RapidIO™ Interconnect Specification

Rev. 1.2, 6/2002

RapidIO Trade Association

Revision History

Revision	Description	Date
1.1	First public release	03/08/2001
1.2	Technical changes: incorporate Rev. 1.1 errata rev. 1.1.1, errata 3, showing 02-02-00009	06/26/2002

Questions regarding the RapidIO Trade Association, specifications, or membership should be forwarded to:

RapidIO Trade Association
Suite 325, 3925 W. Braker Lane
Austin, TX 78759
512-305-0070 Tel.
512-305-0009 FAX.

CONTENTS

1

Part I

1

2

3

4

Part II

1

2

3

4

A

Part III

1

2

Part IV

1

2

3

4

5

6

7

8

A

GLO

IND

CONTENTS

CONTENTS

CONTENTS

CONTENTS

1

Part I

1

2

3

4

Part II

1

2

3

4

A

Part III

1

2

Part IV

1

2

3

4

5

6

7

8

A

GLO

IND

CONTENTS

CONTENTS

CONTENTS

Chapter 5
System Clocking Considerations

CONTENTS

CONTENTS

ILLUSTRATIONS

ILLUSTRATIONS

Part III
Common Transport Specification

Part IV
Physical 8/16 LP-LVDS Specification

ILLUSTRATIONS

ILLUSTRATIONS

TABLES

TABLES

Part III
Common Transport Specification

Part IV
Physical 8/16 LP-LVDS Specification

TABLES

TABLES

About This Book

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.

The original RapidIO specification was developed under a joint development agreement by Motorola Semiconductor Product Sector and Mercury Computer Systems and contributed to the RapidIO Trade Association as the Rev0.1 specification. The following individuals are recognized for their contribution to that effort. The company affiliation noted shows employment at the time of the development of the specification.

Bryan Marietta, Motorola SPS-Somerset, Chief Architect and Specification Editor
Robert Frisch, Mercury Computer Systems, Chief Architect
Daniel Bouvier, Motorola SPS-Somerset, Architect and Evangelist
Brian Young, Motorola SPS-Somerset, Signal Integrity and Modeling
Andy Boughton, Mercury Computer Systems, Architect and Signal Integrity
Craig Lund, Mercury Computer Systems, Logistical Support and Evangelist
Sam Fuller, Motorola SPS-Somerset, Logistical Support and Evangelist

Continuing development and refinement of the RapidIO specification is done within the RapidIO Trade Association Technical Working Group. The following were the TWG officers and the primary TWG representatives of the companies eligible to vote for the approval of this revision of the specification. Company affiliation noted shows employment at the time of the approval of the specification.

Greg Shippen, Motorola SPS-NCSG, Technical Working Group Chair
Bryan Marietta, Motorola SPS-Somerset, Specification Editor
Bill Quackenbush, Cisco, Technical Working Group Secretary
Joe Jacob, DY4/Ixthos
Steve MacArthur, EMC Corporation
Yoanna Baumgartner, IBM Microelectronics
Steve Horne, Intrinsity
Dave Wickliff, Lucent Technologies
Tom Morin, Mercury Computer Systems
Tom Martin, PLX Technology
Douglas Endo, Raytheon Company
Ian Colloff, RedSwitch Inc.
Jonathan Adams, Rydal Research and Development
Frank Van Hooft, Spectrum Signal Processing
Ralph Barrera, Systran
Travis Scheckel, Texas Instruments
Stephane Gagnon, Tundra Semiconductor Corp.
Jason Lawley, Xilinx, Inc.

All RapidIO specification development and definition is conducted under the guidance of the RapidIO Trade Association Steering Committee. The Steering Committee had the following membership at the time of the release of this revision of the specification.

Sam Fuller, RapidIO Trade Association President
Rick Lacerte, Nortel Networks, Steering Committee Chair
Dan Bouvier, Motorola SPS-Somerset, Steering Committee Vice-Chair
Craig Lund, Mercury Computer Systems, Steering Committee Treasurer
Gary Robinson, EMC Corporation, Steering Committee Secretary
Yves Diverres, Alcatel
Bill Quackenbush, Cisco Systems
Louis-Francois Pau, Ericsson
Thomas Collopy, IBM Microelectronics
Dave Wickliff, Lucent Technologies

Book Overview

This preface explains each of the three layers of the RapidIO architecture and their interrelationships:

- Logical layer—The logical layer defines the overall protocol and packet formats, the types of transactions that can be carried out with RapidIO, and how addressing is handled. The logical specifications are partitioned into two parts:
  - *Part I: Input/Output Logical Specification*
  - *Part II: Message Passing Logical Specification*
- 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 *Part III: Common Transport Specification.*
- Physical layer—The physical layer contains the device level interface such as packet transport mechanisms, flow control, electrical characteristics, and low-level error management. RapidIO covers these topics in *Part IV: Physical 8/16 LP-LVDS Specification.*

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.

Parts I and II: Logical Specifications

In RapidIO, the logical layer is subdivided into two specifications that support distributed I/O processing. *Part I: Input/Output Logical Specification* explains how RapidIO supports input-output systems and *Part II: Message Passing Logical Specification* describes the message passing features of the RapidIO interconnect.

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 appendix added that describes in greater detail two examples of RapidIO message passing models, one a simple model and one a more extended model.

Part III: Common Transport Specification

Part III: Common Transport Specification contains three chapters:

- The introduction to *Part 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.

Part IV: Physical 8/16 LP-LVDS Specification

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

- The introduction to *Part 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, “Signal Descriptions,” 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.
- Appendix 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.

Extensions

Extensions to this base set of RapidIO specifications will be published periodically under separate cover. The logical specifications, for example, have an extension already extant: *Part V: Globally Shared Memory Logical Specification*, for applications that support cache-coherency and multi-processing.

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*.

0b*nn*	A 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.

Overview

This overview 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.

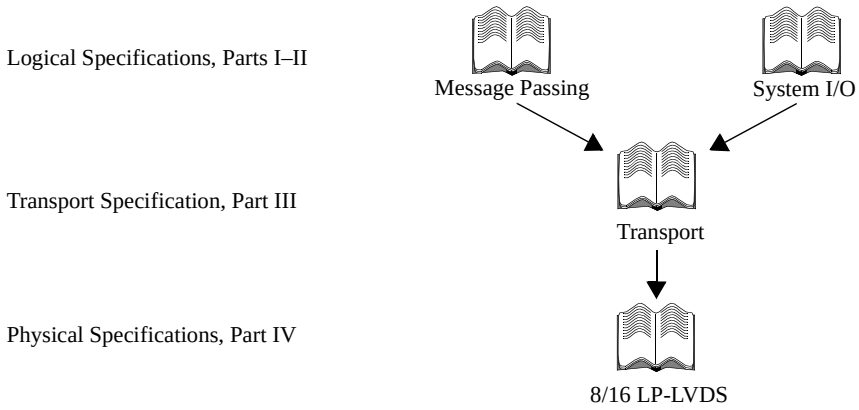


Figure i. RapidIO Layered Hierarchy

The RapidIO architecture is specified in a three-layer hierarchy (Figure i) 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 *Part I: Input/Output Logical Specification*

  — Message passing specifications in *Part 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 *Part 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 *Part 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.

RapidIO™ Interconnect Specification Part I: Input/Output Logical Specification

Rev. 1.2, 06/2002

RapidIO Trade Association

Revision History

Revision	Description	Date
1.1	First public release	03/08/2001
1.2	Technical changes: incorporate Rev. 1.1 errata rev. 1.1.1, errata 3	06/26/2002

Questions regarding the RapidIO Trade Association, specifications, or membership should be forwarded to:

RapidIO Trade Association
Suite 325, 3925 W. Braker Lane
Austin, TX 78759
512-305-0070 Tel.
512-305-0009 FAX.

Part I
Input/Output Logical Specification

Part 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 *Part 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. *Part I: Input/Output Logical Specification* defines the basic I/O system architecture of RapidIO.

I.2 Contents

Following are the contents of *Part 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.

Chapter 1
System Models

This overview introduces some possible devices in a RapidIO system.

1.1 Processing Element Models

Figure 1-1 describes a possible RapidIO-based computing system. The processing element is a computer device such as a processor attached to a local memory and 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.

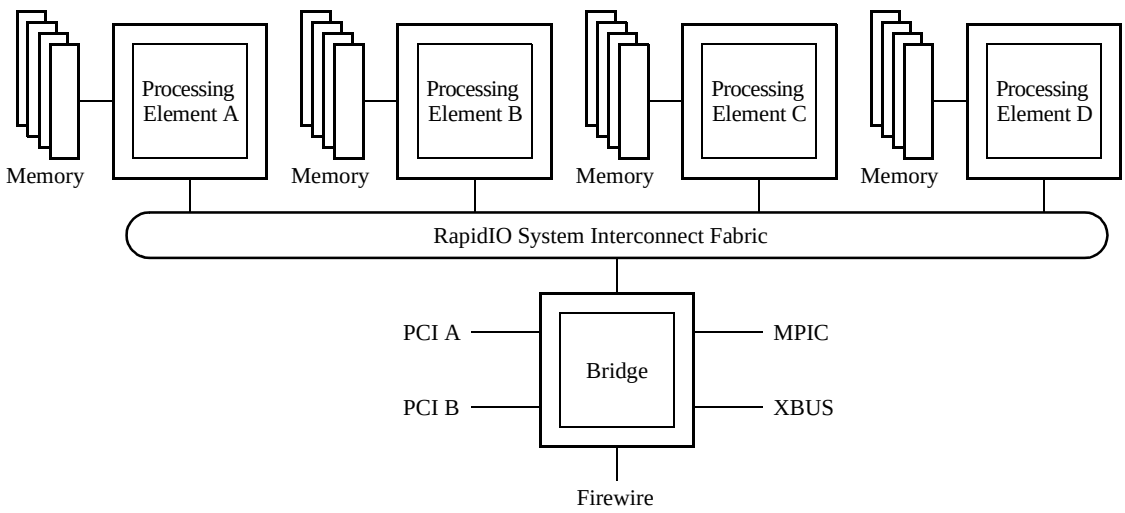


Figure 1-1. A Possible RapidIO-Based Computing System

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 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.

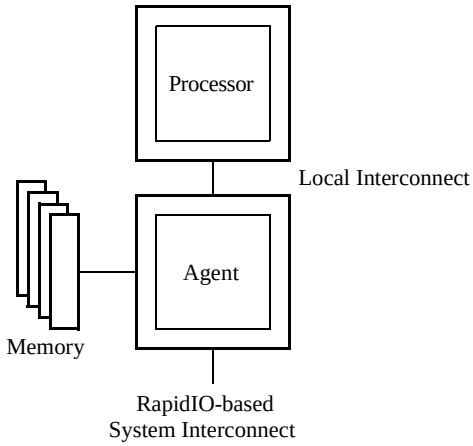


Figure 1-2. Processor-Memory Processing Element Example

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.

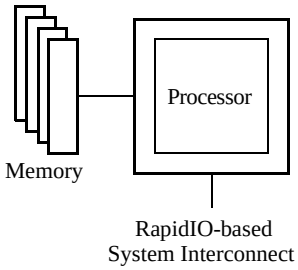


Figure 1-3. Integrated Processor-Memory Processing Element Example

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.

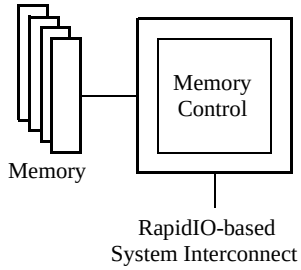


Figure 1-4. Memory-Only Processing Element Example

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.

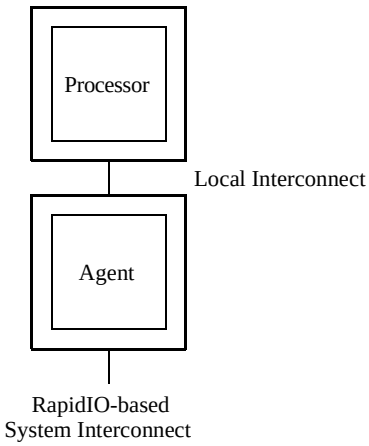


Figure 1-5. Processor-Only Processing Element Example

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 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 Figure 1-6. Behavior of the switches, and the interconnect fabric in general, is addressed in the *RapidIO Common Transport Specification*.

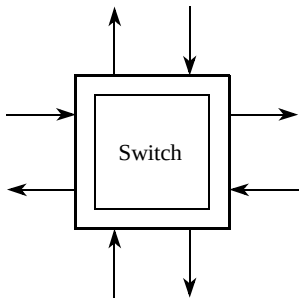


Figure 1-6. Switch Processing Element Example

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.

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 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 through Figure 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.

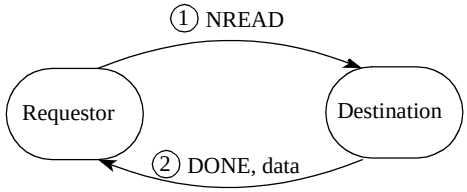


Figure 2-1. Read Operation

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.

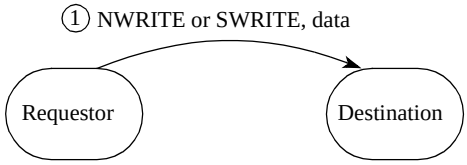


Figure 2-2. Write and Streaming-Write Operations

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.

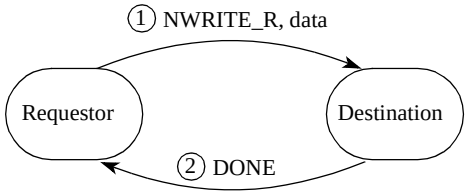


Figure 2-3. Write-With-Response Operation

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.

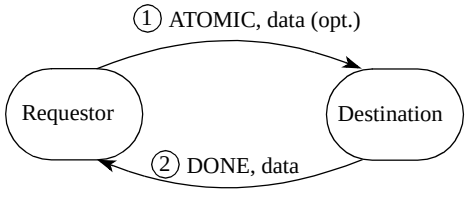


Figure 2-4. Atomic (Read-Modify-Write) Operation

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.

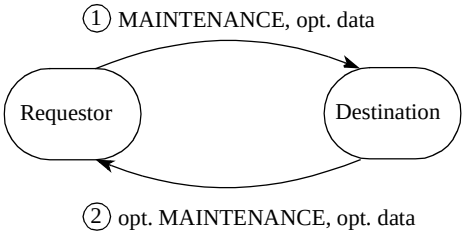
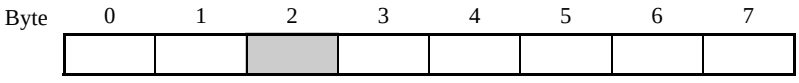


Figure 2-5. Maintenance Operation

2.4 Endian, Byte Ordering, and Alignment

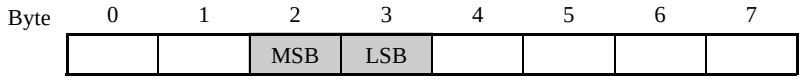
RapidIO has double-word (8-byte) aligned big-endian data payloads. This means that the RapidIO interface to devices that are little-endian shall perform the proper endian transformation to format a data payload.

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.



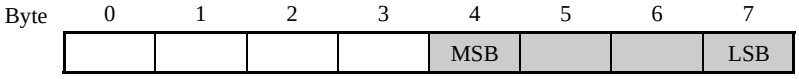
Byte address 0x0000_0002, the proper byte position is shaded.

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.

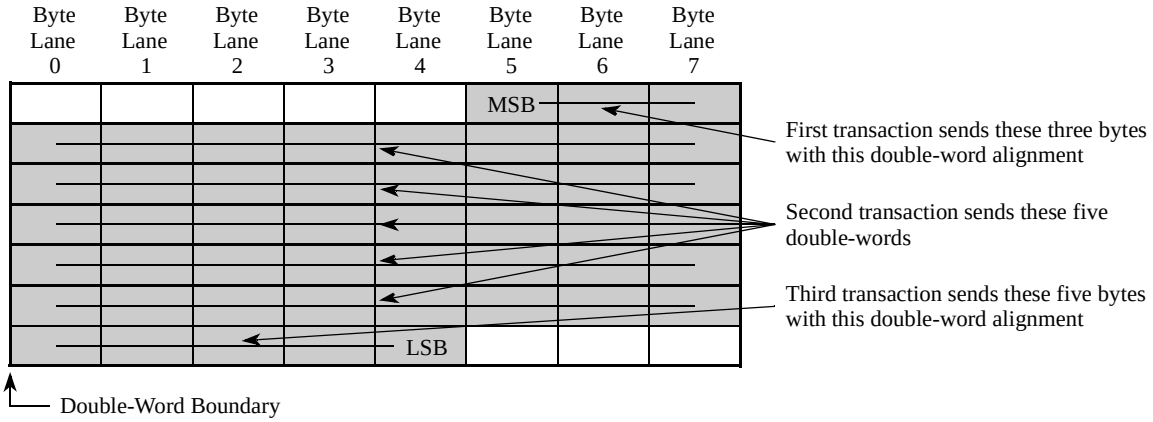


Figure 2-9. Data Alignment Example

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

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

Request Packet Format Type	Transaction Type	Definition	Document Section No.
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 unaligned 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.

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

Field	Definition
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
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

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

wdptr	rdsize	Number of Bytes	Byte Lanes
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
0b1	0b1000	4	0b00001111
0b0	0b1001	6	0b11111100
0b1	0b1001	6	0b00111111
0b0	0b1010	7	0b11111110
0b1	0b1010	7	0b01111111

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

wdptr	wrsize	Number of Bytes	Byte Lanes
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
0b1110	ATOMIC set: set the data (write 0b11111...')
0b1111	ATOMIC clr: clear the data (write 0b00000...')

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.

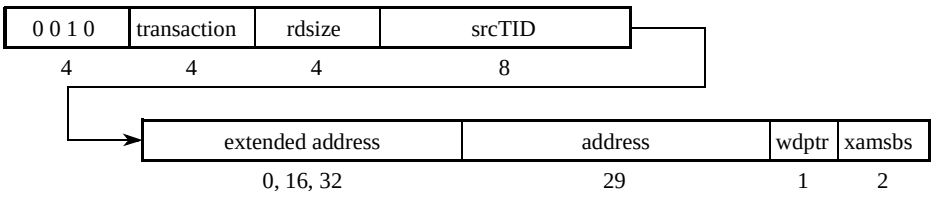


Figure 3-1. Type 2 Packet Bit Stream Format

3.1.6 Type 3–4 Packet Formats (Reserved)

The type 3–4 packet formats are reserved.

3.1.7 Type 5 Packet Format (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.

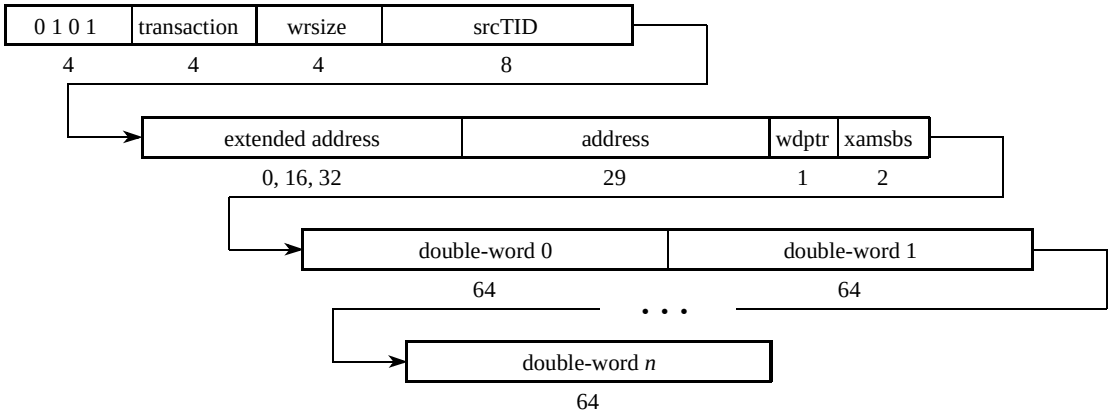


Figure 3-2. Type 5 Packet Bit Stream Format

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.

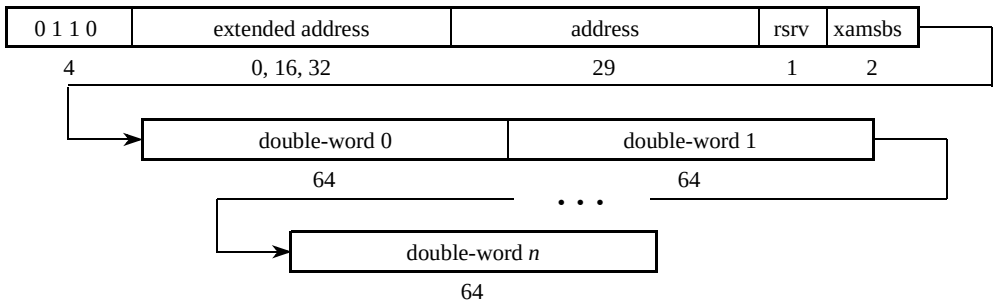


Figure 3-3. Type 6 Packet Bit Stream Format

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.

Table 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.

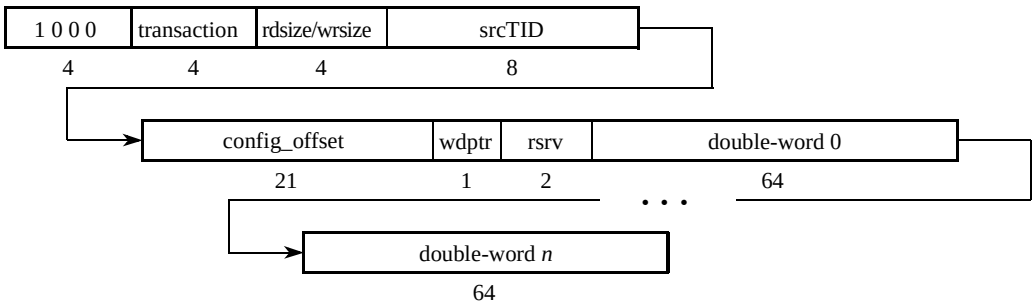


Figure 3-4. Type 8 Request Packet Bit Stream Format

Figure 3-5 displays a type 8 response packet with all its fields.

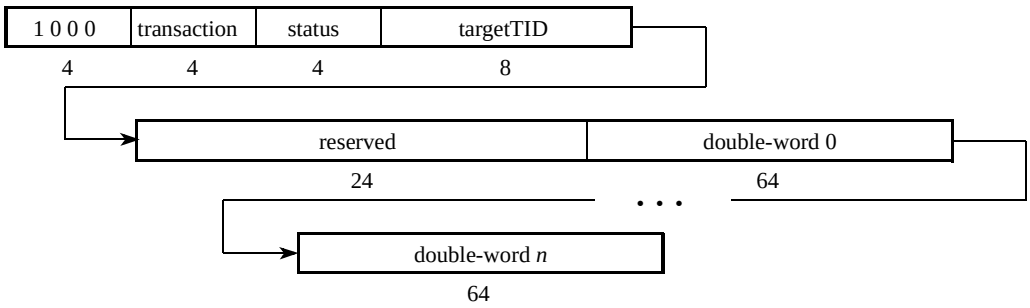


Figure 3-5. Type 8 Response Packet Bit Stream Format

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

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

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 is of type 13.

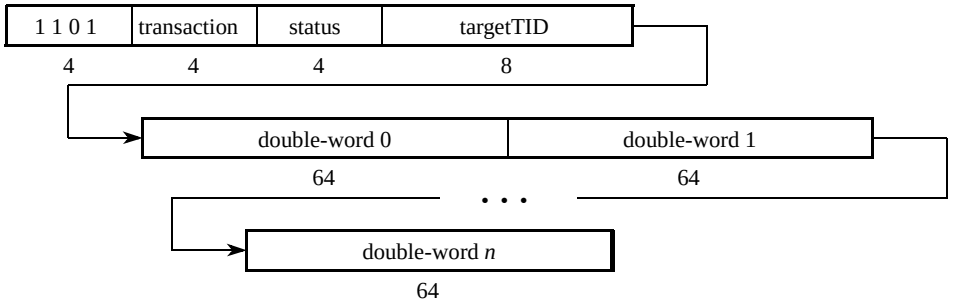


Figure 3-6. Type 13 Packet Bit Stream Format

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.

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

Table 4-1. I/O Register Map (Continued)

Configuration Space Byte Offset	Register Name (Word 0)	Register Name (Word 1)
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

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 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.

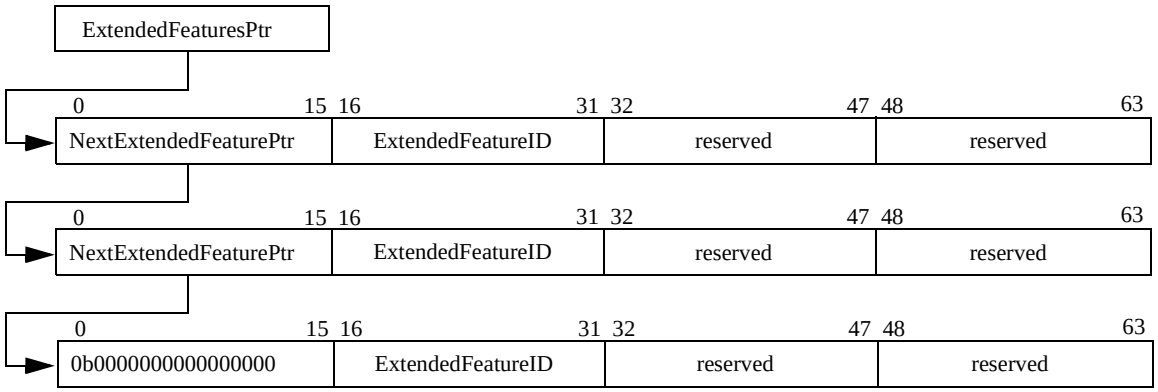


Figure 4-1. Example Extended Features Data Structure

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

Table 4-10. Bit Settings for Destination Operations CAR (Continued)

Bit	Field Name	Description
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

RapidIO™ Interconnect Specification Part II: Message Passing Logical Specification

Rev. 1.2, 06/2002

RapidIO Trade Association

Revision History

Revision	Description	Date
1.1	First public release	03/08/2001
1.2	No technical changes	06/26/2002

Questions regarding the RapidIO Trade Association, specifications, or membership should be forwarded to:

RapidIO Trade Association
Suite 325, 3925 W. Braker Lane
Austin, TX 78759
512-305-0070 Tel.
512-305-0009 FAX.

Part II
Message Passing Logical Specification

Part 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 *Part 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 *Part 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.

Part II

Chapter 1
System Models

This overview introduces some possible devices in a RapidIO system.

1.1 Processing Element Models

Figure 1-1 describes 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.

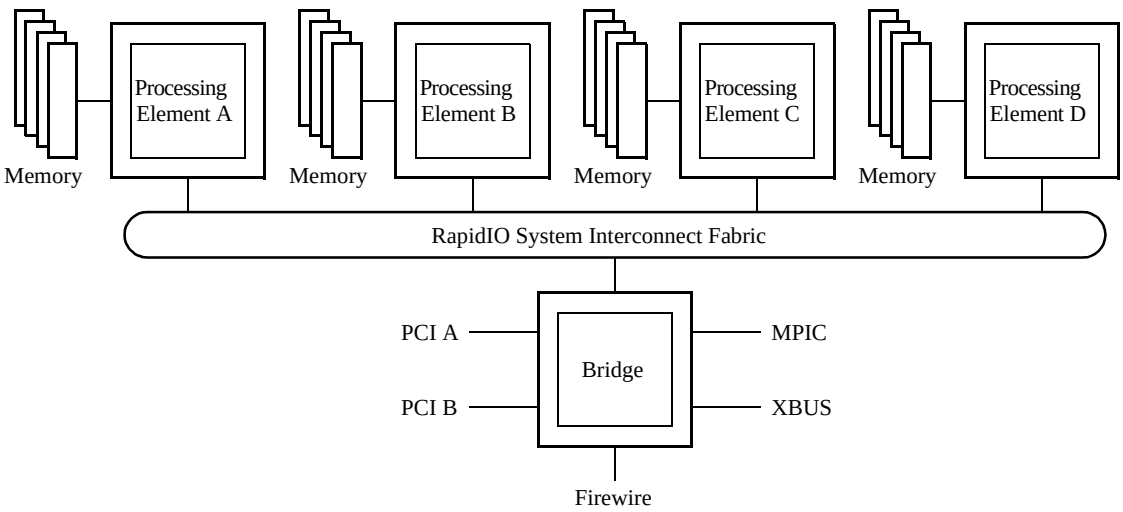


Figure 1-1. A Possible RapidIO-Based Computing System

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.

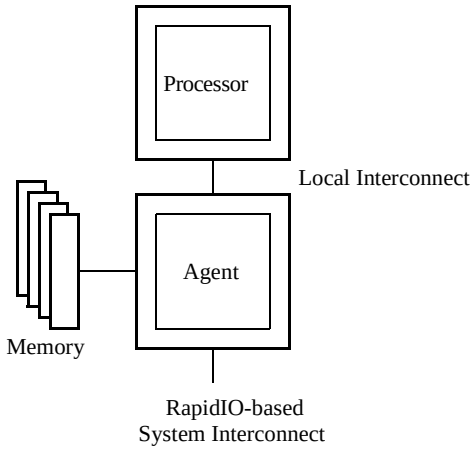


Figure 1-2. Processor-Memory Processing Element Example

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.

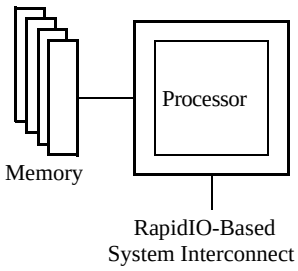


Figure 1-3. Integrated Processor-Memory Processing Element Example

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.

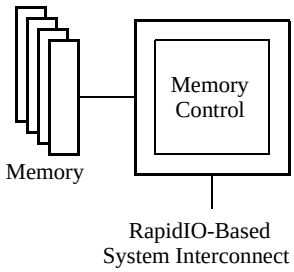


Figure 1-4. Memory-Only Processing Element Example

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.

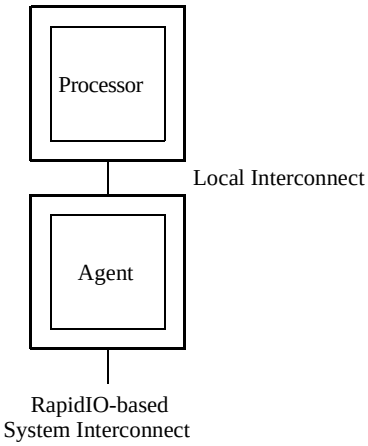


Figure 1-5. Processor-Only Processing Element Example

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.

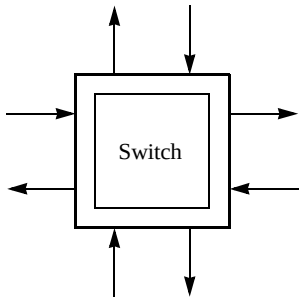


Figure 1-6. Switch Processing Element Example

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 Appendix 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 Appendix 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.

Part II

1

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 2.2.1	Type 10 Section 3.1.4
Data Message	MESSAGE, RESPONSE		Section 2.2.2	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.

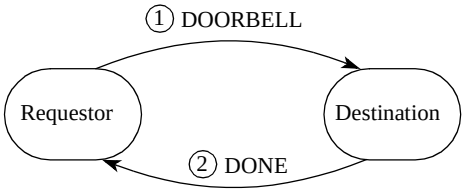


Figure 2-1. Doorbell Operation

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.

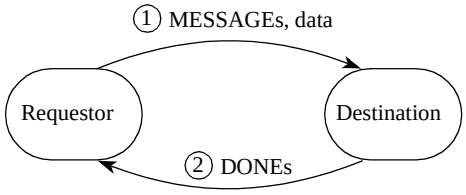


Figure 2-2. Message Operation

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)

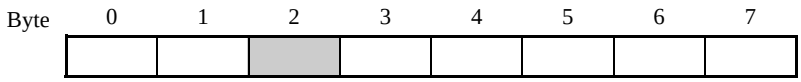
Using this information, the processing element's message passing hardware can determine that the 32 bytes contained in this part of the data message shall be put into local memory at address 0x3040.

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

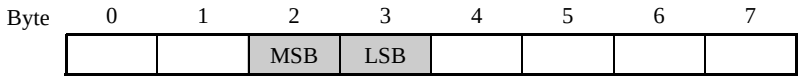
RapidIO has double-word (8-byte) aligned big-endian data payloads. This means that the RapidIO interface to devices that are little-endian shall perform the proper endian transformation at the output to format a data payload.

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.



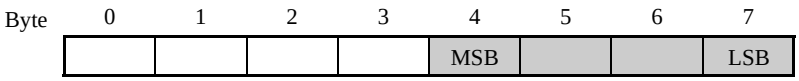
Byte address 0x0000_0002, the proper byte position is shaded.

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

Part II

2

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

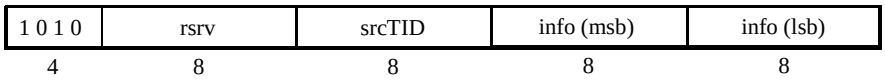


Figure 3-1. Type 10 Packet Bit Stream Format

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”.
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.

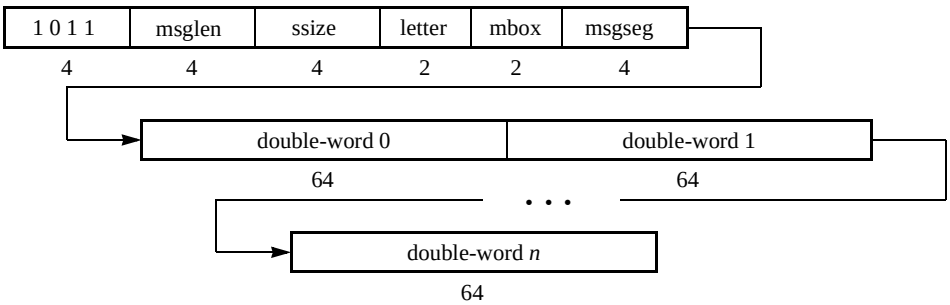


Figure 3-2. Type 11 Packet Bit Stream Format

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 Table 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.

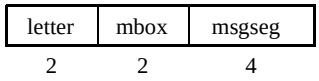


Figure 3-3. target_info Field for Message Responses

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

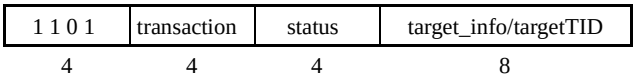


Figure 3-4. Type 13 Packet Bit Stream Format

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.

Part II

3

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 *Part 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	

Table 4-1. Message Passing 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	

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

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

Bit	Field Name	Description
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

Table 4-5. Bit Settings for Destination Operations CAR (Continued)

Bit	Field Name	Description
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 4-2 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.
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.

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

Bit	Field Name	Description
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.
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 Table 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

Part II

4

Appendix 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 appendix 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

Part II

A

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:

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.

For a more detailed description of the message packet format, refer to Section 3.1.5, “Type 11 Packet Format (Message Class).”

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 appendix, 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-0).

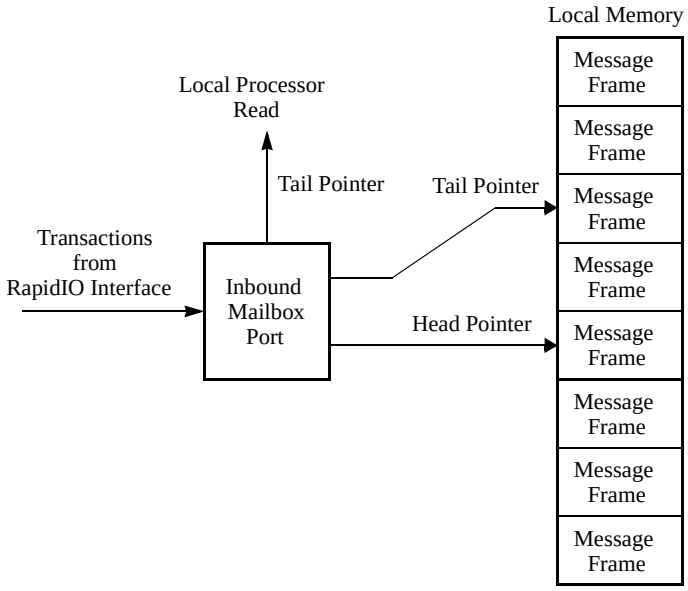


Figure A-0. Simple Inbound Mailbox Port Structure

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-0).

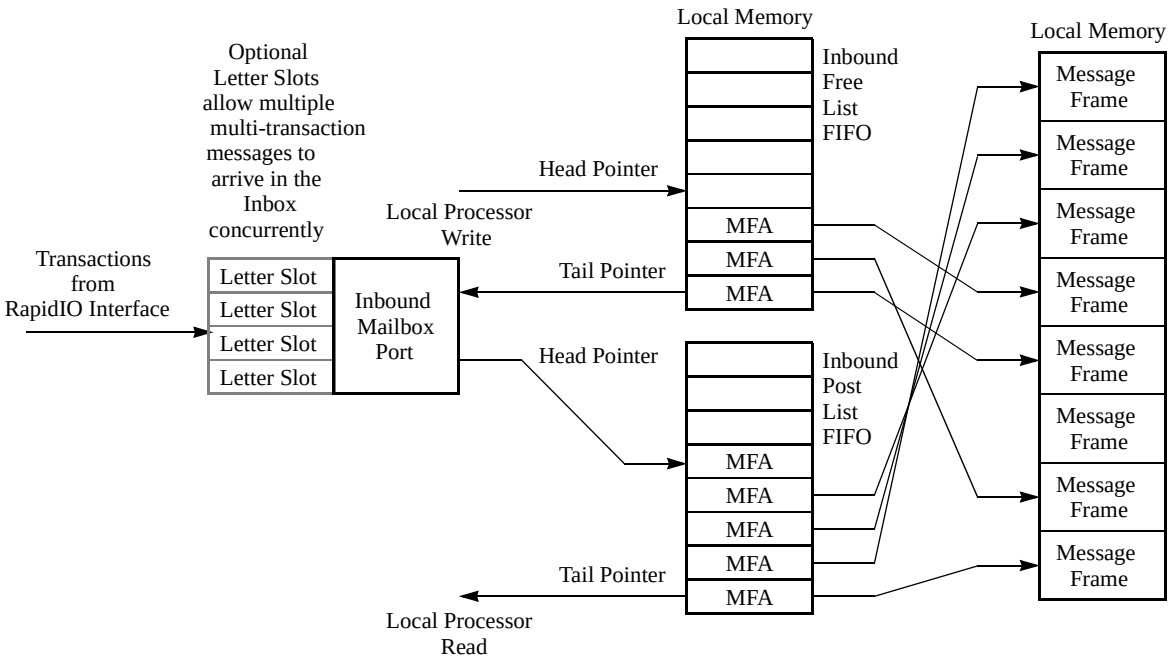


Figure A-0. Inbound Mailbox Structure

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-0) looks similar to the inbox.

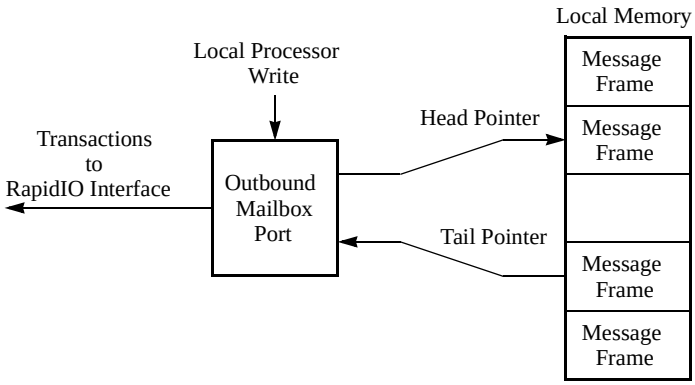


Figure A-0. Outbound Message Queue

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-0). 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.

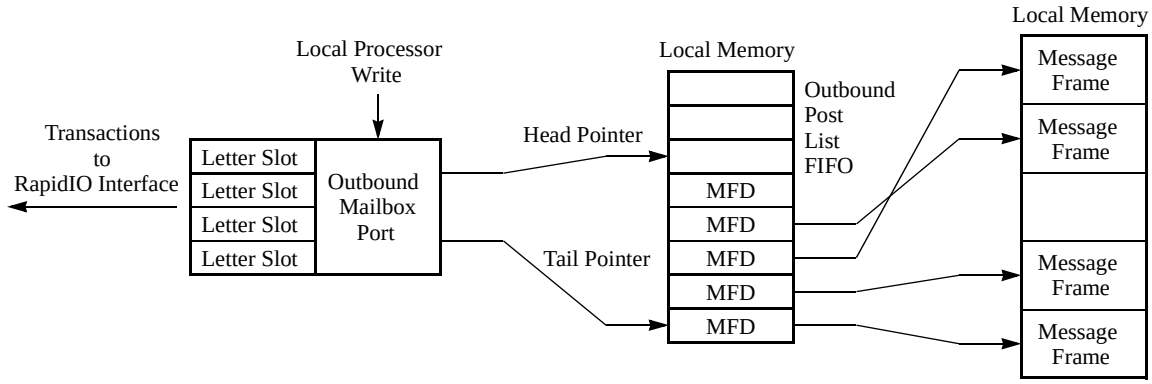


Figure A-0. Extended Outbound Message Queue

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-0.

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.

RapidIO™ Interconnect Specification Part III: Common Transport Specification

Rev. 1.2, 06/2002

RapidIO Trade Association

Revision History

Revision	Description	Date
1.1	First public release	03/08/2001
1.2	No technical changes	06/26/2002

Questions regarding the RapidIO Trade Association, specifications, or membership should be forwarded to:

RapidIO Trade Association
Suite 325, 3925 W. Braker Lane
Austin, TX 78759
512-305-0070 Tel.
512-305-0009 FAX.

Common Transport Specification Part III

Transport Format Descriptions 1

Common Transport Registers 2

Part III Common Transport Specification

1 Transport Format Descriptions

2 Common Transport Registers

Part III
Common Transport Specification

Part 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

Part 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.

Part III

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.

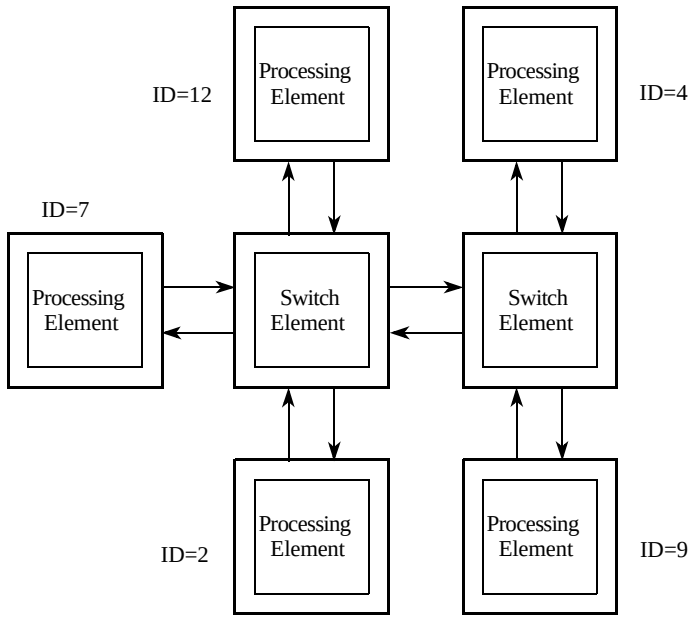


Figure 1-1. A Small Switch-Based System

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.

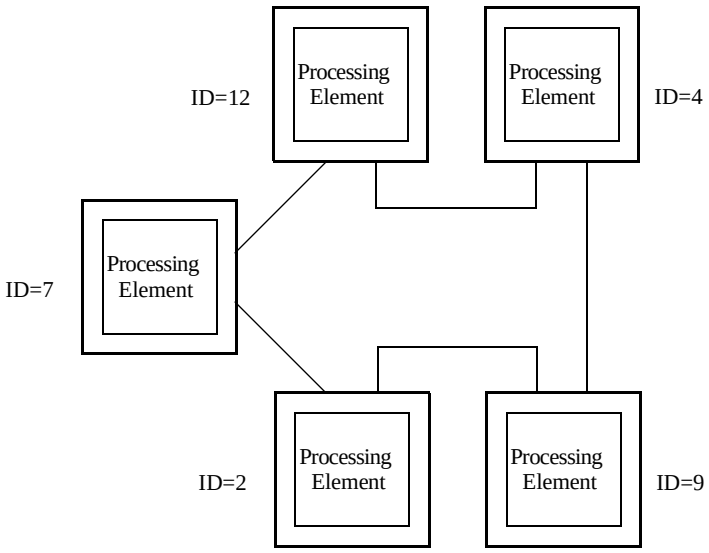


Figure 1-2. A Small Ring-Based System

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*.

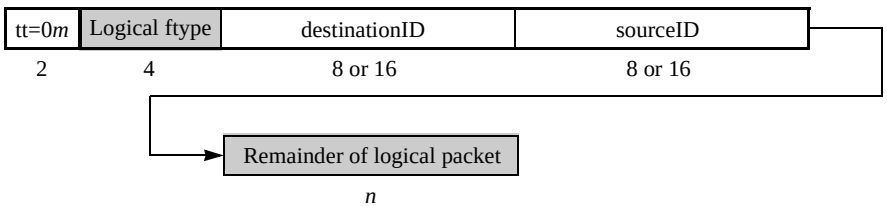


Figure 1-3. Destination-Source Transport Bit Stream

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 *Part I: Input/Output Logical Specification*.

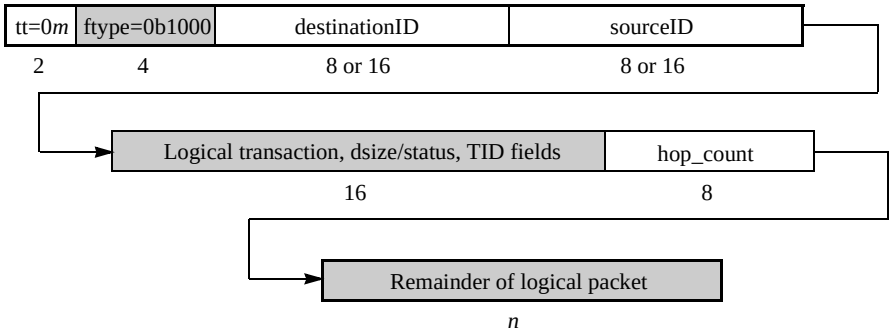


Figure 1-4. Maintenance Packet Transport Bit Stream

Part III

1

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 *Part 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

Table 2-2. 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.

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.

RapidIO™ Interconnect Specification
Part IV: Physical Layer 8/16 LP-LVDS Specification

Rev. 1.2, 6/2002

RapidIO Trade Association

Revision History

Revision	Description	Date
1.1	First public release	03/08/2001
1.2	Technical changes: incorporate Rev. 1.1 errata rev. 1.1.1, errata 3, showing 02-02-00009	06/26/2002

Questions regarding the RapidIO Trade Association, specifications, or membership should be forwarded to:

RapidIO Trade Association
Suite 325, 3925 W. Braker Lane
Austin, TX 78759
512-305-0070 Tel.
512-305-0009 FAX.

Part IV
Physical Layer 8/16 LP-LVDS Specification

Part 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 *Part 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 *Part II: Message Passing Logical Specification. Part I: Input/Output Logical Specification* defines the basic input/output system architecture of RapidIO.

IV.2 Contents

Following are the contents of *Part 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.
- Appendix 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.

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.

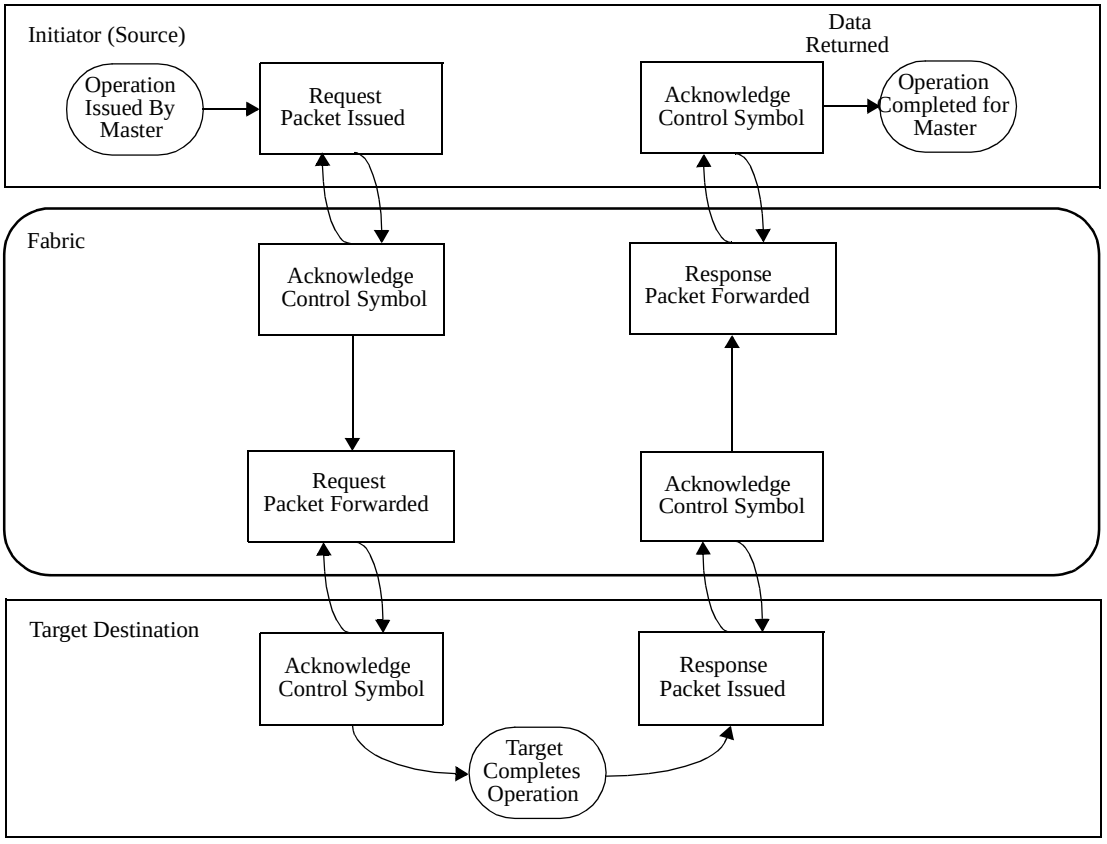


Figure 1-1. Example Transaction with Acknowledge

Part IV

1

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 1.1.2
prio	Sets packet priority: 0b00 - lowest priority 0b01 - medium priority 0b10 - high priority 0b11 - highest priority See Section 1.2.2 for an explanation of prioritizing packets
buf_status	Specifies the number of available packet buffers in the receiving device. See Section 1.2.4 and Table 1-2.
stype	Control symbol type—see Chapter 3, "Control Symbol Formats" for definition.
rsrv	Reserved

Table 1-2. 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	

Part IV

1

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.

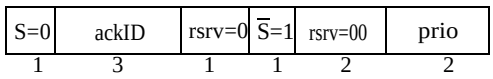


Figure 1-2. Packet Physical Layer Fields Format

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."

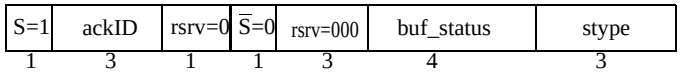


Figure 1-3. Control Symbol Physical Layer Fields Format

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

logical layer packet.

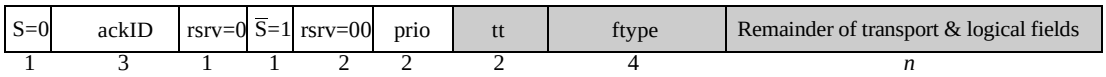


Figure 1-4. Flow Control Fields Bit Stream

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 1-1), 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.

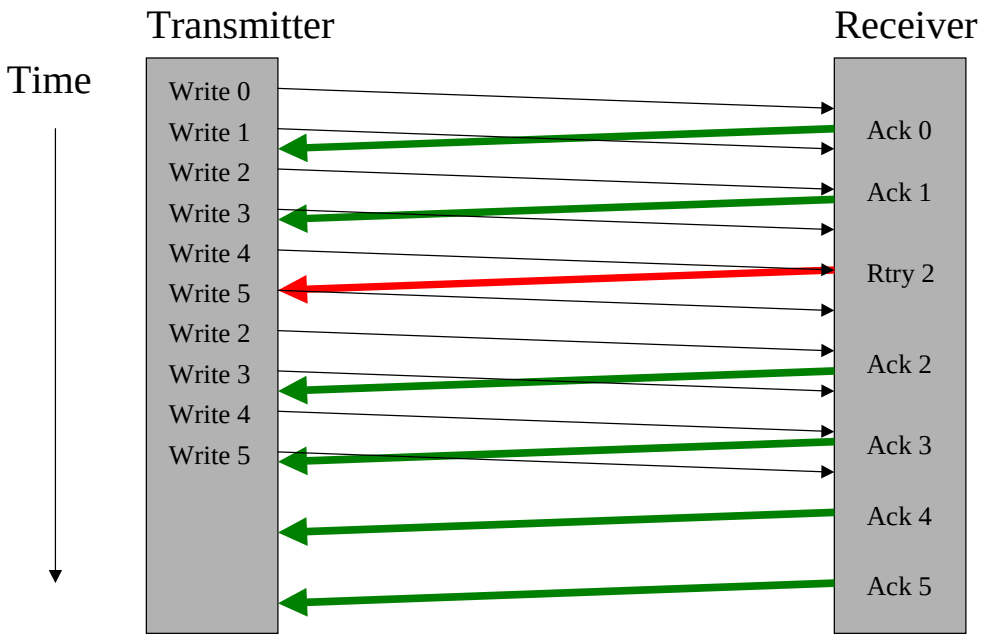
A port operating in receiver-controlled flow control mode accepts or rejects each inbound error-free packet based on whether the receiving port has enough buffer space available at the priority level of the packet. If there is enough buffer space available, the port accepts the packet and transmits a 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.

Part IV

1

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.

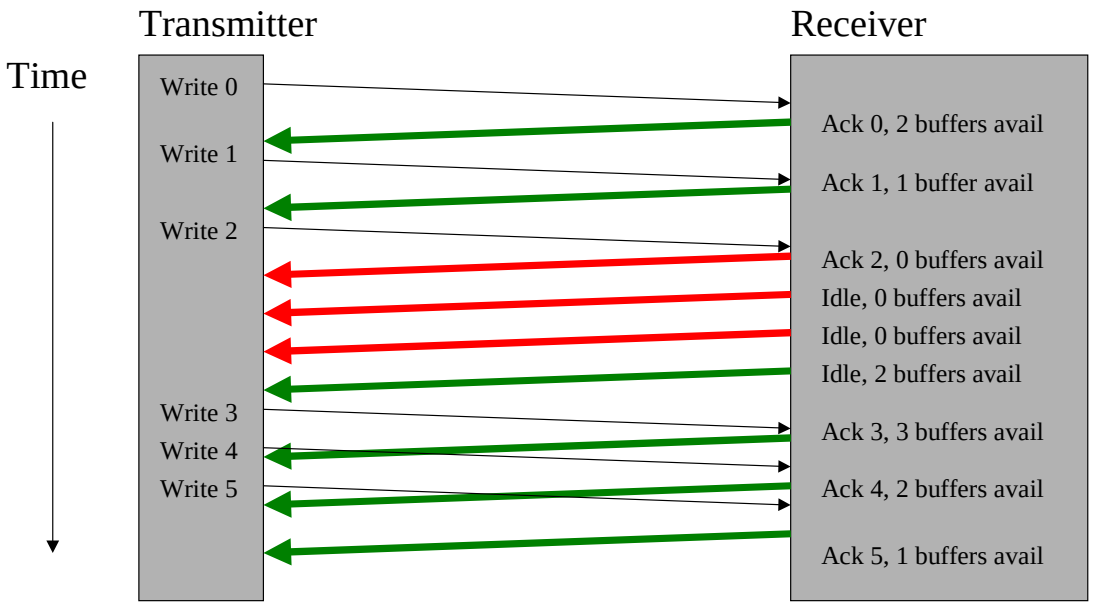
A port informs its link partner when the number of free buffers available for packet reception changes. The new value of buf_status is conveyed in the buf_status field in every control symbol containing the field that the port transmits. Each change in the number of

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 Section 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.

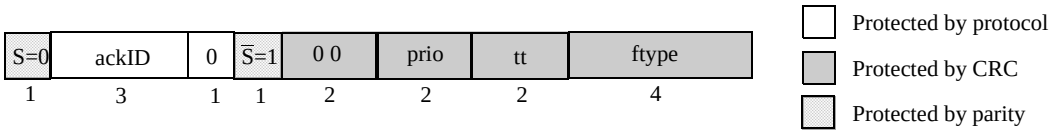


Figure 1-7. Error Coverage of First 16 Bits of Packet Header

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

Part IV

1

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.

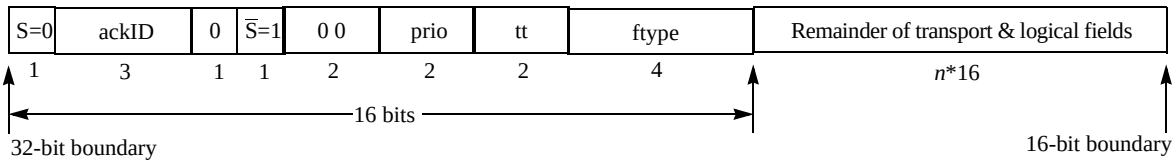


Figure 1-8. Naturally Aligned Packet Bit Stream

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 control 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.

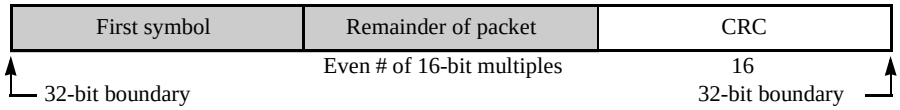


Figure 1-9. Naturally Aligned Packet Bit Stream Example 1

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

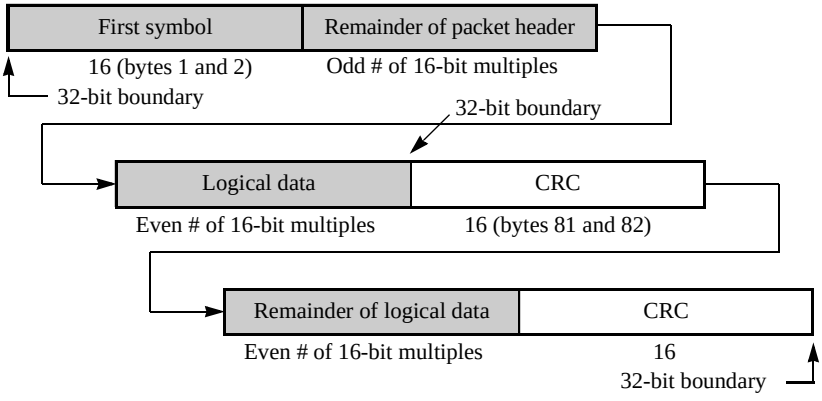


Figure 1-10. Naturally Aligned Packet Bit Stream Example 2

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

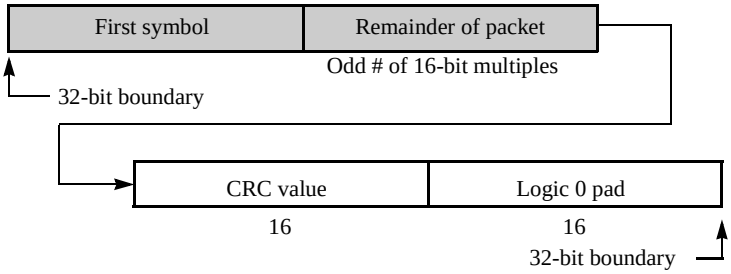


Figure 1-11. Padded Aligned Packet Bit Stream Example 1

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

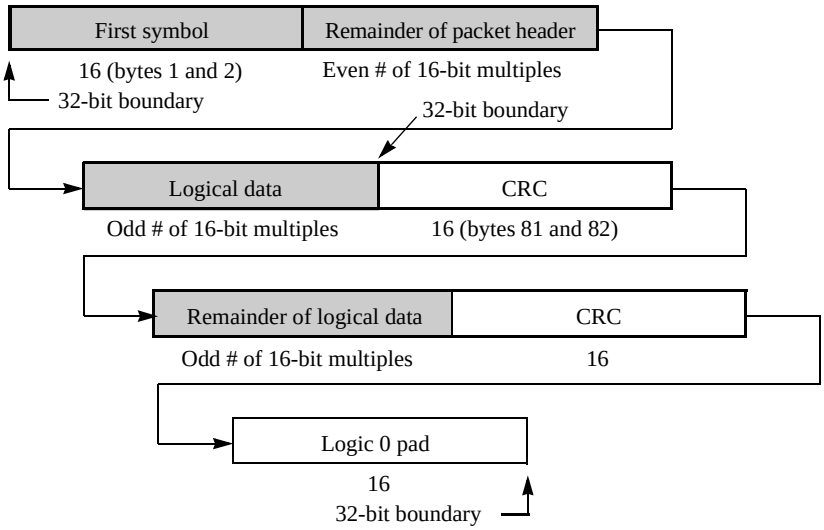


Figure 1-12. Padded Aligned Packet Bit Stream Example 2

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		

Table 1-4. Parallel CRC Intermediate Value Equations (Continued)

Check Bit	e00	e01	e02	e03	e04	e05	e06	e07	e08	e09	e10	e11	e12	e13	e14	e15
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

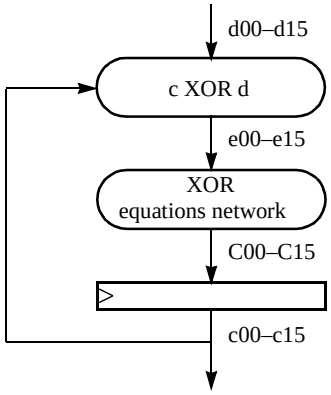


Figure 1-13. CRC Generation Pipeline

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 *Part 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.

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.

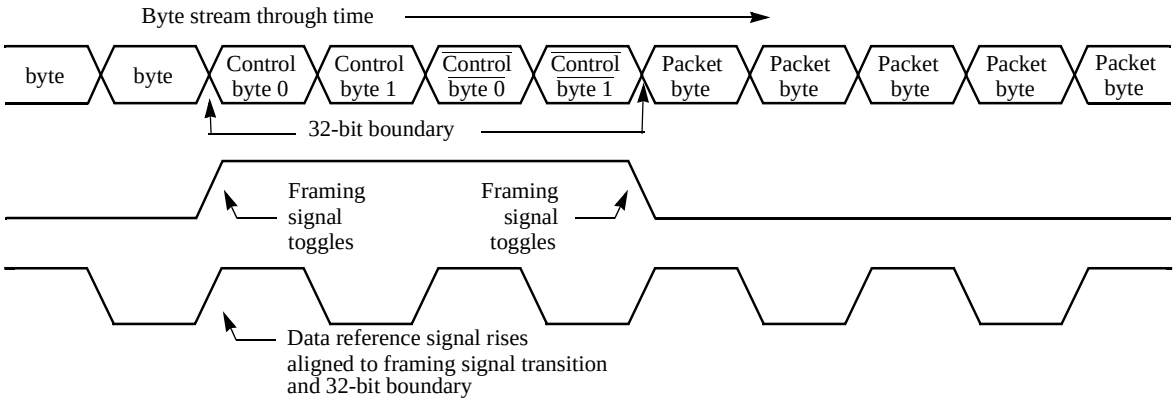


Figure 2-1. Framing Signal Maximum Toggle Rate for 8-bit Port

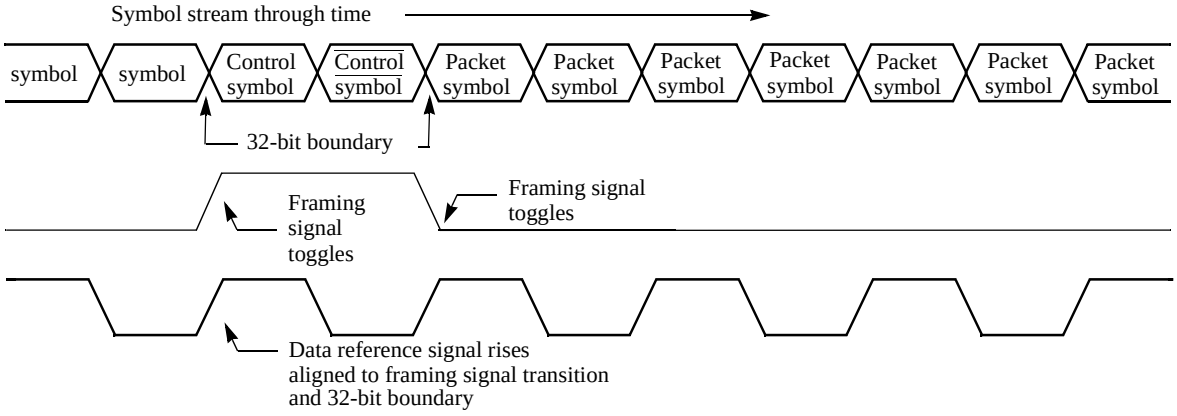


Figure 2-2. Framing Signal Maximum Toggle Rate for 16-bit Port

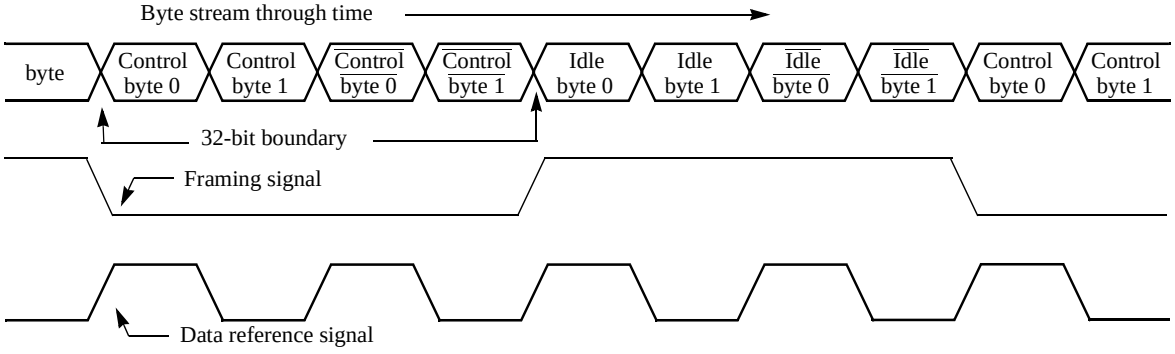


Figure 2-3. Control Symbol Delineation Example for 8-bit Port

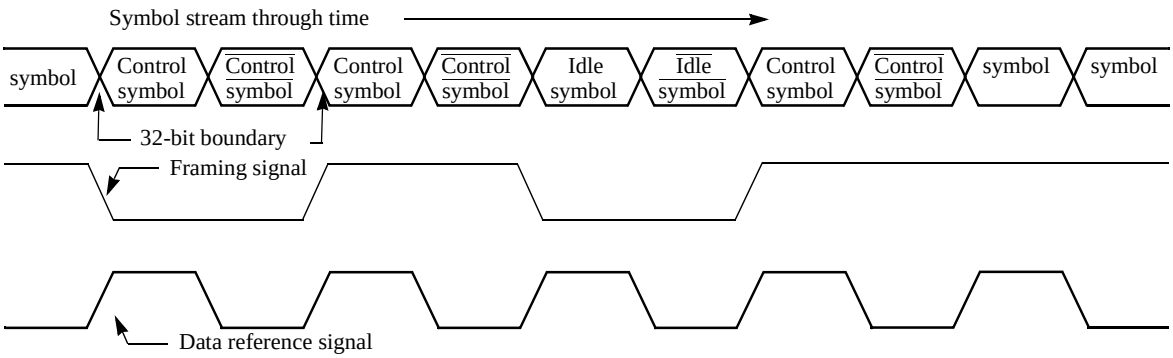


Figure 2-4. Control Symbol Delineation Example for 16-bit Port

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.

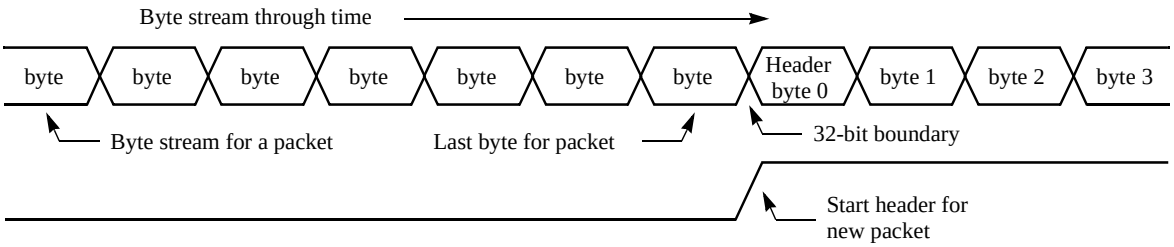


Figure 2-5. Header Marked End of Packet (8-bit Port)

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.

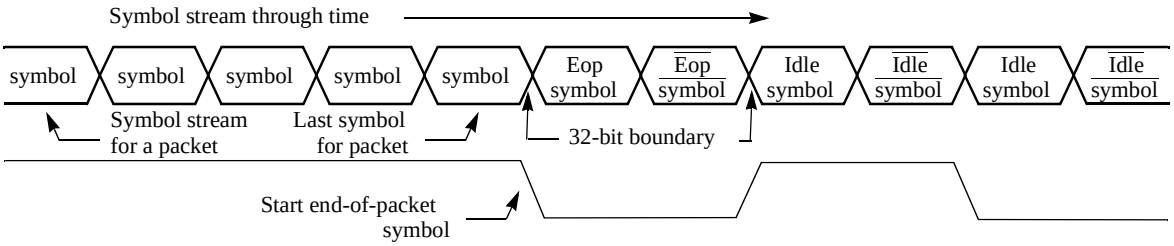


Figure 2-6. End-Of-Packet Control Symbol Marked End of Packet (16-bit Port)

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.

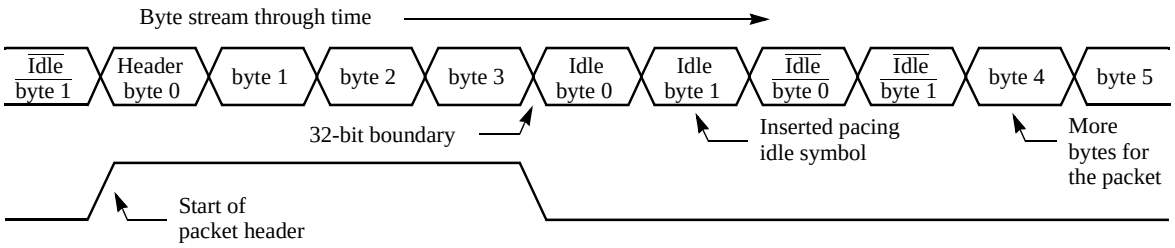


Figure 2-7. Pacing Idle Insertion in Packet (8-bit Port)

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.

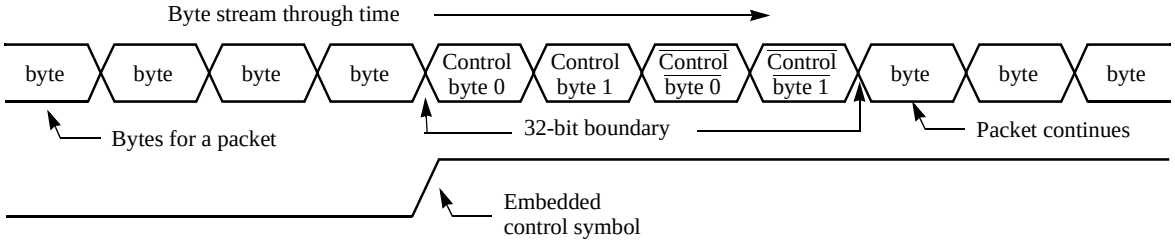


Figure 2-8. Embedded Control Symbols for 8-bit Port

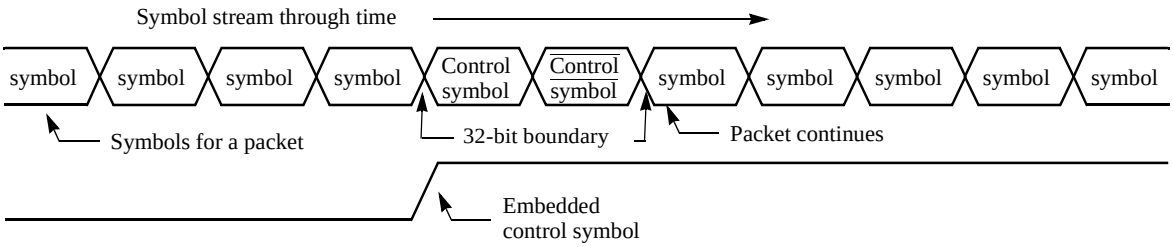


Figure 2-9. Embedded Control Symbols for 16-bit Port

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.

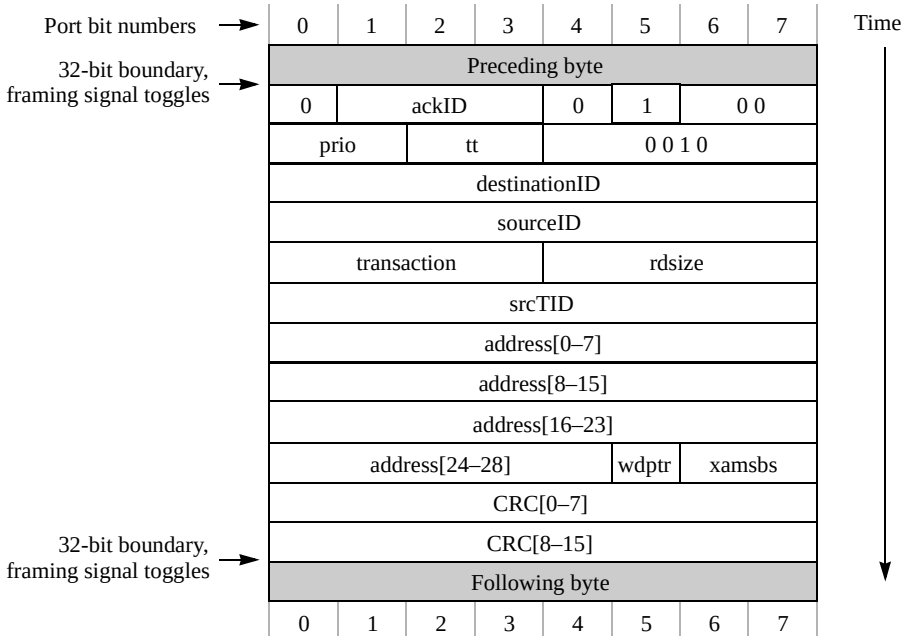


Figure 2-10. Request Packet Transmission Example 1

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

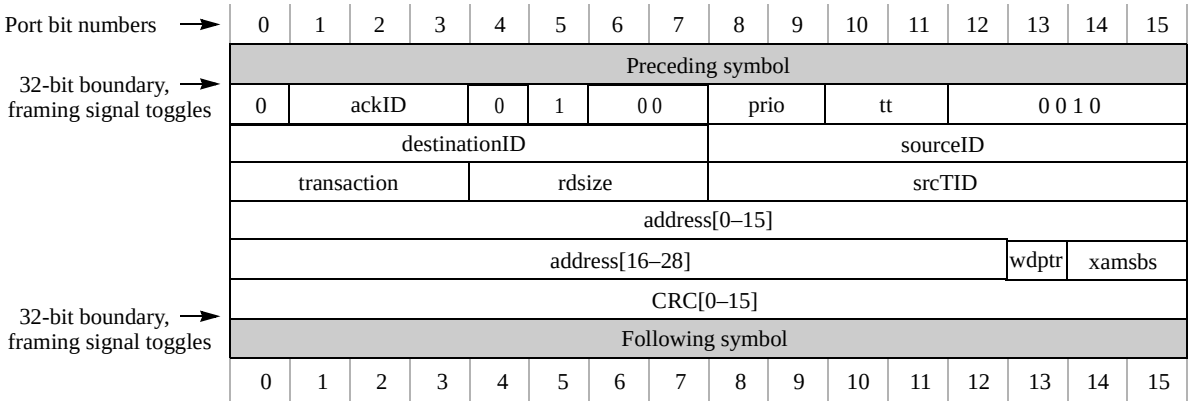


Figure 2-11. Request Packet Transmission Example 2

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.

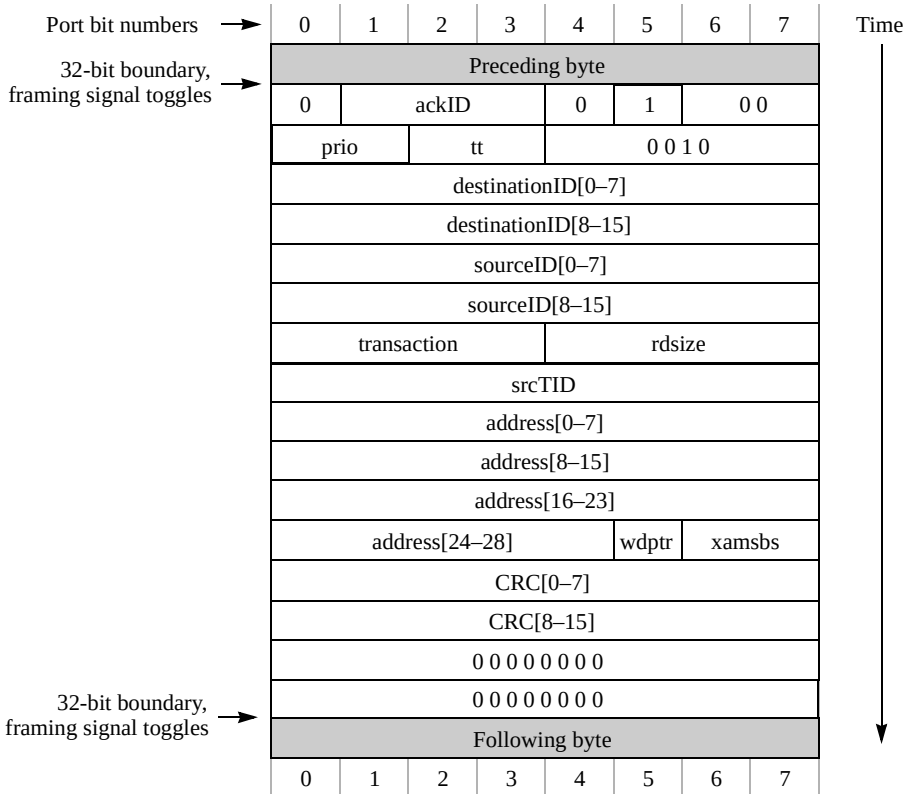


Figure 2-12. Request Packet Transmission Example 3

Figure 2-13 is the same packet as for Figure 2-12 but over the 16-bit port.

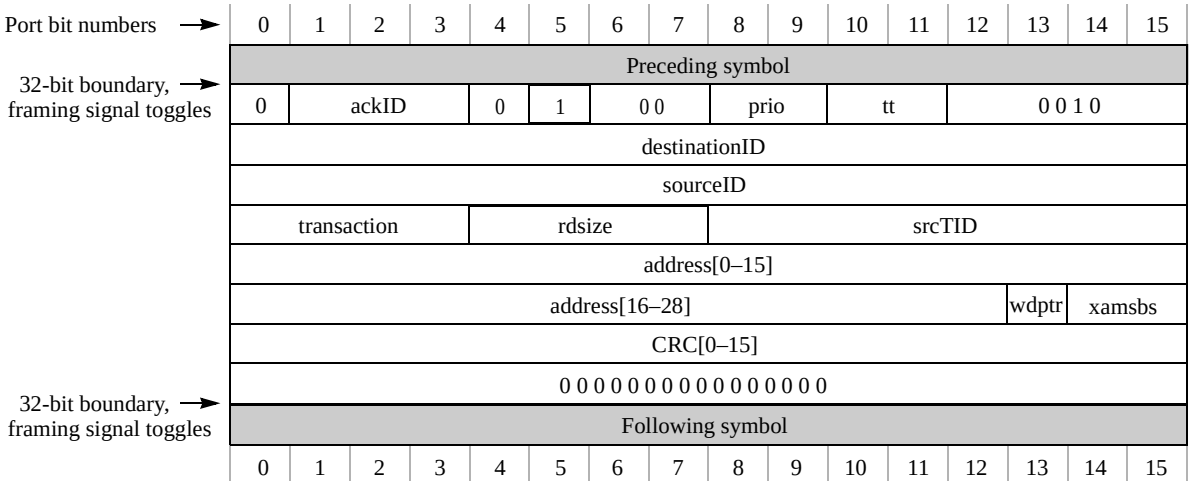


Figure 2-13. Request Packet Transmission Example 4

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.

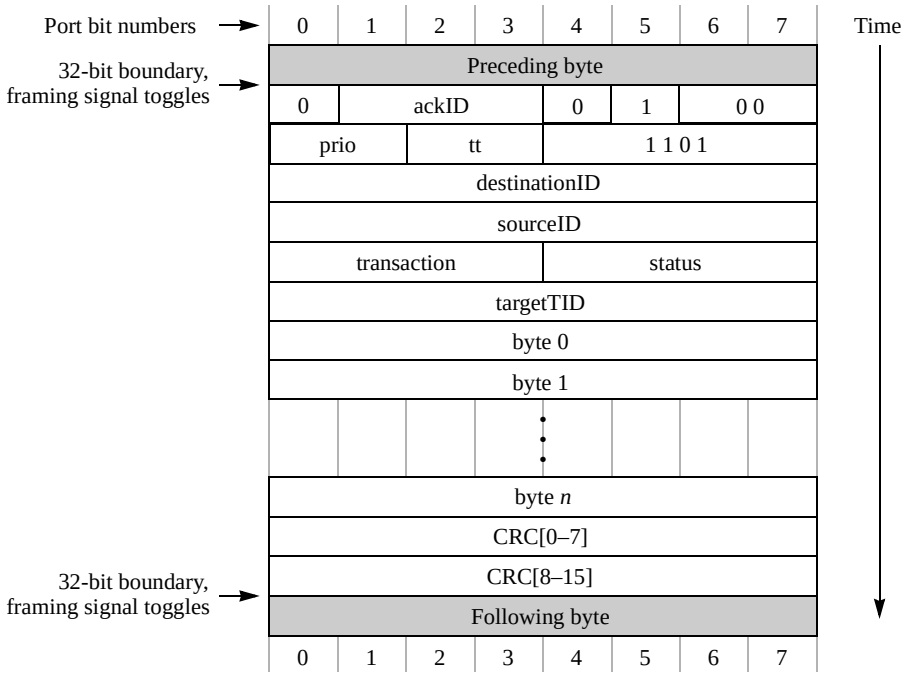


Figure 2-14. Response Packet Transmission Example 1

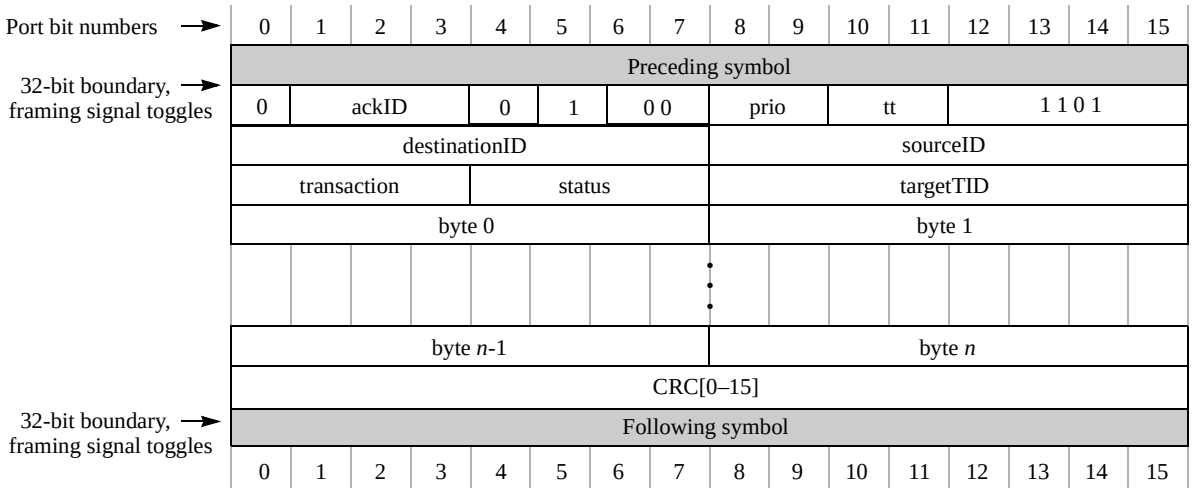


Figure 2-15. Response Packet Transmission Example 2

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.

Part IV

2

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.

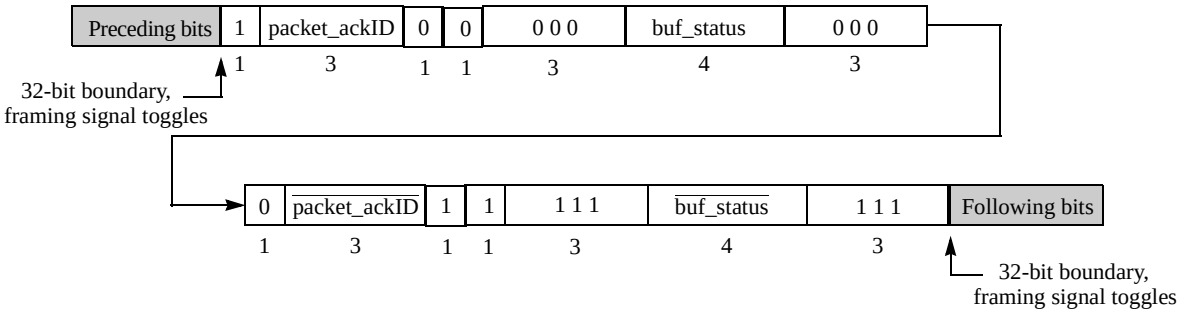


Figure 3-1. Type 0 Packet-Accepted Control Symbol Format

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.

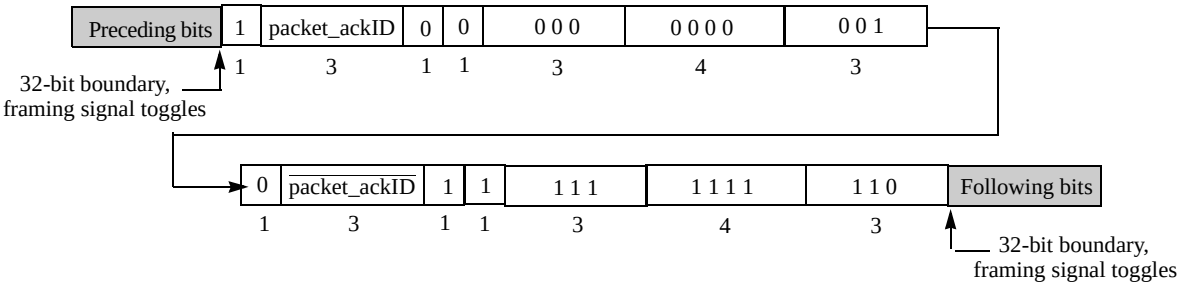


Figure 3-2. Type 1 Packet-Retry Control Symbol Format

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 Table 3-2. 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.

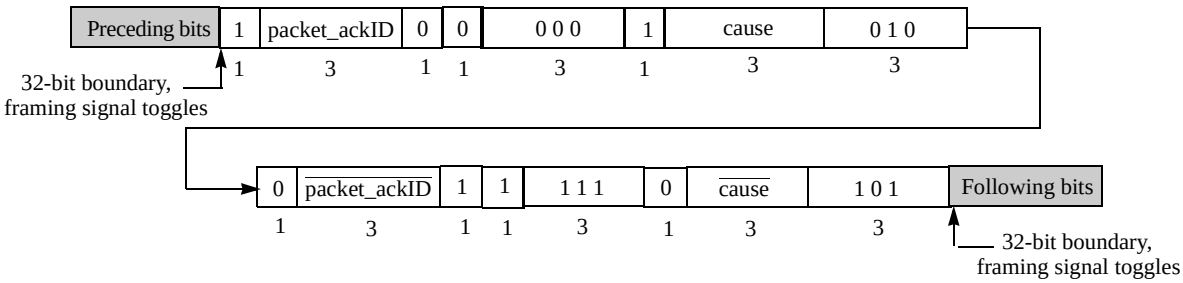


Figure 3-3. Type 2 Packet-Not-Accepted Control Symbol Format

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.

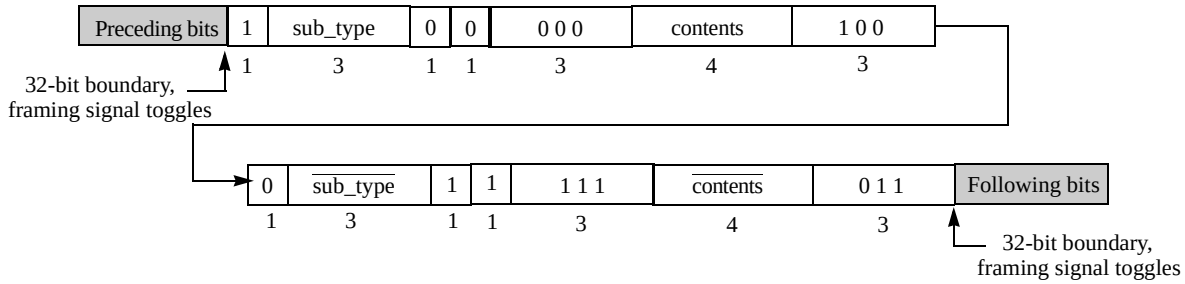


Figure 3-4. Type 4 Packet Control Symbol Format

Part IV

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.

3

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

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

sub_type Field Definition	sub_type Encoding	contents Field Definition
throttle	0b100	Specifies the number of aligned pacing idles that the sender inserts in a packet. The encodings are defined in Table 3-4.
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.

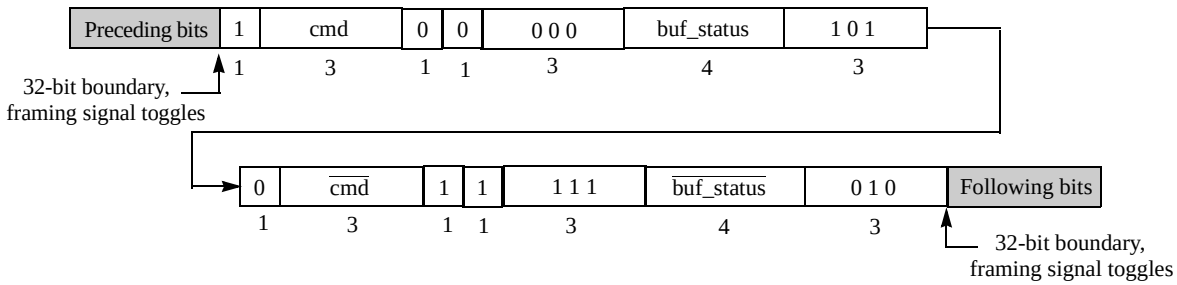


Figure 3-5. Type 5 Link-Request Control Symbol Format

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

Part IV

3

- 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.

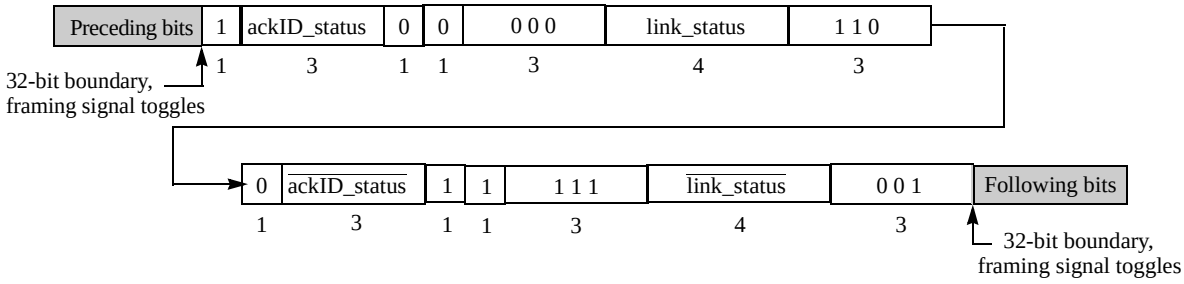


Figure 3-6. Type 6 Link-Response Control Symbol Format

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
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.”

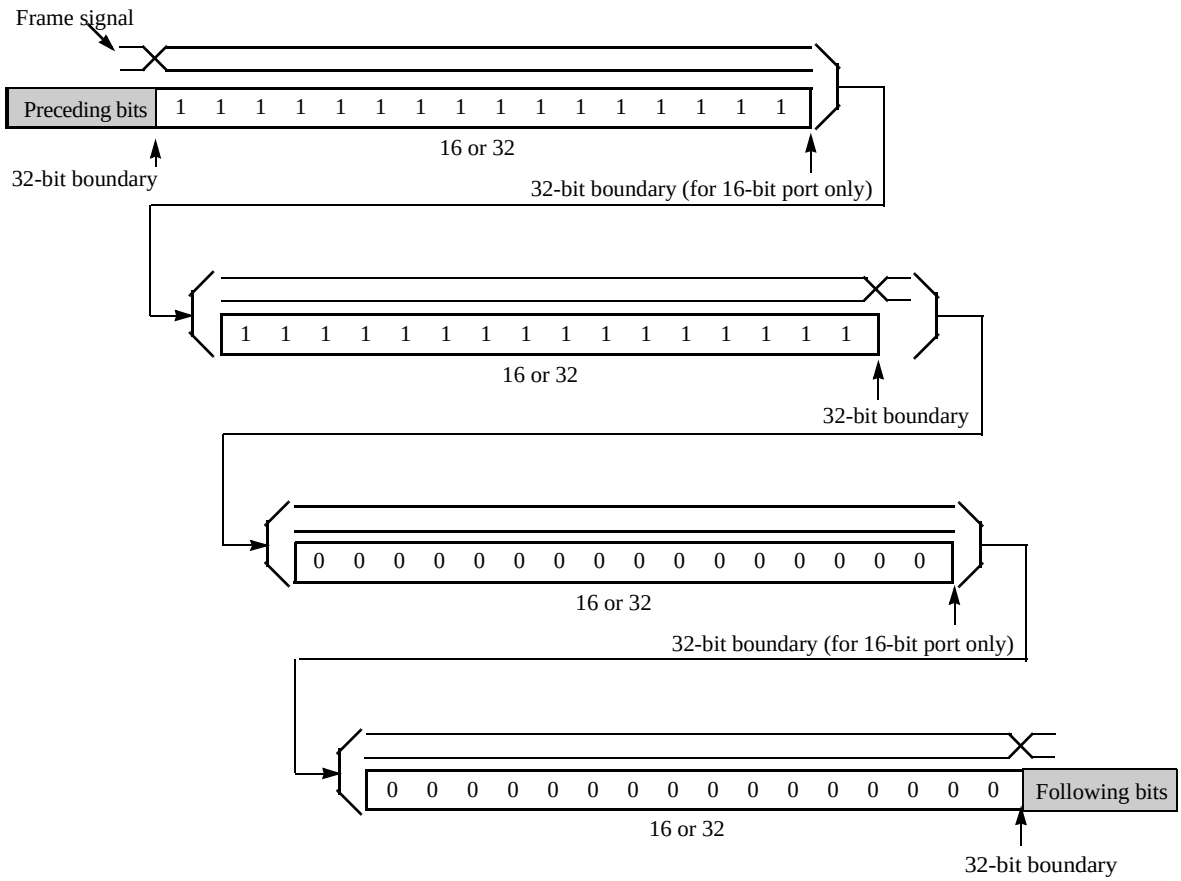


Figure 3-7. Type 7 TRAINING Pattern Format

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.

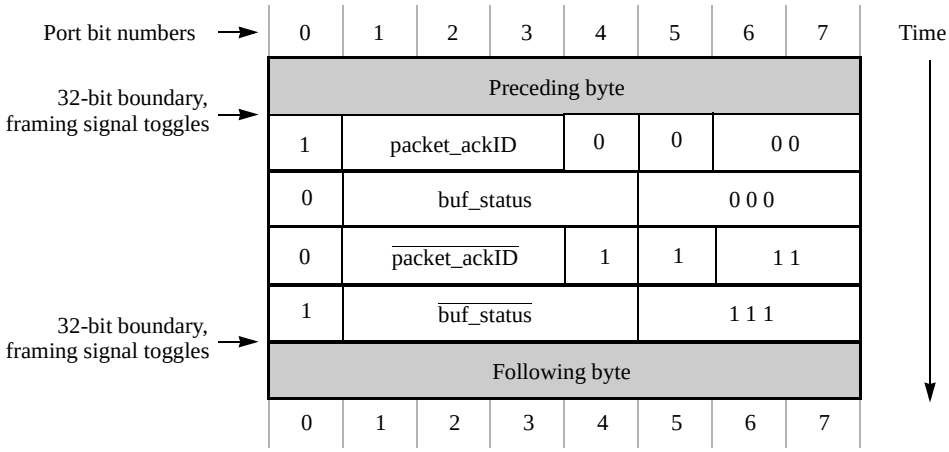


Figure 3-8. Control Symbol Transmission Example 1

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

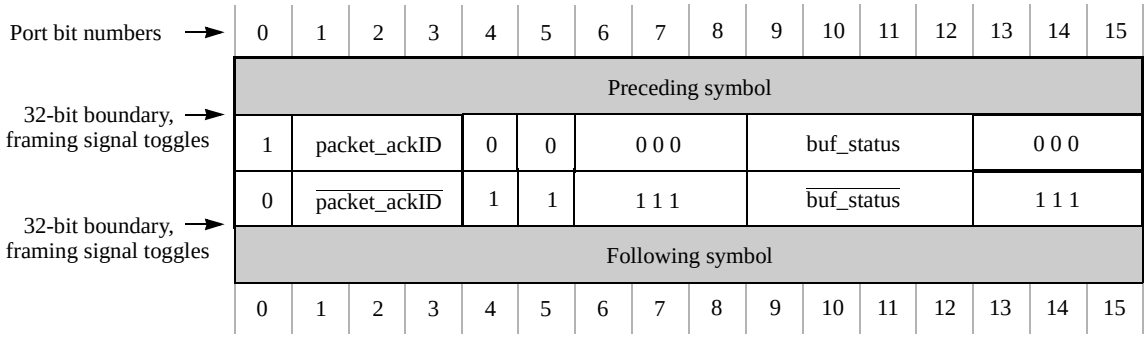


Figure 3-9. Control Symbol Transmission Example 2

Part IV

3

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 *Part 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

Part IV

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

4

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	

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

	Block Byte Offset	Register Name (Word 0)	Register Name (Word 1)
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
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

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)

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

Bit	Name	Reset Value	Description
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

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

	Block Byte Offset	Register Name (Word 0)	Register Name (Word 1)
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

Part IV

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

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.
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
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)

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

Bit	Name	Reset Value	Description
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

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

	Block Byte Offset	Register Name (Word 0)	Register Name (Word 1)
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

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).
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
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

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).

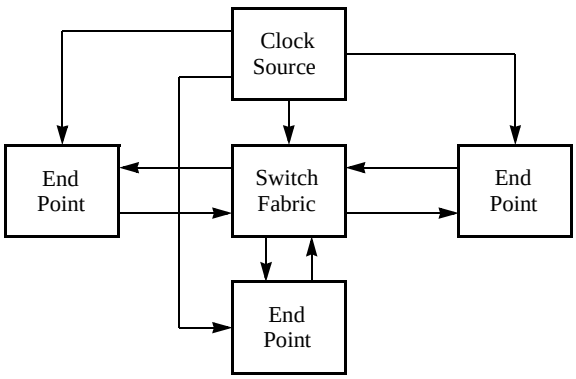


Figure 5-1. Clock Distribution in a Small System

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.

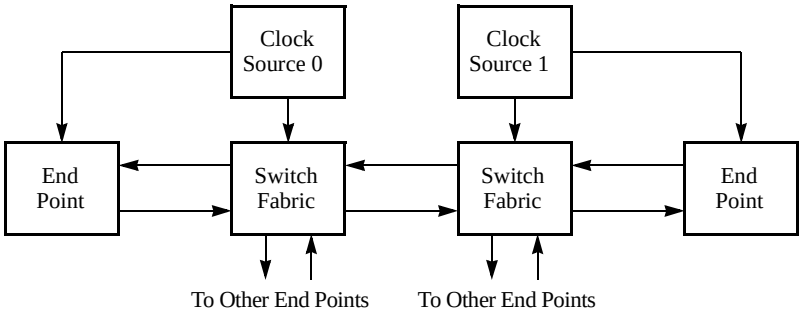


Figure 5-2. Clock Distribution in a Larger System

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.

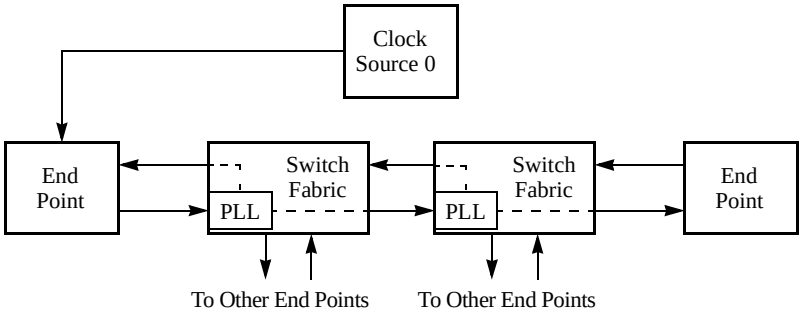


Figure 5-3. Clock Distribution Through the Interconnect

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.

Part IV

5

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.

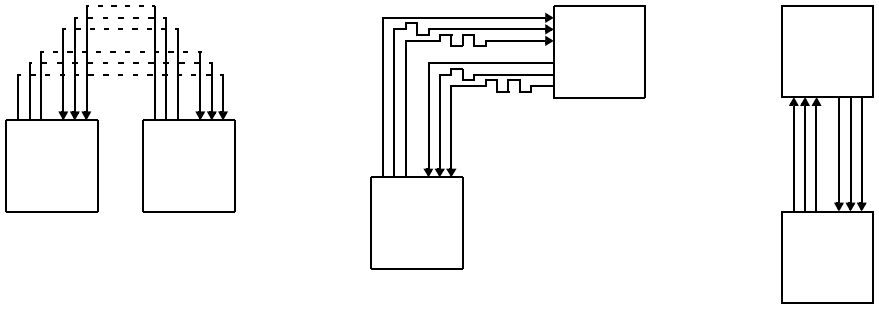


Figure 6-1. Routing for Equalized Skew for Several Placements

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.

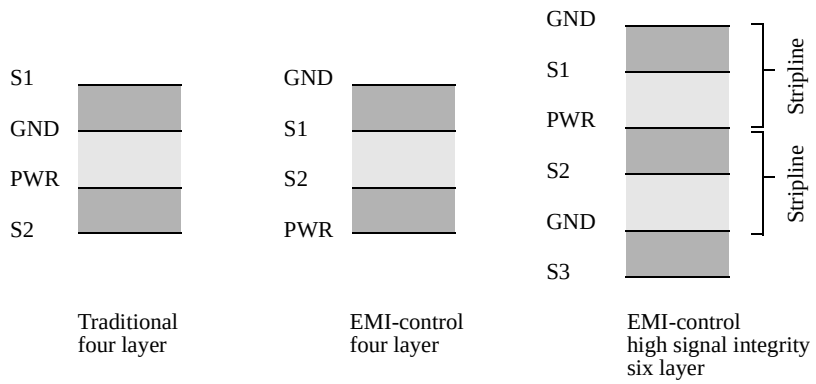


Figure 6-2. Potential PCB Stackups

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".

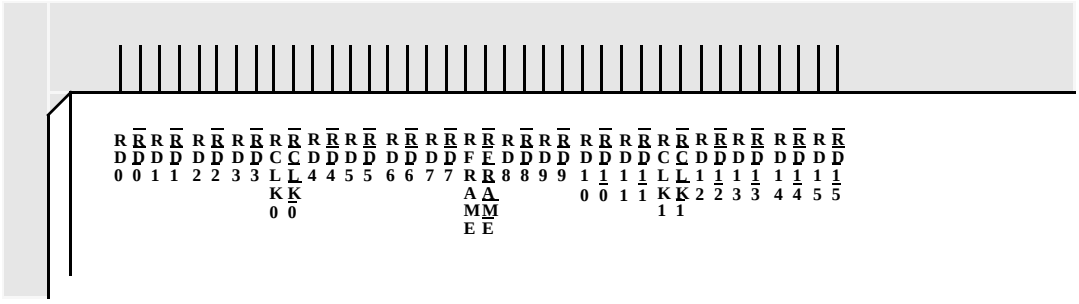


Figure 6-3. Recommended device pin escape, input port, top view of device

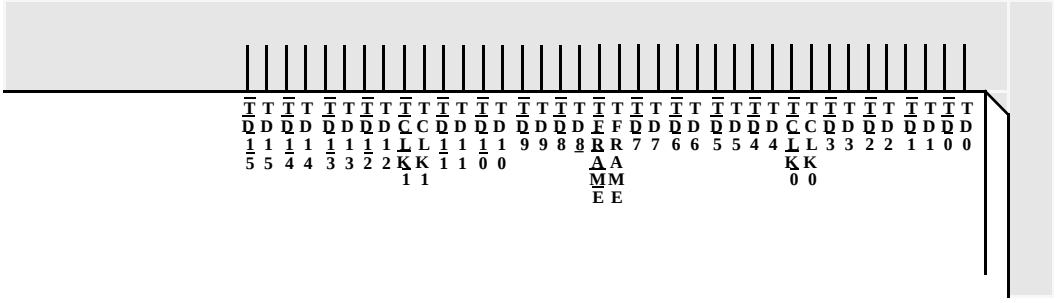


Figure 6-4. Recommended device pin escape, output port, top view of device

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.

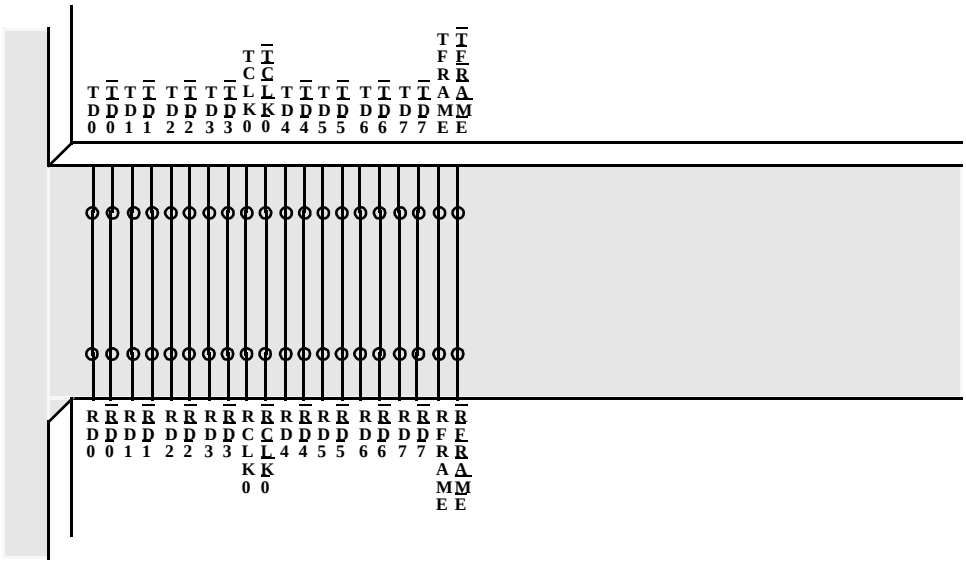


Figure 6-5. Opposed orientation, same side of board

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.

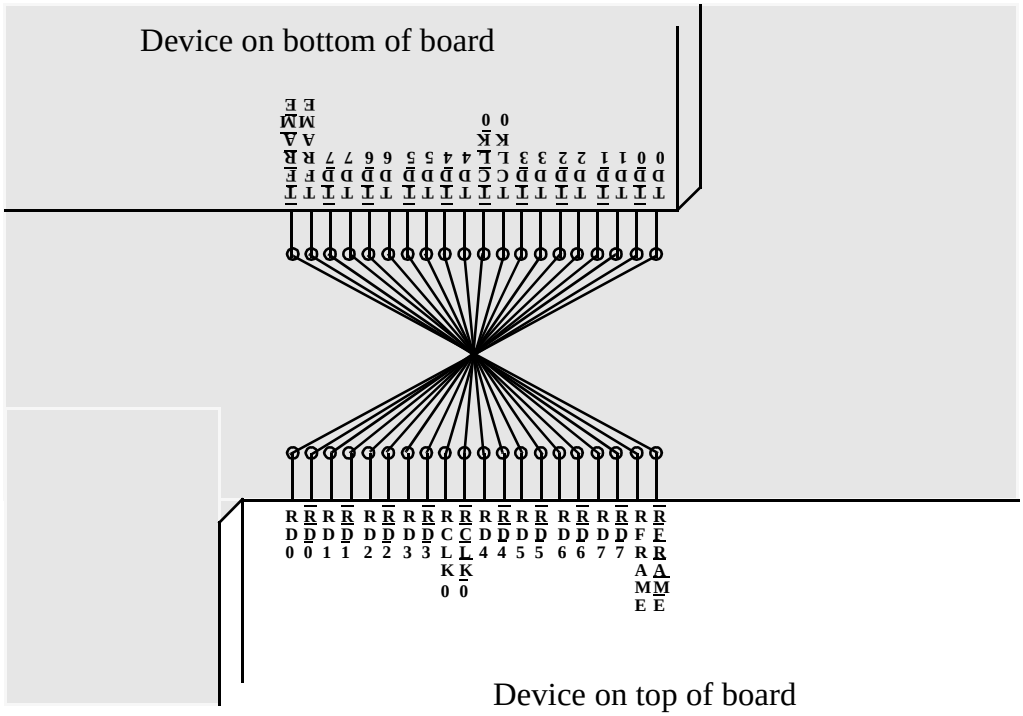


Figure 6-6. Opposed orientation, opposite sides of board

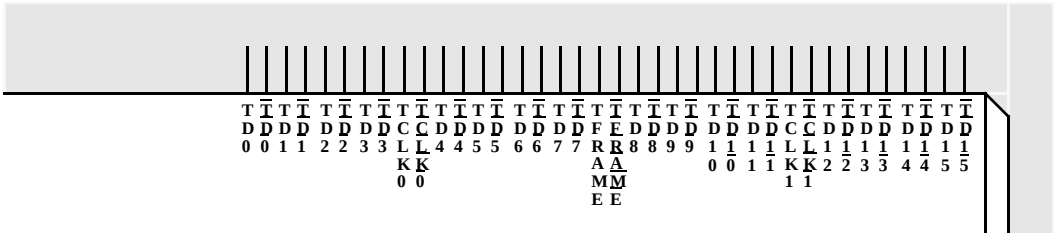


Figure 6-7. Recommended device pin escape, output port reversed, top view of device

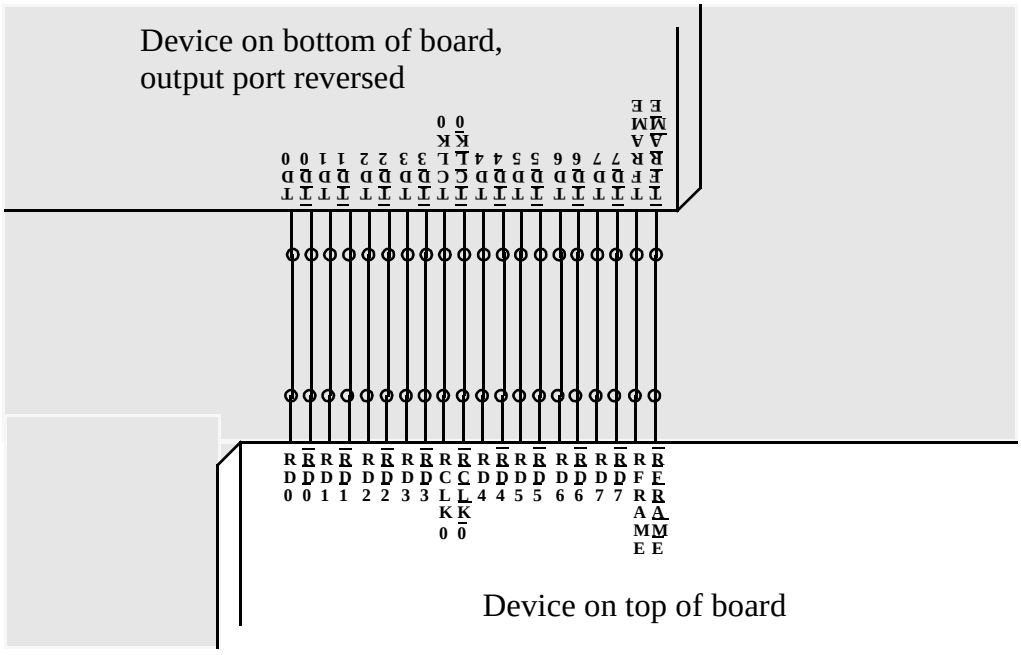


Figure 6-8. Opposed orientation, output port reversed, opposite sides of board

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.	
$\overline{\text{TCLK0}}$	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
$\overline{\text{TD[0-7]}}$	O	Transmit Data complement—This vector is the differential pair of TD[0-7].	Same as TD
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
$\overline{\text{TFRAME}}$	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.	

Table 7-1. Memory Interface Signal Description (Continued)

Signal Name	I/O	Signal Meaning	Timing Comments
$\overline{\text{TCLK1}}$	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
$\overline{\text{TD[8-15]}}$	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.	
$\overline{\text{RCLK0}}$	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.	
$\overline{\text{RD[0-7]}}$	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
$\overline{\text{RFRAME}}$	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.	
$\overline{\text{RCLK1}}$	I	Receive Clock complement—This signal is the differential pair of the RCLK1 signal.	
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.	
$\overline{\text{RD[8-15]}}$	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.

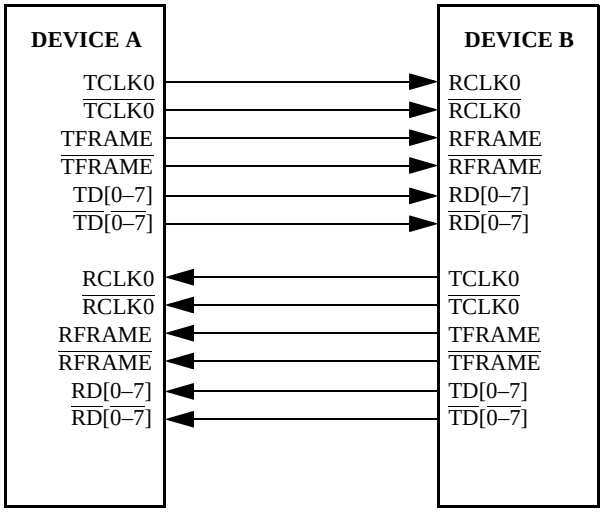


Figure 7-1. RapidIO 8-bit Device to 8-bit Device Interface Diagram

Figure 7-2 shows the connections between an 8-bit wide 8/16 LP-LVDS device and a 16-bit wide device.

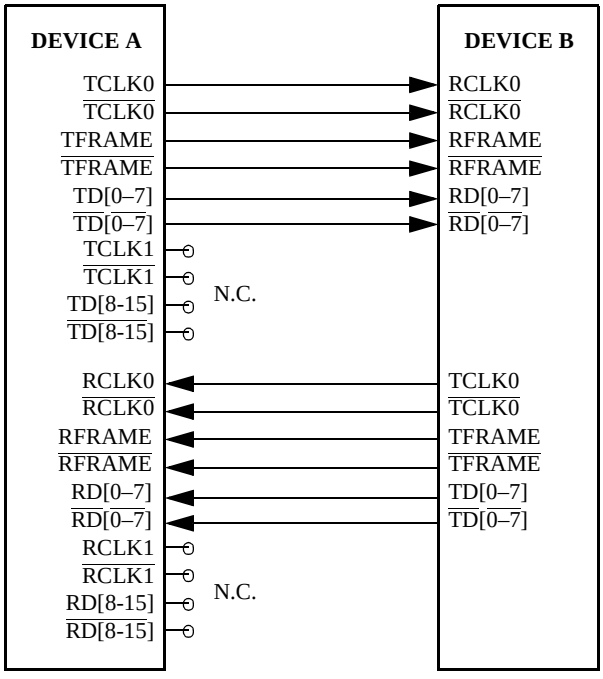


Figure 7-2. RapidIO 8-bit Device to 16-bit Device Interface Diagram

Figure 7-3 shows the connections between two 16-bit wide 8/16 LP-LVDS devices.

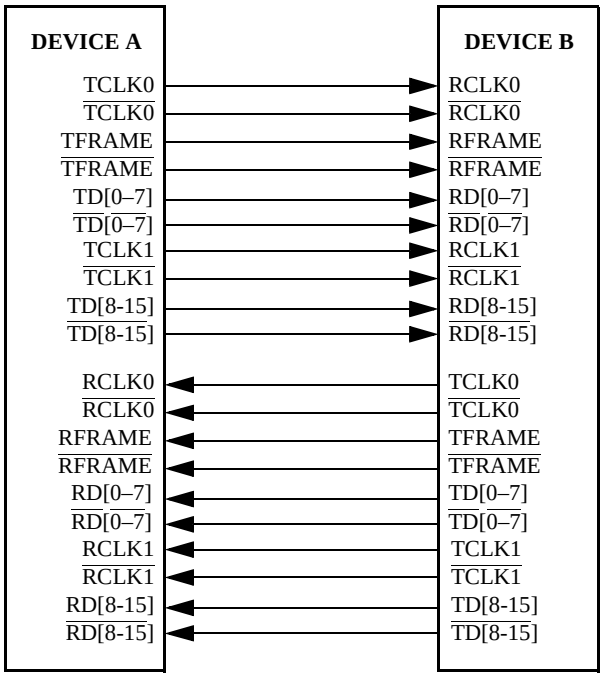


Figure 7-3. RapidIO 16-bit Device to 16-bit Device Interface Diagram

Part IV

7

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	Ω	
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	Ω	

DC driver signal levels are displayed in Figure 8-1.

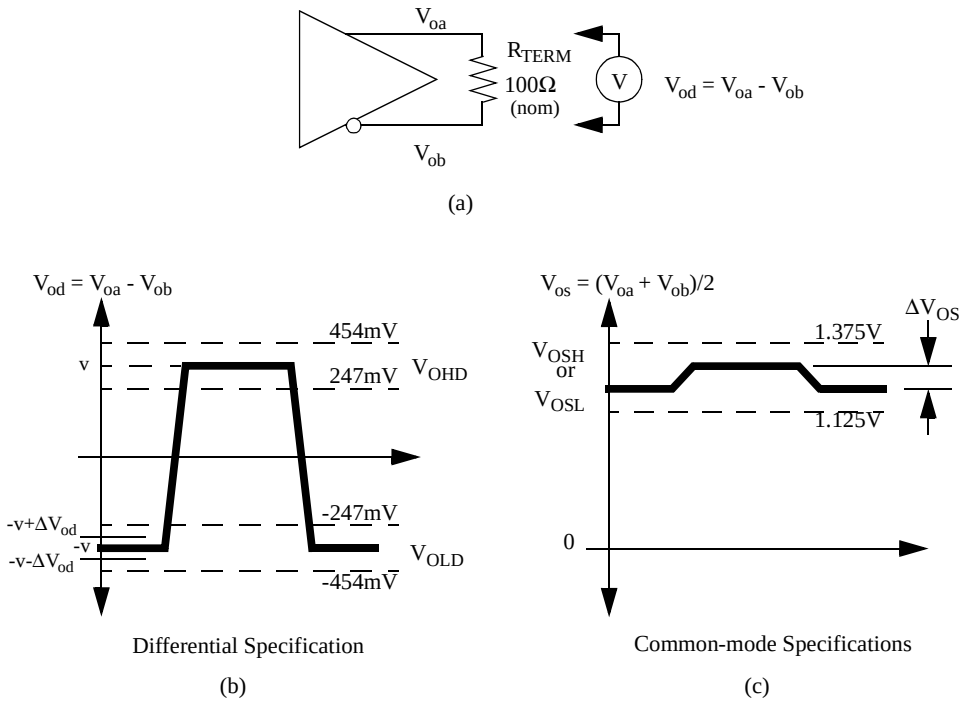


Figure 8-1. DC driver signal levels

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 8-2 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.

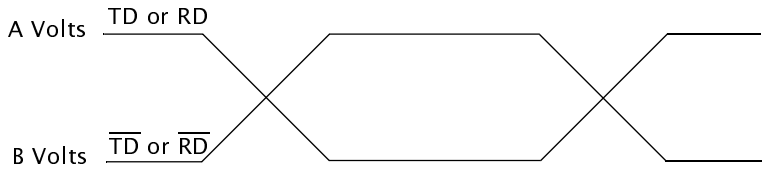


Figure 8-2. Differential Peak-Peak Voltage of Transmitter or Receiver

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.

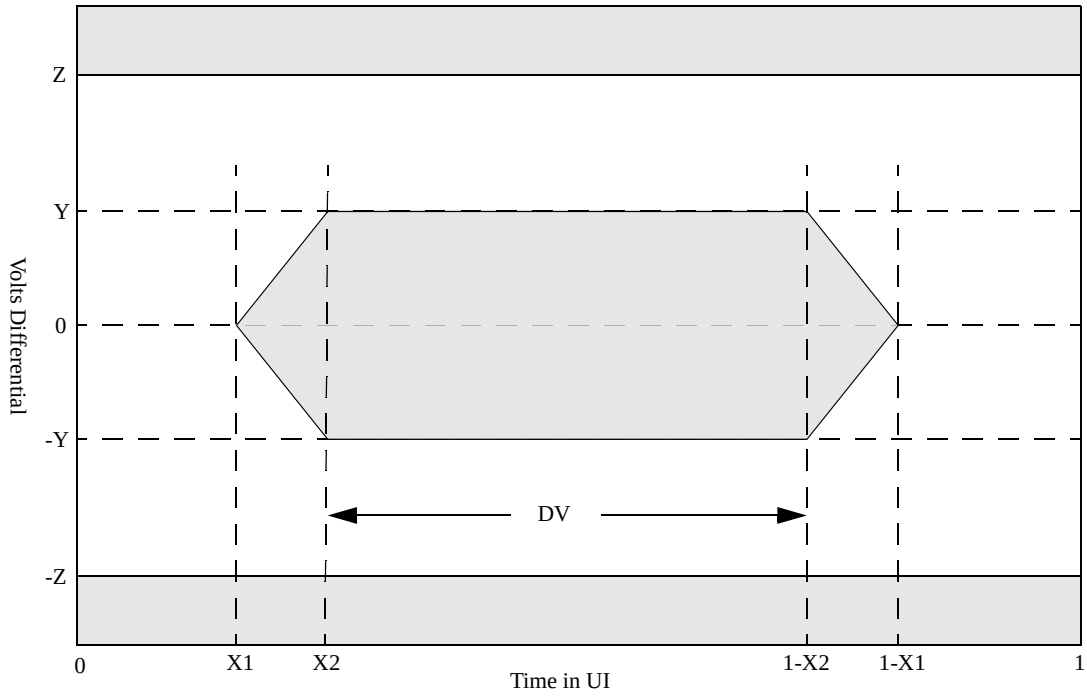


Figure 8-3. Example Compliance Mask

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 8-8, 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

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	
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

Part IV

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

8

Table 8-5. Driver AC Timing Specifications - 1000Mbps Data Rate/500MHz Clock Rate

Characteristic	Symbol	Range		Unit	Notes
		Min	Max		
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

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	

Table 8-7. Driver AC Timing Specifications - 2000Mbps Data Rate/1000MHz Clock Rate

Characteristic	Symbol	Range		Unit	Notes
		Min	Max		
V_{OD} rise time, 20-80% of the peak to peak differential signal swing	t_{RISE}	.1		UI	
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.

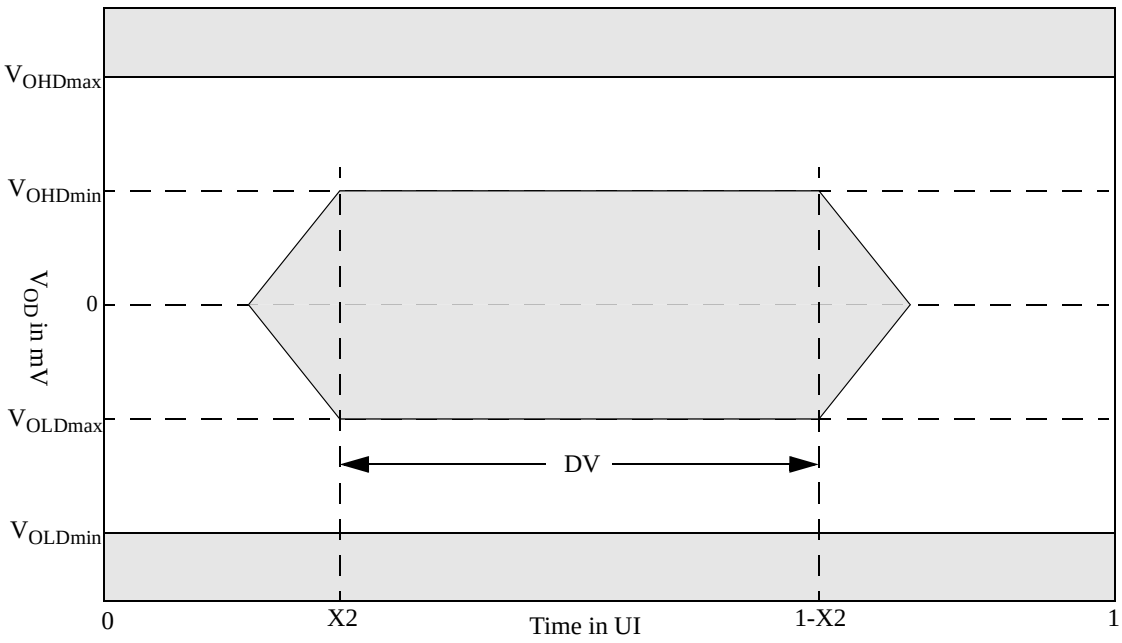


Figure 8-4. RapidIO Transmit Mask

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.

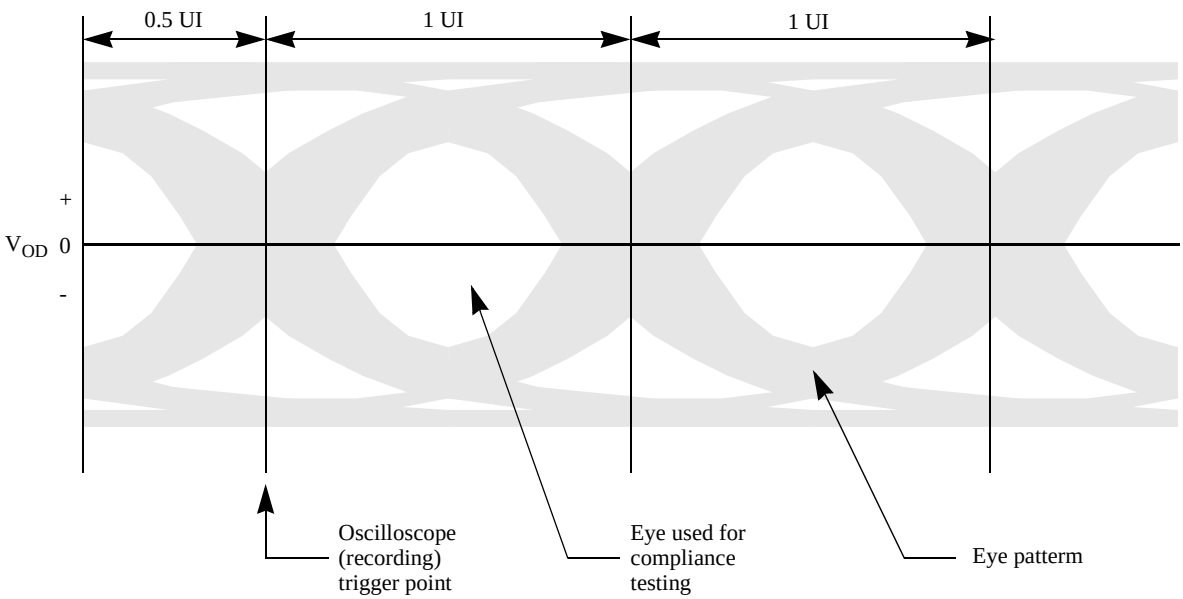


Figure 8-5. Example Driver Output Eye Pattern

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

Table 8-2. Receiver AC Timing Specifications - 750Mbps Data Rate/375MHz Clock Rate

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 8-3. Receiver AC Timing Specifications - 1000Mbps Data Rate/500MHz 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	.425		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 8-4. 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

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
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 8-6. 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.

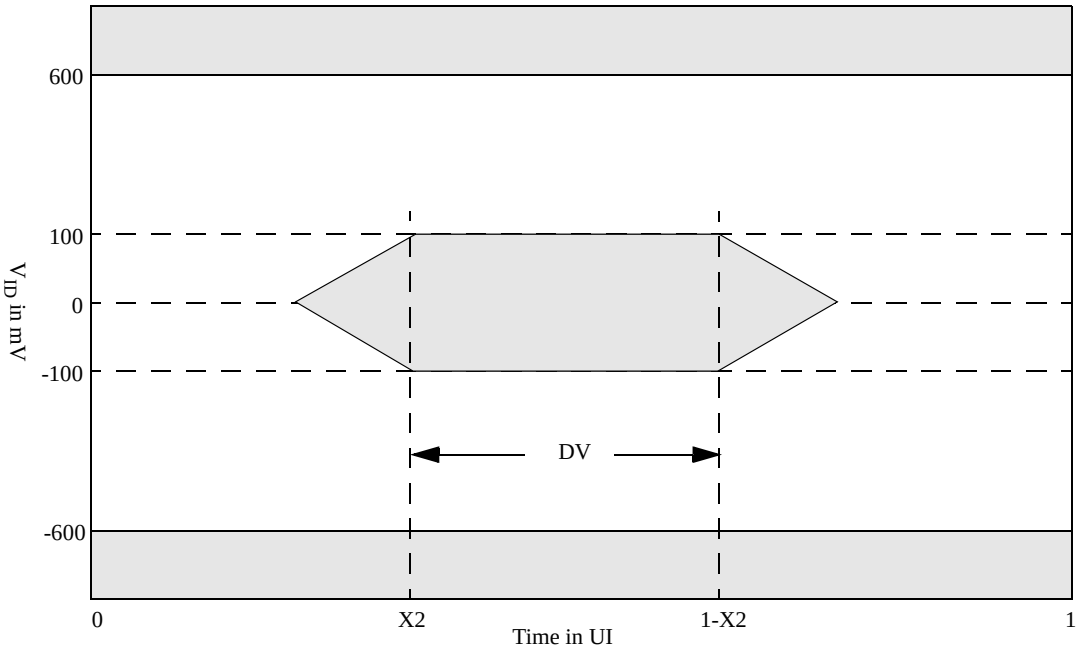


Figure 8-6. RapidIO Receive Mask

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.

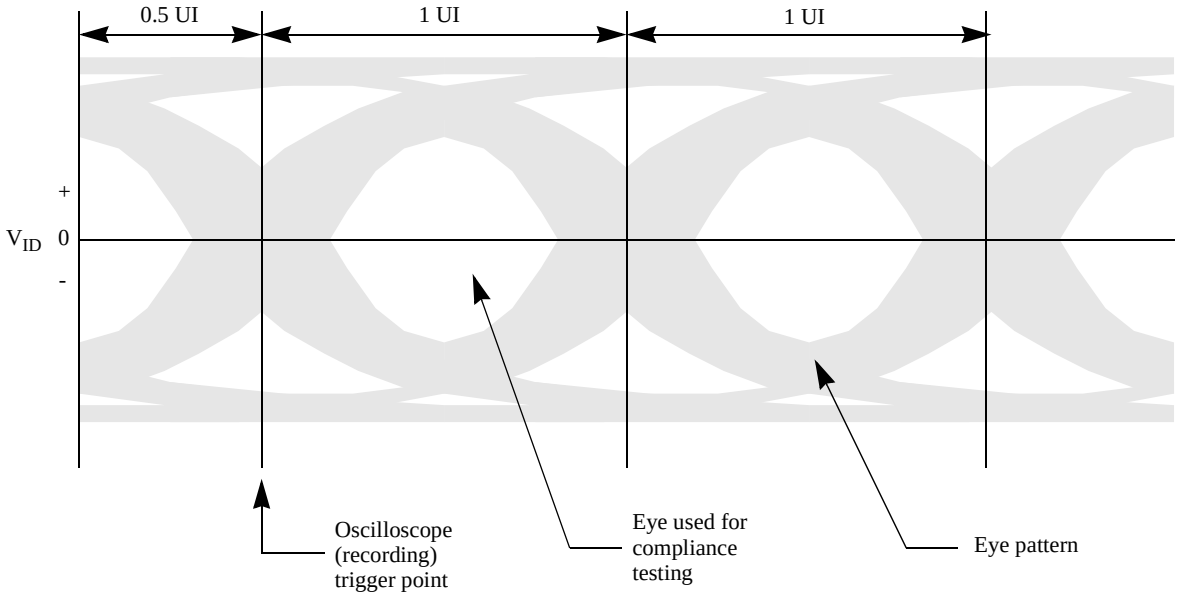


Figure 8-7. Example Receiver Input Eye Pattern

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.

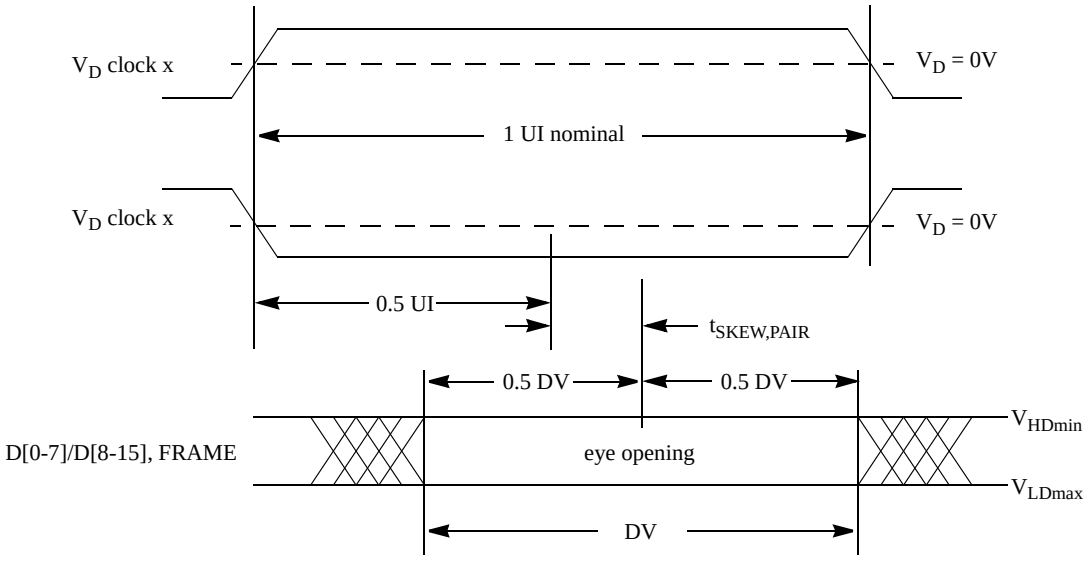


Figure 8-8. Data to Clock Skew

Figure 8-9 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.

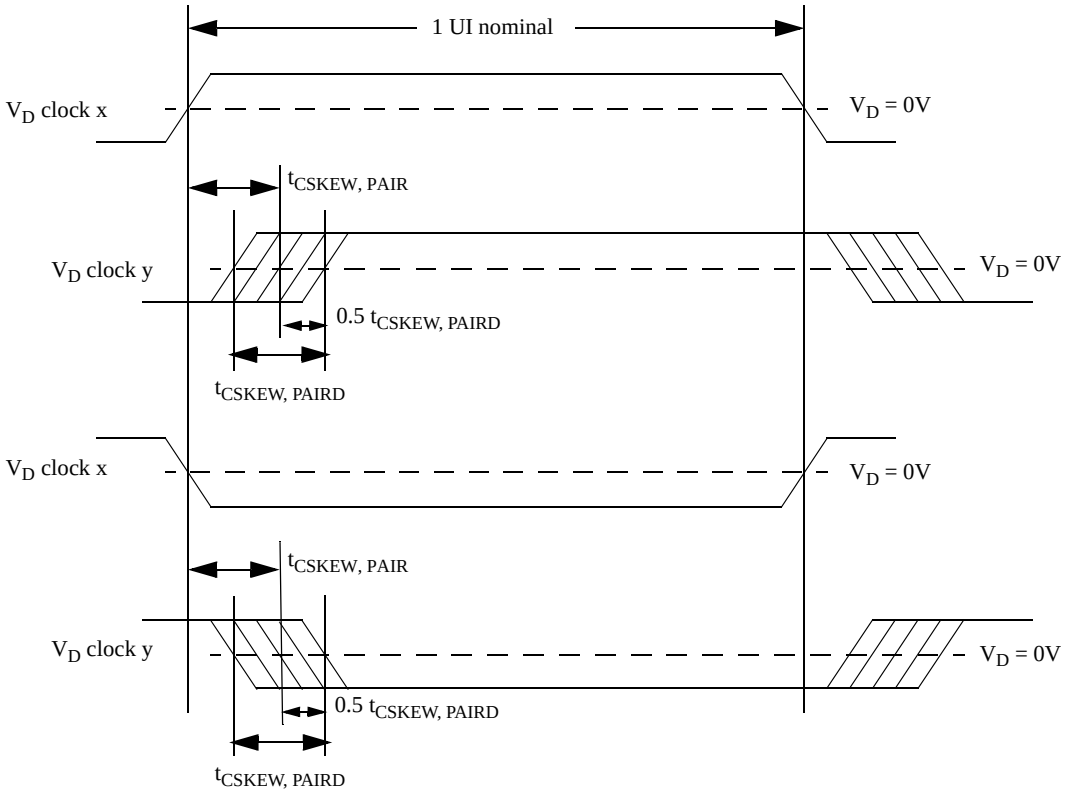


Figure 8-9. Clock to Clock Skew

Figure 8-10 shows the definition of the data to data static skew parameter t_{DPAIR} and how the skew parameters are applied.

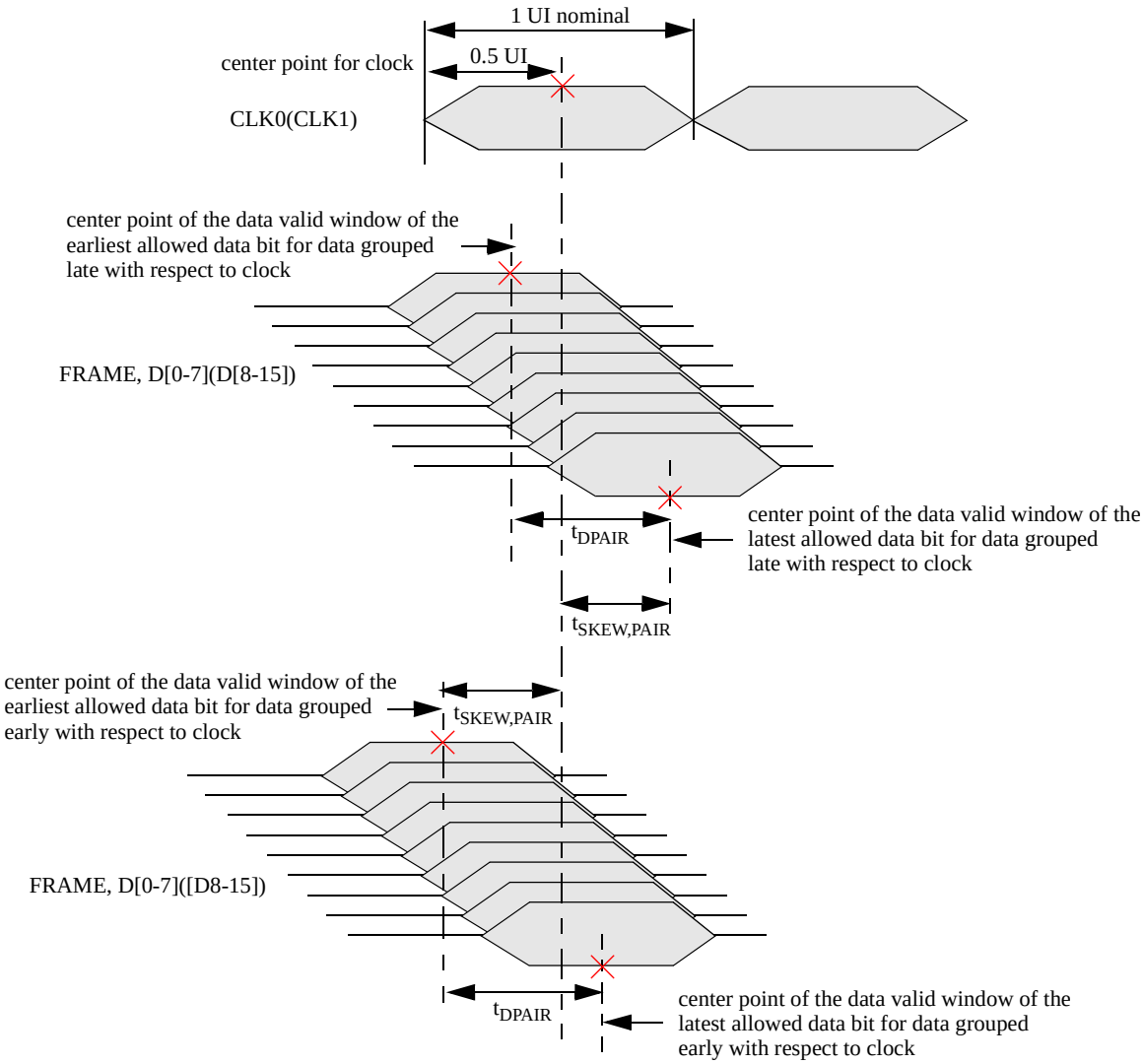


Figure 8-10. Static Skew Diagram

Appendix A
Interface Management (Informative)

This appendix 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" status to the "Port OK" status in the Port n Error and Status CSR when both halves of the state machine are in their OK 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 "OK" states (OK and OK_maint_trn). The optional OK_maint_trn state, shaded in Figure A-1, is used to adjust the device input port sampling circuitry during system operation.

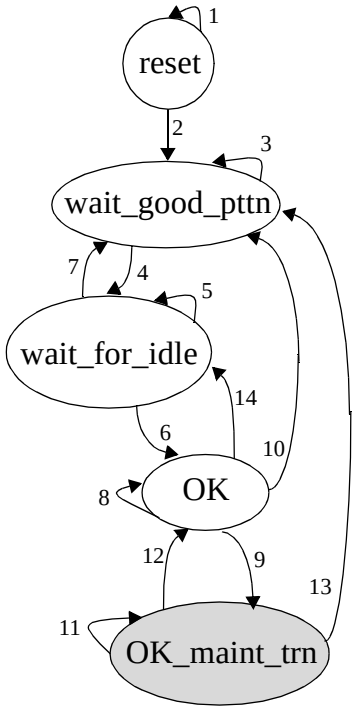


Figure A-1. Input port training state machine

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

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

Part IV

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	Wait for the sampling circuitry to indicate that it is calibrated.	Remain in this state until the sampling circuitry is calibrated.
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 link-request/send-training control symbols are legal.

A

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

Arc	Current State	Next state	cause	Comments
6	wait_for_idle	OK	Idle control symbol has been received	This transition indicates that the input port is ready to start receiving packets and other control symbols. Due to input/link delays the input port may see an extra idle/training pattern sequence when finishing the alignment sequence.
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	OK	OK	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	OK	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.
10	OK	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 OK state is re-entered to attempt to recover possible lost data.
11	OK_maint_trn	OK_maint_trn	Training patterns have 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	OK_maint_trn	OK	The complete sequence of 256 training patterns followed by an idle has been received and the sampling circuitry is still calibrated.	Sampling circuitry has been adjusted.

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

Arc	Current State	Next state	cause	Comments
13	OK_maint_trn	wait_good_pttn	Sampling circuitry is no longer calibrated.	Both the input port and output port restart the alignment 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.
14	OK	wait_for_idle	The input port receives a link-request/send-training control symbol immediately followed by a training pattern	The attached device is no longer calibrated and has re-started the alignment sequence.

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 "OK" states (OK and OK_maint_trn for the input port, and OK, OK_send_trn_req and OK_send_trn_pttn for the output port). The optional OK_send_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 OK_maint_trn state in the input port state machine.

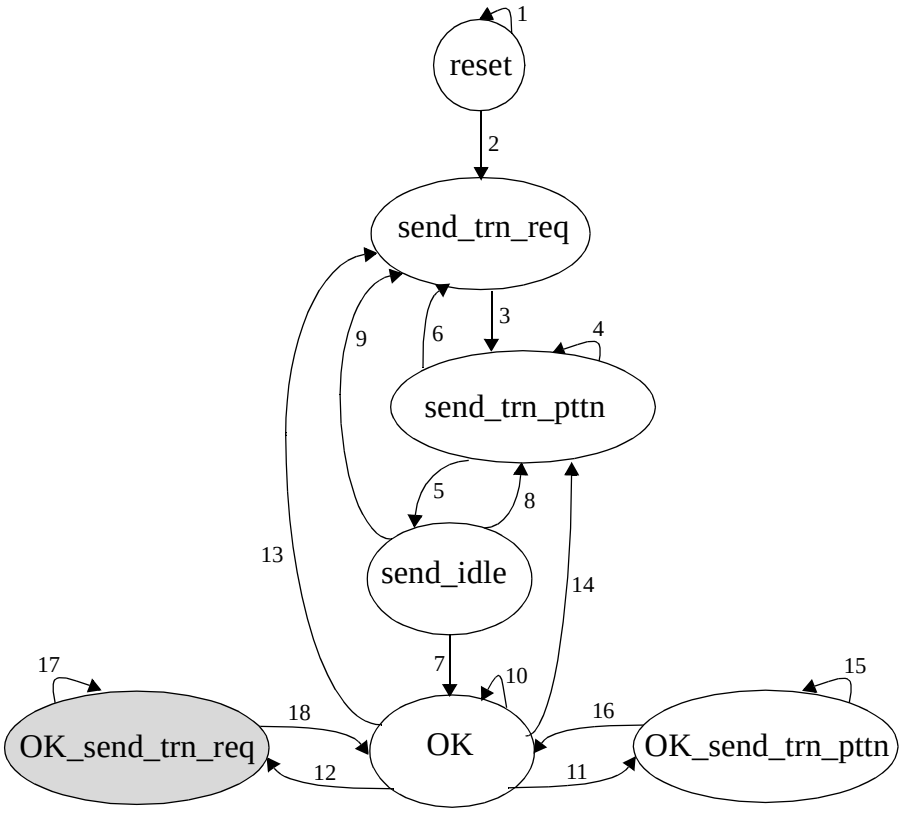


Figure A-2. Output port training state machine

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. In the send_idle state, one idle control symbol is 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_idle	OK	The input port is in state OK	Ready to start sending packets and any control symbol.
8	send_idle	send_trn_pttn	The input port is not in OK or wait_good_pttn state	The output port will send 256 iterations of the of the training pattern
9	send_idle	send_trn_req	The input port is in state wait_good_pttn	Transition to send_trn_req and start over.
10	OK	OK	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.
11	OK	OK_send_trn_pttn	link-request/send-training followed by a packet or control symbol is received on the input port.	This transition occurs when in the OK state and a maintenance training request is received from the attached device.
12	OK	OK_send_trn_req	The input port wants the attached device to send 256 iterations of the training pattern.	This transition occurs when in the OK state and input port sampling circuitry needs to be adjusted, and is associated with the optional input port OK_maint_trn state.
13	OK	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.
14	OK	send_trn_pttn	A link-request/send-training followed by the training pattern is received on the input port.	The attached device has lost synchronization.
15	OK_send_trn_pttn	OK_send_trn_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	OK_send_trn_pttn	OK	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 yet it maintained alignment.

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

Arc	Current State	Next state	cause	Comments
17	OK_send_trn_req	OK_send_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	OK_send_trn_req	OK	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.

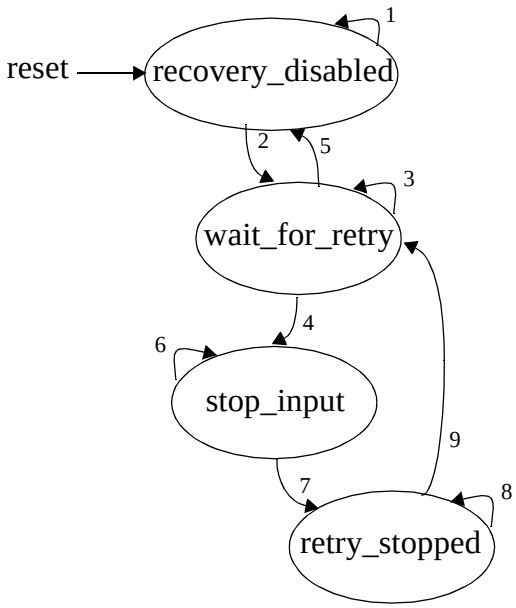


Figure A-3. Input port retry recovery state machine

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.
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.

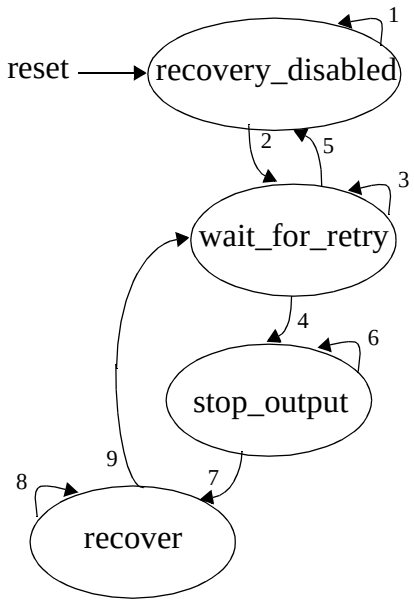


Figure A-4. Output port retry recovery state machine

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.	

Table A-4. 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 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 in Figure A-5.

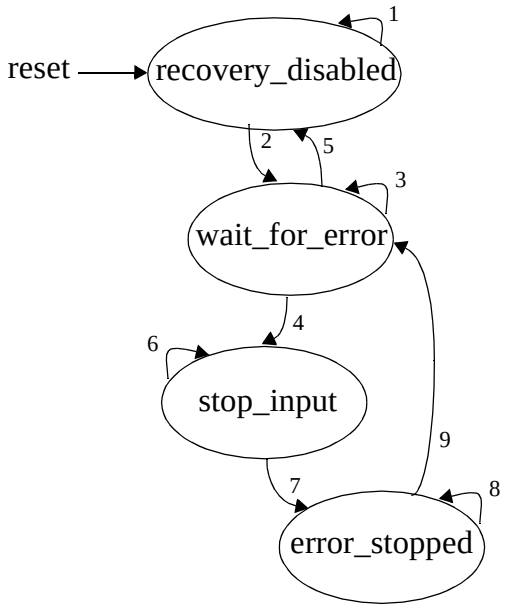


Figure A-5. Input port error recovery state machine

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.	
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.

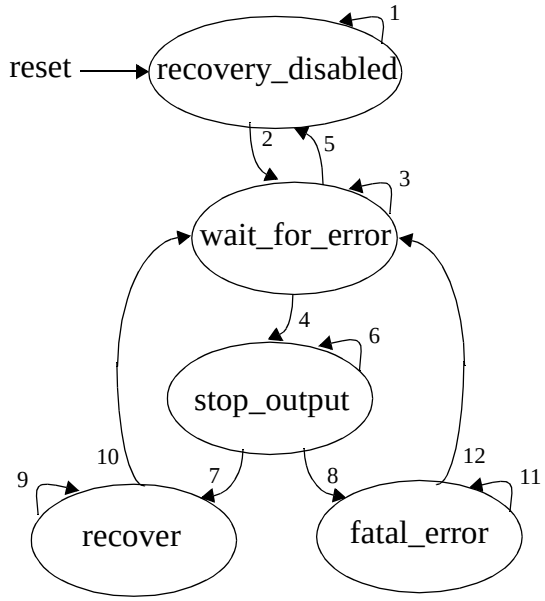


Figure A-6. Output port error recovery state machine

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.
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.

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

Arc	Current State	Next state	cause	Comments
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.

Glossary of Terms and Abbreviations

The glossary contains an alphabetical list of terms, phrases, and abbreviations used in this book.

A

Agent. A processing element that provides services to a processor.

ANSI. American National Standards Institute.

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.

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.

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.

Cache line. A contiguous block of data that is the standard memory access size for a processor within a system.

Capability registers (CARs). A set of read-only registers that allows a processing element to determine another processing element's capabilities.

CCITT. Consultive Communication for International Telegraph and Telephone.

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.

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 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 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.

DLL. Delay lock loop.

Doorbell. A port on a device that is capable of generating an interrupt to a processor.

Double-data-rate clock. A data reference signal that indicates new valid data on both low-to-high and high-to-low transitions of the clock.

Double-word. An eight byte quantity, aligned on eight byte boundaries.

GLO

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.

EOP. End of packet.

Ethernet. A common local area network (LAN) technology.

External processing element. A processing element other than the processing element in question.

F

Field or Field name. A sub-unit of a register, where bits in the register are named and defined.

FIFO. First in, first out.

First symbol. The leading 16 bits of a packet.

FPGA. Field programmable gate array.

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.

Host. A processing element responsible for exploring and initializing all or a portion of a RapidIO based system.

I

I_2O. Intelligent I/O architecture specification.

Initiator. The origin of a packet on the RapidIO interconnect, also referred to as a source.

I/O. Input-output.

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.

LVDS. Low voltage differential signaling.

M

Mailbox. Dedicated hardware that receives messages.

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.

MFA. Message frame address.

MFD. Message frame descriptor.

Microprocessor. A computer processor on a microchip.

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.

NRZ signal. No return to zero signal.

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.

PCB. Printed circuit board.

Peripheral component interface (PCI). A bus commonly used for connecting I/O devices in a system.

GLO

PLL. Phase lock loop.

Port-write. An address-less maintenance 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.

Remote memory. Memory associated with a processing element other than the processing element in question.

ROM. Read-only memory.

S

SECDED. Single error correction, double error detection.

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, typically requiring atomic operations.

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.

Symbol. A 16-bit quantity.

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.

GLO

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.

INDEX

IND

INDEX

IND

INDEX

R

S

T

IND

INDEX

IND