a receiver input between the two signals comprising a differential pair
Multiple Input Skew	S_{MI}		24	ns	Skew at the receiver input between lanes of a multi-lane link
Bit Error Rate	BER		10^{-12}		
Unit Interval	UI	800	800	ps	+/- 100 ppm

1. AC coupling is required to guarantee interoperability between vendors.

Table 8-9. Receiver AC Timing Specifications - 2.5 Gbps Baud Rate

Characteristic	Symbol	Range		Unit	Notes
		Min	Max		
Differential Input Voltage	V_{IN}	175	1600	mV	Peak-peak differential input voltage
Deterministic Jitter	J_D		0.41	UI	Measured at receiver
Total Jitter	J_T		0.65	UI	Measured at receiver
Input Skew	S_I		75	ps	Skew at a receiver input between the two signals comprising a differential pair
Multiple Input Skew	S_{MI}		24	ns	Skew at the receiver input between lanes of a multi-lane link
Bit Error Rate	BER		10^{-12}		
Unit Interval	UI	400	400	ps	+/- 100 ppm

1. AC coupling is required to guarantee interoperability between vendors.

Table 8-10. Receiver AC Timing Specifications - 3.125 Gbps Baud Rate

Characteristic	Symbol	Range		Unit	Notes
		Min	Max		
Differential Input Voltage	V_{IN}	175	1600	mV	Peak-peak differential input voltage
Deterministic Jitter	J_D		0.36	UI	Measured at receiver
Total Jitter	J_T		0.6	UI	Measured at receiver
Input Skew	S_I		75	ps	Skew at a receiver input between the two signals comprising a differential pair
Multiple Input Skew	S_{MI}		22	ns	Skew at the receiver input between lanes of a multi-lane link
Bit Error Rate	BER		10^{-12}		
Unit Interval	UI	320	320	ps	+/- 100 ppm

1. AC coupling is required to guarantee interoperability between vendors.

8.6 Receiver Eye Diagrams

The following receiver eye openings are required to ensure proper operation.

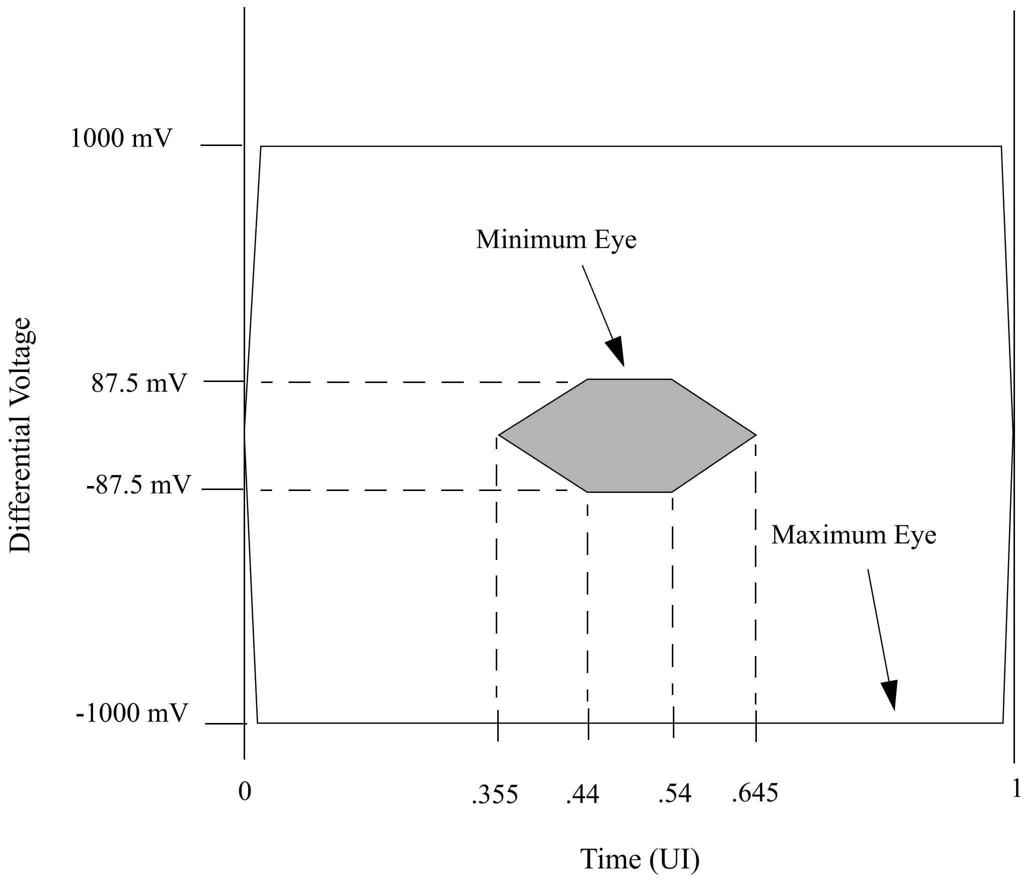


Figure 8-3. 1.25 Gbps Baud Rate Receiver Eye Diagram

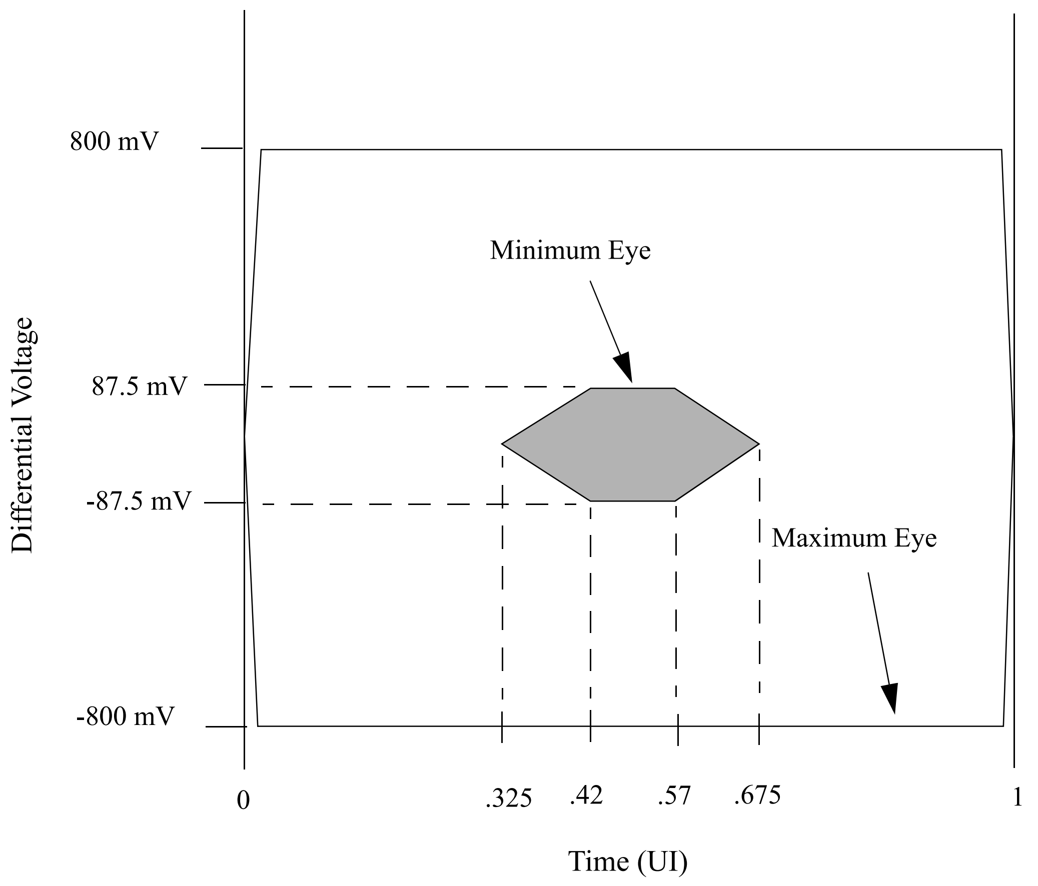


Figure 8.4. 2.5 Gbps Baud Rate Receiver Eye Diagram

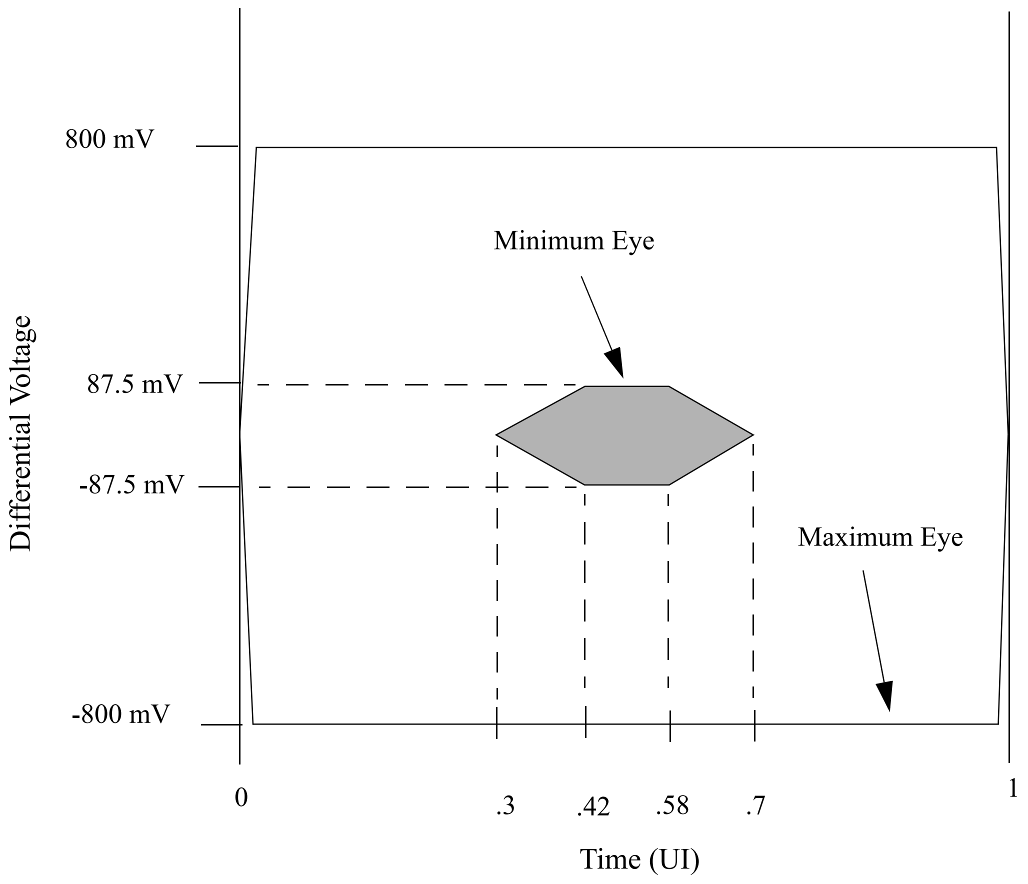


Figure 8-5. 3.125 Gbps Baud Rate Receiver Eye Diagram

A - Interface Management (Informative)

This annex contains state machine descriptions that illustrate a number of behaviors that are described in the *RapidIO Physical Layer 1x/4x LP-Serial Specification*. They are included as examples and are believed to be correct, however, actual implementations should not use the examples directly.

A.1 Packet Retry Mechanism

This section contains the example packet retry mechanism state machine referred to in Section 5.5, “Packet Transmission Protocol”.

Packet retry recovery actually requires two inter-dependent state machines in order to operate, one associated with the input port and the other with the output port on the two connected devices. The two state machines work together to Aattempt recovery from a retry condition.

A.1.1 Input port retry recovery state machine

If a packet cannot be accepted by a receiver for reasons other than error conditions, such as a full input buffer, the receiver follows the state sequence shown in Figure .

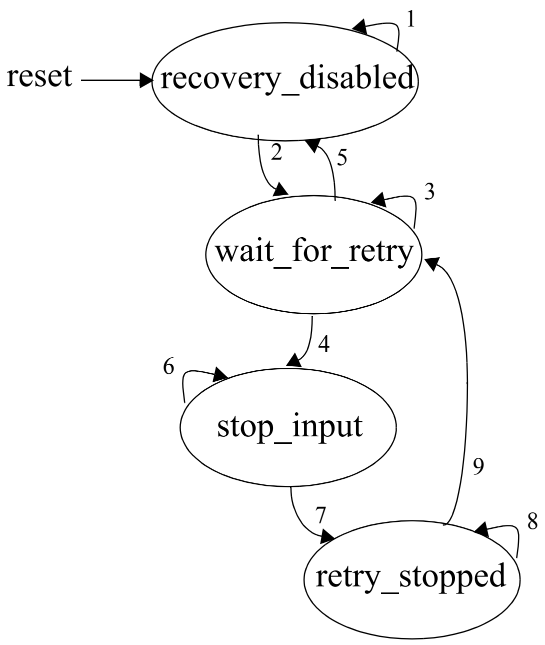


Figure A-1. Input Port Retry Recovery State Machine

Table A-1 describes the state transition arcs for Figure . The states referenced in the comments in quotes are the RapidIO 1x/4x LP-Serial defined status states, not states in this state machine.

Table A-1. Input Port Retry Recovery State Machine Transition Table (continued)

Arc	Current State	Next state	cause	Comments
1	recovery_disabled	recovery_disabled	Remain in this state until the input port is enabled to receive packets.	This is the initial state after reset. The input port can't be enabled before the initialization sequence has been completed, and may be controlled through other mechanisms as well, such as a software enable bit.
2	recovery_disabled	wait_for_retry	Input port is enabled.	
3	wait_for_retry	wait_for_retry	Remain in this state until a packet retry situation has been detected.	
4	wait_for_retry	stop_input	A packet retry situation has been detected.	Usually this is due to an internal resource problem such as not having packet buffers available for low priority packets.
5	wait_for_retry	recovery_disabled	Input port is disabled.	
6	stop_input	stop_input	Remain in this state until described input port stop activity is completed.	Send a packet-retry control symbol with the expected ackID, discard the packet, and don't change the expected ackID. This will force the attached device to initiate recovery starting at the expected ackID. Clear the "Port Normal" state and set the "Input Retry-stopped" state.
7	stop_input	retry_stopped	Input port stop activity is complete.	
8	retry_stopped	retry_stopped	Remain in this state until a restart-from-retry or link request (restart-from-error) control symbol is received or an input port error is encountered.	The "Input Retry-stopped" state causes the input port to silently discard all incoming packets and not change the expected ackID value.
9	retry_stopped	wait_for_retry	Received a restart-from-retry or a link request (restart-from-error) control symbol or an input port error is encountered.	Clear the "Input Retry-stopped" state and set the "Port Normal" state. An input port error shall cause a clean transition between the retry recovery state machine and the error recovery state machine.

A.1.2 Output port retry recovery state machine

On receipt of an error-free packet-retry control symbol, the attached output port follows the behavior shown in Figure . The states referenced in the comments in quotes are the RapidIO 8/16 LP-LVDS defined status states, not states in this state machine.

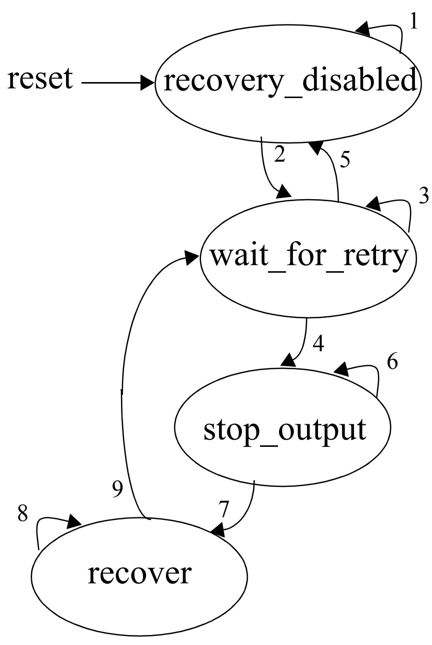


Figure A-2. Output Port Retry Recovery State Machine

Table A-2 describes the state transition arcs for Figure A-2.

Table A-2. Output Port Retry Recovery State Machine Transition Table (continued)

Arc	Current State	Next state	cause	Comments
1	recovery_disabled	recovery_disabled	Remain in this state until the output port is enabled to receive packets.	This is the initial state after reset. The output port can’t be enabled before the initialization sequence has been completed, and may be controlled through other mechanisms as well, such as a software enable bit.
2	recovery_disabled	wait_for_retry	Output port is enabled.	
3	wait_for_retry	wait_for_retry	Remain in this state until a packet-retry control symbol is received.	The packet-retry control symbol shall be error free.
4	wait_for_retry	stop_output	A packet-retry control symbol has been received.	Start the output port stop procedure.
5	wait_for_retry	recovery_disabled	Output port is disabled.	
6	stop_output	stop_output	Remain in this state until the output port stop procedure is completed.	Clear the “Port Normal” state, set the “Output Retry-stopped” state, and stop transmitting new packets.
7	stop_output	recover	Output port stop procedure is complete.	

Table A-2. Output Port Retry Recovery State Machine Transition Table (continued)

Arc	Current State	Next state	cause	Comments
8	recover	recover	Remain in this state until the internal recovery procedure is completed.	The packet sent with the ackID value returned in the packet-retry control symbol and all subsequent packets shall be retransmitted. Output port state machines and the outstanding ackID scoreboard shall be updated with this information, then clear the “Output Retry-stopped” state and set the “Port Normal” state to restart the output port. Receipt of a packet-not-accepted control symbol or other output port error during this procedure shall cause a clean transition between the retry recovery state machine and the error recovery state machine. Send restart-from-retry control symbol.
9	recover	wait_for_retry	Internal recovery procedure is complete.	Retransmission has started, so return to the wait_for_retry state to wait for the next packet-retry control symbol.

A.2 Error Recovery

This section contains the error recovery state machine referred to in Section 5.10.2, “Link Behavior Under Error.”

Error recovery actually requires two inter-dependent state machines in order to operate, one associated with the input port and the other with the output port on the two connected devices. The two state machines work together to attempt recovery.

A.2.1 Input port error recovery state machine

There are a variety of recoverable error types described in detail in Section 5.10.2, “Link Behavior Under Error”. The first group of errors are associated with the input port, and consists mostly of corrupt packet and control symbols. An example of a corrupt packet is a packet with an incorrect CRC. An example of a corrupt control symbol is a control symbol with error on the 5-bit CRC control symbol. The recovery state machine for the input port of a RapidIO link is shown in Figure A-3.

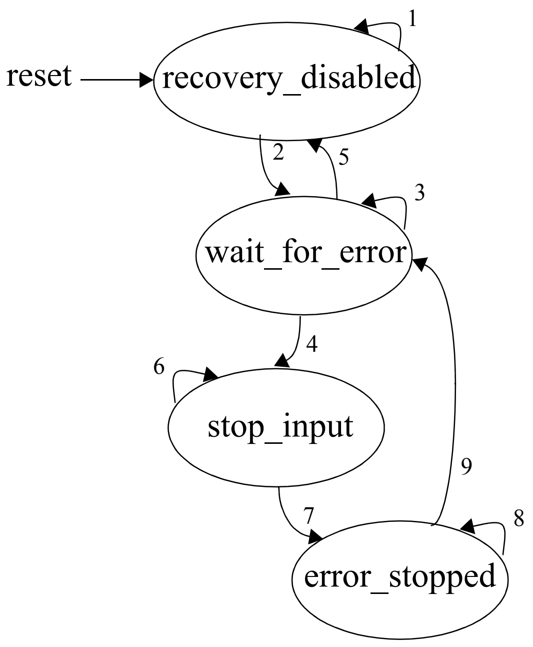


Figure A-3. Input Port Error Recovery State Machine

Table A-3 describes the state transition arcs for figure A-3. The states referenced in the comments in quotes are the RapidIO 1x/4x LP-Serial defined status states, not states in this state machine.

Table A-3. Input Port Error Recovery State Machine Transition

Arc	Current State	Next state	cause	Comments
1	recovery_disabled	recovery_disabled	Remain in this state until error recovery is enabled.	This is the initial state after reset. Error recovery can't be enabled before the initialization sequence has been completed, and may be controlled through other mechanisms as well, such as a software enable bit.
2	recovery_disabled	wait_for_error	Error recovery is enabled.	
3	wait_for_error	wait_for_error	Remain in this state until a recoverable error is detected.	Detected errors and the level of coverage is implementation dependent.
4	wait_for_error	stop_input	A recoverable error has been detected.	An output port associated error will not cause this transition, only an input port associated error.
5	wait_for_error	recovery_disabled	Error recovery is disabled.	
6	stop_input	stop_input	Remain in this state until described input port stop activity is completed.	Send a packet-not-accepted control symbol and, if the error was on a packet, discard the packet and don't change the expected ackID value. This will force the attached device to initiate recovery. Clear the "Port Normal" state and set the "Input Error-stopped" state.
7	stop_input	error_stopped	Input port stop activity is complete.	

Table A-3. Input Port Error Recovery State Machine Transition

Arc	Current State	Next state	cause	Comments
8	error_stopped	error_stopped	Remain in this state until a link request (restart-from-error) control symbol is received.	The "Input Error-stopped" state causes the input port to silently discard all subsequent incoming packets and ignore all subsequent input port errors.
9	error_stopped	wait_for_error	Received a link request (restart-from-error) control symbol.	Clear the "Input Error-stopped" state and set the "Port Normal" state, which will put the input port back in normal operation.

A.2.2 Output port error recovery state machine

The second recoverable group of errors described in Section 5.10.2, "Link Behavior Under Error" is associated with the output port, and is comprised of control symbols that are error-free and indicate that the attached input port has detected a transmission error or some other unusual situation has occurred. An example of this situation is indicated by the receipt of a packet-not-accepted control symbol. The state machine for the output port is shown in Figure A-4.

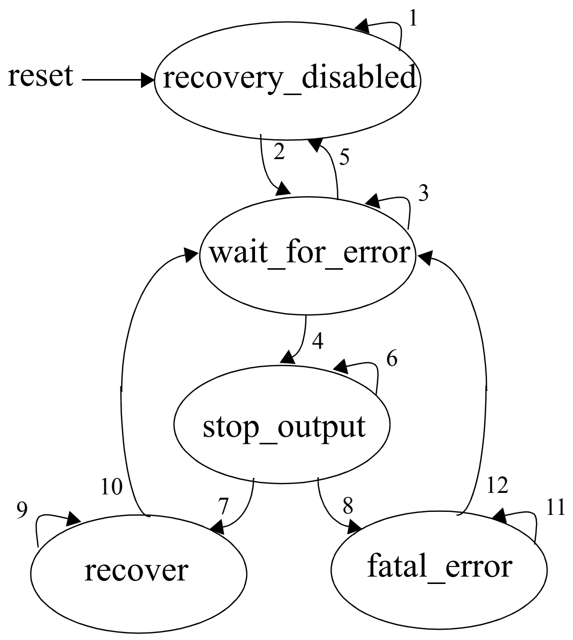


Figure A-4. Output Port Error Recovery State Machine

Table A-4 describes the state transition arcs for figure A-4. The states referenced in the comments in quotes are the RapidIO 8/16 LP-LVDS defined status states, not states in this state machine.

Table A-4. Output Port Error Recovery State Machine Transition

Arc	Current State	Next state	cause	Comments
1	recovery_disabled	recovery_disabled	Remain in this state until error recovery is enabled.	This is the initial state after reset. Error recovery can't be enabled before the initialization sequence has been completed, and may be controlled through other mechanisms as well, such as a software enable bit.
2	recovery_disabled	wait_for_error	Error recovery is enabled.	
3	wait_for_error	wait_for_error	Remain in this state until a recoverable error is detected.	Detected errors and the level of coverage is implementation dependent.
4	wait_for_error	stop_output	A recoverable error has been detected.	An input port associated error will not cause this transition, only an output port associated error.
5	wait_for_error	recovery_disabled	Error recovery is disabled.	
6	stop_output	stop_output	Remain in this state until an exit condition occurs.	Clear the "Port Normal" state, set the "Output Error-stopped" state, stop transmitting new packets, and send a link-request/input-status control symbol. Ignore all subsequent output port errors. The input on the attached device is in the "Input Error-stopped" state and is waiting for a link-request/input-status in order to be re-enabled to receive packets. An implementation may wish to time-out several times before regarding a time-out as fatal using a threshold counter or some other mechanism.
7	stop_output	recover	The link-response is received and returned an outstanding ackID value	An outstanding ackID is a value sent out on a packet that has not been acknowledged yet. In the case where no ackID is outstanding the returned ackID value shall match the next expected/next assigned ackID value, indicating that the devices are synchronized. Recovery is possible, so follow recovery procedure.
8	stop_output	fatal_error	The link-response is received and returned an ackID value that is not outstanding, or timed out waiting for the link-response.	Recovery is not possible, so start error shutdown procedure.

Table A-4. Output Port Error Recovery State Machine Transition

Arc	Current State	Next state	cause	Comments
9	recover	recover	Remain in this state until the internal recovery procedure is completed.	The packet sent with the ackID value returned in the link-response and all subsequent packets shall be retransmitted. All packets transmitted with ackID values preceding the returned value were received by the attached device, so they are treated as if packet-accepted control symbols have been received for them. Output port state machines and the outstanding ackID scoreboard shall be updated with this information, then clear the “Output Error-stopped” state and set the ‘Port Normal” state to restart the output port.
10	recover	wait_for_error	The internal recovery procedure is complete.	retransmission (if any was necessary) has started, so return to the wait_for_error state to wait for the next error.
11	fatal_error	fatal_error	Remain in this state until error shutdown procedure is completed.	Clear the “Output Error-stopped” state, set the “Port Error” state, and signal a system error.
12	fatal_error	wait_for_error	Error shutdown procedure is complete.	Return to the wait_for_error state.

Annex B
(Informative)

Critical Resource Performance Limits

The RapidIO LP-Serial layer is intended for use over links whose length ranges from centimeters to tens of meters. The shortest length links will almost certainly use copper printed circuit board traces. The longer lengths will require the use of fiber optics (optical fiber and electro-optical converters) to overcome the high frequency losses of long copper printed circuit board traces or cable. The longer lengths will also have significant propagation delay which can degrade the usable bandwidth of a link.

The serial protocol is a handshake protocol. Each packet transmitted by a port is assigned an ID (the ackID) and a copy of the packet is retained by the port in a holding buffer until the packet is accepted by the port's link partner. The number of packets that a port can transmit without acknowledgment is limited to the lesser of the number of distinct ackIDs and the number of buffers available to hold unacknowledged packets. Which ever is the limiting resource, ackIDs or holding buffers, will be called the "critical resource".

The concern is the time between the assignment of a critical resource to a packet and the release of that resource as a consequence of the packet being accepted by the link partner. Call this time the resource_release_delay. When the resource_release_delay is less than the time it takes to transmit a number of packets equal to the number of distinct critical resource elements, there is no degradation of link performance. When the resource_release_delay is greater than the time it takes to transmit a number of packets equal to the number of distinct critical resource elements, the transmitter may have to stall from time to time waiting for a free critical resource. This will degraded the usable link bandwidth. The onset of degradation will depend on the average length of transmitted packets and the physical length of the link as reflected in the resource_release_delay.

The following example provides some idea of the impact on link performance of the interaction between link length and a critical resource. For purposes of this example, the following assumptions are made.

1. The link is a 4 lane (4x) link.
2. The link uses optical fiber and electro-optical transceivers to allow link lengths of tens of meters. The propagation delay of the optical fiber is 0.45c.
3. The width of the data path within the port is 4 bytes.
4. The data path and logic within the port run at a clock rate equal to the aggregate unidirectional data rate of the link divided by 32. This is referred to as the logic clock. One cycle of this clock is referred to a one logic clock cycle. (If the aggregate unidirectional baud rate of the link was used to compute the logic clock, the baud rate would be divided by 40. With 8B/10B encoding, the baud rate is 1.25 times the data rate.)
5. The minimum length packet header is used. Write request packets have a length of 12 bytes plus a payload containing an integer multiple of 8 bytes. Read request packets have a length of 12 bytes. Read response packets have a length of 8 bytes plus a payload containing an integer multiple of 8 bytes.
6. The beginning and end of each packet is delimited by a control symbol. A single control symbol may delimit both the end of one packet and the beginning of the next packet.
7. Packet acknowledgments are carried in packet delimiter control symbols when ever possible to achieve the efficiency provided by the dual stype control symbol. This implies that a packet acknowledgment must wait for an end-of-packet control symbol if packet transmission is in progress when the packet acknowledgment becomes available.
8. The logic and propagation delay in the packet transmission direction is comprised of the following components.

Table B-11. Packet Transmission Delay Components

Item	Time required
Generate start-of-packet control symbol (critical resource is available)	1 logic clock cycle
Generate start-of-packet control symbol CRC	1 logic clock cycle
8B/10B encode delimiter and start-of-packet control symbol	1 logic clock cycle
Serialize and transmit delimiter and start-of-packet control symbol	1 logic clock cycle

Table B-11. Packet Transmission Delay Components

Item	Time required
PCB copper and electro-optical transmitter delay	2 ns
Optical fiber delay	fiber_length/0.45c
Electro-optical receiver and pcb copper delay	2 ns
Receive and deserialize delimiter and start-of-packet control symbol	0.5 logic clock cycles
Receive and deserialize packet	depends on packet
Receive and deserialize delimiter and end-of-packet control symbol	1 logic clock cycle
8B/10B decode delimiter and end-of-packet control symbol	1 logic clock cycle
Check CRC of end-of-packet control symbol	1 logic clock cycle
Make packet acceptance decision	1 logic clock cycle

9. The logic and propagation delay in the packet acknowledgment direction is comprised of the following.

Table B-12. Packet Acknowledgment Delay Components

Item	Time required
Wait for end-of-packet if packet transmission is in progress, generate packet-acknowledgment control symbol and control symbol CRC	depends on packet >= 2 logic clock cycles
8B/10B encode delimiter and packet-acknowledgment control symbol	1 logic clock cycle
Serialize and transmit delimiter and packet-acknowledgment control symbol	1 logic clock cycle
PCB copper and electro-optical transmitter delay	2 ns
Optical fiber delay	fiber_length/0.45c
Electro-optical receiver and pcb copper delay	2 ns
Receive and deserialize delimiter and packet-acknowledgment control symbol	0.5 logic clock cycles
8B/10B decode delimiter and packet-acknowledgment control symbol	1 logic clock cycle
Check CRC of packet-acknowledgment control symbol	1 logic clock cycle
Make decision to free critical resource	1 logic clock cycle

The packet times in the above tables depend on packet length which in turn depends on packet type and payload size. Since packet traffic will typically involve a mixture of packet types and payload sizes, the traffic in each direction will be assumed to contain an equal number of read, write and response packets and average payloads of 8, 32, and 64 bytes.

The number of logic clock cycles required to transmit or receive a packet is given in the following table as a function of packet type and payload size.

Table B-13. Packet Delays

Packet Type	Packet Header bytes	Data Payload bytes	Transmit/Receive Time logic clock cycles
Read	12	0	3

Table B-13. Packet Delays

Packet Type	Packet Header bytes	Data Payload bytes	Transmit/Receive Time logic clock cycles
Response	8	8	4
		32	10
		64	18
Write	12	8	5
		32	11
		64	19

Using the above table and the assumed equal number of read, write and response packets, the average number of logic clock cycles to transmit or received a packet is 4, 8, and 13.3 respectively for packet payloads of 8, 32, and 64 bytes. The average wait for the completion of a packet being transmitted is assumed to be 1/2 the transmit time.

The following table gives the maximum length of the optical fiber before the packet transmission rate becomes limited by the critical resource for a 4x link operating at unidirectional data rates of 4.0, 8.0 and 10.0 Gb/s.

Table B-14. Maximum Transmission Distances

Number of Critical Resources Available	Data Payload (Bytes)	Maximum Fiber Length Before Critical Resource Limited (Meters)		
		4.0 Gb/s link	8.0 Gb/s link	10.0 Gb/s link
4	8	-	-	-
	32	4.3	1.9	1.4
	64	11.4	5.5	4.3
8	8	9.7	4.6	3.5
	32	23.6	11.5	9.1
	64	42.2	20.8	16.6
16	8	31.1	15.3	12.1
	32	62.2	30.8	24.6
	64	103.7	51.6	41.1
24	8	52.5	26.0	20.7
	32	100.8	50.2	40.0
	64	165.2	82.3	65.7
32	8	74.0	36.7	29.3
	32	139.5	69.5	55.5
	64	226.7	113.1	90.3

Glossary of Terms and Abbreviations

The glossary contains an alphabetical list of terms, phrases, and abbreviations used in this book.

A

AC Coupling. A method of connecting two devices together that does not pass DC.

Agent. A processing element that provides services to a processor.

ANSI. American National Standards Institute.

B

Big-endian. A byte-ordering method in memory where the address n of a word corresponds to the most significant byte. In an addressed memory word, the bytes are ordered (left to right) 0, 1, 2, 3, with 0 being the most significant byte.

Bridge. A processing element that connects one computer bus to another, allowing a processing element on one bus to access an processing element on the other.

C

Capability registers (CARs). A set of read-only registers that allow a processing element to determine another processing element's capabilities.

Code-group. A 10-bit entity produced by the 8B/10B encoding process and the input to the 8B/10B decoding process.

Command and status registers (CSRs). A set of registers that allow a processing element to control and determine the status of another processing element's internal hardware.

Control symbol. A quantum of information transmitted between two linked devices to manage packet flow between the devices.

CRC. Cyclic redundancy code

D

Deadlock. A situation in which two processing elements that are sharing resources prevent each other from accessing the resources, resulting in a halt of system operation.

Deferred or delayed transaction. The process of the target of a transaction capturing the transaction and completing it after responding to the the source with a retry.

Destination. The termination point of a packet on the RapidIO interconnect, also referred to as a target.

Device. A generic participant on the RapidIO interconnect that sends or receives RapidIO transactions, also called a processing element.

Device ID. The identifier of a processing element connected to the RapidIO interconnect.

Direct Memory Access (DMA). A process element that can independently read and write system memory.

Distributed memory. System memory that is distributed throughout the system, as opposed to being centrally located.

Double word. An eight byte quantity, aligned on eight byte boundaries.

E

EMI. Electromagnetic Interference.

End point. A processing element which is the source or destination of transactions through a RapidIO fabric.

End point device. A processing element which contains end point functionality.

End point free device. A processing element which does not contain end point functionality.

Ethernet. A common local area network (LAN) technology.

External processing element. A processing element other than the processing element in question.

F

Fabric. A series of interconnected switch devices, typically used in reference to a switch fabric.

Field or Field name. A sub-unit of a register, where bits in the register are named and defined.

FIFO. First in, first out.

Full-duplex. Data can be transmitted in both directions between connected processing elements at the same time.

G

Globally shared memory (GSM). Cache coherent system memory that can be shared between multiple processors in a system.

H

Half-word. A two byte or 16-bit quantity, aligned on two byte boundaries.

Header. Typically the first few bytes of a packet, containing control information.

I

Initiator. The origin of a packet on the RapidIO interconnect, also referred to as a source.

I/O. Input-output.

IP. Intellectual Property

ITU. International Telecommunication Union.

L

Little-endian. A byte-ordering method in memory where the address n of a word corresponds to the least significant byte. In an addressed memory word, the bytes are ordered (left to right) 3, 2, 1, 0, with 3 being the most significant byte.

Local memory. Memory associated with the processing element in question.

LP. Link Protocol

LSB. Least significant byte.

LVDS. Low voltage differential signaling.

M	**Message passing**. An application programming model that allows processing elements to communicate through special hardware instead of through memory as with the globally shared memory programming model.
	MSB. Most significant byte.
N	**Non-coherent**. A transaction that does not participate in any system globally shared memory cache coherence mechanism.
O	**Operation. A set of transactions between end point devices in a RapidIO system (requests and associated responses) such as a read or a write.**
P	**Packet**. A set of information transmitted between devices in a RapidIO system.
	Payload. The user data embedded in the RapidIO packet.
	PCB. Printed circuit board.
	PCS. Physical Coding Sublayer.
	PMA. Physical Media Attachment.
	Port-write. An address-less write operation.
	Priority. The relative importance of a transaction or packet; in most systems a higher priority transaction or packet will be serviced or transmitted before one of lower priority.
	Processing Element (**PE**). A generic participant on the RapidIO interconnect that sends or receives RapidIO transactions, also called a device.
	Processor. The logic circuitry that responds to and processes the basic instructions that drive a computer.
R	**Receiver**. The RapidIO interface input port on a processing element.
S	**Sender**. The RapidIO interface output port on a processing element.
	Semaphore. A technique for coordinating activities in which multiple processing elements compete for the same resource.
	Serializer. A device which converts parallel data (such as 8-bit data) to a single bit-wide datastream.
	Source. The origin of a packet on the RapidIO interconnect, also referred to as an initiator.
	SRAM. Static random access memory.
	Switch. A multiple port processing element that directs a packet received on one of its input ports to one of its output ports.
T	**Target**. The termination point of a packet on the RapidIO interconnect, also referred to as a destination.

Transaction. A specific request or response packet transmitted between end point devices in a RapidIO system.

Transaction request flow. A sequence of transactions between two processing elements that have a required completion order at the destination processing element. There are no ordering requirements between transaction request flows.

W

Word. A four byte or 32 bit quantity, aligned on four byte boundaries.

Write port. Hardware within a processing element that is the target of a port-write operation.

Partition VII:

Inter-operability Specification System and Device

VII Partition VII - Inter-operability Specification System and Device

1 Chapter 1 - Overview

This chapter provides an overview of the *RapidIO™ System and Device Inter-operability Specification* document. This document assumes that the reader is familiar with the RapidIO specifications, conventions, and terminology.

1.1 Overview

The RapidIO Architectural specifications set a framework to allow a wide variety of implementations. This document provides a standard set of device and system design solutions to provide for inter-operability. The *RapidIO System and Device Inter-operability Specification* supplements the RapidIO Architecture Specifications as shown in Figure 1-1.

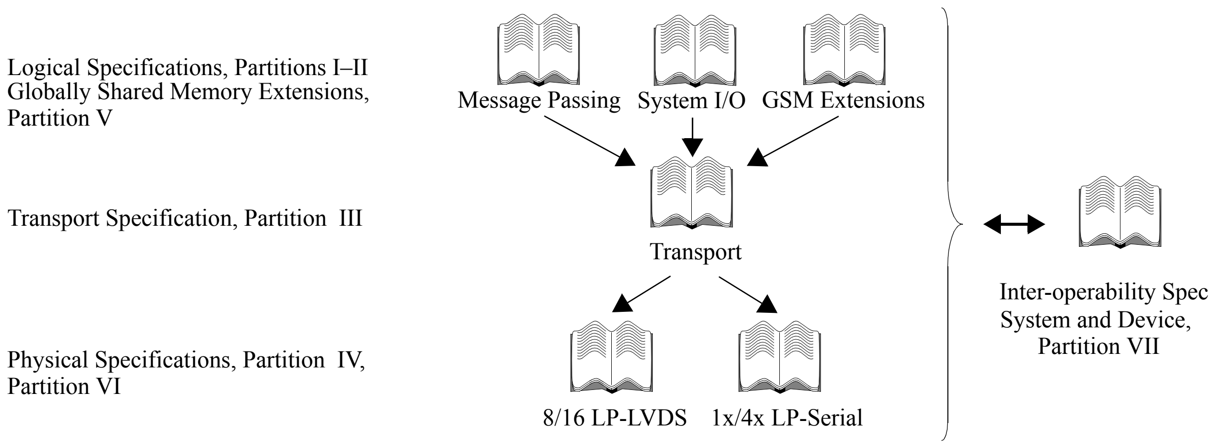

Logical Specifications, Partitions I–II
Globally Shared Memory Extensions, Partition V
Transport Specification, Partition III
Physical Specifications, Partition IV, Partition VI
Message Passing
System I/O
GSM Extensions
Transport
8/16 LP-LVDS
1x/4x LP-Serial
Inter-operability Spec System and Device, Partition VII

Figure 1-1. RapidIO Documentation Hierarchy

Each chapter addresses a different design topic. This revision of the *RapidIO System and Device Inter-operability Specification* document covers the following issues:

Chapter 2, “System Exploration and Initialization”

Chapter 3, “8/16 LP-LVDS Device Class Requirements”

Chapter 4, “PCI Considerations”

Chapter 5, “Globally Shared Memory Devices”

2 Chapter 2- System Exploration and Initialization

There are several basic ways of exploring and initializing a RapidIO system. The simplest method is to somehow define the power-up state of the system components such that all devices have adequate knowledge of the rest of the system to communicate as needed. This is frequently accomplished by shifting initialization information into all of the devices in the machine at boot time from serial ROMs or similar devices. This method is most applicable for relatively static systems and systems where boot-up time is important. A second method, having processors explore and configure the system at boot time, requires more time but is much more flexible in order to support relatively fast changing plug-and-play or hot-swap systems. This document describes a simple form of this second method.

2.1 Boot code access

In most RapidIO applications system initialization requires software for exploring and initializing devices. This is typically done by a processor or set of processors in the system. The boot code for the processor(s) may reside in a ROM local to the processor(s) or on a remote RapidIO agent device. A method of accessing the boot code through an uninitialized system is required if the boot code is located on a remote RapidIO agent device.

After resetting, a processor typically vectors to a fixed address and issues a code fetch. The agent hardware between the processor and the RapidIO fabric is required to take this read request and map it automatically to a NREAD transaction. The transaction is also mapped to a dedicated device ID at the proper address offset to find the boot code. All devices between the processor and the agent device where the boot ROM resides shall default to a state that will route the NREAD transaction to the boot ROM device and route the response back to the processor. The device ID for the agent device where the boot ROM resides is device ID=0xFE (0x00FE for 16-bit device IDs). The processor default device IDs are assigned sequentially starting at 0x00 (0x0000 for 16-bit device IDs).

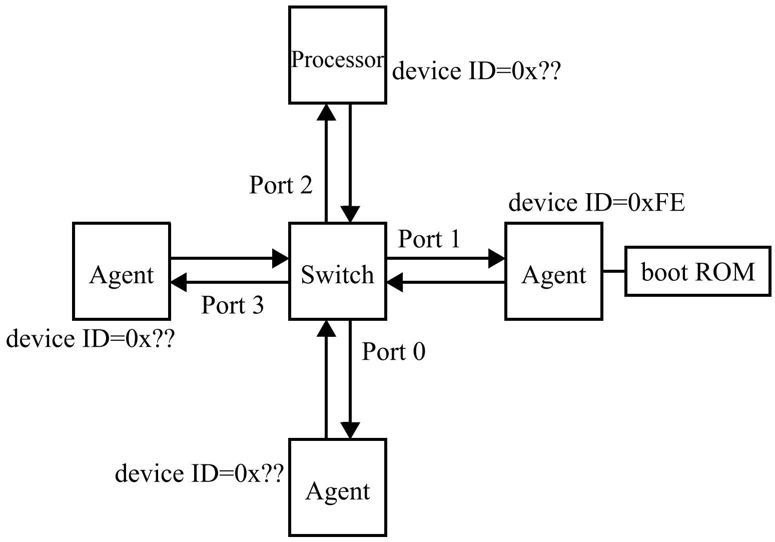


Figure 2-1. Example system with boot ROM

Figure 2-2 shows an example system with the boot ROM residing on an Agent device. The default routing state for the switch device between the processor and the agent shall allow all requests to device ID=0xFE to get to the agent device and all response packets to get from the agent device back to the processor. This means that the switch may also have to know the device ID that the processor will be using while fetching boot code (processor device IDs are assigned starting at 0x00 as described above). For the example in Figure , the system processor defaults to device ID=0x00, and the switch's default state routes device ID=0x00 to port 2.

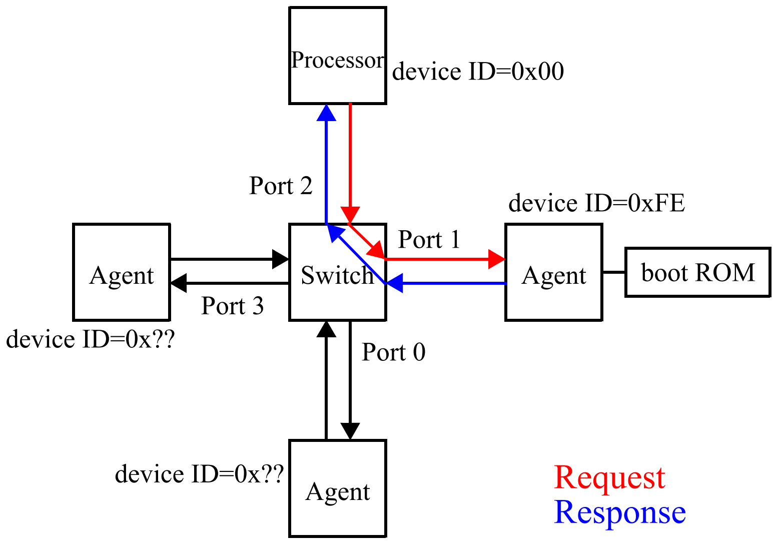


Figure 2-2. Automatically finding the boot ROM

Once the processor is able to begin running boot code, it can begin executing the exploration and initialization of the rest of the system.

2.2 Exploration and initialization

This example algorithm addresses the simple case of a system with a single processor that is responsible for exploring and initializing a system, termed a Host. The exploration and initialization process starts with a number of rules that the component and system designers shall follow.

2.2.1 Exploration and initialization rules

1. A Host shall be able to “reach” all agent devices that it is to be responsible for. This may require mechanisms to generate third party transactions to reach devices that are not transparently visible.
2. Maintenance responses generated by agent and switch devices shall be sent to the port that the maintenance request was received on. For example, consider a device that implements a 5 port switch. The system Host issues a maintenance read request to the switch device, which is received on input port 3. The switch, upon generating the maintenance response to the maintenance read request, must route it to output port 3 even though the switch may have been configured by default to route the response to a port other than port 3 (when the switch is configured it should also route the response to port 3).
3. All devices have CSRs to assist with exploration and initialization procedures. The registers used in this example contain the following information:

   — Base device ID register - This is the default device ID for the device, and it resides in a standard register in the CSR space at offset 0x60. At power-up, the base device ID defaults to logic 0xFF for all agent devices (0xFFFF for 16-bit route fields), with the exception of the boot code device and the Host device. The boot code device (if present) will have it’s device ID default to 0xFE and the Host device will have it’s device ID default to 0x00 as described in Section : "2.1 Boot code access". A device may have multiple device IDs, but only this architecturally defined device ID is used in the exploration and initialization procedure.

   — Master Enable bit - the Master Enable bit is reset at power-up for agent devices and set for Host devices. The Master Enable bit is located in the Physical Layer 8/16 LP-LVDS CSR at block offset 0x3C. If the Master Enable bit is clear the agent device is not allowed to issue requests and is only able to respond to received requests. This bit is used by the system Host to control when agents are allowed to issue transactions into the system. Switches are by default enabled and do not have a Master Enable bit.

   — Discovered bit - the Discovered bit is reset at power-up for agent devices and set for the Host device, and is located in the Physical Layer 8/16 LP-LVDS Port General Control CSR at block offset 0x3C. The system Host device sets this bit when the device has been discovered through the exploration mechanism. The Discovered bit is useful for detecting routing loops, and for hot plug or swap environments.

2.2.2 Exploration and initialization algorithm

If the above rules are followed, all agent devices are now accessible either as an end point that responds to any maintenance transaction or, for switches, via the hop_count mechanism.

The basic algorithm is to explore the system through each end point in sequence by first locating the adjacent device by sending a maintenance read to device ID=0xFF and hop count= 0x00, which is guaranteed to cause the adjacent device to respond. That device is then configured to reach the next device by assigning it a unique base device ID other than 0xFF, setting up route tables to reach the next device, etc.

When all devices in the system have been identified and have unique base device IDs assigned (no devices have a base device ID value=0xFF), the Host can then complete the final device ID assignment and configuration required for the application and enable agent devices to issue requests.

2.2.3 Exploration and initialization example

Figure 2-3 shows the previous example of a small single Host system.

Following the rules defined above, the base device ID value for all devices except the Host and boot ROM device after reset is applied is 0xFF, the Host has it’s Master Enable and Discovered bits set, and the agent devices have their Master Enable and Discovered bits cleared.

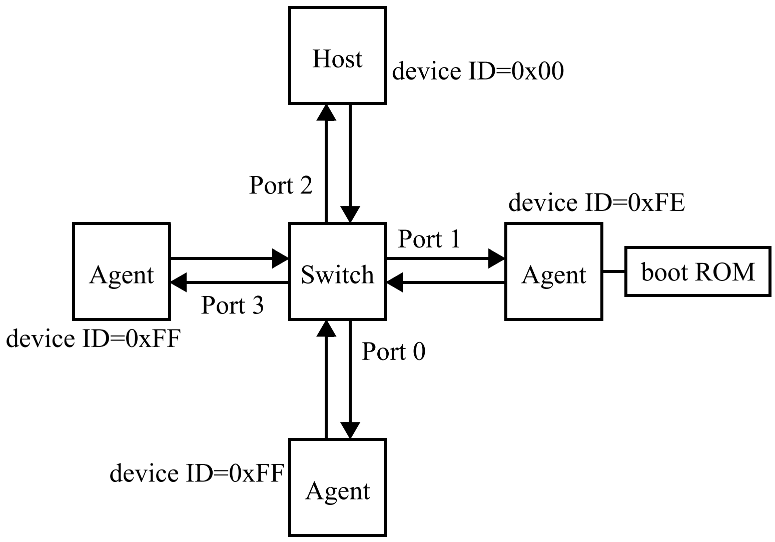
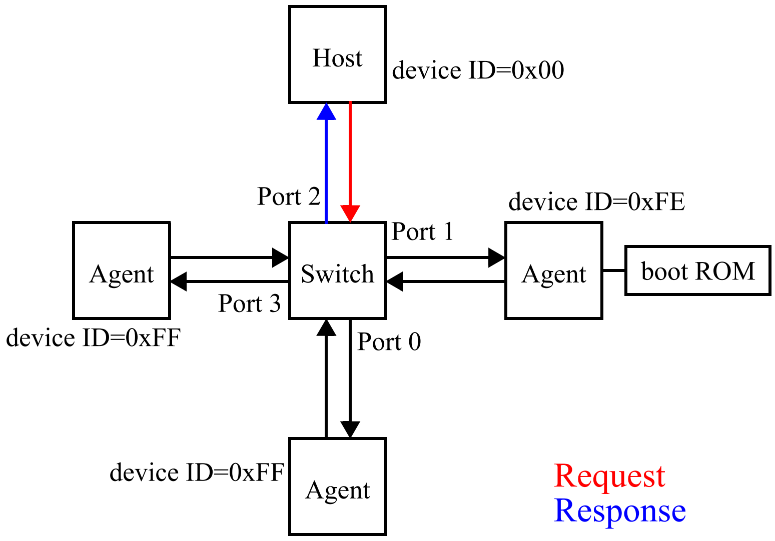


Figure 2-3. Example system

Assigning the Host's base device ID=0x00 is the first step in the process. The next step is to find the adjacent device, so the Host sends a maintenance read of offset 0x00_0000 to device ID=0xFF and hop_count=0x00. The switch consumes the request because the hop_count field is equal to zero and responds by sending the contents of it's Device Identity and Information CARs back to the port the request came from. From the returned information, the software on the Host can identify this as a switch. The Host then reads the switch port information CAR at offset 0x00_0014 to find out which port it is connected to. The response indicates a 4 port switch (which the Host may have already known from the device information register), connected to port 2.

The Host then examines the default routing tables for the switch to find the port route for the boot device ID=0xFE so it can preserve the path to the boot code (which it may still be running), and discovers that the boot device is located through port 1 of the switch. It also sets the switch's Discovered bit.



Figutre 2-4. Finding the adjacent device

The next step is for the Host to configure the switch to route device ID=0xFF to port 0 and device ID=0x00 to port 2 (which it already was because of the boot device in the system) via maintenance write requests to hop_count=0x00. The Host then issues another maintenance read request, this time to device ID=0xFF and hop_count=0x01. The switch discovers that it is not the final destination of the maintenance request packet, so it decrements the hop_count and routes the packet to port 0 and on to the attached agent device. The agent device responds, and the switch routes the response packet to device ID=0x00 back through port 2 to the Host. Again, software identifies the device, sets its Discovered bit, configures it as required, and assigns the base device ID=0x01.

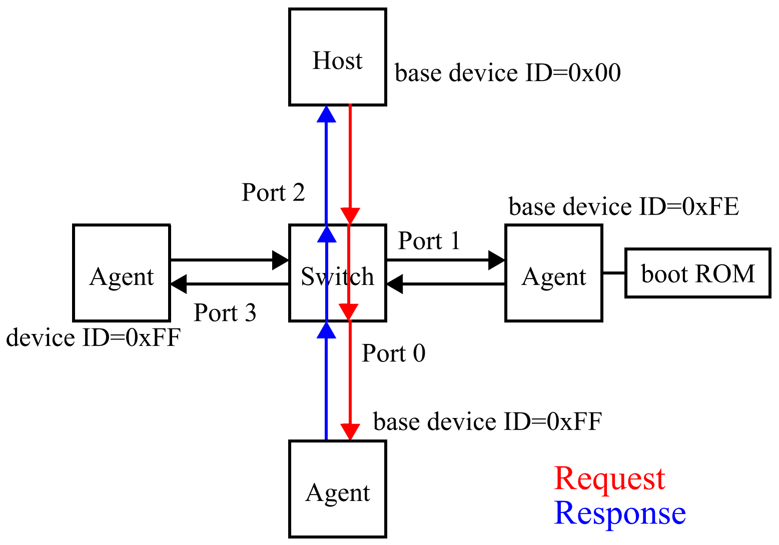


Figure 2-5. Finding the device on switch port 0

The Host then modifies the routing tables to now route device ID=0x01 to port 0. Since the boot device is located through port 1, instead of modifying the routing tables to route device ID=0xFF to port 1, the Host issues a maintenance read of device ID=0xFE (the boot device) and hop_count=0x01. The response identifies the agent on port 1, sets the agent's Discovered bit, and configures it as necessary, leaving the base device ID=0xFE so the Host can continue to execute the boot code.

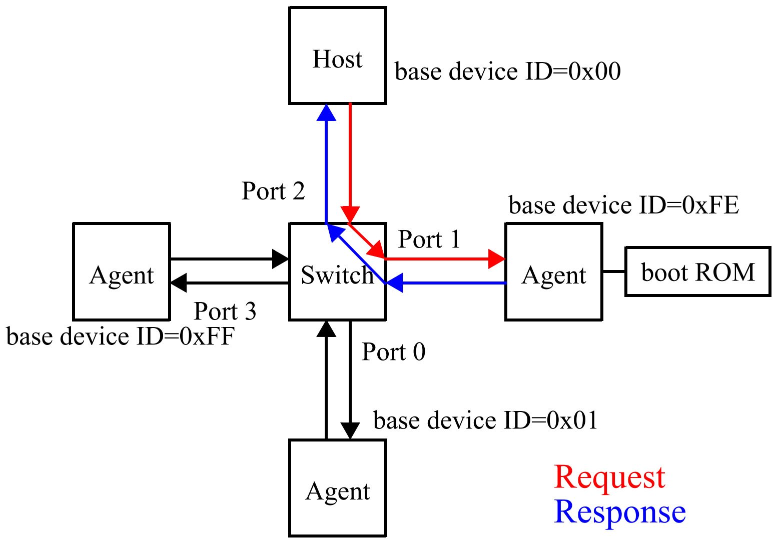


Figure 2-6. Finding the device on switch port 1

For the next iteration, the Host sets the switch device routing table entry for device ID=0xFF to route to port 3 (the Host already knows it is directly connected to port 2), and issues the maintenance read transaction as before.

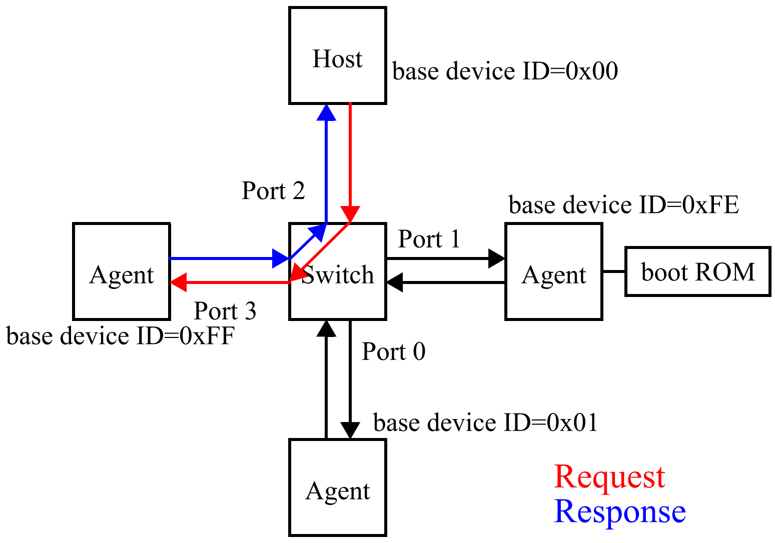


Figure 2-7. Finding the device on switch port 3

When the end point only agent responds with the requested CAR information the Host now knows that exploration is completed (there are no other paths to follow through the fabric), and can finalize configuring the system as shown in Figure 4-8. The agent devices can then have their Master Enable bits set so they can begin to issue transactions into the initialized system. The boot device ID can be changed, if desired, when the Host completes executing code from the boot ROM.

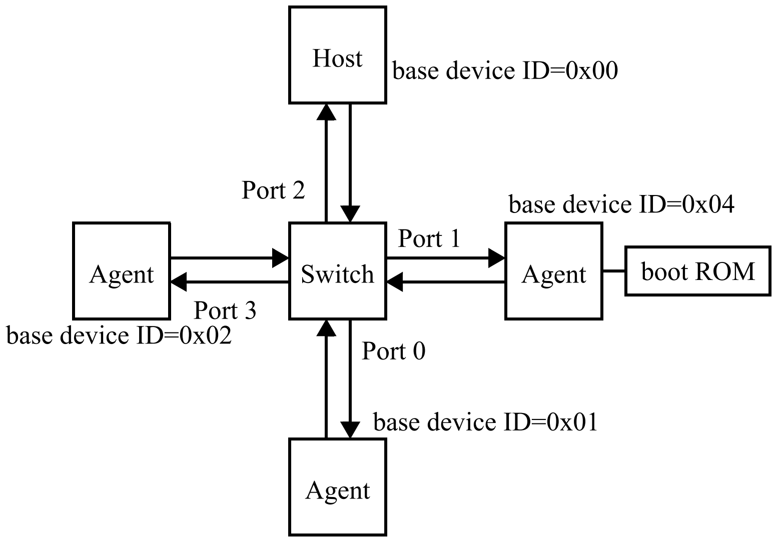


Figure 2-8. Final initialized system state

Variants to this procedure may be desirable. For example, a system may wish to enable some devices before exploration has been completed.

More complex systems with multiple Hosts, failed Host recovery, and hot swap requirements can be addressed with more complex algorithms utilizing the Host base device ID Lock Register and the Component Tag Register in standard registers in the CSR space at offsets 0x68 and 0x6C.

3 Chapter 3 - 8/16 LP-LVDS Device Class Requirements

3.1 Introduction

The RapidIO Architecture specifications allow for a variety of implementations. In order to form standard points of support for RapidIO, this chapter describes the requirements for RapidIO devices adhering to the 8/16 LP-LVDS physical layer specification and corresponding to different measures of functionality. Three device "classes" are defined, each with a minimum defined measure of support. The first class defines the functionality of the least capable device, with subsequent classes expanding the measure of support, in order to establish levels of inter-operability.

3.2 Class Partitioning

Each class includes the functionality defined in all previous class devices and defines the minimum additional functionality for that class. A device is not required to comply *exactly* with a class, but may optionally supply additional features as a value-add for that device. All functions that are not required in any class list are also optional value-adds for a device.

First is a set of requirements that are applicable to all RapidIO compliant devices, including switch devices without end point functionality.

3.2.1 Generic: All devices

3.2.1.1 General requirements

•8 bit wide port

—(refer to *Partition IV: PhysicalLayer 8/16 LP-LVDS Specification*, Section IV.1)

•Support for small (8-bit) transport device ID fields

—(refer to *Partition III: Common Transport Specification*, Section 1.3)

•Support for recovery from a single corrupt packet or control symbol

—(refer to *Partition IV: Physical Layer 8/16 LP-LVDS Specification*, Section 1.3.5)

•Support for packet retry protocol

—(refer to *Partition IV: Physical Layer 8/16 LP-LVDS Specification*, Section 1.2.4)

•Support for throttle based flow control

—(refer to *Partition IV: Physical Layer 8/16 LP-LVDS Specification*, Section 2.3)

•Support for transaction ordering at flow level 1

—(end point programmability for all flow levels is recommended)

—(refer to *Partition IV: Physical Layer 8/16 LP-LVDS Specification*, Section 1.2.2)

•Switch devices maintain error coverage internally

—(refer to *Partition IV: Physical Layer 8/16 LP-LVDS Specification*, Section 1.3.6)

•Support for maximum size (276 byte) packets for switch devices

—(refer to *Partition IV: Physical Layer 8/16 LP-LVDS Specification*, Section 1.3.9)

•Support for maximum size (256 byte) data payloads for end point devices

—(refer to *Partition I: Input/Output Logical Specification*)

•Device must contain the following registers:

– Device Identity CAR
– Device Information CAR
– Assembly Identity CAR
– Assembly Information CAR
– Processing Element Features CAR
– Source Operations CAR
– Destination Operations CAR

— (refer to *Partition I: Input/Output Logical Specification*, Section 4.4)

3.2.1.2 Operation support as target

•Maintenance read

—(switch targeted by hop_count transport field)

—(refer to *Partition I: Input/Output Logical Specification*, Section 2.3.1, Section 3.1.10)

•Maintenance write

—(switch targeted by hop_count transport field)

—(refer to *Partition I: Input/Output Logical Specification*, Section 2.3.1, Section 3.1.10)

3.2.1.3 Operation support as source

- <none>

3.2.2 Class 1: Simple target device

3.2.2.1 General requirements

- all Generic requirements
- Support for 34-bit address packet formats

  —(refer to *Partition I: Input/Output Logical Specification*, Section 4.4.5)

3.2.2.2 Operation support as target

- all Generic requirements
- Write

  —(refer to *Partition I: Input/Output Logical Specification*, Section 2.2.2, Section 3.1.7)
- Streaming-write

  —(refer to *Partition I: Input/Output Logical Specification*, Section 2.2.2, Section 3.1.8)
- Write-with-response

  —(refer to *Partition I: Input/Output Logical Specification*, Section 2.2.3, Section 3.1.7)
- Read

  —(refer to *Partition I: Input/Output Logical Specification*, Section 2.2.1, Section 3.1.5)

3.2.2.3 Operation support as source

- all Generic requirements

3.2.3 Class 2: Simple mastering device

3.2.3.1 General requirements

- all Class 1 requirements

3.2.3.2 Operation support as target

- all Class 1 requirements

3.2.3.3 Operation support as source

- all Class 1 requirements
- Maintenance read

  —(refer to *Partition I: Input/Output Logical Specification*, Section 2.3.1, Section 3.1.10)
- Maintenance write

  —(refer to *Partition I: Input/Output Logical Specification*, Section 2.3.1, Section 3.1.10)
- Write

  —(refer to *Partition I: Input/Output Logical Specification*, Section 2.2.2, Section 3.1.7)
- Write-with-response

  —(refer to *Partition I: Input/Output Logical Specification*, Section 2.2.3, Section 3.1.7)
- Streaming-write

  —(refer to *Partition I: Input/Output Logical Specification*, Section 2.2.2, Section 3.1.8)
- Read

  —(refer to *Partition I: Input/Output Logical Specification*, Section 2.2.1, Section 3.1.5)

3.2.4 Class 3: Complex mastering device

3.2.4.1 General requirements

- all Class 2 requirements

3.2.4.2 Operation support as target

- all Class 2 requirements
- Atomic set

  —(refer to *Partition I: Input/Output Logical Specification*, Section 2.2.4, Section 3.1.7)
- Maintenance port-write

  —(refer to *Partition I: Input/Output Logical Specification*, Section 2.3.1, Section 3.1.10)
- Data message mailbox 0

—(refer to *Partition II: Message Passing Logical Specification*, Section 2.2.2, Section 3.1.8)

3.2.4.3 Operation support as source

- all Class 2 requirements
- Atomic set

  —(refer to *Partition I: Input/Output Logical Specification*, Section 2.2.4, Section 3.1.7)
- Data message to mailbox 0

  —(refer to *Partition II: Message Passing Logical Specification*, Section 2.2.2, Section 3.1.8)

4Chapter 4- PCI Considerations

RapidIO contains a rich enough set of operations and capabilities to allow transport of legacy interconnects such as PCI[1]. While RapidIO and PCI share similar functionality, the two interconnects have different protocols thus requiring a translation function to move transactions between them. A RapidIO to PCI bridge processing element is required to make the necessary translation between the two interconnects. This chapter describes architectural considerations for an implementation of a RapidIO to PCI bridge processing element. This chapter is not intended as an implementation instruction manual, rather, it is to provide direction to the bridge processing element architect and aid in the development of interoperable devices. For this chapter it is assumed that the reader has a thorough understanding of the PCI 2.2 and/or the PCI-X 1.0 specifications.

Figure shows a typical system with devices connected using various RapidIO and PCI bus segments. A host bridge is connected to various peripherals via a PCI bus. A RapidIO bridge is used to translate PCI formatted transactions to the equivalent RapidIO operations to allow access to the rest of the system, including additional subordinate PCI bus segments.

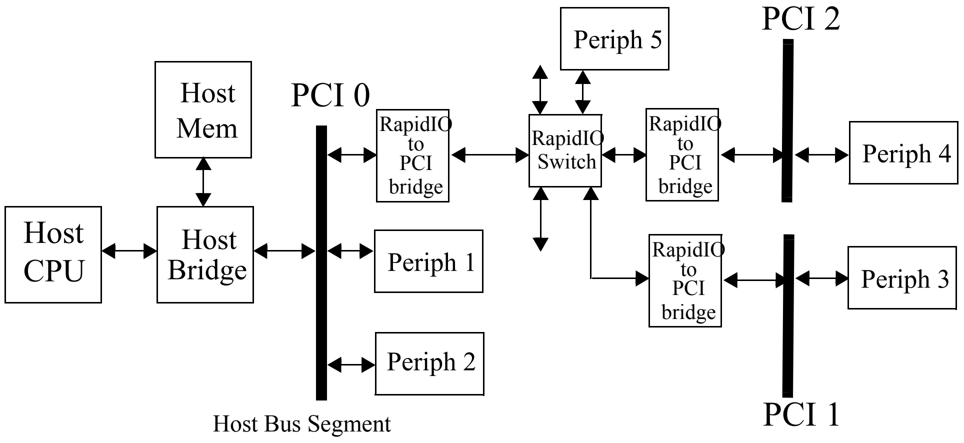


Figure 4-1. Example System with PCI and RapidIO

Where RapidIO is introduced into a legacy system, it is desirable to limit changes to software. For transactions which must travel between RapidIO and PCI it is necessary to map address spaces defined on the PCI bus to those of RapidIO, translate PCI transaction types to RapidIO operations, and maintain the producer/consumer requirements of the PCI bus. This chapter will address each of these considerations for both PCI version 2.2 and PCI-X.

4 Chapter 4 - PCI Considerations

4.1 Address Map Considerations

PCI defines three physical address spaces, specifically, the memory, I/O memory, and configuration spaces. RapidIO, on the other hand, only addresses memory and configuration space. This section discusses memory space. Configuration space is discussed in Section : "6.3 RapidIO to PCI Transaction Mapping". Figure shows a simple example of the PCI memory and I/O address spaces for a host bus segment. In order for devices on the PCI bus to communicate with those connected through RapidIO, it is necessary to provide a memory mapping function. The example PCI host memory map

1.For additional information on the Peripheral Component Interconnect PCI refer to the PCI 2.2 and the PCI-X 1.0 specifications.

uses a 32-bit physical address space resulting in 4 Gbytes of total address space. Host memory is shown at the bottom of the address map and peripheral devices at the top. Consider that the RapidIO to PCI bridge processing element contains a specified window(s) of address space mapped to it using the PCI base address register(s)[1]. The example shown in Figure illustrates the RapidIO bridge address window located in an arbitrary software defined location. Likewise, if it was desired to communicate with PCI legacy I/O devices over RapidIO an I/O window would be assigned to the RapidIO to PCI bridge as shown.

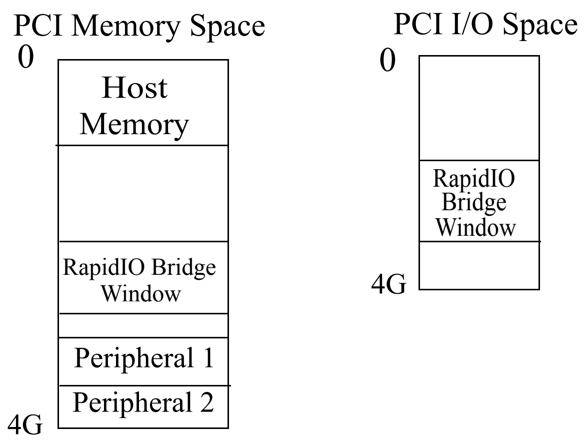


Figure 4-2. Host segment PCI Memory Map Example

Any transactions issued to the bus segment with an address that matches the RapidIO bridge window will be captured by the RapidIO to PCI bridge for forwarding. Once the transaction has been accepted by the RapidIO to PCI bridge processing element it must be translated to the proper RapidIO context as shown in Figure 1-2. For the purposes of this discussion this function is called the Address Mapping and Translation function (AMT). The AMT function is responsible for translating PCI addresses to RapidIO addresses as well as the translation and assignment of the respective PCI and RapidIO transaction types. The address space defined by the RapidIO bridge window may represent more than one subordinate RapidIO target device. A device on PCI bus segment 0 shown in Figure may require access to a peripheral on PCI bus 1, bus 2, or RapidIO Peripheral 5. Because RapidIO uses source addressing (device IDs), the AMT is responsible for translating the PCI address to both a target device ID and associated offset address. In addition to address translation, RapidIO attributes, transaction types, and other necessary delivery information are established.

Similarly, transactions traveling from a RapidIO bus to a PCI bus must also pass through the AMT function. The address and transaction type are translated back into PCI format, and the AMT selects the appropriate address for the transaction. Memory mapping is relied upon for all transactions bridged between PCI and RapidIO.

1.Refer to the PCI 2.2 Specification Chapter 6 for a discussion on PCI address maps and configuration registers

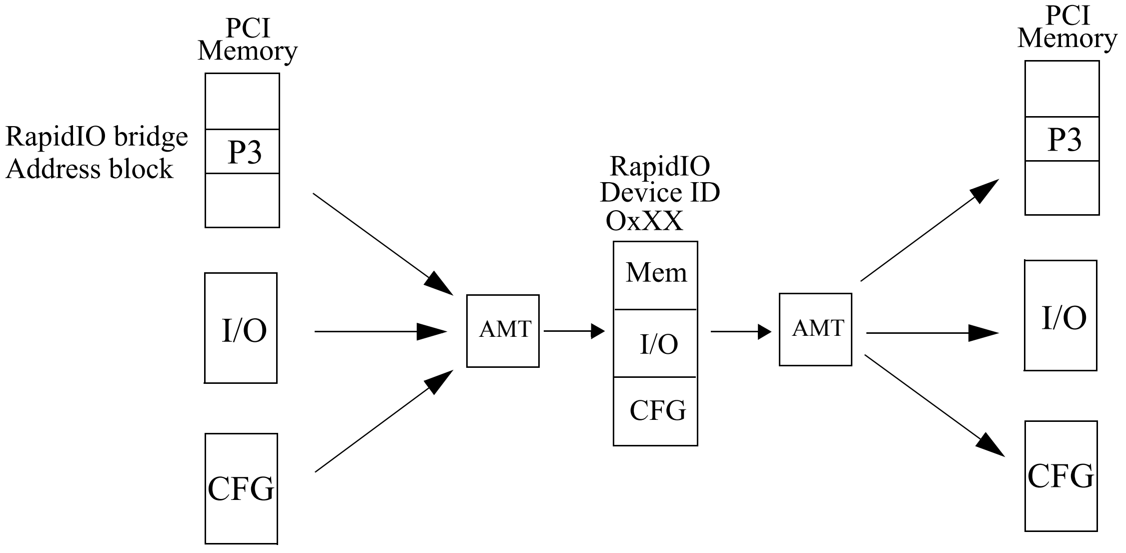


Figure 4-3. AMT and Memory Mapping

4.2 Transaction Flow

In considering the mapping of the PCI bus to RapidIO it is important to understand the transaction flow of PCI transactions through RapidIO.

4.2.1 PCI 2.2 Transaction Flow

The PCI 2.2 specification defines two classes of transaction types, posted and non-posted. Figure shows the route taken by a PCI-RapidIO posted write transaction. Once the request is sent from the PCI Master on the bus, it is claimed by the bridge processing element which uses the AMT to translate it into a RapidIO request. Only when the transaction is in RapidIO format can it be posted to the RapidIO target. In some cases it may be desirable to guarantee end to end delivery of the posted write transaction. For this case the RapidIO NWRITE_R transaction is used which results in a response as shown in the figure 4-4.

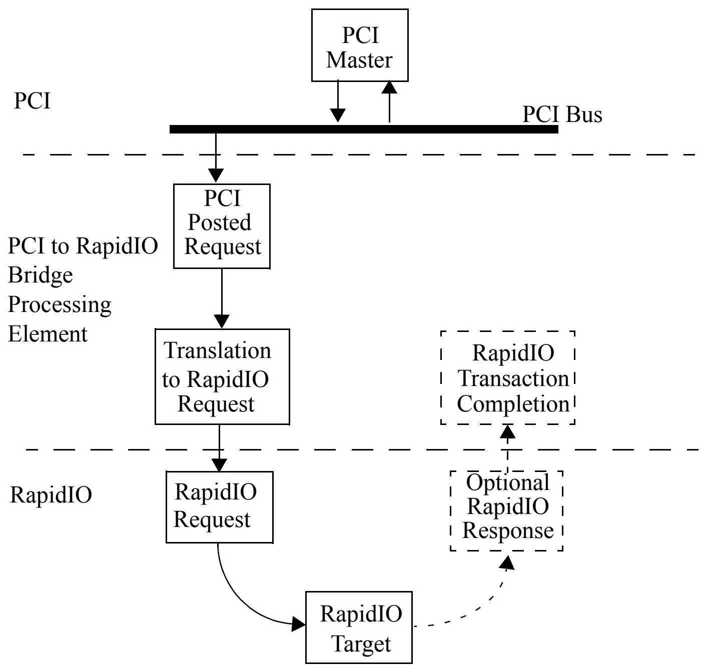

PCI Master
PCI
PCI Bus
PCI Posted Request
PCI to RapidIO Bridge Processing Element
Translation to RapidIO Request
RapidIO Transaction Completion
RapidIO
RapidIO Request
Optional RapidIO Response
RapidIO Target

Figure 4-4. PCI Mastered Posted Write Transaction Flow Diagram

A non-posted PCI transaction is shown in Figure . The transaction is mastered by the PCI agent on the PCI bus and accepted by the RapidIO to PCI bridge. The transaction is retried on the PCI bus if the bridge is unable to complete it within the required time-out period. In this case the transaction is completed as a delayed transaction. The transaction is translated to the appropriate RapidIO operation and issued on the RapidIO port. At some time later a RapidIO response is received and the results are translated back to PCI format. When the PCI master subsequently retries the transaction, the delayed results are returned and the operation is completed.

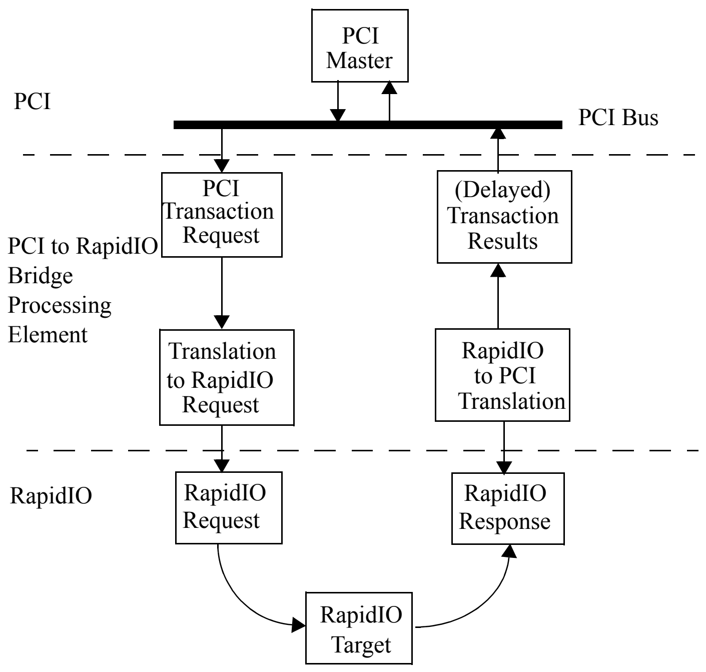

PCI Master
PCI
PCI Bus
PCI Transaction Request
(Delayed) Transaction Results
PCI to RapidIO Bridge Processing Element
Translation to RapidIO Request
RapidIO to PCI Translation
RapidIO
RapidIO Request
RapidIO Response
RapidIO Target

Figure 4-5. PCI Mastered non-posted (delayed) Transaction Flow Diagram

Because PCI allows unbounded transaction data tenures, it may be necessary for the RapidIO to PCI bridge to break the single PCI transaction into multiple RapidIO operations. In addition, RapidIO does not have byte enables and therefore does not support sparse byte transactions. For this case the transaction must be broken into multiple operations as well. “Section , 6.6 Byte Lane and Byte Enable Usage” on page 372 describes this situation in more detail.

A RapidIO mastered operation is shown in Figure . For this case the RapidIO request transaction is received at the RapidIO to PCI bridge. The bridge translates the request into the appropriate PCI command which is then issued to the PCI bus. The PCI target may complete the transaction as a posted, non-posted, or delayed non-posted transaction depending on the command type. Once the command is successfully completed on the PCI bus the results are translated back into the RapidIO format and a response transaction is issued back to the RapidIO Master.

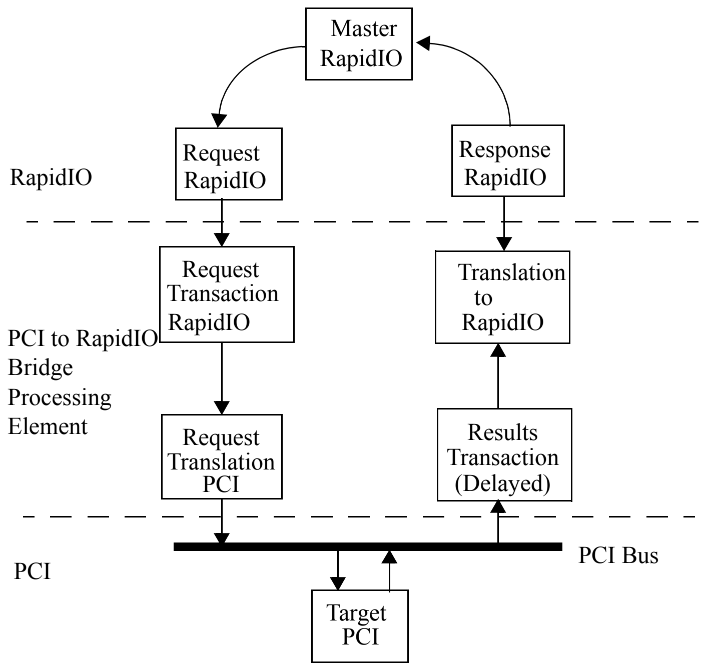


Figure 4-6. RapidIO Mastered Transaction

4.2.2 PCI-X Transaction Flow

The flow of transactions described in the previous section applies to the PCI-X bus as well. PCI-X supports split transactions instead of delayed transactions. The example shown in Figure illustrates a transaction completed with a PCI-X split completion. The PCI-X master issues a transaction. The RapidIO to PCI-X bridge determines that it must complete the transaction as a split transaction, and responds with a split response. The transaction is translated to RapidIO and a request is issued on the RapidIO port. The RapidIO target returns a response transaction which is translated to a PCI-X Split Completion transaction completing the operation. PCI-X allows up to a 4 Kilobyte request. Larger PCI-X requests must be broken into multiple RapidIO operations. The RapidIO to PCI-X bridge may return the results back to the PCI-X Master using multiple Split Completion transactions in a pipelined fashion. Since PCI-X only allows devices to disconnect on 128 byte boundaries it is advantageous to break the large PCI-X request into either 128 or 256 byte RapidIO operations.

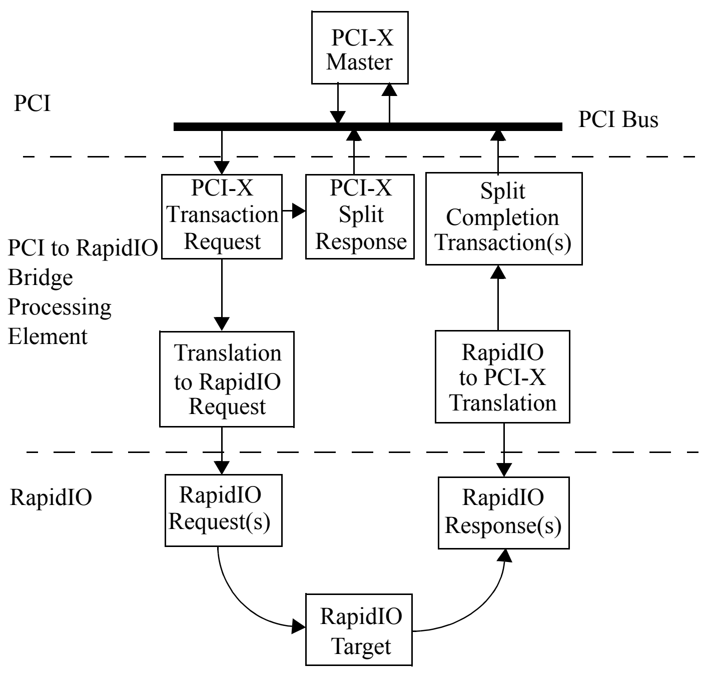

PCI-X Master
PCI
PCI Bus
PCI-X Transaction Request
PCI-X Split Response
Split Completion Transaction(s)
PCI to RapidIO Bridge Processing Element
Translation to RapidIO Request
RapidIO to PCI-X Translation
RapidIO
RapidIO Request(s)
RapidIO Response(s)
RapidIO Target

Figure 6-7. PCI-X Mastered Split Response Transaction

4.3 RapidIO to PCI Transaction Mapping

The RapidIO Logical Input/Output and Globally Shared Memory specifications include the necessary transactions types to map all PCI transactions. Table lists the map of transactions between PCI and RapidIO. A mapping mechanism such as the AMT function described in Section : "6.1 Address Map Considerations" is necessary to assign the proper transaction type based on the address space for which the transaction is targeted.

Table 4-1. PCI 2.2 to RapidIO Transaction Mapping

PCI Command	**RapidIO Transaction**	**Comment**
Interrupt-acknowledge	NREAD	
Special-cycle	NWRITE	
I/O-read	NREAD	
I/O-write	NWRITE_R	
Memory-read, Memory-Read-Line, Memory-Read-Multiple	NREAD or IO_READ_HOME	The PCI memory read transactions can be represented by the NREAD operation. If the operation is targeted to hardware maintained globally coherent memory address space then the I/O Read operation must be used (see “Section , 4.5 Interactions with Globally Shared Memory” on page 365.)
Memory-write, Memory-write-and-invalidate	NWRITE, NWRITE_R, or FLUSH	The PCI Memory Write and Memory-Write-and-Invalidate can be represented by the NWRITE operation. If reliable delivery of an individual write transaction is desired then the NWRITE_R is used. If the operation is targeted to hardware maintained globally coherent memory address space then the Data Cache Flush operation must be used (refer to “Section , 4.5 Interactions with Globally Shared Memory” on page 365.)
Configuration-read	NREAD	
Configuration-write	NWRITE_R	

PCI 2.2 memory transactions do not specify a size. It is possible for a PCI master to read a continuous stream of data from a target or to write a continuous stream of data to a target. Because RapidIO is defined to have a maximum data payload of 256 bytes, PCI transactions that are longer than 256 bytes must be broken into multiple RapidIO operations.

Table-4-2 shows the transaction mapping between PCI-X and RapidIO.

Table 4-2. PCI-X to RapidIO Transaction Mapping

PCI-X Command	**RapidIO Transaction**	**Comment**
Interrupt-acknowledge	NREAD	
Special-cycle	NWRITE	
I/O-read	NREAD	
I/O-write	NWRITE_R	
Memory-read DWORD	NREAD or IO_READ_HOME	The PCI-X memory read DWORD transactions can be represented by the NREAD operation. If the operation is targeted to hardware maintained coherent memory address space then the I/O Read operation must be used (refer to "Section , 4.5 Interactions with Globally Shared Memory" on page 365.) This is indicated in PCI-X using the No Snoop (NS) bit described in Section 2.5 of the PCI-X 1.0 specification.
Memory-write	NWRITE, NWRITE_R, or FLUSH	The PCI-X Memory Write and Memory-Write-and-Invalidate can be represented by the NWRITE operation. If reliable delivery of an individual write transaction is desired then the NWRITE_R is used. If the operation is targeted to hardware maintained coherent memory address space then the Data Cache Flush operation must be used (refer to "Section , 4.5 Interactions with Globally Shared Memory" on page 365.) This is indicated in PCI-X using the No Snoop (NS) bit described in Section 2.5 of the PCI-X 1.0 specification.
Configuration-read	NREAD	
Configuration-write	NWRITE_R	
Split Completion	--	The Split Completion transaction is the result of a request on the PCI-X bus that was terminated by the target with a Split Response. In the case of the RapidIO to PCI-X bridge this would be the artifact of a transaction that either the bridge mastered and received a split response or was the target and issued a split response. This command is equivalent to a RapidIO response transaction and does not traverse the bridge.

Table 4-2. PCI-X to RapidIO Transaction Mapping

PCI-X Command	RapidIO Transaction	Comment
Memory-read-block	NREAD or IO_READ_HOME	The PCI-X memory read transactions can be represented by the NREAD operation. If the operation is targeted to hardware maintained globally coherent memory address space then the I/O Read operation must be used (refer to “Section , 4.5 Interactions with Globally Shared Memory” on page 365.) This is indicated in PCI-X using the No Snoop (NS) bit described in Section 2.5 of the PCI-X 1.0 specification.
Memory-write-block	NWRITE, NWRITE_R, or FLUSH	The PCI-X Memory Write and Memory-Write-and-Invalidate can be represented by the NWRITE operation. If reliable delivery of an individual write transaction is desired then the NWRITE_R is used. If the operation is targeted to hardware maintained globally coherent memory address space then the Data Cache Flush operation must be used (refer to “Section , 4.5 Interactions with Globally Shared Memory” on page 365.) This is indicated in PCI-X using the No Snoop (NS) bit described in Section 2.5 of the PCI-X 1.0 specification.

The PCI-X addendum to the PCI specification adds the ability to do split operations. This results in an operation being broken into a Split Request and one or more Split Completions. As a target of a PCI-X Split Request, the RapidIO to PCI bridge may reply with a Split Response and complete the request using multiple RapidIO operations. The results of these operations are issued on the PCI-X bus as Split Completions. If the RapidIO to PCI-X bridge is the initiator of a Split Request, the target may also indicate that it intends to run the operation as a split transaction with a Split Response. In this case the target would send the results to the RapidIO to PCI-X bridge using Split Completions.

4.4 Operation Ordering and Transaction Delivery

This section discusses what the RapidIO to PCI bridge must do to address the requirements of the ordering rules of the PCI specifications.

4.4.1 Operation Ordering

Section 1.2.1 of the *RapidIO Input/Output Logical Specification* describes a set of ordering rules. The rules guarantee ordered delivery of write data and that results of read operations will contain any data that was previously written to the same location.

For bridge devices, the PCI 2.2 specification has the additional requirement that the results of a read command push ahead posted writes in both directions.

In order for the RapidIO to PCI bridge to be consistent with the PCI 2.2 ordering rules it is necessary to follow the transaction ordering rules listed in section 1.2.1 of the *RapidIO Input/Output Logical Specification*. In addition, the RapidIO to PCI bridge is required to adhere to the following RapidIO rule:

Read responses must push ahead all write requests and write responses.

4.4.2 Transaction Delivery Ordering

The *RapidIO Physical Layer 8/16 LP-LVDS Specification* describes the mechanisms by which transaction ordering and delivery occur through the system. When considering the requirements for the RapidIO to PCI bridge it is first necessary to follow the transaction delivery ordering rules in section 1.2.4.1 of the *RapidIO Physical Layer 8/16 LP-LVDS Specification*. Further, it is necessary to add additional constraints to maintain programming model compatibility with PCI.

As described in Section : "4.4.1 Operation Ordering" above, PCI has an additional transaction ordering requirement over RapidIO. In order to guarantee inter-operability, transaction ordering, and deadlock free operation, it is recommended that devices be restricted to utilizing transaction request flow level 0. In addition, it is recommended that response transactions follow a more strict priority assignment. Table illustrates the priority assignment requirements for transactions in the PCI to RapidIO environment.

Table 6-3. Packet priority assignments for PCI ordering

RapidIO packet type	priority	comment
read request	0	This will push write requests and responses ahead
write request	1	Forces writes to complete in order, but allows write requests to bypass read requests
read response	1	Will force completion of preceding write requests and allows bypass of read requests
write response	2	Will prevent NWRITE_R request based deadlocks

The PCI transaction ordering model requires that a RapidIO device not issue a read request into the system unless it has sufficient resources available to receive and process a higher priority write or response packet in order to prevent deadlock. PCI 2.2 states that read responses cannot pass write transactions. The RapidIO specification provides PCI ordering by issuing priority 0 to read requests, and priority 1 to read responses and PCI writes. Since read responses and writes are issued at the same priority, the read responses will not pass writes.

4.4.3 PCI-X Relaxed Ordering Considerations

The PCI-X specification defines an additional ordering feature called relaxed ordering. If the PCI-X relaxed ordering attribute is set for a read transaction, the results for the read transaction are allowed to pass posted write transactions. PCI-X read transactions with this bit set allow the PCI-X to RapidIO bridge to ignore the rule described in Section : "4.4.1 Operation Ordering". Table shows the results of this additional function.

Table 4-4. Packet priority assignments for PCI-X ordering

RapidIO packet type	priority	comment
read request	0	This will push write requests and responses ahead
write request	1	Forces writes to complete in order, but allows write requests to bypass of read requests
read response	1	When PCI-X Relaxed Ordering attribute is set to 0. Will force completion of preceding write requests and allows bypass of read requests
read response	2, 3	When PCI-X Relaxed Ordering attribute is set to 1. The endpoint may promote the read response to higher priority to allow it to move ahead of posted writes.
write response	2	

4.5 Interactions with Globally Shared Memory

Traditional systems have two notions of system or subsystem cache coherence. The first, non-coherent, means that memory accesses have no effect on the caches in the system. The memory controller reads and writes memory directly, and any cached address becomes incoherent in the system. This behavior requires that all cache coherence with I/O be managed using software mechanisms, as illustrated in Figure .

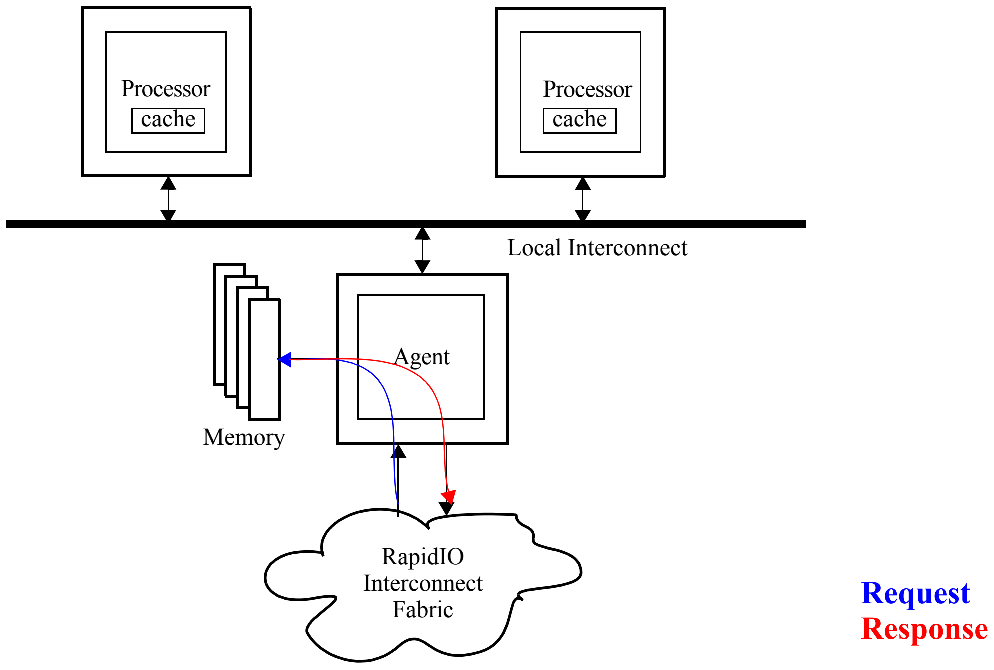


Figure 4-8. Traditional Non-coherent I/O Access Example

The second notion of system cache coherence is that of global coherence. An I/O access to memory causes a snoop cycle to be issued on the processor bus, keeping all of the system caches coherent with the memory, as illustrated in Figure 6-9.

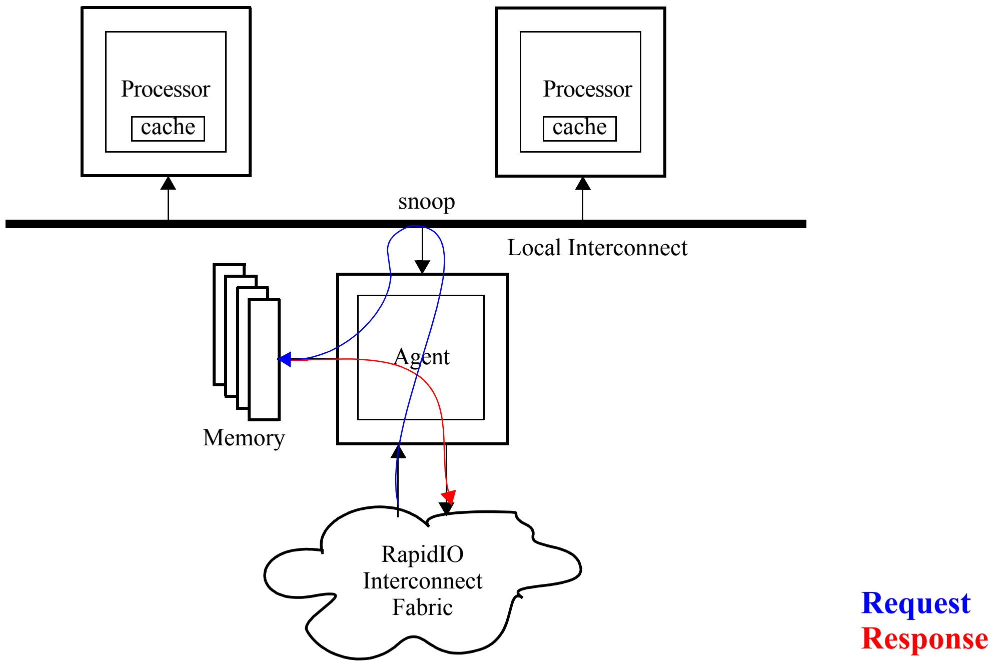


Figure 4-9. Traditional Globally Coherent I/O Access Example

With RapidIO globally shared systems, there is no common bus that can be used in order to issue the snoop, so global coherence requires special hardware support beyond simply snooping the bus. This leads to a third notion of cache coherence, termed local coherence. For local coherence, a snoop on a processor bus local to the targeted memory controller can be used to keep those caches coherent with that part of memory, but not caches associated with other memory controllers, as illustrated in Figure . Therefore, what once was regarded in a system as a "coherent access" is no longer globally coherent, but only locally coherent. Typically, deciding to snoop or not snoop the local processor caches is either determined by design or system architecture policy (always snoop or never snoop), or by an attribute associated with the physical address being accessed. In PCI-X, this attribute is the No Snoop (NS) bit described in Section 2.5 of the PCI-X 1.0 specification.

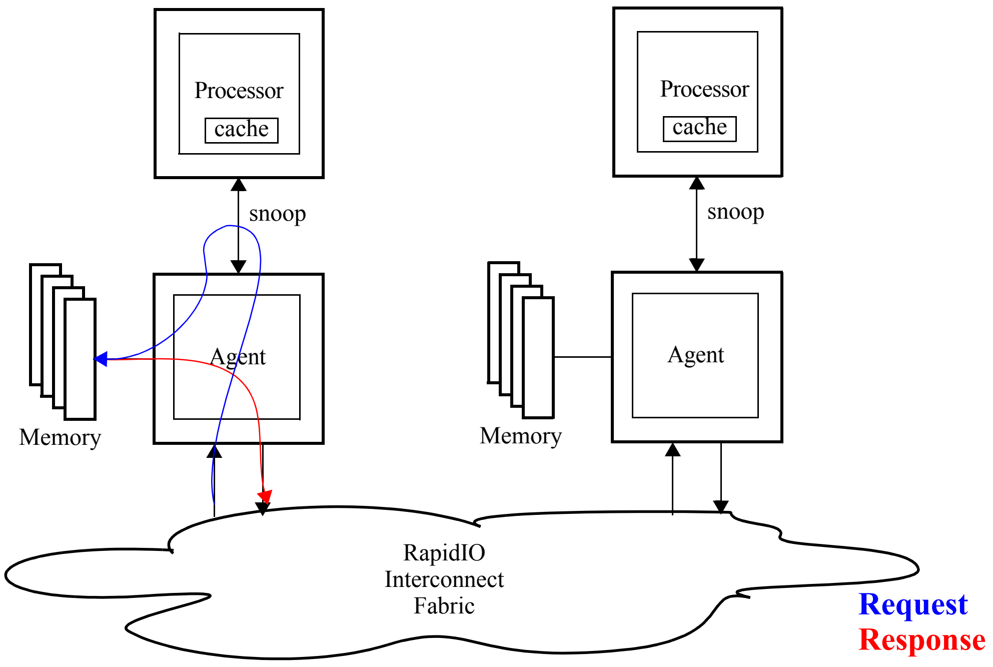


Figure 4-10. RapidIO Locally Coherent I/O Access Example

In order to preserve the concept of global cache coherence for a system, the *RapidIO Globally Shared Memory Logical Specification* defines several operations that allow a RapidIO to PCI bridge processing element to access data in the globally shared space without having to implement all of the cache coherence protocol. These operations are the I/O Read and Data Cache Flush operations (*RapidIO Globally Shared Memory Logical Specification*, sections 3.2.9 and 3.2.10). For PCI-X bridging, these operations can also be used as a way to encode the NO SNOOP attribute for locally as well as globally coherent transactions. The targeted memory controller can be designed to understand the required behavior of such a transaction. These encodings also are useful for tunneling PCI-X transactions between PCI-X bridge devices.

The data payload for an I/O Read operation is defined as the size of the coherence granule for the targeted globally shared memory domain. However, the Data Cache Flush operation allows coherence granule, sub-coherence granule, and sub-double-word writes to be performed.

The IO_READ_HOME transaction is used to indicate to the GSM memory controller that the memory access is globally coherent, so the memory controller finds the latest copy of the requested data within the coherence domain (the requesting RapidIO to PCI bridge processing element is, by definition, not in the coherence domain) without changing the state of the participant caches. Therefore, the I/O Read operation allows the RapidIO to PCI bridge to cleanly extract data from a coherent portion of the system with minimal disruption and without having to be a full participant in the coherence domain.

The Data Cache Flush operation has several uses in a coherent part of a system. One such use is to allow a RapidIO to PCI bridge processing element to write to globally shared portions of the system memory. Analogous to the IO_READ_HOME transaction, the FLUSH transaction is used to indicate to the GSM memory controller that the access is globally coherent. The memory controller forces all of the caches in the coherence domain to invalidate the coherence granule if they have a shared copy (or return the data to memory if one had ownership of the data), and then writes memory with the data supplied with the FLUSH request. This behavior allows the I/O device to cleanly write data to the globally shared address space without having to be a full participant in the coherence domain.

Since the RapidIO to PCI bridge processing element is not part of the coherence domain, it is never the target of a coherent operation.

4.5.1 I/O Read Operation Details

Most of the complexity of the I/O Read operation resides in the memory controller. For the RapidIO to PCI Bridge processing element the I/O Read operation requires some additional attention over the non-coherent read operation. The necessary portions of the I/O Read state machine description in Section 6.10 of the *RapidIO Globally Shared Memory Logical Specification* are extracted below. Refer to Chapter 6 of the GSM specification for state machine definitions and conventions. The GSM specification takes precedence in the case of any discrepancies between the corresponding portions of the GSM specification and this description.

4.5.1.1 Internal Request State Machine

This state machine handles requests to the remote globally shared memory space.

```
remote_request(IO_READ_HOME, mem_id, my_id);
```

4.5.1.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect.

```
switch(remote_response)
case DONE:
        return_data();
        free_entry();
case DONE_INTERVENTION:// must be from third party
        set_received_done_message();
        if (received_data_only_message)
              free_entry();
        else
              // wait for a DATA_ONLY
        endif;
case DATA_ONLY:              // this is due to an intervention, a
                             // DONE_INTERVENTION should come
                             // separately
        set_received_data_only_message();
        if (received_done_message)
              return_data();
              free_entry();
        else
              return_data();        // OK for weak ordering
        endif;
case RETRY:
        remote_request(IO_READ_HOME, received_srcid, my_id);
default
        error();
```

4.5.2 Data Cache Flush Operation Details

As with the I/O Read operation, the complexity for the Data Cache Flush operation resides in the memory controller. The necessary portions of the Data Cache Flush state machine description from Section 6.10 of the GSM logical specification are extracted below. Refer to Chapters 2 and 3 of the GSM specification to determine the size of data payloads for the FLUSH transaction. The GSM specification takes precedence in the case of any discrepancies between the corresponding portions of the GSM specification and this description.

4.5.2.1 Internal Request State Machine

This state machine handles requests to the remote globally shared memory space.

```
remote_request(FLUSH, mem_id, my_id, data);
```

4.5.2.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect.

```
switch (received_response)
case DONE:
        local_response(OK);
        free_entry();
case RETRY:
        remote_request(FLUSH, received_srcid, my_id, data);
default:
```

error();

4.6 Byte Lane and Byte Enable Usage

PCI makes use of byte enables and allows combining and merging of transactions. This may have the result of write transactions with sparse valid bytes. In order to save on transaction overhead, RapidIO does not include byte enables. RapidIO does, however, support a set of byte encodings defined in Chapter 3 of the *RapidIO Input/Output Logical Layer Specification*. PCI to RapidIO operations may be issued with sparse bytes. Should a PCI write transaction with byte enables that do not match a RapidIO byte encoding be issued to a RapidIO to PCI bridge, that operation must be broken into multiple valid RapidIO operations.

4.7 Error Management

Errors that are detected on a PCI bus are signaled using side band signals. The treatment of these signals is left to the system designer and is outside of the PCI specifications. Likewise, this document does not recommend any practices for the delivery of error interrupts in the system.

5 Chapter 5 - Globally Shared Memory Devices

Different processing elements have different requirements when participating in a RapidIO GSM environment. The GSM protocols and address collision tables are written from the point of view of a fully integrated processing element comprised of a local processor, a memory controller, and an I/O controller. Obviously, the complexity and implementation requirements for this assumed device are much greater than required for a typical design. This chapter assumes that the reader is familiar with the *RapidIO Globally Shared Memory Logical Specification*.

Additionally, this chapter contains the Physical Layer 8/16 LP-LVDS and 1x/4x LP-Serial transaction to priority mappings to guarantee that a system maintains cache coherence and is deadlock free.

5.1 Processing Element Behavior

In Chapter 2 of the *RapidIO Globally Shared Memory Logical Specification* are a number of examples of possible processing elements:

- A processor-memory processing element
- A memory-only processing element
- A processor-only processing element
- An I/O processing element
- A switch processing element

Of all of these, only the switch processing element does not have to implement anything additional to exist in a GSM system or sub-system. All of the remaining processing element types are of interest, and all are likely to exist in some form in the marketplace. This chapter is intended to define the portions of the protocol necessary to implement each of these devices. Other processing elements are allowed by the *RapidIO Globally Shared Memory Logical Specification*, for example, a memory-I/O processing element. The portions of the protocol necessary to implement these devices are not addressed in this chapter.

5.1.1 Processor-Memory Processing Element

This processing element is very nearly the same as the assumed processing element used for the state machine description in Chapter 6, and requires nearly all of the described functionality. The following operation behavior is not changed from the Chapter 6 descriptions:

- Read
- Instruction read
- Read for ownership
- Data cache and instruction cache invalidate
- Castout
- TLB invalidate entry and TLB invalidate entry synchronize
- Data cache flush

This leaves the I/O Read operation. Since the processor-memory processing element does not contain an I/O device, this processing element will not generate the I/O read operation, but is required to respond to it. This removes the internal request state machine and portions of the response state machine, requiring the behavior described in Section 2.1.1 below. The only exception to this is the special case where there exists multiple coherence domains. It is possible that a processor in one coherence domain may wish to read data in another coherence domain and thus would require support of the I/O Read operation.

5.1.1.1 I/O Read Operations

This operation is used for I/O reads of globally shared memory space.

5.1.1.1.1 Response State Machine

This machine handles responses to requests made to the RapidIO interconnect made on behalf of a third party.

```
switch(remote_response)
case INTERVENTION:
        update_memory();
        remote_response(DONE_INTERVENTION, original_srcid, my_id);
        free_entry();
case NOT_OWNER,             // data comes from memory, mimic
                            // intervention
case RETRY:
        switch(directory_state)
        case LOCAL_MODIFIED,
        case LOCAL_SHARED:
                            remote_response(DATA_ONLY, original_srcid, my_id,
                            data);
               remote_response(DONE_INTERVENTION, original_srcid,
                            my_id);
               free_entry();
        case REMOTE_MODIFIED:// spin or wait for castout
               remote_request(IO_READ_OWNER, received_srcid, my_id,
                            my_id);
        default:
               error();
default:
        error();
```

5.1.1.1.2 External Request State Machine

This machine handles requests from the system to the local memory or the local processor. This may require making further external requests.

```
if (address_collision)      // use collision tables in
                            // Chapter 7, “Address Collision Resolution Tables”
elseif (IO_READ_HOME)       // remote request to our local memory
        assign_entry();
        switch (directory_state)
        case LOCAL_MODIFIED:
               local_request(READ_LATEST);
               remote_response(DONE, received_srcid, my_id, data);
                            // after push completes
               free_entry();
        case LOCAL_SHARED:
               remote_response(DONE, received_srcid, my_id, data);
               free_entry();
        case REMOTE_MODIFIED:
               remote_request(IO_READ_OWNER, mask_id, my_id, received_srcid);
        case SHARED:
               remote_response(DONE, received_srcid, my_id, data);
               free_entry();
        default:
                error();
        else                            // IO_READ_OWNER request to our caches
            assign_entry();
            local_request(READ_LATEST);// spin until a valid response from
                                        // the caches
            switch (local_response)
            case MODIFIED:              // processor indicated a push;
                                        // wait for it
                if (received_srcid == received_secid)
                                        // original requestor is also home
                                        // module
                    remote_response(INTERVENTION, received_srcid, my_id,
                                        data);
                else
                    remote_response(DATA_ONLY, received_secid, my_id,
                                        data);
                    remote_response(INTERVENTION, received_srcid, my_id);
                endif;
            case INVALID:               // must have cast it out during
                                        // an address collision
                remote_response(NOT_OWNER, received_srcid, my_id);
            default:
                error();
            free_entry();
        endif;
```

5.1.2 Memory-only Processing Element

This processing element is simpler than the assumed processing element used in Chapter 6, removing all of the internal request state machines and portions of all of the external request and response state machines. A memory-only processing element does not receive TLB invalidate entry or TLB invalidate synchronize operations. The required behavior for each operation is described below.

5.1.2.1 Read Operations

This operation is a coherent data cache read.

5.1.2.1.1 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of a third party.

```
switch(remote_response)
case INTERVENTION:
    update_memory();
    update_state(SHARED, original_srcid);
    remote_response(DONE_INTERVENTION, original_srcid, my_id);
    free_entry();
case NOT_OWNER,                 // data comes from memory,
                                // mimic intervention
case RETRY:
    switch(directory_state)
    case LOCAL_SHARED:
        update_state(SHARED, original_srcid);
        remote_response(DATA_ONLY, original_srcid,
                                my_id, data);
        remote_response(DONE_INTERVENTION, original_srcid,
                                my_id);
        free_entry();
    case LOCAL_MODIFIED:
        update_state(SHARED, original_srcid);
        remote_response(DATA_ONLY, original_srcid,
                                my_id, data);
                remote_response(DONE_INTERVENTION, original_srcid,
                                my_id);
                free_entry();
        case REMOTE_MODIFIED:// spin or wait for castout
                remote_request(READ_OWNER, received_srcid,
                                my_id, my_id);
        default:
                error();
default:
        error();
```

5.1.2.1.2 External Request State Machine

This state machine handles read requests from the system to the local memory. This may require making further external requests.

```
if (address_collision)          // use collision tables in
                                // Chapter 7, “Address Collision Resolution Tables”
else                            // READ_HOME
        assign_entry();
        switch (directory_state)
        case LOCAL_MODIFIED:
                local_request(READ);
                update_state(SHARED, received_srcid);
                                // after possible push completes
                remote_response(DONE, received_srcid, my_id, data);
                free_entry();
        case LOCAL_SHARED,
        case SHARED:
                update_state(SHARED, received_srcid);
                remote_response(DONE, received_srcid, my_id, data);
                free_entry();
        case REMOTE_MODIFIED:
                if (mask_id ~= received_srcid)
                                // intervention case
                                remote_request(READ_OWNER, mask_id,
                                my_id, received_srcid);
                else
                                error();// he already owned it;
                                // cache paradox (or I-fetch after d-
                                // store if not fixed elsewhere)
                endif;
        default:
                error();
endif;
```

5.1.2.2 Instruction Read Operations

This operation is a partially coherent instruction cache read.

5.1.2.2.1 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of a third party.

```
switch(remote_response)
case INTERVENTION:
        update_memory();
        update_state(SHARED, original_srcid);
        remote_response(DONE, original_srcid, my_id);
        free_entry();
case NOT_OWNER,                 // data comes from memory,
                    // mimic intervention
case RETRY:
      switch(directory_state)
      case LOCAL_SHARED:
           update_state(SHARED, original_srcid);
           remote_response(DONE, original_srcid, my_id);
           free_entry();
      case LOCAL_MODIFIED:
           update_state(SHARED, original_srcid);
           remote_response(DONE, original_srcid, my_id);
           free_entry();
      case REMOTE_MODIFIED:// spin or wait for castout
           remote_request(READ_OWNER, received_srcid,
                    my_id, my_id);
      default:
           error();
default:
      error();
```

5.1.2.2.2 External Request State Machine

This state machine handles instruction read requests from the system to the local memory. This may require making further external requests.

```
if (address_collision)    // use collision tables in
                          // Chapter 7, "Address Collision Resolution Tables"
else                      // IREAD_HOME
      assign_entry();
      switch (directory_state)
      case LOCAL_MODIFIED:
           local_request(READ);
           update_state(SHARED, received_srcid);
                          // after possible push completes
           remote_response(DONE, received_srcid, my_id, data);
           free_entry();
      case LOCAL_SHARED,
      case SHARED:
           update_state(SHARED, received_srcid);
           remote_response(DONE, received_srcid, my_id, data);
           free_entry();
      case REMOTE_MODIFIED:
           if (mask_id ~= received_srcid)
                          // intervention case
                          remote_request(READ_OWNER, mask_id,
                          my_id, received_srcid);
           else           // he already owned it in his
                          //data cache; cache paradox case
                          remote_request(READ_OWNER, mask_id, my_id, my_id);
           endif;
      default:
           error();
endif;
```

5.1.2.3 Read for Ownership Operations

This is the coherent cache store miss operation.

5.1.2.3.1 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of a third party.

```
switch(received_response)
```

```
case DONE:                              // invalidates for shared
                                        // directory states
if ((mask ~= (my_id OR received_id)) == 0)
                                        // this is the last DONE
        update_state(REMOTE_MODIFIED, original_srcid);
        remote_response(DONE, original_srcid, my_id, data);
        free_entry();
else
        mask <= (mask ~= received_srcid);
                                        // flip the responder's shared bit
endif;                                  // and wait for next DONE
case INTERVENTION:
                                        // remote_modified case
update_memory();                        // for possible coherence error
                                        // recovery
update_state(REMOTE_MODIFIED, original_id);
remote_response(DONE_INTERVENTION, original_id, my_id);
free_entry();
case NOT_OWNER:                         // data comes from memory, mimic
                                        // intervention
switch(directory_state)
case LOCAL_SHARED:
case LOCAL_MODIFIED:
        update_state(REMOTE_MODIFIED, original_srcid);
        remote_response(DATA_ONLY, original_srcid, my_id,
                        data);
        remote_response(DONE, original_srcid, my_id);
        free_entry();
case REMOTE_MODIFIED:
        remote_request(READ_TO_OWN_OWNER, received_srcid,
                        my_id, original_srcid);
default:
        error();
case RETRY:
switch (directory_state)
case LOCAL_MODIFIED,
case LOCAL_SHARED:
        update_state(REMOTE_MODIFIED, original_srcid);
        remote_response(DATA_ONLY, original_srcid, my_id,
                        data);
        remote_response(DONE, original_srcid, my_id);
        free_entry();
case REMOTE_MODIFIED:                   // mask_id must match received_srcid
                                        // or error condition
        remote_request(READ_TO_OWN_OWNER, received_srcid,
                        my_id, my_id);
case SHARED:
        remote_request(DKILL_SHARER, received_srcid, my_id,
                        my_id);
default:
        error();
default:
error();
```

5.1.2.3.2 External Request State Machine

This state machine handles requests from the interconnect to the local memory. This may require making further external requests.

```
if (address_collision)                  // use collision tables
                                        // in Chapter 7, “Address Collision Resolution Tables”
else                                    // READ_TO_OWN_HOME
assign_entry();
switch (directory_state)
case LOCAL_MODIFIED,
case LOCAL_SHARED:
        local_request(READ_TO_OWN);
        remote_response(DONE, received_srcid, my_id, data);
                                        // after possible push
        update_state(REMOTE_MODIFIED, received_srcid);
        free_entry();
case REMOTE_MODIFIED:
        if (mask_id ~= received_srcid)
                                        //intervention case
                    remote_request(READ_TO_OWN_OWNER, mask_id, my_id,
                                        received_srcid);
        else
                    error();            // he already owned it!
        endif;
case SHARED:
        local_request(READ_TO_OWN);
        if (mask == received_srcid)
                                        //requestor is only remote sharer
                    update_state(REMOTE_MODIFIED, received_srcid);
                    remote_response(DONE, received_srcid, my_id, data);
                                        // from memory
                    free_entry();
        else                            //there are other remote sharers
                    remote_request(DKILL_SHARER, (mask ~= received_srcid),
                                        my_id, my_id);
        endif;
default:
        error();
endif;
```

5.1.2.4 Data Cache and Instruction Cache Invalidate Operations

This operation is used with coherent cache store-hit-on-shared, cache operations.

5.1.2.4.1 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of a third party.

```
switch(received_response)
case DONE:                              // invalidates for shared
                                        // directory states
if ((mask ~= (my_id OR received_id)) == 0)
                                        // this is the last DONE
                    update_state(REMOTE_MODIFIED, original_srcid);
                    remote_response(DONE, original_srcid, my_id);
                    free_entry();
else
                    mask <= (mask ~= received_srcid);
                                        // flip the responder’s shared bit
endif;                                  // and wait for next DONE
case RETRY:
remote_request({DKILL_SHARER, IKILL_SHARER}, received_srcid,
                    my_id);             // retry
default:
```

```
error();
```

5.1.2.4.2 External Request State Machine

This state machine handles requests from the system to the local memory. This may require making further external requests.

```
if (address_collision)                    // use collision tables in
                                          // Chapter 7, "Address Collision Resolution Tables"
else                                      // DKILL_HOME or IKILL_HOME
assign_entry();
if (DKILL_HOME)
        switch (directory_state)
        case LOCAL_MODIFIED,              // cache paradoxes; DKILL is
                                          // write-hit-on-shared
        case LOCAL_SHARED,
        case REMOTE_MODIFIED:
                error();
        case SHARED:                      // this is the right case, send
                                          // invalidates to the sharing list
                local_request(DKILL);
                if (mask == received_srcid
                                          // requestor is only remote sharer
                          update_state(REMOTE_MODIFIED, received_srcid);
                          remote_response(DONE, received_srcid, my_id);
                          free_entry();
                else                      // there are other remote sharers
                          remote_request(DKILL_SHARER,
                                          (mask ~= received_srcid), my_id, NULL);
                endif;
        default:
                error();
else                                      // IKILL goes to everyone except the
                                          // requestor
        remote_request(IKILL_SHARER,
                (mask <= (participant_list ~=
                (received_srcid AND my_id), my_id);
endif;
```

5.1.2.5 Castout Operations

This operation is used to return ownership of a coherence granule to home memory, leaving it invalid in the cache.

5.1.2.5.1 External Request State Machine

This machine handles requests from the system to the local memory. This may require making further external requests.

```
assign_entry();
update_memory();
state_update(LOCAL_SHARED, my_id);  // may be LOCAL_MODIFIED if the
                                    // default is owned locally
remote_response(DONE, received_srcid, my_id);
free_entry();
```

5.1.2.6 Data Cache Flush Operations

This operation returns ownership of a coherence granule to home memory and performs a coherent write.

5.1.2.6.1 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of a third party.

```
switch(received_response)
case DONE:                                // invalidates for shared directory
```

```
                                        // states
          if ((mask ~= (my_id OR received_id)) == 0)
                                        // this is the last DONE
             remote_response(DONE, original_srcid, my_id, my_id);
             if (received_data)
                                        // with original request or response
                   update_memory();
             endif;
             update_state(LOCAL_SHARED);// or LOCAL_MODIFIED
             free_entry();
          else
             mask <= (mask ~= received_srcid);
                                        // flip responder's shared bit
          endif;                        // and wait for next DONE
case NOT_OWNER:
          switch(directory_state)
          case LOCAL_SHARED,
          case LOCAL_MODIFIED:
             remote_response(DONE, original_srcid, my_id);
             if (received_data)
                                        // with original request
                   update_memory();
             endif;
             free_entry();
          case REMOTE_MODIFIED:
             remote_request(READ_TO_OWN_OWNER, received_srcid,
                   my_id, original_srcid);
          default:
             error();
case RETRY:
          switch(directory_state)
          case LOCAL_SHARED,
          case LOCAL_MODIFIED:
             remote_response(DONE, original_srcid, my_id);
             if (received_data)
                                        // with original request
                   update_memory();
             endif;
             free_entry();
          case REMOTE_MODIFIED:
             remote_request(READ_TO_OWN_OWNER, received_srcid,
                   my_id, original_srcid);
          case SHARED:
             remote_request(DKILL_SHARER, received_srcid, my_id);
          default:
             error();
default:
          error();
```

5.1.2.6.2 External Request State Machine

This state machine handles requests from the system to the local memory. This may require making further external requests.

```
if (address_collision)                  // use collision tables in
                                        // Chapter 7, "Address Collision Resolution Tables"
else                                    // FLUSH
assign_entry();
```

```
switch (directory_state)
case LOCAL_MODIFIED,
case LOCAL_SHARED:
                local_request(READ_TO_OWN);
                remote_response(DONE, received_srcid, my_id);
                                // after snoop completes
                if (received_data)     // from request or local response
                     update_memory();
                endif;
                update_state(LOCAL_SHARED, my_id);
                                // or LOCAL_MODIFIED
                free_entry();
case REMOTE_MODIFIED:
                if (mask_id ~= received_srcid) // owned elsewhere
                     remote_request(READ_TO_OWN_OWNER, mask_id, my_id,
                                received_srcid);
                else            // requestor owned it; shouldn't
                                // generate a flush
                     error();
                endif;
case SHARED:
                local_request(READ_TO_OWN);
                if (mask == received_srcid)  // requestor is only remote sharer
                     remote_response(DONE, received_srcid, my_id);
                                // after snoop completes
                     if (received_data)// from request or response
                                update_memory();
                     endif;
                     update_state(LOCAL_SHARED, my_id); // or LOCAL_MODIFIED
                     free_entry();
                else            //there are other remote sharers
                     remote_request(DKILL_SHARER, (mask ~= received_srcid), my_id,
                                my_id);
                endif;
default:
                error();
endif;
```

5.1.2.7 I/O Read Operations

This operation is used for I/O reads of globally shared memory space.

5.1.2.7.1 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of a third party.

```
switch(remote_response)
case INTERVENTION:
           update_memory();
           remote_response(DONE_INTERVENTION, original_srcid, my_id);
           free_entry();
case NOT_OWNER,                 // data comes from memory, mimic
                                // intervention
case RETRY:
           switch(directory_state)
           case LOCAL_MODIFIED,
           case LOCAL_SHARED:
                remote_response(DATA_ONLY, original_srcid, my_id,
                     data);
                remote_response(DONE_INTERVENTION, original_srcid,
                        my_id);
                    free_entry();
                case REMOTE_MODIFIED:// spin or wait for castout
                    remote_request(IO_READ_OWNER, received_srcid, my_id,
                        my_id);
                default:
                    error();
default:
                error();
```

5.1.2.7.2 External Request State Machine

This machine handles requests from the system to the local memory. This may require making further external requests.

```
if (address_collision)                  // use collision tables in
                                        // Chapter 7, “Address Collision Resolution Tables”
else                                    // IO_READ_HOME
        assign_entry();
        switch (directory_state)
        case LOCAL_MODIFIED:
                    local_request(READ_LATEST);
                    remote_response(DONE, received_srcid, my_id, data);
                                        // after push completes
                    free_entry();
        case LOCAL_SHARED:
                    remote_response(DONE, received_srcid, my_id, data);
                    free_entry();
        case REMOTE_MODIFIED:
                    remote_request(IO_READ_OWNER, mask_id, my_id, received_srcid);
        case SHARED:
                    remote_response(DONE, received_srcid, my_id, data);
                    free_entry();
        default:
                    error();
endif;
```

5.1.3 Processor-only Processing Element

A processor-only processing element is much simpler than the assumed combined processing described in Chapter 6. Much of the internal request, response, and external request state machines are removed.

5.1.3.1 Read Operations

This operation is a coherent data cache read.

5.1.3.1.1 Internal Request State Machine

This state machine handles requests to remote memory from the local processor.

```
if (address_collision)                  // this is due to an external request
                                        // in progress or a cache
        local_response(RETRY);          // index hazard from a previous request
else                                    // remote - we’ve got to go
                                        // to another module
        assign_entry();
        local_response(RETRY);          // can’t guarantee data before a
                                        // snoop yet
        remote_request(READ_HOME, mem_id, my_id);
endif;
```

5.1.3.1.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of the local processor.

```
switch(remote_response)
case DONE:
        local_response(SHARED);         // when processor re-requests
        return_data();
        free_entry();
case DONE_INTERVENTION:                 // must be from third party
        set_received_done_message();
        if (received_data_only_message)
                free_entry();
        else
                                        // wait for a DATA_ONLY
        endif;
case DATA_ONLY:                         // this is due to an intervention, a
                                        // DONE_INTERVENTIONshould come
                                        // separately
        local_response(SHARED);
        set_received_data_only_message();
        if (received_done_message)
                return_data();
                free_entry();
        else
                return_data();          // OK for weak ordering
        endif;
case RETRY:
        remote_request(READ_HOME, received_srcid, my_id);
default
        error();
```

5.1.3.1.3 External Request State Machine

This state machine handles read requests from the system to the local processor. This may require making further external requests.

```
if (address_collision)                  // use collision tables in
                                        // Chapter 7, "Address Collision Resolution Tables"
else                                    // READ_OWNER
        assign_entry();
        local_request(READ);            // spin until a valid response
                                        // from caches
        switch (local_response)
        case MODIFIED:                  // processor indicated a push;
                                        // wait for it
                cache_state(SHARED or INVALID);
                                        // surrender ownership
                if (received_srcid == received_secid)
                                        // original requestor is also home
                    remote_response(INTERVENTION, received_srcid,
                        my_id, data);
                else
                    remote_response(DATA_ONLY, received_secid,
                        my_id, data);
                    remote_response(INTERVENTION, received_srcid,
                        my_id, data);
                endif;
        case INVALID:                   // must have cast it out
                remote_response(NOT_OWNER, received_srcid, my_id);
        default:
                error();
        free_entry();
endif;
```

5.1.3.2 Instruction Read Operations

This operation is a partially coherent instruction cache read.

5.1.3.2.1 Internal Request State Machine

This state machine handles requests to remote memory from the local processor.

```
if (address_collision)                  // this is due to an external
                                         // request in progress or a cache
            local_response(RETRY);      // index hazard from a previous request
else                                    // remote - we've got to go
                                        // to another module
            assign_entry();
            local_response(RETRY);
                                        // can't guarantee data before a
                                        // snoop yet
            remote_request(IREAD_HOME, mem_id, my_id);
endif;
```

5.1.3.2.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of the local processor.

```
switch(remote_response)
case DONE:
            local_response(SHARED);     // when processor re-requests
            return_data();
            free_entry();
case DONE_INTERVENTION:                 // must be from third party
            set_received_done_message();
            if (received_data_only_message)
                        free_entry();
            else
                                        // wait for a DATA_ONLY
            endif;
case DATA_ONLY:                         // this is due to an intervention; a
                                        // DONE_INTERVENTION should come
                                        // separately
            local_response(SHARED);
            set_received_data_only_message();
            if (received_done_message)
                        return_data();
                        free_entry();
            else
                        return_data();  // OK for weak ordering
            endif;
case RETRY:
            remote_request(IREAD_HOME, received_srcid, my_id);
default
            error();
```

5.1.3.2.3 External Request State Machine

This state machine handles instruction read requests from the system to the local processor.

```
if (address_collision)                  // use collision tables in
                                        // Chapter 7, "Address Collision Resolution Tables"
else                                    // READ_OWNER request to our caches
```

```
        assign_entry();
        local_request(READ);              // spin until a valid response
                                          // from caches
        switch (local_response)
        case MODIFIED:                    // processor indicated a push;
                                          // wait for it
            cache_state(SHARED or INVALID);
                                          // surrender ownership
            if (received_srcid == received_secid)
                                          // original requestor is also home
                remote_response(INTERVENTION, received_srcid,
                    my_id, data);
            else
                remote_response(DATA_ONLY, received_secid,
                    my_id, data);
                remote_response(INTERVENTION, received_srcid,
                    my_id, data);
            endif;
        case INVALID:                     // must have cast it out
            remote_response(NOT_OWNER, received_srcid, my_id);
        default:
            error();
        free_entry();
endif;
```

5.1.3.3 Read for Ownership Operations

This is the coherent cache store miss operation.

5.1.3.3.1 Internal Request State Machine

This state machine handles requests to remote memory from the local processor.

```
if (address_collision)                    // this is due to an external request
                                          // in progress or a cache index
        local_response(RETRY);            // hazard from a previous request
else                                      // remote - we've got to go to another
                                          // module
        assign_entry();
        local_response(RETRY);
        remote_request(READ_TO_OWN_HOME, mem_id, my_id);
endif;
```

5.1.3.3.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of the local processor.

```
switch (received_response)
case DONE:
        local_response(EXCLUSIVE);
        return_data();
        free_entry();
case DONE_INTERVENTION:
        set_received_done_message();
        if (received_data_message)
                free_entry();
        else
                                          // wait for DATA_ONLY
        endif;
case DATA_ONLY:
        set_received_data_message();
```

```
            local_response(EXCLUSIVE);
            if (received_done_message)
                        return_data();
                        free_entry();
            else
                        return_data();          // OK for weak ordering
            endif;                              // and wait for a DONE
case RETRY:                                     // lost at remote memory so retry
            remote_request(READ_TO_OWN_HOME, mem_id, my_id);
default:
            error();
```

5.1.3.3.3 External Request State Machine

This state machine handles requests from the interconnect to the local processor.

```
if (address_collision)                          // use collision tables
                                                // in Chapter 7, "Address Collision Resolution Tables"
elseif(READ_TO_OWN_OWNER                        // request to our caches
assign_entry();
local_request(READ_TO_OWN);                     // spin until a valid response from
                                                // the caches
switch (local_response)
case MODIFIED:                                  // processor indicated a push
            cache_state(INVALID);
                                                // surrender ownership
            if (received_srcid == received_secid)
                                                //the original request is from the home
                  remote_response(INTERVENTION, received_srcid, my_id,
                        data);
            else                                // the original request is from a
                                                // third party
                  remote_response(DATA_ONLY, received_secid, my_id,
                        data);
                  remote_response(INTERVENTION, received_srcid, my_id,
                        data);
            endif;
            free_entry();
case INVALID:                                   // castout address collision
            remote_response(NOT_OWNER, received_srcid, my_id);
default:
            error();
else                                            // DKILL_SHARER request to our caches
assign_entry();
local_request(READ_TO_OWN);
                                                // spin until a valid response from the
                                                // caches
switch (local_response)
case SHARED,
case INVALID:                                   // invalidating for shared cases
            cache_state(INVALID);               // surrender copy
            remote_response(DONE, received_srcid, my_id);
            free_entry();
default:
            error();

endif;
```

5.1.3.4 Data Cache and Instruction Cache Invalidate Operations

This operation is used with coherent cache store-hit-on-shared, cache operations.

5.1.3.4.1 Internal Request State Machine

This state machine handles requests to remote memory from the local processor.

```
if (address_collision)                         // this is due to an external request in
                                               // progress or a cache index
            local_response(RETRY);             // hazard from a previous request
else                                           // remote - we've got to go to another
                                               // module
            assign_entry();
            local_response(RETRY);
            remote_request({DKILL_HOME, IKILL_HOME}, mem_id, my_id);
endif;
```

5.1.3.4.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of the local processor.

```
switch (received_response)
case DONE:
            local_response(EXCLUSIVE);
            free_entry();
case RETRY:
            remote_request({DKILL_HOME, IKILL_HOME}, received_srcid,
            my_id);// retry the transaction
default:
            error();
```

5.1.3.4.3 External Request State Machine

This state machine handles requests from the system to the local processor.

```
if (address_collision)                         // use collision tables in
                                               // Chapter 7, "Address Collision Resolution Tables"
else                                           // DKILL_SHARER or IKILL_SHARER request to our caches
            assign_entry();
            local_request({READ_TO_OWN, IKILL});
                                               // spin until a valid response from the
                                               // caches
            switch (local_response)
            case SHARED,
            case INVALID:                      // invalidating for shared cases
                                               cache_state(INVALID);// surrender copy
                                               remote_response(DONE, received_srcid, my_id);
                                               free_entry();
            default:
                                               error();
endif;
```

5.1.3.5 Castout Operations

This operation is used to return ownership of a coherence granule to home memory, leaving it invalid in the cache. A processor-only processing element is never the target of a castout operation.

5.1.3.5.1 Internal Request State Machine

A castout may require local activity to flush all caches in the hierarchy and break possible reservations.

```
assign_entry();
local_response(OK);
remote_request(CASTOUT, mem_id, my_id, data);
```

5.1.3.5.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of the local processor.

```
switch (received_response)
case DONE:
        free_entry();
default:
        error();
```

5.1.3.6 TLB Invalidate Entry, TLB Invalidate Entry Synchronize Operations

These operations are used for software coherence management of the TLBs.

5.1.3.6.1 Internal Request State Machine

The TLBIE and TLBSYNC transactions are always sent to all domain participants except the sender and are always to the processor, not home memory.

```
assign_entry();
remote_request({TLBIE, TLBSYNC}, participant_id, my_id);
endif;
```

5.1.3.6.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of the local processor. The responses are always from a coherence participant, not a home memory.

```
switch (received_response)
case DONE:
        if ((mask ~= (my_id OR received_id)) == 0)
                                          // this is the last DONE
                free_entry();
        else
                mask <= (mask ~= received_srcid);
                                          // flip the responder’s participant
                                          // bit and wait for next DONE
        endif;
case RETRY:
        remote_request({TLBIE, TLBSYNC}, received_srcid, my_id, my_id);
default
        error();
```

5.1.3.6.3 External Request State Machine

This state machine handles requests from the system to the local memory or the local processor. The requests are always to the local caching hierarchy.

```
assign_entry();
local_request({TLBIE, TLBSYNC});          // spin until a valid response
                                          // from the caches
remote_response(DONE, received_srcid, my_id);
free_entry();
```

5.1.3.7 Data Cache Flush Operations

This operation returns ownership of a coherence granule to home memory and performs a coherent write.

5.1.3.7.1 Internal Request State Machine

This state machine handles requests to remote memory from the local processor.

```
if (address_collision)                    // this is due to an external
                                          // request in progress or a cache index
        local_response(RETRY);            // hazard from a previous request
else                                      // remote - we’ve got to go to
                                          // another module
        assign_entry();
        remote_request(FLUSH, mem_id, my_id, data);
                                      // data is optional
endif;
```

5.1.3.7.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of the local processor.

```
switch (received_response)
case DONE:
        local_response(OK);
        free_entry();
case RETRY:
        remote_request(FLUSH, received_srcid, my_id, data);
                                      // data is optional
default:
        error();
```

5.1.3.7.3 External Request State Machine

This state machine handles requests from the system to the local processor.

```
if (address_collision)                // use collision tables in
                                      // Chapter 7, "Address Collision Resolution Tables"
else if (READ_TO_OWN_OWNER)           // remote request to our caches
        assign_entry();
        local_request(READ_TO_OWN);   // spin until a valid response
                                      // from the caches
        switch (local_response)
        case MODIFIED:                // processor indicated a push,
                                      // wait for it
                cache_state(INVALID);// surrender ownership
                remote_response(DONE, received_srcid, my_id, data);
        case INVALID:
                                      // must have cast it out during an
                                      // address collision
                remote_response(NOT_OWNER, received_srcid, my_id);
        default:
                error();
        free_entry();
else                                  // DKILL_SHARER remote request
                                      // to our caches
        assign_entry();
        local_request(DKILL);         // spin until a valid response from
                                      // the caches
        switch (local_response)
        case MODIFIED:                // cache paradox
                remote_response(ERROR, received_srcid, my_id);
        case INVALID:
                remote_response(DONE, received_srcid, my_id);
        default:
                error();
        free_entry();
endif;
```

5.1.3.8 I/O Read Operations

This operation is used for I/O reads of globally shared memory space. A processor-only processing element never initiates an I/O read operation.

5.1.3.8.1 External Request State Machine

This machine handles requests from the system to the local memory or the local processor. This may require making further external requests.

```
if (address_collision)              // use collision tables in
                                    // Chapter 7, “Address Collision Resolution Tables”
else                                // IO_READ_OWNER request to our caches
        assign_entry();
        local_request(READ_LATEST); // spin until a valid response from
                                    // the caches
        switch (local_response)
        case MODIFIED:              // processor indicated a push;
                                    // wait for it
                if (received_srcid == received_secid)
                                    // original requestor is also home
                                    // module
                        remote_response(INTERVENTION, received_srcid, my_id,
                                    data);
                else
                        remote_response(DATA_ONLY, received_secid, my_id,
                                    data);
                        remote_response(INTERVENTION, received_srcid, my_id);
                endif;
        case INVALID:               // must have cast it out during
                                    // an address collision
                remote_response(NOT_OWNER, received_srcid, my_id);
        default:
                error();
        free_entry();
endif;
```

5.1.4 I/O Processing Element

The simplest GSM processing element is an I/O device. A RapidIO I/O processing element does not actually participate in the globally shared memory environment (it is defined as not in the coherence domain), but is able to read and write data into the GSM address space through special I/O operations that provide for this behavior. These operations are the I/O Read and Data Cache Flush operations. Other than the ability to read and write into the GSM address space, an I/O device has no other operational requirements. Since the I/O processing element is not part of the coherence domain, it is never the target of a coherence transaction and thus does not have to implement any of the related behavior, including the address collision tables.

Requirements for a specific I/O processing element, a RapidIO to PCI/PCI-X bridge, is discussed in “PCI Considerations,” .

5.1.4.1 I/O Read Operations

This operation is used for I/O reads of globally shared memory space.

5.1.4.1.1 Internal Request State Machine

This state machine handles requests to remote memory.

```
remote_request(IO_READ_HOME, mem_id, my_id);
```

5.1.4.1.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect.

```
switch(remote_response)
case DONE:
        return_data();
        free_entry();
case DONE_INTERVENTION:             // must be from third party
        set_received_done_message();
        if (received_data_only_message)
                free_entry();
        else
                // wait for a DATA_ONLY
        endif;
case DATA_ONLY:                         // this is due to an intervention, a
                                        // DONE_INTERVENTION should come
                                        // separately
        set_received_data_only_message();
        if (received_done_message)
                return_data();
                free_entry();
        else
                return_data();          // OK for weak ordering
        endif;
case RETRY:
        remote_request(IO_READ_HOME, received_srcid, my_id);
default
        error();
```

5.1.4.2 Data Cache Flush Operations

This operation returns ownership of a coherence granule to home memory and performs a coherent write.

5.1.4.2.1 Internal Request State Machine

This state machine handles requests to remote memory.

```
remote_request(FLUSH, mem_id, my_id, data);
```

5.1.4.2.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect.

```
switch (received_response)
case DONE:
        local_response(OK);
        free_entry();
case RETRY:
        remote_request(FLUSH, received_srcid, my_id, data);
default:
        error();
```

5.1.5 Switch Processing Element

A switch processing element is required to be able to route all defined packets. Since it is not necessary for a switch to analyze a packet in order to determine how it should be treated outside of examining the priority and the destination device ID, a switch processing element does not have any additional requirements to be used in a globally shared memory environment.

5.2 Transaction to Priority Mappings

The Globally Shared Memory model does not have the concept of an end point to end point request transaction flow like the I/O programming model. Instead, all transaction ordering is managed by the load-store units of the processors participating in the globally shared memory protocol. The GSM logical specification behaviors assume an unordered and resource unconstrained communication fabric. The ordered fabric of the 8/16 LP-LVDS and the 1x/4x LP-Serial physical layers requires the proper transaction to priority mappings to mimic the effect of an unordered fabric to suit the GSM model. These mappings leverage the physical layer ordering and deadlock avoidance rules that are required by the I/O Logical layer. In addition, it is assumed that the latency-critical GSM operations are of necessity higher priority than non-coherent I/O traffic, therefore I/O operations are recommended to be assigned to the lowest system priority flow.

Table shows the GSM transaction to priority mappings.

Table7-1. Transaction to Priority Mapping

Request transaction	Request Packet Priority	Response Packet Priority
READ_TO_OWN_HOME	1	2 or 3
READ_HOME	1	2 or 3
IO_READ_HOME	1	2 or 3
IREAD_HOME	1	2 or 3
DKILL_HOME	1	2 or 3
IKILL_HOME	1	2 or 3
FLUSH (without data)	1	2 or 3
FLUSH (with data)	1	2 or 3
TLBIE	1	2 or 3
TLBSYNC	1	2 or 3
READ_OWNER	2	3
READ_TO_OWN_OWNER	2	3
IO_READ_OWNER	2	3
DKILL_SHARER	2	3
IKILL_SHARER	2	3
CASTOUT	2	3

Partition VIII:

Error Management Extensions Specification

VIII Partition VIII - Error Management Extensions Specification

1 Chapter 1 - Error Management Extensions

The error management extensions describe added requirements in all physical and logical layers. These extensions add definitions to bits that were previously reserved in the Port *n* Control CSR and add new registers that are contained within the Error Management Extended Features Block. This chapter describes the behavior of a device when an error is detected and how the new registers and bits are managed by software and hardware.

1.1 Physical Layer Extensions

The following registers and register bit extensions allow software to monitor and control the reporting of transmission errors:

- Port *n* Error Detect CSR defined in Section 2.2.2.10: "Port n Error Detect CSR (Block Offset 0x40, 80,..., 400, Word 0)"
- Port *n* Error Rate Enable CSR defined in Section 2.2.2.11: "Port n Error Rate Enable CSR (Block Offset 0x40, 80,..., 400, Word 1)"
- Port *n* Error Rate CSR defined in Section 2.2.2.17: "Port n Error Rate CSR (Block Offset 0x68, A8,..., 428, Word 0)"
- Port *n* Error Rate Threshold CSR defined in Section 2.2.2.18: "Port n Error Rate Threshold CSR (Block Offset 0x68, A8,..., 428, Word 1)"
- Port *n* Error Capture Attributes CSR defined in Section 2.2.2.12: "Port n Error Capture Attributes CSR (Block Offset 0x48, 88,..., 408, Word 0)"
- Port *n* Error Capture CSR 0-3 defined in Section 2.2.2.13: "Port n Packet/Control Symbol Error Capture CSR 0 (Block Offset 0x48, 88,..., 408, Word 1)" thru Section 2.2.2.16: "Port n Packet Error Capture CSR 3 (Block Offset 0x58, 98,..., 418, Word 0)"
- Port-write Target device ID CSR defined in Section 2.2.2.8: "Port-write Target deviceID CSR (Block Offset 0x28 Word 0)"
- (Extensions to the) Port *n* Control CSR defined in Section 2.1: "Additions to Existing Registers"
- (Extensions to the) Port *n* Error and Status CSR defined in Section 2.1: "Additions to Existing Registers"

1.1.1 Port Error Detect, Enable and Capture CSRs

The occurrence of a transmission error shall be logged by hardware by setting the appropriate error indication bit in the Port *n* Error Detect CSR. Transmission errors that are enabled for error capture and error counting will have the corresponding bit set by software in the Port *n* Error Rate Enable CSR. When the Capture Valid Info status bit is not set in the Port *n* Error Capture Attributes CSR, information about the next enabled transmission error shall be saved to the Port *n* Error Capture CSRs. The Info Type and Error Type fields shall be updated and the Capture Valid Info status bit shall be set by hardware in the Port *n* Error Capture Attributes CSR to lock the error capture registers. The first 16 bytes of the packet header or the 4 bytes of the control symbol that have a detected error are saved in the capture CSRs. Packets smaller than 16 bytes are captured in their entirety. The Port *n* Error Capture CSRs and the Port *n* Error Capture Attributes CSR are not overwritten by hardware with error capture information for subsequent errors until software writes a zero to the Capture Valid Info bit.

The Port *n* Error Detect CSR does not lock so subsequent error indications shall also be logged there by hardware. By reading the register, software may see the types of transmission errors that have occurred. The Port *n* Error Detect CSR is cleared by writing it with all logic 0s.

1.1.2 Error Reporting Thresholds

Transmission errors are normally hidden from system software since they may be recovered with no loss of data and without software intervention. Two thresholds are defined in the Port *n* Error Rate Threshold CSR which can be set to force a report to system software when the link error rate reaches a level that is deemed by the system to be either degraded or unacceptable. The two thresholds are respectively the Degraded Threshold and the Failed Threshold. These thresholds are used as follows.

When the error rate counter is incremented, the Error Rate Degraded Threshold Trigger provides a threshold value that, when equal to or exceeded by the value in the Error Rate Counter in the Port *n* Error Rate register, shall cause the error reporting logic to set the Output Degraded-encountered bit in the Port *n* Error and Status CSR, and notify the system software as described in Section : "8.3 System Software Notification of Error".

The Error Rate Failed Threshold Trigger, if enabled, shall be larger than the degraded threshold trigger. It provides a threshold value that, when equal to or exceeded by the value in the Error Rate Counter, shall trigger the error reporting logic to set the Output Failed-encountered bit in the Port *n* Error and Status CSR, and notify system software as described in Section : "8.3 System Software Notification of Error".

No action shall be taken if the Error Rate Counter continues to exceed either threshold value after initial notification when additional errors are detected. No action shall be taken when the Error Rate Counter drops below either threshold.

1.1.3 Error Rate Control and Status

The fields in the Port *n* Error Rate CSR are used to monitor the error rate of the link connected to port *n*.

The Error Rate Bias field determines the rate at which the Error Rate Counter is decremented and defines the acceptable error rate of the link for error reporting purposes. In the absence of additional counted link errors, this mechanism allows the system to recover from both Failed and Degraded levels of operation without a software reset of the Error Rate Counter. If the link error rate is less than the decrement rate specified in the Error Rate Bias field, the value of the Error Rate counter will rarely be greater than 0x01 or 0x02.

The Error Rate Counter shall increment when a physical layer error is detected whose associated enable bit is set in the Port *n* Error Rate Enable register. The Error Rate Counter shall decrement at the rate specified by the Error Rate Bias field of the Port *n* Error Rate CSR. The Error Rate Counter shall not underflow (shall not decrement when equal to 0x00) and shall not overflow (shall not increment when equal to 0xFF). The incrementing and decrementing of the Error Rate Counter are in no way affected by the values in the Degraded and Failed thresholds. Software may reset the Error Rate Counter at any time.

The Error Rate Recovery field defines how far above the Error Rate Failed Threshold Trigger in the Port *n* Error Rate Threshold Register the Error Rate Counter is allowed to count. In the absence of additional counted errors, this allows software to control the length of time required for the value of the Error Rate Counter to drop below both the Failed and Degraded Thresholds.

The Peak Error Rate field shall contain the largest value encountered by the Error Rate Counter. This field is loaded whenever the current value of the Peak Error Rate field is exceeded by the value of the Error Rate Counter.

1.1.4 Port Behavior When Error Rate Failed Threshold is Reached

The behavior of a port when the Error Rate Counter in the Port *n* Error Rate CSR reaches the Error Rate Failed Threshold and the threshold is enabled depends upon the values of the Stop on Port Failed-encountered Enable and the Drop Packet Enable bits in the Port *n* Control CSR. The Table I-I below defines the required behavior.

Table 1-1. Port Behavior when Error Rate Failed Threshold has been hit

Stop on Port Failed Encoutered Enable	Drop Packet Enable	Port Behavior	Comments
0	0	The port shall continue to attempt to transmit packets to the connected device if the Output Failed-encountered bit is set and/or if the Error Rate Failed threshold has been met or exceeded..	All devices
0	1	The port shall discard packets that receive a Packet-not-accepted control symbol when the Error Rate Failed Threshold has been met or exceeded. Upon discarding a packet, the port shall set the Ourput Packet-dropped bit in the Port *n* Error and Status CSR. If the output port “heals”, the Error Rate Counter falls below the Error Rate Failed Threshold, the output port shall continue to attempt to forward all packets.	Switch Device Only
1	0	The port shall stop attempting to send packets to the connected device when the Output Failed-encountered bit is set. The output port will congest.	All devices.
1	1	The port shall discard all output packets without attempting to send when the port’s Output Failed-encountered bit is set. Upon discarding a packet, the port shall set Output Packet-dropped bit in the Port *n* Error and Status CSR.	All devices.

1.1.5 Packet Timeout Mechanism in a Switch Device

In some systems, it is either desirable or necessary to bound the length of time a packet can remain in a switch. To enable this functionality, a switch shall monitor the length of time each packet accepted by one of its ports has been in the switch. The acceptance of a packet by a port is signaled by the port issuing a packet-accepted control symbol for the packet. The timing begins when the port accepts the packet.

If a packet remains in a switch longer than the Time-to-Live time specified by the Time-to-Live field of the Packet Time-to-Live CSR as defined in Section 2.2.2.9: "Packet Time-to-live CSR (Block Offset 0x28 Word 1)", the packet shall be discarded rather than forwarded, the Output Packet-Dropped bit shall be set in the Port *n* Error and Status CSR and the system shall be notified as described in Section : "8.3 System Software Notification of Error".

1.2 Logical and Transport Layer Extensions

While the RapidIO link may be working properly, an end point processing element may encounter logical or transport layer errors, or other errors unrelated to its RapidIO ports, while trying to complete a transaction. The “ERROR” status response transaction is the mechanism for the target device to indicate to the source that there is a problem completing the request. Experiencing a time-out waiting for a response is also a symptom of an end point or switch fabric with a problem. These types of errors are logged and reporting enabled with a set of registers that are separate from those used for the Physical Layer errors.

- Logical/Transport Layer Error Detect CSR defined in Section 2.2.2.2: "Logical/Transport Layer Error Detect CSR (Block Offset 0x08 Word 0)"
- Logical/Transport Layer Error Enable CSR defined in Section 2.2.2.3: "Logical/Transport Layer Error Enable CSR (Block Offset 0x08 Word 1)"
- Logical/Transport Layer Capture CSRs defined in Section 2.2.2.4: "Logical/Transport Layer High Address Capture CSR (Block Offset 0x10 Word 0)" to Section 2.2.2.7: "Logical/Transport Layer Control Capture CSR

(Block Offset 0x18 Word 1)"

1.2.1 Logical/Transport Error Detect, Enable and Capture CSRs

When a logical or transport layer error is detected, the appropriate error bit shall be set by the hardware in the Logical/Transport Layer Error Detect CSR. If the corresponding bit is also set in the Logical/Transport Layer Error Enable CSR, the detect register shall lock, the appropriate information is saved in the Logical/Transport Layer Capture registers, all resources held by the transaction are freed, and system software is notified of the error as described in Section : "8.3 System Software Notification of Error". If multiple enabled errors occur during the same clock cycle, multiple bits will be set in the detect register and the contents of the Logical/Transport Layer Capture registers are implementation dependent. Once locked, subsequent errors will not set another error detect bit. The contents of the Logical/Transport Capture CSRs are valid if the bitwise AND of the Logical/Transport Layer Error Detect CSR and the Logical/Transport Layer Error Detect Enable CSR is not equal to zero (0x00000000).

Software shall write the Logical/Transport Detect register with all logic 0s to clear the error detect bits or a corresponding enable bit to unlock the register. Any other recovery actions associated with these types of errors are system dependent and outside the scope of this specification.

1.2.2 Message Passing Error Detection

Message passing is a special case of logical layer error recovery requiring error detection at both the source and destination ends of the message. The source of the message has the request-to-response time-out (defined in the Port Response Time-out Control CSR in the RapidIO Physical Layer specifications) to detect lost request or response packets in the switch fabric. However, in order to not hang the recipient mailbox in the case of a lost request packet for a multiple packet message, the recipient mailbox shall have an analogous response-to-request time-out. This time-out is for sending a response packet to receiving the next request packet of a given message operation, and has the same value as the request-to-response time-out that is already specified. The Logical/Transport Layer Control Capture CSR contains the ‘msg info’ field to capture the critical information of the last received (or sent) message segment before time-out.

1.3 System Software Notification of Error

System software is notified of logical, transport, and physical layer errors in two ways. An interrupt is issued to the local system by an end point device, the method of which is not defined in this specification, or a Maintenance port-write operation is issued by a switch device. Maintenance port-write operations are sent to a predetermined system host (defined in the Port-write Target deviceID CSR in Section 2.2.2.8: "Port-write Target deviceID CSR (Block Offset 0x28 Word 0)"). The sending device sets the Port-write Pending status bit in the Port *n* Error and Status CSR. A 16 byte data payload of the Maintenance Port-write packet contains the contents of several CSRs, the port on the device that encountered the error condition (for port-based errors), and some optional implementation specific additional information as shown in Table . Software indicates that it has seen the port-write operation by clearing the Port-write Pending status bit.

The Component Tag CSR is defined in the *RapidIO Partition III: Common Transport Specification*, and is used to uniquely identify the reporting device within the system. A Port ID field, the Logical/Transport Layer Detect CSR defined in Section 2.2.2.2: "Logical/Transport Layer Error Detect CSR (Block Offset 0x08 Word 0)", and the Port *n* Error Detect CSR defined in Section 2.2.2.10: "Port n Error Detect CSR (Block Offset 0x40, 80,..., 400, Word 0)" are used to describe the encountered error condition.

Table 1-2. Port-write Packet Data Payload for Error Reporting

Data Payload Byte Offset	Word 0		Word 1
0x0	Component Tag CSR		Port *n* Error Detect CSR
0x8	Implementation specific	Port ID (byte)	Logical/Transport Layer Error Detect CSR

1.4 Mechanisms for Software Debug

In most systems, it is difficult to verify the error handling software. The Error management extensions make some registers writable for easier debug.

The Logical/Transport Layer Error Detect register and the Logical/Transport Layer Error Capture registers are writable by software to allow software debug of the system error recovery mechanisms. For software debug, software must write the Logical/Transport Layer Error capture registers with the desired address and device id information then write the Logical/Transport Layer Error Detect register to set an error bit and lock the registers. When an error detect bit is set, the hardware will inform the system software of the error using its standard error reporting mechanism. After the error has been reported, the system software may read and clear registers as necessary to complete its error handling protocol testing.

The Port *n* Error Detect register and the Port *n* Error Capture registers are also writable by software to allow software debug of the system error recovery and thresholding mechanism. For debug, software must write the Port *n* Attributes Error Capture CSR to set the Capture Valid Info bit and then the packet/control symbol information in the other capture registers. Each write of a non-zero value to the Port *n* Error Detect CSR shall cause the Error Rate Counter to increment if the corresponding error bit is enabled in the Port *n* Error Rate Enable CSR. When a threshold is reached, the hardware will inform the system software of the error using its standard error reporting mechanism. After the error has been reported, the system software may read and clear registers as necessary to complete its error handling protocol testing.

2 Chapter 2- Error Management Registers

This section describes the Error Management Extended Features block, and adds a number of new bits to the existing standard physical layer registers. ‘End-point only’ and ‘switch only’ register bits shall be considered reserved when the registers are implemented on devices for which these bits are not required.

2.1 Additions to Existing Registers

The following bits are added to the parallel and serial logical layer specification Port *n* Control CSRs.

Table 2-1. Bit Settings for Port *n* Control CSRs

Bit	Name	Reset Value	Description
28	Stop on Port Failed-encountered Enable	0b0	This bit is used with the Drop Packet Enable bit to force certain behavior when the Error Rate Failed Threshold has been met or exceeded. See Section 1.1.4: "Port Behavior When Error Rate Failed Threshold is Reached" of the Partition VIII: Error Management Extensions for detailed requirements.
29	Drop Packet Enable	0b0	This bit is used with the Stop on Port Failed-encountered Enable bit to force certail behavior when the Error Rate Failed Threshold has been met or exceeded. See Section 1.1.4: "Port Behavior When Error Rate Failed Threshold is Reached" of the Partition VIII: Error Management Extensions for detailed requirements.
30	Port Lockout	0b0	When this bit is cleared, the packets that may be received and issued are controlled by the state of the Output Port Enable and Input Port Enable bits in the Port *n* Control CSR. When this bit is set, this port is stopped and is not enabled to issue or receive any packets; the input port can still follow the training procedure and can still send and respond to link-requests; all received packets return packet-not-accepted control symbols to force an error condition to be signaled by the sending device

The following bits are added to the parallel and serial specification Port *n* Error and Status CSRs.

Table 2-2. Bit Settings for Port *n* Error and Status CSRs

Bit	Name	Reset Value	Description
5	Output Packet-dropped	0b0	Output port has discarded a packet. Once set remains set until written with a logic 1 to clear.
6	Output Failed-encountered	0b0	Output port has encountered a failed condition, meaning that the port's failed error threshold has been reached in the Port *n* Error Rate Threshold register. Once set remains set until written with a logic 1 to clear.
7	Output Degraded-encountered	0b0	Output port has encountered a degraded condition, meaning that the port's degraded error threshold has been reached in the Port *n* Error Rate Threshold register. Once set remains set until written with a logic 1 to clear.

2.2 New Error Management Registers

This section describes the Extended Features block (EF_ID=0h0007) that allows an external processing element to manage the error status and reporting for a processing element. This chapter only describes registers or register bits defined by this extended features block. All registers are 32-bits and aligned to a 32-bit boundary.

Table describes the required behavior for accesses to reserved register bits and reserved registers for the RapidIO Extended Features register space,

Table 2-3. Extended Feature Space Reserved Access Behavior

Byte Offset	Space Name	Item	Initiator behavior	Target behavior
0x100–FFFC	Extended Features Space	Reserved bit	read - ignore returned value[1]	read - return logic 0
			write - preserve current value[2]	write - ignored
		Implementation-defined bit	read - ignore returned value unless implementation-defined function understood	read - return implementation-defined value
			write - preserve current value if implementation-defined function not understood	write - implementation-defined
		Reserved register	read - ignore returned value	read - return logic 0s
			write -	write - ignored

1. Do not depend on reserved bits being a particular value; use appropriate masks to extract defined bits from the read value.
2. All register writes shall be in the form: read the register to obtain the values of all reserved bits, merge in the desired values for defined bits to be modified, and write the register, thus preserving the value of all reserved bits.

2.2.1 Register Map

Table Table shows the register map for the error management registers. This register map is currently only defined for devices with up to 16 RapidIO ports, but can be extended or shortened if more or less port definitions are required for a device. For example, a device with four RapidIO ports is only required to use register map space corresponding to offsets [EF_PTR+0x00] through [EF_PTR+0x138]. Register map offset [EF_PTR+0x140] can be used for another Extended Features block.

Table 2-4. Error Reporting Register Map

Block Byte Offset	Register Name (Word 0)	Register Name (Word 1)
0x0	Error Reporting Block Header	Reserved
0x8	Logical/Transport Layer Error Detect CSR	Logical/Transport Layer Error Enable CSR
0x10	Logical/Transport Layer High Address Capture CSR	Logical/Transport Layer Address Capture CSR
0x18	Logical/Transport Layer Device ID CSR	Logical/Transport Layer Control Capture CSR
0x20	Reserved	
0x28	Port-write Target deviceID CSR	Packet Time-to-live CSR
0x30	Reserved	
0x38	Reserved	
0x40	Port 0 Error Detect CSR	Port 0 Error Rate Enable CSR
0x48	Port 0 Attributes Error Capture CSR	Port 0 Packet/Control Symbol Error Capture CSR 0
0x50	Port 0 Packet Error Capture CSR 1	Port 0 Packet Error Capture CSR 2
0x58	Port 0 Packet Error Capture CSR 3	Reserved
0x60	Reserved	
0x68	Port 0 Error Rate CSR	Port 0 Error Rate Threshold CSR
0x70	Reserved	
0x78	Reserved	
0x80	Port 1 Error Detect CSR	Port 1 Error Rate Enable CSR
0x88	Port 1 Error Capture Attributes CSR	Port 1 Packet/Control Symbol Error Capture CSR 0
0x90	Port 1 Packet Error Capture CSR 1	Port 1 Packet Error Capture CSR 2
0x98	Port 1 Packet Error Capture CSR 3	Reserved
0xA0	Reserved	
0xA8	Port 1 Error Rate CSR	Port 1 Error Rate Threshold CSR
0xB0	Reserved	
0xB8	Reserved	
0xC0–3F8	Assigned to Port 2-14 CSRs	
0x400	Port 15 Error Detect CSR	Port 15 Error Rate Enable CSR
0x408	Port 15 Error Capture Attributes CSR	Port 15 Packet/Control Symbol Error Capture CSR 0
0x410	Port 15 Packet Error Capture CSR 1	Port 15 Packet Error Capture CSR 2
0x418	Port 15 Packet Error Capture CSR 3	Reserved

Table 2-4. Error Reporting Register Map(Continued)

Block Byte Offset	Register Name (Word 0)	Register Name (Word 1)
0x420	Reserved	
0x428	Port 15 Error Rate CSR	Port 15 Error Rate Threshold CSR
0x430	Reserved	
0x438	Reserved	

2.2.2 Command and Status Registers (CSRs)

Refer to Table for the required behavior for access to reserved registers and register bits.

2.2.2.1 Error Reporting Block Header (Block Offset 0x0 Word 0)

The error reporting block header register contains the EF_PTR to the next EF_BLK and the EF_ID that identifies this as the error reporting block header.

Table 2-5. Bit Settings for Port Maintenance Block Header

Bit	Name	Reset Value	Description
0-15	EF_PTR		Hard wired pointer to the next block in the data structure, if one exists
16-31	EF_ID	0x0007	Hard wired Extended Features ID

2.2.2.2 Logical/Transport Layer Error Detect CSR (Block Offset 0x08 Word 0)

This register indicates the error that was detected by the Logical or Transport logic layer. Multiple bits may get set in the register if simultaneous errors are detected during the same clock cycle that the errors are logged.

Table 2-6. Bit Settings for Logical/Transport Layer Error Detect CSR

Bit	Name	Reset Value	Description
0	IO error response	0b0	Received a response of ‘ERROR’ for an IO Logical Layer Request. (end point device only)
1	Message error response	0b0	Received a response of ‘ERROR’ for an MSG Logical Layer Request. (end point device only)
2	GSM error response	0b0	Received a response of ‘ERROR’ for a GSM Logical Layer Request. (end point device only)
3	Message Format Error	0b0	Received MESSAGE packet data payload with an invalid size or segment (MSG logical) (end point device only)
4	Illegal transaction decode	0b0	Received illegal fields in the request/response packet for a supported transaction (IO/MSG/GSM logical) (switch or endpoint device)
5	Illegal transaction target error	0b0	Received a packet that contained a destination ID that is not defined for this end point. End points with multiple ports and a built-in switch function may not report this as an error (Transport) (end point device only)
6	Message Request Time-out	0b0	A required message request has not been received within the specified time-out interval (MSG logical) (end point device only)

Table 2-6. Bit Settings for Logical/Transport Layer Error Detect CSR

Bit	Name	Reset Value	Description
7	Packet Response Time-out	0b0	A required response has not been received within the specified time out interval (IO/MSG/GSM logical) (end point device only)
8	Unsolicited Response	0b0	An unsolicited/unexpected Response packet was received (IO/MSG/GSM logical; only Maintenance response for switches) (switch or endpoint device)
9	Unsupported Transaction	0b0	A transaction is received that is not supported in the Destination Operations CAR (IO/MSG/GSM logical; only Maintenance port-write for switches) (switch or endpoint device)
10-23	—		Reserved
24-31	Implementation Specific error	0x00	An implementation specific error has occurred. (switch or end point device)

2.2.2.3 Logical/Transport Layer Error Enable CSR (Block Offset 0x08 Word 1)

This register contains the bits that control if an error condition locks the Logical/Transport Layer Error Detect and Capture registers and is reported to the system host.

Table 2-7. Bit Settings for Logical/Transport Layer Error Enable CSR

Bit	Name	Reset Value	Description
0	IO error response enable	0b0	Enable reporting of an IO error response. Save and lock original request transaction capture information in all Logical/Transport Layer Capture CSRs. (end point device only)
1	Message error response enable	0b0	Enable reporting of a Message error response. Save and lock transaction capture information in all Logical/Transport Layer Capture CSRs. (end point device only)
2	GSM error response enable	0b0	Enable reporting of a GSM error response. Save and lock original request address in Logical/Transport Layer Address Capture CSRs. Save and lock transaction capture information in all Logical/Transport Layer Device ID and Control CSRs. (end point device only)
3	Message Format Error enable	0b0	Enable reporting of a message format error. Save and lock transaction capture information in Logical/Transport Layer Device ID and Control Capture CSRs. (end point device only)
4	Illegal transaction decode enable	0b0	Enable reporting of an illegal transaction decode error Save and lock transaction capture information in Logical/Transport Layer Device ID and Control Capture CSRs. (switch or end-point device)
5	Illegal transaction target error enable	0b0	Enable reporting of an illegal transaction target error. Save and lock transaction capture information in Logical/Transport Layer Device ID and Control Capture CSRs. (end point device only)
6	Message Request time-out enable	0b0	Enable reporting of a Message Request time-out error. Save and lock transaction capture information in Logical/Transport Layer Device ID and Control Capture CSRs for the last Message request segment packet received. (end point device only)

Table 2-7. Bit Settings for Logical/Transport Layer Error Enable CSR

Bit	Name	Reset Value	Description
7	Packet Response Time-out error enable	0b0	Enable reporting of a packet response time-out error. Save and lock original request address in Logical/Transport Layer Address Capture CSRs. Save and lock original request Destination ID in Logical/Transport Layer Device ID Capture CSR. (end point device only)
8	Unsolicited Response error enable	0b0	Enable reporting of an unsolicited response error. Save and lock transaction capture information in Logical/Transport Layer Device ID and Control Capture CSRs. (switch or end-point device)
9	Unsupported Transaction error enable	0b0	Enable reporting of an unsupported transaction error. Save and lock transaction capture information in Logical/Transport Layer Device ID and Control Capture CSRs. (switch or end-point device)
10-23	—		Reserved
24-31	Implementation Specific error enable	0x00	Enable reporting of an implementation specific error has occurred. Save and lock capture information in appropriate Logical/Transport Layer Capture CSRs.

2.2.2.4 Logical/Transport Layer High Address Capture CSR (Block Offset 0x10 Word 0)

This register contains error information. It is locked when a Logical/Transport error is detected and the corresponding enable bit is set. This register is only required for end point devices that support 66 or 50 bit addresses.

Table 2-8. Bit Settings for Logical/Transport Layer High Address Capture CSR

Bit	Name	Reset Value	Description
0-31	address[0-31]	All 0s	Most significant 32 bits of the address associated with the error (for requests, for responses if available)

2.2.2.5 Logical/Transport Layer Address Capture CSR (Block Offset 0x10 Word 1)

This register contains error information. It is locked when a Logical/Transport error is detected and the corresponding enable bit is set.

Figure 2-9. Bit Settings for Logical/Transport Layer Address Capture CSR

Bit	Name	Reset Value	Description
0-28	address[32-60]	All 0s	Least significant 29 bits of the address associated with the error (for requests, for responses if available)
29	—		Reserved
30-31	xamsbs	0b00	Extended address bits of the address associated with the error (for requests, for responses if available)

2.2.2.6 Logical/Transport Layer Device ID Capture CSR (Block Offset 0x18 Word 0)

This register contains error information. It is locked when an error is detected and the corresponding enable bit is set.

Table 2-10. Bit Settings for Logical/Transport Layer Device ID Capture CSR

Bit	Name	Reset Value	Description
0-7	MSB destinationID	0x00	Most significant byte of the destinationID associated with the error (large transport systems only)
8-15	destinationID	0x00	The destinationID associated with the error
16-23	MSB sourceID	0x00	Most significant byte of the sourceID associated with the error (large transport systems only)
24-31	sourceID	0x00	The sourceID associated with the error

2.2.2.7 Logical/Transport Layer Control Capture CSR (Block Offset 0x18 Word 1)

This register contains error information. It is locked when a Logical/Transport error is detected and the corresponding enable bit is set.

Table 2-11. Bit Settings for Logical/Transport Layer Header Capture CSR 1

Bit	Name	Reset Value	Description
0-3	ftype	0x0	Format type associated with the error
4-7	ttype	0x0	Transaction type associated with the error
8-15	msg info	0x00	letter, mbox, and msgseg for the last Message request received for the mailbox that had an error (Message errors only)
16-31	Implementation specific	0x0000	Implementation specific information associated with the error

2.2.2.8 Port-write Target deviceID CSR (Block Offset 0x28 Word 0)

This register contains the target deviceID to be used when a device generates a Maintenance port-write operation to report errors to a system host.

Table 2-12. Bit Settings for Port-write Target deviceID CSR

Bit	Name	Reset Value	Description
0-7	deviceID_msb	0x00	This is the most significant byte of the port-write target deviceID (large transport systems only)
8-15	deviceID	0x00	This is the port-write target deviceID
16-31	—		Reserved

2.2.2.9 Packet Time-to-live CSR (Block Offset 0x28 Word 1)

The Packet Time-to-live register specifies the length of time that a packet is allowed to exist within a switch device. The maximum value of the Time-to-live variable (0xFFFF) shall correspond to 100 msec. +/-34%. The resolution (minimum step size) of the Time-to-live variable shall be (maximum value of Time-to-live)/(2^{16}-1). The reset value is all logic 0s, which disables the Time-to-live function so that a packet never times out. This register is not required for devices without switch functionality.

Table 2-13. Bit Settings for Packet Time-to-live CSR

Bit	Name	Reset Value	Description
0-15	Time-to-live value	0x0000	Maximum time that a packet is allowed to exist within a switch device
16-31	—		Reserved

2.2.2.10 Port *n* Error Detect CSR (Block Offset 0x40, 80,..., 400, Word 0)

The Port *n* Error Detect Register indicates transmission errors that are detected by the hardware.

Table 2-14. Bit Settings for Port *n* Error Detect CSR

Bit	Name	Reset Value	Description
0	Implementation specific error	0b0	An implementation specific error has been detected
1-7	—		Reserved
8	Received S-bit error	0b0	Received a packet/control symbol with an S-bit parity error (parallel)
9	Received corrupt control symbol	0b0	Received a control symbol with a bad CRC value (serial) Received a control symbol with a true/complement mismatch (parallel)
10	Received acknowledge control symbol with unexpected ackID	0b0	Received an acknowledge control symbol with an unexpected ackID (packet-accepted or packet_retry)
11	Received packet-not-accepted control symbol	0b0	Received packet-not-accepted acknowledge control symbol
12	Received packet with unexpected ackID	0b0	Received packet with unexpected ackID value - out-of-sequence ackID
13	Received packet with bad CRC	0b0	Received packet with a bad CRC value
14	Received packet exceeds 276 Bytes	0b0	Received packet which exceeds the maximum allowed size
15-25	—		Reserved
26	Non-outstanding ackID	0b0	Link_response received with an ackID that is not outstanding
27	Protocol error	0b0	An unexpected packet or control symbol was received
28	Frame toggle edge error	0b0	FRAME signal toggled on falling edge of receive clock (parallel)
29	Delineation error	0b0	FRAME signal toggled on non-32-bit boundary (parallel) Received unaligned /SC/ or /PD/ or undefined code-group (serial)
30	Unsolicited acknowledge control symbol	0b0	An unexpected acknowledge control symbol was received
31	Link time-out	0b0	An acknowledge or link-response control symbol is not received within the specified time-out interval

2.2.2.11 Port *n* Error Rate Enable CSR (Block Offset 0x40, 80,..., 400, Word 1)

This register contains the bits that control when an error condition is allowed to increment the error rate counter in the Port *n* Error Rate Threshold Register and lock the Port *n* Error Capture registers.

Table 2-15. Bit Settings for Port *n* Error Rate Enable CSR

Bit	Name	Reset Value	Description
0	Implementation specific error enable	0b0	Enable error rate counting of implementation specific errors
1-7	—		Reserved
8	Received S-bit error enable	0b0	Enable error rate counting of a packet/control symbol with an S-bit parity error (parallel)
9	Received control symbol with bad CRC enable	0b0	Enable error rate counting of a corrupt control symbol

Table 2-15. Bit Settings for Port *n* Error Rate Enable CSR

Bit	Name	Reset Value	Description
10	Received out-of-sequence acknowledge control symbol enable	0b0	Enable error rate counting of an acknowledge control symbol with an unexpected ackID
11	Received packet-not-accepted control symbol enable	0b0	Enable error rate counting of received packet-not-accepted control symbols
12	Received packet with unexpected ackID enable	0b0	Enable error rate counting of packet with unexpected ackID value - out-of-sequence ackID
13	Received packet with Bad CRC enable	0b0	Enable error rate counting of packet with a bad CRC value
14	Received packet exceeds 276 Bytes enable	0b0	Enable error rate counting of packet which exceeds the maximum allowed size
15-25	—		Reserved
26	Non-outstanding ackID enable	0b0	Enable error rate counting of link-responses received with an ackID that is not outstanding
27	Protocol error enable	0b0	Enable error rate counting of protocol errors
28	Frame toggle edge error enable	0b0	Enable error rate counting of frame toggle edge errors
29	Delineation error	0b0	Enable error rate counting of delineation errors
30	Unsolicited acknowledge control symbol	0b0	Enable error rate counting of unsolicited acknowledge control symbol errors
31	Link time-out	0b0	Enable error rate counting of link time-out errors

2.2.2.12 Port *n* Error Capture Attributes CSR (Block Offset 0x48, 88,..., 408, Word 0)

The error capture attribute register indicates the type of information contained in the Port *n* error capture registers. In the case of multiple detected errors during the same clock cycle one of the errors must be reflected in the Error type field. The error that is reflected is implementation dependent.

Table 2-16. Bit Settings for Port *n* Attributes Error Capture CSR

Bit	Name	Reset Value	Description
0-1	Info type	0b00	Type of information logged 00 - packet 01 - control symbol (only error capture register 0 is valid) 10 - implementation specific (capture register contents are implementation specific) 11 - undefined (S-bit error), capture as if a packet (parallel physical layer only)
2	—		Reserved
3-7	Error type	0x00	The encoded value of the bit in the Port *n* Error Detect CSR that describes the error captured in the Port *n* Error Capture CSRs.
8-27	Implementation Dependent	All 0s	Implementation Dependent Error Information
28-30	—		Reserved

Table 2-16. Bit Settings for Port *n* Attributes Error Capture CSR

Bit	Name	Reset Value	Description
31	Capture valid info	0b0	This bit is set by hardware to indicate that the Packet/control symbol capture registers contain valid information. For control symbols, only capture register 0 will contain meaningful information.

2.2.2.13 Port *n* Packet/Control Symbol Error Capture CSR 0 (Block Offset 0x48, 88,..., 408, Word 1)

Captured control symbol information includes the true and complement of the control symbol. This is exactly what arrives on the RapidIO interface with bits 0-7 of the capture register containing the least significant byte of the 32-bit quantity. This register contains the first 4 bytes of captured packet symbol information.

Table 2-17. Bit Settings for Port *n* Packet/Control Symbol Error Capture CSR 0

Bit	Name	Reset Value	Description
0-31	Capture 0	All 0s	True and Complement of Control Symbol (parallel) or Control Character and Control Symbol (serial) or Bytes 0 to 3 of Packet Header

2.2.2.14 Port *n* Packet Error Capture CSR 1 (Block Offset 0x50, 90,..., 410, Word 0)

Error capture register 1 contains bytes 4 through 7 of the packet header.

Table 2-18. Bit Settings for Port *n* Packet Error Capture CSR 1

Bit	Name	Reset Value	Description
0-31	Capture 1	All 0s	Bytes 4 thru 7 of the packet header.

2.2.2.15 Port *n* Packet Error Capture CSR 2 (Block Offset 0x50, 90,..., 410, Word 1)

Error capture register 2 contains bytes 8 through 11 of the packet header.

Table 2-19. Bit Settings for Port *n* Packet Error Capture CSR 2

Bit	Name	Reset Value	Description
0-31	Capture 2	All 0s	Bytes 8 thru 11of the packet header.

2.2.2.16 Port *n* Packet Error Capture CSR 3 (Block Offset 0x58, 98,..., 418, Word 0)

Error capture register 3 contains bytes 12 through 15 of the packet header.

Table 2-20. Bit Settings for Port *n* Packet Error Capture CSR 3

Bit	Name	Reset Value	Description
0-31	Capture 3	All 0s	Bytes 12 thru 15 of the packet header.

2.2.2.17 Port *n* Error Rate CSR (Block Offset 0x68, A8,..., 428, Word 0)

The Port *n* Error Rate register is a 32-bit register used with the Port *n* Error Rate Threshold register to monitor and control the reporting of transmission errors, shown in Table 2-21.

Table 2-21. Bit Settings for Port *n* Error Rate CSR

Bit	Name	Reset Value	Description
0-7	Error Rate Bias	0x80	These bits provide the error rate bias value 0x00 - do not decrement the error rate counter 0x01 - decrement every 1ms (+/-34%) 0x02 - decrement every 10ms (+/-34%) 0x04 - decrement every 100ms (+/-34%) 0x08 - decrement every 1s (+/-34%) 0x10 - decrement every 10s (+/-34%) 0x20 - decrement every 100s (+/-34%) 0x40 - decrement every 1000s (+/-34%) 0x80 - decrement every 10000s (+/-34%) other values are reserved
8-13	—		Reserved
14-15	Error Rate Recovery	0b00	These bits limit the incrementing of the error rate counter above the failed threshold trigger. 0b00 - only count 2 errors above 0b01 - only count 4 errors above 0b10 - only count 16 error above 0b11 - do not limit incrementing the error rate count
16-23	Peak Error Rate	0x00	This field contains the peak value attained by the error rate counter.
24-31	Error Rate Counter	0x00	These bits maintain a count of the number of transmission errors that have been detected by the port, decremented by the Error Rate Bias mechanism, to create an indication of the link error rate.

2.2.2.18 Port *n* Error Rate Threshold CSR (Block Offset 0x68, A8,..., 428, Word 1)

The Port *n* Error Rate Threshold register is a 32-bit register used to control the reporting of the link status to the system host.

Table 2-22. Bit Settings for Port *n* Error Rate Threshold CSR

Bit	Name	Reset Value	Description
0-7	Error Rate Failed Threshold Trigger	0xFF	These bits provide the threshold value for reporting an error condition due to a possibly broken link. 0x00 - Disable the Error Rate Failed Threshold Trigger 0x01 - Set the error reporting threshold to 1 0x02 - Set the error reporting threshold to 2 ... 0xFF - Set the error reporting threshold to 255
8-15	Error Rate Degraded Threshold Trigger	0xFF	These bits provide the threshold value for reporting an error condition due to a degrading link. 0x00 - Disable the Error Rate Degraded Threshold Trigger 0x01 - Set the error reporting threshold to 1 0x02 - Set the error reporting threshold to 2 ... 0xFF - Set the error reporting threshold to 255
16-31	—		Reserved

Annex A

(Informative)

Error Management

This section is intended to provide useful information/background on the application of the error management capabilities. This section is a guideline, not part of the specification.

A.1 Limitations of Error Management

The RapidIO hardware that implements the Error Management extensions is able to log transmission errors and errors that occur at a higher level. Some error scenarios require no software intervention and recovery procedures are done totally by the hardware.

Some error scenarios detected require fault management software for recovery to be successful. For example, some types of logical layer errors on a Read or Write operation may be recoverable by killing the software process using the affected memory space and removing the memory space from the available system resource pool. It may also be possible for software to retry the operation, possibly through a different path in the switch fabric. Since such fault management software is typically tightly coupled to a particular system and/or implementation, it is considered outside of the scope of this specification.

Another area of fault recovery that requires fault management software to be implemented is correcting of system state after an error during an atomic operation. The swap style Atomic operations are possibly recoverable through software and require software convention to uniquely identify attempts to take locks. For example, if the request is lost and times out, software can examine the current lock value to determine if the request or the associated response was the transaction that was lost in the switch fabric. For all other Atomic operations (such as the Atomic set operation), it is impossible to correct the system state in the presence of a ‘lost packet’ type of error.

The use of RapidIO message packets relies on the use of higher layer protocols for error management. Since end points that communicate via messaging are typically running a variety of higher layer protocols, error reporting of both request and response time-outs is done locally by the message queue management controller. Note that side effect errors can occur, for example, ERROR responses or RETRY responses during an active (partially completed) message, which may complicate the recovery procedure. The recovery strategies for messages lost in this manner are outside of the scope of this specification.

Globally Shared Memory systems that encounter a logical or transport layer error are typically not recoverable by any mechanism as this usually means that the processor caches are no longer coherent with the main memory system. Historically, recovery from such errors requires a complete reboot of the machine after the component that caused the error is repaired or replaced.

A.2 Hot-insertion/extraction

Hot-insertion can be regarded as an error condition in which a new part of the system is detected, therefore, hot-insertion of a Field Replaceable Unit (FRU) can be handled utilizing the above described mechanisms. This section describes two approaches for hot insertion. The first generally applies to high availability systems, or systems where FRUs need to brought into the system in a controlled manner. The second generally applies to systems where availability is less of a concern, for example, a trusted system or a system without a system host.

At system boot time, the system host identifies all of the unattached links in the machine through system discovery and puts them in a locked mode, whereby all incoming packets are to be rejected, leaving the drivers and receivers enabled. This is done by setting the Discovered bit in the Port General Control CSR and the Port Lockout bit in the Port *n* Control CSR. Note that whenever an FRU is removed, the port lockout bit should be used to ensure that whatever new FRU is inserted cannot access the system until the system host allows it. When a FRU is hot-inserted connecting to a switch device, the now connected link will automatically start the training sequence. When training is complete (the Port OK bit in the Port *n* Error and Status CSR is now set), the locked port generates a Maintenance port-write operation to notify the system host of the new connection, and sets the Port-write Pending bit.

On receipt of the port-write, the system host is responsible for bringing the inserted FRU into the system in a controlled manner. The system host can communicate with the inserted FRU using Maintenance operations after clearing all error conditions, if any, clearing the Port Lockout bit and clearing the Output and Input Port Enable bits in the Port *n* Control CSR. This procedure allows the system host to access the inserted FRU safely, without exposing itself to incorrect behavior by the inserted FRU.

In order to issue Maintenance operations to the inserted FRU, the system host must first make sure that the ackID values for both ends are consistent. Since the inserted FRU has just completed a power-up reset sequence, both it's transmit and receive ackID values are the reset value of 0x00. The system host can set the switch device's transmit and receive ackID values to also be 0x00 through the Port *n* Local ackID Status CSR if they are not already in that state, and can then issue packets normally.

The second method for hot insertion would allow the replaced FRU to bring itself into the system, which is necessary for a system in which the FRU is the system host itself. In this approach, the Port Lockout bit is not set and instead the Output and Input Port Enable bits are set for any unconnected port, allowing inserted FRUs free access to the system without reliance on a system host. Also, a port-write operation is not generated when the training sequence completes and the link is active, so a host is not notified of the event. However, this method leaves the system vulnerable to corruption from a misbehaving hot-inserted FRU.

As with the first case, the system host must make the ackID values for both link partners match in order to begin sending packets. In order to accomplish this, the system host generates a link-request/link-status to the attached device to obtain it's expected receiver value using the Port *n* Link Maintenance Request and Response CSRs. It can then set its transmit ackID value to match. Next, the system host generates a Maintenance write operation to set the attached device's Port *n* Local ackID Status CSR to set the transmit ackID value to match the receive ackID value in the system host. Upon receipt of the maintenance write, the attached device sets it's transmit ackID value as instructed, and generates the maintenance response using the new value. Packet transmission can now proceed normally.

Hot extraction from a port's point of view behaves identically to a very rapidly failing link and therefore can utilize the above described error reporting mechanism. Hot extraction is ideally done in a controlled fashion by taking the FRU to be removed out of the system as a usable resource through the system management software so that extraction does not cause switch fabric congestion or result in a loss of data.

The required mechanical aspects of hot-insertion and hot-extraction are not addressed in this specification.

A.3 Port-write

The error management specification includes only one destination for port-write operations, while designers of reliable systems would assume that two is the minimum number. This section explains the rationale for only having one port-write destination.

It is assumed that in the event of an error on a link that both ends of the link will see the error. Thus, there are two parties who can be reporting on any error. In the case that the sole link between an end point and a switch fails completely, the switch is expected to see and report the error. When one of a set of redundant links between an end point and a switch device fails, it is expected that the switch and possibly the end point will report the failure.

When a link between two switches fails, it is assumed that there are multiple paths to the controlling entity available for the port-write to travel. The switches will be able to send at least one, and possibly two, reports to the system host. It is assumed that it is possible to set up a switch's routing parameters such that the traffic to the system host will follow separate paths from each switch.

In some reliable systems, the system host is implemented as multiple redundant subsystems. It is assumed in RapidIO that only one subsystem is actually in control at any one time, and so should be the recipient of all port-writes. If the subsystem that should be in control is detected to be insane, it is the responsibility of the rest of the control subsystem to change the destination for port-writes to be the new subsystem that is in control.

Glossary of Terms and Abbreviations

The glossary contains an alphabetical list of terms, phrases, and abbreviations used in this book.

D

Degraded threshold. Bits 8-15 of the Port n Error Rate Threshold CSR. An application-specific level that indicates an unacceptable error rate resulting in degraded throughput, when equal to the error rate count.

F

Failed threshold. Bits 0-7 of the Port n Error Rate Threshold CSR. An application-specific level that indicates an error rate due to a broken link, when equal to the error rate count.

H

Hot-insertion. Hot-insertion is the insertion of a processing element into a powered-up system.

Hot-extraction. Hot-extraction is the removal of a processing element from a powered-up system.

L

Logical/Transport error. A logical/transport error is one that cannot be resolved using the defined transmission error recovery sequence, results in permanent loss of data or causes system corruption. Recovery may possible under software control.

N

Non-reporting processing element. A non-reporting processing element depends upon an attached device (usually a switch) to report its logged errors to the system host on its behalf.

O

Operation. A set of transactions between end point devices in a RapidIO system (requests and associated responses) such as a read or a write.

Ownership. A processing element has the only valid copy of a coherence granule and is responsible for returning it to home memory.

P

Physical error. A physical error occurs only in the physical layer.

Port healing. The process whereby software resets the error rate count, or allows it to decrement as required by the error rate bias field of the Port n Error Rate CSR.

R

Read operation. An operation used to obtain a globally shared copy of a coherence granule.

Reporting processing element. A reporting processing element is capable of reporting its logged errors to the system host.

S

Switch processing element. One of three processing elements, a switch processing element, or switch, is capable of logging and reporting errors to the host system.

T

Transmission error. A transmission error is one that can be resolved using the defined transmission error recovery sequence, results in no permanent loss of data and does not cause system corruption. Recovery may also be possible under software control using mechanisms outside of the scope of this specification.