RapidIO™ Interconnect Specification Part 1: Input/Output Logical Specification

3.2, 1/2016

RapidIO.org

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
1.3	Technical changes: incorporate Rev 1.2 errata 1 as applicable, the following errata showings: 03-05-00006.001, 03-12-00001.001, 04-02-00001.002 and the following new features showings: 04-05-00005.001 Converted to ISO-friendly templates, re-formatted	02/23/2005
2.0	Technical changes: errata showings 06-11-00000.001, 06-11-00001.004	06/14/2007
2.1	Technical changes: errata showing 07-06-00000.010	07/09/2009
2.2	Technical changes: errata showings 09-09-00001.002, 10-08-00000.003, 10-08-00001.005, Consolidated Comments on 11-01-00000.000	05/05/2011
3.0	Changed RTA contact information. No technical changes.	11/9/2013
3.1	Addition of LCS Disable functionality.	09/18/2014
3.2	No technical changes.	01/28/2016

Table of Contents

Chapter 1 Overview

1.1 Introduction 9
1.2 Overview 9
1.3 Features of the Input/Output Specification 10
1.3.1 Functional Features 10
1.3.2 Physical Features 10
1.3.3 Performance Features 10
1.4 Contents 11
1.5 Terminology 11
1.6 Conventions 11

Chapter 2 System Models

2.1 Introduction 13
2.2 Processing Element Models 13
2.2.1 Processor-Memory Processing Element Model 13
2.2.2 Integrated Processor-Memory Processing Element Model 14
2.2.3 Memory-Only Processing Element Model 14
2.2.4 Processor-Only Processing Element 15
2.2.5 I/O Processing Element 15
2.2.6 Switch Processing Element 15
2.3 System Issues 16
2.3.1 Operation Ordering 16
2.3.2 Transaction Delivery 18
2.3.2.1 Unordered Delivery System Issues 18
2.3.2.2 Ordered Delivery System Issues 19
2.3.3 Deadlock Considerations 19

Chapter 3 Operation Descriptions

3.1 Introduction 21
3.2 I/O Operations Cross Reference 22
3.3 I/O Operations 22
3.3.1 Read Operations 23
3.3.2 Write and Streaming-Write Operations 23
3.3.3 Write-With-Response Operations 24
3.3.4 Atomic (Read-Modify-Write) Operations 24
3.4 System Operations 25
3.4.1 Maintenance Operations 25
3.5 Endian, Byte Ordering, and Alignment 25

Table of Contents

Chapter 4 Packet Format Descriptions

4.1 Request Packet Formats ... 29
4.1.1 Addressing and Alignment ... 30
4.1.2 Field Definitions for All Request Packet Formats ... 30
4.1.3 Type 0 Packet Format (Implementation-Defined) ... 33
4.1.4 Type 1 Packet Format (Reserved) ... 33
4.1.5 Type 2 Packet Format (Request Class) ... 33
4.1.6 Type 3–4 Packet Formats (Reserved) ... 34
4.1.7 Type 5 Packet Format (Write Class) ... 34
4.1.8 Type 6 Packet Format (Streaming-Write Class) ... 35
4.1.9 Type 7 Packet Format (Reserved) ... 36
4.1.10 Type 8 Packet Format (Maintenance Class) ... 36
4.1.11 Type 9–11 Packet Formats (Reserved) ... 38
4.2 Response Packet Formats ... 38
4.2.1 Field Definitions for All Response Packet Formats ... 38
4.2.2 Type 12 Packet Format (Reserved) ... 39
4.2.3 Type 13 Packet Format (Response Class) ... 39
4.2.4 Type 14 Packet Format (Reserved) ... 40
4.2.5 Type 15 Packet Format (Implementation-Defined) ... 40

Chapter 5 Input/Output Registers

5.1 Register Summary ... 41
5.2 Reserved Register, Bit and Bit Field Value Behavior ... 42
5.3 Extended Features Data Structure ... 43
5.4 Capability Registers (CARs) ... 45
5.4.1 Device Identity CAR ... 45
5.4.2 Device Information CAR ... 46
5.4.3 Assembly Identity CAR ... 47
5.4.4 Assembly Information CAR ... 48
5.4.5 Processing Element Features CAR ... 49
5.4.6 Switch Port Information CAR ... 50
5.4.7 Source Operations CAR ... 51
5.4.8 Destination Operations CAR ... 52
5.5 Command and Status Registers (CSRs) ... 53
5.5.1 Processing Element Logical Layer Control CSR ... 53
5.5.2 Local Configuration Space Base Address 0 CSR ... 54
5.5.3 Local Configuration Space Base Address 1 CSR ... 55

List of Figures

2-1 A Possible RapidIO-Based Computing System....13
2-2 Processor-Memory Processing Element Example....14
2-3 Integrated Processor-Memory Processing Element Example....14
2-4 Memory-Only Processing Element Example....15
2-5 Processor-Only Processing Element Example....15
2-6 Switch Processing Element Example....16
3-1 Read Operation....23
3-2 Write and Streaming-Write Operations....24
3-3 Write-With-Response Operation....24
3-4 Atomic (Read-Modify-Write) Operation....25
3-5 Maintenance Operation....25
3-6 Byte Alignment Example....26
3-7 Half-Word Alignment Example....26
3-8 Word Alignment Example....26
3-9 Data Alignment Example....27
4-1 Type 2 Packet Bit Stream Format....34
4-2 Type 5 Packet Bit Stream Format....35
4-3 Type 6 Packet Bit Stream Format....36
4-4 Type 8 Request Packet Bit Stream Format....37
4-5 Type 8 Response Packet Bit Stream Format....38
4-6 Type 13 Packet Bit Stream Format....39
5-1 Example Extended Features Data Structure....44

List of Figures

Blank page

List of Tables

4-1 Request Packet Type to Transaction Type Cross Reference 29
4-2 General Field Definitions for All Request Packets..31
4-3 Read Size (rdsize) Definitions 31
4-4 Write Size (wrsize) Definitions 32
4-5 Transaction Fields and Encodings for Type 2 Packets 34
4-6 Transaction Fields and Encodings for Type 5 Packets 35
4-7 Specific Field Definitions and Encodings for Type 8 Packets 37
4-8 Response Packet Type to Transaction Type Cross Reference.......................................38
4-9 Field Definitions and Encodings for All Response Packets 38
5-1 I/O Register Map 41
5-2 Configuration Space Reserved Access Behavior...42
5-3 Bit Settings for Device Identity CAR...45
5-4 Bit Settings for Device Information CAR 46
5-5 Bit Settings for Assembly Identity CAR 47
5-6 Bit Settings for Assembly Information CAR..48
5-7 Bit Settings for Processing Element Features CAR...49
5-8 Bit Settings for Switch Port Information CAR..50
5-9 Bit Settings for Source Operations CAR 51
5-10 Bit Settings for Destination Operations CAR...52
5-11 Bit Settings for Processing Element Logical Layer Control CSR................................53
5-12 Bit Settings for Local Configuration Space Base Address 0 CSR 54
5-13 Bit Settings for Local Configuration Space Base Address 1 CSR 55

List of Tables

Blank page

Chapter 1 Overview

1.1 Introduction

This chapter provides an overview of the *RapidIO Part 1: Input/Output Logical Specification*, including a description of the relationship between this specification and the other specifications of the RapidIO interconnect.

1.2 Overview

The *RapidIO Part 1: Input/Output Logical Specification* is one of the RapidIO logical layer specifications that define the interconnect's overall protocol and packet formats. This layer contains the information necessary for end points to process a transaction. Other RapidIO logical layer specifications include *RapidIO Part 2: Message Passing Logical Specification* and *RapidIO Part 5: Globally Shared Memory 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 encodings for the transport and physical layers lower in the specification hierarchy.

RapidIO is a definition of a system interconnect. System concepts such as processor programming models, memory coherency models and caching are beyond the scope of the RapidIO architecture. The support of memory coherency models, through caches, memory directories (or equivalent, to hold state and speed up remote memory access) is the responsibility of the end points (processors, memory, and possibly I/O devices), using RapidIO operations. RapidIO provides the operations to construct a wide variety of systems, based on programming models that range from strong consistency through total store ordering to weak ordering. Inter-operability between end points supporting different coherency/caching/directory models is not guaranteed. However, groups of end-points with conforming models can be linked to others conforming to different models on the same RapidIO fabric. These different groups can communicate through RapidIO messaging or I/O operations. Any reference to these areas within the RapidIO architecture specification are for illustration only.

1.3 Features of the Input/Output Specification

The following are features of the RapidIO I/O specification designed to satisfy the needs of various applications and systems:

1.3.1 Functional Features

- A rich variety of transaction types, such as DMA-style read and writes, that allow efficient I/O systems to be built.
- 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.
- 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.

1.3.2 Physical Features

- RapidIO packet definition is independent of the width of the physical interface to other devices on the interconnect fabric.
- The protocols and packet formats are independent of the physical interconnect topology. The protocols work whether the physical fabric is a point-to-point ring, a bus, a switched multi-dimensional network, a duplex serial connection, and so forth.
- RapidIO is not dependent on the bandwidth or latency of the physical fabric.
- The protocols handle out-of-order packet transmission and reception.
- Certain devices have bandwidth and latency requirements for proper operation. RapidIO does not preclude an implementation from imposing these constraints within the system.

1.3.3 Performance Features

- Packet headers must be as small as possible to minimize the control overhead and be organized for fast, efficient assembly and disassembly.
- 48- and 64-bit addresses are required in the future, and must be supported initially.
- Multiple transactions must be allowed concurrently in the system, otherwise a majority of the potential system throughput is wasted.

1.4 Contents

Following are the contents of the *RapidIO Part 1: Input/Output Logical Specification:*

- Chapter 1, “Overview” (this chapter) provides an overview of the specification
- Chapter 2, “System Models,” introduces some possible devices that could participate in a RapidIO system environment. Transaction ordering and deadlock prevention are discussed.
- Chapter 3, “Operation Descriptions,” describes the set of operations and transactions supported by the RapidIO I/O protocols.
- Chapter 4, “Packet Format Descriptions,” contains the packet format definitions for the I/O specification. The two basic types, request and response packets, with their sub-types and fields are defined.
- Chapter 5, “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 I/O logical specification are explained. Refer to the other RapidIO logical, transport, and physical specifications of interest to determine a complete list of registers and bit definitions.

1.5 Terminology

Refer to the Glossary at the back of this document.

1.6 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.
x	This value is a don’t care

Chapter 2 System Models

2.1 Introduction

This overview introduces some possible devices in a RapidIO system.

2.2 Processing Element Models

Figure 2-1 describes a possible RapidIO-based computing system. The processing element is a computer device such as a processor attached to a local memory and 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.

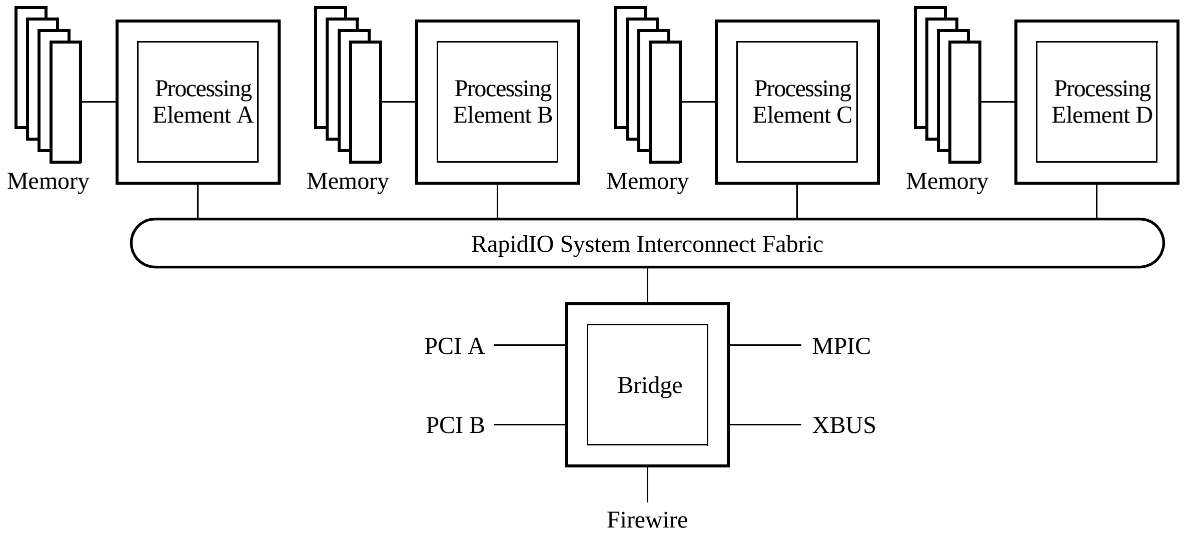


Figure 2-1. A Possible RapidIO-Based Computing System

The following sections describe several possible processing elements.

2.2.1 Processor-Memory Processing Element Model

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

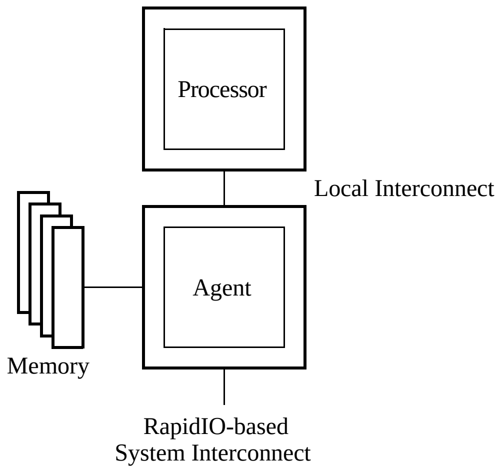


Figure 2-2. Processor-Memory Processing Element Example

2.2.2 Integrated Processor-Memory Processing Element Model

Another form of a processor-memory processing element is a fully integrated component that is designed specifically to connect to a RapidIO interconnect system as shown in Figure 2-3. This type of device integrates a memory system and other support logic with a processor on the same piece of silicon or within the same package.

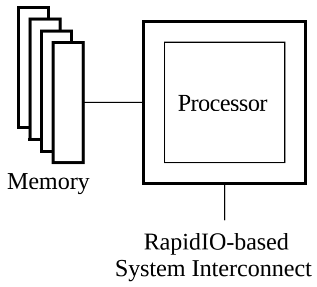


Figure 2-3. Integrated Processor-Memory Processing Element Example

2.2.3 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 2-4. 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.

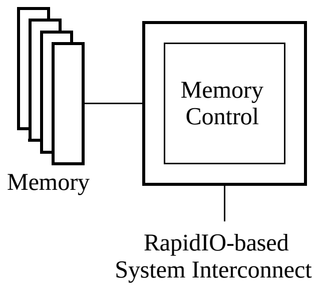


Figure 2-4. Memory-Only Processing Element Example

2.2.4 Processor-Only Processing Element

Similar to a memory-only element, a processor-only element has no local memory. A processor-only processing element is shown in Figure 2-5.

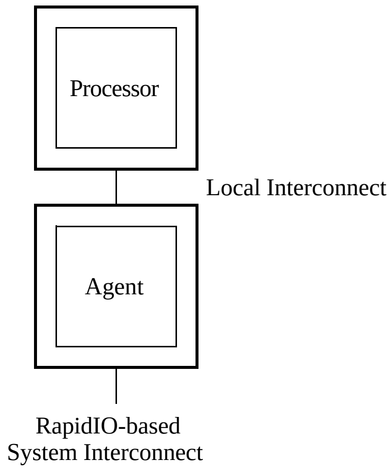


Figure 2-5. Processor-Only Processing Element Example

2.2.5 I/O Processing Element

This type of processing element is shown as the bridge in Figure 2-1. 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.

2.2.6 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 hybrid processing element may combine a switch with end point functionality. A possible switch is shown in Figure 2-6. Behavior of the switches, and the interconnect fabric in general, is addressed in the *RapidIO Common Transport Specification*.

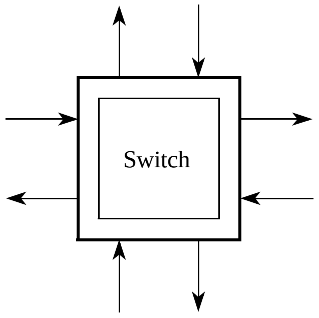


Figure 2-6. Switch Processing Element Example

2.3 System Issues

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

2.3.1 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 an 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). RapidIO 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 flowID A having the lowest priority, flowID 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.

2.3.2 Transaction Delivery

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

2.3.2.1 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.

2.3.2.2 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.

2.3.3 Deadlock Considerations

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

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

In order to produce a system with no chance of deadlock it is required that a deadlock free topology be provided for response-less operations. Dependency loops to single direction packets can exist in unconstrained switch topologies. Often the dependency loop can be avoided with simple routing rules. Topologies like hypercubes or three-dimensional meshes physically contain loops. In both cases, routing is done in several dimensions (x,y,z). If routing is constrained to the x dimension, then y, then z (dimension ordered routing), 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 3 Operation Descriptions

3.1 Introduction

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 4, “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 must accept and not complete the subsequent out-of-order requests until the missing transactions in the sequence have been received and completed.

3.2 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 3.3.1	Type 2 Section 4.1.5
Write	NWRITE	Write operation	Write	Section 3.3.2	Type 5 Section 4.1.7
Write-with-response	NWRITE_R, RESPONSE	Write operation	Write	Section 3.3.3	Type 5 Section 4.1.7
Streaming-write	SWRITE	Write operation	Write	Section 3.3.2	Type 6 Section 4.1.8
Atomic (read-modify-write)	ATOMIC, RESPONSE	Read-modify-write operation	Write	Section 3.3.4	Type 2 Section 4.1.5 Type 5 Section 4.1.7
Maintenance	MAINTENANCE	System exploration, initialization, and maintenance operation	not applicable	Section 3.4.1	Type 8 Section 4.1.10

3.3 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 3-6 through Figure 3-8.

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

3.3.1 Read Operations

The read operation, consisting of the NREAD and RESPONSE transactions (typically a DONE response) as shown in Figure 3-1, 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.

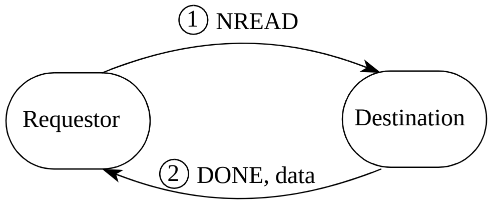


Figure 3-1. Read Operation

3.3.2 Write and Streaming-Write Operations

The write and streaming-write operations, consisting of the NWRITE and SWRITE transactions as shown in Figure 3-2, 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 4-4. 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.

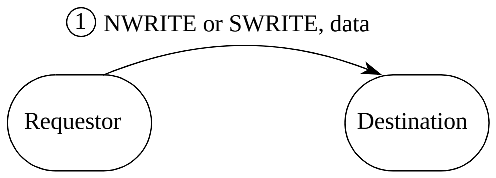


Figure 3-2. Write and Streaming-Write Operations

3.3.3 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 3-3, 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.

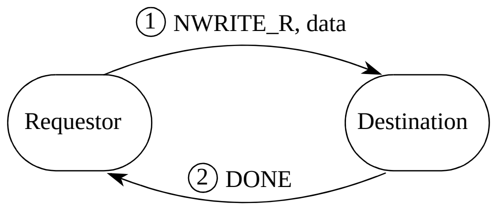


Figure 3-3. Write-With-Response Operation

3.3.4 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 3-4, 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 5-9 and Table 5-10). Of these, only test-and-swap, compare-and-swap, 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.

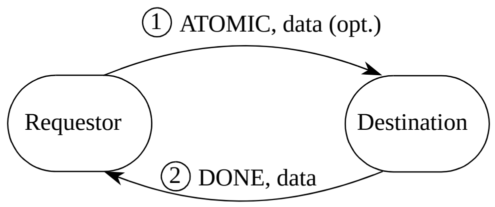


Figure 3-4. Atomic (Read-Modify-Write) Operation

3.4 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 3-6 through Figure 3-8.

3.4.1 Maintenance Operations

The maintenance operation, which can consist of more than one MAINTENANCE transaction as shown in Figure 3-5, 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.

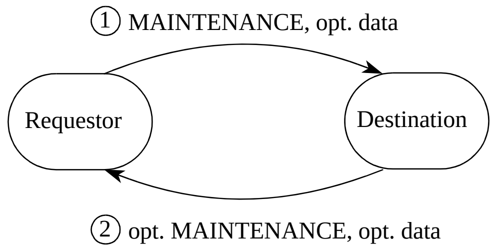
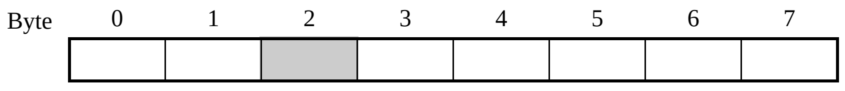


Figure 3-5. Maintenance Operation

3.5 Endian, Byte Ordering, and Alignment

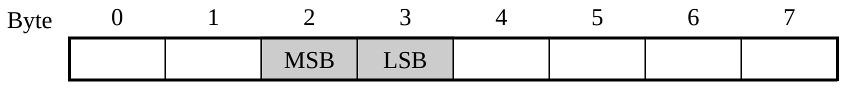
RapidIO has double-word (8-byte) aligned big-endian data payloads. This means that the RapidIO interface to devices that are little-endian shall perform the proper endian transformation to format a data payload.

Operations that specify data quantities that are less than 8 bytes shall have the bytes aligned to their proper byte position within the big-endian double-word, as in the examples shown in Figure 3-6 through Figure 3-8.



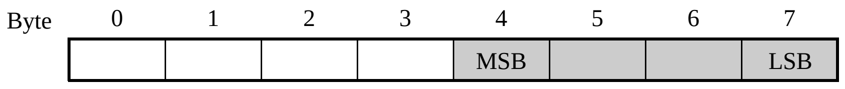
Byte address 0x0000_0002, the proper byte position is shaded.

Figure 3-6. Byte Alignment Example



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

Figure 3-7. Half-Word Alignment Example



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

Figure 3-8. 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 3-9 shows a 48-byte data stream that a processing element wishes to write to another processing element through the interconnect fabric. The data displayed in the figure is big-endian and double-word aligned with the bytes to be written shaded in grey. Because the start of the stream and the end of the stream are not aligned to a double-word boundary, the sending processing element shall break the stream into three transactions as shown in the figure.

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

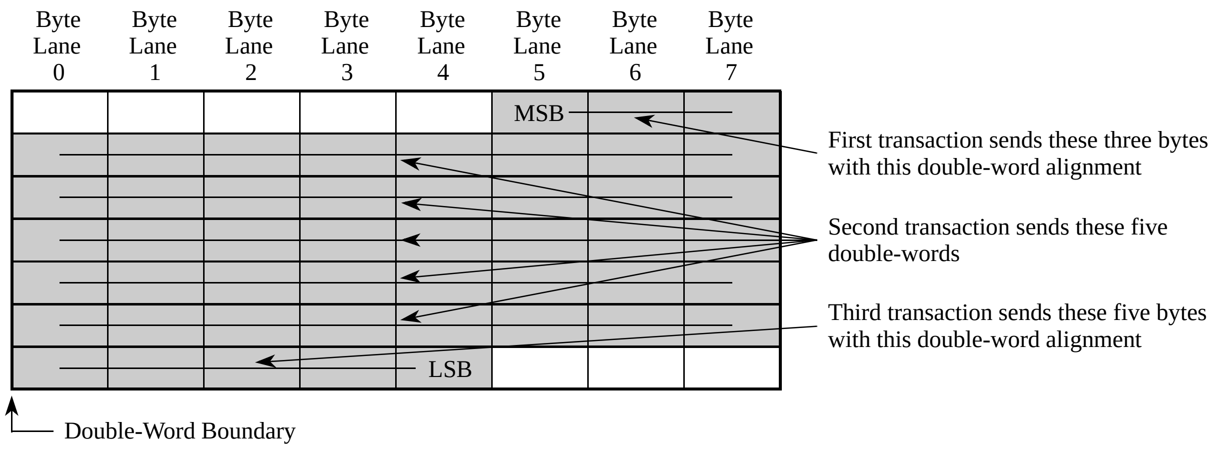

Byte Lane 0
Byte Lane 1
Byte Lane 2
Byte Lane 3
Byte Lane 4
Byte Lane 5
Byte Lane 6
Byte Lane 7
MSB
LSB
First transaction sends these three bytes with this double-word alignment
Second transaction sends these five double-words
Third transaction sends these five bytes with this double-word alignment
Double-Word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

Figure 3-9. Data Alignment Example

Blank page

Chapter 4 Packet Format Descriptions

This chapter contains the packet format definitions for the *RapidIO Part 1: 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.

4.1 Request Packet Formats

A request packet is issued by a processing element that needs a remote processing element to accomplish some activity on its behalf, such as a memory read operation. The request packet format types and their transactions for the I/O Logical Specification are shown in Table 4-1 below.

Table 4-1. Request Packet Type to Transaction Type Cross Reference

Request Packet Format Type	Transaction Type	Definition	Document Section No.
Type 0	Implementation-defined	Defined by the device implementation	Section 4.1.3
Type 1	—	Reserved	Section 4.1.4
Type 2	ATOMIC set	Read-write 1s to specified address	Section 4.1.5
	ATOMIC clear	Read-write 0s to specified address	
	ATOMIC increment	Read-increment-write to specified address	
	ATOMIC decrement	Read-decrement-write to specified address	
	NREAD	Read specified address	
Type 3-4	—	Reserved	Section 4.1.6

Table 4-1. Request Packet Type to Transaction Type Cross Reference (Continued)

Request Packet Format Type	Transaction Type	Definition	Document Section No.
Type 5	ATOMIC test-and-swap	Read-test=0-swap-write to specified address	Section 4.1.7
	ATOMIC swap	Read-write to specified address	
	ATOMIC compare-and-swap	Read-test=first data-write second data to specified address	
	NWRITE	Write specified address	
	NWRITE_R	Write specified address, notify source of completion	
Type 6	SWRITE	Write specified address	Section 4.1.8
Type 7	—	Reserved	Section 4.1.9
Type 8	MAINTENANCE	Read or write device configuration registers and perform other system maintenance tasks	Section 4.1.10
Type 9-11	—	Reserved	Section 4.1.11

4.1.1 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 3.5, “Endian, Byte Ordering, and Alignment.”

4.1.2 Field Definitions for All Request Packet Formats

Table 4-2 through Table 4-4 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 4-2. 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 4-3, Table 4-4 and Section 3.5.
rdsize	Data size for read transactions, used in conjunction with the word pointer (wdptr) bit—see Table 4-3 and Section 3.5.
wrsize	Write data size for sub-double-word transactions, used in conjunction with the word pointer (wdptr) bit—see Table 4-4 and Section 3.5. 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	Bits [0-28] of byte address [0-31] of the double-word physical address

Table 4-3. 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

Table 4-3. Read Size (rdsize) Definitions (Continued)

wdptr	rdsize	Number of Bytes	Byte Lanes
0b1	0b1000	4	0b00001111
0b0	0b1001	6	0b11111100
0b1	0b1001	6	0b00111111
0b0	0b1010	7	0b11111110
0b1	0b1010	7	0b01111111
0b0	0b1011	8	0b11111111
0b1	0b1011	16	
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 4-4. 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

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

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

4.1.3 Type 0 Packet Format (Implementation-Defined)

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

4.1.4 Type 1 Packet Format (Reserved)

The type 1 packet format is reserved.

4.1.5 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 4-5. 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 4-5. 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 4-1 displays the type 2 packet with all its fields. The field value 0b0010 in Figure 4-1 specifies that the packet format is of type 2.

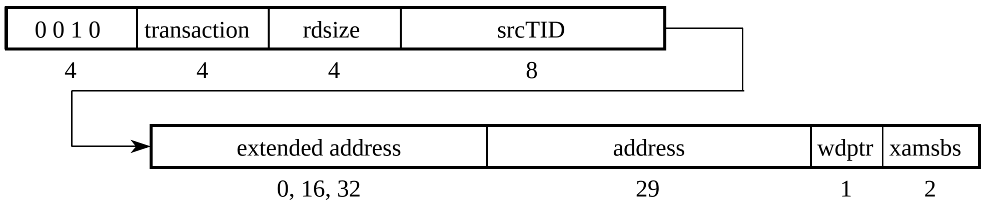


Figure 4-1. Type 2 Packet Bit Stream Format

4.1.6 Type 3–4 Packet Formats (Reserved)

The type 3–4 packet formats are reserved.

4.1.7 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 4-4. 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 4-6. 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.

The ATOMIC swap transaction has the same addressing scheme and data payload restrictions as the ATOMIC test-and-swap transaction.

The ATOMIC compare-and-swap operation is different from the other ATOMIC operations in that it requires two double-words (16 bytes) of data payload.

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

Table 4-6. Transaction Fields and Encodings for Type 5 Packets

Encoding	Transaction Field
0b0000–0011	Reserved
0b0100	NWRITE transaction
0b0101	NWRITE_R transaction
0b0110–1011	Reserved
0b1100	ATOMIC swap: read and return the data, unconditionally write with supplied data.
0b1101	ATOMIC compare-and-swap: read and return the data, if the read data is equal to the first 8 bytes of data payload, write the second 8 bytes of data to the memory location.
0b1110	ATOMIC test-and-swap: read and return the data, compare to 0, write with supplied data if compare is true
0b1111	Reserved

Figure 4-2 displays the type 5 packet with all its fields. The field value 0b0101 in Figure 4-2 specifies that the packet format is of type 5.

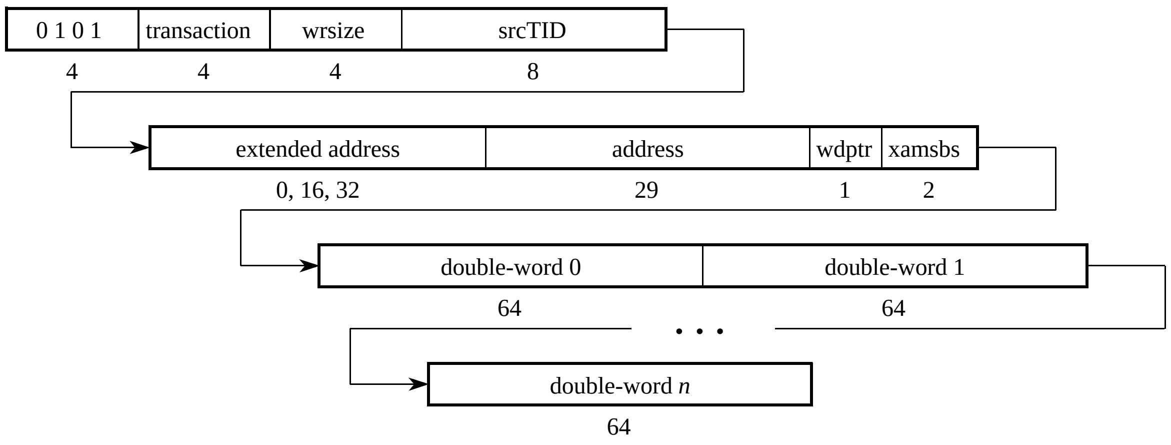


Figure 4-2. Type 5 Packet Bit Stream Format

4.1.8 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 4-4.

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 4-3 displays the type 6 packet with all its fields. The field value 0b0110 in Figure 4-3 specifies that the packet format is of type 6.

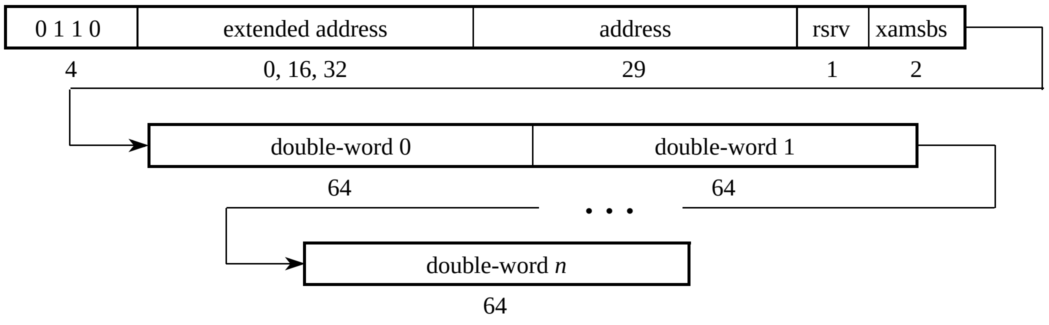


Figure 4-3. Type 6 Packet Bit Stream Format

4.1.9 Type 7 Packet Format (Reserved)

The type 7 packet format is reserved.

4.1.10 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 4-3 and Table 4-4. 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. Maintenance read responses with a status of ERROR may optionally include data in the 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 4-7. Fields that are not specific to type 8 packets are described in Table 4-2.

Table 4-7. 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 4-4 displays a type 8 request (read or write) packet with all its fields. The field value 0b1000 in Figure 4-4 specifies that the packet format is of type 8. The srcTID and config_offset fields are reserved for port-write requests.

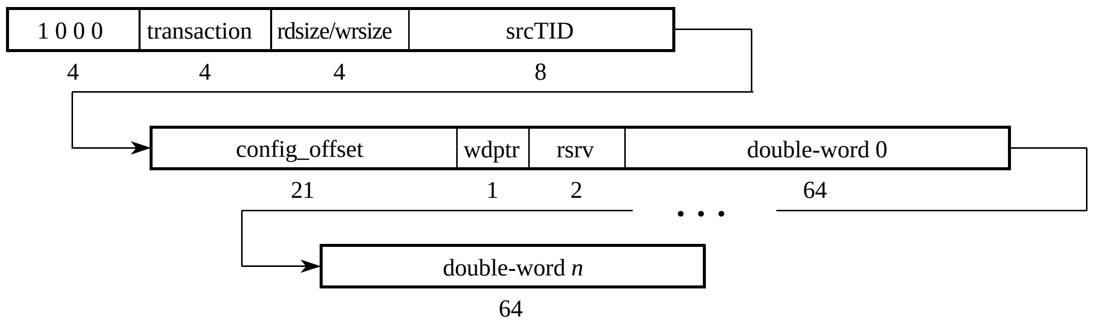


Figure 4-4. Type 8 Request Packet Bit Stream Format

Figure 4-5 displays a type 8 response packet with all its fields.

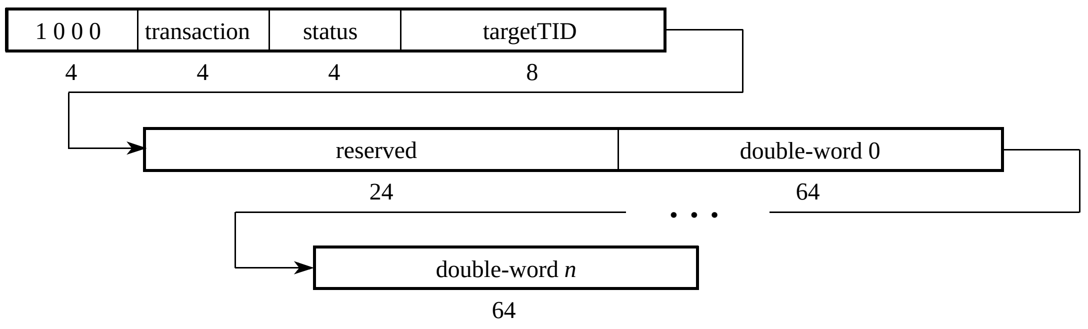


Figure 4-5. Type 8 Response Packet Bit Stream Format

4.1.11 Type 9–11 Packet Formats (Reserved)

The type 9–11 packet formats are reserved.

4.2 Response Packet Formats

A response transaction is issued by a processing element when it has completed a request made 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 4-8.

Table 4-8. Response Packet Type to Transaction Type Cross Reference

Response Packet Format Type	Transaction Type	Definition	Document Section Number
Type 12	—	Reserved	Section 4.2.2
Type 13	RESPONSE	Issued by a processing element when it completes a request by a remote element.	Section 4.2.3
Type 14	—	Reserved	Section 4.2.4
Type 15	Implementation-defined	Defined by the device implementation	Section 4.2.5

4.2.1 Field Definitions for All Response Packet Formats

The field definitions in Table 4-9 apply to more than one of the response packet formats.

Table 4-9. 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 4-9. 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

4.2.2 Type 12 Packet Format (Reserved)

The type 12 packet format is reserved.

4.2.3 Type 13 Packet Format (Response Class)

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

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

When a RESPONSE packet is generated with an "ERROR" status, the Transaction field value may be 0 (RESPONSE transaction with no data) or 8 (RESPONSE transaction with data). In both cases, the RESPONSE packet with an "ERROR" status has no data payload. Processing elements which accept RESPONSE packets with an "ERROR" status shall accept Transaction field values of 0 (RESPONSE transaction with no data) and 8 (RESPONSE transaction with data).

Figure 4-6 illustrates the format and fields of type 13 packets. The field value 0b1101 in Figure 4-6 specifies that the packet format is of type 13.

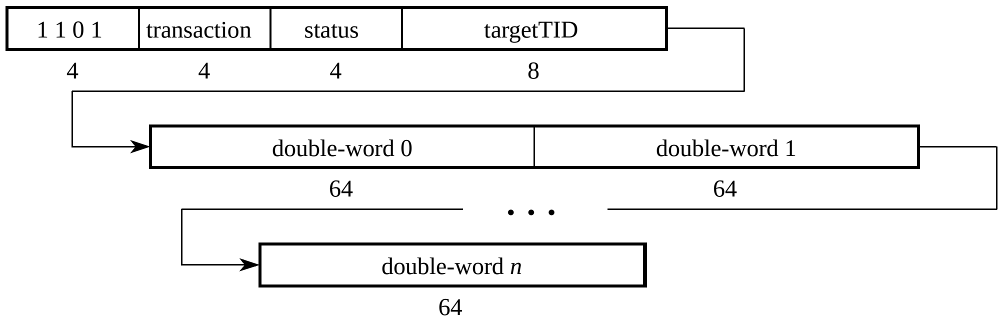


Figure 4-6. Type 13 Packet Bit Stream Format

4.2.4 Type 14 Packet Format (Reserved)

The type 14 packet format is reserved.

4.2.5 Type 15 Packet Format (Implementation-Defined)

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

Chapter 5 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.

5.1 Register Summary

Table 5-1 shows the register map for this RapidIO specification. These capability registers (CARs) and command and status registers (CSRs) can be accessed using 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 5-1. I/O Register Map

Configuration Space Byte Offset	Register Name
0x0	Device Identity CAR
0x4	Device Information CAR
0x8	Assembly Identity CAR
0xC	Assembly Information CAR
0x10	Processing Element Features CAR
0x14	Switch Port Information CAR
0x18	Source Operations CAR
0x1C	Destination Operations CAR

Table 5-1. I/O Register Map (Continued)

Configuration Space Byte Offset	Register Name
0x20–48	Reserved
0x4C	Processing Element Logical Layer Control CSR
0x50	Reserved
0x58	Local Configuration Space Base Address 0 CSR
0x5C	Local Configuration Space Base Address 1 CSR
0x60–FC	Reserved
0x100–FFFC	Extended Features Space
0x10000–FFFFFC	Implementation-defined Space

5.2 Reserved Register, Bit and Bit Field Value Behavior

Table 5-2 describes the required behavior for accesses to reserved register bits and reserved registers for the RapidIO register space,

Table 5-2. Configuration Space Reserved Access Behavior

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

Table 5-2. 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.

When a writable bit field is set to a reserved value, device behavior is implementation specific.

5.3 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 5-1 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.

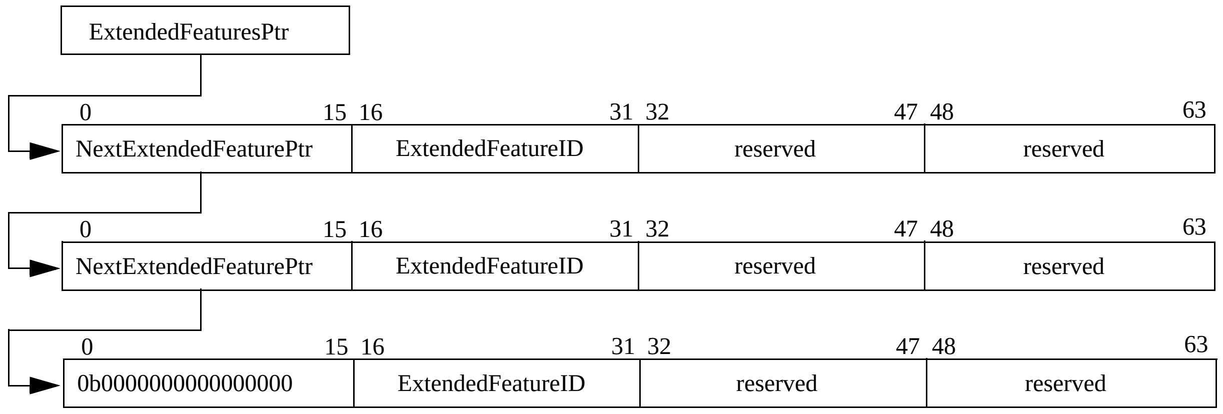


Figure 5-1. Example Extended Features Data Structure

5.4 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 5-2 for the required behavior for accesses to reserved registers and register bits.

CARs are big-endian with bit 0 the most significant bit.

5.4.1 Device Identity CAR (Configuration Space Offset 0x0)

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 RapidIO.org.

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 5-3.

Table 5-3. Bit Settings for Device Identity CAR

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

5.4.2 Device Information CAR (Configuration Space Offset 0x4)

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 5-4.

Table 5-4. Bit Settings for Device Information CAR

Bit	Field Name	Description
0-31	DeviceRev	Device revision level

5.4.3 Assembly Identity CAR (Configuration Space Offset 0x8)

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 RapidIO.org.

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 5-5.

Table 5-5. Bit Settings for Assembly Identity CAR

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

If the value of the AssyIdentity and/or AssyVendorIdentity field is not uniquely known at the time of device fabrication, such field shall be writable through a path and mechanism independent from RapidIO and shall be written with the appropriate value before the register can be accessed through RapidIO. The means for setting the values of these fields is independent of RapidIO and are outside the scope of this specification.

5.4.4 Assembly Information CAR (Configuration Space Offset 0xC)

This register contains additional information about the assembly; see Table 5-6

Table 5-6. 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

If the value of the AssyRev field is not uniquely known at the time of device fabrication, such field shall be writable through a path and mechanism independent from RapidIO and shall be written with the appropriate value before the register can be accessed through RapidIO. The means for setting the values of this field is independent of RapidIO and are outside the scope of this specification.

5.4.5 Processing Element Features CAR (Configuration Space Offset 0x10)

This register identifies the major functionality provided by the processing element; see Table 5-7.

Table 5-7. 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

5.4.6 Switch Port Information CAR (Configuration Space Offset 0x14)

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 5-8.

Table 5-8. Bit Settings for Switch Port Information CAR

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

5.4.7 Source Operations CAR (Configuration Space Offset 0x18)

This register defines the set of RapidIO IO logical operations that can be issued by this processing element; see Table 5-9. 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. For devices that have only switch functionality only bit 29 is valid. RapidIO switches shall be able to route any packet.

Table 5-9. 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
20-21	—	Reserved
22	Atomic (compare-and-swap)	PE can support an atomic compare-and-swap operation
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	Atomic (swap)	PE can support an atomic swap operation
29	Port-write	PE can support a port-write operation
30–31	Implementation Defined	Defined by the device implementation

5.4.8 Destination Operations CAR (Configuration Space Offset 0x1C)

This register defines the set of RapidIO I/O operations that can be supported by this processing element; see Table 5-10. 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 5-10. Bit Settings for Destination Operations CAR

Bit	Field Name	Description
0-13	—	Reserved
14-15	Implementation Defined	Defined by the device implementation
16	Read	PE can support a read operation
17	Write	PE can support a write operation
18	Streaming-write	PE can support a streaming-write operation
19	Write-with-response	PE can support a write-with-response operation
20-21	—	Reserved
22	Atomic (compare-and-swap)	PE can support an atomic compare-and-swap operation
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	Atomic (swap)	PE can support an atomic swap operation
29	Port-write	PE can support a port-write operation
30-31	Implementation Defined	Defined by the device implementation

5.5 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 5-2 for the required behavior for accesses to reserved registers and register bits.

5.5.1 Processing Element Logical Layer Control CSR (Configuration Space Offset 0x4C)

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

Table 5-11. Bit Settings for Processing Element Logical Layer Control CSR

Bit	Field Name	Description
0-25	—	Reserved
26-27	LCS Disable	LCS Disable controls the operation and function of the Local Configuration Space Base Address CSR 0 and 1 CSRs. LCS Disable shall support the following values: 0b00-0b01 - Non-maintenance Logical I/O requests whose address falls in the address range defined by the Local Configuration Space Base Address CSR 0 and 1 CSRs shall be processed. The Local Configuration Space Base Address CSR 0 and 1 CSRs shall be readable and writeable. 0b10 - Non-maintenance Logical I/O requests whose address falls in the address range defined by the Local Configuration Space Base Address CSR 0 and 1 CSRs shall not be processed. The Local Configuration Space Base Address CSR 0 and 1 CSRs shall be read only. 0b11 - Reserved. Operation of the Local Configuration Space Base Address CSR 0 and 1 is undefined. This field shall be readable and writeable. The reset value of this field shall be 0b00 when the LCS Disable Present field reset value is 0. The reset value of this field shall be 0b10 when the LCS Disable Present field reset value is 1. This field shall be implemented when the LCS Disable Present bit is implemented and indicates the presence of this field. This field shall not be implemented when the LCS Disable Present bit is not implemented.
28	LCS Disable Present	Indicates whether or not the LCS Disable field is implemented. 0 - The LCS Disable field is not present 1 - The LCS Disable field is present This field shall be read only. The reset value of this field is implementation specific. Implementation of this field is optional.
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

5.5.2 Local Configuration Space Base Address 0 CSR (Configuration Space Offset 0x58)

The local configuration space base address 0 register specifies the most significant bits of the local physical address double-word offset for the processing element's configuration register space. See Section 5.5.3 below for a detailed description.

Table 5-12. Bit Settings for Local Configuration Space Base Address 0 CSR

Bit	Field Name	Description
0	—	Reserved
1-16	LCSBA	Reserved for a 34-bit local physical address Reserved for a 50-bit local physical address Bits 0-15 of a 66-bit local physical address
17-31	LCSBA	Reserved for a 34-bit local physical address Bits 0-14 of a 50-bit local physical address Bits 16-30 of a 66-bit local physical address

5.5.3 Local Configuration Space Base Address 1 CSR (Configuration Space Offset 0x5C)

The local configuration space base address 1 register specifies the least significant bits of the local physical address double-word offset for the processing element's configuration register space, allowing 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 operations. The double-word offset is right-justified in the register.

Table 5-13. Bit Settings for Local Configuration Space Base Address 1 CSR

Bit	Field Name	Description
0	LCSBA	Reserved for a 34-bit local physical address Bit 15 of a 50-bit local physical address Bit 31 of a 66-bit local physical address
1-31	LCSBA	Bits 0-30 of a 34-bit local physical address Bits 16-46 of a 50-bit local physical address Bits 32-62 of a 66-bit local physical address

Blank page

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.

B

Big-endian. A byte-ordering method in memory where the address n of a word corresponds to the most significant byte. In an addressed memory word, the bytes are ordered (left to right) 0, 1, 2, 3, with 0 being the most significant byte.

Bridge. A processing element that connects one computer bus to another, allowing a processing element on one bus to access an processing element on the other.

C

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.

Command and status registers (CSRs). A set of registers that allows a processing element to control and determine the status of another processing element's internal hardware.

D

Deadlock. A situation in which two processing elements that are sharing resources prevent each other from accessing the resources, resulting in a halt of system operation.

Destination. The termination point of a packet on the RapidIO interconnect, also referred to as a target.

Device. A generic participant on the RapidIO interconnect that sends or receives RapidIO transactions, also called a processing element.

Direct Memory Access (**DMA**). The process of accessing memory in a device by specifying the memory address directly.

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

E

End point. A processing element which is the source or destination of transactions through a RapidIO fabric.

End point device. A processing element which contains end point functionality.

End point free device. A processing element which does not contain end point functionality.

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

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

F

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

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.

I

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.

M

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.

MSB. Most significant byte.

N

Non-coherent. A transaction that does not participate in any system globally shared memory cache coherence mechanism.

O

Operation. A set of transactions between end point devices in a RapidIO system (requests and associated responses) such as a read or a write.

P

Packet. A set of information transmitted between devices in a RapidIO system.

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

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

Sender. The RapidIO interface output port on a processing element.

Source. The origin of a packet on the RapidIO interconnect, also referred to as an initiator.

SRAM. Static random access memory.

Switch. A multiple port processing element that directs a packet received on one of its input ports to one of its output ports.

T

Target. The termination point of a packet on the RapidIO interconnect, also referred to as a destination.

Transaction. A specific request or response packet transmitted between end point devices in a RapidIO system.

Transaction request flow. A sequence of transactions between two processing elements that have a required completion order at the destination processing element. There are no ordering requirements between transaction request flows.

W

Word. A four byte or 32 bit quantity, aligned on four byte boundaries.

RapidIO™ Interconnect Specification Part 2: Message Passing Logical Specification

3.2, 1/2016

RapidIO.org

Revision History

Revision	Description	Date
1.1	First public release	03/08/2001
1.2	No technical changes	06/26/2002
1.3	Technical changes: incorporate Rev 1.2 errata 1 as applicable, the following errata showings: 03-05-00006.001, 03-07-00001.001, 04-02-00001.002, 04-05-00001.002 and the following new features showings: 02-05-00013.001 Converted to ISO-friendly templates; re-formatted	02/23/2005
2.0	No technical changes	06/14/2007
2.1	No technical changes	07/09/2009
2.2	Technical changes: errata showing: 10-08-00001.005, Consolidated Comments on 11-01-00000.000	05/05/2011
3.0	Changed RTA contact information. No technical changes.	10/11/2013
3.1	No technical changes.	09/18/2014
3.2	No technical changes.	01/28/2016

Questions regarding RapidIO.org, specifications, or membership should be forwarded to:

RapidIO.org
8650 Spicewood Springs #145-515
Austin, TX 78759
512-827-7680 Tel.

Table of Contents

Chapter 1 Overview

1.1 Introduction.......... 9
1.2 Overview.......... 9
1.3 Features of the Message Passing Specification 9
1.3.1 Functional Features.......... 9
1.3.2 Physical Features 10
1.3.3 Performance Features 10
1.4 Contents 10
1.5 Terminology.......... 11
1.6 Conventions 11

Chapter 2 System Models

2.1 Introduction.......... 13
2.2 Processing Element Models.......... 13
2.2.1 Processor-Memory Processing Element Model.......... 13
2.2.2 Integrated Processor-Memory Processing Element Model 14
2.2.3 Memory-Only Processing Element Model 14
2.2.4 Processor-Only Processing Element.......... 15
2.2.5 I/O Processing Element 15
2.2.6 Switch Processing Element.......... 15
2.3 Message Passing System Model.......... 16
2.3.1 Data Message Operations 17
2.3.2 Doorbell Message Operations.......... 18
2.4 System Issues.......... 18
2.4.1 Operation Ordering.......... 18
2.4.2 Transaction Delivery.......... 18
2.4.3 Deadlock Considerations.......... 19

Chapter 3 Operation Descriptions

3.1 Introduction.......... 21
3.2 Message Passing Operations Cross Reference 22
3.3 Message Passing Operations.......... 22
3.3.1 Doorbell Operations.......... 22
3.3.2 Data Message Operations 23
3.4 Endian, Byte Ordering, and Alignment 24

Table of Contents

Chapter 4 Packet Format Descriptions

4.1 Introduction.......... 27
4.2 Request Packet Formats.......... 27
4.2.1 Field Definitions for All Request Packet Formats.......... 27
4.2.2 Type 0 Packet Format (Implementation-Defined).......... 28
4.2.3 Type 1–9 Packet Formats (Reserved).......... 28
4.2.4 Type 10 Packet Formats (Doorbell Class).......... 28
4.2.5 Type 11 Packet Format (Message Class).......... 28
4.3 Response Packet Formats 30
4.3.1 Field Definitions for All Response Packet Formats 30
4.3.2 Type 12 Packet Format (Reserved) 31
4.3.3 Type 13 Packet Format (Response Class) 31
4.3.4 Type 14 Packet Format (Reserved) 32
4.3.5 Type 15 Packet Format (Implementation-Defined).......... 32

Chapter 5 Message Passing Registers

5.1 Introduction.......... 33
5.2 Register Summary.......... 33
5.3 Reserved Register, Bit and Bit Field Value Behavior 34
5.4 Capability Registers (CARs) 36
5.4.1 Source Operations CAR.......... 36
5.4.2 Destination Operations CAR 37
5.5 Command and Status Registers (CSRs).......... 38
0.1 Introduction.......... 39
0.2 Definitions and Goals 39
0.3 Message Operations.......... 40
0.4 Inbound Mailbox Structure.......... 41
0.4.1 Simple Inbox.......... 42
0.4.2 Extended Inbox.......... 42
0.4.3 Received Messages.......... 43
0.5 Outbound Message Queue Structure 44
0.5.1 Simple Outbox.......... 44
0.5.2 Extended Outbox 45

List of Figures

2-1 A Possible RapidIO-Based Computing System....13
2-2 Processor-Memory Processing Element Example....14
2-3 Integrated Processor-Memory Processing Element Example....14
2-4 Memory-Only Processing Element Example....15
2-5 Processor-Only Processing Element Example....15
2-6 Switch Processing Element Example....16
3-1 Doorbell Operation....23
3-2 Message Operation....23
3-3 Byte Alignment Example....24
3-4 Half-Word Alignment Example....24
3-5 Word Alignment Example....25
4-1 Type 10 Packet Bit Stream Format....28
4-2 Type 11 Packet Bit Stream Format....30
4-3 target_info Field for Message Responses....32
4-4 Type 13 Packet Bit Stream Format....32
A-1 Simple Inbound Mailbox Port Structure....42
A-2 Inbound Mailbox Structure....43
A-3 Outbound Message Queue....44
A-4 Extended Outbound Message Queue....45

List of Figures

Blank page

List of Tables

3-1 Message Passing Operations Cross Reference 22
4-1 Request Packet Type to Transaction Type Cross Reference 27
4-2 General Field Definitions for All Request Packets....28
4-3 Specific Field Definitions for Type 10 Packets 28
4-4 Specific Field Definitions and Encodings for Type 11 Packets 29
4-5 Response Packet Type to Transaction Type Cross Reference....30
4-6 Field Definitions and Encodings for All Response Packets 31
4-7 Specific Field Definitions for Type 13 Packets 31
5-1 Message Passing Register Map....33
5-2 Configuration Space Reserved Access Behavior....34
5-3 Bit Settings for Source Operations CAR 36
5-4 Bit Settings for Destination Operations CAR....37

List of Tables

Blank page

Chapter 1 Overview

1.1 Introduction

Part 2 is intended for users who need to understand the message passing architecture of the RapidIO interconnect.

1.2 Overview

The *RapidIO Part 2: 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. Other RapidIO logical layer specifications include *RapidIO Part 1: Input/Output Logical Specification* and *RapidIO Part 5: Globally Shared Memory 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.

1.3 Features of the Message Passing Specification

The following are features of the RapidIO I/O specification designed to satisfy the needs of various applications and systems:

1.3.1 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.
- Message passing 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.

1.3.2 Physical Features

- RapidIO packet definition is independent of the width of the physical interface to other devices on the interconnect fabric.
- The protocols and packet formats are independent of the physical interconnect topology. The protocols work whether the physical fabric is a point-to-point ring, a bus, a switched multi-dimensional network, a duplex serial connection, and so forth.
- RapidIO is not dependent on the bandwidth or latency of the physical fabric.
- The protocols handle out-of-order packet transmission and reception.
- Certain devices have bandwidth and latency requirements for proper operation. RapidIO does not preclude an implementation from imposing these constraints within the system.

1.3.3 Performance Features

- Packet headers must be as small as possible to minimize the control overhead and be organized for fast, efficient assembly and disassembly.
- Multiple transactions must be allowed concurrently in the system, otherwise a majority of the potential system throughput is wasted.

1.4 Contents

Following are the contents of *RapidIO Part 2: Message Passing Logical Specification:*

- Chapter 1, “Overview” (this chapter) provides an overview of the specification

- Chapter 2, “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 3, “Operation Descriptions,” describes the set of operations and transactions supported by the RapidIO message passing protocols.
- Chapter 4, “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 5, “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.
- Annex A, “Message Passing Interface (Informative),” contains an informative discussion on possible programming models for the message passing logical layer.

1.5 Terminology

Refer to the Glossary at the back of this document.

1.6 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.
x	This value is a don’t care

Chapter 2 System Models

2.1 Introduction

This overview introduces some possible devices in a RapidIO system.

2.2 Processing Element Models

Figure 2-1 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.

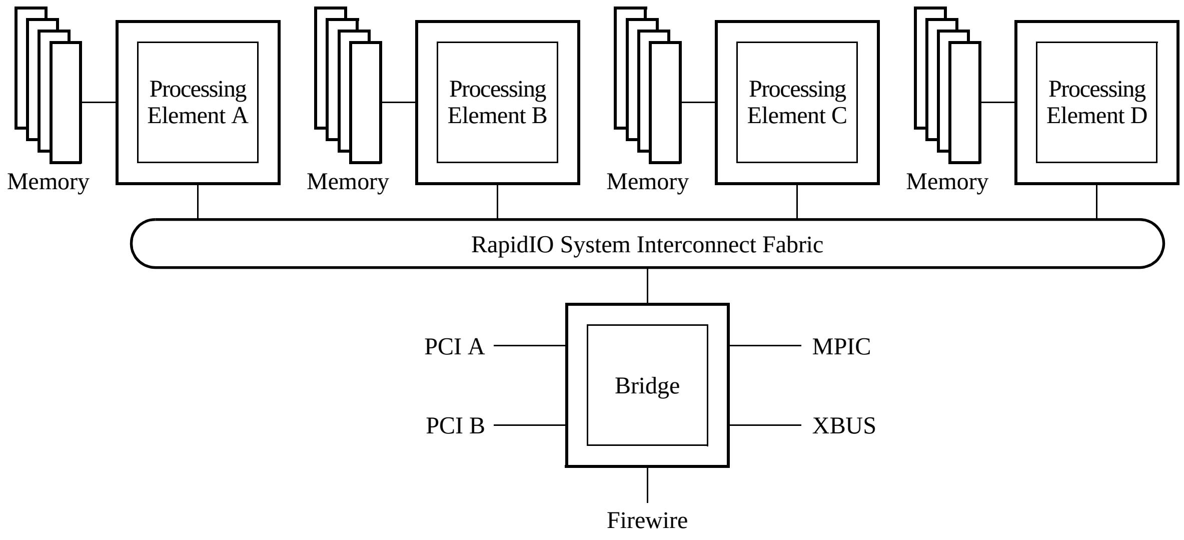


Figure 2-1. A Possible RapidIO-Based Computing System

The following sections describe several possible processing elements.

2.2.1 Processor-Memory Processing Element Model

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

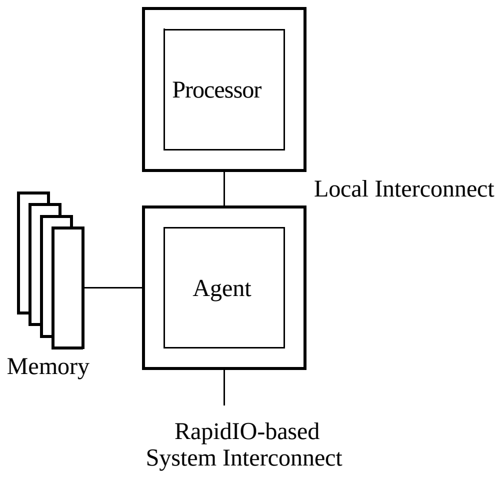


Figure 2-2. Processor-Memory Processing Element Example

2.2.2 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 2-3. This type of device integrates a memory system and other support logic with a processor on the same piece of silicon or within the same package.

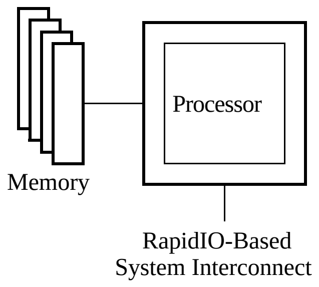


Figure 2-3. Integrated Processor-Memory Processing Element Example

2.2.3 Memory-Only Processing Element Model

A different processing element may not contain a processor at all, but may be a memory-only device as in Figure 2-4. This type of device is much simpler than a processor 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.

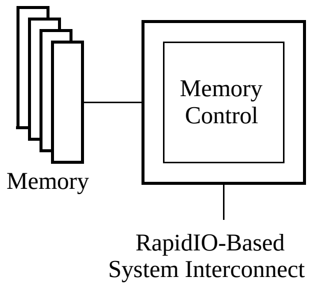


Figure 2-4. Memory-Only Processing Element Example

2.2.4 Processor-Only Processing Element

Similar to a memory-only element, a processor-only element has no local memory. A processor-only processing element is shown in Figure 2-5.

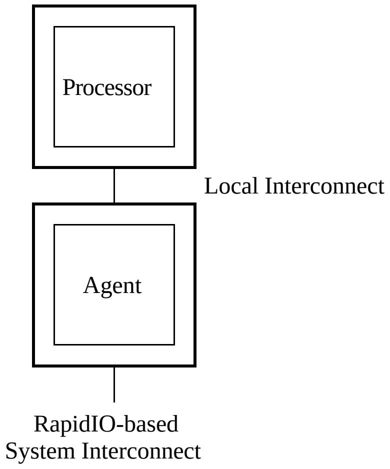


Figure 2-5. Processor-Only Processing Element Example

2.2.5 I/O Processing Element

This type of processing element is shown as the bridge in Figure 2-1. 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.

2.2.6 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 2-6. Behavior of the switches, and the interconnect fabric in general, is addressed in the *RapidIO Common Transport Specification*.

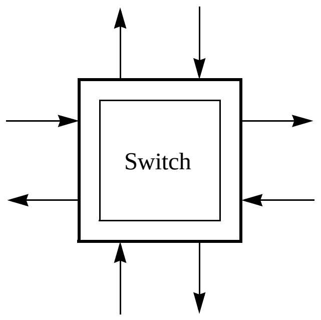


Figure 2-6. Switch Processing Element Example

2.3 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 2-1, 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.

2.3.1 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 4, “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.

For multiple packet messages, 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.

For single packet messages, the letter and mailbox fields instead allow four concurrent letters to sixty-four possible mailboxes. As for multiple packet messages, if the mailbox number is used as a priority indicator, mailbox number 0 is the highest priority and mailbox 63 is the lowest.

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

There is further discussion of the data message operation programming model and the necessary hardware support in Annex A, “Message Passing Interface (Informative)”.

2.3.2 Doorbell Message Operations

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

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

There is further discussion of the doorbell message operation programming model and the necessary hardware support in Annex A, “Message Passing Interface (Informative)”.

2.4 System Issues

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

2.4.1 Operation Ordering

The *RapidIO Part 2: 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.

2.4.2 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.

2.4.3 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.

Chapter 3 Operation Descriptions

3.1 Introduction

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 4, “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. A transaction request which receives a RETRY response must be re-transmitted in order to complete the operation.
- 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.

3.2 Message Passing Operations Cross Reference

Table 3-1 contains a cross-reference list of the message passing operations defined in this RapidIO specification and their system usage.

Table 3-1. Message Passing Operations Cross Reference

Operation	Transactions Used	Possible System Usage	Description	Packet Format
Doorbell	DOORBELL, RESPONSE		Section 3.3.1	Type 10 Section 4.2.4
Data Message	MESSAGE, RESPONSE		Section 3.3.2	Type 11 Section 4.2.5

3.3 Message Passing Operations

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

- Doorbell
- Data Message

3.3.1 Doorbell Operations

The doorbell operation, consisting of the DOORBELL and RESPONSE transactions (typically a DONE response) as shown in Figure 3-1, 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 4.2.4, “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.

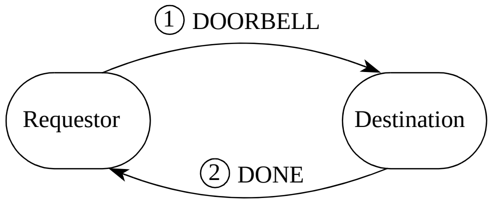


Figure 3-1. Doorbell Operation

3.3.2 Data Message Operations

The data message operation, consisting of the MESSAGE and RESPONSE transactions (typically a DONE response) as shown in Figure 3-2, 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.

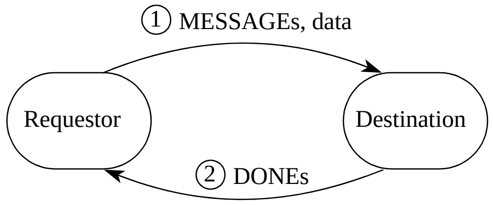


Figure 3-2. Message Operation

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)

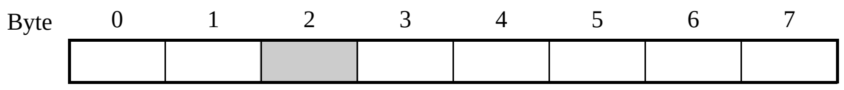
Using this information, the processing element’s message passing hardware can determine that the 32 bytes contained in this part of the data message shall be put into local memory at address 0x3040.

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.

3.4 Endian, Byte Ordering, and Alignment

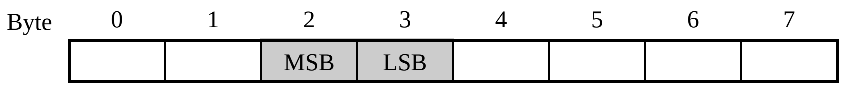
RapidIO has double-word (8-byte) aligned big-endian data payloads. This means that the RapidIO interface to devices that are little-endian shall perform the proper endian transformation at the output to format a data payload.

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 3-3 through Figure 3-5.



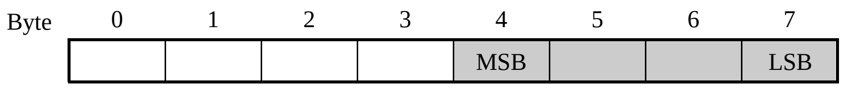
Byte address 0x0000_0002, the proper byte position is shaded.

Figure 3-3. Byte Alignment Example



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

Figure 3-4. Half-Word Alignment Example



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

Figure 3-5. Word Alignment Example

Blank page

Chapter 4 Packet Format Descriptions

4.1 Introduction

This chapter contains the packet format definitions for the *RapidIO Part 2: 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.

4.2 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 Part 2: Message Passing Logical Specification* are shown in Table 4-1.

Table 4-1. 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 4.2.2
Type 1–9	—	Reserved	Section 4.2.3
Type 10	DOORBELL	Send a short message	Section 4.2.4
Type 11	MESSAGE	Send a message	Section 4.2.5

4.2.1 Field Definitions for All Request Packet Formats

The field definitions in Table 4-2 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 4-2. 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

4.2.2 Type 0 Packet Format (Implementation-Defined)

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

4.2.3 Type 1–9 Packet Formats (Reserved)

The type 1–9 formats are reserved.

4.2.4 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 4-1 specifies that the packet format is of type 10.

Definitions and encodings of fields specific to type 10 packets are provided in Table 4-3. Fields that are not specific to type 10 packets are described in Table 4-2.

Table 4-3. Specific Field Definitions for Type 10 Packets

Field	Encoding	Definition
info	—	Software-defined information field

Figure 4-1 displays a type 10 packet with all its fields.

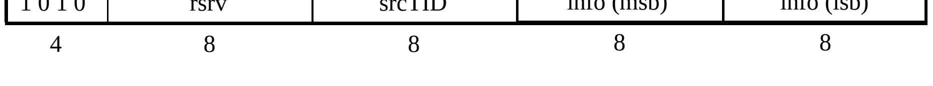


Figure 4-1. Type 10 Packet Bit Stream Format

4.2.5 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 4-4. Fields that are not specific to type 11 packets are described in Table 4-2.

Table 4-4. 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 3.3.2, “Data Message Operations”.
msgseg	—	For multiple packet data message operations, 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 3.3.2, “Data Message Operations”.
xmbox	—	For single packet data message operations, specifies the upper 4 bits of the mailbox targeted by the packet. xmbox || mbox are specified as follows: 0000 00 - mailbox 0 0000 01 - mailbox 1 0000 10 - mailbox 2 0000 11 - mailbox 3 0001 00 - mailbox 4 .... 1111 11 - mailbox 63
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 3.3.2, “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 4-2 displays a type 11 packet with all its fields. The value 0b1011 in Figure 4-2 specifies that the packet format is of type 11.

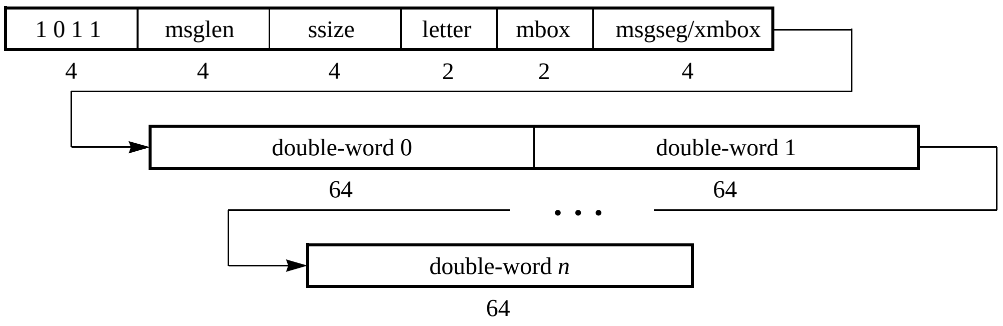


Figure 4-2. Type 11 Packet Bit Stream Format

The combination of the letter, mbox, and the msgseg or xmbox 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. Care must be taken to prevent aliasing of the combination of these values.

4.3 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 4-5.

Table 4-5. Response Packet Type to Transaction Type Cross Reference

Response Packet Format Type	Transaction Type	Definition	Document Section Number
Type 12	—	Reserved	Section 4.3.2
Type 13	RESPONSE	Issued by a processing element when it completes a request by a remote element.	Section 4.3.3
Type 14	—	Reserved	Section 4.3.4
Type 15	Implementation-defined	Defined by the device implementation	Section 4.3.5

4.3.1 Field Definitions for All Response Packet Formats

The field definitions in Table 4-6 apply to more than one of the response packet formats. Fields that are unique to the type 13 format are defined in Section 4.3.3, “Type 13 Packet Format (Response Class).”

Table 4-6. Field Definitions and Encodings for All Response Packets

Field	Encoding	Sub-Field	Definition
transaction	0b0000		RESPONSE transaction with no data payload (including DOORBELL RESPONSE)
	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; re-transmission of the request is needed to complete the transaction
	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

4.3.2 Type 12 Packet Format (Reserved)

The type 12 packet format is reserved.

4.3.3 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 4-7. Fields that are not specific to type 13 packets are described in Table 4-6.

Table 4-7. Specific Field Definitions for Type 13 Packets

Field	Sub-Field	Definition
target_info	As shown in Figure 4-3, 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 4-3).

Figure 4-3 shows the format of the target_info field for message responses.

letter	mbox	msgseg
2	2	4

Figure 4-3. target_info Field for Message Responses

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

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

Figure 4-4. Type 13 Packet Bit Stream Format

4.3.4 Type 14 Packet Format (Reserved)

The type 14 packet format is reserved.

4.3.5 Type 15 Packet Format (Implementation-Defined)

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

Chapter 5 Message Passing Registers

5.1 Introduction

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

5.2 Register Summary

Table 5-1 shows the register map for this RapidIO specification. These capability registers (CARs) and command and status registers (CSRs) can be accessed using *Part 1: Input/Output Logical Specification* maintenance operations. Any register offsets not defined are considered reserved for this specification unless otherwise stated. Other registers required for a processing element are defined in other applicable RapidIO specifications and by the requirements of the specific device and are beyond the scope of this specification. Read and write accesses to reserved register offsets shall terminate normally and not cause an error condition in the target device. Writes to CAR (read-only) space shall terminate normally and not cause an error condition in the target device.

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

Table 5-1. Message Passing Register Map

Configuration Space Byte Offset	Register Name
0x0-14	Reserved
0x18	Source Operations CAR
0x1C	Destination Operations CAR
0x20–FC	Reserved

Table 5-1. Message Passing Register Map (Continued)

Configuration Space Byte Offset	Register Name
0x100–FFFC	Extended Features Space
0x10000–FFFFFC	Implementation-defined Space

5.3 Reserved Register, Bit and Bit Field Value Behavior

Table 5-2 describes the required behavior for accesses to reserved register bits and reserved registers for the RapidIO register space,

Table 5-2. Configuration Space Reserved Access Behavior

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

Table 5-2. 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.

When a writable bit field is set to a reserved value, device behavior is implementation specific.

5.4 Capability Registers (CARs)

Every processing element shall contain a set of registers that allows an external processing element to determine its capabilities using the I/O logical maintenance read operation. All registers are 32 bits wide and are organized and accessed in 32-bit (4 byte) quantities, although some processing elements may optionally allow larger accesses. CARs are read-only. Refer to Table 5-2 for the required behavior for accesses to reserved registers and register bits.

CARs are big-endian with bit 0 the most significant bit.

5.4.1 Source Operations CAR (Configuration Space Offset 0x18)

This register defines the set of RapidIO message passing logical operations that can be issued by this processing element; see Table 5-3. It is assumed that a processing element can generate I/O logical maintenance read and write requests if it is required to access CARs and CSRs in other processing elements. RapidIO switches shall be able to route any packet.

Table 5-3. 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

5.4.2 Destination Operations CAR (Configuration Space Offset 0x1C)

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

Table 5-4. Bit Settings for Destination Operations CAR

Bit	Field Name	Description
0–13	—	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

5.5 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 5-2 for the required behavior for accesses to reserved registers and register bits.

Currently there are no CSRs defined by the message passing logical layer specification.

Annex A Message Passing Interface (Informative)

A.1 Introduction

The *RapidIO Part 2: 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 this specification. This appendix is provided as a reference model for message passing and is not intended to be all encompassing.

A.2 Definitions and Goals

A system may be made up of several processors and distributed memory elements. These processors may be tightly coupled and operating under a monolithic operating system in certain applications. When this is true the operating system is tasked with managing the pool of processors and memory to solve a set of tasks. In most of these cases, it is most efficient for the processors to work out of a common hardware-maintained coherent memory space. This allows processors to communicate initialization and completion of tasks through the use of semaphores, spin locks, and inter-process interrupts. Memory is managed centrally by the operating system with a paging protection scheme.

In other such distributed systems, processors and memory may be more loosely coupled. Several operating systems or kernels may be coexistent in the system, each kernel being responsible for a small part of the entire system. It is necessary to have a communication mechanism whereby kernels can communicate with other kernels in a system of this nature. Since this is a shared nothing environment, it is also desirable to have a common hardware and software interface mechanism to accomplish this communication. This model is typically called message passing.

In these message passing systems, two mechanisms typically are used to move data from one portion of memory space to another. The first mechanism is called direct memory access (DMA), the second is messaging. The primary difference between the two models is that DMA transactions are steered by the source whereas messages are steered by the target. This means that a DMA source not only requires access to a target but must also have visibility into the target's address space. The message source only requires access to the target and does not need visibility into the target's address space. In distributed systems it is common to find a mix of DMA and messaging deployed.

The RapidIO architecture contains a packet transport mechanism that can aid in the distributed shared nothing environment. The RapidIO message passing model meets several goals:

- A message is constructed of one or more transactions that can be sent and received through a possibly unordered interconnect
- A sender can have a number of outstanding messages queued for sending
- A sender can send a higher priority message before a lower priority message and can also preempt a lower priority message to send a higher priority one and have the lower priority message resume when the higher is complete (prioritized concurrency)
- A sender requires no knowledge of the receiver's internal structure or memory map
- A receiver of a message has complete control over it's local address space
- A receiver can have a number of outstanding messages queued for servicing if desired
- A receiver can receive a number of concurrent multiple-transaction messages if desired

A.3 Message Operations

The *RapidIO Part 2: 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 4.2.5, "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 4.2.4, "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.4 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 Part 2: 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.4.1 Simple Inbox

Probably the most simple inbound mailbox structure is that of a single-register port or direct map into local memory space (see Figure A-1).

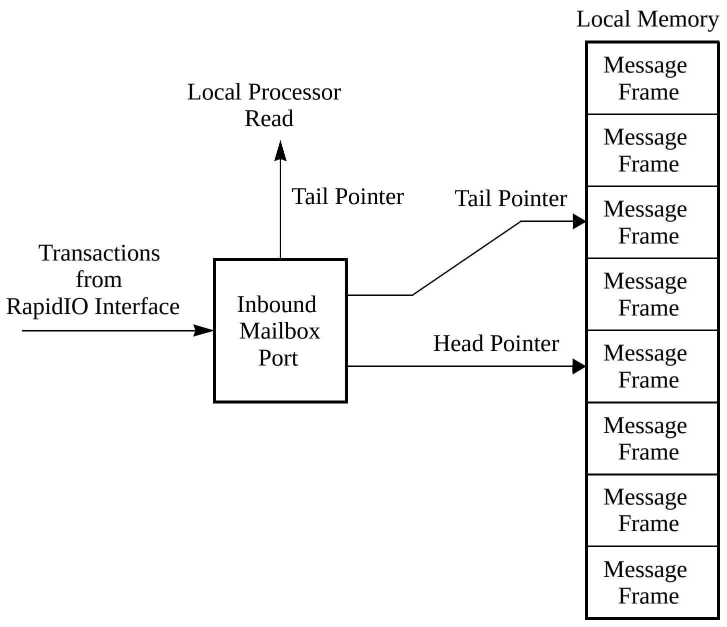


Figure A-1. Simple Inbound Mailbox Port Structure

In this structure, the inbound single transaction message is posted to either a register, set of registers, or circular queue in local memory. In the case of the circular queue, hardware maintains a head and tail pointer that points at a fixed window of pre-partitioned message frames in memory. Whenever the head pointer equals the tail pointer, no more messages can be accepted and they are retried on the RapidIO interface. When messages are posted, the local processor is interrupted. The interrupt service routine reads the mailbox port that contains the message located at the tail pointer. The message frame is equal to the largest message operation that can be received.

The RapidIO MESSAGE transaction allows up to four such inbound mailbox ports per target address. The DOORBELL transaction is defined as a single mailbox port.

A.4.2 Extended Inbox

A second more extensible structure similar to that used in the intelligent I/O (I_2O) specification, but managed differently, also works for the receiver (see Figure A-2).

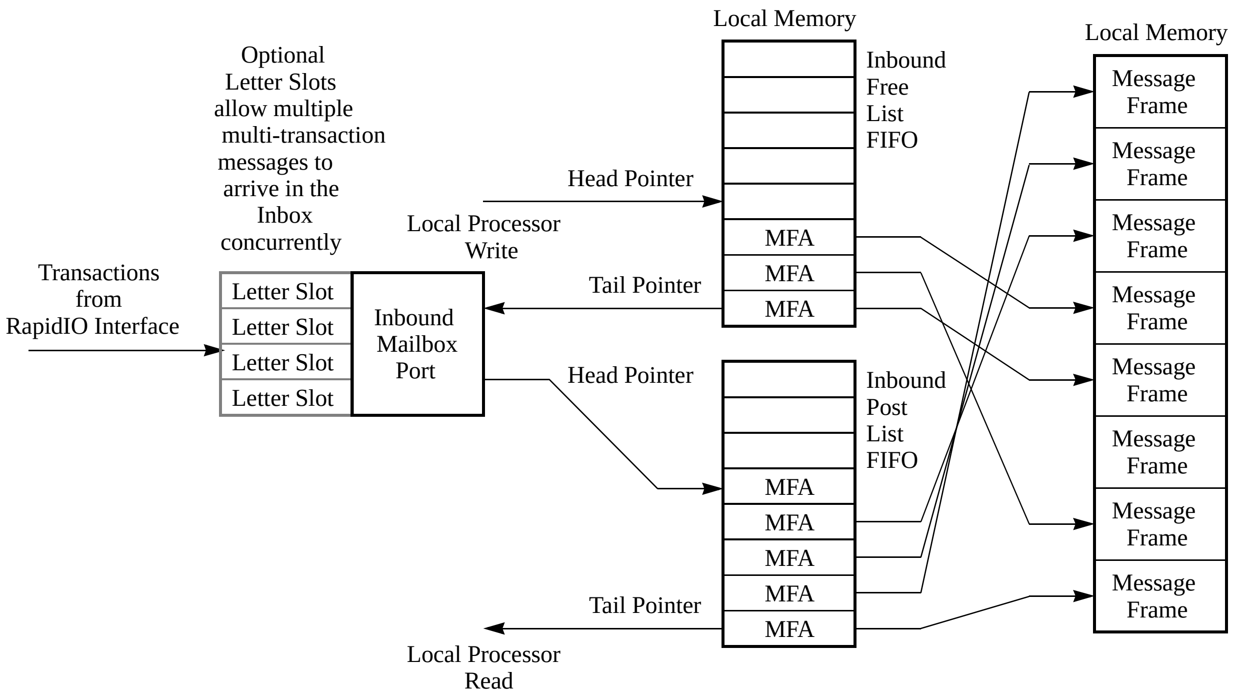


Figure A-2. Inbound Mailbox Structure

One of these structures is required for each priority level supported in an implementation. There are inbound post and free list FIFOs which function as circular queues of a fixed size. The message frames are of a size equal to the maximum message size that can be accepted by the receiver. Smaller messages can be accepted if allowed by the overlaying software. The sender only specifies the mailbox and does not request the frame pointer and perform direct memory access as with I_2O, although the I_2O model can be supported in software with this structure. All pointers are managed by the inbound hardware and the local processor. Message priority and letter number are managed by software.

The advantage of the extended structure is that it allows local software to service message frames in any order. It also allows memory regions to be moved in and out of the message structure instead of forcing software to copy the message to a different memory location.

A.4.3 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 2.3.1, “Data Message Operations” and Section 3.3.2, “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.5 Outbound Message Queue Structure

Queueing messages in RapidIO is accomplished either through a simple or a more extended outbox.

A.5.1 Simple Outbox

Generation of a message can be as simple as writing to a memory-mapped descriptor structure either in local registers or memory. The outbound message queue (see Figure A-3) looks similar to the inbox.

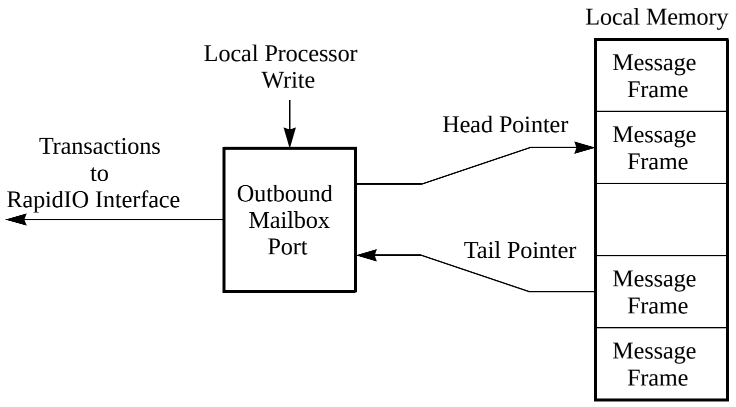


Figure A-3. Outbound Message Queue

The local processor reads a port in the outbound mailbox to obtain the position of a head pointer in local memory. If the read results in a pre-determined pattern the message queue is full. The processor then writes a descriptor structure and message to that location. When it is done, it writes the message port to advance the head point and mark the message as queued. The outbound mailbox hardware then reads the messages pointed to by the tail pointer and transfers them to the target device pointed at by the message descriptor.

One of these structures is required for each priority level of outbound messages supported.

A.5.2 Extended Outbox

A more extensible method of queueing messages is again a two-level approach (see Figure A-4). Multiple structures are required if concurrent operation is desired in an implementation. The FIFO is a circular queue of some fixed size. The message frames are of a size that is equal to the maximum message operation size that can be accepted by the receivers in the system. Smaller message operations can be sent if allowed by the hardware and the overlaying software. As with the receive side, the outbound slots can be virtual and any letter number can be handled by an arbitrary letter slot.

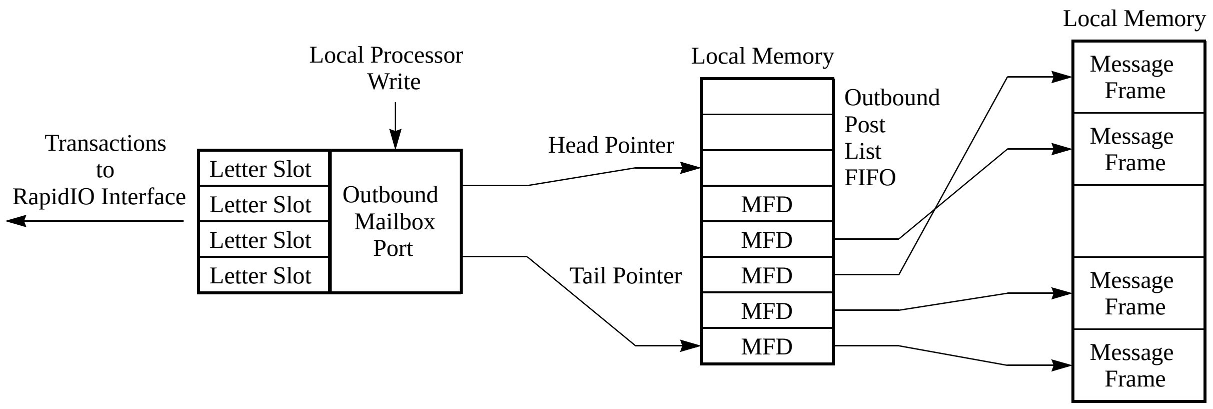


Figure A-4. Extended Outbound Message Queue

When the local processor wishes to send a message, it stores the message in local memory, writes the message frame descriptor (MFD) to the outbound mailbox port (which in-turn writes it to the location indicated by the outbound post FIFO head pointer), and increments the head pointer.

The advantage of this method is that software can have pre-set messages stored in local memory. Whenever it needs to communicate an event to a specific end point it writes the address of the message frame to the outbound mailbox, and the outbound mailbox generates the message transactions and completes the operation.

If the outbound post list FIFO head and tail pointers are not equal, there is a message waiting to be sent. This causes the outbound mailbox port to read the MFD pointed to by the outbound post list tail pointer and then increment the pointer (this may cause a memory read to prefetch the next MFD). The hardware then uses the information stored in the MFD to read the message frame, packetize it, and transmit it to the receiver. Multiple messages can be transmitted concurrently if there is hardware to support them, differentiated by the letter slots in Figure A-4.

If the free list head and tail pointer are the same, the FIFO is empty and there are no more MFDs to be processed. Underflow conditions are fatal because they indicate improper system behavior. This information can also be part of a status register.

Because the outbound and inbound hardware are independent entities, it is possible for more complex outbound mailboxes to communicate with less complex inboxes by simply reducing the complexity of the message descriptor to match. Likewise simple outboxes can communicate with complex inboxes. Software can determine the capabilities of a device during initial system setup. The capabilities of a devices message hardware are stored in the port configuration registers.

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.

B

Big-endian. A byte-ordering method in memory where the address n of a word corresponds to the most significant byte. In an addressed memory word, the bytes are ordered (left to right) 0, 1, 2, 3, with 0 being the most significant byte.

Bridge. A processing element that connects one computer bus to another, allowing a processing element on one bus to access an processing element on the other.

C

Capability registers (CARs). A set of read-only registers that 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.

D

Deadlock. A situation in which two processing elements that are sharing resources prevent each other from accessing the resources, resulting in a halt of system operation.

Destination. The termination point of a packet on the RapidIO interconnect, also referred to as a target.

Device. A generic participant on the RapidIO interconnect that sends or receives RapidIO transactions, also called a processing element.

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.

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

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

E

End point. A processing element which is the source or destination of transactions through a RapidIO fabric.

End point device. A processing element which contains end point functionality.

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

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

F

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

FIFO. First in, first out.

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.

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.

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.

MSB. Most significant byte.

N

Non-coherent. A transaction that does not participate in any system globally shared memory cache coherence mechanism.

O

Operation. A set of transactions between end point devices in a RapidIO system (requests and associated responses) such as a read or a write.

P

Packet. A set of information transmitted between devices in a RapidIO system.

PCB. Printed circuit board.

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

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.

S

Sender. The RapidIO interface output port on a processing element.

Semaphore. A technique for coordinating activities in which multiple processing elements compete for the same resource, typically requiring atomic operations.

Source. The origin of a packet on the RapidIO interconnect, also referred to as an initiator.

Switch. A multiple port processing element that directs a packet received on one of its input ports to one of its output ports.

T

Target. The termination point of a packet on the RapidIO interconnect, also referred to as a destination.

Transaction. A specific request or response packet transmitted between end point devices in a RapidIO system.

W

Word. A four byte or 32 bit quantity, aligned on four byte boundaries.

RapidIO™ Interconnect Specification Part 3: Common Transport Specification

3.2, 1/2016

RapidIO.org

Revision History

Revision	Description	Date
1.1	First public release	03/08/2001
1.2	No technical changes	06/26/2002
1.3	Technical changes: the following new features showings: 03-01-00002.006, 03-03-00002.002 Converted to ISO-friendly templates; re-formatted	02/23/2005
2.0	Technical changes: errata showing 06-02-00001.005	06/14/2007
2.1	No technical changes	07/09/2009
2.1	Removed confidentiality markings for public release	08/13/2009
2.2	Technical changes: errata showings 10-08-00000.003, 10-08-00001.005, Consolidated Comments on 11-01-00000.000	05/05/2011
3.0	Changed RTA contact information. Technical changes: Support for 32-bit device IDs in packet formats. Added register block for Dev32 standard routing table support. Added Annex A Dev32 Programming Model Examples (Informative) Changed RTA contact information. Added Implementation Specific field to “Standard Route Default Port CSR”	10/11/2013
3.1	Minor typographical errors corrected. Technical changes: Added implementation specific bits to “Standard Route Cfg Port Select CSR”.	09/18/2014
3.2	Added resolution of Errata 13 Dev32 Routing Table Examples Correction.	01/28/2016

Table of Contents

Chapter 1 Overview

1.1 Introduction........ 11
1.2 Overview........ 11
1.3 Transport Layer Features........ 11
1.3.1 Functional Features........ 11
1.3.2 Physical Features........ 12
1.3.3 Performance Features........ 12
1.4 Contents........ 12
1.5 Terminology........ 12
1.6 Conventions........ 12

Chapter 2 Transport Format Description

2.1 Introduction........ 15
2.2 System Topology........ 15
2.2.1 Switch-Based Systems........ 15
2.2.2 Ring-Based Systems........ 16
2.3 System Packet Routing........ 17
2.4 Field Alignment and Definition........ 18
2.5 Routing Maintenance Packets........ 19

Chapter 3 Common Transport Registers

3.1 Introduction........ 21
3.2 Register Summary........ 21
3.3 Reserved Register, Bit and Bit Field Value Behavior........ 22
3.4 Capability Registers (CARs)........ 24
3.4.1 Processing Element Features CAR........ 24
3.4.2 Switch Route Table Destination ID Limit CAR........ 25
3.5 Command and Status Registers (CSRs)........ 26
3.5.1 Base Device ID CSR........ 26
3.5.2 Dev32 Base Device ID CSR........ 27
3.5.3 Host Base Device ID Lock CSR........ 28
3.5.4 Component Tag CSR........ 29
3.5.5 Standard Route Cfg Destination ID Select CSR........ 30
3.5.6 Standard Route Cfg Port Select CSR........ 31
3.5.7 Standard Route Default Port CSR........ 33
3.6 Switch Routing Table Register Block........ 34
3.6.1 Register Map........ 34
3.6.2 Switch Routing Table Register Block Header........ 36
3.6.3 Broadcast Routing Table Control CSR........ 37

Table of Contents

3.6.4 Broadcast Level 0 Info CSR 38
3.6.5 Broadcast Level 1 Info CSR 39
3.6.6 Broadcast Level 2 Info CSR 40
3.6.7 Port n Routing Table Control CSRs 41
3.6.8 Port n Level 0 Info CSRs 42
3.6.9 Port n Level 1 Info CSRs 43
3.6.10 Port n Level 2 Info CSRs 44
3.7 Routing Table Group Register Format 45
3.7.1 Broadcast Level 0 Group x Entry y Routing Table Entry CSR 46
3.7.2 Broadcast Level 1 Group x Entry y Routing Table Entry CSR 47
3.7.3 Broadcast Level 2 Group x Entry y Routing Table Entry CSR 48
3.7.4 Level 0 Group x Entry y Routing Table Entry CSR 49
3.7.5 Level 1 Group x Entry y Routing Table Entry CSR 50
3.7.6 Level 2 Group x Entry y Routing Table Entry CSR 51

Annex A Dev32 Hierarchical Programming Model (Informative)

A.1 Dev32 Configuration Examples 53
A.1.1 Example 1: Routing 0x00_11_20_** to Port 14 53
A.1.2 Example 2: Routing 0x00_11_0X_** to Port X 54
A.1.3 Example 3: Routing 0x00_ZZ_**_** to Port 15, ZZ=[0,0x10] 55
A.1.4 Example 4: All Other Packets Must Be Dropped 56
A.1.5 Example 5: Flat Routing Table Operation 56

List of Figures

2-1 A Small Switch-Based System 16
2-2 A Small Ring-Based System..17
2-3 Destination-Source Transport Bit Stream..18
2-4 Maintenance Packet Transport Bit Stream 19

List of Figures

Blank page

List of Tables

2-1 tt Field Definition....18
3-1 Common Transport Register Map 21
3-2 Configuration Space Reserved Access Behavior....22
3-3 Bit Settings for Processing Element Features CAR....24
3-4 Bit Settings for Switch Route Table Destination ID Limit CAR 25
3-5 Bit Settings for Base Device ID CSR 26
3-6 Bit Settings for Base Device ID CSR 27
3-7 Bit Settings for Host Base Device ID Lock CSR 28
3-8 Bit Settings for Component ID CSR....29
3-9 Bit Settings for Standard Route Configuration Destination ID Select CSR....30
3-10 Bit Settings for Standard Route Configuration Port Select CSR....32
3-11 Bit Settings for Standard Route Default Port CSR....33
3-12 Switch Routing Table Register Map....34
3-13 Bit Settings for Switch Routing Table Register Block Header 36
3-14 Bit Settings for Port n Routing Table Control CSR....37
3-15 Bit Settings for Broadcast Level 0 Info CSR....38
3-16 Bit Settings for Broadcast Level 1 Info CSR....39
3-17 Bit Settings for Broadcast Level 2 Info CSR....40
3-18 Bit Settings for Port n Routing Table Control CSRs....41
3-19 Bit Settings for Port n Level 0 Info CSRs 42
3-20 Bit Settings for Port n Level 1 Info CSRs 43
3-21 Bit Settings for Port n Level 2 Info CSRs 44
3-22 Bit Settings for Broadcast Level 0 Group x Entry y Routing Table Entry CSR 46
3-23 Level 1 Group x Entry y Routing Table Entry CSR....47
3-24 Bit Settings for Broadcast Level 2 Group x Entry y Routing Table Entry CSR 48
3-25 Bit Settings for Level 0 Group x Entry y Routing Table Entry CSR 49
3-26 Bit Settings for Level 1 Group x Entry y Routing Table Entry CSR 50
3-27 Bit Settings for Level 2 Group x Entry y Routing Table Entry CSR 51
A-1 Example Port 7 Routing Table Register Block Registers....53
A-2 Example 1 Accesses....54
A-3 Example 2 Accesses....54
A-4 Example 3 Accesses....55
A-5 Example 5 Port 7 Routing Table Register Block Registers....56
A-6 Example 5 Accesses....57

List of Tables

Blank page

Chapter 1 Overview

1.1 Introduction

This chapter provides an overview of the *RapidIO Part 3: Common Transport Specification*, including a description of the relationship between this specification and the other specifications of the RapidIO interconnect.

1.2 Overview

The *RapidIO Part 3: 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.

1.3 Transport Layer Features

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

1.3.1 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.

1.3.2 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.

1.3.3 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.
- Certain devices have bandwidth and latency requirements for proper operation. RapidIO does not preclude an implementation from imposing these constraints within the system.

1.4 Contents

RapidIO Part 3: Common Transport Specification contains three chapters:

- Chapter 1, “Overview” (this chapter) provides an overview of the specification
- Chapter 2, “Transport Format Description,” describes the routing methods used in RapidIO for sending packets across the systems of switches described in this chapter.
- Chapter 3, “Common Transport Registers,” describes the visible register set that allows an external processing element to determine the capabilities, configuration, and status of a processing element using this RapidIO transport layer definition.

1.5 Terminology

Refer to the Glossary at the back of this document.

1.6 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.
x	This value is a don’t care

Blank page

Chapter 2 Transport Format Description

2.1 Introduction

This chapter contains the transport format definition for the *RapidIO Part 3: 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.

2.2 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.

2.2.1 Switch-Based Systems

A RapidIO system can be organized around the concept of switches. Figure 2-1 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.

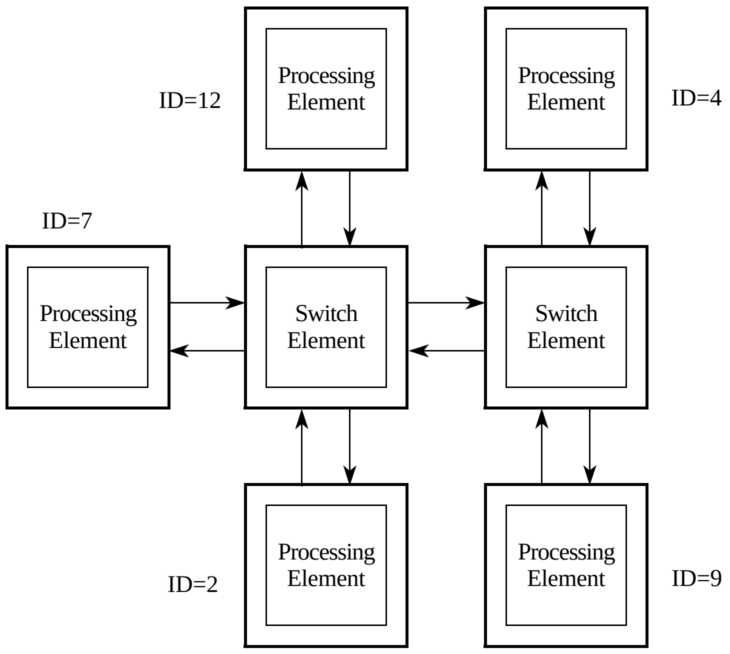


Figure 2-1. A Small Switch-Based System

2.2.2 Ring-Based Systems

A simplification of the switch structure is a ring as shown in Figure 2-2. 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.

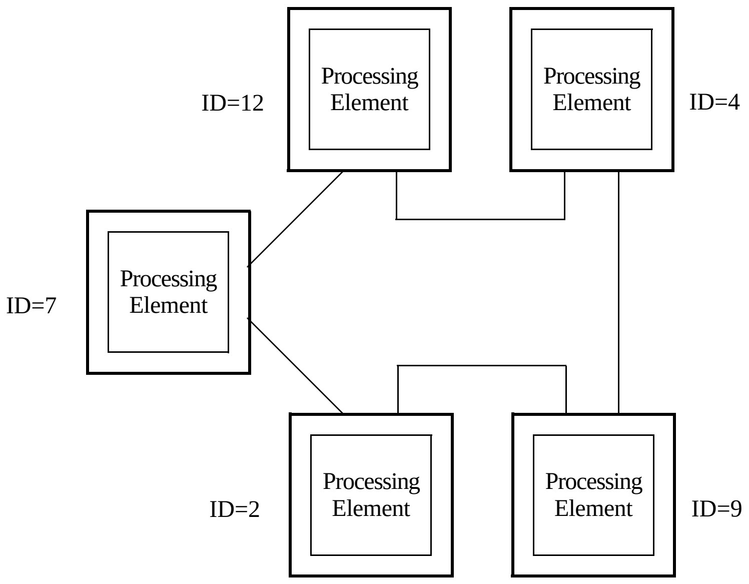


Figure 2-2. A Small Ring-Based System

2.3 System Packet Routing

There are many algorithms that can be used for routing through a system. The *RapidIO Part 3: 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. All devices must provide a mechanism such that the receiver of a request shall process any received supported request regardless of device ID values. 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. Many applications and system designers may require that devices also provide some means to restrict processing requests to a list of acceptable device IDs. For those applications, it is recommended, however, that a device process all maintenance read requests. The behavior when a request is received with a device ID that is not on the acceptable list is implementation-dependant. 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.

2.4 Field Alignment and Definition

The *RapidIO Part 3: 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 three different sizes of the device ID fields: Dev32 (32-bit), Dev16 (16-bit), and a Dev8 (8-bit), as shown in Table 2-1. The three sizes of device ID fields allow three different system scalability points to optimize packet header overhead, and only affix additional transport field overhead if the additional addressing is required. The Dev32 fields enable large system scalability while allowing packets to be routed to specific functions within a device based on destination ID. The Dev8 fields allow a maximum of 256 devices to be attached to the fabric. The Dev16 fields allow systems with up to 65,536 devices. The Dev32 fields allow systems with up to 4,294,967,296 devices.

Table 2-1. tt Field Definition

tt	Definition
0b00	Dev8 8-bit deviceID fields
0b01	Dev16 16-bit deviceID fields
0b10	Dev32 32-bit deviceID fields
0b11	Reserved

Figure 2-3 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, destinationID and sourceID fields) are the transport fields added to the logical packet by the common transport specification.

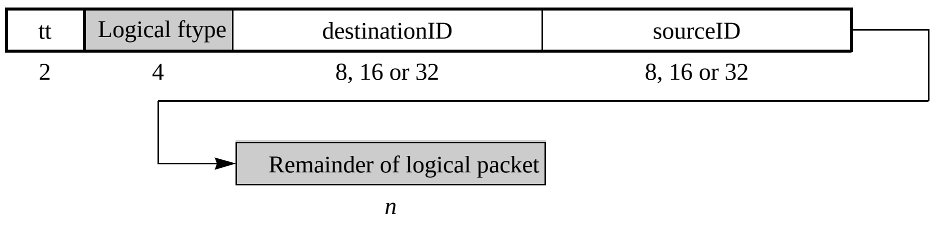


Figure 2-3. Destination-Source Transport Bit Stream

2.5 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. A switch processing element shall decrement the hop count when it forwards a maintenance packet. 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 2-4 shows the transport layer fields added to a maintenance logical packet. Maintenance logical packets can be found in the *RapidIO Part 1: Input/Output Logical Specification*.

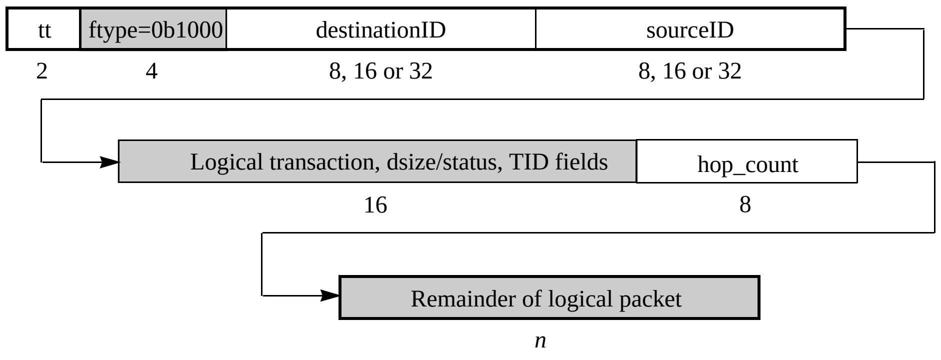

tt
ftype=0b1000
destinationID
sourceID
2
4
8, 16 or 32
8, 16 or 32
Logical transaction, dsize/status, TID fields
hop_count
16
8
Remainder of logical packet
n

Figure 2-4. Maintenance Packet Transport Bit Stream

Blank page

Chapter 3 Common Transport Registers

3.1 Introduction

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.

3.2 Register Summary

Table 3-1 shows the register address map for this RapidIO specification. These capability registers (CARs) and command and status registers (CSRs) can be accessed using *RapidIO Part 1: 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 3-1. Common Transport Register Map

Configuration Space Byte Offset	Register Name
0x0-C	Reserved
0x10	Processing Element Features CAR
0x14–30	Reserved
0x34	Switch Route Table Destination ID Limit CAR
0x38-5C	Reserved

Table 3-1. Common Transport Register Map (Continued)

Configuration Space Byte Offset	Register Name
0x60	Base Device ID CSR
0x64	Dev32 Base Device ID CSR
0x68	Host Base Device ID Lock CSR
0x6C	Component Tag CSR
0x70	Standard Route Configuration Destination ID Select CSR
0x74	Standard Route Configuration Port Select CSR
0x78	Standard Route Default Port CSR
0x7C–FC	Reserved
0x100–FFFC	Extended Features Space
0x10000–FFFFFC	Implementation-defined Space

3.3 Reserved Register, Bit and Bit Field Value Behavior

Table 3-2 describes the required behavior for accesses to reserved register bits and reserved registers for the RapidIO register space,

Table 3-2. 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

Table 3-2. Configuration Space Reserved Access Behavior (Continued)

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

[1]Do not depend on reserved bits being a particular value; use appropriate masks to extract defined bits from the read value.

[2]All register writes shall be in the form: read the register to obtain the values of all reserved bits, merge in the desired values for defined bits to be modified, and write the register, thus preserving the value of all reserved bits.

When a writable bit field is set to a reserved value, device behavior is implementation specific.

3.4 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 3-2 for the required behavior for accesses to reserved registers and register bits.

CARs are big-endian with bit 0 the most significant bit.

3.4.1 Processing Element Features CAR (Configuration Space Offset 0x10)

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

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

Bits	Name	Description
0–18	—	Reserved
19	Dev32 Support	0b0 - PE does not support Common Transport Dev32 0b1 - PE supports Common Transport Dev32
20-21	—	Reserved
22	Extended route table configuration support	0b0 - Switch PE does not support the extended route table configuration mechanism 0b1 - Switch PE supports the extended route table configuration mechanism (can only be set if bit 23 is set and bit 19 is clear)
23	Standard route table configuration support	0b0 - Switch PE does not support the standard route table configuration mechanism 0b1 - Switch PE supports the standard route table configuration mechanism
24–26	—	Reserved
27	Dev16 support	0b0 - PE does not support Dev16 0b1 - PE supports Dev16
28–31	—	Reserved

3.4.2 Switch Route Table Destination ID Limit CAR (Configuration Space Offset 0x34)

The Switch Route Table Destination ID Limit CAR specifies the maximum destination ID value that can be programmed with the standard route table configuration mechanism, and thereby indirectly defining the size of the route table. A route table access or extended route table access attempt to destination IDs greater than that specified in this register will have undefined results. This register shall be implemented if bit 23 of the Processing Element Features CAR is set. The bit settings are shown in Table 3-4.

Table 3-4. Bit Settings for Switch Route Table Destination ID Limit CAR

Bits	Name	Description
0–15	—	Reserved
16–31	Max_destID	Maximum configurable destination ID 0x0000 - 1 destination ID 0x0001 - 2 destination IDs 0x0002 - 3 destination IDs ... 0xFFFF - 65536 destination IDs

3.5 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 3-2 for the required behavior for accesses to reserved registers and register bits.

3.5.1 Base Device ID CSR (Configuration Space Offset 0x60)

The base device ID CSR contains the Dev8 and Dev16 base device ID values for the processing element. A device can have multiple device ID values but these are not defined in a standard CSR. The bit settings are shown in Table 3-5.

Table 3-5. Bit Settings for Base Device ID CSR

Bits	Name	Reset Value	Description
0-7	—		Reserved
8-15	Dev8_Base_deviceID	see footnote[1]	This is the Dev8 device ID of the device (endpoint devices only)
16–31	Dev16_base_deviceID	see footnote[2]	This is the Dev16 device ID of the device (must be valid for endpoint devices when bit 27 of the Processing Element Features CAR is set)

[1]The Dev8_Base_deviceID reset value is implementation dependent

[2]The Dev16_base_deviceID reset value is implementation dependent

3.5.2 Dev32 Base Device ID CSR (Configuration Space Offset 0x64)

The Dev32 base device ID CSR contains the Dev32 base device ID value for the processing element. A device can have multiple device ID values but these are not defined in a standard CSR. The bit settings are shown in Table 3-6.

Table 3-6. Bit Settings for Base Device ID CSR

Bits	Name	Reset Value	Description
0-31	Dev32_Base_DeviceID	see footnote[1]	This is the Dev32 device ID of the device (must be valid for endpoint devices when bit 19 of the Processing Element Features CAR is set)

[1]The Dev32_Base_DeviceID reset value is implementation dependent

3.5.3 Host Base Device ID Lock CSR (Configuration Space Offset 0x68)

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 Device ID Lock CSR is a write-once/reset-able register which provides a lock function. Once the Host Base Device ID Lock CSR is written, all subsequent writes to the register are ignored, except in the case that the value written matches the value contained in the register. In this case, the register is re-initialized to 0x0000_FFFF. After writing the Host Base Device ID Lock CSR 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 3-7.

Table 3-7. Bit Settings for Host Base Device ID Lock CSR

Bits	Name	Reset Value	Description
0-15	Host_base_Dev32ID	0x0000	If the Processing Element Features CAR Dev32 Support bit is 0, then this field is Reserved and shall have a constant value of 0. If the Processing Element Features CAR Dev32 Support bit is 1, this field contains the most significant 16 bits of the Dev32 base device ID for the PE that is initializing this PE.
16–31	Host_base_deviceID	0xFFFF	This is the base device ID for the PE that is initializing this PE.

3.5.4 Component Tag CSR (Configuration Space Offset 0x6C)

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 3-8.

Table 3-8. 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.

3.5.5 Standard Route Cfg Destination ID Select CSR (Configuration Space Offset 0x70)

The Standard Route Configuration Destination ID Select CSR specifies the destination ID entry in the switch routing table to access when the Standard Route Configuration Port Select CSR is read or written.

The Ext_config_en bit controls whether the extended route table configuration mechanism is enabled. If the extended route table configuration mechanism is enabled, the specified destination ID and the next three sequential destination IDs are written or read when the Standard Route Configuration Port Select CSR is accessed. Extended accesses that increment past the maximum specifiable destination ID (for example, starting an extended access at device ID 0xFF in a Dev8 transport system) have undefined results.

This register is required if bit 23 of the Processing Element Features CAR is set. The bit settings are shown in Table 3-9.

Table 3-9. Bit Settings for Standard Route Configuration Destination ID Select CSR

Bits	Name	Reset Value	Description
0	Ext_config_en	0b0	Extended Configuration Enable 0b0 - Extended configuration support is disabled 0b1 - Extended configuration support is enabled (only valid if bit 22 of the Processing Element Features CAR is set)
1-15	—		Reserved
16-23	Config_destID_msb	0x00	Configuration destination ID most significant byte (only valid if bit 27 of the Processing Element Features CAR is set and the processing element is configured to operate in Dev16 transport mode)
24-31	Config_destID	0x00	Configuration destination ID

3.5.6 Standard Route Cfg Port Select CSR (Configuration Space Offset 0x74)

When written, the Standard Route Configuration Port Select CSR updates the switch output port configuration for packets with the destination ID selected by the Standard Route Configuration Destination ID Select CSR. When read, the Standard Route Configuration Port Select CSR returns the switch output port configuration for packets with the destination ID selected by the Standard Route Configuration Destination ID Select CSR.

If the extended route table configuration mechanism is enabled, when the Standard Route Configuration Port Select register is written the following route table configurations are carried out:

- destination ID Config_destID is routed to output port Config_output_port
- destination ID Config_destID+1 is routed to output port Config_output_port1
- destination ID Config_destID+2 is routed to output port Config_output_port2
- destination ID Config_destID+3 is routed to output port Config_output_port3

For reads of the Standard Route Configuration Port Select CSR, the configuration information is returned in the corresponding fashion.

After complete system initialization the switch output port route configuration information read may not be consistent with previously read values due to the capabilities and features of the particular switch. This register shall be implemented if bit 23 of the Processing Element Features CAR is set. The bit settings are shown in Table 3-10.

Table 3-10. Bit Settings for Standard Route Configuration Port Select CSR

Bits	Name	Reset Value	Description
0-3	Cop3_msb_or_imp_spec	0x00	Cop3_msb_or_imp_spec: This field shall be reserved if extended route table mechanism is not enabled and bit 19 of the Processing Element Features CAR is clear. When bit 19 of the Processing Element Features CAR is clear, and the extended route table mechanism is enabled, this field contains the most significant 4 bits of the configuration output port3 value. When bit 19 of the Processing Element Features CAR is set, this field optionally controls implementation-specific routing functionality: • Bits in this field that do not control implementation specific routing functionality shall be read only, with a fixed value of 0. • Implementation-specific routing functionality may be active if any bit in this field is set. • Implementation-specific routing functionality shall not be active if all bits in this field are clear.
4-7	Config_output_port3_lsb	0x00	Configuration output port3 - This field shall be reserved if the extended route table mechanism is not enabled. If the extended route table mechanism is enabled, this field contains the least significant 4 bits of the config output port3 value.
8-15	Config_output_port2	0x00	Configuration output port2 - This field shall be reserved if extended route table mechanism is not enabled
16-21	Config_output_port1_msb	0x00	Most significant 6 bits of the output port 1 value if the extended route table mechanism is enabled. This field shall be reserved if extended route table mechanism is not enabled.
22-23	Config_output_port1_lsb	0b00	Least significant 2 bits of the output port 1 value if the extended route table mechanism is enabled. Most significant 2 bits of the route value if bit 19 of the Processing Element Features CAR is set. This field shall be reserved if extended route table mechanism is not enabled and bit 19 of the Processing Element Features CAR is clear.
24-31	Config_output_port	see footnote[1]	Configuration output port. If bit 19 of the Processing Element Features CAR is set, the routing table value read and written is found in the Config_output_port1_lsb and Config_output_port fields.

[1]The Config_output_port reset value is implementation dependent

3.5.7 Standard Route Default Port CSR (Configuration Space Offset 0x78)

The Standard Route Default Port CSR specifies the port to which packets with destinations IDs that are greater than that specified in the Switch Route Table Destination ID Limit CAR are routed. This register is required if bit 23 of the Processing Element Features CAR is set. The bit settings are shown in Table 3-11.

Table 3-11. Bit Settings for Standard Route Default Port CSR

Bits	Name	Reset Value	Description
0–3	Implementation Specific	0x0	These bits optionally control implementation-specific routing functionality. This field is allowed when bit 19 of the Processing Element Features CAR is set. If bit 19 of the Processing Element Features CAR is clear, this field is reserved. Bits in this field that do not control implementation specific routing functionality shall be read only, with a fixed value of 0. Implementation-specific routing functionality may be active if any bit in this field is set. Implementation-specific routing functionality shall not be active if all bits in this field are clear.
4–21	—		Reserved
22-23	Route Type	0b11	Extended value for packet routing. This field is required when bit 19 of the Processing Element Features CAR is set. If bit 19 of the Processing Element Features CAR is clear, this field is reserved.
24–31	Default_output_port	0x00	Default output port When bit 19 of the Processing Element Features CAR is set, Route Type concatenated with default_output_port shall be encoded as follows: 0x000 to 0x0FF - Egress Port Number 0x00 to 0xFF 0x100 to 0x1FF - Multicast Mask Number 0x00 to 0xFF 0x200 to 0x2FF - Reserved. 0x300 - Drop Packet 0x301 to 0x3FF - Reserved. Selection of an Egress Port Number which is not supported by the device, or a Multicast Mask Number which is not supported by the device, shall result in implementation specific routing behavior.

3.6 Switch Routing Table Register Block

A switch device which has bit 19 set in the Processing Element Features CAR shall implement this register block.

3.6.1 Register Map

The register map for the routing table registers shall be as specified by Table 3-12. 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+0xBC]. Register map offset [EF_PTR+0x140] can be used for another Extended Features block.

Table 3-12. Switch Routing Table Register Map

	Block Byte Offset	**Register Name**
General	0x0	Routing Table Register Block Header
	0x4-0x1C	Reserved
Broadcast	0x20	Broadcast Routing Table Control CSR
	0x24-0x2C	Reserved
	0x30	Broadcast Level 0 Info CSR
	0x34	Broadcast Level 1 Info CSR
	0x38	Broadcast Level 2 Info CSR
	0x3C	Reserved
Port 0	0x40	Port 0 Routing Table Control CSR
	0x44-0x4C	Reserved
	0x50	Port 0 Level 0 Info CSR
	0x54	Port 0 Level 1 Info CSR
	0x58	Port 0 Level 2 Info CSR
	0x5C	Reserved
Port 1	0x60	Port 1 Routing Table Control CSR
	0x64-0x6C	Reserved
	0x70	Port 1 Level 0 Info CSR
	0x74	Port 1 Level 1 Info CSR
	0x78	Port 1 Level 2 Info CSR
	0x7C	Reserved

Table 3-12. Switch Routing Table Register Map

	Block Byte Offset	Register Name
Ports 2-14	0x80–21C	Assigned to Port 2-14 CSRs
Port 15	0x220	Port 15 Routing Table Control CSR
	0x224-0x22C	Reserved
	0x230	Port 15 Level 0 Info CSR
	0x234	Port 15 Level 1 Info CSR
	0x238	Port 15 Level 2 Info CSR
	0x23C	Reserved

3.6.2 Switch Routing Table Register Block Header (Block Offset 0x0)

The switch routing table register block header register contains the EF_PTR to the next EF_BLK and the EF_ID that identifies this as the switch routing table registers block header. The use and meaning of the bits and bit fields of this register shall be as specified in Table 3-13. The register shall be read-only.

Table 3-13. Bit Settings for Switch Routing Table Register Block Header

Bit	Name	Reset Value	Description
0-15	EF_PTR	see footnote[1]	Hard wired pointer to the next block in the data structure, if one exists
16-31	EF_ID	0x000E	Hard wired Extended Features ID

[1]The EF_PTR reset value is implementation dependent

3.6.3 Broadcast Routing Table Control CSR (Block Offset 0x20)

Writes to this register are broadcast to all Port n Routing Table Control CSRs. The use and meaning of the bits and bit fields of this register shall be as specified in Table 3-14. Unless otherwise specified, the bits and bit fields in this register are write only.

Table 3-14. Bit Settings for Port n Routing Table Control CSR

Bit	Name	Reset Value	Description
0	Three Levels	0b1	0 - Routing table entries support a contiguous range of device IDs, starting with device ID 0x00/0x0000/0x00000000. 1 - Routing table entries support a hierarchical routing scheme
1	Dev32 Route Control	0b0	0 - Dev32 Device IDs are routed using Byte 0 for Level 0, Byte 1 for Level 1, and Byte 2 for Level 2 1 - Dev32 Device IDs are routed using Byte 1 for Level 0, Byte 2 for Level 1, and Byte 3 for level 2 Reserved if Three Levels is clear.
2-31	___		Reserved

3.6.4 Broadcast Level 0 Info CSR (Block Offset 0x30)

This register shall communicate the location of the Broadcast Level 0 routing table group. Writes to the Broadcast Level 0 routing table group affect all Port n Level 0 routing groups. The use and meaning of the bits and bit fields of this register shall be as specified in Table 3-15. This register shall be read only.

Table 3-15. Bit Settings for Broadcast Level 0 Info CSR

Bit	Name	Reset Value	Description
0-7	Num_L0_Groups	see footnote[1]	Communicates the number of 256 entry routing table groups for Level 0. When Three Levels is 0, Num_L0_Groups shall communicate the number of groups available for routing. When Three Levels is 1, Num_L0_Groups shall be 1. Num_L0_Groups is encoded as follows: 0x00 - 256 Groups 0x01 - 1 Group 0x02 - 2 Groups 0x03 - 3 Groups ... 0xFF - 255 Groups
8-21	L0_Group_Ptr	see footnote[2]	The L0_Group_Ptr value shall be the maintenance offset of the first entry in the first routing table group for level 0, divided by 1024. The maintenance offset of the first entry in the first routing group for level 0 shall be a 1024 byte aligned address. The L0_Group_Ptr value shall indicate an address in Implementation Defined register space. Writes to the broadcast routing table group entries pointed to by this register shall cause the corresponding routing table group entries for all ports to assume the value written. Implementation specific behavior shall occur for writes to routing table group entries if the contents of the Port n Routing Table Control CSRs are not the same for all ports .
22-31	___	All 0’s	Reserved

[1]The Num_L0_Groups reset value is implementation dependent

[2]The L0_Group_Ptr reset value is implementation dependent

3.6.5 Broadcast Level 1 Info CSR (Block Offset 0x34)

This register shall communicate the location of the Broadcast Level 1 routing table group. When Three Levels is 0, this register is reserved. The use and meaning of the bits and bit fields of this register shall be as specified in Table 3-16. This register shall be read only.

Table 3-16. Bit Settings for Broadcast Level 1 Info CSR

Bit	Name	Reset Value	Description
0-7	Num_L1_Groups	see footnote[1]	Communicates the number of 256 entry routing table groups for Level 1. Num_L1_Groups shall be encoded as follows: 0x00 - 256 Groups 0x01 - 1 Group 0x02 - 2 Groups 0x03 - 3 Groups ... 0xFF - 255 Groups
8-21	L1_Group_Ptr	see footnote[2]	The L1_Group_Ptr value shall be the maintenance offset of the first entry in the first routing table group for level 1, divided by 1024. The maintenance offset of the first entry in the first routing group for level 1 shall be a 1024 byte aligned address. The L1_Group_Ptr value shall indicate an address in Implementation Defined register space. Writes to the broadcast routing table group entries pointed to by this register shall cause the corresponding routing table group entries for all ports to assume the value written. Implementation specific behavior shall occur for writes to routing table group entries if the contents of the Port n Routing Table Control CSRs are not the same for all ports .
22-31	___	All 0’s	Reserved

[1]The Num_L1_Groups reset value is implementation dependent

[2]The L1_Group_Ptr reset value is implementation dependent

3.6.6 Broadcast Level 2 Info CSR (Block Offset 0x38)

This register shall communicate the location of the Level 2 routing table group for Port n. When Three Levels is 0, this register is reserved. The use and meaning of the bits and bit fields of this register shall be as specified in Table 3-17. This register shall be read only.

Table 3-17. Bit Settings for Broadcast Level 2 Info CSR

Bit	Name	Reset Value	Description
0-7	Num_L2_Groups	see footnote[1]	Communicates the number of 256 entry routing table groups for Level 2. Num_L2_Groups shall be encoded as follows: 0x00 - 256 Groups 0x01 - 1 Group 0x02 - 2 Groups 0x03 - 3 Groups ... 0xFF - 255 Groups
8-21	L2_Group_Ptr	see footnote[2]	The L2_Group_Ptr value shall be the maintenance offset of the first entry in the first broadcast routing table group for level 2, divided by 1024. The maintenance offset of the first entry in the first broadcast routing group for level 2 shall be a 1024 byte aligned address. The L2_Group_Ptr value shall indicate an address in Implementation Defined register space. Writes to the broadcast routing table group entries pointed to by this register shall cause the corresponding routing table group entries for all ports to assume the value written. Implementation specific behavior shall occur for writes to broadcast routing table group entries if the contents of the Port n Routing Table Control CSRs are not the same for all ports .
22-31	___	All 0’s	Reserved

[1]The Num_L2_Groups reset value is implementation dependent

[2]The L2_Group_Ptr reset value is implementation dependent

3.6.7 Port *n* Routing Table Control CSRs (Block Offset 0x40 + (0x20 * n))

These registers shall control the routing mode for all ports whose Port n Level 0 Info CSR L0_Group_Ptr field value is the same. The use and meaning of the bits and bit fields of these registers shall be as specified in Table 3-18. Unless otherwise specified, the bits and bit fields in these registers are read/write.

Table 3-18. Bit Settings for Port *n* Routing Table Control CSRs

Bit	Name	Reset Value	Description
0	Three Levels	0b1	0 - Routing table entries support a contiguous range of device IDs, starting with device ID 0x00/0x0000/0x00000000. 1 - Routing table entries support a hierarchical routing scheme
1	Dev32 Route Control	0b0	0 - Dev32 Device IDs are routed using Byte 0 for Level 0, Byte 1 for Level 1, and Byte 2 for Level 2 1 - Dev32 Device IDs are routed using Byte 1 for Level 0, Byte 2 for Level 1, and Byte 3 for level 2 Reserved if Three Levels is clear.
2-31	___		Reserved

3.6.8 Port *n* Level 0 Info CSRs (Block Offset 0x50 + (0x20 * n))

These registers shall communicate the location of the Level 0 routing table group for Port n. The use and meaning of the bits and bit fields of these registers shall be as specified in Table 3-19. These registers shall be read only.

Table 3-19. Bit Settings for Port *n* Level 0 Info CSRs

Bit	Name	Reset Value	Description
0-7	Num_L0_Groups	see footnote[1]	Communicates the number of 256 entry routing table groups for Level 0. When Three Levels is 0, Num_L0_Groups shall communicate the number of groups available for routing. When Three Levels is 1, Num_L0_Groups shall be 1. Num_L0_Groups is encoded as follows: 0x00 - 256 Groups 0x01 - 1 Group 0x02 - 2 Groups 0x03 - 3 Groups ... 0xFF - 255 Groups
8-21	L0_Group_Ptr	see footnote[2]	The L0_Group_Ptr value shall be the maintenance offset of the first entry in the first routing table group for level 0, divided by 1024. The maintenance offset of the first entry in the first routing group for level 0 shall be a 1024 byte aligned address. The L0_Group_Ptr value shall indicate an address in Implementation Defined register space. All ports with identical L0_Group_Ptr values shall have identical Level 0 routing behavior.
22-31	___	All 0's	Reserved

[1]The Num_L0_Groups reset value is implementation dependent

[2]The L0_Group_Ptr reset value is implementation dependent

3.6.9 Port *n* Level 1 Info CSRs (Block Offset 0x54 + (0x20 * n))

These registers shall communicate the location of the Level 1 routing table group for Port n. When the Three Levels of the Port n Routing Table Control CSRs is 0, these registers shall be reserved. The use and meaning of the bits and bit fields of these registers shall be as specified in Table 3-20. These registers shall be read only.

Table 3-20. Bit Settings for Port *n* Level 1 Info CSRs

Bit	Name	Reset Value	Description
0-7	Num_L1_Groups	see footnote[1]	Communicates the number of 256 entry routing table groups for Level 1. Num_L1_Groups shall be encoded as follows: 0x00 - 256 Groups 0x01 - 1 Group 0x02 - 2 Groups 0x03 - 3 Groups ... 0xFF - 255 Groups
8-21	L1_Group_Ptr	see footnote[2]	The L0_Group_Ptr value shall be the maintenance offset of the first entry in the first routing table group for level 0, divided by 1024. The maintenance offset of the first entry in the first routing group for level 0 shall be a 1024 byte aligned address. The L1_Group_Ptr value shall indicate an address in Implementation Defined register space. All ports with identical L1_Group_Ptr values shall have identical Level 1 packet routing behavior.
22-31	___	All 0’s	Reserved

[1]The Num_L1_Groups reset value is implementation dependent

[2]The L1_Group_Ptr reset value is implementation dependent

3.6.10 Port *n* Level 2 Info CSRs (Block Offset 0x58 + (0x20 * n))

These registers shall communicate the location of the Level 2 routing table group for Port n. When the Three Levels of the Port n Routing Table Control CSRs is 0, these registers shall be reserved. The use and meaning of the bits and bit fields of these registers shall be as specified in Table 3-21. These registers shall be read only.

Table 3-21. Bit Settings for Port *n* Level 2 Info CSRs

Bit	Name	Reset Value	Description
0-7	Num_L2_Groups	see footnote[1]	Communicates the number of 256 entry routing table groups for Level 2. Num_L2_Groups shall be encoded as follows: 0x00 - 256 Groups 0x01 - 1 Group 0x02 - 2 Groups 0x03 - 3 Groups ... 0xFF - 255 Groups
8-21	L2_Group_Ptr	see footnote[2]	The L2_Group_Ptr value shall be the maintenance offset of the first entry in the first routing table group for level 2, divided by 1024. The maintenance offset of the first entry in the first routing group for level 2 shall be a 1024 byte aligned address. The L1_Group_Ptr value shall indicate an address in Implementation Defined register space. All ports with identical L2_Group_Ptr values shall have identical Level 2 routing behavior.
22-31	___	All 0’s	Reserved

[1]The Num_L2_Groups reset value is implementation dependent

[2]The L2_Group_Ptr reset value is implementation dependent

3.7 Routing Table Group Register Format

A group of routing table entries consists of 256 consecutive registers. The L0_Group_Ptr, L1_Group_Ptr, and L2_Group_Ptr point to the first entry of the first group of a number of contiguous groups of register entries.

The address of registers in a routing table group is computed using three values, denoted as Group_Ptr, X, and Y, where:

- Group_Ptr is the value of the "Lz_Group_Ptr" field found in the Port n Level z Info CSRs
- X is the group number
- Y is the entry number within the group

As shown in the following register format definitions, the register address is computed as:

(Group_Ptr * 0x400) + (X * 0x400) + (Y * 4).

For example, assume that the Port n Level z CSR value is 0x03048C00 and it is necessary to address entry number 127 in group number 3. The address computation is:

(0x123 * 0x400) + (3 * 0x400) + (127 * 4) = 0x499FC

3.7.1 Broadcast Level 0 Group x Entry y Routing Table Entry CSR (Offset = (L0_Group_Ptr*0x400) + (x * 0x400) + (y*4))

Writes to the Broadcast Level 0 Group x Entry y Routing Table Entry CSRs shall cause the corresponding Port n Level 0 Group x Entry y Routing Table Entry CSRs for all ports to assume the value written. The use and meaning of the bits and bit fields of this register shall be as specified in Table 3-22. The bits and bit fields in this register are write only.

Table 3-22. Bit Settings for Broadcast Level 0 Group x Entry y Routing Table Entry CSR

Bit	Name	Reset Value	Description
0-3	Implementation-defined	Impl. Spec	These bits control implementation specific behavior. When these bits are written to 0x0, no implementation specific function shall be invoked.
4-21	___		Reserved
22-31	Routing Value	0x300	Routing table entry 0x000 to 0x0FF - Egress Port Number 0x00 to 0xFF 0x100 to 0x1FF - Multicast Mask Number 0x00 to 0xFF. Refer to Part 11. 0x200 to 0x2FF - Level 1 Group number 0x00 to 0xFF 0x300 - Drop Packet 0x301 - Use value found in Standard Port Default Route CSR 0x302 to 0x3FF - Reserved. Selection of an Egress Port Number, Multicast Mask or Level 1 Group Number which does not exist in the device shall result in implementation specific behavior. When the Three Levels field of the Port n Routing Table Control CSR is clear, the values 0x200 through 0x2FF shall result in implementation specific routing behavior.

3.7.2 Broadcast Level 1 Group x Entry y Routing Table Entry CSR (Offset = (L1_Group_Ptr*0x400) + (x * 0x400) + (y*4))

Writes to the Broadcast Level 1 Group x Entry y Routing Table Entry CSRs shall cause the corresponding Port n Level 1 Group x Entry y Routing Table Entry CSRs for all ports to assume the value written. The use and meaning of the bits and bit fields of this register shall be as specified in Table 3-23. The bits and bit fields in this register are write only.

Table 3-23. Level 1 Group x Entry y Routing Table Entry CSR

Bit	Name	Reset Value	Description
0-3	Implementation-defined	Impl. Spec	These bits control implementation specific behavior. When these bits are written to 0x0, no implementation specific function shall be invoked.
4-21	___		Reserved
22-31	Routing Value	0x300	Routing table entry 0x000 to 0x0FF - Egress Port Number 0x00 to 0xFF 0x100 to 0x1FF - Multicast Mask Number 0x00 to 0xFF. Refer to Part 11. 0x200 to 0x2FF - Level 2 Group number 0x00 to 0xFF 0x300 - Drop Packet 0x301 - Use value found in Standard Port Default Route CSR 0x302 to 0x3FF - Reserved. Selection of an Egress Port Number, Multicast Mask or Level 2 Group Number which does not exist in the device shall result in implementation specific behavior.

3.7.3 Broadcast Level 2 Group x Entry y Routing Table Entry CSR (Offset = (L2_Group_Ptr*0x400) + (x * 0x400) + (y*4))

Writes to the Broadcast Level 2 Group x Entry y Routing Table Entry CSRs shall cause the corresponding Port n Level 2 Group x Entry y Routing Table Entry CSRs for all ports to assume the value written. The use and meaning of the bits and bit fields of this register shall be as specified in Table 3-24. The bits and bit fields in this register are write only.

Table 3-24. Bit Settings for Broadcast Level 2 Group x Entry y Routing Table Entry CSR

Bit	Name	Reset Value	Description
0-3	Implementation-defined	Impl. Spec	These bits control implementation specific behavior. When these bits are written to 0x0, no implementation specific function shall be invoked.
4-21	___		Reserved
22-31	Routing Value	0x300	Routing table entry 0x000 to 0x0FF - Egress Port Number 0x00 to 0xFF 0x100 to 0x1FF - Multicast Mask Number 0x00 to 0xFF. Refer to Part 11. 0x200 to 0x2FF - Reserved. 0x300 - Drop Packet 0x301 - Route packet using the Standard Port Default Route CSR 0x302 to 0x3FF - Reserved. Selection of an Egress Port Number or Multicast Mask Number which does not exist in the device shall result in implementation specific behavior.

3.7.4 Level 0 Group x Entry y Routing Table Entry CSR (Offset = (L0_Group_Ptr*0x400) + (x * 0x400) + (y*4))

This register shall control the routing mode for all ports whose Port n Level 0 Info CSR L0_Group_Ptr field value is the same. The use and meaning of the bits and bit fields of this register shall be as specified in Table 3-25. Unless otherwise specified, the bits and bit fields in this register are read/write.

Table 3-25. Bit Settings for Level 0 Group x Entry y Routing Table Entry CSR

Bit	Name	Reset Value	Description
0-3	Implementation-defined	Impl. Spec	These bits control implementation specific behavior. When these bits are written to 0x0, no implementation specific function shall be invoked.
4-21	___		Reserved
22-31	Routing Value	0x300	Routing table entry 0x000 to 0x0FF - Egress Port Number 0x00 to 0xFF 0x100 to 0x1FF - Multicast Mask Number 0x00 to 0xFF. Refer to Part 11. 0x200 to 0x2FF - Level 1 Group number 0x00 to 0xFF 0x300 - Drop Packet 0x301 - Use value found in Standard Port Default Route CSR 0x302 to 0x3FF - Reserved. Selection of an Egress Port Number, Multicast Mask or Level 1 Group Number which does not exist in the device shall result in implementation specific behavior. When the Three Levels field of the Port n Routing Table Control CSR is clear, the values 0x200 through 0x2FF shall result in implementation specific routing behavior.

3.7.5 Level 1 Group x Entry y Routing Table Entry CSR (Offset = (L1_Group_Ptr*0x400) + (x * 0x400) + (y*4))

This register shall control the routing mode for all ports whose Port n Level 1 Info CSR L1_Group_Ptr field value is the same. The use and meaning of the bits and bit fields of this register shall be as specified in Table 3-26. Unless otherwise specified, the bits and bit fields in this register are read/write.

Table 3-26. Bit Settings for Level 1 Group x Entry y Routing Table Entry CSR

Bit	Name	Reset Value	Description
0-3	Implementation-defined	Impl. Spec	These bits control implementation specific behavior. When these bits are written to 0x0, no implementation specific function shall be invoked.
4-21	___		Reserved
22-31	Routing Value	0x300	Routing table entry 0x000 to 0x0FF - Egress Port Number 0x00 to 0xFF 0x100 to 0x1FF - Multicast Mask Number 0x00 to 0xFF. Refer to Part 11. 0x200 to 0x2FF - Level 2 Group number 0x00 to 0xFF 0x300 - Drop Packet 0x301 - Use value found in Standard Port Default Route CSR 0x302 to 0x3FF - Reserved. Selection of an Egress Port Number, Multicast Mask or Level 2 Group Number which does not exist in the device shall result in implementation specific behavior.

3.7.6 Level 2 Group x Entry y Routing Table Entry CSR (Offset = (L2_Group_Ptr*0x400) + (x * 0x400) + (y*4))

This register shall control the routing mode for all ports whose Port n Level 2 Info CSR L2_Group_Ptr field value is the same. The use and meaning of the bits and bit fields of this register shall be as specified in Table 3-27. Unless otherwise specified, the bits and bit fields in this register are read/write.

Table 3-27. Bit Settings for Level 2 Group x Entry y Routing Table Entry CSR

Bit	Name	Reset Value	Description
0-3	Implementation-defined	Impl. Spec	These bits control implementation specific behavior. When these bits are written to 0x0, no implementation specific function shall be invoked.
4-21	___		Reserved
22-31	Routing Value	0x300	Routing table entry 0x000 to 0x0FF - Egress Port Number 0x00 to 0xFF 0x100 to 0x1FF - Multicast Mask Number 0x00 to 0xFF. Refer to Part 11. 0x200 to 0x2FF - Reserved. 0x300 - Drop Packet 0x301 - Route packet using the Standard Port Default Route CSR 0x302 to 0x3FF - Reserved. Selection of an Egress Port Number or Multicast Mask Number which does not exist in the device shall result in implementation specific behavior.

Blank page

Annex A Dev32 Hierarchical Programming Model (Informative)

A.1 Dev32 Configuration Examples

This chapter provides several examples of how to use the Dev32 routing table programming interface. The given examples build upon each other while proceeding through the sections. References to the order of operations within the examples run from the top of a list to the bottom unless otherwise stated.

Initially assume a switch with 16 ports which supports Dev32 device IDs. Assume that the switch must support the following routing hierarchy, where "**" means "All Values":

- Device ID 0x00_11_20_** must be routed to port 14.
- Device IDs 0x00_11_0X_** must be routed to port X, where X is 0 to 13.
- Device IDs 0x00_ZZ_**_** must be routed to port 15 when ZZ is 0 to 0x10.
- All other packets must be dropped.

Further assume that Port 7 must be programmed to support the above hierarchy, and has initial register values as follows:

Table A-1. Example Port 7 Routing Table Register Block Registers

Register Name	Register Address	Register Value
Switch Routing Table Register Block Header	0x8000	N/A
Port 7 Routing Table Control CSR	0x8120	0x8000_0000
Port 7 Level 0 Info CSR	0x8130	0x0107_0000
Port 7 Level 1 Info CSR	0x8134	0x0307_0400
Port 7 Level 2 Info CSR	0x8138	0x0407_1000

A.1.1 Example 1: Routing 0x00_11_20_** to Port 14

To route the Dev32 destination IDs 0x00_11_20_** to Port 14, make use of routing table group 0 for Level 0, and routing table group 1 for Level 1 and Level 2. Specific entries for each level must be programmed.

Table A-2. Example 1 Accesses

Register Name	Register Address	Register Value	Description
Port 7 Level 0 Group 0 Entry 0 Routing Table Entry CSR	0x0007_0000	0x0000_0201	Map Level 0 Group 0 index 0x00 to Level 1 Group 1
Port 7 Level 1 Group 1 Entry 0x11 Routing Table Entry CSR	0x0007_0844	0x0000_0201	Map Level 1 Group 1 index 0x11 to Level 2 Group 1
Port 7 Level 2 Group 1 Entry 0x20 Routing Table Entry CSR	0x0007_1480	0x0000_000E	Map Level 2 Group 1 index 0x20 to Port 14.

A.1.2 Example 2: Routing 0x00_11_0X_** to Port X

This example builds upon the configuration put in place by Example 1. Routing configuration is therefore complete for Level 0 and Level 1, so what remains is to complete the Level 2 programming.

Table A-3. Example 2 Accesses

Register Name	Register Address	Register Value	Description
Port 7 Level 2 Group 1 Entry 0x00 Routing Table Entry CSR	0x0007_1400	0x0000_0000	Map Level 2 Group 1 index 0x00 to Port 0.
Port 7 Level 2 Group 1 Entry 0x01 Routing Table Entry CSR	0x0007_1404	0x0000_0001	Map Level 2 Group 1 index 0x01 to Port 1.
Port 7 Level 2 Group 1 Entry 0x02 Routing Table Entry CSR	0x0007_1408	0x0000_0002	Map Level 2 Group 1 index 0x02 to Port 2.
Port 7 Level 2 Group 1 Entry 0x03 Routing Table Entry CSR	0x0007_140C	0x0000_0003	Map Level 2 Group 1 index 0x03 to Port 3.
Port 7 Level 2 Group 1 Entry 0x04 Routing Table Entry CSR	0x0007_1410	0x0000_0004	Map Level 2 Group 1 index 0x04 to Port 4.
Port 7 Level 2 Group 1 Entry 0x05 Routing Table Entry CSR	0x0007_1414	0x0000_0005	Map Level 2 Group 1 index 0x05 to Port 5.
Port 7 Level 2 Group 1 Entry 0x06 Routing Table Entry CSR	0x0007_1418	0x0000_0006	Map Level 2 Group 1 index 0x06 to Port 6.
Port 7 Level 2 Group 1 Entry 0x07 Routing Table Entry CSR	0x0007_141C	0x0000_0007	Map Level 2 Group 1 index 0x07 to Port 7.
Port 7 Level 2 Group 1 Entry 0x08 Routing Table Entry CSR	0x0007_1420	0x0000_0008	Map Level 2 Group 1 index 0x08 to Port 8.
Port 7 Level 2 Group 1 Entry 0x09 Routing Table Entry CSR	0x0007_1424	0x0000_0009	Map Level 2 Group 1 index 0x09 to Port 9.
Port 7 Level 2 Group 1 Entry 0x0A Routing Table Entry CSR	0x0007_1428	0x0000_000A	Map Level 2 Group 1 index 0x0A to Port A.
Port 7 Level 2 Group 1 Entry 0x0B Routing Table Entry CSR	0x0007_142C	0x0000_000B	Map Level 2 Group 1 index 0x0B to Port B.

Table A-3. Example 2 Accesses

Register Name	Register Address	Register Value	Description
Port 7 Level 2 Group 1 Entry 0x0C Routing Table Entry CSR	0x0007_1430	0x0000_000C	Map Level 2 Group 1 index 0x0C to Port C.
Port 7 Level 2 Group 1 Entry 0x0D Routing Table Entry CSR	0x0007_1434	0x0000_000D	Map Level 2 Group 1 index 0x0D to Port D.

A.1.3 Example 3: Routing 0x00_ZZ_**_** to Port 15, ZZ=[0,0x10]

This example builds upon the configuration put in place by Example 1. Routing configuration is therefore complete for Level 0, so what remains is to program the Level 1 registers.

Table A-4. Example 3 Accesses

Register Name	Register Address	Register Value	Description
Port 7 Level 1 Group 1 Entry 0x00 Routing Table Entry CSR	0x0007_0800	0x0000_000F	Map Level 1 Group 1 index 0x00 to Port 15.
Port 7 Level 1 Group 1 Entry 0x01 Routing Table Entry CSR	0x0007_0804	0x0000_000F	Map Level 1 Group 1 index 0x01 to Port 15.
Port 7 Level 1 Group 1 Entry 0x02 Routing Table Entry CSR	0x0007_0808	0x0000_000F	Map Level 1 Group 1 index 0x02 to Port 15.
Port 7 Level 1 Group 1 Entry 0x03 Routing Table Entry CSR	0x0007_080C	0x0000_000F	Map Level 1 Group 1 index 0x03 to Port 15.
Port 7 Level 1 Group 1 Entry 0x04 Routing Table Entry CSR	0x0007_0810	0x0000_000F	Map Level 1 Group 1 index 0x04 to Port 15.
Port 7 Level 1 Group 1 Entry 0x05 Routing Table Entry CSR	0x0007_0814	0x0000_000F	Map Level 1 Group 1 index 0x05 to Port 15.
Port 7 Level 1 Group 1 Entry 0x06 Routing Table Entry CSR	0x0007_0818	0x0000_000F	Map Level 1 Group 1 index 0x06 to Port 15.
Port 7 Level 1 Group 1 Entry 0x07 Routing Table Entry CSR	0x0007_081C	0x0000_000F	Map Level 1 Group 1 index 0x07 to Port 15.
Port 7 Level 1 Group 1 Entry 0x08 Routing Table Entry CSR	0x0007_0820	0x0000_000F	Map Level 1 Group 1 index 0x08 to Port 15.
Port 7 Level 1 Group 1 Entry 0x09 Routing Table Entry CSR	0x0007_0824	0x0000_000F	Map Level 1 Group 1 index 0x09 to Port 15.
Port 7 Level 1 Group 1 Entry 0x0A Routing Table Entry CSR	0x0007_0828	0x0000_000F	Map Level 1 Group 1 index 0x0A to Port 15.
Port 7 Level 1 Group 1 Entry 0x0B Routing Table Entry CSR	0x0007_082C	0x0000_000F	Map Level 1 Group 1 index 0x0B to Port 15.
Port 7 Level 1 Group 1 Entry 0x0C Routing Table Entry CSR	0x0007_0830	0x0000_000F	Map Level 1 Group 1 index 0x0C to Port 15.
Port 7 Level 1 Group 1 Entry 0x0D Routing Table Entry CSR	0x0007_0834	0x0000_000F	Map Level 1 Group 1 index 0x0D to Port 15.
Port 7 Level 1 Group 1 Entry 0x0E Routing Table Entry CSR	0x0007_0838	0x0000_000F	Map Level 1 Group 1 index 0x0E to Port 15.

Table A-4. Example 3 Accesses

Register Name	Register Address	Register Value	Description
Port 7 Level 1 Group 1 Entry 0x0F Routing Table Entry CSR	0x0007_083C	0x0000_000F	Map Level 1 Group 1 index 0x0F to Port 15.
Port 7 Level 1 Group 1 Entry 0x10 Routing Table Entry CSR	0x0007_0840	0x0000_000F	Map Level 1 Group 1 index 0x10 to Port 15.

A.1.4 Example 4: All Other Packets Must Be Dropped

This example builds upon the configuration put in place by Example 3. Routing for Dev32 device IDs has been configured. Dev16 and Dev8 deviceIDs are routed using Group 0 of Level 1 and Level 2, respectively. The default values for all entries is to drop packets. Nothing more needs to be programmed to drop all Dev16 and Dev8 deviceIDs.

A.1.5 Example 5: Flat Routing Table Operation

This example illustrates the “flat” programming model, in which device IDs are supported sequentially by the routing tables. Dev16 device IDs of the form 0x00** are treated as Dev8 device IDs.

Initially assume a switch with 16 ports which supports Dev32 device IDs. Assume that the switch must support the following routing hierarchy, where “**” means “All Values”:

- Device ID 0x01_20 must be routed to port 14.
- Device IDs 0x00_0X must be routed to port X, where X is 0 to 13.
- Device IDs 0x02_00 and 0x03_00 must be routed to port 15.
- All other packets must be dropped.

Further assume that Port 7 must be programmed to support the above hierarchy, and has initial register values as follows:

Table A-5. Example 5 Port 7 Routing Table Register Block Registers

Register Name	Register Address	Register Value
Switch Routing Table Register Block Header	0x8000	N/A
Port 7 Routing Table Control CSR	0x8120	0x8000_0000
Port 7 Level 0 Info CSR	0x8130	0x0107_0000
Port 7 Level 1 Info CSR	0x8134	0x0307_0400
Port 7 Level 2 Info CSR	0x8138	0x0407_1000

The following register accesses must be performed:

Table A-6. Example 5 Accesses

Register Name	Register Address	Register Value	Description
Port 7 Routing Table Control CSR	0x0000_8120	0x0000_0000	Change to Flat Routing Table Model
Port 7 Level 0 Info CSR	0x0000_8130	0x0407_0000	Read Level 0 Info to determine how many DeviceIDs are supported. Four groups are supported, or destIDs 0x0000 through 0x03FF.
Port 7 Level 1 Info CSR	0x0000_8134	0x0000_0000	Read Level 1 Info, confirm register is reserved
Port 7 Level 2 Info CSR	0x0000_8138	0x0000_0000	Read Level 2 Info, confirm register is reserved
Port 7 Level 0 Group 1 Entry 0x20	0x0007_0480	0x0000_000E	Route DestID 0x0120 to port 14.
Port 7 Level 0 Group 0 Entry 0x00	0x0007_0000	0x0000_0000	Route DestID 0x0000 to port 0.
Port 7 Level 0 Group 0 Entry 0x01	0x0007_0004	0x0000_0001	Route DestID 0x0001 to port 1.
Port 7 Level 0 Group 0 Entry 0x02	0x0007_0008	0x0000_0002	Route DestID 0x0002 to port 2.
Port 7 Level 0 Group 0 Entry 0x03	0x0007_000C	0x0000_0003	Route DestID 0x0003 to port 3.
Port 7 Level 0 Group 0 Entry 0x04	0x0007_0010	0x0000_0004	Route DestID 0x0004 to port 4.
Port 7 Level 0 Group 0 Entry 0x05	0x0007_0014	0x0000_0005	Route DestID 0x0005 to port 5.
Port 7 Level 0 Group 0 Entry 0x06	0x0007_0018	0x0000_0006	Route DestID 0x0006 to port 6.
Port 7 Level 0 Group 0 Entry 0x07	0x0007_001C	0x0000_0007	Route DestID 0x0007 to port 7.
Port 7 Level 0 Group 0 Entry 0x08	0x0007_0020	0x0000_0008	Route DestID 0x0008 to port 8.
Port 7 Level 0 Group 0 Entry 0x09	0x0007_0024	0x0000_0009	Route DestID 0x0009 to port 9.
Port 7 Level 0 Group 0 Entry 0x0A	0x0007_0028	0x0000_000A	Route DestID 0x000A to port 10.
Port 7 Level 0 Group 0 Entry 0x0B	0x0007_002C	0x0000_000B	Route DestID 0x000B to port 11.
Port 7 Level 0 Group 0 Entry 0x0C	0x0007_0030	0x0000_000C	Route DestID 0x000C to port 12.
Port 7 Level 0 Group 0 Entry 0x0D	0x0007_0034	0x0000_000D	Route DestID 0x000D to port 13.
Port 7 Level 0 Group 2 Entry 0x00	0x0007_0800	0x0000_000F	Route DestID 0x0200 to port 15.
Port 7 Level 0 Group 3 Entry 0x00	0x0007_0C00	0x0000_000F	Route DestID 0x0300 to port 15.

Glossary of Terms and Abbreviations

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

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.

Broadcast. The concept of sending a packet to all processing elements in a system.

C

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

Command and status registers (CSRs). A set of registers that allows a processing element to control and determine the status of another processing element’s internal hardware.

D

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.

E

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.

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.

H

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

I

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

I/O. Input-output.

M

MSB. Most significant byte.

Multicast. The concept of sending a packet to more than one processing elements in a system.

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.

Processing Element (**PE**). A generic participant on the RapidIO interconnect that sends or receives RapidIO transactions, also called a device.

S

Source. The origin of a packet on the RapidIO interconnect, also referred to as an initiator.

Switch. A multiple port processing element that directs a packet received on one of its input ports to one of its output ports.

T

Target. The termination point of a packet on the RapidIO interconnect, also referred to as a destination.

Transaction. A specific request or response packet transmitted between end point devices in a RapidIO system.

RapidIO™ Interconnect Specification Part 4: Physical Layer 8/16 LP-LVDS Specification

3.0, 10/2013

RapidIO.org

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
1.3	Technical changes: incorporate Rev 1.2 errata 1 as applicable, the following errata showings: 03-07-00000.002, 03-12-00000.002, 03-12-00002.004 and the following new features showings: 02-06-00001.004, 02-03-00003.004, 04-08-00013.002, 04-09-00022.002 Convert to ISO-friendly templates; re-formatted	02/23/2005
2.0	Technical changes: errata showing 06-04-00000.003 new features showing 05-04-00001.005	06/14/2007
2.1	No technical changes	07/09/2009
2.2	Technical changes: errata showing 10-08-00001.005, Consolidated Comments on 11-01-00000.000	05/05/2011
3.0	Changed RTA contact information. No technical changes. This part of the specification is now deprecated.	10/11/2013

Table of Contents

Chapter 1 Overview

1.1 Introduction.. 13
1.2 Overview.. 13
1.3 Features of the Input/Output Specification.. 13
1.3.1 Functional features.. 13
1.3.2 Physical Features.. 14
1.3.3 Performance Features.. 14
1.4 Contents.. 14
1.5 Terminology.. 15
1.6 Conventions.. 15

Chapter 2 Physical Layer Protocol

2.1 Introduction.. 17
2.2 Packet Exchange Protocol.. 17
2.2.1 Packet and Control Alignment.. 18
2.2.2 Acknowledge Identification.. 19
2.3 Field Placement and Definition.. 19
2.3.1 Flow Control Fields Format.. 19
2.3.2 Packet Priority and Transaction Request Flows.. 21
2.3.3 Transaction and Packet Delivery.. 22
2.3.3.1 Transaction and Packet Delivery Ordering Rules.. 23
2.3.3.2 Deadlock Avoidance.. 24
2.3.4 Resource Allocation.. 26
2.3.4.1 Receiver-Controlled Flow Control.. 26
2.3.4.2 Transmitter-Controlled Flow Control.. 28
2.3.4.3 Receive Buffer Management.. 29
2.3.4.4 Effective Number of Free Receive Buffers.. 30
2.3.4.5 Speculative Packet Transmission.. 31
2.3.5 Flow Control Mode Negotiation.. 31
2.4 Error Detection and Recovery.. 32
2.4.1 Control Symbol Protection.. 32
2.4.2 Packet Protection.. 33
2.4.3 Lost Packet Detection.. 34
2.4.4 Implementation Note: Transactional Boundaries.. 35
2.4.5 Link Behavior Under Error.. 35
2.4.5.1 Recoverable Errors.. 35
2.4.5.1.1 Packet Errors.. 36
2.4.5.1.2 Control Symbol Errors.. 36
2.4.5.1.3 Indeterminate errors.. 37
2.4.5.1.4 Timeout Error.. 37

Table of Contents

2.4.6 CRC Operation 37
2.4.7 CRC Code 40
2.5 Maximum Packet Size 42
2.6 Link Maintenance Protocol 42
2.6.1 Command Descriptions 43
2.6.1.1 Reset and Safety Lockouts 43
2.6.1.2 Input-status 44
2.6.1.3 Send-training 44
2.6.2 Status Descriptions 44

Chapter 3 Packet and Control Symbol Transmission

3.1 Introduction 47
3.2 Packet Start and Control Symbol Delineation 47
3.3 Packet Termination 49
3.4 Packet Pacing 50
3.5 Embedded Control Symbols 52
3.6 Packet to Port Alignment 52
3.7 System Maintenance 56
3.7.1 Port and Link Initialization 56
3.7.1.1 Sampling Window Alignment 56
3.7.1.1.1 Port Width Mode Selection 56
3.7.1.1.2 Input Sampling Window Alignment 57
3.7.1.1.3 Training Pattern 57
3.7.1.1.4 Training Pattern Transmission 58
3.7.1.1.5 Ports Not Requiring Port Initialization 58
3.7.1.1.6 Ports Requiring Port Initialization 59
3.7.1.1.7 Port Initialization Process 59
3.7.1.2 Link Initialization 60
3.7.1.3 Maintenance Training 60
3.7.1.4 Unexpected Training Pattern Reception 61
3.7.2 Multicast-Event 61
3.8 Power Management 62

Chapter 4 Control Symbol Formats

4.1 Introduction 63
4.2 Acknowledgment Control Symbol Formats 63
4.2.1 Packet-Accepted Control Symbol 64
4.2.2 Packet-Retry Control Symbol 64
4.2.3 Packet-Not-Accepted Control Symbol 64
4.2.4 Canceling Packets 65
4.3 Packet Control Symbol Formats 66
4.4 Link Maintenance Control Symbol Formats 67
4.5 Reserved Symbol Formats 70
4.6 Implementation-defined Symbol Formats 70

Table of Contents

4.7 Control Symbol to Port Alignment 70

Chapter 5 8/16 LP-LVDS Registers

5.1 Introduction 73
5.2 Register Map 73
5.3 Reserved Register, Bit and Bit Field Value Behavior 74
5.4 Capability Registers (CARs) 76
5.4.1 Processing Element Features CAR 76
5.5 Generic End Point Devices 77
5.5.1 Register Map 77
5.5.2 Command and Status Registers (CSRs) 78
5.5.2.1 8/16 LP-LVDS Register Block Header 78
5.5.2.2 Port Link Timeout Control CSR 78
5.5.2.3 Port Response Timeout Control CSR 79
5.5.2.4 Port General Control CSR 79
5.5.2.5 Port n Error and Status CSRs 80
5.5.2.6 Port n Control CSR 81
5.6 Generic End Point Devices, software assisted error recovery option 83
5.6.1 Register Map 83
5.6.2 Command and Status Registers (CSRs) 85
5.6.2.1 8/16 LP-LVDS Register Block Header 85
5.6.2.2 Port Link Timeout Control CSR 85
5.6.2.3 Port Response Timeout Control CSR 86
5.6.2.4 Port General Control CSR 86
5.6.2.5 Port n Link Maintenance Request CSRs 87
5.6.2.6 Port n Link Maintenance Response CSRs 87
5.6.2.7 Port n Local ackID Status CSRs 88
5.6.2.8 Port n Error and Status CSRs 89
5.6.2.9 Port n Control CSR 90
5.7 Generic End Point Free Devices 92
5.7.1 Register Map 92
5.7.2 Command and Status Registers (CSRs) 93
5.7.2.1 8/16 LP-LVDS Register Block Header 93
5.7.2.2 Port Link Timeout Control CSR 93
5.7.2.3 Port General Control CSR 94
5.7.2.4 Port n Error and Status CSRs 95
5.7.2.5 Port n Control CSR 96
5.8 Generic End Point Free Devices, software assisted error recovery option 98
5.8.1 Register Map 98
5.8.2 Command and Status Registers (CSRs) 100
5.8.2.1 8/16 LP-LVDS Register Block Header 100
5.8.2.2 Port Link Timeout Control CSR 100
5.8.2.3 Port General Control CSR 101
5.8.2.4 Port n Link Maintenance Request CSRs 101
5.8.2.5 Port n Link Maintenance Response CSRs 102

Table of Contents

5.8.2.6 Port n Local ackID Status CSRs 102
5.8.2.7 Port n Error and Status CSRs 103
5.8.2.8 Port n Control CSR 104

Chapter 6 System Clocking Considerations

6.1 Introduction 107
6.2 Example Clock Distribution 107
6.3 Elasticity Mechanism 108

Chapter 7 Board Routing Guidelines

7.1 Introduction 111
7.2 Impedance 111
7.3 Skew 111
7.4 PCB Stackup 112
7.5 Termination 113
7.6 Additional Considerations 113
7.6.1 Single Board Environments 113
7.6.2 Single Connector Environments 113
7.6.3 Backplane Environments 114
7.7 Recommended pin escape ordering 114

Chapter 8 Signal Descriptions

8.1 Introduction 117
8.2 Signal Definitions 117
8.3 RapidIO Interface Diagrams 119

Chapter 9 Electrical Specifications

9.1 Introduction 121
9.2 Overview 121
9.3 DC Specifications 122
9.4 AC Specifications 124
9.4.1 Concepts and Definitions 124
9.4.2 Driver Specifications 127
9.4.3 Receiver Specifications 133

Annex A Interface Management (Informative)

A.1 Introduction 141
A.2 Link Initialization and Maintenance Mechanism 141
A.2.1 Input port training state machine 141
A.2.2 Output port training state machine 144

Table of Contents

A.3 Packet Retry Mechanism 147
A.3.1 Input port retry recovery state machine 147
A.3.2 Output port retry recovery state machine 148
A.4 Error Recovery 151
A.4.1 Input port error recovery state machine 151
A.4.2 Output port error recovery state machine 152

Table of Contents

Blank page

List of Figures

2-1 Example Transaction with Acknowledge 18
2-2 Packet Physical Layer Fields Format....20
2-3 Control Symbol Physical Layer Fields Format....20
2-4 Flow Control Fields Bit Stream 21
2-5 Receiver-Controlled Flow Control 28
2-6 Transmitter-Controlled Flow Control 29
2-7 Error Coverage of First 16 Bits of Packet Header 33
2-8 Naturally Aligned Packet Bit Stream....33
2-9 Naturally Aligned Packet Bit Stream Example 1 39
2-10 Naturally Aligned Packet Bit Stream Example 2 39
2-11 Padded Aligned Packet Bit Stream Example 1....39
2-12 Padded Aligned Packet Bit Stream Example 2....40
2-13 CRC Generation Pipeline....41
3-1 Framing Signal Maximum Toggle Rate for 8-bit Port 48
3-2 Framing Signal Maximum Toggle Rate for 16-bit Port 48
3-3 Control Symbol Delineation Example for 8-bit Port....48
3-4 Control Symbol Delineation Example for 16-bit Port....49
3-5 Header Marked End of Packet (8-bit Port) 50
3-6 End-Of-Packet Control Symbol Marked End of Packet (16-bit Port) 50
3-7 Pacing Idle Insertion in Packet (8-bit Port) 51
3-8 Embedded Control Symbols for 8-bit Port 52
3-9 Embedded Control Symbols for 16-bit Port 52
3-10 Request Packet Transmission Example 1 53
3-11 Request Packet Transmission Example 2 53
3-12 Request Packet Transmission Example 3 54
3-13 Request Packet Transmission Example 4 54
3-14 Response Packet Transmission Example 1....55
3-15 Response Packet Transmission Example 2....55
4-1 Type 0 Packet-Accepted Control Symbol Format....64
4-2 Type 1 Packet-Retry Control Symbol Format 64
4-3 Type 2 Packet-Not-Accepted Control Symbol Format....65
4-4 Type 4 Packet Control Symbol Format 66
4-5 Type 5 Link-Request Control Symbol Format 68
4-6 Type 6 Link-Response Control Symbol Format....68
4-7 Control Symbol Transmission Example 1 70
4-8 Control Symbol Transmission Example 2 71
6-1 Clock Distribution in a Small System....107
6-2 Clock Distribution in a Larger System 108
6-3 Clock Distribution Through the Interconnect....108
7-1 Routing for Equalized Skew for Several Placements 112
7-2 Potential PCB Stackups 112

List of Figures

7-3 Recommended device pin escape, input port, top view of device 114
7-4 Recommended device pin escape, output port, top view of device 114
7-5 Opposed orientation, same side of board 115
7-6 Opposed orientation, opposite sides of board 115
7-7 Recommended device pin escape, output port reversed, top view of device 116
7-8 Opposed orientation, output port reversed, opposite sides of board 116
8-1 RapidIO 8-bit Device to 8-bit Device Interface Diagram 119
8-2 RapidIO 8-bit Device to 16-bit Device Interface Diagram 119
8-3 RapidIO 16-bit Device to 16-bit Device Interface Diagram 120
9-1 DC driver signal levels 123
9-2 Differential Peak-Peak Voltage of Transmitter or Receiver 124
9-3 Example Compliance Mask 126
9-4 RapidIO Transmit Mask 131
9-5 Example Driver Output Eye Pattern 132
9-6 RapidIO Receive Mask 135
9-7 Example Receiver Input Eye Pattern 137
9-8 Data to Clock Skew 138
9-9 Clock to Clock Skew 139
9-10 Static Skew Diagram 140
A-1 Input port training state machine 142
A-2 Output port training state machine 144
A-3 Input port retry recovery state machine 147
A-4 Output port retry recovery state machine 149
A-5 Input port error recovery state machine 151
A-6 Output port error recovery state machine 153

List of Tables

2-1 Fields that Control Packet Flow....19
2-2 buf_status Field Definition 20
2-3 Transaction Request Flow to Priority Mapping 21
2-4 Transaction Request Flow to Priority and Critical Request Flow Mapping....22
2-5 Parallel CRC Intermediate Value Equations 40
2-6 Maximum Packet Size 42
2-7 Secondary Link Maintenance Command Summary 43
2-8 Link Status Indicators 44
4-1 Field Definitions for Acknowledgment Control Symbols....63
4-2 cause Field Definition....65
4-3 sub_type and contents Field Definitions....66
4-4 Throttle Control Symbol contents Field Definition....67
4-5 cmd Field Definition....68
4-6 ackID_status Field Definition....69
4-7 link_status Field Definition 69
5-1 8/16 LP-LVDS Register Map 74
5-2 Configuration Space Reserved Access Behavior....74
5-3 Bit Settings for Processing Element Features CAR....76
5-4 8/16 LP-LVDS Register Map - Generic End Point Devices 77
5-5 Bit Settings for 8/16 LP-LVDS Register Block Header....78
5-6 Bit Settings for Port Link Timeout Control CSR 78
5-7 Bit Settings for Port Response Timeout Control CSR....79
5-8 Bit Settings for Port General Control CSRs 79
5-9 Bit Settings for Port n Error and Status CSRs 80
5-10 Bit Settings for Port n Control CSRs....81
5-11 8/16 LP-LVDS Register Map - Generic End Point Devices (SW assisted) 83
5-12 Bit Settings for 8/16 LP-LVDS Register Block Header....85
5-13 Bit Settings for Port Link Timeout Control CSR 85
5-14 Bit Settings for Port Response Timeout Control CSR....86
5-15 Bit Settings for Port General Control CSRs 86
5-16 Bit Settings for Port n Link Maintenance Request CSRs 87
5-17 Bit Settings for Port n Link Maintenance Response CSRs....87
5-18 Bit Settings for Port n Local ackID Status CSRs....88
5-19 Bit Settings for Port n Error and Status CSRs 89
5-20 Bit Settings for Port n Control CSRs....90
5-21 8/16 LP-LVDS Register Map - Generic End Point Free Devices 92
5-22 Bit Settings for 8/16 LP-LVDS Register Block Header....93
5-23 Bit Settings for Port Link Timeout Control CSR 93
5-24 Bit Settings for Port General Control CSRs 94
5-25 Bit Settings for Port n Error and Status CSRs 95
5-26 Bit Settings for Port n Control CSRs....96
5-27 8/16 LP-LVDS Register Map - Generic End Point-free Devices (SW assisted)....98

List of Tables

5-28 Bit Settings for 8/16 LP-LVDS Register Block Header 100
5-29 Bit Settings for Port Link Timeout Control CSR 100
5-30 Bit Settings for Port General Control CSRs 101
5-31 Bit Settings for Port n Link Maintenance Request CSRs 101
5-32 Bit Settings for Port n Link Maintenance Response CSRs....102
5-33 Bit Settings for Port n Local ackID Status CSRs....102
5-34 Bit Settings for Port n Error and Status CSRs 103
5-35 Bit Settings for Port n Control CSRs 104
8-1 8/16 LP-LVDS Signal Descriptions 117
9-1 RapidIO 8/16 LP-LVDS Driver Specifications (DC) 122
9-2 RapidIO 8/16 LP-LVDS Receiver Specifications (DC) 122
9-3 Driver AC Timing Specifications - 500Mbps Data Rate/250MHz Clock Rate 127
9-4 Driver AC Timing Specifications - 750Mbps Data Rate/375MHz Clock Rate 128
9-5 Driver AC Timing Specifications - 1000Mbps Data Rate/500MHz Clock Rate 128
9-6 Driver AC Timing Specifications - 1500Mbps Data Rate/750MHz Clock Rate 129
9-7 Driver AC Timing Specifications - 2000Mbps Data Rate/1000MHz Clock Rate 130
9-8 Receiver AC Timing Specifications - 500Mbps Data Rate/250MHz Clock Rate....133
9-9 Receiver AC Timing Specifications - 750Mbps Data Rate/375MHz Clock Rate....133
9-10 Receiver AC Timing Specifications - 1000Mbps Data Rate/500MHz Clock Rate....134
9-11 Receiver AC Timing Specifications - 1500Mbps Data Rate/750MHz Clock Rate....134
9-12 Receiver AC Timing Specifications - 2000Mbps Data Rate/1000MHz Clock Rate....135
A-1 Input port training state machine transition table 142
A-2 Output port training state machine transition table....145
A-3 Input port retry recovery state machine transition table 148
A-4 Output port retry recovery state machine transition table....149
A-5 Input port error recovery state machine transition table 152
A-6 Output port error recovery state machine transition table 153

Chapter 1 Overview

1.1 Introduction

This chapter provides an overview of the *RapidIO Part 4: 8/16 LP-LVDS Physical Layer Specification*, including a description of the relationship between this specification and the other specifications of the RapidIO interconnect.

1.2 Overview

The *RapidIO Part 4: 8/16 LP-LVDS Physical Layer Specification* is one of the RapidIO physical layer specifications that define the device to device communications protocol and packet formats. Other RapidIO physical layer specifications include the *RapidIO Part 6: 1x/4x LP-Serial Physical Layer Specification*.

The physical layer defines the signal definitions, flow control and error management for RapidIO. An 8-bit and 16-bit parallel (8/16 LP-LVDS), point-to-point interface is defined in this specification. 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.

1.3 Features of the Input/Output Specification

The following are features of the RapidIO I/O specification designed to satisfy the needs of various applications and systems:

1.3.1 Functional features

- 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.

1.3.2 Physical Features

- 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.

1.3.3 Performance Features

- 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.

1.4 Contents

RapidIO Part 4: 8/16 LP-LVDS Physical Layer Specification contains nine chapters and an appendix:

- Chapter 1, “Overview” (this chapter) provides an overview of the specification
- Chapter 2, “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 3, “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 4, “Control Symbol Formats,” explains the physical layer control formats that manage the packet delivery protocols mentioned in Chapter 2.
- Chapter 5, “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 6, “System Clocking Considerations,” discusses the RapidIO synchronous clock and how it is distributed in a typical switch configuration.
- Chapter 7, “Board Routing Guidelines,” explains board layout guidelines and application environment considerations for the RapidIO architecture.
- Chapter 8, “Signal Descriptions,” contains the signal pin descriptions for a RapidIO end point device.
- Chapter 9, “Electrical Specifications,” describes the low voltage differential signaling (LVDS) electrical specifications of the RapidIO 8/16 LP-LVDS device.
- Annex A, “Interface Management (Informative),” contains information pertinent to interface management in a RapidIO system, including SECDED error tables, error recovery, link initialization, and packet retry state machines.

1.5 Terminology

Refer to the Glossary at the back of this document.

1.6 Conventions

||	Concatenation, used to indicate that two fields are physically associated as consecutive bits
ACTIVE_HIGH	Names of active high signals are shown in uppercase text with no overbar. Active-high signals are asserted when high and not asserted when low.
$\overline{\text{ACTIVE\_LOW}}$	Names of active low signals are shown in uppercase text with an overbar. Active low signals are asserted when low and not asserted when high.
italics	Book titles in text are set in italics.
REG[FIELD]	Abbreviations or acronyms for registers are shown in uppercase text. Specific bits, fields, or ranges appear in brackets.
TRANSACTION	Transaction types are expressed in all caps.
operation	Device operation types are expressed in plain text.
n	A decimal value.
[n-m]	Used to express a numerical range from n to m.
0bnn	A binary value, the number of bits is determined by the number of digits.
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.
x	This value is a don’t care

Chapter 2 Physical Layer Protocol

2.1 Introduction

This chapter describes the *RapidIO Part 4: 8/16 LP-LVDS Physical Layer Specification* 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.

2.2 Packet Exchange Protocol

The *RapidIO Part 4: 8/16 LP-LVDS Physical Layer 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 2-1 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. An end point device shall transmit an acknowledge control symbol for a request before the response transaction corresponding to that request.

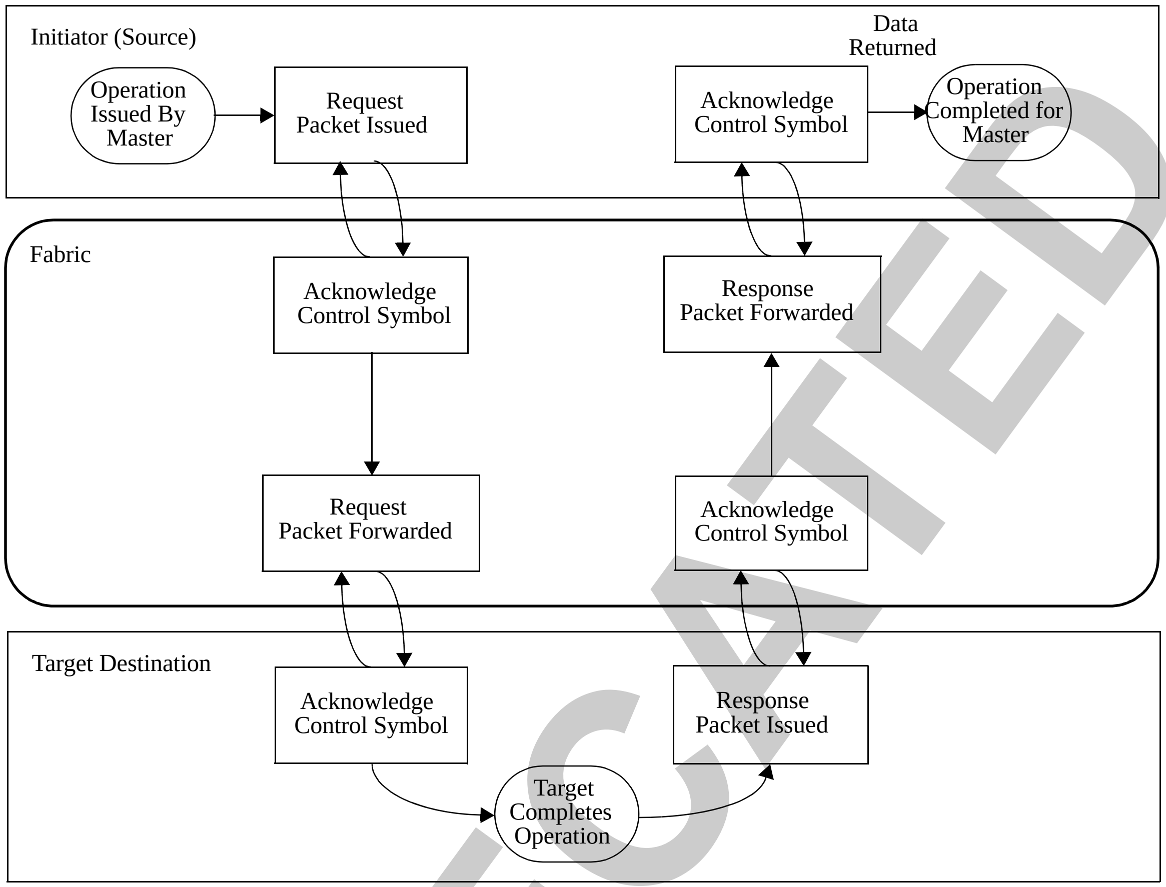

Initiator (Source)
Operation Issued By Master
Request Packet Issued
Acknowledge Control Symbol
Data Returned
Operation Completed for Master
Fabric
Acknowledge Control Symbol
Response Packet Forwarded
Request Packet Forwarded
Acknowledge Control Symbol
Target Destination
Acknowledge Control Symbol
Response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

Figure 2-1. Example Transaction with Acknowledge

2.2.1 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 2-11 and Figure 2-12 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 8, “Signal Descriptions”.

2.2.2 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 4, "Control Symbol Formats."

2.3 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.

2.3.1 Flow Control Fields Format

The fields used to control packet flow in the system are described in Table 2-1.

Table 2-1. 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 2.2.2
CRF	Critical Request Flow is an optional bit that differentiates between flows of equal priority If Critical Request Flow is not supported, this bit is reserved See Section 2.3.2 for an explanation of prioritizing packets
prio	Sets packet priority: 0b00 - lowest priority 0b01 - medium priority 0b10 - high priority 0b11 - highest priority See Section 2.3.2 for an explanation of prioritizing packets
buf_status	Specifies the number of available packet buffers in the receiving device. See Section 2.3.4 and Table 2-2.
stype	Control symbol type—see Chapter 4, "Control Symbol Formats" for definition.
rsrv	Reserved

Table 2-2. 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	

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

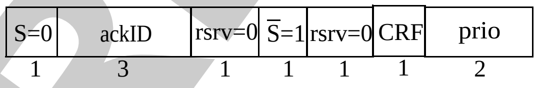


Figure 2-2. Packet Physical Layer Fields Format

Figure 2-3 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 4, “Control Symbol Formats.”

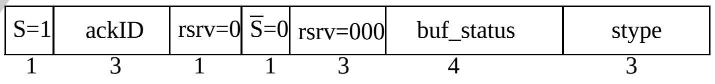


Figure 2-3. Control Symbol Physical Layer Fields Format

Figure 2-4 shows how the physical layer fields are prefixed to the combined transport and logical layer packet.

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

Figure 2-4. 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 Part 3: 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.

2.3.2 Packet Priority and Transaction Request Flows

Each packet has a priority, and optionally a critical request flow, 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 RapidIO for several purposes which include transaction ordering and deadlock prevention. The critical request flow is carried in the CRF bit. It allows a flow to be designated as a critical or preferred flow with respect to other flows of the same priority. Support for critical request flows is strongly encouraged.

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 (and optionally the CRF bit) of the packet. If the CRF bit is not supported, 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 2-3.

Table 2-3. 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

If the CRF bit is supported, the transaction request flows are mapped similarily as specified in Table 2-4. Devices that do not support the CRF bit treat it as reserved, setting it to logic 0 on transmit and ignoring it on receive.

Table 2-4. Transaction Request Flow to Priority and Critical Request Flow Mapping

Flow	System Priority	CRF Bit Setting	Request Packet Priority	Response Packet Priority
F or higher	Highest	1	2	3
E	Higher than A, B, C, D	0	2	3
D	Higher than A, B, C	1	1	2 or 3
C	Higher than A, B	0	1	2 or 3
B	Higher than A	1	0	1, 2, or 3
A	Lowest	0	0	1, 2, or 3

The mapping of transaction request flows allows a RapidIO transport fabric to maintain transaction request flow ordering without the fabric having any knowledge of transaction types or their interdependencies. This allows a RapidIO 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 and critical request flow for each path through the fabric and can maintain transaction request flow priority by never allowing a lower priority packet to pass a higher priority packet taking the same path through the fabric. In the case of congestion or some other restriction, a set CRF bit indicates that a flow of a priority can pass a flow of the same priority without the CRF bit set.

2.3.3 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 2-1), 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.3, "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 2.4.5.

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

2.3.3.1 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 or critical request flow 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, the same critical request flow, 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).**

2.3.3.2 Deadlock Avoidance

To allow a RapidIO protocol to evolve without changing the switching fabric, switch processing elements are not required, with the sole exception of ftype 8 maintenance transactions, to discern between packet types, their functions or their interdependencies. Switches, for instance, are not required to discern between packets carrying request transactions and packets carrying response transactions. As a result, it is possible for two end points, A and B to each fill all of their output buffers, the fabric connecting them and the other end point’s input buffers with read requests. This would result in an input to output dependency loop in each end point in which there would be no buffer space to hold the responses necessary to complete any of the outstanding read requests.

To break input to output dependencies, end point processing elements must have the ability to issue outbound response packets even if outbound request packets awaiting transmission are congestion blocked by the connected device. Two techniques are provided to break input to output dependencies. First, a response packet (a packet carry a response transaction) is always assigned an initial priority one priority level greater than the priority of the associated request packet (the packet carrying the associated request transaction). The CRF bit setting is assigned the same value as was received in the associated request packet if the CRF bit is supported by the receiving end point.

This requirement is specified in Table 1-3 and Table 2-4. It breaks the dependency cycle at the request flow level. Second, the end point processing element that is the source of the response packet may additionally raise the priority of the response packet to a priority higher than the minimum required by Table 1-3 and Table 2-4 if necessary for the packet to be accepted by the connected device. This additional increase in response packet priority above the minimum required by Table 1-3 and Table 2-4 is called promotion. An end point processing element may promote a response packet only to the degree necessary for the packet to be accepted by the connected device.

The following rules define the deadlock prevention mechanism:

Deadlock Prevention Rules:

1. **A RapidIO fabric shall be dependency cycle free for all operations that do not require a response.** (This rule is necessary as there are no mechanisms provided in the fabric to break dependency cycles for operations not requiring responses.)

2. **A packet carrying a request transaction that requires a response shall not be issued at the highest priority.** (This rule ensures that an end point processing element can issue a response packet at a priority higher then the priority of the associated request. This rule in combination with rule 3 are basis for the priority assignments in Table 1-3 and Table 2-4.)
3. **A packet carrying a response shall have a priority at least one priority level higher than the priority of the associated request.** (This rule in combination with rule 2 are basis for the priority assignments in Table 1-3 and Table 2-4.)
4. **A switch processing element port shall accept an error-free packet of priority N if there is no packet of priority greater than or equal to N that was previously received by the port and is still waiting in the switch to be forwarded.** (This rule has multiple implications which include but are not limited to the following. First, a switch processing element port must have at least as many maximum length packet input buffers as there are priority levels. Second, a minimum of one maximum length packet input buffer must be reserved for each priority level. A input buffer reserved for priority N might be restricted to only priority N packets or might be allowed to hold packets of priority greater than or equal to N, either approach complies with the rule.)
5. **A switch processing element port that transmits a priority N packet that is forced to retry by the connected device shall select a packet of priority greater than N, if one is available, for transmission.** (This guarantees that packets of a given priority will not block higher priority packets.)
6. **An end point processing element port shall accept an error-free packet of priority N if the port has enough space for the packet in the input buffer space of the port allocated for packets of priority N.** (Lack of input buffer space is the only reason an end point may retry a packet.)
7. **The decision of an end point processing element to accept or retry an error-free packet of priority N shall not be dependent on the ability of the end point to issue request packets of priority less than or equal to N from any of its ports.** (This rule works in conjunction with rule 6. It prohibits a device’s inability to issue packets of priority less than or equal to N, due to congestion in the connected device, from resulting in a lack of buffers to receive inbound packets of priority greater than or equal to N which in turn would result in packets of priority greater than or equal to N being forced to retry. The implications and some ways of complying with this rule are presented in the following paragraphs.)

One implication of Rule 7 is that a port may not fill all of its buffers that can be used to hold packets awaiting transmission with packets carrying request transactions. If this situation was allowed to occur and the output was blocked due to congestion in the connected device, read transactions could not be processed (no place to put the response packet), input buffer space would become filled and all subsequent inbound request packets would be forced to retry violating Rule 7.

Another implication is that a port must have a way of preventing output blockage at priority less than or equal to N, due to congestion in the connected device, from resulting in a lack of input buffer space for inbound packets of priority greater than or equal to N. There are multiple ways of doing this.

One way is to provide a port with input buffer space for at least four maximum length packets and reserve input buffer space for higher priority packets in a manner similar to that required by Rule 4 for switches. In this case, output port blockage at priority less than or equal to N will not result in blocking inbound packets of priority greater than or equal to N as any responses packets they generate will be of priority greater than N which is not congestion blocked. The port must however have the ability to select packets of priority greater than N for transmission from the packets awaiting transmission. This approach does not require the use of response packet priority promotion.

Alternatively, a port that does not have enough input buffer space for at least four maximum length packets or that does not reserve space for higher priority packets can use the promotion mechanism to increase the priority of response packets until they are accepted by the connected device. This allows output buffer space containing response packets to be freed even though all request packets awaiting transmission are congestion blocked.

As an example, suppose an end point processing element has a blocked input port because all available resources are being used for a response packet that the processing element is trying to send. If the response packet is retried by the downstream processing element, raising the priority of the response packet until it is accepted allows the processing element's input port to unblock so the system can make forward progress.

2.3.4 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.

2.3.4.1 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 1s 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.

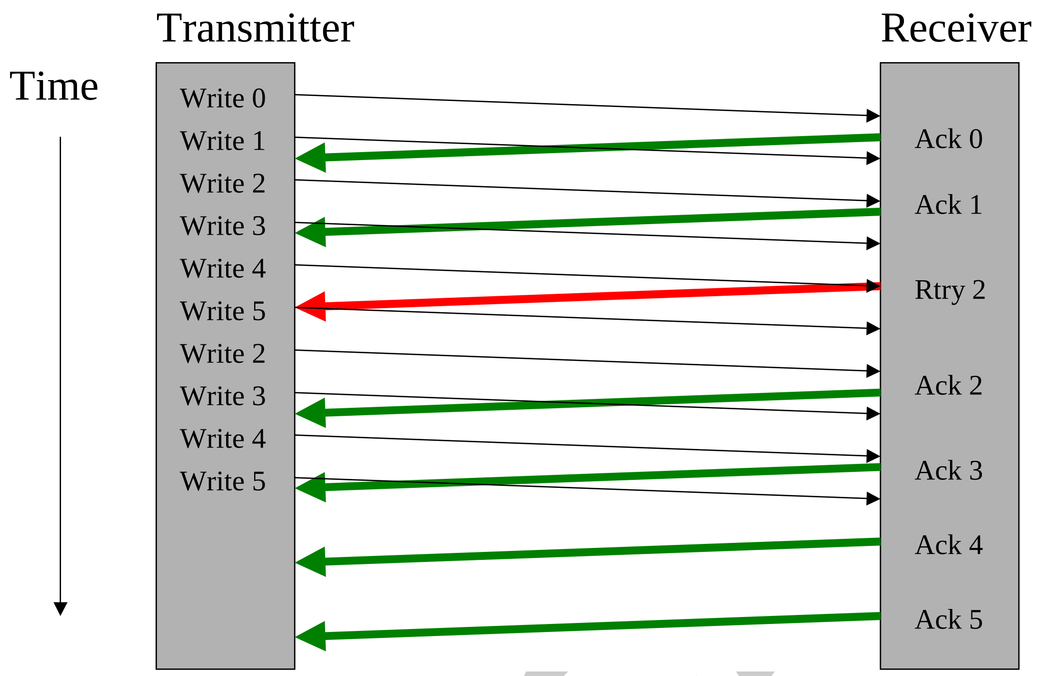
A port operating in receiver-controlled flow control mode accepts or rejects each inbound packet based on whether the receiving port has enough buffer space available at the priority level of the packet. If there is enough buffer space available, the port accepts the packet and transmits a packet-accepted 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) a packet, it behaves as described in Section 2.3.3, “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 2.3.3. 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 2-5 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 2-5. Receiver-Controlled Flow Control



2.3.4.2 Transmitter-Controlled Flow Control

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

A port signals its link partner that it is operating in transmitter-controlled flow control mode by setting the buf_status field to a value different from all 1s 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.

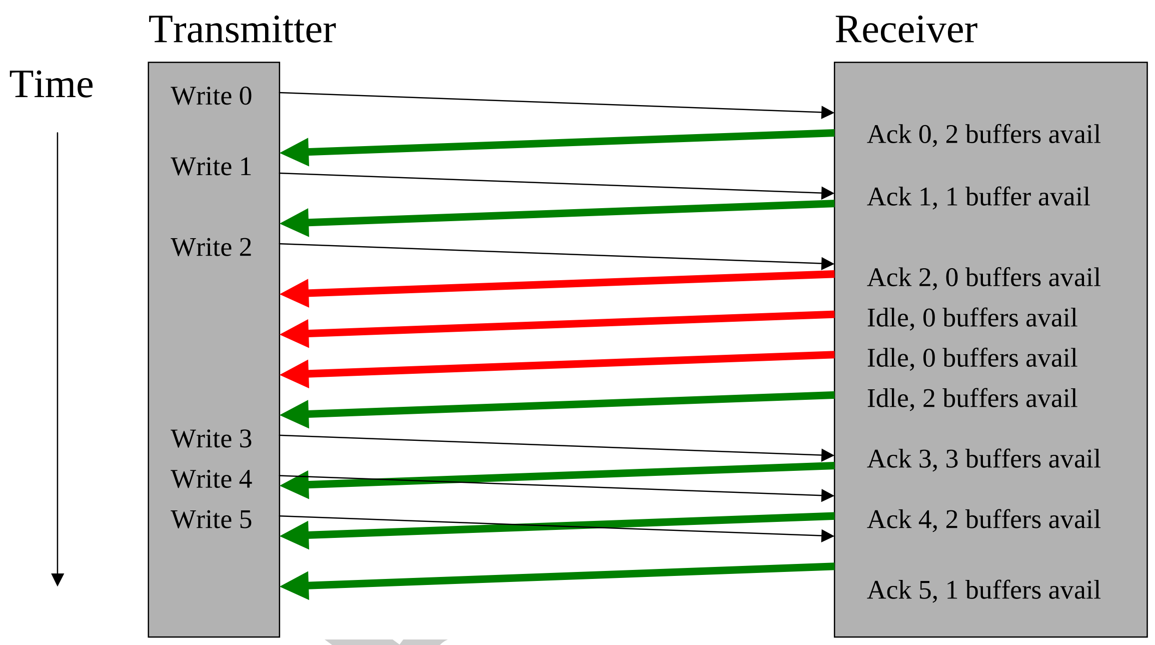
The number of free buffers that a port has available for packet reception is conveyed to its link partner by the value of the buf_status field in control symbols that the port transmits. The value conveyed by the buf_status field is the number of maximum length packet buffers currently available for packet reception up to the limit that can be 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 2.3.3.

A simple example of transmitter-controlled flow control is shown in Figure 2-6.

Figure 2-6. Transmitter-Controlled Flow Control



2.3.4.3 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 2.3.2, “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.

2.3.4.4 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 = received_buf_status;
```

2.3.4.5 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.

2.3.5 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.

2.4 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.

2.4.1 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 2.2.1. 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.

2.4.2 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 2-7 shows the error coverage for the first 16 bits of a packet header. CRC protects the prio, tt, and ftype fields and one 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.

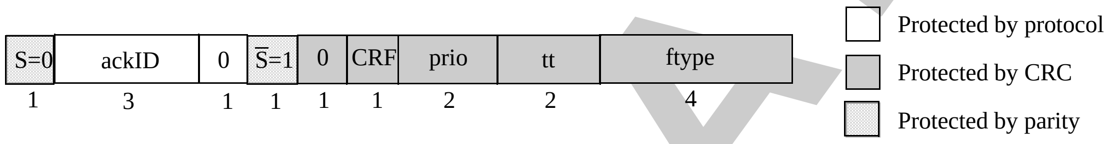


Figure 2-7. Error Coverage of First 16 Bits of Packet Header

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

NOTE:

All packets defined in the combination of this specification and the RapidIO interconnect logical and common transport specifications are now evenly divisible by 16 bits, or the complete packets are now naturally 16-bit aligned. This is illustrated in Figure 2-8. 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.

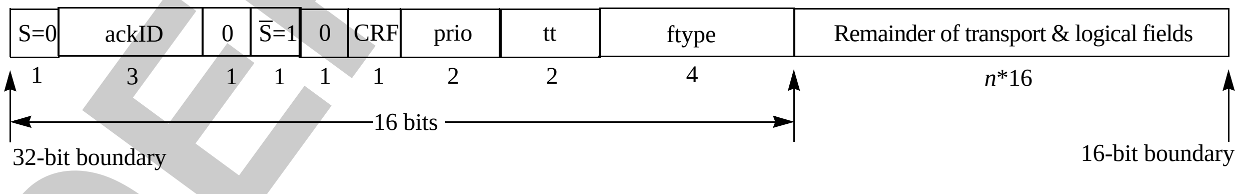


Figure 2-8. Naturally Aligned Packet Bit Stream

2.4.3 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 timeout 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 timeout is not signaled. The response to this error condition is implementation dependent.

The RapidIO specifications assume an implementation has timeout counters for the physical layer, the port link timeout counters, and counters for the logical layer, the port response timeout 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 5, “8/16 LP-LVDS Registers,” on page 73. 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 timeout occurs between the transmission of a packet and the receipt of an acknowledgment control symbol. This timeout 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 timeout should be treated as a packet acknowledge error.

The logical layer timeout occurs between the issuance of a request packet that requires a response packet and the receipt of that response packet. This timeout 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 timeout will occur. This timeout 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 timeout 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 timeout implementation must look for both responses, regardless of which links they are returned on.

2.4.4 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.

2.4.5 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.

2.4.5.1 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 timeout 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 timeout counters for the affected packet and all subsequently transmitted data packets. This prevents the generation of spurious timeout 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.

2.4.5.1.1 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 2.4.4. 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 an undefined ackID value back to the sender. The sender then initiates recovery as described in Section 2.4.5.1.2 for a packet acknowledge error.

2.4.5.1.2 Control Symbol Errors

There are three types of detectable control symbol errors: a packet acknowledge error, a corrupt control symbol error, and an uncorrupted protocol violating control symbol.

A packet acknowledge error is a packet-accepted or packet-retry control symbol which has an unexpected ackID value or an unexpected packet-not-accepted control symbol. This error shall cause 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 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 shall then back up to the appropriate unaccepted packet and begin re-transmission.

The following is an example of a packet acknowledge error and recovery from that error. 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

A corrupt control symbol is detected as a mismatch between the true and complement 16-bit halves of the aligned control symbol. A corrupt control symbol shall cause the receiver to enter the “Input Error-stopped” state and send a packet-not-accepted control symbol. This informs the sending device that a transmission error has occurred and the sender shall enter the recovery process described above, sending a restart-from-error control symbol.

An uncorrupted protocol violating control symbol is a control symbol that is received unexpectedly. Some examples of this type of error are:

- an unsolicited packet-accepted or packet-retry control symbol
- a restart-from-retry control symbol received while in the Input OK state,
- a 2nd link-request received before returning a link-response, or
- a packet-accepted received before packet transmission has completed

Such errors are an indication of either an otherwise undetectable multi-bit error on the link or a blatant protocol violation by the sender. Such errors may cause unpredictable behavior and the link may not be recoverable. Upon detecting such an error, the receiver shall enter Input Error-stopped state and/or Output Error-stopped state if attempting to recover from these types of errors.

2.4.5.1.3 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.

2.4.5.1.4 Timeout Error

A link timeout on an acknowledge control symbol for a packet is treated like an acknowledge control symbol with an unexpected ackID value.

2.4.6 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 2-7. 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 0s 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. A corrupt pad may or may not cause a CRC error to be detected, depending upon the implementation.

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 2-9 is an example of a naturally 32-bit aligned packet of less than or equal to 80 bytes.

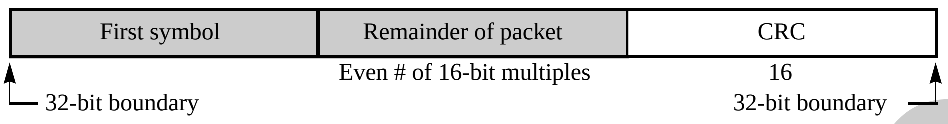


Figure 2-9. Naturally Aligned Packet Bit Stream Example 1

Figure 2-10 is an example of a naturally 32-bit aligned packet of greater than 80 bytes.

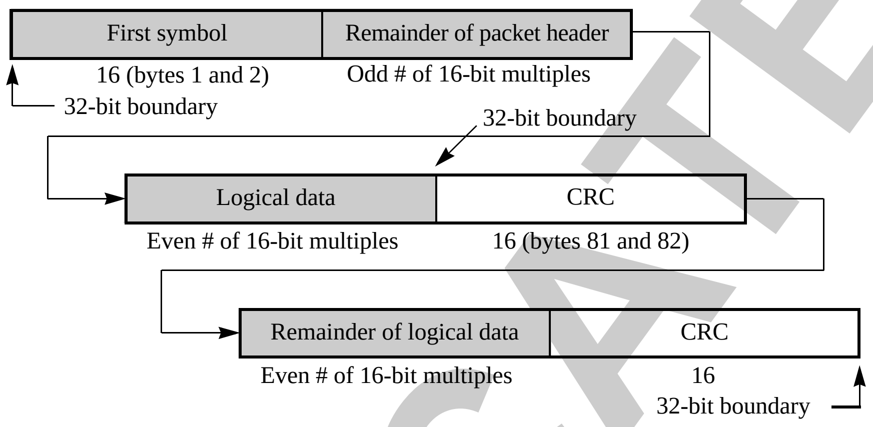

First symbol
Remainder of packet header
16 (bytes 1 and 2)
Odd # of 16-bit multiples
32-bit boundary
32-bit boundary
Logical data
CRC
Even # of 16-bit multiples
16 (bytes 81 and 82)
Remainder of logical data
CRC
Even # of 16-bit multiples
16
32-bit boundary

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

Figure 2-11 is an example of a padded 32-bit aligned packet of less than or equal to 80 bytes.

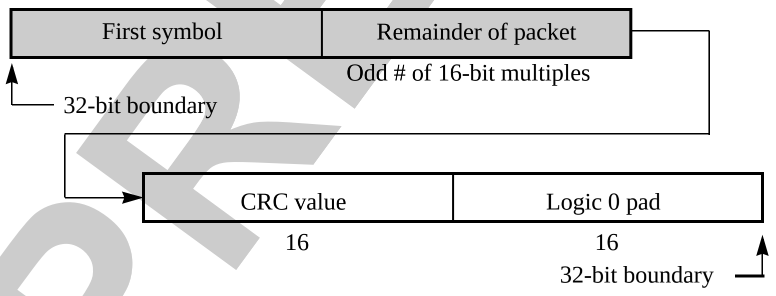


Figure 2-11. Padded Aligned Packet Bit Stream Example 1

Figure 2-12 is an example of a padded 32-bit aligned packet of greater than 80 bytes.

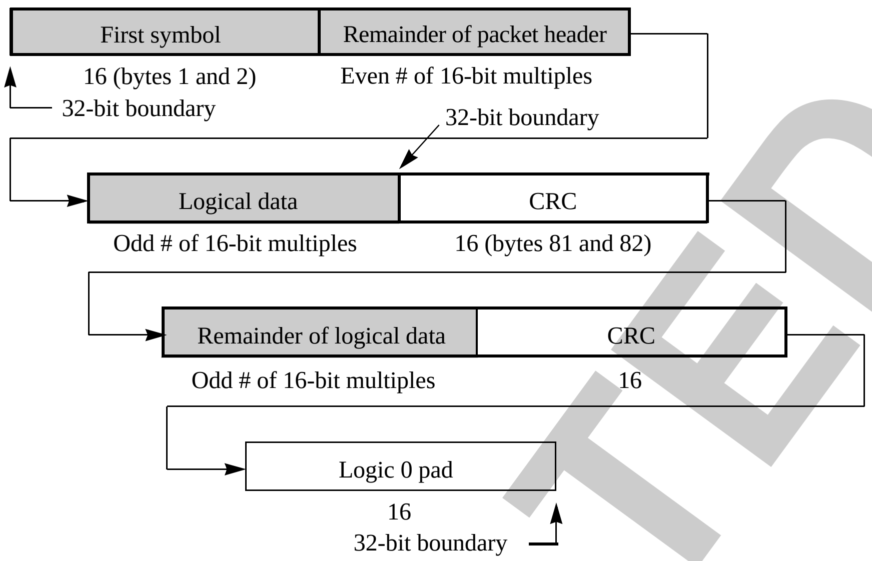


Figure 2-12. Padded Aligned Packet Bit Stream Example 2

2.4.7 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 2-5. Equivalent implementations of other widths can be employed.

Table 2-5. Parallel CRC Intermediate Value Equations

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

Table 2-5. Parallel CRC Intermediate Value Equations (Continued)

Check Bit	e00	e01	e02	e03	e04	e05	e06	e07	e08	e09	e10	e11	e12	e13	e14	e15
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 2-13

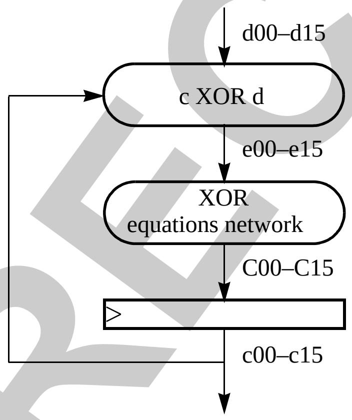


Figure 2-13. CRC Generation Pipeline

2.5 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.

The maximum packet size of 276 bytes is achieved as shown below:

Table 2-6. Maximum Packet Size

Field	Size (bytes)	Layer	Notes
Header	2	Physical, Transport, Logical	
Source ID	2	Transport	
Destination ID	2	Transport	
Trans/wrsize	1	Logical	
srcTID	1	Logical	
Address	8	Logical	Includes Extended_address, Address, Wdptr, and Xambs
Payload	256	Logical	
CRC	4	Physical	Extra two CRC bytes for packets greater than 80 bytes
Total	**276**		

2.6 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 3.7.1, “Port and Link Initialization”), attempt automatic error recovery as described above in Section 2.4.5, “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 *RapidIO Part 1: 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 2.4.5.1 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.

2.6.1 Command Descriptions

Table 2-7 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 2-7. 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

2.6.1.1 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.

When issuing a reset with four consecutive reset commands, care must be taken to account for all effects associated with the reset event. Consult *RapidIO Part 8: Error Management Extensions Specification* for more information.

2.6.1.2 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.

2.6.1.3 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 3.7.1.1, “Sampling Window Alignment.” The send-training command does not generate a link-response control symbol.

2.6.2 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 2-8.

Table 2-8. 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.

Blank page

Chapter 3 Packet and Control Symbol Transmission

3.1 Introduction

This RapidIO chapter defines the *RapidIO Part 4: 8/16 LP-LVDS Physical Layer Specification* 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.

3.2 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 2.4.2, “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 (stype) 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 3-1 and Figure 3-3 for an 8-bit port and Figure 3-2 and Figure 3-4 for a 16-bit port.

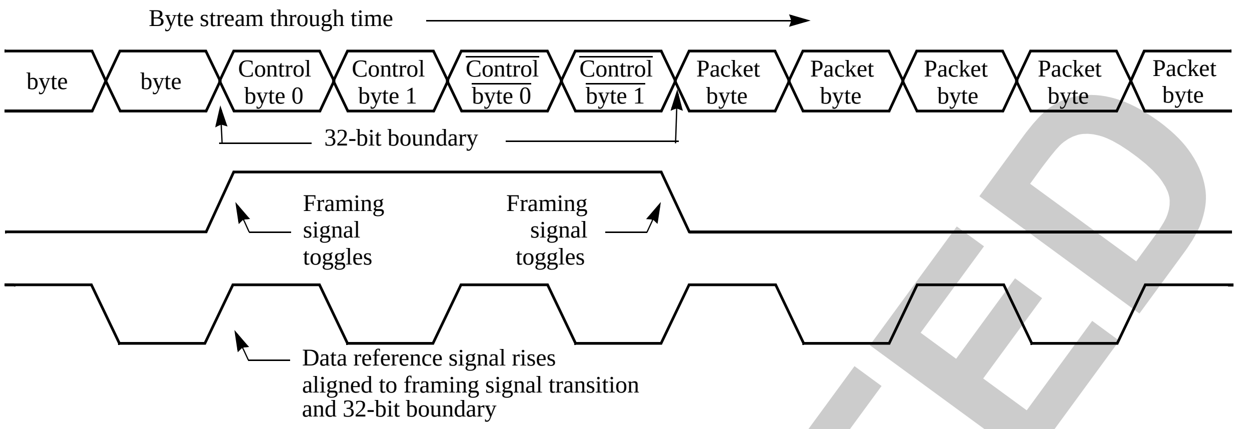


Figure 3-1. Framing Signal Maximum Toggle Rate for 8-bit Port

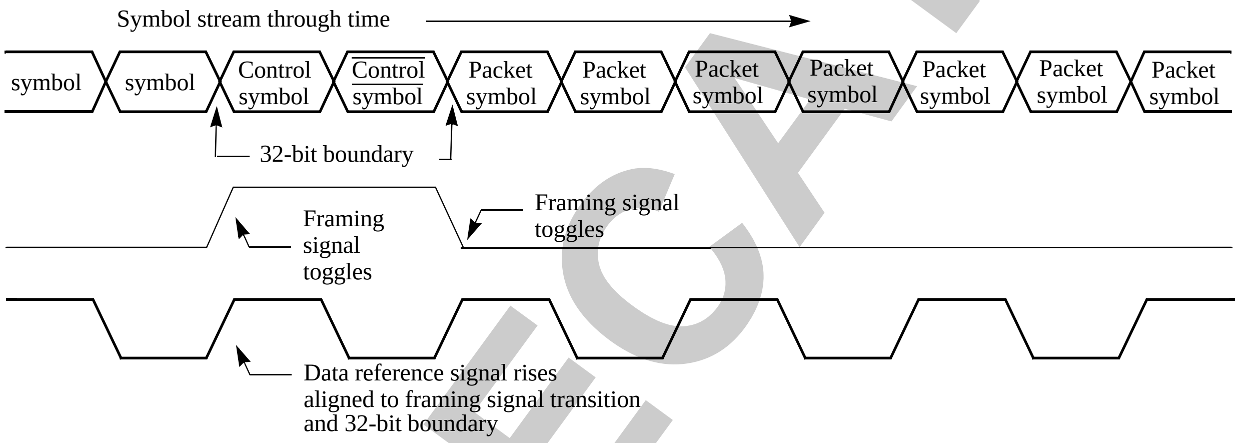


Figure 3-2. Framing Signal Maximum Toggle Rate for 16-bit Port

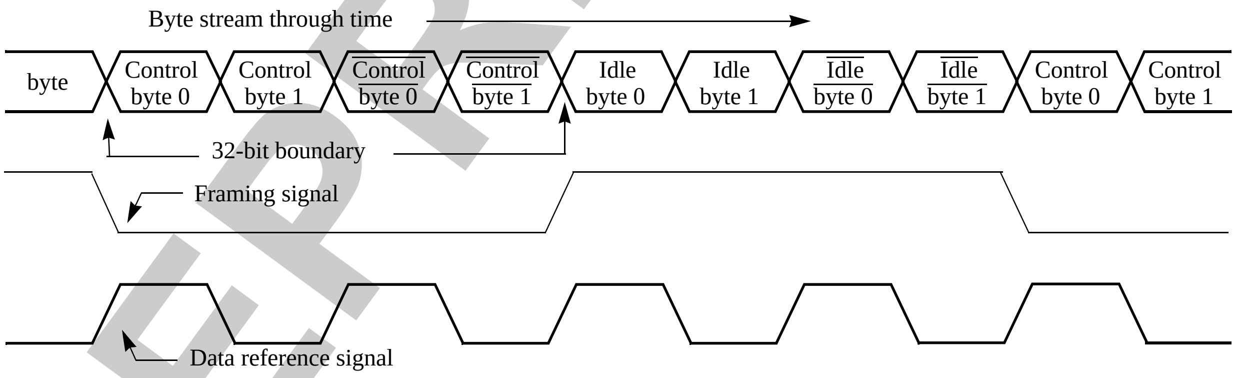


Figure 3-3. Control Symbol Delineation Example for 8-bit Port

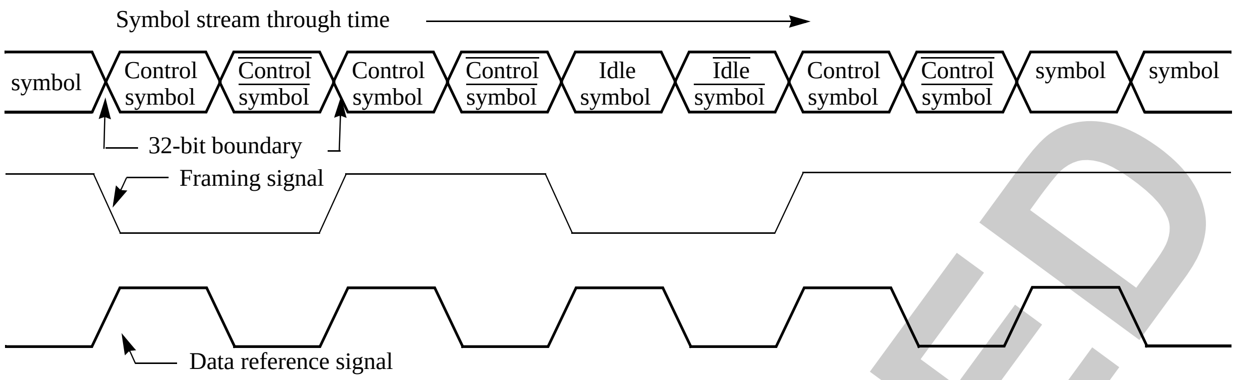


Figure 3-4. Control Symbol Delineation Example for 16-bit Port

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 2.4.5.1, “Recoverable Errors” and Section A.4, “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 3-1 through Figure 3-4.

3.3 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 2.4, “Error Detection and

Recovery."

A port receiving a canceled packet shall drop the packet. The cancelation of a packet shall not result in the generation of any errors. If the packet was canceled because the sender received a packet-not-accepted control symbol, the error that caused the packet-not-accepted to be sent shall be reported in the normal manner.

If a port receiving a canceled packet has not previously acknowledged the packet and is not in an "Input Stopped" stopped state (Retry-Stopped or Error-Stopped), the port shall immediately enter the Input Retry-stopped state and follow the Packet Retry mechanism specified in Section 2.3.3, "Transaction and Packet Delivery", if the packet was canceled with a control symbol other than a restart-from-retry or a link-request/input-status control symbol. As part of the Packet Retry mechanism, the port sends a packet-retry control symbol to the sending port indicating that the canceled packet was not accepted.

Figure 3-5 is an example of a new packet marking the end of a packet.

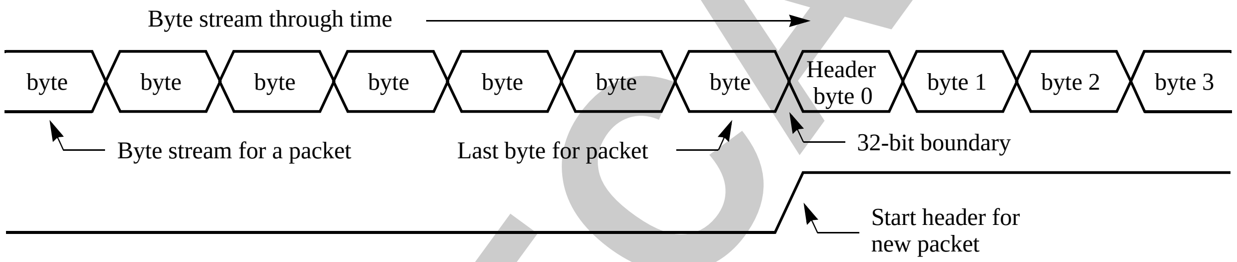


Figure 3-5. Header Marked End of Packet (8-bit Port)

Figure 3-6 is an example of an aligned end-of-packet control symbol marking the end of a packet. The stomp, link-request, and restart-from-retry control symbol cases look similar.

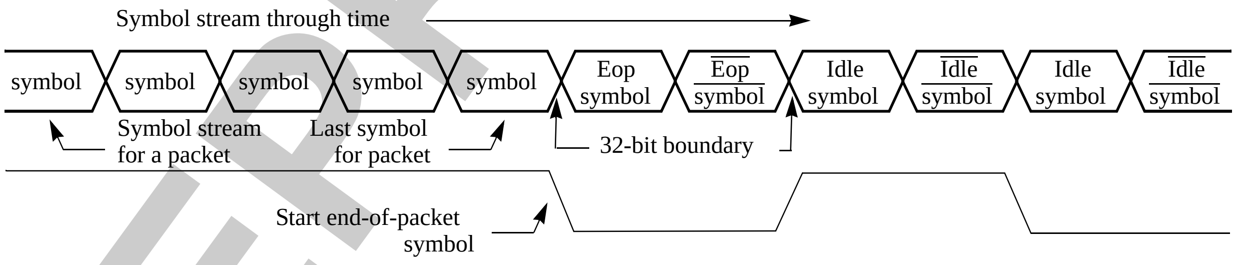

Symbol stream through time
symbol
symbol
symbol
symbol
symbol
Eop symbol
$\overline{\text{Eop}}$ symbol
Idle symbol
$\overline{\text{Idle}}$ symbol
Idle symbol
$\overline{\text{Idle}}$ symbol
Symbol stream for a packet
Last symbol for packet
32-bit boundary
Start end-of-packet symbol

Figure 3-6. End-Of-Packet Control Symbol Marked End of Packet (16-bit Port)

3.4 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 3-7 shows an example of packet pacing. These idle control symbols are ignored by the receiving device, and more data is sent when it becomes available. Pacing idle control symbols can be embedded anywhere in a packet where they can be legally delineated.

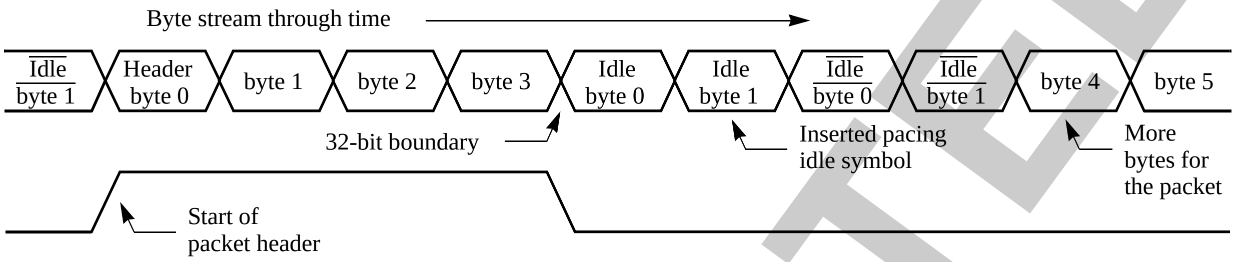

Byte stream through time
$\overline{\text{Idle}}$ byte 1
Header byte 0
byte 1
byte 2
byte 3
Idle byte 0
Idle byte 1
$\overline{\text{Idle}}$ byte 0
$\overline{\text{Idle}}$ byte 1
byte 4
byte 5
32-bit boundary
Inserted pacing idle symbol
More bytes for the packet
Start of packet header

Figure 3-7. 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 6, “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.

3.5 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.

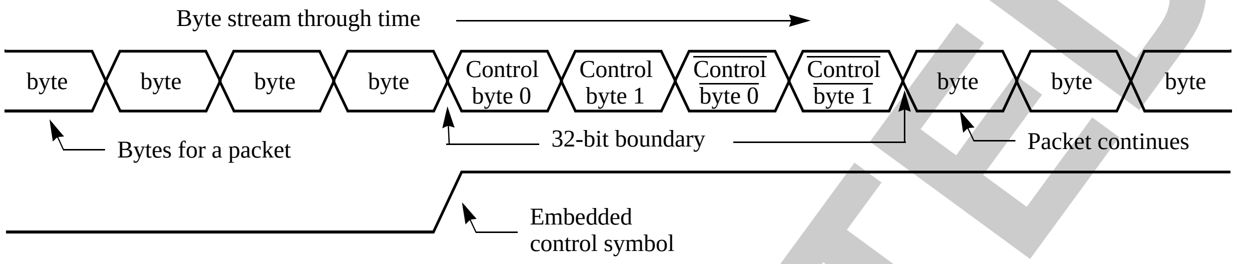


Figure 3-8. Embedded Control Symbols for 8-bit Port

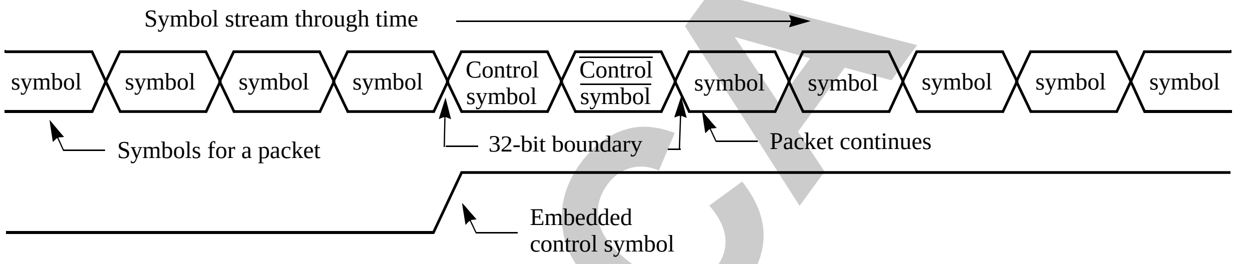


Figure 3-9. Embedded Control Symbols for 16-bit Port

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.

3.6 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 4.7, “Control Symbol to Port Alignment.”

Figure 3-10 shows the byte transmission ordering on a port through time using a small transport format ftype 2 packet from the *RapidIO Part 1: Input/Output Logical Specification* and *RapidIO Part 3: 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.

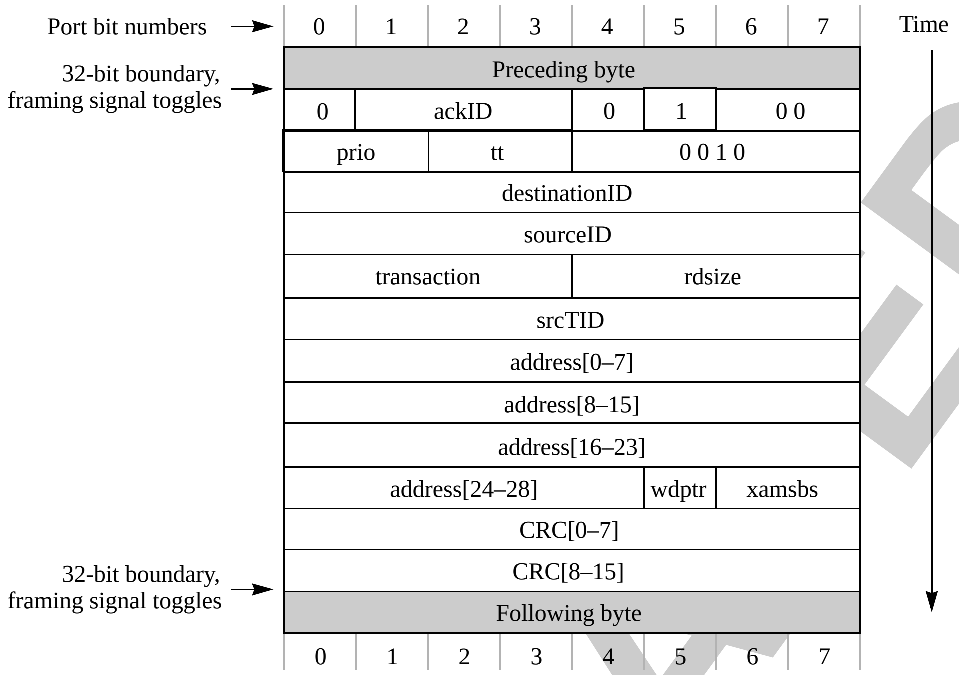


Figure 3-10. Request Packet Transmission Example 1

Figure 3-11 shows the same packet transmitted over a 16-bit port.

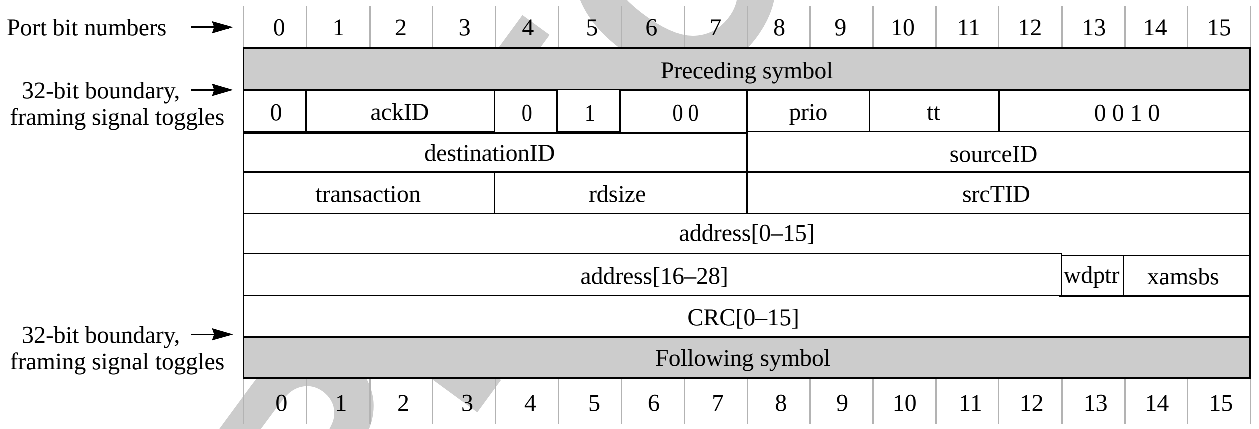


Figure 3-11. Request Packet Transmission Example 2

Figure 3-12 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.

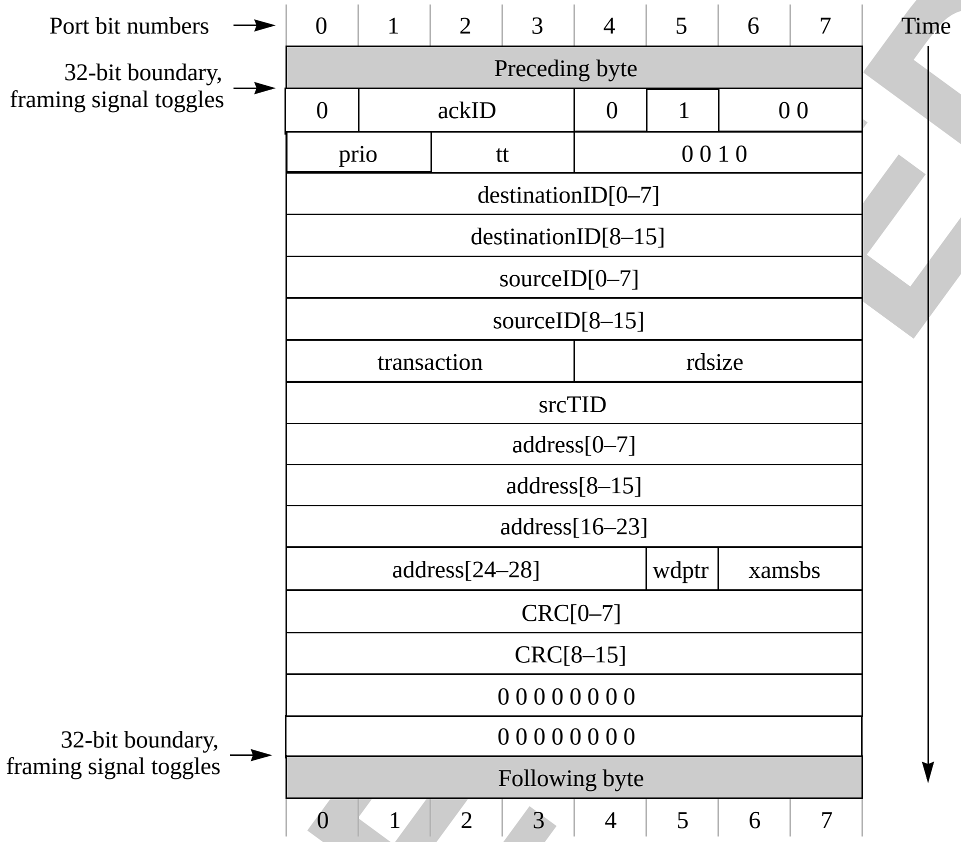


Figure 3-12. Request Packet Transmission Example 3

Figure 3-13 is the same packet as for Figure 3-12 but over the 16-bit port.

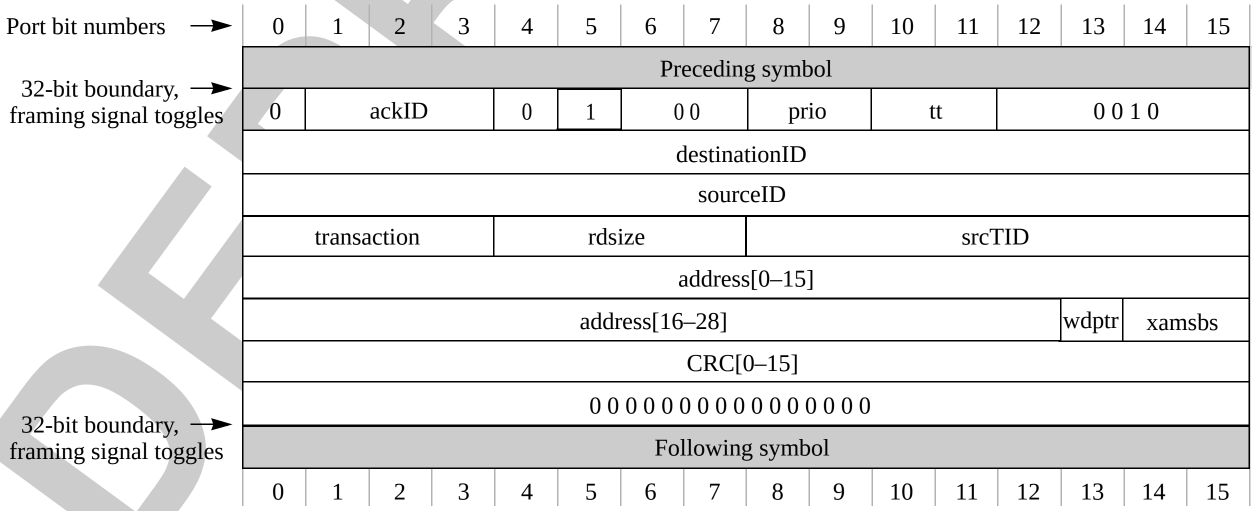


Figure 3-13. Request Packet Transmission Example 4

Figure 3-14 and Figure 3-15 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.

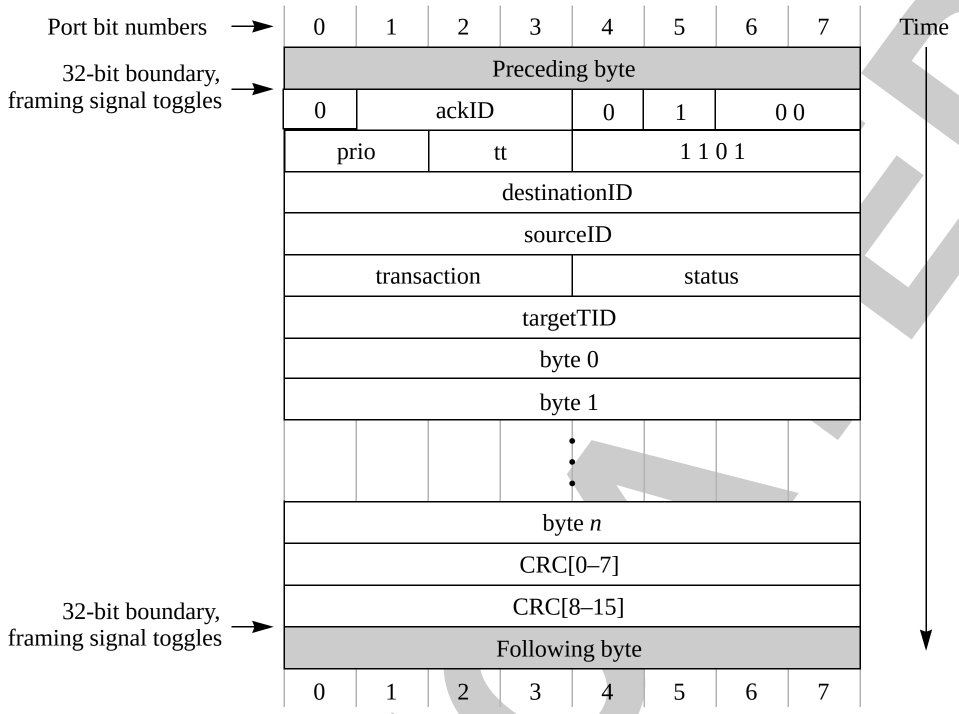


Figure 3-14. Response Packet Transmission Example 1

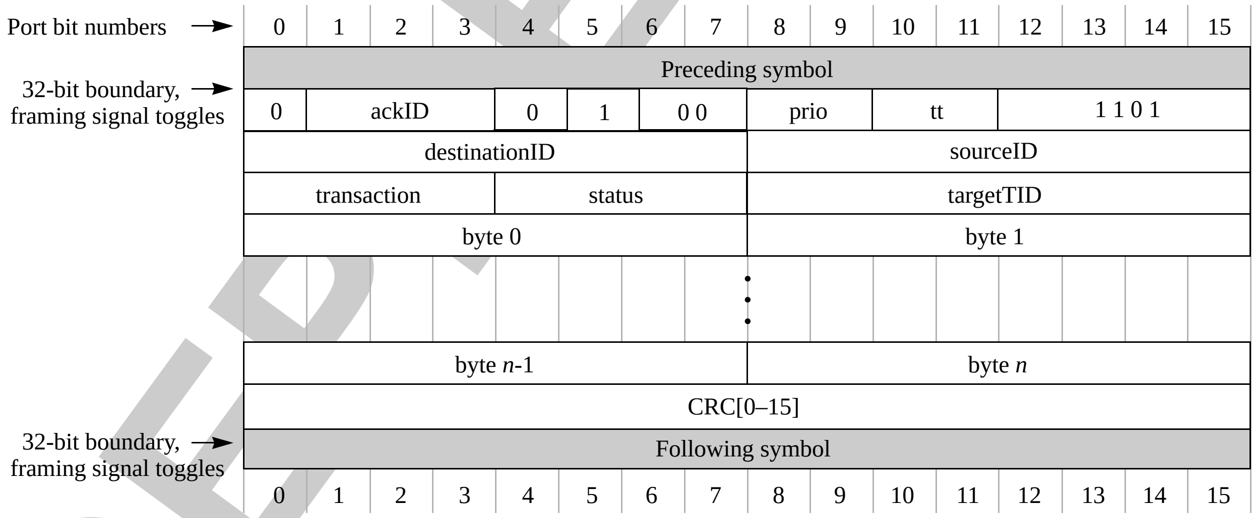


Figure 3-15. Response Packet Transmission Example 2

3.7 System Maintenance

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

3.7.1 Port and Link Initialization

Before an LP-LVDS port can exchange packets with the port to which it is connected, the port may require initialization and the link connecting the two ports must be initialized.

Many ports, especially those operating at higher data rates, must adjust the timing of when they sample input data from the link in order to achieve an acceptable or optimal received bit error rate. This process is called training and is executed while the port is receiving a special training pattern. In addition, each 16-bit wide port must decide whether to operate in 8-bit or 16-bit unless it has been statically configured for 8-bit or 16-mode.

Link initialization requires that each port must receive at least one idle control symbol from the connected port before beginning normal operation.

The state of a port is indicated by bits 30 (Port OK) and 31 (Port Uninitialized) of the associated Port *n* Error and Status Register. Bit 31 is set when the port is in the Uninitialized state and cleared when the port initialization process is completed and the port is initialized. Bit 30 is set when the port has received an idle control symbol from the connected port and the port is in the normal operation mode and is cleared if the port or the connected port is uninitialized or if there is no connected port (the port is not receiving valid input signals).

3.7.1.1 Sampling Window Alignment

Any port whose link receiver input sample timing is not fixed and any 16-bit port whose width mode is not statically configured require port initialization.

3.7.1.1.1 Port Width Mode Selection

All 16-bit LP-LVDS ports shall be capable of operating in both 8-bit and 16-bit modes.The width mode in which a 16-bit port operates may be either statically or dynamically configured. If the width mode of a port is statically configured, the port will operate correctly only if connected to a port operating in the same width mode. If the width mode of a 16-bit port is not statically configured, the width mode is determined as part of the port initialization process.

When operating in 8-bit mode, only the signal pairs CLK0/$\overline{\text{CLK0}}$, FRAME/$\overline{\text{FRAME}}$ and D[0-7]/$\overline{\text{D[0-7]}}$ shall be used. The 16-bit mode output signal pairs TCLK1/$\overline{\text{TCLK1}}$, and TD[8-15]/$\overline{\text{TD[8-15]}}$ may be driven as outputs, but the input signal pairs RCLK1/$\overline{\text{RCLK1}}$ and RD[8-15]/$\overline{\text{RD[8-16]}}$ shall be ignored as inputs.

Dynamic port width selection shall be based on the presence of valid signals at the inputs of the CLK0, CLK1, FRAME D[0-7] and D[8-15] receivers. If valid signals are present at the inputs of the CLK0, CLK1, FRAME D[0-7] and D[8-15] receivers, the port shall operate in 16-bit mode. If valid signals are present at the inputs of the CLK0, FRAME and D[0-7] receivers, but not at the inputs of the CLK1 and D[8-15] receivers, the port shall operate in 8-bit mode. If valid signals are not present at the inputs of the CLK0, FRAME and D[0-7] receivers, the width mode is undefined and the port shall not exit Uninitialized state.

3.7.1.1.2 Input Sampling Window Alignment

Input sampling window alignment is the process in which a port adjusts the timing of when input data is sample by the link receiver. The timing is adjusted to achieve an acceptable or optimal received bit error rate. The process is also called “training”. When the process is successfully completed, the port is said to be “aligned” or “trained”. The process or algorithm used by a port to align the input sampling window is not specified.

Sampling window alignment is done while the port is receiving a special data pattern called the training pattern. A special data pattern is required to ensure that enough transition timing information is available to the receiver to correctly adjust the input sample timing and to ensure that bytes transmitted by a port operating in 8-bit or that half-words transmitted by a port operating in 16-bit are correctly recovered by the link receiver.

There are two types of training, initialization training and maintenance training. Initialization training is used when a device powers up or is reset or when a port loses input sampling window alignment due to events such as excessive system noise or power fluctuations. Maintenance training is used when a port is nominally input sampling window aligned, but in need of some “fine-tuning” of the input sampling window timing to maintain an acceptable or optimum received bit error rate.

3.7.1.1.3 Training Pattern

The training pattern shall be the bit sequence 0b11110000. The training pattern shall be transmitted left to right with the left most bit transmitted first and the right most bit transmitted last.

When transmitted, the training pattern shall be transmitted simultaneously on all of the data signals, D[0-7] for an 8-bit port or a 16-bit port statically configured to operate in 8-bit mode, D[0-15] for a 16-port not statically configured to operate in 8-bit mode, and the training pattern or is complement shall be transmitted on the FRAME signal. The training pattern or its complement is selected for transmission on the FRAME signal such that the FRAME signal shall toggle at the beginning of training pattern transmission. The training pattern shall never be transmitted on a CLK signal. The training pattern shall never be transmitted on signals D[8-15] of a 16-bit port if the port is statically configured to operate in 8-bit mode.

The result of these rules is that during training pattern transmission, FRAME and data signals transmitted by the port have the following properties.

- The FRAME signal toggles at the beginning of training pattern transmission. (Individual data bits may or may not toggle at the beginning of training pattern transmission depending on their value during the bit time immediately preceding the training pattern.)
- After the first bit of the training pattern, FRAME and all data bits all toggle at the same nominal time.
- Each byte transmitted by a port transmitting in 8-bit mode is either all ones, 0xFF, or all zeros, 0x00.
- Each half-word transmitted by a port transmitting in 16-bit mode is either all ones, 0xFFFF, or all zeros, 0x0000.

The reception of the training pattern by an initialized port is readily identified by looking at RD[0-7] when FRAME toggles. If the received value of RD[0-7] is either all ones, 0xFF, or all zeros, 0x00, the training pattern is being received.

3.7.1.1.4 Training Pattern Transmission

When transmitted, the training pattern shall be transmitted in bursts. Each burst shall contain 256 repetitions of the training pattern. Each burst shall be followed by either a link-request/send-training or an idle control symbol.

The training pattern shall be transmitted by an initialized port only at request of the connected port. The link-request/send-training control symbol is used to request that the connected port transmit the training pattern. A port that is not initialized and therefore unable to reliably receive control symbols assumes that the connected port is sending link-request/send-training control symbols and therefore continuously transmits training sequence bursts with each burst followed by a link-request/send-training control symbol as specified by the Port Initialization Process.

The training pattern shall neither be embedded in a packet nor used to terminate a packet.

3.7.1.1.5 Ports Not Requiring Port Initialization

Similarly, a 16-bit port with fixed input sampling window timing and whose width mode is statically configured does not require port initialization. On device power-up and on device reset, such ports shall enter and remain in the Initialized state and shall never be in the Uninitialized state. Such ports shall transmit idle control symbols until an idle control symbol is received from the connected port. Upon the reception of an idle control symbol from the connected port, the port shall transmit an idle control symbol, set the “port OK” bit in its Port *n* Control and Status Register, enter the normal operation state and may then begin the transmission of packets and non-idle control symbols.

If while waiting to receive an idle control symbol from the connected port, the reception by the port of a link-request/send-training control symbol from the connected port immediately followed by the training pattern indicates that the connected port in not initialized. When this occurs, the port shall stop sending idle control symbols and repeatedly send training pattern bursts, each burst followed by an idle control symbol, until an idle control symbol is received from the connected port indicating that the connected port is now initialized. Upon receiving an idle control symbol from the connected port, the port shall complete transmission of the current training pattern burst, transmit an idle control symbol, set the “port OK” bit in its Port *n* Control and Status Register and enter the normal operation state. The port may then transmit packets and non-idle control symbols.

3.7.1.1.6 Ports Requiring Port Initialization

Ports that do not have fixed input sampling window timing and 16-bit ports whose width mode is not statically configured require port initialization. Such ports shall enter the Uninitialized state on device power-up and device reset. Such a port shall also enter the Uninitialized state if the port loses correct input sampling window timing due to events such as excessive system noise or power fluctuations. The algorithm used to determine when a port has lost input sample window alignment is not specified. A port in the Uninitialized state shall execute the Port Initialization Process to exit the Uninitialized state.

The output signals of a LP-LVDS port may be erratic when the device containing the port is powering up or being reset. For example, the output drivers may be temporarily disabled, the signals may have erratic HIGH or LOW times and/or the clock signals may stop toggling. A LP-LVDS port must be tolerant of such behavior and shall properly initialize after the signals from the connected port return to normal and comply with the LP-LVDS electrical specifications.

3.7.1.1.7 Port Initialization Process

Upon entering the Uninitialized state, a port shall execute the following Port Initialization Process.

- The port sets the “Port Uninitialized” bit and clears the “Port OK” bit in its Port *n* Control and Status Register.
- The port transmits a link-request/send-training control symbol followed by one or more bursts of the training sequence. The port continuously transmits training pattern bursts, each followed by a link-request/send-training or idle control symbol, until the port has achieved input sample timing alignment and has received an idle control symbol from the connected port.
- The port attempts to detect a valid clock signal on its CLK0 input and, if present, on its CLK1 input and to detect the training pattern on its FRAME and D[0-7] inputs and, if present, on its D[8-15] inputs.
- Once valid input signals are detected, a 16-bit ports whose width mode is not statically configured determines the width of the connected port and selects the matching width mode.

- Once the width mode of the port is established, either statically or dynamically, the port attempts to achieve input sampling window timing alignment. While attempting to achieve input sampling window timing alignment, the port shall transmit a link-request/send-training control symbol after each training pattern burst.
- When the port achieves input sampling window timing alignment, it clears the "Port Uninitialized" bit in the Port *n* Control and Status Register and transmits an idle control symbol after each training pattern burst instead of a link-request/send-training control symbol. This indicates to the connected port that the port has completed input sampling window alignment.
- Upon receiving an idle control symbol from the connected port, indicating that the connected port has completed input sampling window alignment, the port completes transmitting the current training pattern burst, sends an idle control symbol, sets the "Port OK" bit in the Port *n* Control and Status Register and enters normal operation.

3.7.1.2 Link Initialization

After a port is in the Initialized state, the port shall not begin transmission of packets and control symbols other than the idle control until it has received an idle control symbol from the connected port. The reception of an idle control symbol indicates that the connected port is in the Initialized state and is ready to receive packets and non-idle control symbols. When both ports connected by a link have received an idle control symbol from the connected port, the link is initialized.

3.7.1.3 Maintenance Training

Depending upon their implementation, some ports may require occasional adjustment of their input sampling window timing while in the Initialized state to maintain an optimal received bit error rate. Such adjustment is called maintenance training. A port requiring maintenance training shall do the following.

- The port shall transmit a single link-request/send-training control symbol and then resume normal transmit operation.
- If the port is not able to complete maintenance training with one burst of the training pattern, the port may transmit additional link-request/send-training control symbols and shall resume normal transmit operation after transmitting each link-request/send-training control symbol.

A port requiring maintenance training shall not transmit the training pattern after transmitting a link-request/send-training control symbol. (The transmission by a port of a link-request/send-training control symbol followed by the training pattern indicates that the port has become uninitialized.)

A port receiving a link-request/send-training control symbol that is not followed by the training pattern shall end the transmission of packets and control symbols as quickly as possible without violating the link protocol, transmit one burst of the training pattern followed by an idle control symbol and then resume normal operation.

3.7.1.4 Unexpected Training Pattern Reception

At any time, the reception by an initialized port of unsolicited training pattern, whether or not preceded by a link-request/sent-training control symbol, indicates that the connected port is in the Uninitialized state. When this occurs, the port receiving the unsolicited training pattern shall repeatedly transmit training pattern bursts, each burst followed by an idle control symbol, until an idle control symbol is received from the connected port indicating that the connected port is now initialized. Upon receiving an idle control symbol from the connected port, the port shall complete transmission of the current training pattern burst, transmit an idle control symbol, set the “port OK” bit in its Port *n* Control and Status Register and enter the normal operation state. The port may then transmit packets and non-idle control symbols.

Once a link has been initialized, the reception of unsolicited training pattern is a protocol violation. It indicates that the sending port has lost input sampling window alignment and has most likely not received some previously sent packets and control symbols. Once the link has been initialized, a port receiving an unsolicited training pattern shall enter the output Error-stopped state. The port shall execute the Output Error-stopped recovery process specified in Section 2.4.5.1.2, “Control Symbol Errors” once communication with the connected port has been re-established.

3.7.2 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 4.3 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).

3.8 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.

Chapter 4 Control Symbol Formats

4.1 Introduction

This chapter defines the *RapidIO Part 4: 8/16 LP-LVDS Physical Layer Specification* control symbols described in Chapter 2, “Physical Layer Protocol.” Note that the S bit defined in Section 2.3.1 is always set to logic 1 and the $\overline{S}$ bit (also defined in Section 2.3.1) 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. For forward compatibility, control symbols received by a port with a reserved field encoding shall be ignored and not cause an error to be reported.

4.2 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 3.5, “Embedded Control Symbols.”

Field definitions for the acknowledgment control symbols are shown in Table 4-1.

Table 4-1. 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 2.3.1
cause	cause field indicates the type of error encountered by an input port, defined in Table 4-2

4.2.1 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 4-1.

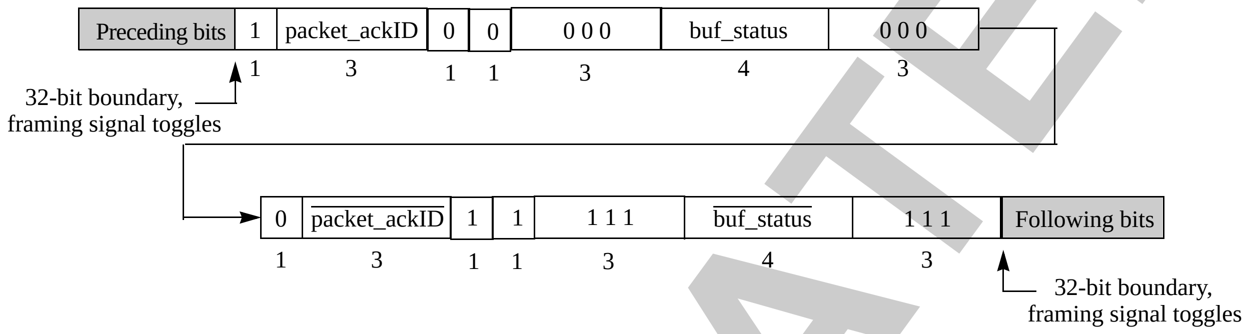


Figure 4-1. Type 0 Packet-Accepted Control Symbol Format

4.2.2 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 4-2.

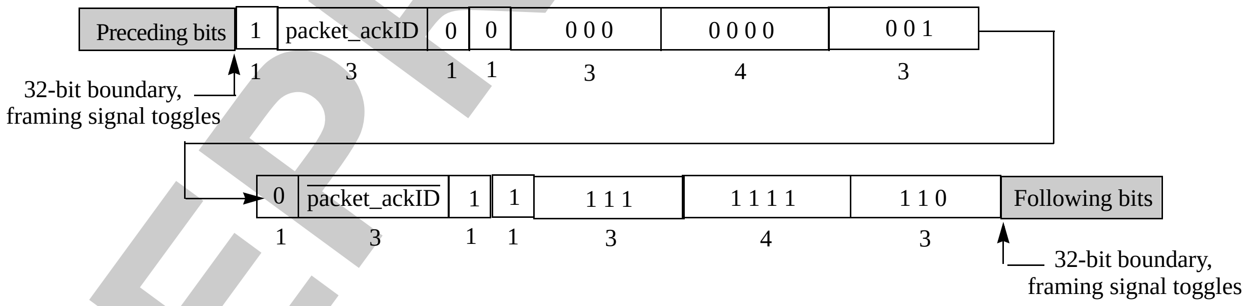


Figure 4-2. Type 1 Packet-Retry Control Symbol Format

4.2.3 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 4-2. 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 4-3.

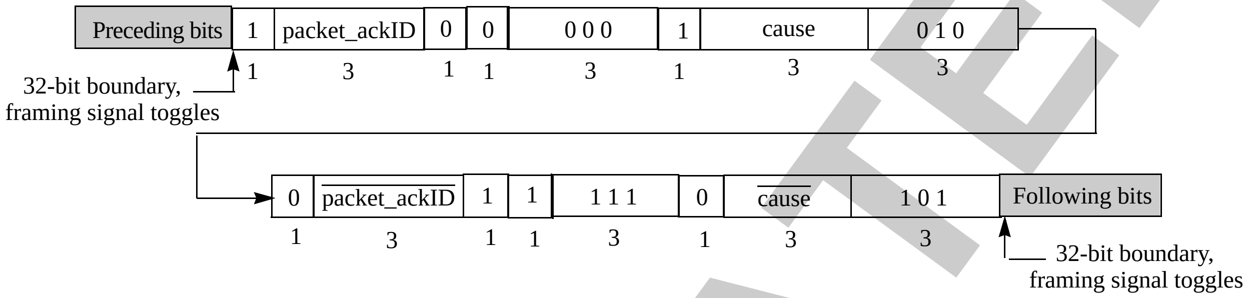


Figure 4-3. Type 2 Packet-Not-Accepted Control Symbol Format

The cause field shall be used to display informational fields useful for debug. Table 4-2 displays the reasons a packet may not be accepted, indicated by the cause field.

Table 4-2. 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

4.2.4 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 3, “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.

4.3 Packet Control Symbol Formats

Packet control symbols are used for packet delineation, transmission, pacing, and other link interface control functions as described in Chapter 3, “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 4-4.

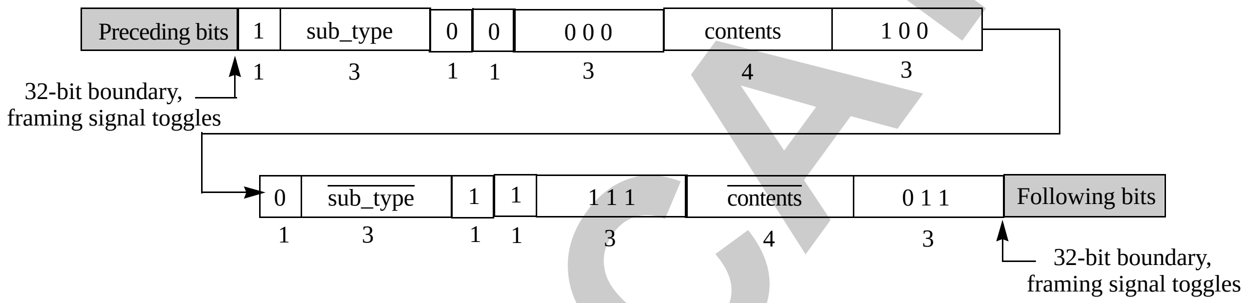


Figure 4-4. Type 4 Packet Control Symbol Format

Table 4-3 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 2.3.1, whose encodings are specified in Table 2-2. 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 4-4. For the stomp and restart-from-retry control symbols, the contents field is unused and shall be tied to all logic 0s and ignored by the receiving device.

Table 4-3. 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 2.3.1, encodings are defined in Table 2-2.
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 2.3.1, encodings are defined in Table 2-2.
restart-from-retry	0b011	Unused, contents=0b0000

Table 4-3. 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 4-4.
Multicast-event	0b101	Used as a buf_status field that indicates the number of maximally sized packets that can be received. Described in Section 2.3.1, encodings are defined in Table 2-2.
Reserved	0b110-111	

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

Table 4-4. 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

4.4 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 2.6. 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 always cancels a packet whose transmission is in progress and can also be sent between packets. Under error conditions a link-request/input-status control symbol acts as a restart-from-error control symbol as described in Section 2.4.5.1, “Recoverable Errors.” This control symbol format is displayed in Figure 4-5.

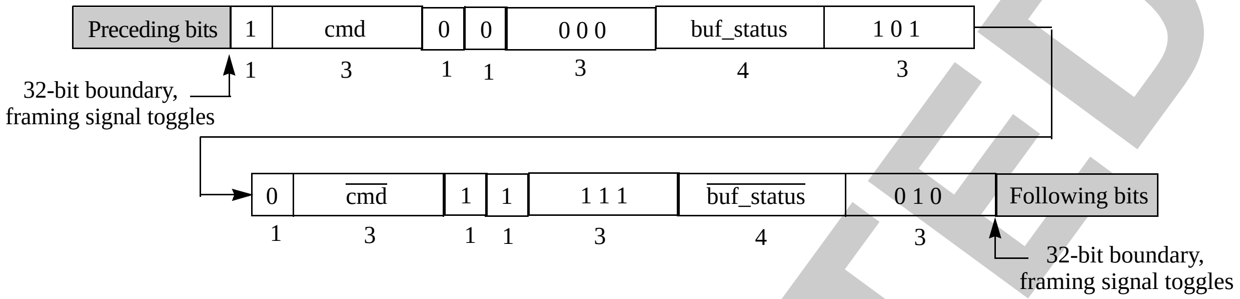

Preceding bits
1
cmd
0
0
0 0 0
buf_status
1 0 1
1
3
1
1
3
4
3
32-bit boundary, framing signal toggles
0
$\overline{\text{cmd}}$
1
1
1 1 1
$\overline{\text{buf\_status}}$
0 1 0
Following bits
1
3
1
1
3
4
3
32-bit boundary, framing signal toggles

Figure 4-5. Type 5 Link-Request Control Symbol Format

The cmd, or command, field of the link-request control symbol format is defined in Table 4-5.

Table 4-5. cmd Field Definition

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

• The link-response control symbol is used by a device to respond to a link-request control symbol as described in the link maintenance protocol described in Section 2.6. 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 4-6.

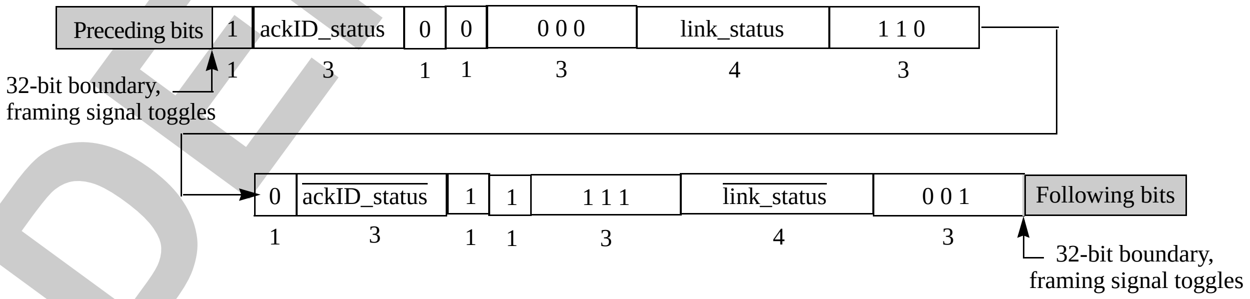

Preceding bits
1
ackID_status
0
0
0 0 0
link_status
1 1 0
1
3
1
1
3
4
3
32-bit boundary, framing signal toggles
0
$\overline{\text{ackID\_status}}$
1
1
1 1 1
$\overline{\text{link\_status}}$
0 0 1
Following bits
1
3
1
1
3
4
3
32-bit boundary, framing signal toggles

Figure 4-6. Type 6 Link-Response Control Symbol Format

The ackID_status field of the link-response format is defined in Table 4-6.

Table 4-6. 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 4-7. Note that the ackID information is included in both fields for additional error coverage if the receiver is working properly (encodings 8-15).

Table 4-7. 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.

4.5 Reserved Symbol Formats

The control symbol corresponding to stype 0b011 is reserved.

4.6 Implementation-defined Symbol Formats

The control symbol corresponding to stype 0b111 is implementation defined. In general, implementation-defined control symbols will result in inter-operability problems with devices that are not designed to handle them. Inter-operability problems can include undefined and/or inconsistant behavior, data corruption, or system failure.

4.7 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 3.6, “Packet to Port Alignment.”

Figure 4-7 shows the byte transmission ordering on an 8-bit port through time using an aligned packet-accepted control symbol as an example.

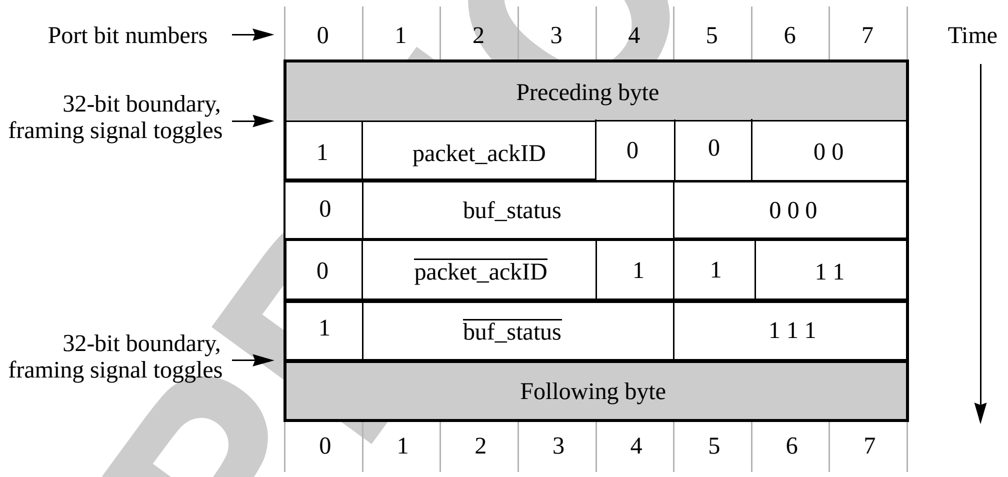

Port bit numbers
0
1
2
3
4
5
6
7
Time
32-bit boundary, framing signal toggles
Preceding byte
1
packet_ackID
0
0
0 0
0
buf_status
0 0 0
0
$\overline{\text{packet\_ackID}}$
1
1
1 1
32-bit boundary, framing signal toggles
1
$\overline{\text{buf\_status}}$
1 1 1
Following byte

Figure 4-7. Control Symbol Transmission Example 1

Figure 4-8 shows the same control symbol over the 16-bit interface.

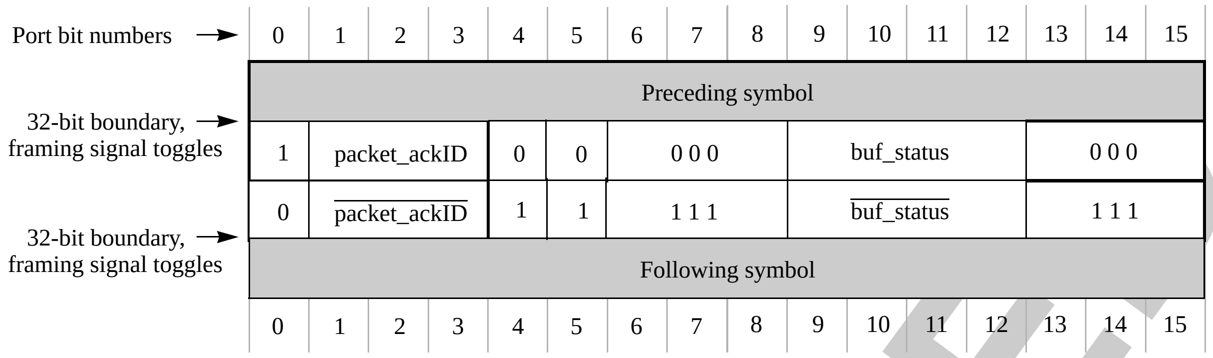

Port bit numbers
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
32-bit boundary, framing signal toggles
32-bit boundary, framing signal toggles
Preceding symbol
1
packet_ackID
0
0
0 0 0
buf_status
0 0 0
0
$\overline{\text{packet\_ackID}}$
1
1
1 1 1
$\overline{\text{buf\_status}}$
1 1 1
Following symbol

Figure 4-8. Control Symbol Transmission Example 2

Blank page

Chapter 5 8/16 LP-LVDS Registers

5.1 Introduction

This chapter describes the *RapidIO Part 4: 8/16 LP-LVDS Physical Layer Specification* visible register 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.

There are four types of 8/16 LP-LVDS devices, an end point device, an end point device with additional software recovery registers, an end point free (or switch) device, and an end point free device with additional software recovery registers. Each has a different set of CSRs, specified in Section 5.5, Section 5.6, Section 5.7, and Section 5.8, respectively. All four device types have the same CARs, specified in Section 5.4.

5.2 Register Map

These registers utilize the Extended Features blocks and can be accessed using *RapidIO Part 1: 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.

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

Table 5-1. 8/16 LP-LVDS Register Map

Configuration Space Byte Offset	Register Name
0x0-C	Reserved
0x10	Processing Element Features CAR
0x14–FC	Reserved
0x100–FFFC	Extended Features Space
0x10000–FFFFFC	Implementation-defined Space

5.3 Reserved Register, Bit and Bit Field Value Behavior

Table 5-2 describes the required behavior for accesses to reserved register bits and reserved registers for the RapidIO register space,

Table 5-2. Configuration Space Reserved Access Behavior

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

Table 5-2. 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.

When a writable bit field is set to a reserved value, device behavior is implementation specific.

5.4 Capability Registers (CARs)

Every processing element shall contain a set of registers that allows an external processing element to determine its capabilities using the I/O logical maintenance read operation. All registers are 32 bits wide and are organized and accessed in 32-bit (4 byte) quantities, although some processing elements may optionally allow larger accesses. CARs are read-only. Refer to Table 5-2 for the required behavior for accesses to reserved registers and register bits.

CARs are big-endian with bit 0 the most significant bit.

5.4.1 Processing Element Features CAR (Configuration Space Offset 0x10)

The processing element features CAR identifies the major functionality provided by the processing element. The bit settings are shown in Table 5-3.

Table 5-3. Bit Settings for Processing Element Features CAR

Bits	Name	Description
0–24	—	Reserved
25	Implementation-defined	Implementation-defined
26	CRF Support	PE supports the Critical Request Flow (CRF) indicator 0b0 - Critical Request Flow is not supported 0b1 - Critical Request Flow is supported
27–31	—	Reserved

5.5 Generic End Point Devices

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

5.5.1 Register Map

Table 5-4 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 + 0xBC]. Register map offset [EF_PTR + 0xC0] can be used for another Extended Features block.

Table 5-4. 8/16 LP-LVDS Register Map - Generic End Point Devices

	Block Byte Offset	Register Name
General	0x0	8/16 LP-LVDS Register Block Header
	0x4–1C	Reserved
	0x20	Port Link Timeout Control CSR
	0x24	Port Response Timeout Control CSR
	0x28-38	Reserved
	0x3C	Port General Control CSR
Port 0	0x40-54	Reserved
	0x58	Port 0 Error and Status CSR
	0x5C	Port 0 Control CSR
Port 1	0x60-74	Reserved
	0x78	Port 1 Error and Status CSR
	0x7C	Port 1 Control CSR
Ports 2-14	0x80–218	Assigned to Port 2-14 CSRs
Port 15	0x220-234	Reserved
	0x238	Port 15 Error and Status CSR
	0x23C	Port 15 Control CSR

5.5.2 Command and Status Registers (CSRs)

Refer to Table 5-2 for the required behavior for accesses to reserved registers and register bits.

5.5.2.1 8/16 LP-LVDS Register Block Header (Block Offset 0x0)

The 8/16 LP-LVDS register block header register contains the EF_PTR to the next EF_BLK and the EF_ID that identifies this as the generic end point 8/16 LP-LVDS register block header.

Table 5-5. Bit Settings for 8/16 LP-LVDS Register Block Header

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

5.5.2.2 Port Link Timeout Control CSR (Block Offset 0x20)

The port link timeout control register contains the timeout timer value for all ports on a device. This timeout 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 timeout interval, and represents between 3 and 5 seconds.

Table 5-6. Bit Settings for Port Link Timeout Control CSR

Bit	Name	Reset Value	Description
0–23	timeout_value	All 1s	timeout interval value
24-31	—		Reserved

5.5.2.3 Port Response Timeout Control CSR (Block Offset 0x24)

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

Table 5-7. Bit Settings for Port Response Timeout Control CSR

Bit	Name	Reset Value	Description
0–23	timeout_value	All 1s	timeout interval value
24-31	—		Reserved

5.5.2.4 Port General Control CSR (Block Offset 0x3C)

The bits accessible through the Port General Control CSR are bits that apply to all ports on a device. There is a single copy of each such bit per device. These bits are also accessible through the Port General Control CSR of any other physical layers implemented on a device.

Table 5-8. 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

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

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

Table 5-9. 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).

5.5.2.6 Port *n* Control CSR (Block Offsets 0x5C, 7C, ..., 23C)

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

Table 5-10. 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. Control symbols are not affected and are sent normally. This is the recommended state after device reset. 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. 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. This is the recommended state after device reset. 0b1 - port is enabled to respond to any packet
6	Input Port Receiver Disable	0b0	Input port receiver enable: 0b0 - input port receivers are enabled 0b1 - input port receivers are disabled and are unable to receive to any packets or control symbols
7	—		Reserved
8	Error Checking Disable	0b0	This bit disables all RapidIO transmission error checking 0b0 - Error checking and recovery is enabled 0b1 - Error checking and recovery is disabled Device behavior when error checking and recovery is disabled and an error condition occurs is undefined
9	Multicast-event Participant	see footnote[5]	Send incoming multicast-event control symbols to this port (multiple port devices only)
10-13	—		Reserved
14	Enumeration Boundary	see footnote[6]	An enumeration boundary aware system enumeration algorithm shall honor this flag. The algorithm, on either the ingress or the egress port, shall not enumerate past a port with this bit set. This provides for software enforced enumeration domains within the RapidIO fabric.
15-19	—		Reserved

Table 5-10. Bit Settings for Port *n* Control CSRs (Continued)

Bit	Name	Reset Value	Description
20-27	Implementation-defined		Implementation-defined
28-30	—		Reserved
31	Port Type		This indicates the port type (read only) 0b0 - Parallel port 0b1 - Reserved

[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

[6]The enumeration boundary reset value is implementation dependent. Provision shall be made to allow the reset value of this bit to be configurable on a per system basis if this feature is supported.

5.6 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.

5.6.1 Register Map

Table 5-11 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 + 0xBC]. Register map offset [EF_PTR + 0xC0] can be used for another Extended Features block.

Table 5-11. 8/16 LP-LVDS Register Map - Generic End Point Devices (SW assisted)

	Block Byte Offset	Register Name
General	0x0	8/16 LP-LVDS Register Block Header
	0x4–1C	Reserved
	0x20	Port Link Timeout Control CSR
	0x24	Port Response Timeout Control CSR
	0x28-38	Reserved
	0x3C	Port General Control CSR
Port 0	0x40	Port 0 Link Maintenance Request CSR
	0x44	Port 0 Link Maintenance Response CSR
	0x48	Port 0 Local ackID Status CSR
	0x4C-54	Reserved
	0x58	Port 0 Error and Status CSR
	0x5C	Port 0 Control CSR
Port 1	0x60	Port 1 Link Maintenance Request CSR
	0x64	Port 1 Link Maintenance Response CSR
	0x68	Port 1 Local ackID Status CSR
	0x6C-74	Reserved
	0x78	Port 1 Error and Status CSR
	0x7C	Port 1 Control CSR

Table 5-11. 8/16 LP-LVDS Register Map - Generic End Point Devices (SW assisted)

	Block Byte Offset	Register Name
Ports 2-14	0x80–218	Assigned to Port 2-14 CSRs
Port 15	0x220	Port 15 Link Maintenance Request CSR
	0x224	Port 15 Link Maintenance Response CSR
	0x228	Port 15 Local ackID Status CSR
	0x22C-234	Reserved
	0x238	Port 15 Error and Status CSR
	0x23C	Port 15 Control CSR

5.6.2 Command and Status Registers (CSRs)

Refer to Table 5-2 for the required behavior for accesses to reserved registers and register bits.

5.6.2.1 8/16 LP-LVDS Register Block Header (Block Offset 0x0)

The 8/16 LP-LVDS register block header register contains the EF_PTR to the next EF_BLK and the EF_ID that identifies this as the generic end point 8/16 LP-LVDS register block header.

Table 5-12. Bit Settings for 8/16 LP-LVDS Register Block Header

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

5.6.2.2 Port Link Timeout Control CSR (Block Offset 0x20)

The port link timeout control register contains the timeout timer value for all ports on a device. This timeout 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 timeout interval, and represents between 3 and 5 seconds.

Table 5-13. Bit Settings for Port Link Timeout Control CSR

Bit	Name	Reset Value	Description
0–23	timeout_value	All 1s	timeout interval value
24-31	—		Reserved

5.6.2.3 Port Response Timeout Control CSR (Block Offset 0x24)

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

Table 5-14. Bit Settings for Port Response Timeout Control CSR

Bit	Name	Reset Value	Description
0–23	timeout_value	All 1s	timeout interval value
24-31	—		Reserved

5.6.2.4 Port General Control CSR (Block Offset 0x3C)

The bits accessible through the Port General Control CSR are bits that apply to all ports on a device. There is a single copy of each such bit per device. These bits are also accessible through the Port General Control CSR of any other physical layers implemented on a device.

Table 5-15. 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

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

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 5-16. 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.

5.6.2.6 Port *n* Link Maintenance Response CSRs (Block Offsets 0x44, 64, ..., 224)

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 4.4, “Link Maintenance Control Symbol Formats.” This register is read-only.

Table 5-17. 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

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

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 5-18. Bit Settings for Port *n* Local ackID Status CSRs

Bit	Name	Reset Value	Description
0	Clr_outstanding_ackIDs	0b0	Writing 0b1 to this bit causes all outstanding unacknowledged packets to be discarded. This bit should only be written when trying to recover a failed link. This bit is always logic 0 when read.
1-4	—		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.
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.

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

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

Table 5-19. 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).

5.6.2.9 Port *n* Control CSR (Block Offsets 0x5C, 7C, ..., 23C)

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

Table 5-20. 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. Control symbols are not affected and are sent normally. This is the recommended state after device reset. 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. 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. This is the recommended state after device reset. 0b1 - port is enabled to respond to any packet
6	Input Port Receiver Disable	0b0	Input port receiver enable: 0b0 - input port receivers are enabled 0b1 - input port receivers are disabled and are unable to receive to any packets or control symbols
7	—		Reserved
8	Error Checking Disable	0b0	This bit disables all RapidIO transmission error checking 0b0 - Error checking and recovery is enabled 0b1 - Error checking and recovery is disabled Device behavior when error checking and recovery is disabled and an error condition occurs is undefined
9	Multicast-event Participant	see footnote[5]	Send incoming multicast-event control symbols to this port (multiple port devices only)
10-13	—		Reserved
14	Enumeration Boundary	see footnote[6]	An enumeration boundary aware system enumeration algorithm shall honor this flag. The algorithm, on either the ingress or the egress port, shall not enumerate past a port with this bit set. This provides for software enforced enumeration domains within the RapidIO fabric.
15-19	—		Reserved

Table 5-20. Bit Settings for Port *n* Control CSRs (Continued)

Bit	Name	Reset Value	Description
20-27	Implementation-defined		Implementation-defined
28-30	—		Reserved
31	Port Type		This indicates the port type (read only) 0b0 - Parallel port 0b1 - Reserved

[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

[6]The enumeration boundary reset value is implementation dependent. Provision shall be made to allow the reset value of this bit to be configurable on a per system basis if this feature is supported.

5.7 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.

5.7.1 Register Map

Table 5-21 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 + 0xBC]. Register map offset [EF_PTR + 0xC0] can be used for another Extended Features block.

Table 5-21. 8/16 LP-LVDS Register Map - Generic End Point Free Devices

	Block Byte Offset	Register Name
General	0x0	8/16 LP-LVDS Register Block Header
	0x4–1C	Reserved
	0x20	Port Link Timeout Control CSR
	0x24-38	Reserved
	0x3C	Port General Control CSR
Port 0	0x40-54	Reserved
	0x58	Port 0 Error and Status CSR
	0x5C	Port 0 Control CSR
Port 1	0x60-74	Reserved
	0x78	Port 1 Error and Status CSR
	0x7C	Port 1 Control CSR
Ports 2-14	0x80–218	Assigned to Port 2-14 CSRs
Port 15	0x220-234	Reserved
	0x238	Port 15 Error and Status CSR
	0x23C	Port 15 Control CSR

5.7.2 Command and Status Registers (CSRs)

Refer to Table 5-2 for the required behavior for accesses to reserved registers and register bits.

5.7.2.1 8/16 LP-LVDS Register Block Header (Block Offset 0x0)

The 8/16 LP-LVDS register block header register contains the EF_PTR to the next EF_BLK and the EF_ID that identifies this as the generic end point 8/16 LP-LVDS register block header.

Table 5-22. Bit Settings for 8/16 LP-LVDS Register Block Header

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

5.7.2.2 Port Link Timeout Control CSR (Block Offset 0x20)

The port link timeout control register contains the timeout timer value for all ports on a device. This timeout 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 timeout interval, and represents between 3 and 5 seconds.

Table 5-23. Bit Settings for Port Link Timeout Control CSR

Bit	Name	Reset Value	Description
0–23	timeout_value	All 1s	timeout interval value
24-31	—		Reserved

5.7.2.3 Port General Control CSR (Block Offset 0x3C)

The bits accessible through the Port General Control CSR are bits that apply to all ports on a device. There is a single copy of each such bit per device. These bits are also accessible through the Port General Control CSR of any other physical layers implemented on a device.

Table 5-24. Bit Settings for Port General Control CSRs

Bit	Name	Reset Value	Description
0-1	—		Reserved
2	Discovered	0b0	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

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

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

Table 5-25. 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).

5.7.2.5 Port *n* Control CSR (Block Offsets 0x5C, 7C, ..., 23C)

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

Table 5-26. 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. Control symbols are not affected and are sent normally. This is the recommended state after device reset. 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. 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. This is the recommended state after device reset. 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 output port (multiple port devices only)
10-13	—		Reserved
14	Enumeration Boundary	see footnote[6]	An enumeration boundary aware system enumeration algorithm shall honor this flag. The algorithm, on either the ingress or the egress port, shall not enumerate past a port with this bit set. This provides for software enforced enumeration domains within the RapidIO fabric.
15-19	—		Reserved

Table 5-26. Bit Settings for Port *n* Control CSRs (Continued)

Bit	Name	Reset Value	Description
20-27	Implementation-defined		Implementation-defined
28-30	—		Reserved
31	Port Type		This indicates the port type (read only) 0b0 - Parallel port 0b1 - Reserved

[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

[6]The enumeration boundary reset value is implementation dependent. Provision shall be made to allow the reset value of this bit to be configurable on a per system basis if this feature is supported.

5.8 Generic End Point Free Devices, software assisted error recovery option

This section describes the 8/16 LP-LVDS registers for a general device that does not contain end point device functionality that supports software assisted error recovery. Typically these devices are switches. 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=0x0009.

5.8.1 Register Map

Table 5-11 shows the register map for generic RapidIO 8/16 LP-LVDS end point-free 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 + 0xBC]. Register map offset [EF_PTR + 0xC0] can be used for another Extended Features block.

Table 5-27. 8/16 LP-LVDS Register Map - Generic End Point-free Devices (SW assisted)

	Block Byte Offset	Register Name
General	0x0	8/16 LP-LVDS Register Block Header
	0x4–1C	Reserved
	0x20	Port Link Timeout Control CSR
	0x24-38	Reserved
	0x3C	Port General Control CSR
Port 0	0x40	Port 0 Link Maintenance Request CSR
	0x44	Port 0 Link Maintenance Response CSR
	0x48	Port 0 Local ackID Status CSR
	0x4C-54	Reserved
	0x58	Port 0 Error and Status CSR
	0x5C	Port 0 Control CSR
Port 1	0x60	Port 1 Link Maintenance Request CSR
	0x64	Port 1 Link Maintenance Response CSR
	0x68	Port 1 Local ackID Status CSR
	0x6C-74	Reserved
	0x78	Port 1 Error and Status CSR
	0x7C	Port 1 Control CSR

Table 5-27. 8/16 LP-LVDS Register Map - Generic End Point-free Devices (SW assisted)

	Block Byte Offset	Register Name
Ports 2-14	0x80–218	Assigned to Port 2-14 CSRs
Port 15	0x220	Port 15 Link Maintenance Request CSR
	0x224	Port 15 Link Maintenance Response CSR
	0x228	Port 15 Local ackID Status CSR
	0x22C-234	Reserved
	0x238	Port 15 Error and Status CSR
	0x23C	Port 15 Control CSR

5.8.2 Command and Status Registers (CSRs)

Refer to Table 5-2 for the required behavior for accesses to reserved registers and register bits.

5.8.2.1 8/16 LP-LVDS Register Block Header (Block Offset 0x0)

The 8/16 LP-LVDS register block header register contains the EF_PTR to the next EF_BLK and the EF_ID that identifies this as the generic end point 8/16 LP-LVDS register block header.

Table 5-28. Bit Settings for 8/16 LP-LVDS Register Block Header

Bit	Name	Reset Value	Description
0-15	EF_PTR		Hard-wired pointer to the next block in the data structure, if one exists
16-31	EF_ID	0x0009	Hard-wired Extended Features ID

5.8.2.2 Port Link Timeout Control CSR (Block Offset 0x20)

The port link timeout control register contains the timeout timer value for all ports on a device. This timeout 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 timeout interval, and represents between 3 and 5 seconds.

Table 5-29. Bit Settings for Port Link Timeout Control CSR

Bit	Name	Reset Value	Description
0–23	timeout_value	All 1s	timeout interval value
24-31	—		Reserved

5.8.2.3 Port General Control CSR (Block Offset 0x3C)

The bits accessible through the Port General Control CSR are bits that apply to all ports on a device. There is a single copy of each such bit per device. These bits are also accessible through the Port General Control CSR of any other physical layers implemented on a device.

Table 5-30. Bit Settings for Port General Control CSRs

Bit	Name	Reset Value	Description
0-1	—		Reserved
2	Discovered	0b0	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

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

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 5-31. 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.

5.8.2.5 Port *n* Link Maintenance Response CSRs (Block Offsets 0x44, 64, ..., 224)

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 4.4, “Link Maintenance Control Symbol Formats.” This register is read-only.

Table 5-32. 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

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

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 5-33. Bit Settings for Port *n* Local ackID Status CSRs

Bit	Name	Reset Value	Description
0	Clr_outstanding_ackIDs	0b0	Writing 0b1 to this bit causes all outstanding unacknowledged packets to be discarded. This bit should only be written when trying to recover a failed link. This bit is always logic 0 when read.
1-4	—		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.
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.

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

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

Table 5-34. 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).

5.8.2.8 Port *n* Control CSR (Block Offsets 0x5C, 7C, ..., 23C)

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

Table 5-35. 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. Control symbols are not affected and are sent normally. This is the recommended state after device reset. 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. 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. This is the recommended state after device reset. 0b1 - port is enabled to respond to any packet
6	Input Port Receiver Disable	0b0	Input port receiver enable: 0b0 - input port receivers are enabled 0b1 - input port receivers are disabled and are unable to receive to any packets or control symbols
7	—		Reserved
8	Error Checking Disable	0b0	This bit disables all RapidIO transmission error checking 0b0 - Error checking and recovery is enabled 0b1 - Error checking and recovery is disabled Device behavior when error checking and recovery is disabled and an error condition occurs is undefined
9	Multicast-event Participant	see footnote[5]	Send incoming multicast-event control symbols to this port (multiple port devices only)
10-13	—		Reserved
14	Enumeration Boundary	see footnote[6]	An enumeration boundary aware system enumeration algorithm shall honor this flag. The algorithm, on either the ingress or the egress port, shall not enumerate past a port with this bit set. This provides for software enforced enumeration domains within the RapidIO fabric.
15-19	—		Reserved

Table 5-35. Bit Settings for Port *n* Control CSRs (Continued)

Bit	Name	Reset Value	Description
20-27	Implementation-defined		Implementation-defined
28-30	—		Reserved
31	Port Type		This indicates the port type (read only) 0b0 - Parallel port 0b1 - Reserved

[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

[6]The enumeration boundary reset value is implementation dependent. Provision shall be made to allow the reset value of this bit to be configurable on a per system basis if this feature is supported.

Chapter 6 System Clocking Considerations

6.1 Introduction

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.

6.2 Example Clock Distribution

Clock distribution in a small system is straightforward. It is assumed that clocking is provided from a single clock source (Figure 6-1).

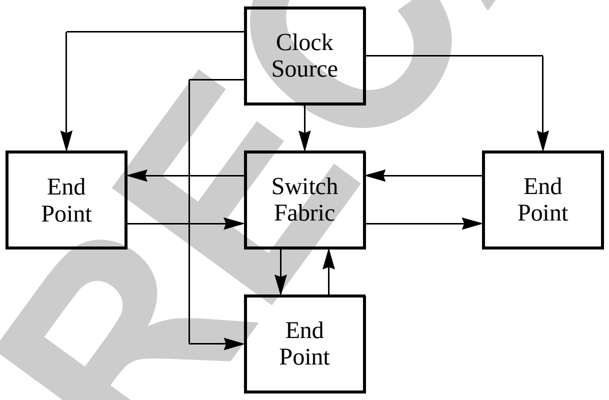


Figure 6-1. Clock Distribution in a Small System

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 6-2 displays the clock distribution in a larger system.

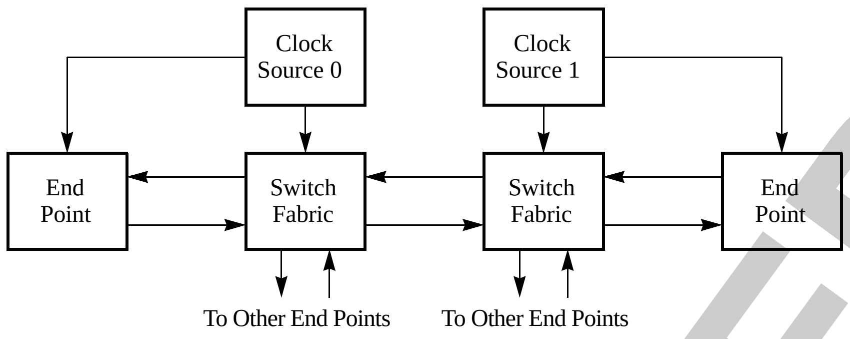


Figure 6-2. Clock Distribution in a Larger System

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 6-3, then additive clock jitter must be taken into account.

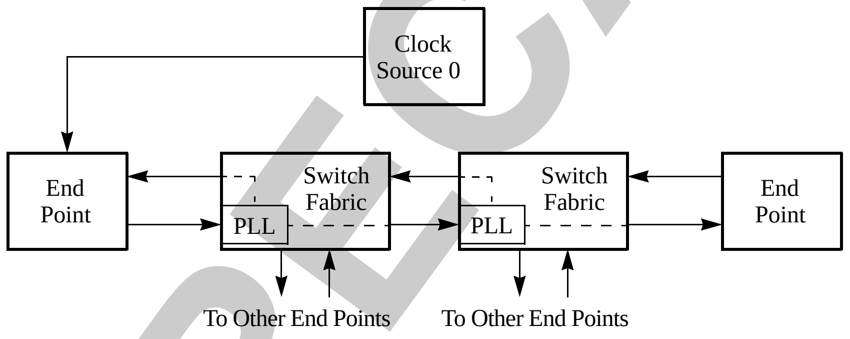


Figure 6-3. Clock Distribution Through the Interconnect

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.

6.3 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.

Blank page

Chapter 7 Board Routing Guidelines

7.1 Introduction

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.

7.2 Impedance

Interconnect design should follow standard practice for differential pairs. To minimize reflections from the receiver's 100 Ohm termination, each side of the coupled pair should have a characteristic impedence of 50 Ohms (i..e. 100 Ohms differential impedence). 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.

7.3 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 7-1.

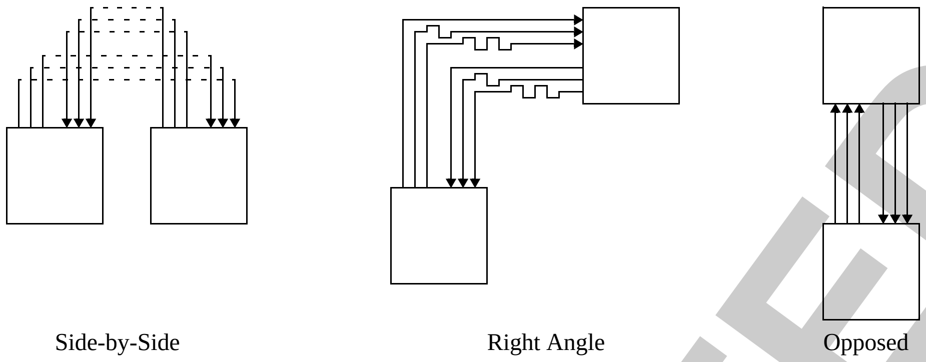


Figure 7-1. Routing for Equalized Skew for Several Placements

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.

7.4 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 7-2.

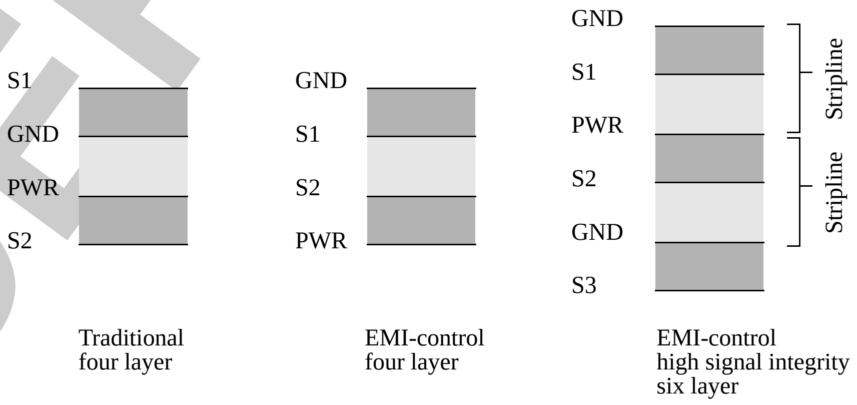


Figure 7-2. Potential PCB Stackups

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.

7.5 Termination

Depending upon the individual device characteristics and the requirements of the particular application, the board route may be required to encompass external devices such as terminating resistors or networks. The effect of such devices on the board route must be carefully analyzed and controlled.

7.6 Additional Considerations

The application environment for a RapidIO channel may place additional constraints on the PCB design.

7.6.1 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.

7.6.2 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.

7.6.3 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.

7.7 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 7-1) is shown graphically in Figure 7-3 and Figure 7-4. 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 8, “Signal Descriptions”.

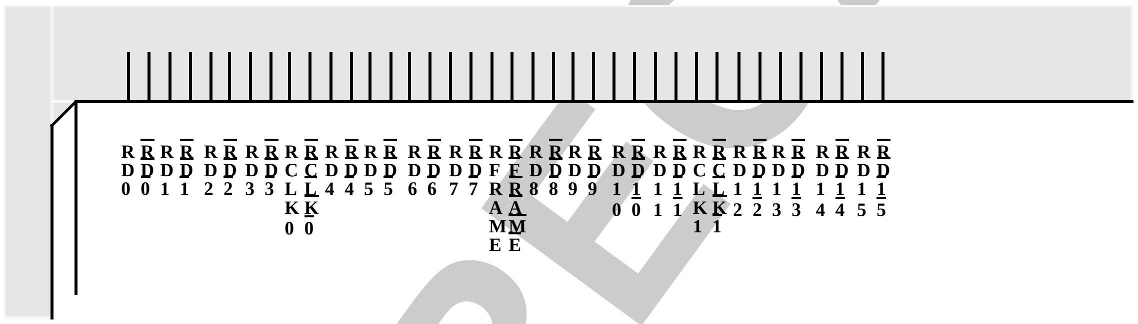


Figure 7-3. Recommended device pin escape, input port, top view of device

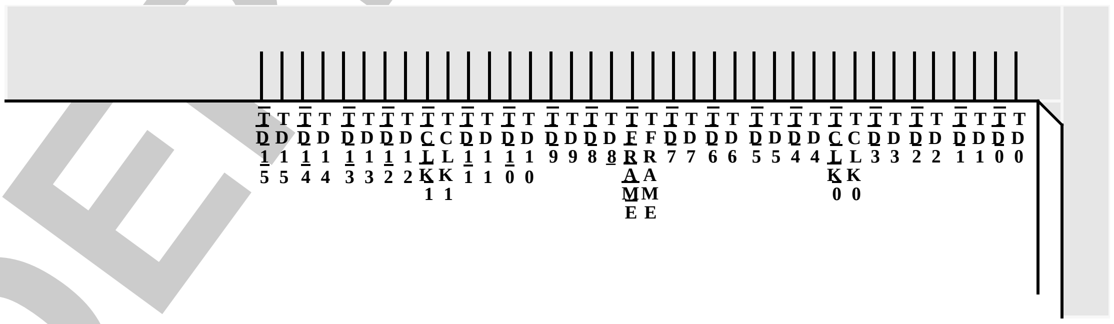


Figure 7-4. Recommended device pin escape, output port, top view of device

These pin escapes allow clean board routes that provide maximum performance connections between two devices as can be seen in the example in Figure 7-5 below.

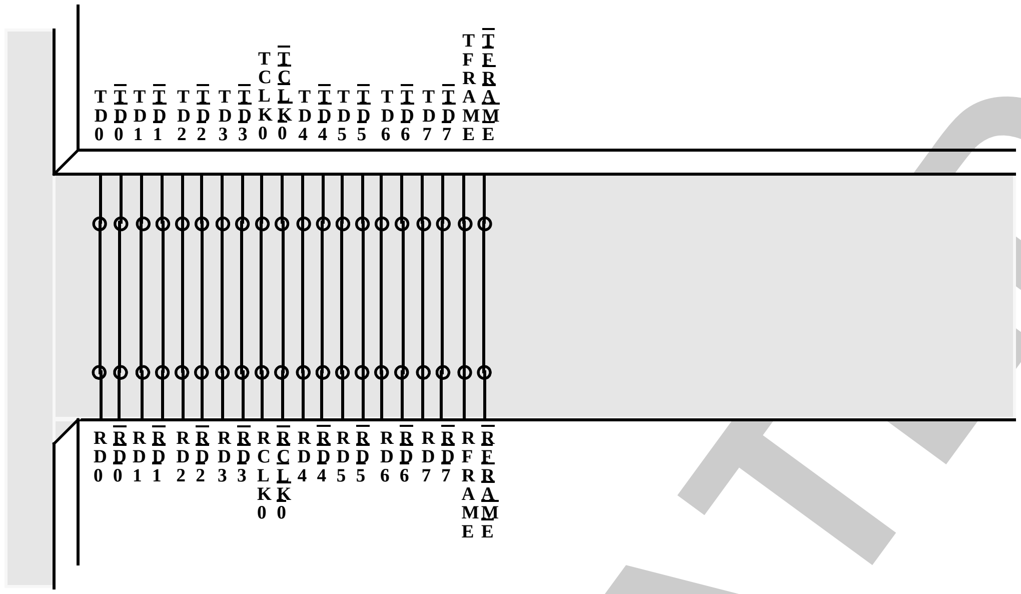


Figure 7-5. Opposed orientation, same side of board

If the attached devices are mounted with certain device orientations the bit wires become crossed. An example of this situation is shown in Figure 7-6. 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 7-6 and Figure 7-7.

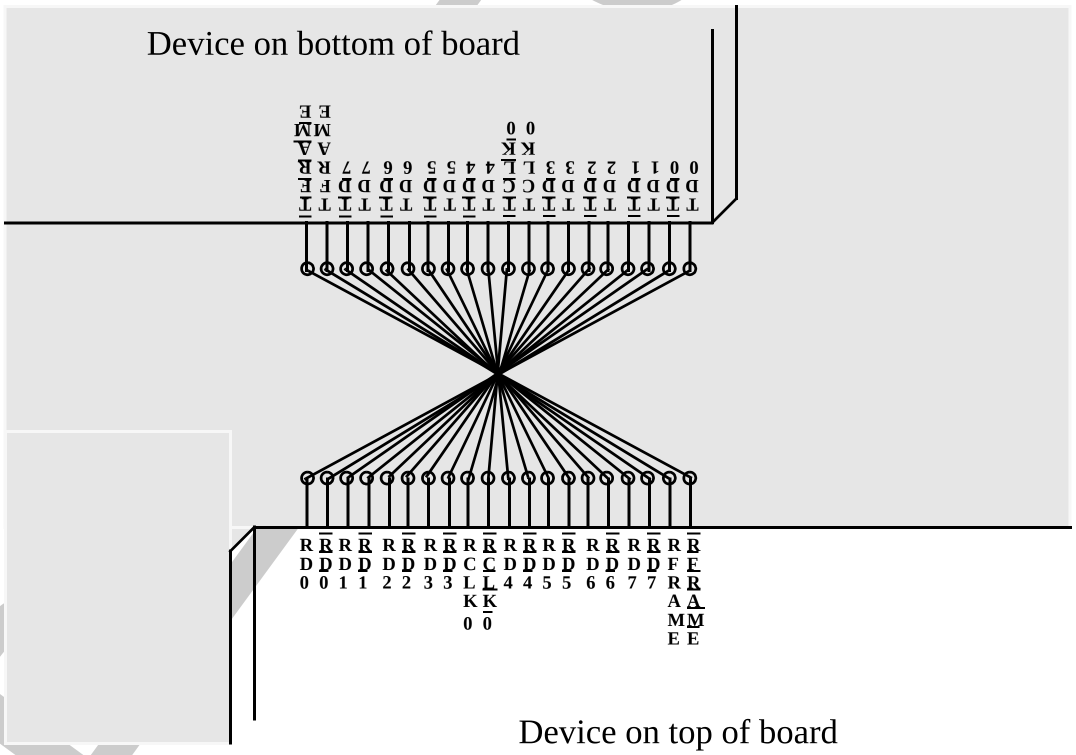


Figure 7-6. Opposed orientation, opposite sides of board

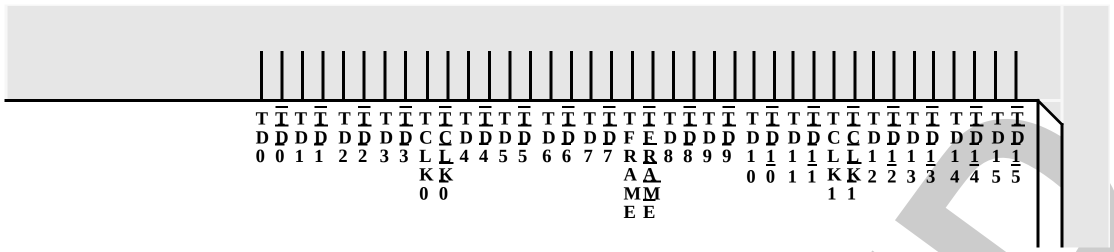


Figure 7-7. Recommended device pin escape, output port reversed, top view of device

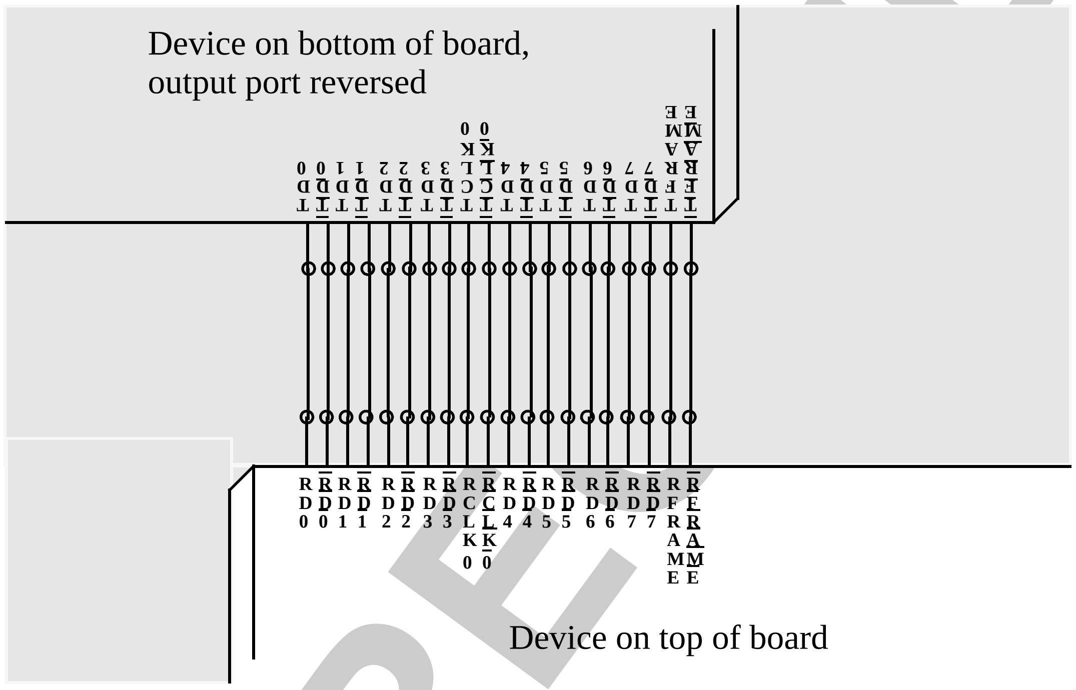


Figure 7-8. Opposed orientation, output port reversed, opposite sides of board

Chapter 8 Signal Descriptions

8.1 Introduction

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 9, “Electrical Specifications.”

8.2 Signal Definitions

Table 8-1 provides a summary of the RapidIO signal pins as well as a short description of their functionality.

Table 8-1. 8/16 LP-LVDS Signal Descriptions

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.	
$\overline{\text{TCLK1}}$	O	Transmit Clock complement—This signal is the differential pair of the TCLK1 signal.	

Table 8-1. 8/16 LP-LVDS Signal Descriptions (Continued)

Signal Name	I/O	Signal Meaning	Timing Comments
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.	

8.3 RapidIO Interface Diagrams

Figure 8-1 shows the signal interface diagram connecting two 8-bit devices together with the RapidIO 8/16 LP-LVDS interconnect.

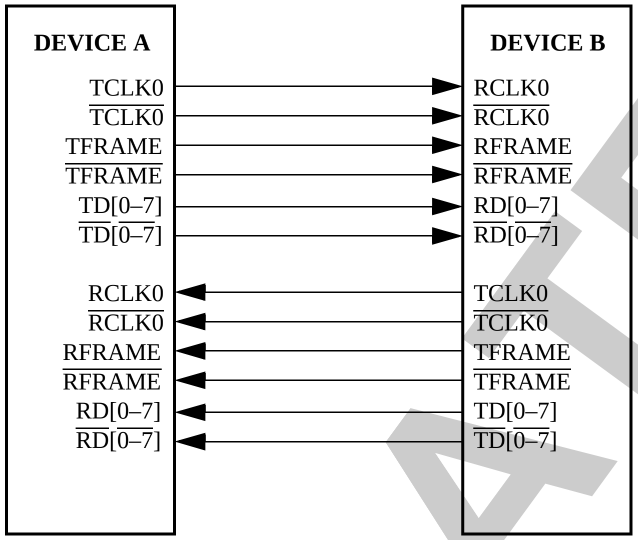

DEVICE A
TCLK0
$\overline{\text{TCLK0}}$
TFRAME
$\overline{\text{TFRAME}}$
TD[0–7]
$\overline{\text{TD[0–7]}}$
RCLK0
$\overline{\text{RCLK0}}$
RFRAME
$\overline{\text{RFRAME}}$
RD[0–7]
$\overline{\text{RD[0–7]}}$
DEVICE B
RCLK0
$\overline{\text{RCLK0}}$
RFRAME
$\overline{\text{RFRAME}}$
RD[0–7]
$\overline{\text{RD[0–7]}}$
TCLK0
$\overline{\text{TCLK0}}$
TFRAME
$\overline{\text{TFRAME}}$
TD[0–7]
$\overline{\text{TD[0–7]}}$

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

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

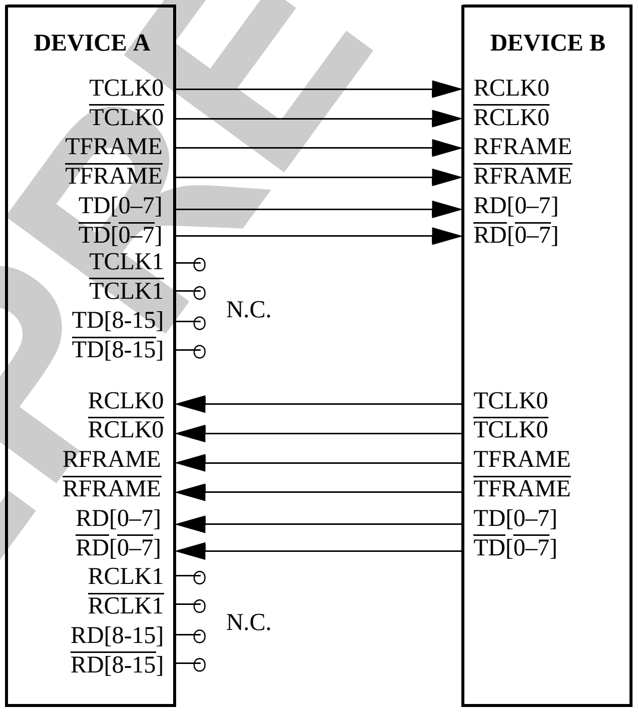

DEVICE A
TCLK0
$\overline{\text{TCLK0}}$
TFRAME
$\overline{\text{TFRAME}}$
TD[0–7]
$\overline{\text{TD[0–7]}}$
TCLK1
$\overline{\text{TCLK1}}$
TD[8-15]
$\overline{\text{TD[8-15]}}$
N.C.
RCLK0
$\overline{\text{RCLK0}}$
RFRAME
$\overline{\text{RFRAME}}$
RD[0–7]
$\overline{\text{RD[0–7]}}$
RCLK1
$\overline{\text{RCLK1}}$
RD[8-15]
$\overline{\text{RD[8-15]}}$
N.C.
DEVICE B
RCLK0
$\overline{\text{RCLK0}}$
RFRAME
$\overline{\text{RFRAME}}$
RD[0–7]
$\overline{\text{RD[0–7]}}$
TCLK0
$\overline{\text{TCLK0}}$
TFRAME
$\overline{\text{TFRAME}}$
TD[0–7]
$\overline{\text{TD[0–7]}}$

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

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

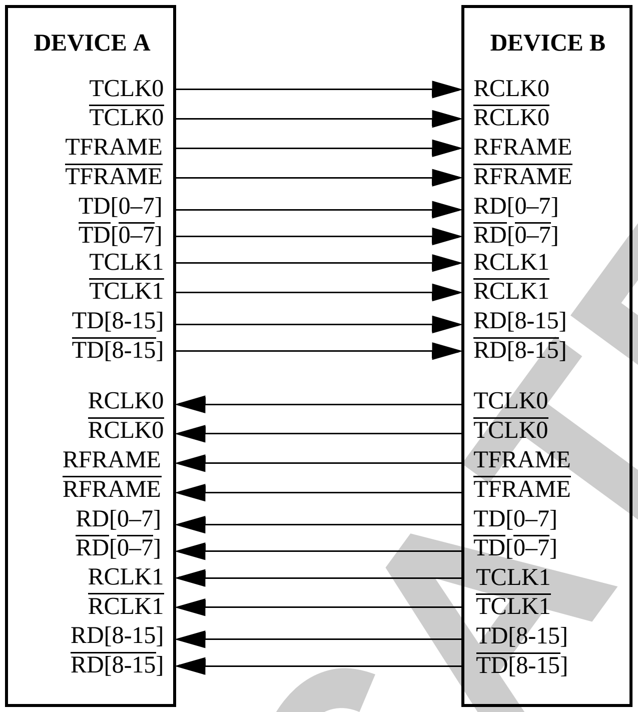

DEVICE A
DEVICE B
TCLK0
$\overline{\text{TCLK0}}$
TFRAME
$\overline{\text{TFRAME}}$
TD[0–7]
$\overline{\text{TD[0–7]}}$
TCLK1
$\overline{\text{TCLK1}}$
TD[8-15]
$\overline{\text{TD[8-15]}}$
RCLK0
$\overline{\text{RCLK0}}$
RFRAME
$\overline{\text{RFRAME}}$
RD[0–7]
$\overline{\text{RD[0–7]}}$
RCLK1
$\overline{\text{RCLK1}}$
RD[8-15]
$\overline{\text{RD[8-15]}}$

Figure 8-3. RapidIO 16-bit Device to 16-bit Device Interface Diagram

Chapter 9 Electrical Specifications

9.1 Introduction

This chapter contains the driver and receiver AC and DC electrical specifications for a *RapidIO Part 4: 8/16 LP-LVDS Physical Layer Specification* compliant device. The interface defined is a parallel differential low-power high-speed signal interface.

9.2 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.

9.3 DC Specifications

RapidIO driver and receiver DC specifications are displayed in Table 9-1 and Table 9-2. Power variation is +/- 5%. Resistor tolerances are +/- 1%.

Table 9-1. 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 9-1
Differential output low voltage	V_{OLD}	-454	-247	mV	Bridged 100Ω load See Figure 9-1
Differential offset voltage	ΔV_{OD}		50	mV	Bridged 100Ω load $\lvert V_{OHD}+V_{OLD}\rvert$. See Figure 9-1
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 9-1
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 9-2. 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}	80	120	Ω	For on-chip termination.[1]

[1]Off-chip termination value and tolerance is vendor defined consistant with the return loss specification. Receiver input impedance shall exhibit a differential return loss better than 10 dB from DC to (1.6 * AC Clock Frequency). The differential return loss must measured at and include effects due to the receiver itself, associated circuitry such as ESD structures, chip packaging, and any external termination structures related to the receiver. The reference impedance for measurement is 100 ohms.

DC driver signal levels are displayed in Figure 9-1.

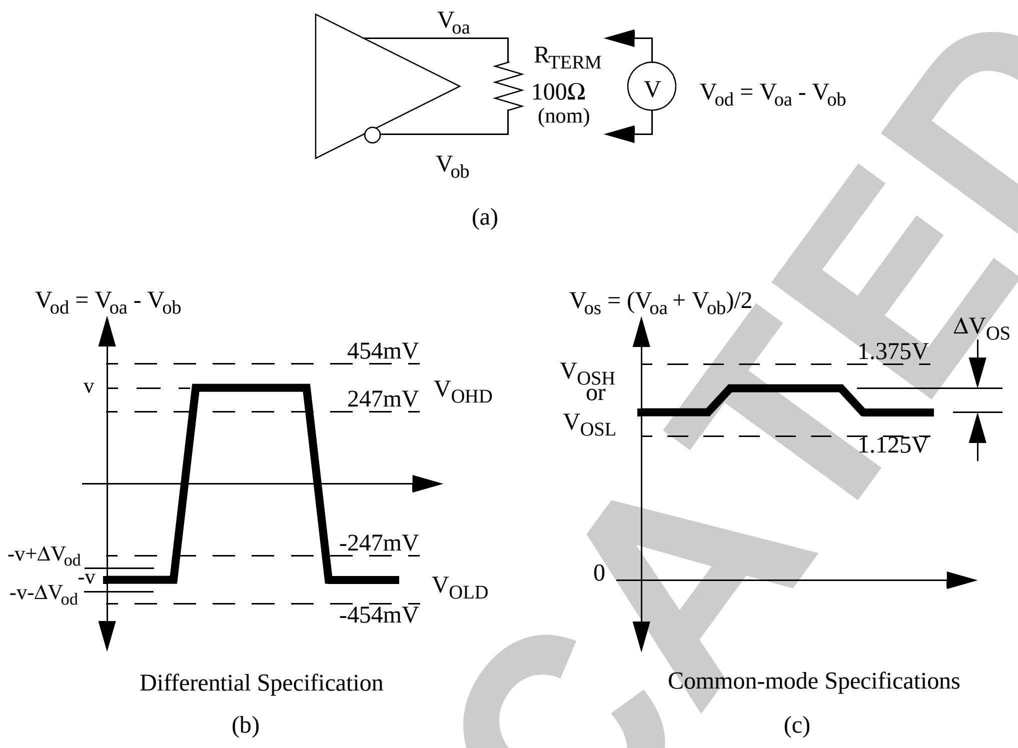


Figure 9-1. DC driver signal levels

9.4 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.

9.4.1 Concepts and Definitions

This section specifies signals using differential voltages. Figure 9-2 shows how the signals are defined. The figure shows waveforms for either a transmitter output (TD and $\overline{\text{TD}}$) or a receiver input (RD and $\overline{\text{RD}}$). 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, $\overline{\text{TD}}$, RD and $\overline{\text{RD}}$ 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.

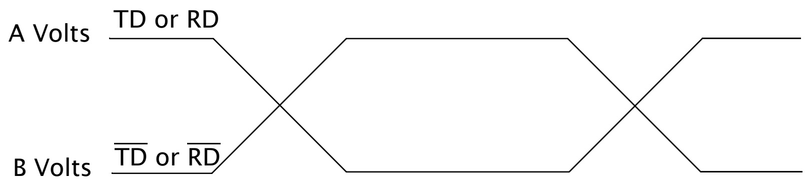


Figure 9-2. Differential Peak-Peak Voltage of Transmitter or Receiver

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, $\overline{\text{TD}}$, RD and $\overline{\text{RD}}$ 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 9-3. 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.

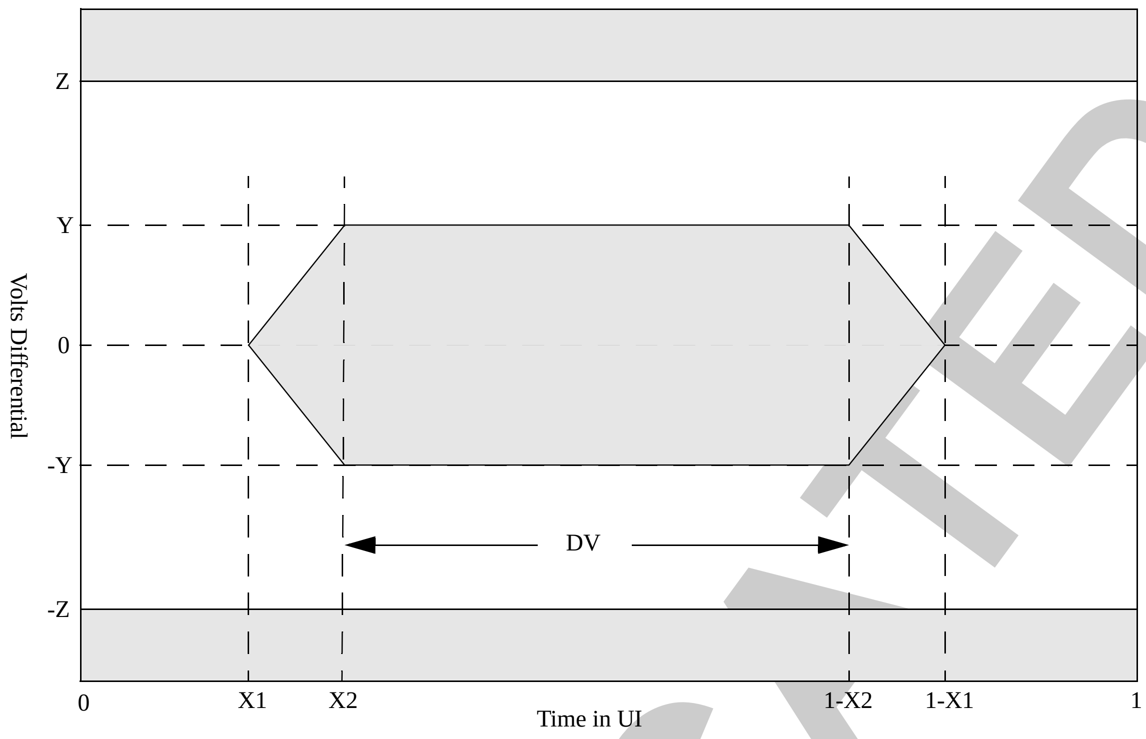


Figure 9-3. Example Compliance Mask

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.

9.4.2 Driver Specifications

Driver AC timing specifications are given in Table 9-3 through Table 9-7 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.

Driver DC termination is not specified (in accordance with TIA/EIA-644-A), but is recommended for devices targeting higher data rates. This termination is intended to reduce data reflections in the matched data interconnect. The value and location of this termination and the methods of test and measurement are left to the individual vendor.

Table 9-3. 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 9-4
Differential output low voltage	V_{OLD}	-540	-200	mV	See Figure 9-4
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 9-4
Allowable static skew between any two data outputs within a 8 bit/9 bit group	t_{DPAIR}		.09	UI	See Figure 9-10

Table 9-3. Driver AC Timing Specifications - 500Mbps Data Rate/250MHz Clock Rate

Characteristic	Symbol	Range		Unit	Notes
		Min	Max		
Allowable static skew of data outputs to associated clock	$t_{SKEW,PAIR}$	-.09	.09	UI	See Figure 9-8, Figure 9-10
Clock to clock static skew	$t_{CSKEW, PAIR}$		.09	UI	See Figure 9-9
Clock to clock dynamic skew	$t_{CSKEW, PAIRD}$		.2	UI	See Figure 9-9

Table 9-4. 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 9-4
Differential output low voltage	V_{OLD}	-540	-200	mV	See Figure 9-4
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 9-4
Allowable static skew between any two data outputs within a 8 bit/9 bit group	t_{DPAIR}		.1	UI	See Figure 9-10
Allowable static skew of data outputs to associated clock	$t_{SKEW,PAIR}$	-.1	.1	UI	See Figure 9-8, Figure 9-10
Clock to clock static skew	$t_{CSKEW, PAIR}$		.15	UI	See Figure 9-9
Clock to clock dynamic skew	$t_{CSKEW, PAIRD}$		.2	UI	See Figure 9-9

Table 9-5. 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 9-4
Differential output low voltage	V_{OLD}	-540	-200	mV	See Figure 9-4
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	

Table 9-5. Driver AC Timing Specifications - 1000Mbps Data Rate/500MHz 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	.575		UI	Measured using the RapidIO Transmit Mask shown in Figure 9-4
Allowable static skew between any two data outputs within a 8 bit/9 bit group	t_{DPAIR}		.1	UI	See Figure 9-10
Allowable static skew of data outputs to associated clock	$t_{SKEW,PAIR}$	-.1	.1	UI	See Figure 9-8, Figure 9-10
Clock to clock static skew	$t_{CSKEW, PAIR}$		.15	UI	See Figure 9-9
Clock to clock dynamic skew	$t_{CSKEW, PAIRD}$		.2	UI	See Figure 9-9

Table 9-6. 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 9-4
Differential output low voltage	V_{OLD}	-540	-200	mV	See Figure 9-4
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 9-4
Allowable static skew between any two data outputs within a 8 bit/9 bit group	t_{DPAIR}		.2	UI	See Figure 9-10
Allowable static skew of data outputs to associated clock	$t_{SKEW,PAIR}$	-.2	.2	UI	See Figure 9-8, Figure 9-10
Clock to clock static skew	$t_{CSKEW, PAIR}$		.15	UI	See Figure 9-9
Clock to clock dynamic skew	$t_{CSKEW, PAIRD}$		.2	UI	See Figure 9-9

Table 9-7. 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 9-4
Differential output low voltage	V_{OLD}	-540	-200	mV	See Figure 9-4
Unit interval		500	500	ps	Requires +/-100ppm long term frequency stability
Duty cycle of the clock output	DC	48	52	%	Measured at V_{OD}=0V
V_{OD} fall time, 20-80% of the peak to peak differential signal swing	t_{FALL}	.1		UI	
V_{OD} rise time, 20-80% of the peak to peak differential signal swing	t_{RISE}	.1		UI	
Data Valid	DV	.5		UI	Measured using the RapidIO Transmit Mask shown in Figure 9-4
Allowable static skew between any two data outputs within a 8 bit/9 bit group	t_{DPAIR}		.2	UI	See Figure 9-10
Allowable static skew of data outputs to associated clock	$t_{SKEW,PAIR}$	-.2	.2	UI	See Figure 9-8, Figure 9-10
Clock to clock static skew	$t_{CSKEW, PAIR}$		.2	UI	See Figure 9-9
Clock to clock dynamic skew	$t_{CSKEW, PAIRD}$		.2	UI	See Figure 9-9

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 9-4. 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.

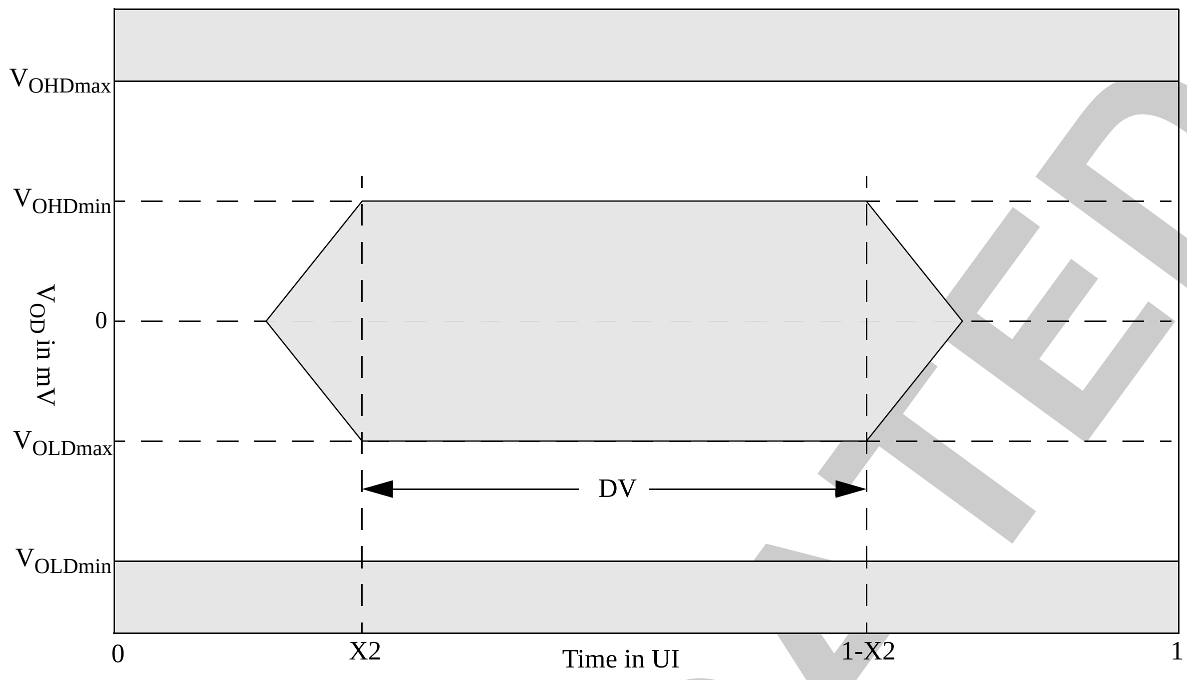
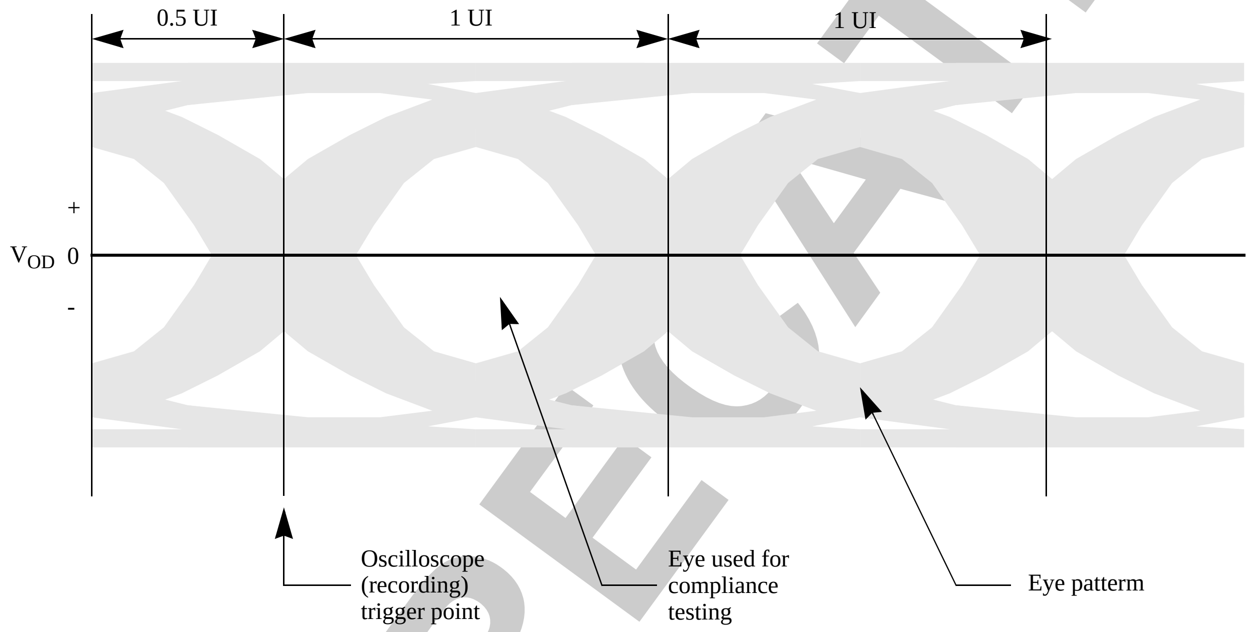


Figure 9-4. RapidIO Transmit Mask

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 9-5. In this example, there is no skew between the signal under test and the associated clock used to trigger the recordings. If skew was present, the eye pattern would be shifted to the left or right relative to the oscilloscope trigger point.



0.5 UI
1 UI
1 UI
+
V_{OD} 0
-
Oscilloscope (recording) trigger point
Eye used for compliance testing
Eye patterm

Figure 9-5. Example Driver Output Eye Pattern

9.4.3 Receiver Specifications

Receiver AC timing specifications are given in Table 9-8 through Table 9-12 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 9-8. 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 9-6
Allowable static skew between any two data inputs within a 8 bit/9 bit group	t_{DPAIR}		.19	UI	See Figure 9-10
Allowable static skew of data inputs to associated clock	$t_{SKEW,PAIR}$	-.15	.15	UI	See Figure 9-8, Figure 9-10
Clock to clock static skew	$t_{CSKEW, PAIR}$		.14	UI	See Figure 9-9
Clock to clock dynamic skew	$t_{CSKEW, PAIRD}$		.3	UI	See Figure 9-9

Table 9-9. 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 9-6

Table 9-9. 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 9-10
Allowable static skew of data inputs to associated clock	$t_{SKEW,PAIR}$	-.2	.2	UI	See Figure 9-8, Figure 9-10
Clock to clock static skew	$t_{CSKEW, PAIR}$		.2	UI	See Figure 9-9
Clock to clock dynamic skew	$t_{CSKEW, PAIRD}$		.3	UI	See Figure 9-9

Table 9-10. 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 9-6
Allowable static skew between any two data inputs within a 8 bit/9 bit group	t_{DPAIR}		.3	UI	See Figure 9-10
Allowable static skew of data inputs to associated clock	$t_{SKEW,PAIR}$	-.2	.2	UI	See Figure 9-8, Figure 9-10
Clock to clock static skew	$t_{CSKEW, PAIR}$		.2	UI	See Figure 9-9
Clock to clock dynamic skew	$t_{CSKEW, PAIRD}$		.3	UI	See Figure 9-9

Table 9-11. 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 9-6
Allowable static skew between any two data inputs within a 8 bit/9 bit group	t_{DPAIR}		.4	UI	See Figure 9-10
Allowable static skew of data inputs to associated clock	$t_{SKEW,PAIR}$	-.25	.25	UI	See Figure 9-8, Figure 9-10
Clock to clock static skew	$t_{CSKEW, PAIR}$		.3	UI	See Figure 9-9
Clock to clock dynamic skew	$t_{CSKEW, PAIRD}$		.3	UI	See Figure 9-9

Table 9-12. 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 9-6
Allowable static skew between any two data inputs within a 8 bit/9 bit group	t_{DPAIR}		.4	UI	See Figure 9-10
Allowable static skew of data inputs to associated clock	$t_{SKEW,PAIR}$	-.25	.25	UI	See Figure 9-8, Figure 9-10
Clock to clock static skew	$t_{CSKEW, PAIR}$		.3	UI	See Figure 9-9
Clock to clock dynamic skew	$t_{CSKEW, PAIRD}$		.3	UI	See Figure 9-9

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 9-6. 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.

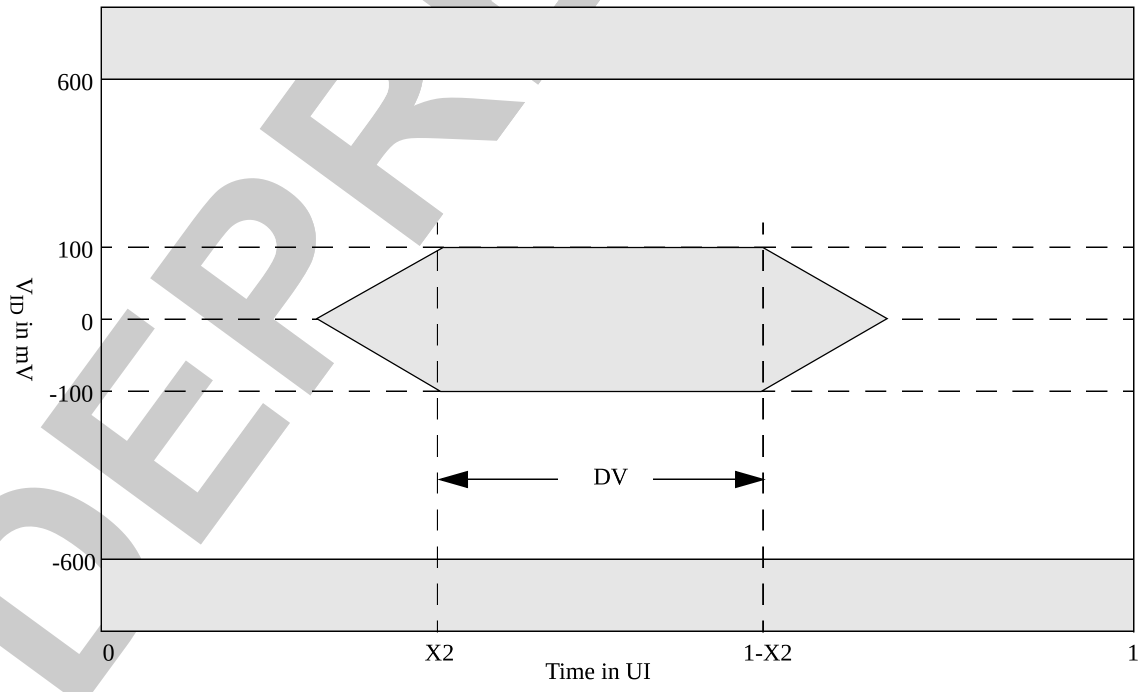


Figure 9-6. RapidIO Receive Mask

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 9-7. In this example, there is no skew between the signal under test and the associated clock used to trigger the recordings. If skew was present, the eye pattern would be shifted to the left or right relative to the oscilloscope trigger point.

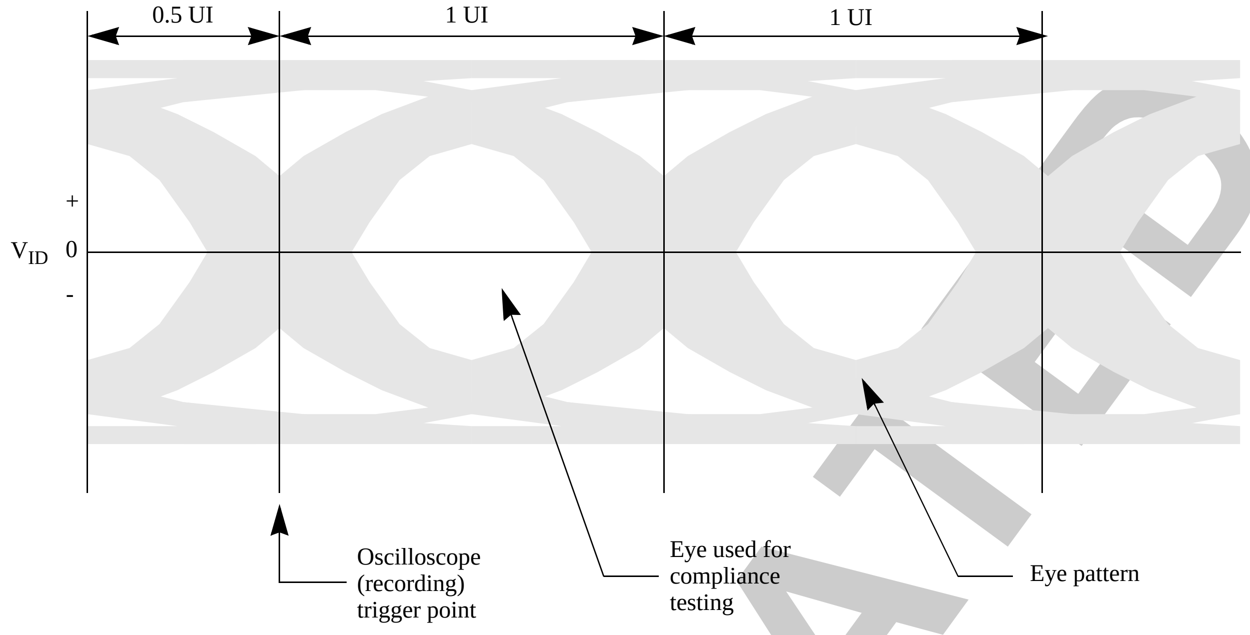

0.5 UI
1 UI
1 UI
+
V_ID 0
-
Oscilloscope (recording) trigger point
Eye used for compliance testing
Eye pattern

Figure 9-7. Example Receiver Input Eye Pattern

Figure 9-8 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.

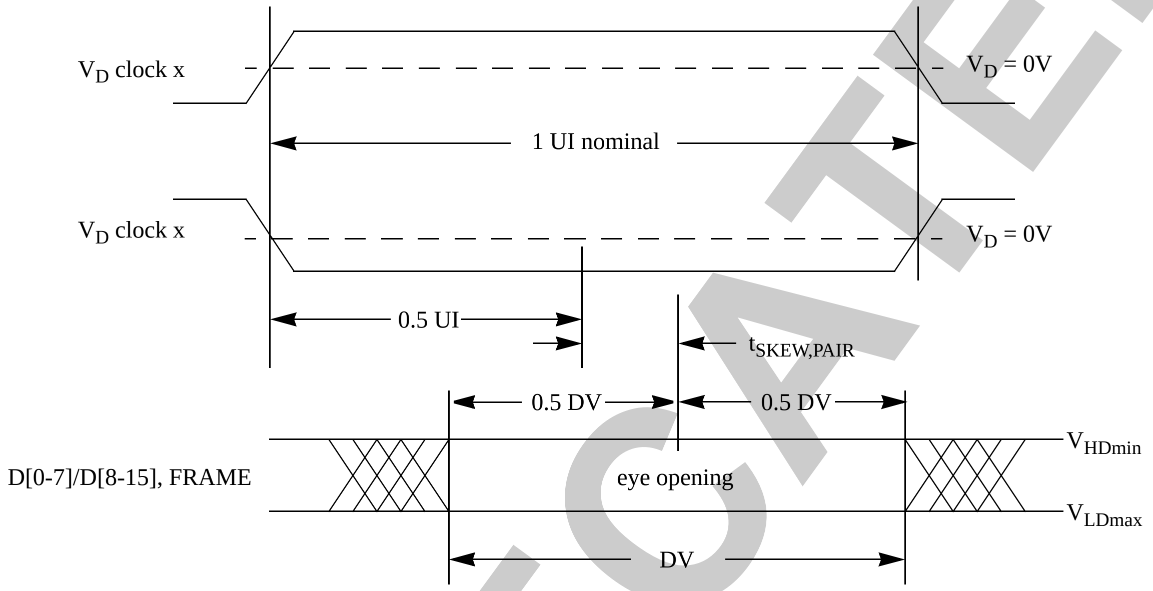


Figure 9-8. Data to Clock Skew

Figure 9-9 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.

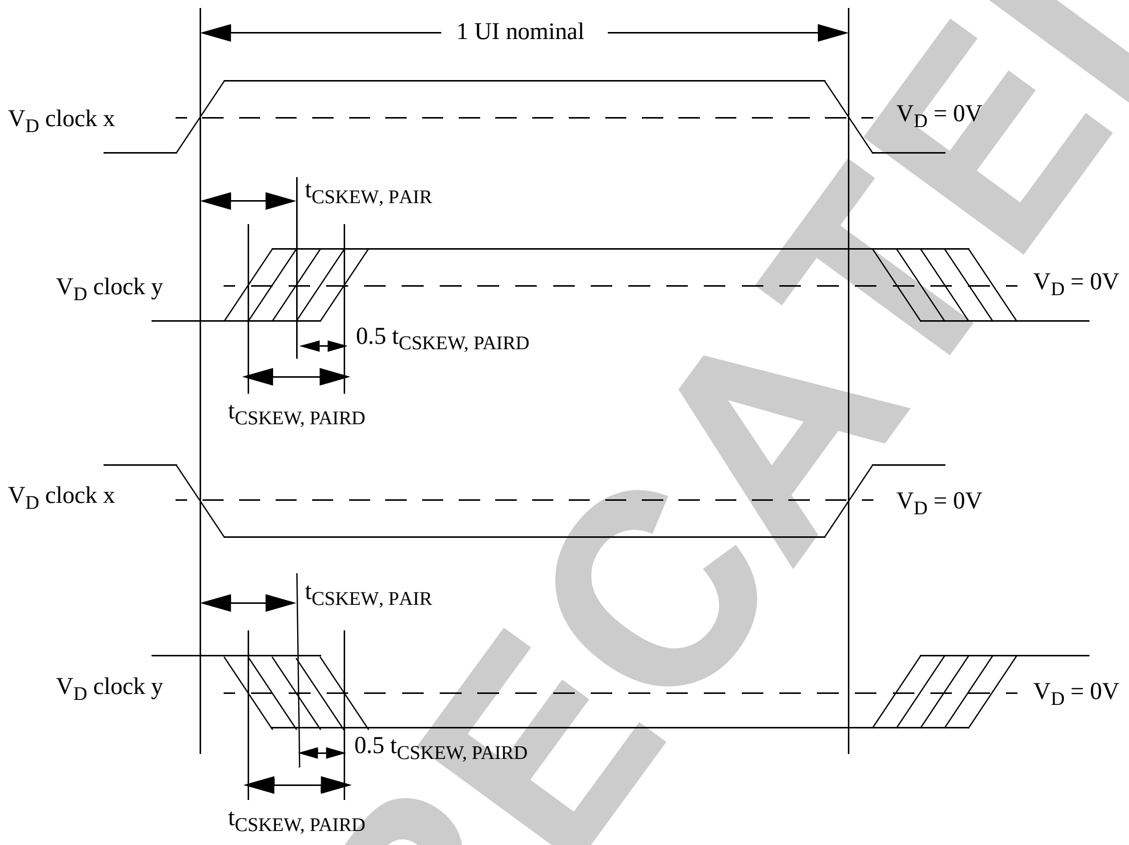


Figure 9-9. Clock to Clock Skew

Figure 9-10 shows the definition of the data to data static skew parameter t_{DPAIR} and how the skew parameters are applied.

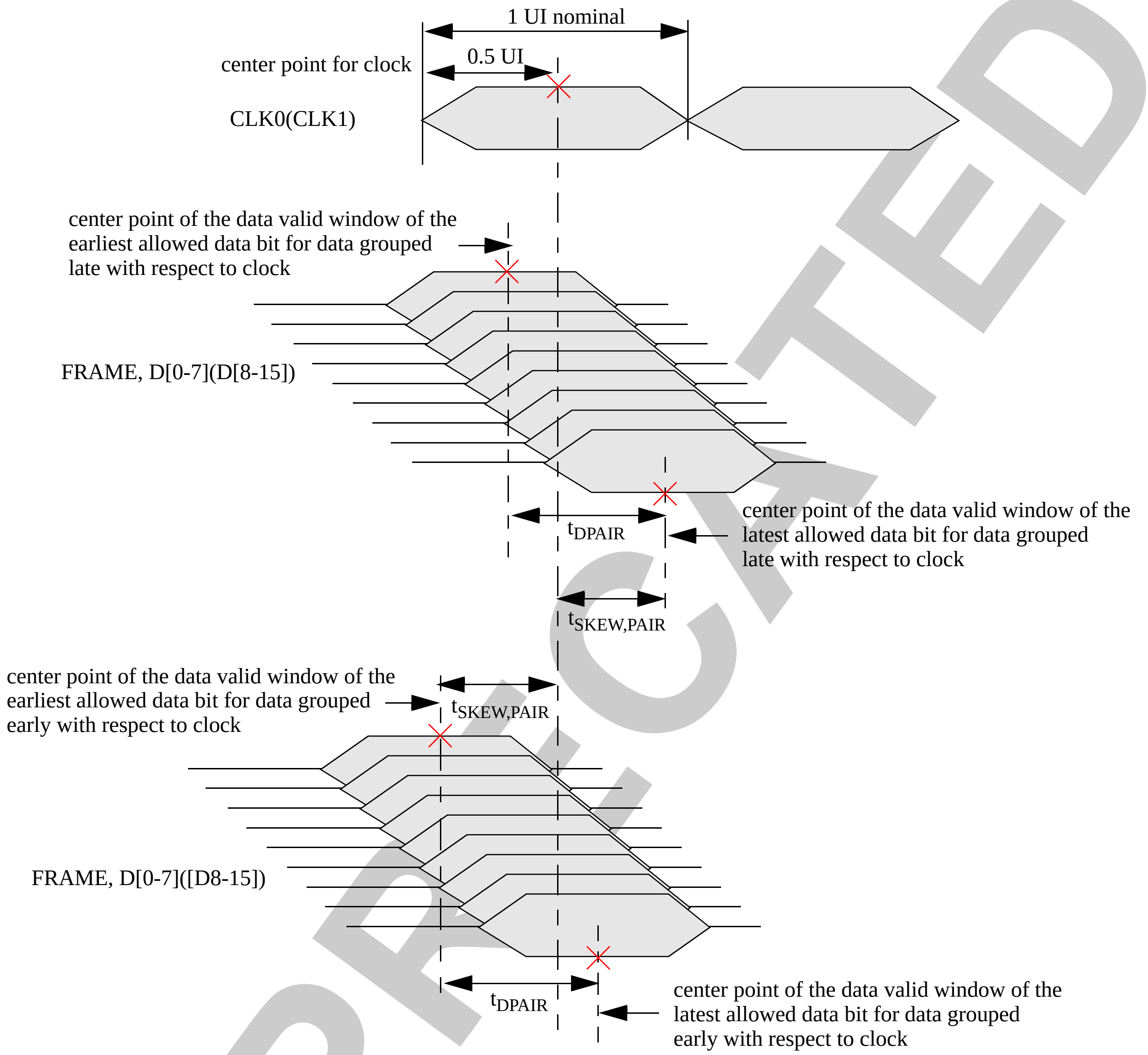

1 UI nominal
0.5 UI
center point for clock
CLK0(CLK1)
center point of the data valid window of the earliest allowed data bit for data grouped late with respect to clock
FRAME, D[0-7](D[8-15])
tDPAIR
center point of the data valid window of the latest allowed data bit for data grouped late with respect to clock
tSKEW,PAIR
center point of the data valid window of the earliest allowed data bit for data grouped early with respect to clock
tSKEW,PAIR
FRAME, D[0-7]([D8-15])
tDPAIR
center point of the data valid window of the latest allowed data bit for data grouped early with respect to clock

Figure 9-10. Static Skew Diagram

Annex A Interface Management (Informative)

A.1 Introduction

This appendix contains state machine descriptions that illustrate a number of behaviors that are described in the *RapidIO Part 4: 8/16 LP-LVDS Physical Layer Specification*. They are included as examples and are believed to be correct, however, actual implementations should not use the examples directly.

A.2 Link Initialization and Maintenance Mechanism

This section contains the link training and initialization state machine referred to in Section 3.7.1.1, “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.2.1 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.

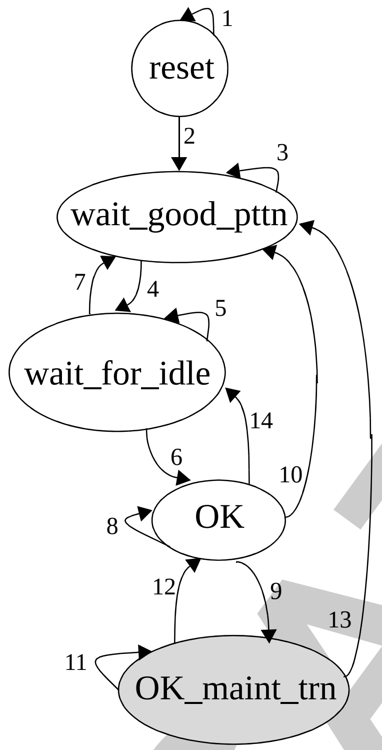


Figure A-1. Input port training state machine

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

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

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.

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.
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.2.2 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.

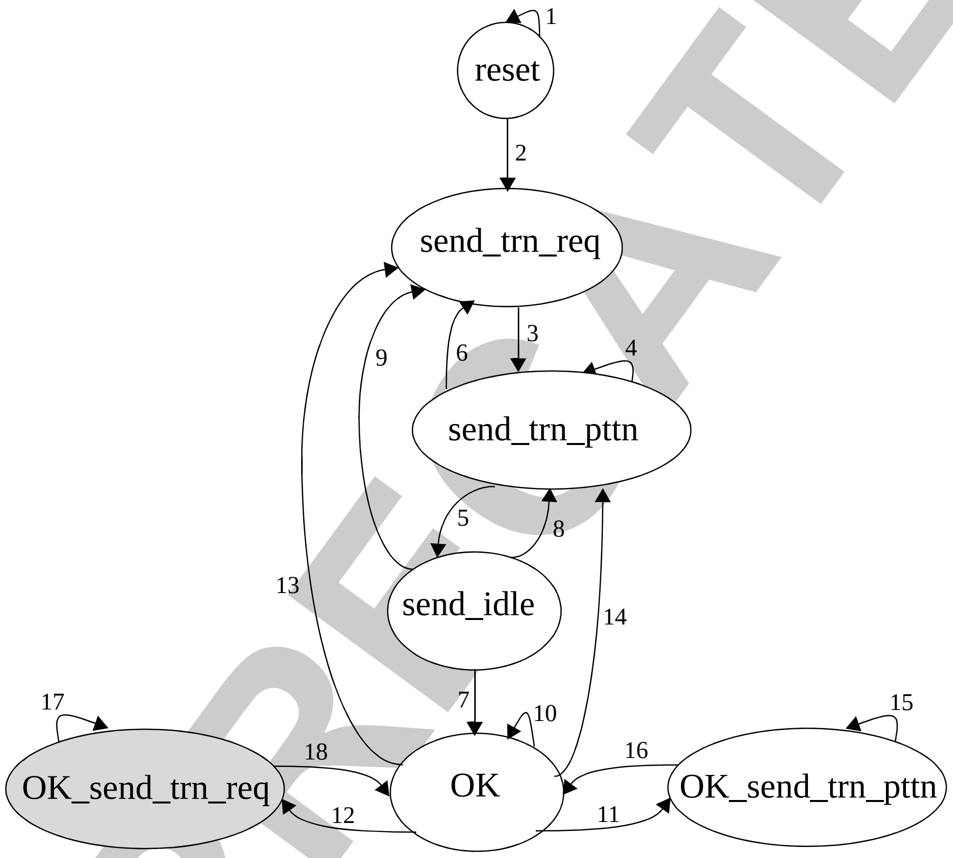


Figure A-2. Output port training state machine

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.
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.

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

Arc	Current State	Next state	cause	Comments
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.
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.3 Packet Retry Mechanism

This section contains the example packet retry mechanism state machine referred to in Section 2.3.3, “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.3.1 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.

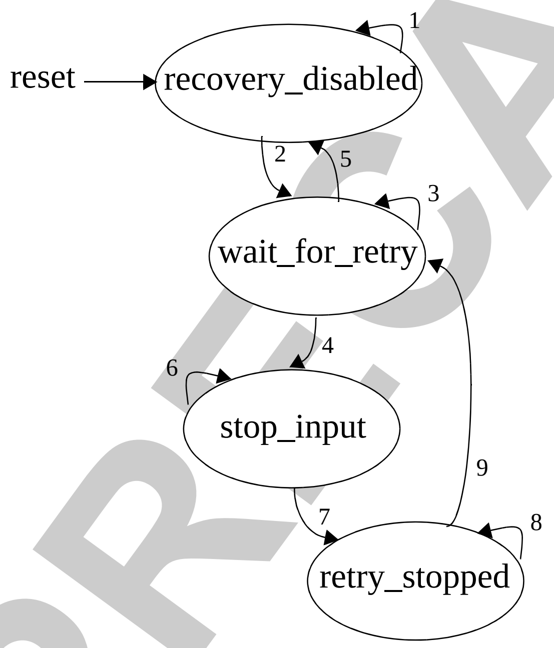


Figure A-3. Input port retry recovery state machine

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.3.2 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.

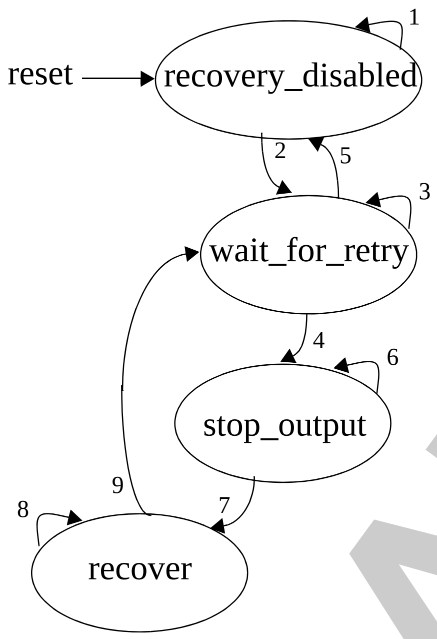


Figure A-4. Output port retry recovery state machine

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.4 Error Recovery

This section contains the error recovery state machine referred to in Section 2.4.5, “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.4.1 Input port error recovery state machine

There are a variety of recoverable error types described in detail in Section 2.4.5, “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.

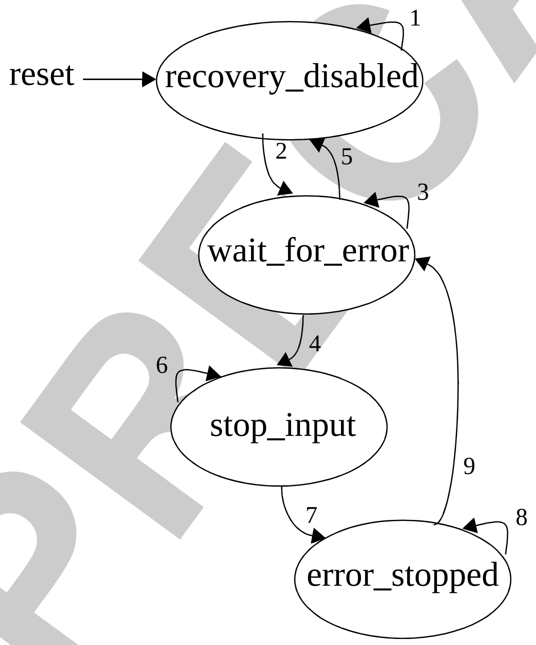


Figure A-5. Input port error recovery state machine

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.4.2 Output port error recovery state machine

The second recoverable group of errors described in Section 2.4.5, "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.

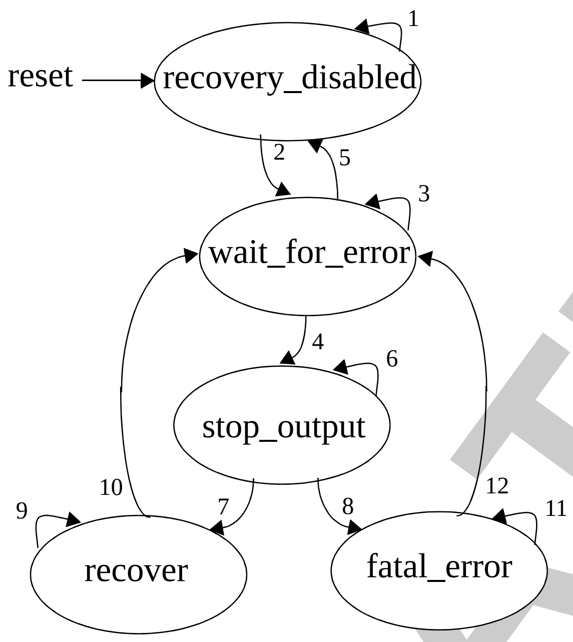


Figure A-6. Output port error recovery state machine

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 timeout several times before regarding a timeout 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.

Blank page

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.

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.

C

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.

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.

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.

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.

First symbol. The leading 16 bits of a packet.

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

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

I/O. Input-output.

L

LVDS. Low voltage differential signaling.

M

Multicast. The concept of sending a packet to more than one processing elements in a system.

N

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.

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.

S

SECDED. Single error correction, double error detection.

Sender. The RapidIO interface output port on a processing element.

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.

RapidIO™ Interconnect Specification Part 5: Globally Shared Memory Logical Specification

3.2, 1/2016

RapidIO.org

Revision History

Revision	Description	Date
1.1	Incorporated comment review changes.	03/08/2001
1.2	Technical changes: incorporate Rev. 1.1 errata rev. 1.1.1, errata 3	06/26/2002
1.3	Technical changes: incorporate Rev 1.2 errata 1 as applicable Converted to ISO-friendly templates	02/23/2005
2.0	No technical changes.	06/14/2007
2.1	No technical changes.	07/09/2009
2.2	Technical changes: errata showing: 10-08-00001.005, Consolidated Comments on 11-01-00000.000	05/05/2011
3.0	Changed RTA contact information. No technical changes	10/11/2013
3.1	No technical changes.	09/18/2014
3.2	No technical changes.	01/28/2016

Table of Contents

Chapter 1 Overview

1.1 Introduction 11
1.2 Overview 11
1.2.1 Memory System 12
1.3 Features of the Globally Shared Memory Specification 13
1.3.1 Functional Features 13
1.3.2 Physical Features 14
1.3.3 Performance Features 14
1.4 Contents 14
1.5 Terminology 15
1.6 Conventions 15

Chapter 2 System Models

2.1 Introduction 17
2.2 Processing Element Models 17
2.2.1 Processor-Memory Processing Element Model 18
2.2.2 Integrated Processor-Memory Processing Element Model 19
2.2.3 Memory-Only Processing Element Model 19
2.2.4 Processor-Only Processing Element 20
2.2.5 I/O Processing Element 20
2.2.6 Switch Processing Element 20
2.3 Programming Models 21
2.3.1 Globally Shared Memory System Model 21
2.3.1.1 Software-Managed Cache Coherence Programming Model 23
2.4 System Issues 23
2.4.1 Operation Ordering 23
2.4.2 Transaction Delivery 23
2.4.3 Deadlock Considerations 24

Chapter 3 Operation Descriptions

3.1 Introduction 25
3.2 GSM Operations Cross Reference 26
3.3 GSM Operations 27
3.3.1 Read Operations 28
3.3.2 Instruction Read Operations 29
3.3.3 Read-for-Ownership Operations 31
3.3.4 Data Cache Invalidate Operations 33
3.3.5 Castout Operations 34
3.3.6 TLB Invalidate-Entry Operations 35

Table of Contents

3.3.7 TLB Invalidate-Entry Synchronization Operations 35
3.3.8 Instruction Cache Invalidate Operations 35
3.3.9 Data Cache Flush Operations 36
3.3.10 I/O Read Operations 38
3.4 Endian, Byte Ordering, and Alignment 40

Chapter 4 Packet Format Descriptions

4.1 Introduction 43
4.2 Request Packet Formats 43
4.2.1 Addressing and Alignment 44
4.2.2 Data Payloads 44
4.2.3 Field Definitions for All Request Packet Formats 47
4.2.4 Type 0 Packet Format (Implementation-Defined) 49
4.2.5 Type 1 Packet Format (Intervention-Request Class) 50
4.2.6 Type 2 Packet Format (Request Class) 51
4.2.7 Type 3–4 Packet Formats (Reserved) 52
4.2.8 Type 5 Packet Format (Write Class) 52
4.2.9 Type 6–11 Packet Formats (Reserved) 53
4.3 Response Packet Formats 53
4.3.1 Field Definitions for All Response Packet Formats 53
4.3.2 Type 12 Packet Format (Reserved) 54
4.3.3 Type 13 Packet Format (Response Class) 54
4.3.4 Type 14 Packet Format (Reserved) 54
4.3.5 Type 15 Packet Format (Implementation-Defined) 55

Chapter 5 Globally Shared Memory Registers

5.1 Introduction 57
5.2 Register Summary 57
5.3 Reserved Register, Bit and Bit Field Value Behavior 58
5.4 Capability Registers (CARs) 60
5.4.1 Source Operations CAR 60
5.4.2 Destination Operations CAR 61
5.5 Command and Status Registers (CSRs) 62

Chapter 6 Communication Protocols

6.1 Introduction 63
6.2 Definitions 63
6.2.1 General Definitions 64
6.2.2 Request and Response Definitions 66
6.2.2.1 System Request 66
6.2.2.2 Local Request 66
6.2.2.3 System Response 67

Table of Contents

6.2.2.4 Local Response 67
6.3 Operation to Protocol Cross Reference 67
6.4 Read Operations 68
6.4.1 Internal Request State Machine 68
6.4.2 Response State Machine 68
6.4.3 External Request State Machine 70
6.5 Instruction Read Operations 72
6.5.1 Internal Request State Machine 72
6.5.2 Response State Machine 72
6.5.3 External Request State Machine 73
6.6 Read for Ownership Operations 75
6.6.1 Internal Request State Machine 75
6.6.2 Response State Machine 75
6.6.3 External Request State Machine 78
6.7 Data Cache and Instruction Cache Invalidate Operations 79
6.7.1 Internal Request State Machine 79
6.7.2 Response State Machine 79
6.7.3 External Request State Machine 80
6.8 Castout Operations 82
6.8.1 Internal Request State Machine 82
6.8.2 Response State Machine 82
6.8.3 External Request State Machine 82
6.9 TLB Invalidate Entry, TLB Invalidate Entry Synchronize Operations 83
6.9.1 Internal Request State Machine 83
6.9.2 Response State Machine 83
6.9.3 External Request State Machine 83
6.10 Data Cache Flush Operations 84
6.10.1 Internal Request State Machine 84
6.10.2 Response State Machine 84
6.10.3 External Request State Machine 86
6.11 I/O Read Operations 88
6.11.1 Internal Request State Machine 88
6.11.2 Response State Machine 88
6.11.3 External Request State Machine 89

Chapter 7 Address Collision Resolution Tables

7.1 Introduction 91
7.2 Resolving an Outstanding READ_HOME Transaction 92
7.3 Resolving an Outstanding IREAD_HOME Transaction 93
7.4 Resolving an Outstanding READ_OWNER Transaction 94
7.5 Resolving an Outstanding READ_TO_OWN_HOME Transaction 95
7.6 Resolving an Outstanding READ_TO_OWN_OWNER Transaction 97
7.7 Resolving an Outstanding DKILL_HOME Transaction 98
7.8 Resolving an Outstanding DKILL_SHARER Transaction 100
7.9 Resolving an Outstanding IKILL_HOME Transaction 101

Table of Contents

7.10 Resolving an Outstanding IKILL_SHARER Transaction 102
7.11 Resolving an Outstanding CASTOUT Transaction.. 103
7.12 Resolving an Outstanding TLBIE or TLBSYNC Transaction 104
7.13 Resolving an Outstanding FLUSH Transaction 105
7.14 Resolving an Outstanding IO_READ_HOME Transaction 107
7.15 Resolving an Outstanding IO_READ_OWNER Transaction 109

List of Figures

1-1 A Snoopy Bus-Based System 12
1-2 A Distributed Memory System 13
2-1 A Possible RapidIO-Based Computing System....17
2-2 Processor-Memory Processing Element Example 18
2-3 Integrated Processor-Memory Processing Element Example....19
2-4 Memory-Only Processing Element Example 19
2-5 Processor-Only Processing Element Example....20
2-6 Switch Processing Element Example 21
3-1 Read Operation to Remote Shared Coherence Granule....28
3-2 Read Operation to Remote Modified Coherence Granule....28
3-3 Read Operation to Local Modified Coherence Granule 29
3-4 Instruction Read Operation to Remote Shared Coherence Granule 30
3-5 Instruction Read Operation to Remote Modified Coherence Granule 30
3-6 Instruction Read Operation to Local Modified Coherence Granule....30
3-7 Instruction Read Operation Paradox Case 31
3-8 Read-for-Ownership Operation to Remote Shared Coherence Granule....31
3-9 Read-for-Ownership Operation to Remote Modified Coherence Granule....32
3-10 Read-for-Ownership Operation to Local Shared Coherence Granule 32
3-11 Read-for-Ownership Operation to Local Modified Coherence Granule 32
3-12 Data Cache Invalidate Operation to Remote Shared Coherence Granule 33
3-13 Data Cache Invalidate Operation to Local Shared Coherence Granule....34
3-14 Castout Operation on Remote Modified Coherence Granule....34
3-15 TLB Invalidate-Entry Operation....35
3-16 TLB Invalidate-Entry Synchronization Operation 35
3-17 Instruction Cache Invalidate Operation to Remote Sharable Coherence Granule....36
3-18 Instruction Cache Invalidate Operation to Local Sharable Coherence Granule....36
3-19 Flush Operation to Remote Shared Coherence Granule....37
3-20 Flush Operation to Remote Modified Coherence Granule 38
3-21 Flush Operation to Local Shared Coherence Granule 38
3-22 Flush Operation to Local Modified Coherence Granule 38
3-23 I/O Read Operation to Remote Shared Coherence Granule 39
3-24 I/O Read Operation to Remote Modified Coherence Granule 39
3-25 I/O Read Operation to Local Modified Coherence Granule....39
3-26 Byte Alignment Example....40
3-27 Half-Word Alignment Example....40
3-28 Word Alignment Example 40
3-29 Data Alignment Example....41
4-1 Type 1 Packet Bit Stream Format....50
4-2 Type 2 Packet Bit Stream Format....52
4-3 Type 5 Packet Bit Stream Format....52
4-4 Type 13 Packet Bit Stream Format....54

List of Figures

Blank page

List of Tables

2-1 RapidIO Memory Directory Definition 22
3-1 GSM Operations Cross Reference 26
4-1 Request Packet Type to Transaction Type Cross Reference 43
4-2 Coherent 32-Byte Read Data Return Ordering 45
4-3 Coherent 64-Byte Read Data Return Ordering 45
4-4 Coherent 32-Byte Write Data Payload 46
4-5 Coherent 64-Byte Write Data Payloads 46
4-6 General Field Definitions for All Request Packets 47
4-7 Read Size (rdsize) Definitions 48
4-8 Write Size (wrsize) Definitions 49
4-9 Specific Field Definitions and Encodings for Type 1 Packets 50
4-10 Transaction Field Encodings for Type 2 Packets 51
4-11 Transaction Field Encodings for Type 5 Packets 52
4-12 Request Packet Type to Transaction Type Cross Reference 53
4-13 Field Definitions and Encodings for All Response Packets 53
5-1 GSM Register Map 57
5-2 Configuration Space Reserved Access Behavior 58
5-3 Bit Settings for Source Operations CAR 60
5-4 Bit Settings for Destination Operations CAR 61
6-1 Operation to Protocol Cross Reference 67
7-1 Address Collision Resolution for READ_HOME 92
7-2 Address Collision Resolution for IREAD_HOME 93
7-3 Address Collision Resolution for READ_OWNER 94
7-4 Address Collision Resolution for READ_TO_OWN_HOME 95
7-5 Address Collision Resolution for READ_TO_OWN_OWNER 97
7-6 Address Collision Resolution for DKILL_HOME 98
7-7 Address Collision Resolution for DKILL_SHARER 100
7-8 Address Collision Resolution for IKILL_HOME 101
7-9 Address Collision Resolution for IKILL_SHARER 102
7-10 Address Collision Resolution for CASTOUT 103
7-11 Address Collision Resolution for Software Coherence Operations 104
7-12 Address Collision Resolution for Participant FLUSH 105
7-13 Address Collision Resolution for Non-participant FLUSH 106
7-14 Address Collision Resolution for Participant IO_READ_HOME 107
7-15 Address Collision Resolution for Non-participant IO_READ_HOME 108
7-16 Address Collision Resolution for Participant IO_READ_OWNER 109
7-17 Address Collision Resolution for Non-participant IO_READ_OWNER 110

List of Tables

Blank page

Chapter 1 Overview

1.1 Introduction

This chapter provides an overview of the *RapidIO Part 5: Globally Shared Memory Logical Specification*, including a description of the relationship between this specification and the other specifications of the RapidIO interconnect.

1.2 Overview

Although RapidIO is targeted toward the message passing programming model, it supports a globally shared distributed memory (GSM) model as defined by this specification. The globally shared memory programming model is the preferred programming model for modern general-purpose multiprocessing computer systems, which requires cache coherency support in hardware. This addition of GSM enables both distributed I/O processing and general purpose multiprocessing to co-exist under the same protocol.

The *RapidIO Part 5: Globally Shared Memory Logical Specification* is one of the RapidIO logical layer specifications that define the interconnect's overall protocol and packet formats. This layer contains the information necessary for end points to process a transaction. Other RapidIO logical layer specifications include *RapidIO Part 1: Input/Output Logical Specification* and *RapidIO Part 2: 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. Necessary bits are added to the logical encodings for the transport and physical layers lower in the specification hierarchy.

RapidIO is a definition of a system interconnect. System concepts such as processor programming models, memory coherency models and caching are beyond the scope of the RapidIO architecture. The support of memory coherency models, through caches, memory directories (or equivalent, to hold state and speed up remote memory access) is the responsibility of the end points (processors, memory, and possibly I/O devices), using RapidIO operations. RapidIO provides the operations to construct a wide variety of systems, based on programming models that range from strong consistency through total store ordering to weak ordering. Inter-operability between end points supporting different coherency/caching/directory models is not guaranteed. However, groups of end-points with conforming models can be linked to others conforming to different models on the same RapidIO fabric. These different groups can communicate through RapidIO messaging or I/O operations. Any reference to these areas within the RapidIO architecture specification are for illustration only.

The *RapidIO Interconnect Globally Shared Memory Logical Specification* assumes that the reader is familiar with the concepts and terminology of cache coherent systems in general and with CC-NUMA systems in specific. Further information on shared memory concepts can be found in:

Daniel E. Lenoski and Wolf-Dietrich Weber, “Scalable Shared-Memory Multiprocessing”, Morgan Kaufmann, 1995.

and

David Culler, Jaswinder Pal Singh, and Anoop Gupta: “Parallel Computer Architecture: A Hardware/Software Approach”, Morgan Kaufmann, 1998

1.2.1 Memory System

Under the globally shared distributed memory programming model, memory may be physically located in different places in the machine yet may be shared amongst different processing elements. Typically, mainstream system architectures have addressed shared memory using transaction broadcasts sometimes known as bus-based snoopy protocols. These are usually implemented through a centralized memory controller for which all devices have equal or uniform access. Figure 1-1 shows a typical bus-based shared memory system.

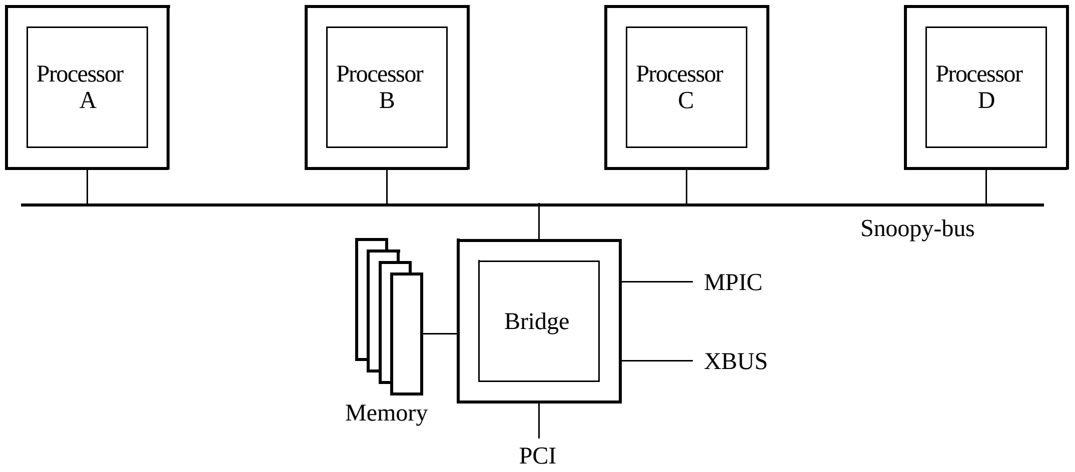


Figure 1-1. A Snoopy Bus-Based System

Super computers, massively parallel, and clustered machines that have distributed memory systems must use a different technique from broadcasting for maintaining memory coherency. Because a broadcast snoopy protocol in these machines is not efficient given the number of devices that must participate and the latency and transaction overhead involved, coherency mechanisms such as memory directories or distributed linked lists are required to keep track of where the most current copy of data resides. These schemes are often referred to as cache coherent non-uniform memory access (CC-NUMA) protocols. A typical distributed memory system architecture is shown in Figure 1-2.

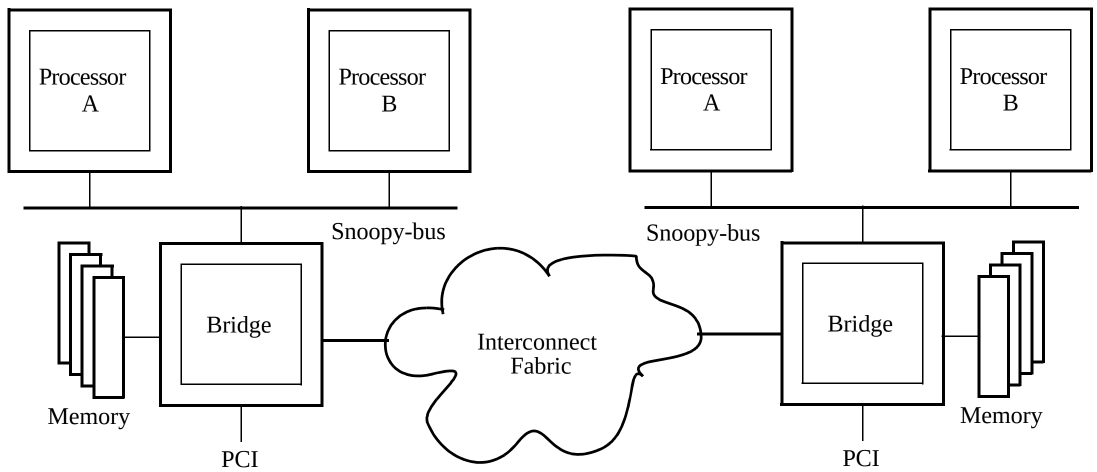


Figure 1-2. A Distributed Memory System

For RapidIO, a relatively simple directory-based coherency scheme is chosen. For this method each memory controller is responsible for tracking where the most current copy of each data element resides in the system. RapidIO furnishes a variety of ISA specific cache control and operating system support operations such as block flushes and TLB synchronization mechanisms.

To reduce the directory overhead required, the architecture is optimized around small clusters of 16 processors known as coherency domains. With the concept of domains, it is possible for multiple coherence groupings to coexist in the interconnect as tightly coupled processing clusters.

1.3 Features of the Globally Shared Memory Specification

The following are features of the RapidIO GSM specification designed to satisfy the needs of various applications and systems:

1.3.1 Functional Features

- A cache coherent non-uniform memory access (CC-NUMA) system architecture is supported to provide a globally shared memory model because physics is forcing component interfaces in many high-speed designs to be point-to-point instead of traditional bus-based.
- The size of processor memory requests are either in the cache coherence granularity, or smaller. The coherence granule size may be different for different processor families or implementations.

- Instruction sets in RapidIO support a variety of cache control and other operations such as block flushes. These functions are supported to run legacy applications and operating systems.

1.3.2 Physical Features

- RapidIO packet definition is independent of the width of the physical interface to other devices on the interconnect fabric.
- The protocols and packet formats are independent of the physical interconnect topology. The protocols work whether the physical fabric is a point-to-point ring, a bus, a switched multi-dimensional network, a duplex serial connection, and so forth.
- RapidIO is not dependent on the bandwidth or latency of the physical fabric.
- The protocols handle out-of-order packet transmission and reception.
- Certain devices have bandwidth and latency requirements for proper operation. RapidIO does not preclude an implementation from imposing these constraints within the system.

1.3.3 Performance Features

- Packet headers must be as small as possible to minimize the control overhead and be organized for fast, efficient assembly and disassembly.
- 48- and 64-bit addresses are required in the future, and must be supported initially.
- An interventionist (non-memory owner, direct-to-requestor data transfer, analogous to a cache-to-cache transfer) protocol saves a large amount of latency for memory accesses that cause another processing element to provide the requested data.
- Multiple transactions must be allowed concurrently in the system, otherwise a majority of the potential system throughput is wasted.

1.4 Contents

Following are the contents of the *RapidIO Interconnect Globally Shared Memory Logical Specification:*

- Chapter 1, “Overview,” describes the set of operations and transactions supported by the RapidIO globally shared memory protocols.
- Chapter 2, “System Models,” introduces some possible devices that could participate in a RapidIO GSM system environment. The chapter explains the memory directory-based mechanism that tracks memory accesses and maintains cache coherence. Transaction ordering and deadlock prevention are also covered.

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

1.5 Terminology

Refer to the Glossary at the back of this document.

1.6 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.
x	This value is a don’t care

Chapter 2 System Models

2.1 Introduction

This overview introduces some possible devices in a RapidIO system.

2.2 Processing Element Models

Figure 2-1 describes a possible RapidIO-based computing system. The processing element is a computer device such as a processor attached to a local memory and also attached to a RapidIO system interconnect. The bridge part of the system provides I/O subsystem services such as high-speed PCI interfaces and gigabit ethernet ports, interrupt control, and other system support functions. Multiple processing elements require cache coherence support in the RapidIO protocol to preserve the traditional globally shared memory programming model (discussed in Section 2.3.1, “Globally Shared Memory System Model”).

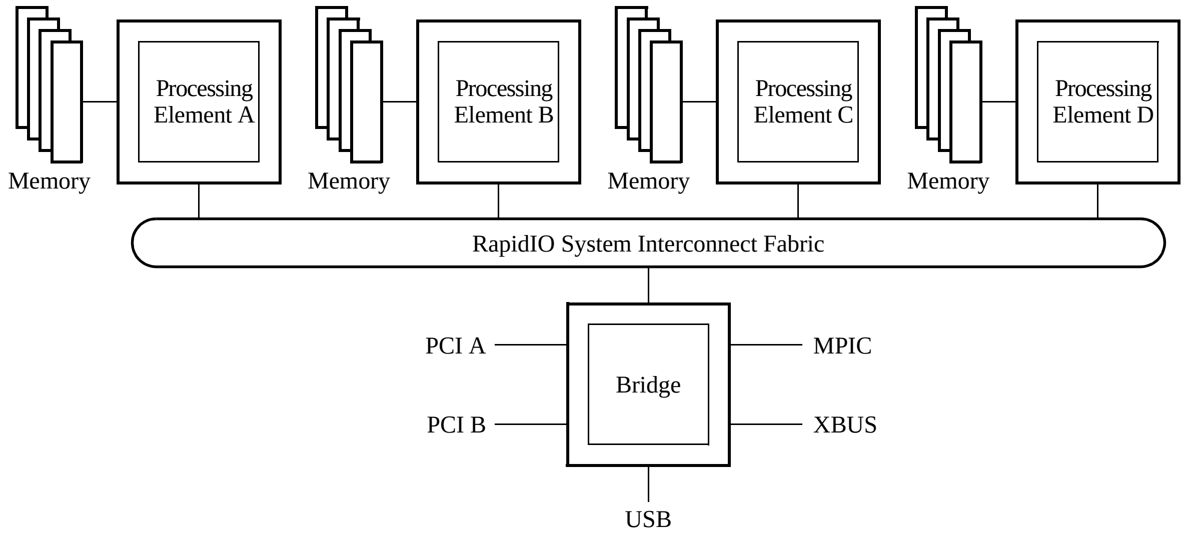


Figure 2-1. A Possible RapidIO-Based Computing System

A processing element containing a processor typically has associated with it a caching hierarchy to improve system performance. The RapidIO protocol supports a set of operations sufficient to fulfill the requirements of a processor with a caching hierarchy and associated support logic such as a processing element.

RapidIO is defined so that many types of devices can be designed for specific applications and connected to the system interconnect. These devices may participate in the cache coherency protocol, act as a DMA device, utilize the message passing facilities to communicate with other devices on the interconnect, and so forth. A bridge could be designed, for example, to use the message passing facility to pass ATM packets to and from a processing element for route processing. The following sections describe several possible processing elements.

2.2.1 Processor-Memory Processing Element Model

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

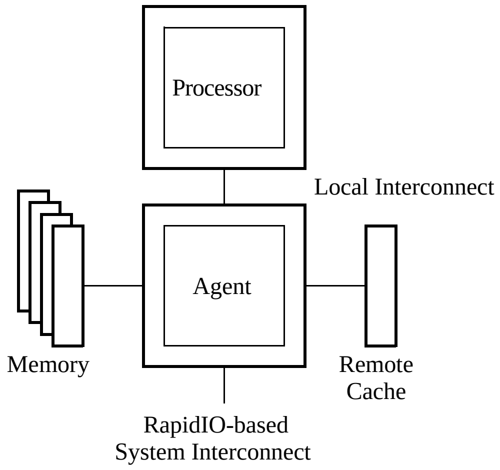


Figure 2-2. Processor-Memory Processing Element Example

In support of the remote accesses, the agent maintains a cache of remote accesses that includes all remote data currently residing in and owned by the local processor. This cache may be either external or internal to the agent device.

Agent caching is necessary due to the construction of the RapidIO cache coherence protocol combined with the cache hierarchy behavior in modern processors. Many modern processors have multiple level non-inclusive caching structures that are maintained independently. This implies that when a coherence granule is cast out of the processor, it may or may not be returning ownership of the granule to the memory system. The RapidIO protocol requires that ownership of a coherence granule be guaranteed to be returned to the system on demand and without ambiguous cache state changes as with the castout behavior. The remote cache can guarantee that a coherence granule requested by the system is owned locally and can be returned to the home memory (the physical memory containing the coherence granule) on demand. A processing element that is fully integrated would also need to support this behavior.

2.2.2 Integrated Processor-Memory Processing Element Model

Another form of a processor-memory processing element is a fully integrated component that is designed specifically to connect to a RapidIO interconnect system as shown in Figure 2-3. This type of device integrates a memory system and other support logic with a processor on the same piece of silicon or within the same package. Because such a device is designed specifically for RapidIO, a remote cache is not required because the proper support can be designed into the processor and its associated logic rather than requiring an agent to compensate for a stand alone processor’s behavior.

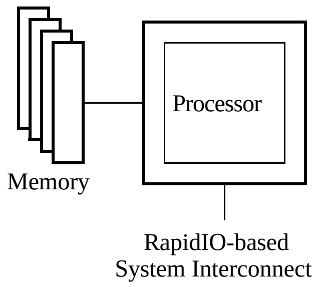


Figure 2-3. Integrated Processor-Memory Processing Element Example

2.2.3 Memory-Only Processing Element Model

A different processing element may not contain a processor at all, but may be a memory-only device as in Figure 2-4. This type of device is much simpler than a processor as it is only responsible for responding to requests from the external system, not from local requests as in the processor-based model. As such, its memory is remote for all processors in the system.

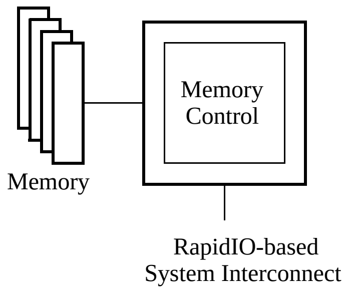


Figure 2-4. Memory-Only Processing Element Example

2.2.4 Processor-Only Processing Element

Similar to a memory-only element, a processor-only element has no local memory. A processor-only processing element is shown in Figure 2-5.

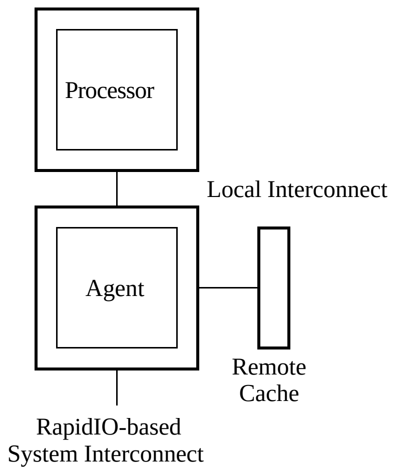


Figure 2-5. Processor-Only Processing Element Example

2.2.5 I/O Processing Element

This type of processing element is shown as the bridge in Figure 2-1. This device has distinctly different behavior than a processor or a memory. An I/O device only needs to move data into and out of local or remote memory in a cache coherent fashion. This means that if the I/O device needs to read from memory, it only needs to obtain a known good copy of the data to write to the external device (such as a disk drive or video display). If the I/O device needs to write to memory, it only needs to get ownership of the coherence granule returned to the home memory and not take ownership for itself. Both of these operations have special support in the RapidIO protocol.

2.2.6 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 2-6.

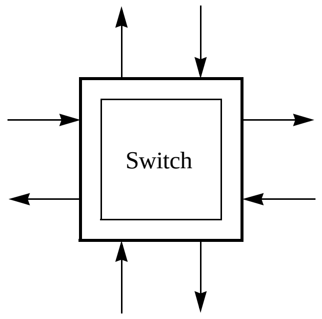


Figure 2-6. Switch Processing Element Example

2.3 Programming Models

RapidIO supports applications developed under globally shared memory and software-managed cache coherence programming models.

2.3.1 Globally Shared Memory System Model

The preferred programming model for modern computer systems provides memory that is accessible from all processors in a cache coherent fashion. This model is also known as GSM, or globally shared memory. For traditional bus-based computer systems this is not a difficult technical problem to solve because all participants in the cache coherence mechanism see all memory activity simultaneously, meaning that communication between processors is very fast and handled without explicit software control. However, in a non-uniform memory access system, this simultaneous memory access visibility is not the case.

With a distributed memory system, cache coherence needs to be maintained through some tracking mechanism that keeps records of memory access activity and explicitly notifies specific cache coherence participant processing elements when a cache coherence hazard is detected. For example, if a processing element wishes to write to a memory address, all participant processing elements that have accessed that coherence granule are notified to invalidate that address in their caches. Only when all of the participant processing elements have completed the invalidate operation and replied back to the tracking mechanism is the write allowed to proceed.

The tracking mechanism preferred for the RapidIO protocol is the memory directory based system model. This system model allows efficient, moderate scalability with a reasonable amount of information storage required for the tracking mechanism.

Cache coherence is defined around the concept of domains. The RapidIO protocol assumes a memory directory based cache coherence mechanism. Because the storage requirements for the directory can be high, the protocol was optimized assuming a 16-participant domain size as a reasonable coherence scalability limit. With this limit in mind, a moderately scalable system of 16 participants can be designed, possibly using a multicast mechanism in the transport layer for better efficiency. This size does not limit a system designer from defining a larger or a smaller coherent system such as the four processing element system in Figure 2-1 on page 17 since the number of domains and the number of participants is flexible. The total number of coherence domains and the scalability limit are determined by the number of transport bits allowed by the appropriate transport layer specification.

Table 2-1 describes an example of the directory states assumed for the RapidIO protocol for a small four-processing element cache coherent system (the table assumes that processor 0 is the local processor). Every coherence granule that is accessible by a remote processing element has this 4-bit field associated with it, so some state storage is required for each globally shared granule. The least significant bit (the right most, bit 3) indicates that a processing element has taken ownership of a coherence granule. The remaining three bits indicate that processing elements have accessed that coherence granule, or the current owner if the granule has been modified, with bit 0 corresponding to processor 3, bit 1 corresponding to processor 2, and bit 2 corresponding to processor 1. These bits are also known as the sharing mask or sharing list.

Owing to the encoding of the bits, the local processing element is always assumed to have accessed the granule even if it has not. This definition allows us to know exactly which processing elements have participated in the cache coherency protocol for each shared coherence granule at all times. Other state definitions can be implemented as long as they encompass the MSL (modified, shared, local) state functionality described here.

Table 2-1. RapidIO Memory Directory Definition

State	Description
0000	Processor 0 (local) shared
0001	Processor 0 (local) modified
0010	Processor 1, 0 shared
0011	Processor 1 modified
0100	Processor 2, 0 shared
0101	Processor 2 modified
0110	Processor 2, 1, 0 shared
0111	Illegal
1000	Processor 3, 0 shared
1001	Processor 3 modified
1010	Processor 3, 1, 0 shared
1011	Illegal
1100	Processor 3, 2, 0 shared
1101	Illegal

Table 2-1. RapidIO Memory Directory Definition (Continued)

1110	Processor 3, 2, 1, 0 shared
1111	Illegal

When a coherence granule is referenced, the corresponding 4-bit coherence state is examined by the memory controller to determine if the access can be handled in memory, or if data must be obtained from the current owner (a shared granule is owned by the home memory). Coherence activity in the system is started using the cache coherence protocol, if it is necessary to do so, to complete the memory operation.

2.3.1.1 Software-Managed Cache Coherence Programming Model

The software-managed cache coherence programming model depends upon the application programmer to guarantee that the same coherence granule is not resident in more than one cache in the system simultaneously if it is possible for that coherence granule to be written by one of the processors. The application software allows sharing of written data by using cache manipulation instructions to flush these coherence granules to memory before they are read by another processor. This programming model is useful in transaction and distributed processing types of systems.

2.4 System Issues

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

2.4.1 Operation Ordering

Operation completion ordering in a globally shared memory system is managed by the completion units of the processing elements participating in the coherence protocol and by the coherence protocol itself.

2.4.2 Transaction Delivery

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

2.4.3 Deadlock Considerations

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

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

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

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

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

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

A further consideration is that of the read-for-ownership operation colliding with a castout of the requested memory address by another processing element. In order for the read-for-ownership operation to complete the underlying castout operation must complete. Therefore the castout must be given higher preference in the system in order to move ahead of other operations in order to break up the dependency.

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

Chapter 3 Operation Descriptions

3.1 Introduction

This chapter describes the set of operations and transactions supported by the RapidIO globally-shared memory (GSM) protocols. The opcodes and packet formats are described in Chapter 4, “Packet Format Descriptions.” The complete protocols are described in Chapter 6, “Communication Protocols.”

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

A number of possible response transactions can be received by a requesting processing element:

- A DONE response indicates to the requestor that the desired transaction has completed and also returns data for read-type transactions as described above.
- The INTERVENTION, DONE_INTERVENTION, and DATA_ONLY responses are generated as part of the processing element-to-processing element (as opposed to processing element-to-home memory) data transfer mechanism defined by the cache coherence protocol. The INTERVENTION and DONE_INTERVENTION responses are abbreviated as INTERV and DONE_INTERV in this chapter.
- The NOT_OWNER and RETRY responses are received when there are address conflicts within the system that need resolution.
- An ERROR response means that the target of the transaction encountered an unrecoverable error and could not complete the transaction.

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

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

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

The transaction behaviors are also described as state machine behavior in Chapter 6, “Communication Protocols”.

3.2 GSM Operations Cross Reference

Table 3-1 contains a cross reference of the GSM operations defined in this RapidIO specification and their system usage.

Table 3-1. GSM Operations Cross Reference

Operation	Transactions Used	Possible System Usage	Description	Packet Format	Protocol
Read	READ_HOME, READ_OWNER, RESPONSE	CC-NUMA operation	Section 3.3.1 page 28	Types 1 and 2: Section 4.2.5 page 50 and Section 4.2.6 page 51	Section 6.4 page 68
Instruction read	IREAD_HOME, READ_OWNER, RESPONSE	Combination of CC-NUMA and software-maintained coherence of instruction caches	Section 3.3.2 page 29	Type 2 Section 4.2.6 page 51	Section 6.4 page 68
Read-for-ownership	READ_TO_OWN_HOME, READ_TO_OWN_OWNER, DKILL_SHARER RESPONSE	CC-NUMA operation	Section 3.3.3 page 31	Types 1 and 2: Section 4.2.5 page 50 and Section 4.2.6 page 51	Section 6.6 page 75
Data cache invalidate	DKILL_HOME, DKILL_SHARER, RESPONSE	CC-NUMA operation; software-maintained coherence operation	Section 3.3.4 page 33	Type 2 Section 4.2.6 page 51	Section 6.7 page 79
Castout	CASTOUT, RESPONSE	CC-NUMA operation	Section 3.3.5 page 34	Type 5 Section 4.2.8 page 52	Section 6.8 page 82

Table 3-1. GSM Operations Cross Reference (Continued)

Operation	Transactions Used	Possible System Usage	Description	Packet Format	Protocol
TLB invalidate-entry	TLBIE, RESPONSE	Software-maintained coherence of page table entries	Section 3.3.6	Type 2 Section 4.2.6	Section 6.9
TLB invalidate-entry synchronize	TLBSYNC, RESPONSE	Software-maintained coherence of page table entries	Section 3.3.7	Type 2 Section 4.2.6	Section 6.9
Instruction cache invalidate	IKILL_HOME, IKILL_SHARER, RESPONSE,	Software-maintained coherence of instruction caches	Section 3.3.8	Type 2 Section 4.2.6	Section 6.7

Data cache flush	FLUSH, DKILL_SHARER, READ_TO_OWN_OWNER, RESPONSE	CC-NUMA flush instructions;
		CC-NUMA write-through cache support;
		CC-NUMA DMA I/O device support;
		software-maintained coherence operation.
			Section 3.3.9	Types 2 and 5:
				Section 4.2.6 and
				Section 4.2.8
					Section 6.10

I/O read	IO_READ_HOME, IO_READ_OWNER, INTERV, RESPONSE	CC-NUMA DMA, I/O DMA device support	Section 3.3.10	Types 1 and 2: Section 4.2.5 and Section 4.2.6	Section 6.11

3.3 GSM Operations

A set of transactions are used to support GSM (cache coherence) operations to cacheable memory space. The following descriptions assume that all requests are to system memory rather than to some other type of device.

GSM operations occur based on the size of the coherence granule. Changes in the coherence granule for a system do not change any of the operation protocols, only the data payload size. The only exceptions to this are flush and I/O read operations, which may request (in the case of an I/O read), or have (in the case of a flush) a sub-coherence granule to support coherent I/O and write-through caches. Flush operations may also have no data payload in order to support cache manipulation instructions.

Some transactions are sent to multiple recipients in the process of completing an operation. These transactions can be sent either as a number of directed transactions or as a single transaction if the transport layer has multicast capability. Multicast capability and operation is defined in the appropriate RapidIO transport layer specification.

3.3.1 Read Operations

The READ_HOME, READ_OWNER, and RESPONSE transactions are used during a read operation by a processing element that needs a shared copy of cache-coherent data from the memory system. A read operation always returns one coherence granule-sized data payload.

The READ_HOME transaction is used by a processing element that needs to read a shared copy of a coherence granule from a remote home memory on another processing element.

The READ_OWNER transaction is used by a home memory processing element that needs to read a shared copy of a coherence granule that is owned by a remote processing element.

The following types of read operations are possible:

- If the requested data exists in the memory directory as shared, the data can be returned immediately from memory with a DONE RESPONSE transaction and the requesting processing element's device ID is added to the sharing mask as shown in Figure 3-1.

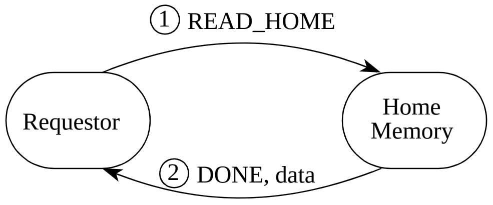


Figure 3-1. Read Operation to Remote Shared Coherence Granule

- If the requested data exists in the memory directory as modified, the up-to-date (current) data must be obtained from the owner. The home memory then sends a READ_OWNER request to the processing element that owns the coherence granule. The owner passes a copy of the data to the original requestor and to memory, memory is updated, and the directory state is changed from modified and owner to shared by the previous owner and the requesting processing element's device ID as shown in Figure 3-2.

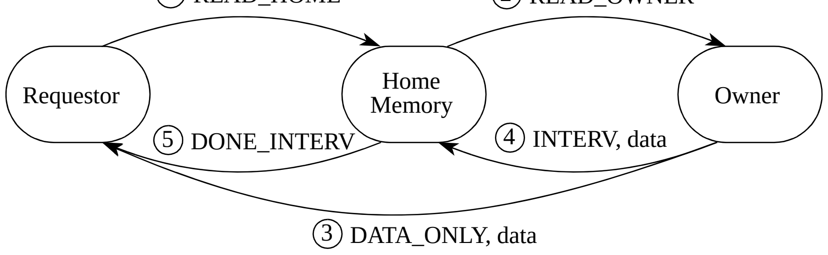


Figure 3-2. Read Operation to Remote Modified Coherence Granule

- If the processing element requesting a modified coherence granule happens to be the home for the memory, some of the transactions can be eliminated as shown in Figure 3-3.

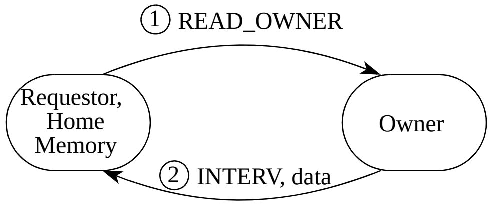


Figure 3-3. Read Operation to Local Modified Coherence Granule

3.3.2 Instruction Read Operations

Some processors have instruction caches that do not participate in the system cache coherence mechanism. Additionally, the instruction cache load may also load a shared instruction and data cache lower in the cache hierarchy. This can lead to a situation where the instruction cache issues a shared read operation to the system for a coherence granule that is owned by that processor’s data cache, resulting in a cache coherence paradox to the home memory directory.

Due to this situation, an instruction read operation must behave like a coherent shared read relative to the memory directory and as a non-coherent operation relative to the requestor. Therefore, the behavior of the instruction read operation is nearly identical to a data read operation with the only difference being the way that the apparent coherence paradox is managed.

The IREAD_HOME and RESPONSE transactions are used during an instruction read operation by a processing element that needs a copy of sharable instructions from the memory system. An instruction read operation always returns one coherence granule-sized data payload. Use of the IREAD_HOME transaction rather than the READ_HOME transaction allows the memory directory to properly handle the paradox case without sacrificing coherence error detection in the system. The IREAD_HOME transaction participates in address collision detection at the home memory but does not participate in address collision detection at the requestor.

The following types of instruction read operations are possible:

- If the requested instructions exists in the memory directory as shared, the instructions can be returned immediately from memory and the requesting processing element’s device ID is added to the sharing mask as shown in Figure 3-4.

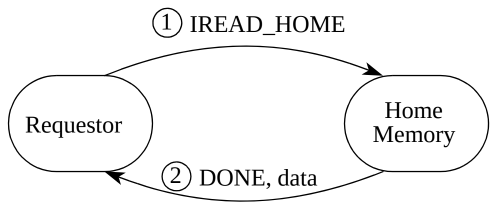


Figure 3-4. Instruction Read Operation to Remote Shared Coherence Granule

- If the requested data exists in the memory directory as modified, the up-to-date (current) data must be obtained from the owner. The home memory then sends a READ_OWNER request to the processing element that owns the coherence granule. The owner passes a copy of the data to the original requestor and to memory, memory is updated, and the directory state is changed from modified and owner to shared by the previous owner and the requesting processing element's device ID as shown in Figure 3-5.

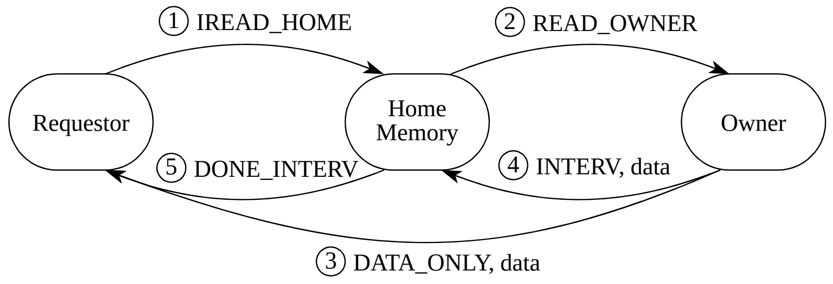


Figure 3-5. Instruction Read Operation to Remote Modified Coherence Granule

- If the processing element requesting a modified coherence granule happens to be the home for the memory the READ_OWNER transaction is used to obtain the coherence granule as shown in Figure 3-6.

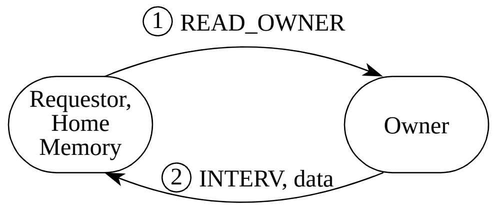


Figure 3-6. Instruction Read Operation to Local Modified Coherence Granule

- The apparent paradox case is if the requesting processing element is the owner of the coherence granule as shown in Figure 3-7. The home memory sends a READ_OWNER transaction back to the requesting processing element with the source and secondary ID set to the home memory ID, which indicates that the response behavior should be an INTERVENTION transaction rather than an INTERVENTION and a DATA_ONLY transaction as shown in Figure 3-5.

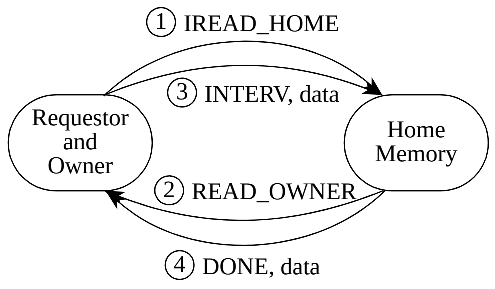


Figure 3-7. Instruction Read Operation Paradox Case

3.3.3 Read-for-Ownership Operations

The READ_TO_OWN_HOME, READ_TO_OWN_OWNER, DKILL_SHARER, and RESPONSE transactions are used during read-for-ownership operations by a processing element that needs to write to a coherence granule that does not exist in its caching hierarchy. A read-for-ownership operation always returns one coherence granule-sized data payload. These transactions are used as follows:

- The READ_TO_OWN_HOME transaction is used by a processing element that needs to read a writable copy of a coherence granule from a remote home memory on another processing element. This transaction causes a copy of the data to be returned to the requestor, from memory if the data is shared, or from the owner if it is modified.
- The READ_TO_OWN_OWNER transaction is used by a home memory processing element that needs to read a writable copy of a coherence granule that is owned by a remote processing element.
- The DKILL_SHARER transaction is used by the home memory processing element to invalidate shared copies of the coherence granule in remote processing elements.

Following are descriptions of the read-for-ownership operations:

- If the coherence granule is shared, DKILL_SHARER transactions are sent to the participants indicated in the sharing mask, which results in a cache invalidate operation for the recipients as shown in Figure 3-8.

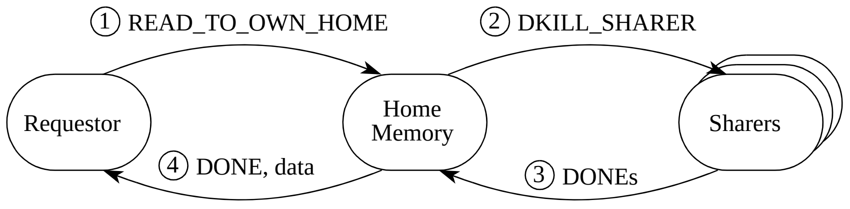


Figure 3-8. Read-for-Ownership Operation to Remote Shared Coherence Granule

- If the coherence granule is modified, a READ_TO_OWN_OWNER transaction is sent to the owner, who sends a copy of the data to the requestor (intervention) and marks the address as invalid as shown in Figure 3-9. The final memory directory state shows that the coherence granule is modified and owned by the requestor’s device ID.

Because the coherence granule in the memory directory was marked as modified, home memory does not necessarily need to be updated. However, the RapidIO protocol requires that a processing element return the modified data and update the memory, allowing some attempt for data recovery if a coherence problem occurs.

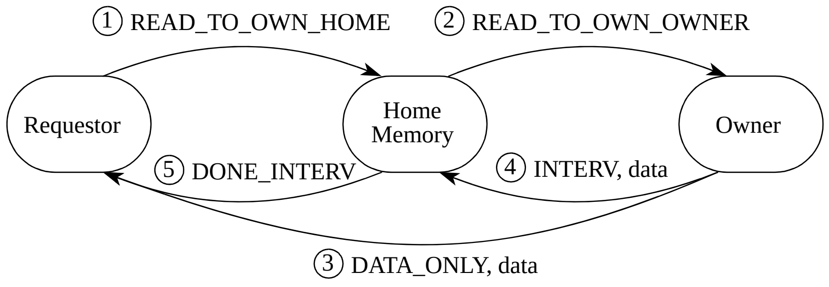


Figure 3-9. Read-for-Ownership Operation to Remote Modified Coherence Granule

- If the requestor is on the same processing element as the home memory and the coherence granule is shared, a DKILL_SHARER transaction is sent to all sharing processing elements (see Figure 3-10). The final directory state is marked as modified and owned by the local requestor.

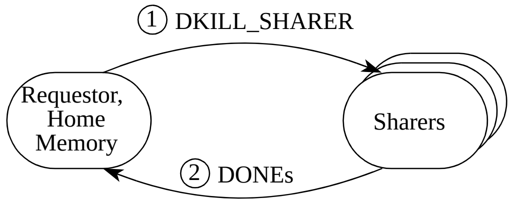


Figure 3-10. Read-for-Ownership Operation to Local Shared Coherence Granule

- If the requestor is on the same processing element as the home memory and the coherence granule is owned by a remote processing element, a READ_TO_OWN_OWNER transaction is sent to the owner (see Figure 3-11). The final directory state is marked as modified and owned by the local requestor.

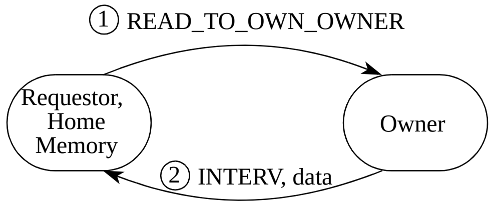


Figure 3-11. Read-for-Ownership Operation to Local Modified Coherence Granule

3.3.4 Data Cache Invalidate Operations

The DKILL_HOME, DKILL_SHARER, and RESPONSE transactions are requests to invalidate a coherence granule in all of the participants in the coherence domain as follows:

- The DKILL_HOME transaction is used by a processing element to invalidate a data coherence granule that has home memory in a remote processing element.
- The DKILL_SHARER transaction is used by the home memory processing element to invalidate shared copies of the data coherence granule in remote processing elements.

Data cache invalidate operations are also useful for systems that implement software-maintained cache coherence. In this case, a requestor may send DKILL_HOME and DKILL_SHARER transactions directly to other processing elements without going through home memory as in a CC-NUMA system. The transactions used for the data cache invalidate operation depend on whether the requestor is on the same processing element as the home memory of the coherence granule as follows:

- If the requestor is not on the same processing element as the home memory of the coherence granule, a DKILL_HOME transaction is sent to the remote home memory processing element. This causes the home memory for the shared coherence granule to send a DKILL_SHARER to all processing elements marked as sharing the granule in the memory directory state except for the requestor (see Figure 3-12). The final memory state shows that the coherence granule is modified and owned by the requesting processing element's device ID.

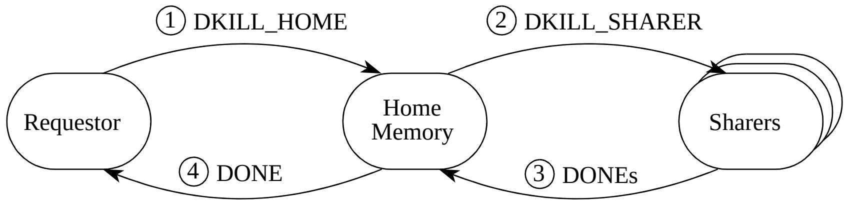


Figure 3-12. Data Cache Invalidate Operation to Remote Shared Coherence Granule

- If the requestor is on the same processing element as the home memory of the coherence granule, the home memory sends a DKILL_SHARER transaction to all processing elements marked as sharing the coherence granule in the memory directory. The final memory state shows the coherence granule modified and owned by the local processor (see Figure 3-13).

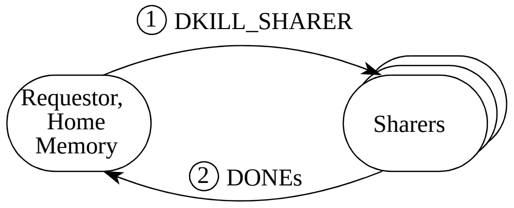


Figure 3-13. Data Cache Invalidate Operation to Local Shared Coherence Granule

3.3.5 Castout Operations

The CASTOUT and RESPONSE transactions are used in a castout operation by a processing element to relinquish its ownership of a coherence granule and return it to the home memory. The CASTOUT can be treated as a low-priority transaction unless there is an address collision with an incoming request, at which time it must become a high-priority transaction. The CASTOUT causes the home memory to be updated with the most recent data and changes the directory state to owned by home memory and shared (or owned, depending upon the default directory state) by the local processing element (see Figure 3-14).

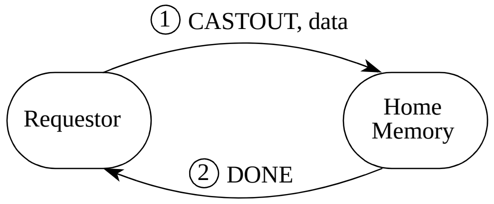


Figure 3-14. Castout Operation on Remote Modified Coherence Granule

A CASTOUT transaction does not participate in address collision detection at the home memory to prevent deadlocks or cache paradoxes caused by packet-to-packet timing in the interconnect fabric. For example, consider a case where processing element A is performing a CASTOUT that collides with an incoming READ_OWNER transaction. If the CASTOUT is not allowed to complete at the home memory, the system will deadlock. If the read operation that caused the READ_OWNER completes (through intervention) before the CASTOUT transaction is received at the home memory, the CASTOUT will appear to be illegal because the directory state will have changed.

3.3.6 TLB Invalidate-Entry Operations

The TLBIE and RESPONSE transactions are used for TLB invalidate-entry operations. If the processor TLBs do not participate in the cache coherence protocol, the TLB invalidate-entry operation is used when page table translation entries need to be modified. The TLBIE transaction is sent to all participants in the coherence domain except for the original requestor. A TLBIE transaction has no effect on the memory directory state for the specified address and does not participate in address collisions (see Figure 3-15).

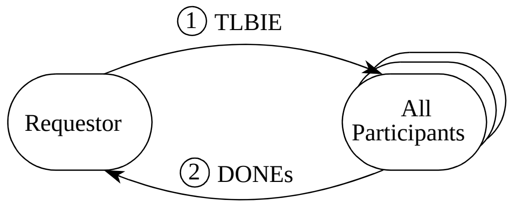


Figure 3-15. TLB Invalidate-Entry Operation

3.3.7 TLB Invalidate-Entry Synchronization Operations

The TLBSYNC and RESPONSE transactions are used for TLB invalidate-entry synchronization operations. It is used to force the completion of outstanding TLBIE transactions at the participants. The DONE response for a TLBSYNC transaction is only sent when all preceding TLBIE transactions have completed. This operation is necessary due to possible indeterminate completion of individual TLBIE transactions when multiple TLBIE transactions are being executed simultaneously. The TLBSYNC transaction is sent to all participants in the coherence domain except for the original requestor. The transaction has no effect on the memory directory state for the specified address and does not participate in address collisions (see Figure 3-16).

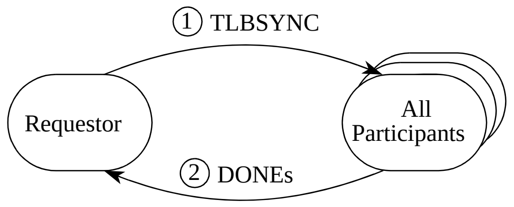


Figure 3-16. TLB Invalidate-Entry Synchronization Operation

3.3.8 Instruction Cache Invalidate Operations

The IKILL_HOME, IKILL_SHARER, and RESPONSE transactions are used during instruction cache invalidate operations to invalidate shared copies of an instruction coherence granule in remote processing elements. Instruction cache invalidate operations are needed if the processor instruction caches do not participate in the cache coherence protocol, requiring instruction cache coherence to be maintained by software.

An instruction cache invalidate operation has no effect on the memory directory state for the specified address and does not participate in address collisions. Following are descriptions of the instruction cache invalidate operations:

- If the requestor is not on the same processing element as the home memory of the coherence granule, an IKILL_HOME transaction is sent to the remote home memory processing element. This causes the home memory for the shared coherence granule to send an IKILL_SHARER to all processing element participants in the coherence domain because the memory directory state only properly tracks data, not instruction, accesses. (See Figure 3-17.)

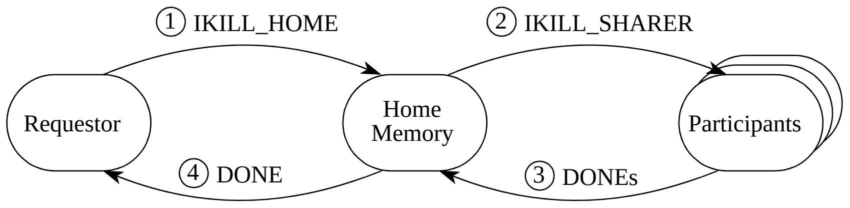


Figure 3-17. Instruction Cache Invalidate Operation to Remote Sharable Coherence Granule

- If the requestor is on the same processing element as the home memory of the coherence granule, the home memory sends an IKILL_SHARER transaction to all processing element participants in the coherence domain as shown in Figure 3-18.

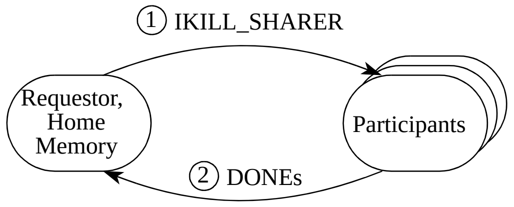


Figure 3-18. Instruction Cache Invalidate Operation to Local Sharable Coherence Granule

3.3.9 Data Cache Flush Operations

The FLUSH, DKILL_SHARER, READ_TO_OWN_OWNER, and RESPONSE transactions are used for data cache flush operations, which return ownership of a coherence granule back to the home memory if it is modified and invalidate all copies if the granule is shared. A flush operation with associated data can be used to implement an I/O system write operation and to implement processor write-through and cache manipulation operations. These transactions are used as follows:

- The FLUSH transaction is used by a processing element to return the ownership and current data of a coherence granule to home memory. The data payload for the FLUSH transaction is typically the size of the coherence granule for the system but may be multiple double-words or one double-word or less. FLUSH transactions without a data payload are used to support cache manipulation operations. The memory directory state is changed to owned by home memory and shared (or modified, depending upon the processing element's normal default state) by the local processing element.

- The DKILL_SHARER transaction is used by the home memory processing element to invalidate shared copies of the data coherence granule in remote processing elements.
- The READ_TO_OWN_OWNER transaction is used by a home memory processing element that needs to retrieve ownership of a coherence granule that is owned by a remote processing element.

The FLUSH transaction is able to specify multiple double-word and sub-double-word data payloads; however, they must be aligned to byte, half-word, word, or double-word boundaries. Multiple double-word FLUSH transactions cannot exceed the number of double-words in the coherence granule. The write size and alignment for the FLUSH transaction are specified in Table 4-8. Unaligned and non-contiguous operations are not supported and must be broken into multiple FLUSH transactions by the sending processing element.

A flush operation internal to a processing element that would cause a FLUSH transaction for a remote coherence granule owned by that processing element (for example, attempting a cache write-through operation to a locally owned remote coherence granule) must generate a CASTOUT rather than a FLUSH transaction to properly implement the RapidIO protocol. Issuing a FLUSH under these circumstances generates a memory directory state paradox error in the home memory processing element.

Following are descriptions of the flush operations:

- If a flush operation is to a remote shared coherence granule, the FLUSH transaction is sent to the home memory, which sends a DKILL_SHARER transaction to all of the processing elements marked in the sharing list except for the requesting processing element. The processing elements that receive the DKILL_SHARER transaction invalidate the specified address if it is found shared in their caching hierarchy (see Figure 3-19).

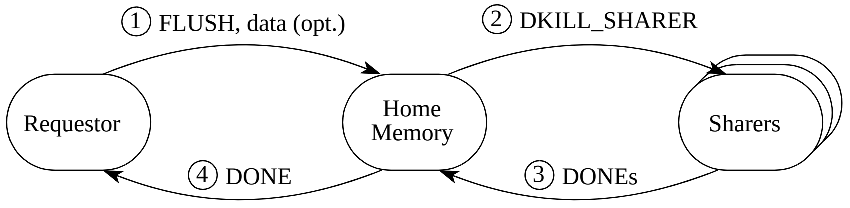


Figure 3-19. Flush Operation to Remote Shared Coherence Granule

- If the coherence granule is owned by a remote processing element, the home memory sends a READ_TO_OWN_OWNER transaction to it with the secondary (intervention) ID set to the home memory ID instead of the requestor ID. The owner then invalidates the coherence granule in its caching hierarchy and returns the coherence granule data (see Figure 3-20).

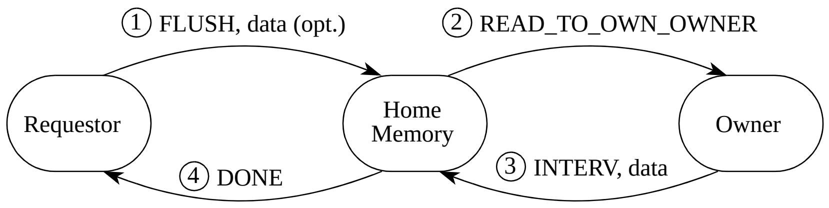


Figure 3-20. Flush Operation to Remote Modified Coherence Granule

- If the requestor and the home memory for the coherence granule are in the same processing element, DKILL_SHARER transactions are sent to all participants marked in the sharing list (see Figure 3-21).

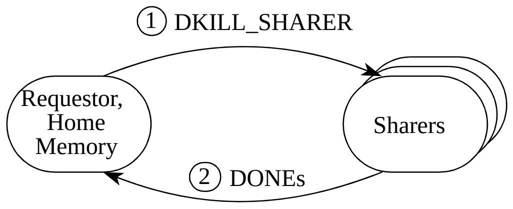


Figure 3-21. Flush Operation to Local Shared Coherence Granule

- If the requestor and the home memory for the coherence granule are in the same processing element but the coherence granule is owned by a remote processing element, a READ_TO_OWN_OWNER transaction is sent to the owner (see Figure 3-22).

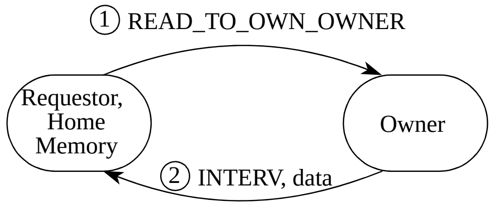


Figure 3-22. Flush Operation to Local Modified Coherence Granule

3.3.10 I/O Read Operations

The IO_READ_HOME, IO_READ_OWNER, and RESPONSE transactions are used during I/O read operations by a processing element that needs a current copy of cache-coherent data from the memory system, but does not need to be added to the sharing list in the memory directory state. The I/O read operation is most useful for DMA I/O devices. An I/O read operation always returns the requested size data payload. The requested data payload size can not exceed the size of the coherence granule. These transactions are used as follows:

- The IO_READ_HOME transaction is used by a requestor that is not in the same processing element as the home memory for the coherence granule.
- The IO_READ_OWNER transaction is used by a home memory processing element that needs to read a copy of a coherence granule owned by a remote processing element.

Following are descriptions of the I/O operations:

- If the requested data exists in the memory directory as shared, the data can be returned immediately from memory and the sharing mask is not modified (see Figure 3-24).

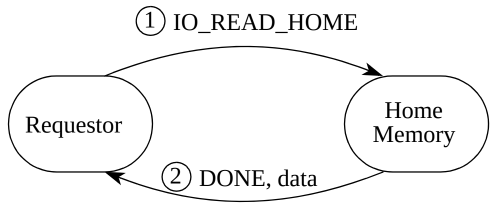


Figure 3-23. I/O Read Operation to Remote Shared Coherence Granule

- If the requested data exists in the memory directory as modified, the home memory sends an IO_READ_OWNER transaction to the processing element that owns the coherence granule. The owner passes a copy of the data to the requesting processing element (intervention) but retains ownership of and responsibility for the coherence granule (see Figure 3-24 and Figure 3-25).

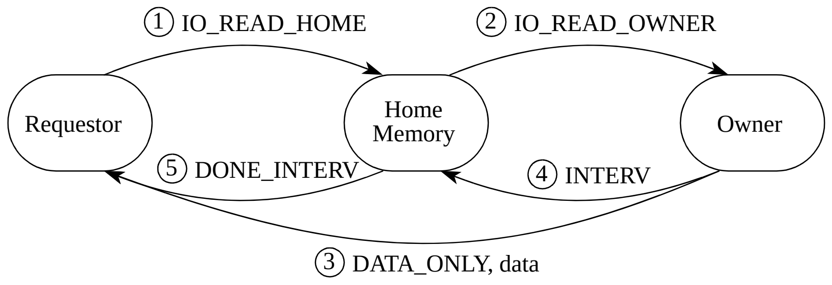


Figure 3-24. I/O Read Operation to Remote Modified Coherence Granule

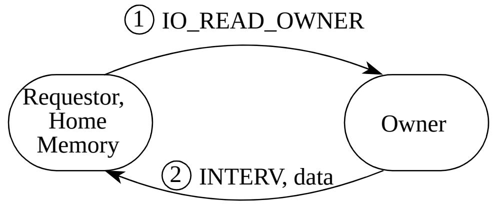
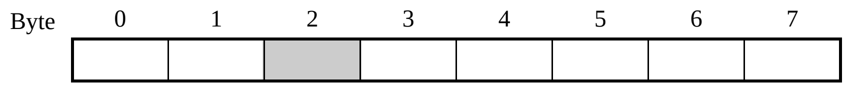


Figure 3-25. I/O Read Operation to Local Modified Coherence Granule

3.4 Endian, Byte Ordering, and Alignment

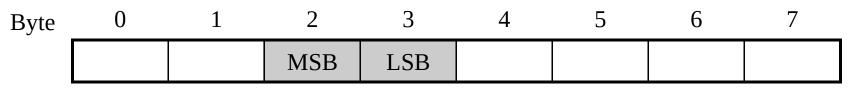
RapidIO has double-word (8-byte) aligned big-endian data payloads. This means that the RapidIO interface to devices that are little-endian shall perform the proper endian transformation to format a data payload.

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 3-26 through Figure 3-28.



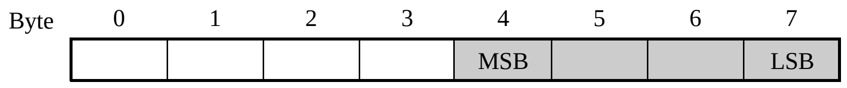
Byte address 0x0000_0002, the proper byte position is shaded.

Figure 3-26. Byte Alignment Example



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

Figure 3-27. Half-Word Alignment Example



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

Figure 3-28. Word Alignment Example

For write operations, a processing element shall properly align data transfers to a double-word boundary for transmission to the destination. This alignment may require breaking up a data stream into multiple transactions if the data is not naturally aligned. A number of data payload sizes and double-word alignments are defined to minimize this burden. Figure 3-29 shows a 48-byte data stream that a processing element wishes to write to another processing element through the interconnect fabric. The data displayed in the figure is big-endian and double-word aligned with the bytes to be written shaded in grey. Because the start of the stream and the end of the stream are not aligned to a double-word boundary, the sending processing element shall break the stream into three transactions as shown in the figure.

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

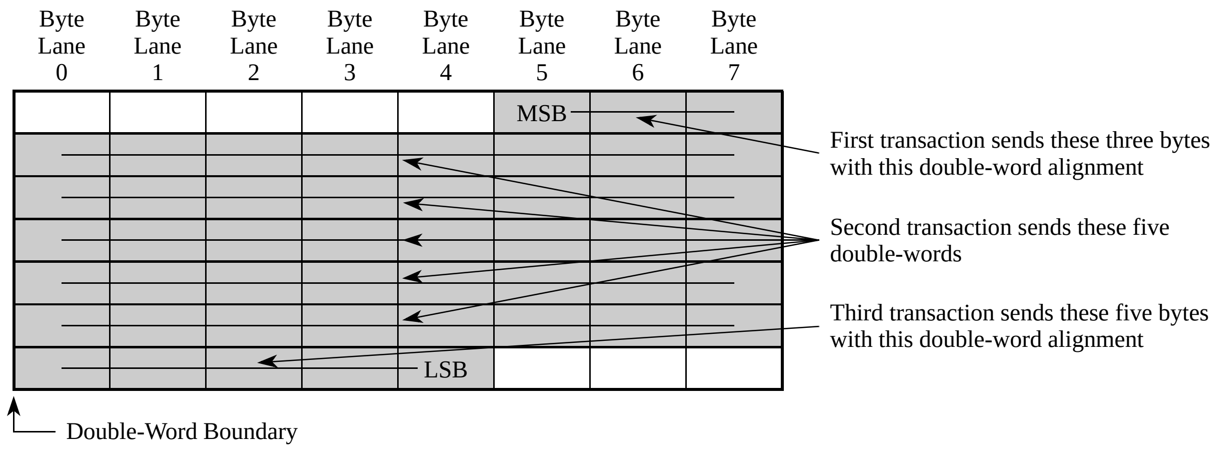

Byte Lane 0
Byte Lane 1
Byte Lane 2
Byte Lane 3
Byte Lane 4
Byte Lane 5
Byte Lane 6
Byte Lane 7
MSB
LSB
First transaction sends these three bytes with this double-word alignment
Second transaction sends these five double-words
Third transaction sends these five bytes with this double-word alignment
Double-Word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

Figure 3-29. Data Alignment Example

Blank page

Chapter 4 Packet Format Descriptions

4.1 Introduction

This chapter contains the packet format definitions for the *RapidIO Interconnect Globally Shared Memory Logical Specification*. There are four types of globally shared memory packet formats:

- Request
- Response
- Implementation-defined
- Reserved

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

4.2 Request Packet Formats

A request packet is issued by a processing element that needs a remote processing element to accomplish some activity on its behalf, such as a memory read operation. The request packet format types and their transactions for the *RapidIO Interconnect Globally Shared Memory Logical Specification* are shown in Table 4-1.

Table 4-1. Request Packet Type to Transaction Type Cross Reference

Request Packet Format Type	Transaction Type	Definition	Document Section No.
Type 0	Implementation-defined	Defined by the device implementation	Section 4.2.4
Type 1	READ_OWNER	Read shared copy of remotely owned coherence granule	Section 4.2.5
	READ_TO_OWN_OWNER	Read for store of remotely owned coherence granule	
	IO_READ_OWNER	Read for I/O of remotely owned coherence granule	

Table 4-1. Request Packet Type to Transaction Type Cross Reference (Continued)

Request Packet Format Type	Transaction Type	Definition	Document Section No.
Type 2	READ_TO_OWN_HOME	Read for store of home memory for coherence granule	Section 4.2.6
	READ_HOME	Read shared copy of home memory for coherence granule	
	IO_READ_HOME	Read for I/O of home memory for coherence granule	
	DKILL_HOME	Invalidate to home memory of coherence granule	
	IKILL_HOME	Invalidate to home memory of coherence granule	
	TLBIE	Invalidate TLB entry	
	TLBSYNC	Synchronize TLB invalidates	
	IREAD_HOME	Read shared copy of home memory for instruction cache	
	FLUSH	Force return of ownership of coherence granule to home memory, no update to coherence granule	
	IKILL_SHARER	Invalidate cached copy of coherence granule	
	DKILL_SHARER	Invalidate cached copy of coherence granule	
Type 3–4	—	Reserved	Section 4.2.7
Type 5	CASTOUT	Return ownership of coherence granule to home memory	Section 4.2.8
	FLUSH (with data)	Force return of ownership of coherence granule to home memory, update returned coherence granule	
Type 6–11	—	Reserved	Section 4.2.9

4.2.1 Addressing and Alignment

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

The coherence-granule-sized cache-coherent write requests and read responses are aligned to a double-word boundary within the coherence granule, with the specified data payload size matching that of the coherence granule. Sub-double-word data payloads must be padded and properly aligned within the 8-byte boundary. Non-contiguous or unaligned transactions that would ordinarily require a byte mask are not supported. A sending device that requires this behavior must break the operation into multiple request transactions. An example of this is shown in Section 3.4, “Endian, Byte Ordering, and Alignment.”

4.2.2 Data Payloads

Cache coherent systems are very sensitive to memory read latency. One way of reducing the latency is by returning the requested, or critical, double-word first upon a read request. Subsequent double-words are then returned in a sequential fashion. Table 4-2 and Table 4-3 show the return ordering for 32- and 64-byte coherence granules. Sub-double-word data payloads due to I/O read operations start with the requested size as shown.

Table 4-2. Coherent 32-Byte Read Data Return Ordering

Requested Double-word	Double-word Return Ordering
0	0, 1, 2, 3
1	1, 2, 3, 0
2	2, 3, 0, 1
3	3, 0, 1, 2

Table 4-3. Coherent 64-Byte Read Data Return Ordering

Requested Double-word	Double-word Return Ordering
0	0, 1, 2, 3, 4, 5, 6, 7
1	1, 2, 3, 0, 4, 5, 6, 7
2	2, 3, 0, 1, 4, 5, 6, 7
3	3, 0, 1, 2, 4, 5, 6, 7
4	4, 5, 6, 7, 0, 1, 2, 3
5	5, 6, 7, 4, 0, 1, 2, 3
6	6, 7, 4, 5, 0, 1, 2, 3
7	7, 4, 5, 6, 0, 1, 2, 3

Data payloads for cache coherent write-type transactions are always linear starting with the specified address at the first double-word to be written, (including flush transactions that are not the size of the coherence granule). Data payloads that cross the coherence granule boundary can not be specified. This implies that all castout transactions start with the first double-word in the coherence granule. Table 4-4 and Table 4-5 show the cache-coherent write-data ordering for 32- and 64-byte coherence granules, respectively.

Table 4-4. Coherent 32-Byte Write Data Payload

Starting Double-word	Number of Double-words	Double-word Data Ordering Within Coherence Granule
0	1	0
0	2	0, 1
0	3	0, 1, 2
0	4	0, 1, 2, 3
1	1	1
1	2	1, 2
1	3	1, 2, 3
2	1	2
2	2	2, 3
3	1	3

Table 4-5. Coherent 64-Byte Write Data Payloads

Starting Double-word	Number of Double-words	Double-word Data Ordering Within Coherence Granule
0	1	0
0	2	0, 1
0	3	0, 1, 2
0	4	0, 1, 2, 3
0	5	0, 1, 2, 3, 4
0	6	0, 1, 2, 3, 4, 5
0	7	0, 1, 2, 3, 4, 5, 6
0	8	0, 1, 2, 3, 4, 5, 6, 7
1	1	1
1	2	1, 2
1	3	1, 2, 3
1	4	1, 2, 3, 4
1	5	1, 2, 3, 4, 5
1	6	1, 2, 3, 4, 5, 6
1	7	1, 2, 3, 4, 5, 6, 7
2	1	2
2	2	2, 3
2	3	2, 3, 4
2	4	2, 3, 4, 5
2	5	2, 3, 4, 5, 6
2	6	2, 3, 4, 5, 6, 7
3	1	3
3	2	3, 4

Table 4-5. Coherent 64-Byte Write Data Payloads (Continued)

Starting Double-word	Number of Double-words	Double-word Data Ordering Within Coherence Granule
3	3	3, 4, 5
3	4	3, 4, 5, 6
3	5	3, 4, 5, 6, 7
4	1	4
4	2	4, 5
4	3	4, 5, 6
4	4	4, 5, 6, 7
5	1	5
5	2	5, 6
5	3	5, 6, 7
6	1	6
6	2	6, 7
7	1	7

4.2.3 Field Definitions for All Request Packet Formats

Fields that are unique to type 1, type 2, and type 5 formats are defined in their sections. Bit fields that are defined as “reserved” shall be assigned to logic 0s when generated and ignored when received. Bit field encodings that are defined as “reserved” shall not be assigned when the packet is generated. A received reserved encoding is regarded as an error if a meaningful encoding is required for the transaction and function, otherwise it is ignored. Implementation-defined fields shall be ignored unless the encoding is understood by the receiving device. All packets described are bit streams from the first bit to the last bit, represented in the figures from left to right respectively.

The following field definitions in Table 4-6 apply to all of the request packet formats.

Table 4-6. General Field Definitions for All Request Packets

Field	Definition
ftype	Format type, represented as a 4-bit value; is always the first four bits in the logical packet stream.
wdptr	Word pointer, used in conjunction with the data size (rdsize and wrsize) fields—see Table 4-7, Table 4-8, and Section 3.4.
rdsize	Data size for read transactions, used in conjunction with the word pointer (wdptr) bit—see Table 4-7 and Section 3.4.
wrsize	Write data size for sub-double-word transactions, used in conjunction with the word pointer (wdptr) bit—see Table 4-8 and Section 3.4. For writes greater than one double-word, the size is the maximum payload.
rsrv	Reserved
srcTID	The packet’s transaction ID.

Table 4-6. General Field Definitions for All Request Packets (Continued)

Field	Definition
transaction	The specific transaction within the format class to be performed by the recipient; also called type or ttype.
extended address	Optional. Specifies the most significant 16 bits of a 50-bit physical address or 32 bits of a 66-bit physical address.
xamsbs	Extended address most significant bits. Further extends the address specified by the address and extended address fields by 2 bits. This field provides 34-, 50-, and 66-bit addresses to be specified in a packet with the xamsbs as the most significant bits in the address.
address	Least significant 29 bits (bits [0-28] of byte address [0-31]) of the double-word physical address

Table 4-7. Read Size (rdsize) Definitions

wdptr	rdsize	Number of Bytes	Byte Lanes	Comment
0b0	0b0000	1	0b10000000	I/O read only
0b0	0b0001	1	0b01000000	I/O read only
0b0	0b0010	1	0b00100000	I/O read only
0b0	0b0011	1	0b00010000	I/O read only
0b1	0b0000	1	0b00001000	I/O read only
0b1	0b0001	1	0b00000100	I/O read only
0b1	0b0010	1	0b00000010	I/O read only
0b1	0b0011	1	0b00000001	I/O read only
0b0	0b0100	2	0b11000000	I/O read only
0b0	0b0101	3	0b11100000	I/O read only
0b0	0b0110	2	0b00110000	I/O read only
0b0	0b0111	5	0b11111000	I/O read only
0b1	0b0100	2	0b00001100	I/O read only
0b1	0b0101	3	0b00000111	I/O read only
0b1	0b0110	2	0b00000011	I/O read only
0b1	0b0111	5	0b00011111	I/O read only
0b0	0b1000	4	0b11110000	I/O read only
0b1	0b1000	4	0b00001111	I/O read only
0b0	0b1001	6	0b11111100	I/O read only
0b1	0b1001	6	0b00111111	I/O read only
0b0	0b1010	7	0b11111110	I/O read only
0b1	0b1010	7	0b01111111	I/O read only
0b0	0b1011	8	0b11111111	I/O read only
0b1	0b1011	16		I/O read only
0b0	0b1100	32		
0b1	0b1100	64		
0b0-1	0b1101 0b1111			Reserved

Table 4-8. Write Size (wrsize) Definitions

wdptr	wrsize	Number of Bytes	Byte Lanes
0b0	0b0000	1	0b10000000
0b0	0b0001	1	0b01000000
0b0	0b0010	1	0b00100000
0b0	0b0011	1	0b00010000
0b1	0b0000	1	0b00001000
0b1	0b0001	1	0b00000100
0b1	0b0010	1	0b00000010
0b1	0b0011	1	0b00000001
0b0	0b0100	2	0b11000000
0b0	0b0101	3	0b11100000
0b0	0b0110	2	0b00110000
0b0	0b0111	5	0b11111000
0b1	0b0100	2	0b00001100
0b1	0b0101	3	0b00000111
0b1	0b0110	2	0b00000011
0b1	0b0111	5	0b00011111
0b0	0b1000	4	0b11110000
0b1	0b1000	4	0b00001111
0b0	0b1001	6	0b11111100
0b1	0b1001	6	0b00111111
0b0	0b1010	7	0b11111110
0b1	0b1010	7	0b01111111
0b0	0b1011	8	0b11111111
0b1	0b1011	16 maximum	
0b0	0b1100	32 maximum	
0b1	0b1100	64 maximum	
0b0-1	0b1101-1111	Reserved	

4.2.4 Type 0 Packet Format (Implementation-Defined)

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

4.2.5 Type 1 Packet Format (Intervention-Request Class)

Type 1 request packets never include data. They are the only request types that can cause an intervention, so the secondary domain, secondary ID, and secondary transaction ID fields are required. The total number of bits available for the secondary domain and secondary ID fields (shown in Figure 4-1) is determined by the size of the transport field defined in the appropriate transport layer specification, so the size (labeled *m* and *n*, respectively) of these fields are not specified. The division of the bits between the logical coherence domain and device ID fields is determined by the specific application. For example, an 8 bit transport field allows 16 coherence domains of 16 participants.

The type 1 packet format is used for the READ_OWNER, READ_TO_OWN_OWNER, and IO_READ_OWNER transactions that are specified in the transaction sub-field column defined in Table 4-9. Type 1 packets are issued only by a home memory controller to allow the third party intervention data transfer.

Definitions and encodings of fields specific to type 1 packets are displayed in Table 4-9. Fields that are not specific to type 1 packets are described in Table 4-6.

Table 4-9. Specific Field Definitions and Encodings for Type 1 Packets

Field	Encoding	Sub-Field	Definition
secID	—		Original requestor’s, or secondary, ID for intervention
secTID	—		Original requestor’s, or secondary, transaction ID for intervention
sec_domain	—		Original requestor’s, or secondary, domain for intervention
transaction	0b0000	READ_OWNER	
	0b0001	READ_TO_OWN_OWNER	
	0b0010	IO_READ_OWNER	
	0b0011–1111	Reserved	

Figure 4-1 displays a type 1 packet with all its fields. The field value 0b0001 in Figure 4-1 specifies that the packet format is of type 1.

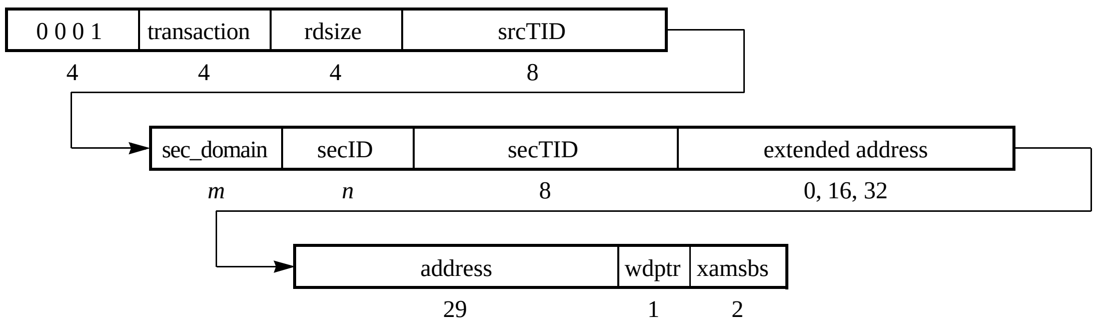


Figure 4-1. Type 1 Packet Bit Stream Format

4.2.6 Type 2 Packet Format (Request Class)

Type 2 request packets never include data. They cannot cause an intervention so the secondary domain and ID fields specified in the intervention-request format are not required. This format is used for the READ_HOME, IREAD_HOME, READ_TO_OWN_HOME, IO_READ_HOME, DKILL_HOME, DKILL_SHARER, IKILL_HOME, IKILL_SHARER, TLBIE, and TLBSYNC transactions as specified in the transaction field defined in Table 4-10. Type 2 packets for READ_HOME, IREAD_HOME, READ_TO_OWN_HOME, IO_READ_HOME, FLUSH without data, DKILL_HOME, and IKILL_HOME transactions are issued to home memory by a processing element. DKILL_SHARER and IKILL_SHARER transactions are issued by a home memory to the sharers of a coherence granule. DKILL_HOME, DKILL_SHARER, IKILL_HOME, IKILL_SHARER, FLUSH without data, and TLBIE are address-only transactions so the rdsize and wdptr fields are ignored and shall be set to logic 0. TLBSYNC is a transaction-type-only transaction so both the address, xamsbs, rdsize, and wdptr fields shall be set to logic 0.

The transaction field encodings for type 2 packets are displayed in Table 4-10. Fields that are not specific to type 2 packets are described in Table 4-6.

Table 4-10. Transaction Field Encodings for Type 2 Packets

Encoding	Transaction Field
0b0000	READ_HOME
0b0001	READ_TO_OWN_HOME
0b0010	IO_READ_HOME
0b0011	DKILL_HOME
0b0100	Reserved
0b0101	IKILL_HOME
0b0110	TLBIE
0b0111	TLBSYNC
0b1000	IREAD_HOME
0b1001	FLUSH without data
0b1010	IKILL_SHARER
0b1011	DKILL_SHARER
0b1100–1111	Reserved

Figure 4-2 displays a type 2 packet with all its fields. The field value 0b0010 in Figure 4-2 specifies that the packet format is of type 2.

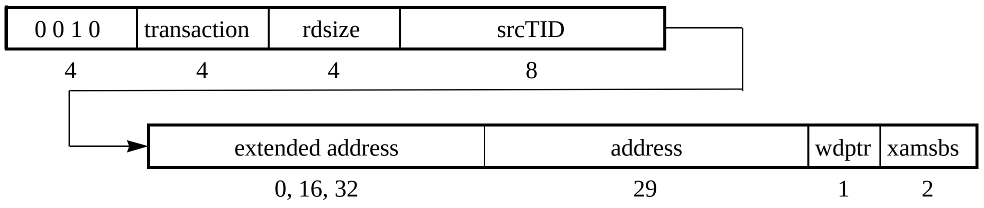


Figure 4-2. Type 2 Packet Bit Stream Format

4.2.7 Type 3–4 Packet Formats (Reserved)

The type 3–4 packet formats are reserved.

4.2.8 Type 5 Packet Format (Write Class)

Type 5 packets always contain data. A data payload that consists of a single double-word or less has sizing information as defined in Table 4-8. The wrsize field specifies the maximum size of the data payload for multiple double-word transactions. The FLUSH with data and CASTOUT transactions use type 5 packets as defined in Table 4-11. Note that type 5 transactions always contain data.

Fields that are not specific to type 5 packets are described in Table 4-6.

Table 4-11. Transaction Field Encodings for Type 5 Packets

Encoding	Transaction Field
0b0000	CASTOUT
0b0001	FLUSH with data
0b0010–1111	Reserved

Figure 4-3 displays a type 5 packet with all its fields. The field value 0b0101 in Figure 4-3 specifies that the packet format is of type 5.

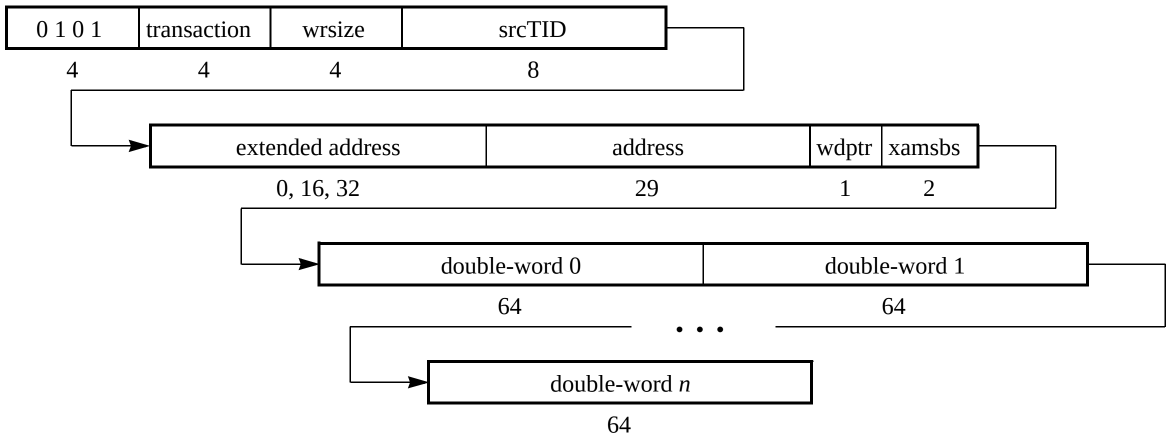


Figure 4-3. Type 5 Packet Bit Stream Format

4.2.9 Type 6–11 Packet Formats (Reserved)

The type 6–11 packet formats are reserved.

4.3 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 4-12.

Table 4-12. Request Packet Type to Transaction Type Cross Reference

Request Packet Format Type	Transaction Type	Definition	Document Section No.
Type 12	—	Reserved	Section 4.3.2
Type 13	RESPONSE	Issued by a processing element when it completes a request by a remote element.	Section 4.3.3
Type 14	—	Reserved	Section 4.3.4
Type 15	Implementation-defined	Defined by the device implementation	Section 4.3.5

4.3.1 Field Definitions for All Response Packet Formats

The field definitions in Table 4-13 apply to more than one of the response packet formats.

Table 4-13. Field Definitions and Encodings for All Response Packets

Field	Encoding	Sub-Field	Definition
transaction	0b0000		RESPONSE transaction with no data payload
	0b0001–0111		Reserved
	0b1000		RESPONSE transaction with data payload
	0b1001–1111		Reserved
targetTID	—		The corresponding request packet’s transaction ID

Table 4-13. Field Definitions and Encodings for All Response Packets (Continued)

status	Type of status and encoding		
	0b0000	DONE	Requested transaction has been successfully completed
	0b0001	DATA_ONLY	This is a data only response
	0b0010	NOT_OWNER	Not owner of requested coherence granule
	0b0011	RETRY	Requested transaction is not accepted; must retry the request
	0b0100	INTERVENTION	Update home memory with intervention data
	0b0101	DONE_INTERVENTION	Done for a transaction that resulted in an intervention
	0b0110	—	Reserved
	0b0111	ERROR	Unrecoverable error detected
	0b1000–1011	—	Reserved
	0b1100–1111	Implementation	Implementation defined—Can be used for additional information such as an error code

4.3.2 Type 12 Packet Format (Reserved)

The type 12 packet format is reserved.

4.3.3 Type 13 Packet Format (Response Class)

The type 13 packet format returns status, data (if required), and the requestor’s transaction ID. A RESPONSE packet with an “ERROR” status or a response that is not expected to have a data payload never has a data payload. The type 13 format is used for response packets to all request transactions.

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

Figure 4-4 illustrates the format and fields of type 13 packets. The field value 0b1101 in Figure 4-4 specifies that the packet format is of type 13.

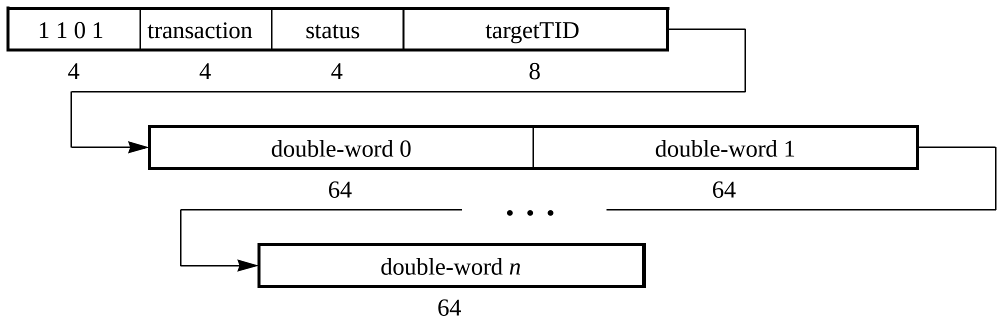


Figure 4-4. Type 13 Packet Bit Stream Format

4.3.4 Type 14 Packet Format (Reserved)

The type 14 packet format is reserved.

4.3.5 Type 15 Packet Format (Implementation-Defined)

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

Blank page

Chapter 5 Globally Shared Memory Registers

5.1 Introduction

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

5.2 Register Summary

Table 5-1 shows the register map for this RapidIO specification. These capability registers (CARs) and command and status registers (CSRs) can be accessed using the *RapidIO Part 1: Input/Output Logical Specification* maintenance operations. Any register offsets not defined are considered reserved for this specification unless otherwise stated. Other registers required for a processing element are defined in other applicable RapidIO specifications and by the requirements of the specific device and are beyond the scope of this specification. Read and write accesses to reserved register offsets shall terminate normally and not cause an error condition in the target device. Writes to CAR (read-only) space shall terminate normally and not cause an error condition in the target device.

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

Table 5-1. GSM Register Map

Configuration Space Byte Offset	Register Name
0x0-14	Reserved
0x18	Source Operations CAR
0x1C	Destination Operations CAR
0x20-FC	Reserved

Table 5-1. GSM Register Map (Continued)

Configuration Space Byte Offset	Register Name
0x100-FFFC	Extended Features Space
0x10000-FFFFFC	Implementation-defined Space

5.3 Reserved Register, Bit and Bit Field Value Behavior

Table 5-2 describes the required behavior for accesses to reserved register bits and reserved registers for the RapidIO register space,

Table 5-2. Configuration Space Reserved Access Behavior

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

Table 5-2. 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.

When a writable bit field is set to a reserved value, device behavior is implementation specific.

5.4 Capability Registers (CARs)

Every processing element shall contain a set of registers that allows an external processing element to determine its capabilities using the I/O logical maintenance read operation. All registers are 32 bits wide and are organized and accessed in 32-bit (4 byte) quantities, although some processing elements may optionally allow larger accesses. CARs are read-only. Refer to Table 5-2 for the required behavior for accesses to reserved registers and register bits.

CARs are big-endian with bit 0 the most significant bit.

5.4.1 Source Operations CAR (Configuration Space Offset 0x18)

This register defines the set of RapidIO GSM logical operations that can be issued by this processing element; see Table 5-3. It is assumed that a processing element can generate I/O logical maintenance read and write requests if it is required to access CARs and CSRs in other processing elements. RapidIO switches shall be able to route any packet.

Table 5-3. Bit Settings for Source Operations CAR

Bit	Field Name	Description
0	Read	PE can support a read operation
1	Instruction read	PE can support an instruction read operation
2	Read-for-ownership	PE can support a read-for-ownership operation
3	Data cache invalidate	PE can support a data cache invalidate operation
4	Castout	PE can support a castout operation
5	Data cache flush	PE can support a data cache flush operation
6	I/O read	PE can support an I/O read operation
7	Instruction cache invalidate	PE can support an instruction cache invalidate operation
8	TLB invalidate-entry	PE can support a TLB invalidate-entry operation
9	TLB invalidate-entry sync	PE can support a TLB invalidate-entry sync operation
10–13	—	Reserved
14–15	Implementation Defined	Defined by the device implementation
16–29	—	Reserved
30–31	Implementation Defined	Defined by the device implementation

5.4.2 Destination Operations CAR (Configuration Space Offset 0x1C)

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

Table 5-4. Bit Settings for Destination Operations CAR

Bit	Field Name	Description
0	Read	PE can support a read operation
1	Instruction read	PE can support an instruction read operation
2	Read-for-ownership	PE can support a read-for-ownership operation
3	Data cache invalidate	PE can support a data cache invalidate operation
4	Castout	PE can support a castout operation
5	Data cache flush	PE can support a flush operation
6	I/O read	PE can support an I/O read operation
7	Instruction cache invalidate	PE can support an instruction cache invalidate operation
8	TLB invalidate-entry	PE can support a TLB invalidate-entry operation
9	TLB invalidate-entry sync	PE can support a TLB invalidate-entry sync operation
10–13	—	Reserved
14-15	Implementation Defined	Defined by the device implementation
16-29	—	Reserved
30-31	Implementation Defined	Defined by the device implementation

5.5 Command and Status Registers (CSRs)

The RapidIO Globally Shared Memory Logical Specification does not define any command and status registers (CSRs).

Chapter 6 Communication Protocols

6.1 Introduction

This chapter contains the RapidIO globally shared memory (GSM) communications protocol definitions. Three state machines are required for a processing element on the RapidIO interface: one for local system accesses to local and remote space, one for remote accesses to local space, and one for handling responses made by the remote system to requests from the local system. The protocols are documented as pseudo-code partitioned by operation type. The RapidIO protocols as defined here assume a directory state definition that uses a modified bit with the local processor always sharing as described in Chapter 2, “System Models.” The protocols can be easily modified to use an alternate directory scheme that allows breaking the SHARED state into a REMOTE_SHARED and a REMOTE_AND_LOCAL_SHARED state pair.

Similarly, it may be desirable for an implementation to have an UNOWNED state instead of defaulting to LOCAL_SHARED or LOCAL_MODIFIED. These optimizations only affect the RapidIO transaction issuing behavior within a processing element, not the globally shared memory protocol itself. This flexibility allows a variety of local processor cache state coherence definitions such as MSI or MESI.

Some designs may not have a source of local system requests, for example, the memory only processing element described in Section 2.2.3, “Memory-Only Processing Element Model”. The protocols for these devices are much less complicated, only requiring the external request state machine and a portion of the response state machine. Similarly, a design may not have a local memory controller, which is also a much less complicated device, requiring only a portion of the internal request and response state machines. The protocols assume a processor element and memory processing element as described in Figure 2-2.

6.2 Definitions

The general definitions of Section 6.2.1 apply throughout the protocol, and the requests and responses of state machines are defined in Section 6.2.2, “Request and Response Definitions.”

6.2.1 General Definitions

address_collision	An address match between the new request and an address currently being serviced by the state machines or some other address-based internal hazard. This frequently causes a retry of the new request.
assign_entry()	Assign resources (such as a queue entry) to service a request, mark the address as able to participate in address collision detection (if appropriate), and assign a transaction ID
data	Any data associated with the transaction; this field is frequently null
directory_state	The memory directory state for the address being serviced
error()	Signal an error (usually through an interrupt structure) to software, usually to indicate a coherence violation problem
free_entry()	Release all resources assigned to this transaction, remove it from address collision detection, and deallocate the transaction ID
local	Memory local to the processing element
local_request(m,n,...)	A local request to a local processor caused by an incoming external request that requires a snoop of the processor’s caches
local_response(m,n,.)	A local response to a local request; usually indicates the cache state for the requesting processor to mark the requested data
LOCAL_RTYPE	This is the response from the local agent to the local processor in response to a local request.
LOCAL_TTYPE	This is the transaction type for a request passed from the RapidIO interconnect to a local device.
(mask <= (mask ~= received_srcid))	“Assign the mask field to the old mask field with the received ID bit cleared.” This result is generated when a response to a multicast is received and it is not the last one expected.
((mask ~= (my_id OR received_id)) == 0)	“The mask field not including my ID or the received ID equals 0.” This result indicates that we have received all of the expected responses to a multicast request.
(mask ~= my_id)	“The sharing mask not including my ID.” This result is used for multicast operations where the requestor is in the sharing list but does not need to be included in the multicast transaction because it is the source of the transaction.
(mask <= (participant_list ~= my_id))	“The sharing mask includes all participants except my ID.” This result is used for the IKILL operation, which does not

	use the memory directory information.
(mask <= (participant_list ~= (received_srcid AND my_id)))	“The sharing mask includes all participants except the requestor’s and my IDs.” This result is used for the IKILL operation, which does not use the memory directory information.
(mask == received_srcid)	“The sharing mask only includes the requestor’s ID.” This result is used for the DKILL operation to detect a write-hit-on-shared case where the requestor has the only remote copy of the coherence granule.
original_srcid	The ID of the initial requestor for a transaction, saved in the state associated with the transaction ID
received_data	The response contained data
received_data_only_message	Flag set by set_received_data_only_message()
received_done_message	Flag set by set_received_done_message()
remote_request(m,n,...)	Make a request to the interconnect fabric
remote_response(m,n,...)	Send a response to the interconnect fabric
RESPONSE_TTYPE	This is the RapidIO transaction type for a response to a request
return_data()	Return data to the local requesting processor, either from memory or from a interconnect fabric buffer; the source can be determined from the context
secondary_id	The third party identifier for intervention responses; the processing element ID concatenated with the processing element domain.
set_received_data_only_message()	Remember that a DATA_ONLY response was received for this transaction ID
set_received_done_message()	Remember that a DONE response was received for this transaction ID
source_id	The source device identifier; the processing element ID concatenated with the processing element domain
target_id	The destination device identifier; the processing element ID

	concatenated with the processing element domain
TRANSACTION	The RapidIO transaction type code for the request
update_memory()	Write memory with data received from a response
update_state(m,n,...)	Modify the memory directory state to reflect the new system status

6.2.2 Request and Response Definitions

Following are the formats used in the pseudocode to describe request and response transactions sent between processing elements and the formats of local requests and responses between the cache coherence controller and the local cache hierarchy and memory controllers.

6.2.2.1 System Request

The system request format is:

```
remote_request(TRANSACTION, target_id, source_id, secondary_id, data)
```

which describes the necessary RapidIO request to implement the protocol.

6.2.2.2 Local Request

The local request format is:

```
local_request(LOCAL_TTYPE)
```

that is the necessary local processor request to implement the protocol; the pseudocode assumes a generic local bus. A local request also examines the remote cache as part of the processing element’s caching hierarchy. The local transactions are defined as:

DKILL	Causes the processor to transition the coherence granule to invalid regardless of the current state; data is not pushed if current state is modified
IKILL	Causes the processor to invalidate the coherence granule in the instruction cache
READ	Causes the processor to transition the coherence granule to shared and push data if necessary
READ_LATEST	Causes the processor to push data if modified but not transition the cache state
READ_TO_OWN	Causes the processor to transition the coherence granule to invalid and push data
TLBIE	Causes the processor to invalidate the specified translation look-aside buffer entry
TLBSYNC	Causes the processor to indicate when all outstanding TLBIEs have completed

6.2.2.3 System Response

The system response format is:

remote_response(RESPONSE_TTYPE, target_id, source_id, data (opt.))

which is the proper response to implement the protocol.

6.2.2.4 Local Response

The local response format is:

local_response(LOCAL_RTYPE)

In general, a transaction ID (TID) is associated with each device ID in order to uniquely identify a request. This TID is frequently a queue index in the source processing element. These TIDs are not explicitly called out in the pseudocode below. The local responses are defined as:

EXCLUSIVE	The processor has exclusive access to the coherence granule
OK	The transaction requested by the processor has or will complete properly
RETRY	Causes the processor to re-issue the transaction; this response may cause a local bus spin loop until the protocol allows a different response
SHARED	The processor has a shared copy of the coherence granule

6.3 Operation to Protocol Cross Reference

Table 6-1 contains a cross reference of the operations defined in the *RapidIO Interconnect Globally Shared Memory Logical Specification* and their system usage.

Table 6-1. Operation to Protocol Cross Reference

Operations	Protocol
Read	Section 6.4
Instruction read	Section 6.4
Read for ownership	Section 6.6
Data cache invalidate	Section 6.7
Instruction cache invalidate	Section 6.7
Castout	Section 6.8
TLB invalidate entry	Section 6.9
TLB invalidate entry synchronize	Section 6.9
Data cache flush	Section 6.10
I/O read	Section 6.11

6.4 Read Operations

This operation is a coherent data cache read; refer to the description in Section 3.3.1.

6.4.1 Internal Request State Machine

This state machine handles requests to both local and remote memory from the local processor.

```
if (address_collision)                                  // this is due to an external request
                                                        // in progress or a cache
            local_response(RETRY);                      // index hazard from a previous request
elseif (local)                                          // our local memory
            switch (directory_state)
            case LOCAL_MODIFIED:                        // local modified is OK if we default
                                                        // local memory to owned
                        local_response(EXCLUSIVE);
                        return_data();
            case LOCAL_SHARED,                          // local, owned by memory
            case SHARED:                                // shared local and remote
                        local_response(SHARED);
                        return_data();                  // keep directory state
                                                        // the way it was
            case REMOTE_MODIFIED:
                        local_response(SHARED);
                        assign_entry();                 // this means to assign
                                                        // a transaction ID,
                                                        // usually a queue entry
                        remote_request(READ_OWNER, mask_id, my_id, my_id);
            default:
                        error();
else                                                    // remote - we’ve got to go
                                                        // to another processing element
            assign_entry();
            local_response(RETRY);                      // can’t guarantee data before a
                                                        // snoop yet
            remote_request(READ_HOME, mem_id, my_id);
endif;
```

6.4.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of the local system or a third party.

```
if (my_id == mem_id == original_srcid)                  // original requestor is home memory
            switch(remote_response)                     // matches my_id only for
                                                        // REMOTE_MODIFIED case
            case INTERVENTION:
                        update_memory();
                        update_state(SHARED, original_srcid);
                        return_data();
                        free_entry();
            case NOT_OWNER,                             // due to address collision or
            case RETRY:                                 // passing requests
                        switch (directory_state)
                        case LOCAL_MODIFIED:
                                    local_response(EXCLUSIVE);
                                                        // when processor re-requests
                                    return_data();
                                    free_entry();
                        case LOCAL_SHARED:
```

```
                    local_response(SHARED);
                                        // when processor re-requests
                    return_data();
                    free_entry();
                case REMOTE_MODIFIED:   // mask_id must match received_srcid

                                        //or error; spin or wait for castout
                    remote_request(READ_OWNER, received_srcid,
                            my_id, my_id);
                default:
                    error();
        default
                error();
elseif(my_id == mem_id ~== original_id          // i'm home memory working for
                                                // a third party
        switch(remote_response)
        case INTERVENTION:
                update_memory();
                update_state(SHARED, original_srcid);
                remote_response(DONE_INTERVENTION, original_srcid, my_id);
                free_entry();
        case NOT_OWNER,                         // data comes from memory,
                                                // mimic intervention
        case RETRY:
                switch(directory_state)
                case LOCAL_SHARED:
                    update_state(SHARED, original_srcid);
                    remote_response(DATA_ONLY, original_srcid,
                            my_id, data);
                    remote_response(DONE_INTERVENTION, original_srcid,
                            my_id);
                    free_entry();
                case LOCAL_MODIFIED:
                    update_state(SHARED, original_srcid);
                    remote_response(DATA_ONLY, original_srcid,
                            my_id, data);
                    remote_response(DONE_INTERVENTION, original_srcid,
                            my_id);
                    free_entry();
                case REMOTE_MODIFIED:   // spin or wait for castout
                    remote_request(READ_OWNER, received_srcid,
                            my_id, my_id);
                default:
                    error();
        default:
                error();
else                                            // my_id ~= mem_id - I'm
                                                // requesting a remote
                                                // memory location
        switch(remote_response)
        case DONE:
                local_response(SHARED);         // when processor re-requests
                return_data();
                free_entry();
        case DONE_INTERVENTION:                 // must be from third party
                set_received_done_message();
                if (received_data_only_message)
                    free_entry();
                else
                                                // wait for a DATA_ONLY
                endif;
        case DATA_ONLY:                         // this is due to an intervention, a
                                                // DONE_INTERVENTION should come
                                                // separately
                local_response(SHARED);
                set_received_data_only_message();
                if (received_done_message)
```

```
                        return_data();
                        free_entry();
                else
                        return_data();          // OK for weak ordering
                endif;
        case RETRY:
                remote_request(READ_HOME, received_srcid, my_id);
        default
                error();
endif;
```

6.4.3 External Request State Machine

This state machine handles requests from the system to the local memory or the local system. This may require making further external requests.

```
if (address_collision)                                  // use collision tables in
                                                        // Chapter 7, "Address Collision Resolution
Tables"
elseif (READ_HOME)                                      // remote request to our local memory
        assign_entry();
        switch (directory_state)
        case LOCAL_MODIFIED:
                local_request(READ);
                update_state(SHARED, received_srcid);
                                                        // after possible push completes
                remote_response(DONE, received_srcid, my_id, data);
                free_entry();
        case LOCAL_SHARED,
        case SHARED:
                update_state(SHARED, received_srcid);
                remote_response(DONE, received_srcid, my_id, data);
                free_entry();
        case REMOTE_MODIFIED:
                if (mask_id ~= received_srcid)
                                                        // intervention case
                        remote_request(READ_OWNER, mask_id,
                                my_id, received_srcid);
                else
                        error();                        // he already owned it;
                                                        // cache paradox (or I-fetch after d-
                                                        // store if not fixed elsewhere)
                endif;
        default:
                error();
else                                                    // READ_OWNER request to our caches
        assign_entry();
        local_request(READ);                            // spin until a valid response
                                                        // from caches
        switch (local_response)
        case MODIFIED:                                  // processor indicated a push;
                                                        // wait for it
                cache_state(SHARED or INVALID);
                                                        // surrender ownership
                if (received_srcid == received_secid)
                                                        // original requestor is also home
                        remote_response(INTERVENTION, received_srcid,
                                my_id, data);
                else
                        remote_response(DATA_ONLY, received_secid,
                                my_id, data);
                        remote_response(INTERVENTION, received_srcid,
                                my_id, data);
                endif;
        case INVALID:                                   // must have cast it out
```

```
                    remote_response(NOT_OWNER, received_srcid, my_id);
          default;
                    error();
          free_entry();
endif;
```

6.5 Instruction Read Operations

This operation is a partially coherent instruction cache read; refer to the description in Section 3.3.2.

6.5.1 Internal Request State Machine

This state machine handles requests to both local and remote memory from the local processor.

```
if (address_collision)                                  // this is due to an external
                                                        // request in progress or a cache
            local_response(RETRY);                      // index hazard from a previous request
elseif (local)                                          // our local memory
            switch (directory_state)
            case LOCAL_MODIFIED:                        // local modified is OK if we default
                                                        // local memory to owned
                        local_response(EXCLUSIVE);
                        return_data();
            case LOCAL_SHARED,                          // local, owned by memory
            case SHARED:                                // shared local and remote
                        local_response(SHARED);
                        return_data();                  // keep directory state the way it was
            case REMOTE_MODIFIED:
                        local_response(SHARED);
                        assign_entry();                 // this means to assign a transaction
                                                        // ID, usually a queue entry
                        remote_request(READ_OWNER, mask_id, my_id, my_id);
            default:
                        error();
else                                                    // remote - we've got to go
                                                        // to another processing element
            assign_entry();
            local_response(RETRY);
                                                        // can't guarantee data before a
                                                        // snoop yet
            remote_request(IREAD_HOME, mem_id, my_id);
endif;
```

6.5.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of the local system or a third party.

```
if (my_id == mem_id == original_srcid)                  // original requestor is home memory
            error();
elseif(my_id == mem_id ~== original_id)                 // i'm home memory working for a
                                                        // third party
            switch(remote_response)
            case INTERVENTION:
                        update_memory();
                        update_state(SHARED, original_srcid);
                        remote_response(DONE, original_srcid, my_id);
                        free_entry();
            case NOT_OWNER,                             // data comes from memory,
                                                        // mimic intervention
            case RETRY:
                        switch(directory_state)
                        case LOCAL_SHARED:
                                    update_state(SHARED, original_srcid);
                                    remote_response(DONE, original_srcid, my_id);
                        free_entry();
                case LOCAL_MODIFIED:
                        update_state(SHARED, original_srcid);
                        remote_response(DONE, original_srcid, my_id);
                        free_entry();
                case REMOTE_MODIFIED:         // spin or wait for castout
                        remote_request(READ_OWNER, received_srcid,
                                my_id, my_id);
                default:
                        error();
        default:
                error();
else                                          // my_id ~= mem_id - I'm requesting
                                              // a remote memory location
        switch(remote_response)
        case DONE:
                local_response(SHARED);       // when processor re-requests
                return_data();
                free_entry();
        case DONE_INTERVENTION:               // must be from third party
                set_received_done_message();
                if (received_data_only_message)
                        free_entry();
                else
                                              // wait for a DATA_ONLY
                endif;
        case DATA_ONLY:                       // this is due to an intervention; a
                                              // DONE_INTERVENTION should come
                                              // separately
                local_response(SHARED);
                set_received_data_only_message();
                if (received_done_message)
                        return_data();
                        free_entry();
                else
                        return_data();        // OK for weak ordering
                endif;
        case RETRY:
                remote_request(IREAD_HOME, received_srcid, my_id);
        default
                error();
endif;
```

6.5.3 External Request State Machine

This state machine handles requests from the system to the local memory or the local system. This may require making further external requests.

```
if (address_collision)                        // use collision tables in
                                              // Chapter 7, "Address Collision Resolution
Tables"
elseif(IREAD_HOME)                            // remote request to our local memory
        assign_entry();
        switch (directory_state)
        case LOCAL_MODIFIED:
                local_request(READ);
                update_state(SHARED, received_srcid);
                                              // after possible push completes
                remote_response(DONE, received_srcid, my_id, data);
                free_entry();
        case LOCAL_SHARED,
        case SHARED:
                update_state(SHARED, received_srcid);
                remote_response(DONE, received_srcid, my_id, data);
                free_entry();
```

```
            case REMOTE_MODIFIED:
                if (mask_id ~= received_srcid)
                                          // intervention case
                    remote_request(READ_OWNER, mask_id,
                        my_id, received_srcid);
                else                      // he already owned it in his
                                          //data cache; cache paradox case
                    remote_request(READ_OWNER, mask_id, my_id, my_id);
                endif;
            default:
                error();
    endif;
```

6.6 Read for Ownership Operations

This is the coherent cache store miss operation.

6.6.1 Internal Request State Machine

This state machine handles requests to both local and remote memory from the local processor.

```
if (address_collision)                                  // this is due to an external request
                                                        // in progress or a cache index
        local_response(RETRY);                          // hazard from a previous request
elseif (local)                                          // our local memory
        switch (directory_state
        case LOCAL_MODIFIED,                            // local modified is OK if we
                                                        // default memory to owned locally
        case LOCAL_SHARED:
                local_response(EXCLUSIVE);              // give ownership to processor
                return_data();
                if (directory_state == LOCAL_SHARED)
                        update_state(LOCAL_MODIFIED)
                endif;
        case REMOTE_MODIFIED:                           // owned by another, get a copy
                                                        // and ownership
                assign_entry();
                local_response(RETRY);                  // retry
                remote_request(READ_TO_OWN_OWNER, mask_id, my_id, my_id);
        case SHARED:                                    // invalidate the sharing list
                assign_entry();
                local_response(RETRY);   // retry
                remote_request(DKILL_SHARER, (mask ~= my_id), my_id, my_id);
        default:
                error();
else                                                    // remote - we've got to go to another
                                                        // processing element
        assign_entry();
        local_response(RETRY);
        remote_request(READ_TO_OWN_HOME, mem_id, my_id);
endif;
```

6.6.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of the local system or a third party.

```
if (my_id == mem_id == original_srcid)                  // original requestor is home memory
        switch (received_response)
        case DONE:                                      // SHARED, so invalidate case
                if ((mask ~= (my_id OR received_id)) == 0)
                                                        // this is the last DONE
                        local_response(EXCLUSIVE);
                        return_data();
                        update_state(LOCAL_MODIFIED);
                        free_entry();
                else
                        mask <= (mask ~= received_srcid);
                                                        // flip the responder's shared
                                                        // bit and wait for next DONE
                endif;
        case NOT_OWNER:                                 // due to address collision with
                                                        // CASTOUT or FLUSH
                switch(directory_state)
                case LOCAL_MODIFIED,:
                        local_response(EXCLUSIVE);
                        return_data();
                        free_entry();
                case LOCAL_SHARED:
                        local_response(EXCLUSIVE);
                        return_data();
                        update_state(LOCAL_MODIFIED);
                        free_entry();
                case REMOTE_MODIFIED:
                                        // spin or wait for castout
                        remote_request(READ_TO_OWN_OWNER, mask_id,
                                my_id, my_id);
                default:
                        error();
        case INTERVENTION:                      // remotely owned
                local_response(EXCLUSIVE);
                return_data();
                update_state(LOCAL_MODIFIED);
                free_entry();
        case RETRY:
                switch (directory_state)
                case LOCAL_MODIFIED:
                        local_response(EXCLUSIVE);
                        return_data();
                        free_entry();
                case LOCAL_SHARED:
                        local_response(EXCLUSIVE);
                        return_data();
                        update_state(LOCAL_MODIFIED);
                        free_entry();
                case REMOTE_MODIFIED:           //mask_id must match received_srcid
                                                // or error condition
                        remote_request(READ_TO_OWN_OWNER, received_srcid,
                                my_id, my_id);
                case SHARED:
                        remote_request(DKILL_SHARER, received_srcid, my_id,
                                my_id);
                default:
                        error();
        default:
                error();
elseif (my_id == mem_id ~= original_srcid)
                                        // i’m home memory working
                                        // for a third party
        switch(received_response)
        case DONE:                              // invalidates for shared
                                                // directory states
                if ((mask ~= (my_id OR received_id)) == 0)
                                                // this is the last DONE
                        update_state(REMOTE_MODIFIED, original_srcid);
                        remote_response(DONE, original_srcid, my_id, data);
                        free_entry();
                else
                        mask <= (mask ~= received_srcid);
                                                // flip the responder’s shared bit
                endif;                          // and wait for next DONE
        case INTERVENTION:
                                                // remote_modified case
                update_memory();                // for possible coherence error
                                                // recovery
                update_state(REMOTE_MODIFIED, original_id);
                remote_response(DONE_INTERVENTION, original_id, my_id);
                free_entry();
        case NOT_OWNER:                         // data comes from memory, mimic
                                                // intervention
                switch(directory_state)
                case LOCAL_SHARED:
                case LOCAL_MODIFIED:
                        update_state(REMOTE_MODIFIED, original_srcid);
                        remote_response(DATA_ONLY, original_srcid, my_id,
                                data);
                        remote_response(DONE, original_srcid, my_id);
                        free_entry();
                case REMOTE_MODIFIED:
                        remote_request(READ_TO_OWN_OWNER, received_srcid,
                                my_id, original_srcid);
                default:
                        error();
        case RETRY:
                switch (directory_state)
                case LOCAL_MODIFIED,
                case LOCAL_SHARED:
                        update_state(REMOTE_MODIFIED, original_srcid);
                        remote_response(DATA_ONLY, original_srcid, my_id,
                                data);
                        remote_response(DONE, original_srcid, my_id);
                        free_entry();
                case REMOTE_MODIFIED:           // mask_id must match received_srcid
                                                // or error condition
                        remote_request(READ_TO_OWN_OWNER, received_srcid,
                                my_id, my_id);
                case SHARED:
                        remote_request(DKILL_SHARER, received_srcid, my_id,
                                my_id);
                default:
                        error();
        default:
                error();
else                                            // my_id ~= mem_id - I'm requesting
                                                // a remote memory location
        switch (received_response)
        case DONE:
                local_response(EXCLUSIVE);
                return_data();
                free_entry();
        case DONE_INTERVENTION:
                set_received_done_message();
                if (received_data_message)
                        free_entry();
                else
                                                // wait for DATA_ONLY
                endif;
        case DATA_ONLY:
                set_received_data_message();
                local_response(EXCLUSIVE);
                if (received_done_message)
                        return_data();
                        free_entry();
                else
                        return_data();          // OK for weak ordering
                endif;                          // and wait for a DONE
        case RETRY:                             // lost at remote memory so retry
                remote_request(READ_TO_OWN_HOME, mem_id, my_id);
        default:
                error();
endif;
```

6.6.3 External Request State Machine

This state machine handles requests from the interconnect to the local memory or the local system. This may require making further external requests.

```
if (address_collision)                                      // use collision tables
                                                            // in Chapter 7, “Address Collision Resolution
Tables”
elseif (READ_TO_OWN_HOME)                                   // remote request to our local memory
            assign_entry();
            switch (directory_state)
            case LOCAL_MODIFIED,
            case LOCAL_SHARED:
                      local_request(READ_TO_OWN);
                      remote_response(DONE, received_srcid, my_id, data);
                                                            // after possible push
                      update_state(REMOTE_MODIFIED, received_srcid);
                      free_entry();
            case REMOTE_MODIFIED:
                      if (mask_id ~= received_srcid)
                                                            //intervention case
                                remote_request(READ_TO_OWN_OWNER, mask_id, my_id,
                                          received_srcid);
                      else
                                error();                    // he already owned it!
                      endif;
            case SHARED:
                      local_request(READ_TO_OWN);
                      if (mask == received_srcid)
                                                            //requestor is only remote sharer
                                update_state(REMOTE_MODIFIED, received_srcid);
                                remote_response(DONE, received_srcid, my_id, data);
                                                            // from memory
                                free_entry();
                      else                                  //there are other remote sharers
                                remote_request(DKILL_SHARER, (mask ~= received_srcid),
                                          my_id, my_id);
                      endif;
            default:
                      error();
elseif(READ_TO_OWN_OWNER)                                   // request to our caches
            assign_entry();
            local_request(READ_TO_OWN);                     // spin until a valid response from
                                                            // the caches
            switch (local_response)
            case MODIFIED:                                  // processor indicated a push
                      cache_state(INVALID);
                                                            // surrender ownership
                      if (received_srcid == received_secid)
                                                            //the original request is from the home
                                remote_response(INTERVENTION, received_srcid, my_id,
                                          data);
                      else                                  // the original request is from a
                                                            // third party
                                remote_response(DATA_ONLY, received_secid, my_id,
                                          data);
                                remote_response(INTERVENTION, received_srcid, my_id,
                                          data);
                      endif;
                      free_entry();
            case INVALID:                                   // castout address collision
                      remote_response(NOT_OWNER, received_srcid, my_id);
            default:
                      error();
endif;
```

6.7 Data Cache and Instruction Cache Invalidate Operations

This operation is used with coherent cache store-hit-on-shared, cache operations; refer to the description in Section 3.3.4.

6.7.1 Internal Request State Machine

This state machine handles requests to both local and remote memory from the local processor.

```
if (address_collision)                                 // this is due to an external request in
                                                       // progress or a cache index
            local_response(RETRY);                     // hazard from a previous request
elseif (local)                                         // our local memory and we won
            if (DKILL)                                 // DKILL checks the directory
                    switch (directory_state)
                    case LOCAL_MODIFIED,               // local modified is OK if we default
                                                       // memory to owned locally
                    case LOCAL_SHARED:
                            local_response(EXCLUSIVE);
                            if (LOCAL_SHARED)
                                    update_state(LOCAL_MODIFIED, my_id);
                            endif;
                    case REMOTE_MODIFIED:              // cache paradox; DKILL is
                                                       // write-hit-on-shared
                            error();
                    case SHARED:
                            local_response(RETRY);
                            assign_entry();            // Multicast if possible otherwise
                                                       // issue direct to each sharer
                            remote_request(DKILL_SHARER, (mask ~= my_id), my_id);
                    default:
                            error();
            else                                       // IKILL always goes to everyone
                    remote_request(IKILL_SHARER,
                            (mask <= (participant_list ~= my_id)), my_id);
            endif;
else                                                   // remote - we've got to go to another
                                                       // processing element
            assign_entry();
            local_response(RETRY);
            remote_request({DKILL_HOME, IKILL_HOME}, mem_id, my_id);
endif;
```

6.7.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of the local system or a third party.

```
if (my_id == mem_id == original_srcid)                 // original requestor is home memory
            switch (received_response)
            case DONE:                                 // shared cases
                    if ((mask ~= (my_id OR received_id)) == 0)
                                                       // this is the last DONE
                            if (DKILL)                 // don't update state for IKILLs
                                    update_state(LOCAL_MODIFIED);
                            endif;
                            free_entry();
                    else
```

```
                        mask <= (mask ~= received_srcid);
                                                // flip the responder's shared bit and
                endif;                          // wait for next DONE
        case RETRY:
                remote_request({DKILL_SHARER, IKILL_SHARER}, received_srcid,
                        my_id);                 // retry the transaction
        default:
                error();
elseif (my_id == mem_id ~= original_srcid)
                                                // i'm home memory working
                                                // for a third party
        switch(received_response)
        case DONE:                              // invalidates for shared
                                                // directory states
                if ((mask ~= (my_id OR received_id)) == 0)
                                                // this is the last DONE
                        if (DKILL)              // don't update state for IKILLs
                                update_state(REMOTE_MODIFIED, original_srcid);
                        endif;
                        remote_response(DONE, original_srcid, my_id);
                        free_entry();
                else
                        mask <= (mask ~= received_srcid);
                                                // flip the responder's shared bit
                endif;                          // and wait for next DONE
        case RETRY:
                remote_request({DKILL_SHARER, IKILL_SHARER}, received_srcid,
                        my_id);                 // retry
        default:
                error();
else                                            // my_id ~= mem_id - I'm requesting
                                                // a remote memory location
        switch (received_response)
        case DONE:
                local_response(EXCLUSIVE);
                free_entry();
        case RETRY:
                remote_request({DKILL_HOME, IKILL_HOME}, received_srcid,
                        my_id);                 // retry the transaction
        default:
                error();
endif;
```

6.7.3 External Request State Machine

This state machine handles requests from the system to the local memory or the local system. This may require making further external requests.

```
if (address_collision)                          // use collision tables in
                                                // Chapter 7, "Address Collision Resolution
Tables"
elseif (DKILL_HOME || IKILL_HOME)               // remote request to our local memory
        assign_entry();
        if (DKILL_HOME)
                switch (directory_state)
                case LOCAL_MODIFIED,            // cache paradoxes; DKILL is
                                                // write-hit-on-shared
                case LOCAL_SHARED,
                case REMOTE_MODIFIED:
                        error();
                case SHARED:                    // this is the right case, send
                                                // invalidates to the sharing list
                        local_request(DKILL);
                        if (mask == received_srcid
                                                // requestor is only remote sharer
```

```
                                                  if (DKILL)// don't update state for (IKILLs)
                                                            update_state(REMOTE_MODIFIED,
received_srcid);
                                                  endif;
                                                  remote_response(DONE, received_srcid, my_id);
                                                  free_entry();
                                        else                // there are other remote sharers
                                                  remote_request(DKILL_SHARER,
                                                            (mask ~= received_srcid), my_id, NULL);
                                        endif;
                              default:
                                        error();
                    else                                    // IKILL goes to everyone except the
                                                            // requestor
                              remote_request(IKILL_SHARER,
                                        (mask <= (participant_list ~=
                                        (received_srcid AND my_id), my_id);
else                                                        // DKILL_SHARER or IKILL_SHARER to
our caches
                    assign_entry();
                    local_request({READ_TO_OWN, IKILL});
                                                            // spin until a valid response from the
                                                            // caches
                    switch (local_response)
                    case SHARED,
                    case INVALID:                           // invalidating for shared cases
                              cache_state(INVALID);         // surrender copy
                              remote_response(DONE, received_srcid, my_id);
                              free_entry();
                    default:
                              error();
endif;
```

6.8 Castout Operations

This operation is used to return ownership of a coherence granule to home memory, leaving it invalid in the cache; refer to the description in Section 3.3.5.

6.8.1 Internal Request State Machine

A castout is always done to remote memory space. A castout may require local activity to flush all caches in the hierarchy.

```
if (local)                                               // our local memory
        switch (directory_state)
        case LOCAL_MODIFIED:                             // if the processor is doing a castout
                                                         // this is the only legal state
                local_response(OK);
                update_memory();
                update_state(LOCAL_SHARED);
        default:
                error();
else                                                     // remote - we've got to go to another
                                                         // processing element
        assign_entry();
        local_response(OK);
        remote_request(CASTOUT, mem_id, my_id, data);
endif;
```

6.8.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of the local system or a third party.

```
switch (received_response)
case DONE:
        free_entry();
default:
        error();
```

6.8.3 External Request State Machine

This state machine handles requests from the system to the local memory or the local system. This may require making further external requests.

```
assign_entry();
update_memory();
state_update(LOCAL_SHARED, my_id);                       // may be LOCAL_MODIFIED if the
                                                         // default is owned locally
remote_response(DONE, received_srcid, my_id);
free_entry();
```

6.9 TLB Invalidate Entry, TLB Invalidate Entry Synchronize Operations

These operations are used for software coherence management of the TLBs; refer to the descriptions in Section 3.3.6 and Section 3.3.7.

6.9.1 Internal Request State Machine

The TLBIE and TLBSYNC transactions are always sent to all domain participants except the sender and are always to the processor not home memory.

```
assign_entry();
remote_request({TLBIE, TLBSYNC}, participant_id, my_id);
endif;
```

6.9.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of the local system. The responses are always from a coherence participant, not a home memory.

```
switch (received_response)
case DONE:
        if ((mask ~= (my_id OR received_id)) == 0)
                                                // this is the last DONE
                free_entry();
        else
                mask <= (mask ~= received_srcid);
                                                // flip the responder’s participant
                                                // bit and wait for next DONE
        endif;
case RETRY:
        remote_request({TLBIE, TLBSYNC}, received_srcid, my_id, my_id);
default
        error();
```

6.9.3 External Request State Machine

This state machine handles requests from the system to the local memory or the local system. The requests are always to the local caching hierarchy.

```
assign_entry();
local_request({TLBIE, TLBSYNC});                // spin until a valid response
                                                // from the caches
remote_response(DONE, received_srcid, my_id);
free_entry();
```

6.10 Data Cache Flush Operations

This operation returns ownership of a coherence granule to home memory and performs a coherent write; refer to the description in Section 3.3.9.

6.10.1 Internal Request State Machine

This state machine handles requests to both local and remote memory from the local processor.

```
if (address_collision)                                      // this is due to an external
                                                            // request in progress or a cache index
        local_response(RETRY);                              // hazard from a previous request
elseif (local) // our local memory
        switch (directory_state)
        case LOCAL_MODIFIED,
        case LOCAL_SHARED:
                local_response(OK);
                update_memory();
        case REMOTE_MODIFIED:
                assign_entry();
                remote_request(READ_TO_OWN_OWNER, mask_id, my_id, my_id);
        case SHARED:
                assign_entry();
                remote_request(DKILL_SHARER, (mask ~= my_id), my_id);
        default:
                error();
else                                                        // remote - we've got to go to
                                                            // another processing element
        assign_entry();
        remote_request(FLUSH, mem_id, my_id, data);
                                                            // data is optional
endif;
```

6.10.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of the local system or a third party.

```
if (my_id == mem_id == original_srcid)                      // original requestor is home memory
        switch (received_response)
        case DONE:
                if ((mask ~= (my_id OR received_id)) == 0)
                                                            // this is the last DONE
                        if (received_data)
                                                            // with local request or response
                                update_memory();
                        endif;
                        update_state(LOCAL_SHARED);  // or LOCAL_MODIFIED
                        local_response(OK);
                        free_entry();
                else
                        mask <= (mask ~= received_srcid);
                                                            // flip responder's shared bit
                endif;                                      // and wait for next DONE
        case NOT_OWNER:
                switch(directory_state)
                case LOCAL_SHARED,
                case LOCAL_MODIFIED:
                        if (received_data)
                                                            // with local request from memory
                        update_memory();
                    endif;
                    update_state(LOCAL_SHARED);  // or LOCAL_MODIFIED
                    local_response(OK);
                    free_entry();
                case REMOTE_MODIFIED:
                    remote_request(READ_TO_OWN_OWNER,      mask_id,      my_id,
my_id);
                default:
                    error();
            case RETRY:
                switch (directory_state)
                case LOCAL_MODIFIED,
                case LOCAL_SHARED:
                    if (received_data)
                                            // with local request
                        update_memory();
                                            // if there was some write data
                    endif;
                    update_state(LOCAL_SHARED);  // or LOCAL_MODIFIED
                    local_response(OK);
                    free_entry();
                case REMOTE_MODIFIED:       // mask_id must match
                                            // received_srcid or error
                    remote_request(READ_TO_OWN_OWNER, received_srcid,
                        my_id, my_id);
                case SHARED:
                    remote_request(DKILL_SHARER, received_srcid, my_id,
                        my_id);
                default:
                    error();
            default:
                error();
elseif (my_id == mem_id ~= original_srcid)
                                            // i'm home memory working for a third
                                            // party
            switch(received_response)
            case DONE:                      // invalidates for shared directory
                                            // states
                if ((mask ~= (my_id OR received_id)) == 0)
                                            // this is the last DONE
                    remote_response(DONE, original_srcid, my_id, my_id);
                    if (received_data)
                                            // with original request or response
                        update_memory();
                    endif;
                    update_state(LOCAL_SHARED);// or LOCAL_MODIFIED
                    free_entry();
                else
                    mask <= (mask ~= received_srcid);
                                            // flip responder's shared bit
                endif;                      // and wait for next DONE
            case NOT_OWNER:
                switch(directory_state)
                case LOCAL_SHARED,
                case LOCAL_MODIFIED:
                    remote_response(DONE, original_srcid, my_id);
                    if (received_data)
                                            // with original request
                        update_memory();
                    endif;
                    free_entry();
                case REMOTE_MODIFIED:
                    remote_request(READ_TO_OWN_OWNER, received_srcid,
                        my_id, my_id);
                default:
                    error();
            case RETRY:
                switch(directory_state)
                case LOCAL_SHARED,
                case LOCAL_MODIFIED:
                    remote_response(DONE, original_srcid, my_id);
                    if (received_data)
                                            // with original request
                        update_memory();
                    endif;
                    free_entry();
                case REMOTE_MODIFIED:
                    remote_request(READ_TO_OWN_OWNER, received_srcid,
                        my_id, my_id);
                case SHARED:
                    remote_request(DKILL_SHARER, received_srcid, my_id);
                default:
                    error();
            default:
                error();
else                                        // my_id ~= mem_id - I'm requesting
                                            // a remote memory location
            switch (received_response)
            case DONE:
                local_response(OK);
                free_entry();
            case RETRY:
                remote_request(FLUSH, received_srcid, my_id, data);
                                            // data is optional
            default:
                error();
endif;
```

6.10.3 External Request State Machine

This state machine handles requests from the system to the local memory or the local system. This may require making further external requests.

```
if (address_collision)                      // use collision table in
                                            // Chapter 7, "Address Collision Resolution
Tables"
elseif (FLUSH)                              // remote request to our local memory
            assign_entry();
            switch (directory_state)
            case LOCAL_MODIFIED,
            case LOCAL_SHARED:
                local_request(READ_TO_OWN);
                remote_response(DONE, received_srcid, my_id);
                                            // after snoop completes
                if (received_data)          // from request or local response
                    update_memory();
                endif;
                update_state(LOCAL_SHARED, my_id);
                                            // or LOCAL_MODIFIED
                free_entry();
            case REMOTE_MODIFIED:
                if (mask_id ~= received_srcid)      // owned elsewhere
                    remote_request(READ_TO_OWN_OWNER, mask_id, my_id,
                        my_id);     // secondary TID is a don't care since data is
                                            // not forwarded to original requestor
                else                        // requestor owned it; shouldn't
                                            // generate a flush
                    error();
                endif;
            case SHARED:
                local_request(READ_TO_OWN);
```

```
                    if (mask == received_srcid)          // requestor is only remote sharer
                              remote_response(DONE, received_srcid, my_id);
                                                         // after snoop completes
                              if (received_data)         // from request or response
                                        update_memory();
                              endif;
                              update_state(LOCAL_SHARED, my_id); // or LOCAL_MODIFIED
                              free_entry();
                    else                                 //there are other remote sharers
                              remote_request(DKILL_SHARER, (mask ~= received_srcid), my_id,
                                                         my_id);
                    endif;
          default:
                    error();
endif;
```

6.11 I/O Read Operations

This operation is used for I/O reads of globally shared memory space; refer to the description in Section 3.3.10.

6.11.1 Internal Request State Machine

This state machine handles requests to both local and remote memory from the local processor.

```
if (address_collision)                                    // this is due to an external request
                                                          // in progress or a cache index hazard
            local_response(RETRY);                        // from a previous request
elseif (local)                                            // our local memory
            local_response(OK);
            switch (directory_state)
            case LOCAL_MODIFIED:                          // local modified is OK if we default
                                                          // local memory to owned
                      local_request(READ_LATEST);
                      return_data())                      // after possible push
            case LOCAL_SHARED,
            case SHARED:
                      return_data();                      // keep directory state the way it was
            case REMOTE_MODIFIED:
                      assign_entry();
                      remote_request(IO_READ_OWNER, mask_id, my_id, my_id);
            default:
                      error();
else                                                      // remote - we’ve got to go to
                                                          // another processing element
            assign_entry();
            local_response(OK);
            remote_request(IO_READ_HOME, mem_id, my_id);
endif;
```

6.11.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of the local system or a third party.

```
if (my_id == mem_id == original_srcid)
                                                          // original requestor is home memory
            switch(remote_response)                       // matches my_id only for
                                                          // REMOTE_MODIFIED case
            case INTERVENTION:
                      return_data();
                      free_entry();
            case NOT_OWNER,                               // due to address collision or
                                                          // passing requests
            case RETRY:
                      switch (directory_state)
                      case LOCAL_MODIFIED:
                      case LOCAL_SHARED
                                return_data();
                                free_entry();
                      case REMOTE_MODIFIED:               // mask_id must match received_srcid or
                                                          // error; spin or wait for castout
                                remote_request(IO_READ_OWNER, received_srcid, my_id,
                                          my_id);
                      default:
                                error();
            default
                  error();
elseif(my_id == mem_id ~== original_id)              // i'm home memory working for a third
                                                     // party
            switch(remote_response)
            case INTERVENTION:
                  update_memory();
                  remote_response(DONE_INTERVENTION, original_srcid, my_id);
                  free_entry();
            case NOT_OWNER,                          // data comes from memory, mimic
                                                     // intervention
            case RETRY:
                  switch(directory_state)
                  case LOCAL_MODIFIED,
                  case LOCAL_SHARED:
                        remote_response(DATA_ONLY, original_srcid, my_id,
                              data);
                        remote_response(DONE_INTERVENTION, original_srcid,
                              my_id);
                        free_entry();
                  case REMOTE_MODIFIED:          // spin or wait for castout
                        remote_request(IO_READ_OWNER, received_srcid, my_id,
                              my_id);
                  default:
                        error();
            default:
                  error();
else                                                 // my_id ~= mem_id - I'm requesting a
                                                     // remote memory location
            switch(remote_response)
            case DONE:
                  return_data();
                  free_entry();
            case DONE_INTERVENTION:                  // must be from third party
                  set_received_done_message();
                  if (received_data_only_message)
                        free_entry();
                  else
                        // wait for a DATA_ONLY
                  endif;
            case DATA_ONLY:                          // this is due to an intervention, a
                                                     // DONE_INTERVENTION should come
                                                     // separately
                  set_received_data_only_message();
                  if (received_done_message)
                        return_data();
                        free_entry();
                  else
                        return_data();           // OK for weak ordering
                  endif;
            case RETRY:
                  remote_request(IO_READ_HOME, received_srcid, my_id);
            default
                  error();
endif;
```

6.11.3 External Request State Machine

This state machine handles requests from the system to the local memory or the local system. This may require making further external requests.

```
if (address_collision)                               // use collision tables in
                                                     // Chapter 7, "Address Collision Resolution
Tables"
elseif (IO_READ_HOME)                                // remote request to our local memory
```

```
        assign_entry();
        switch (directory_state)
        case LOCAL_MODIFIED:
            local_request(READ_LATEST);
            remote_response(DONE, received_srcid, my_id, data);
                                        // after push completes
            free_entry();
        case LOCAL_SHARED:
            remote_response(DONE, received_srcid, my_id, data);
            free_entry();
        case REMOTE_MODIFIED:
            remote_request(IO_READ_OWNER, mask_id, my_id, received_srcid);
        case SHARED:
            remote_response(DONE, received_srcid, my_id, data);
            free_entry();
        default:
            error();
else                                    // IO_READ_OWNER request to our caches
        assign_entry();
        local_request(READ_LATEST);     // spin until a valid response from
                                        // the caches
        switch (local_response)
        case MODIFIED:                  // processor indicated a push;
                                        // wait for it
            if (received_srcid == received_secid)
                                        // original requestor is also home
                                        // memory
                remote_response(INTERVENTION, received_srcid, my_id,
                    data);
            else
                remote_response(DATA_ONLY, received_secid, my_id,
                    data);
                remote_response(INTERVENTION, received_srcid, my_id);
            endif;
        case INVALID:                   // must have cast it out during
                                        // an address collision
            remote_response(NOT_OWNER, received_srcid, my_id);
        default:
            error();
        free_entry();
endif;
```

Chapter 7 Address Collision Resolution Tables

7.1 Introduction

Address collisions are conflicts between incoming cache coherence requests to a processing element and outstanding cache coherence requests within it. A collision is usually due to a match between the associated addresses, but also may be because of a conflict for some internal resource such as a cache index. Within a processing element, actions taken in response to an address collision vary depending upon the outstanding request and the incoming request. These actions are described in Table 7-1 through Table 7-17. Non-cache coherent transactions (transactions specified in other RapidIO logical specifications) do not cause address collisions.

Some of the table entries specify that an outstanding request should be canceled at the local processor and that the incoming transaction then be issued immediately to the processor. This choosing between transactions is necessary to prevent deadlock conditions between multiple processing elements vying for ownership of a coherence granule.

7.2 Resolving an Outstanding READ_HOME Transaction

Table 7-1 describes the address collision resolution for an incoming transaction that collides with an outstanding READ_HOME transaction.

Table 7-1. Address Collision Resolution for READ_HOME

Outstanding Request	Incoming Request	Resolution
READ_HOME	READ_HOME	Generate “ERROR” response
READ_HOME	IREAD_HOME	Generate “ERROR” response
READ_HOME	READ_OWNER	Generate “NOT_OWNER” response
READ_HOME	READ_TO_OWN_HOME	Generate “ERROR” response
READ_HOME	READ_TO_OWN_OWNER	Generate “NOT_OWNER” response
READ_HOME	DKILL_HOME	Generate “ERROR” response
READ_HOME	DKILL_SHARER	If outstanding request, wait for all expected responses. If final response is “DONE”, return data if necessary and forward DKILL_SHARER to processor then generate a “DONE” response. If final response is “RETRY”, cancel the read at the processor and forward DKILL_SHARED to processor then generate a “DONE” response If no outstanding request, cancel the read at the processor and forward DKILL_SHARER to processor then generate a “DONE” response (this case should be very rare).
READ_HOME	CASTOUT	Generate “ERROR” response
READ_HOME	TLBIE	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
READ_HOME	TLBSYNC	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
READ_HOME	IKILL_HOME	Generate “ERROR” response
READ_HOME	IKILL_SHARER	No collision, forward to processor then generate “DONE” response (software must maintain instruction cache coherence)
READ_HOME	FLUSH	Generate “ERROR” response
READ_HOME	IO_READ_HOME	Generate “ERROR” response
READ_HOME	IO_READ_OWNER	Generate “NOT_OWNER” response

7.3 Resolving an Outstanding IREAD_HOME Transaction

Table 7-2 describes the address collision resolution for an incoming transaction that collides with an outstanding IREAD_HOME transaction.

Table 7-2. Address Collision Resolution for IREAD_HOME

Outstanding Request	Incoming Request	Resolution
IREAD_HOME	READ_HOME	Generate “ERROR” response
IREAD_HOME	IREAD_HOME	Generate “ERROR” response
IREAD_HOME	READ_OWNER	No collision, forward to processor then generate “DONE” response (software must maintain instruction cache coherence)
IREAD_HOME	READ_TO_OWN_HOME	Generate “ERROR” response
IREAD_HOME	READ_TO_OWN_OWNER	No collision, forward to processor then generate “DONE” response (software must maintain instruction cache coherence)
IREAD_HOME	DKILL_HOME	Generate “ERROR” response
IREAD_HOME	DKILL_SHARER	No collision, forward to processor then generate “DONE” response (software must maintain instruction cache coherence)
IREAD_HOME	CASTOUT	Generate “ERROR” response
IREAD_HOME	TLBIE	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
IREAD_HOME	TLBSYNC	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
IREAD_HOME	IKILL_HOME	Generate “ERROR” response
IREAD_HOME	IKILL_SHARER	No collision, forward to processor then generate “DONE” response (software must maintain instruction cache coherence)
IREAD_HOME	FLUSH	Generate “ERROR” response
IREAD_HOME	IO_READ_HOME	Generate “ERROR” response
IREAD_HOME	IO_READ_OWNER	No collision, forward to processor then generate “DONE” response (software must maintain instruction cache coherence)

7.4 Resolving an Outstanding READ_OWNER Transaction

Table 7-3 describes the address collision resolution for an incoming transaction that collides with an outstanding READ_OWNER transaction.

Table 7-3. Address Collision Resolution for READ_OWNER

Outstanding Request	Incoming Request	Resolution
READ_OWNER	READ_HOME	Generate “RETRY” response
READ_OWNER	IREAD_HOME	Generate “RETRY” response
READ_OWNER	READ_OWNER	Generate “ERROR” response
READ_OWNER	READ_TO_OWN_HOME	Generate “RETRY” response
READ_OWNER	READ_TO_OWN_OWNER	Generate “ERROR” response
READ_OWNER	DKILL_HOME	Generate “RETRY” response
READ_OWNER	DKILL_SHARER	Generate “ERROR” response
READ_OWNER	CASTOUT	No collision, update directory state, generate “DONE” response (CASTOUT bypasses address collision detection)
READ_OWNER	TLBIE	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
READ_OWNER	TLBSYNC	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
READ_OWNER	IKILL_HOME	No collision, forward to processor, send IKILL_SHARER to all participants except requestor (software must maintain instruction cache coherence)
READ_OWNER	IKILL_SHARER	Generate “ERROR” response
READ_OWNER	FLUSH	Generate “RETRY” response
READ_OWNER	IO_READ_HOME	Generate “RETRY” response
READ_OWNER	IO_READ_OWNER	Generate “ERROR” response

7.5 Resolving an Outstanding READ_TO_OWN_HOME Transaction

Table 7-4 describes the address collision resolution for an incoming transaction that collides with an outstanding READ_TO_OWN_HOME transaction.

Table 7-4. Address Collision Resolution for READ_TO_OWN_HOME

Outstanding Request	Incoming Request	Resolution
READ_TO_OWN_HOME	READ_HOME	Generate “ERROR” response
READ_TO_OWN_HOME	IREAD_HOME	Generate “ERROR” response
READ_TO_OWN_HOME	READ_OWNER	If outstanding request, wait for all expected responses. If final response is “DONE”, return data if necessary and forward READ_OWNER to processor and generate an “DONE_INTERVENTION” with data response and a “DATA_ONLY” to originator as in Section 3.3.1. If final response is “RETRY” generate an “ERROR” response If no outstanding request generate an “NOT_OWNER” response.
READ_TO_OWN_HOME	READ_TO_OWN_HOME	Generate “ERROR” response
READ_TO_OWN_HOME	READ_TO_OWN_OWNER	If outstanding request, wait for all expected responses. If final response is “DONE”, return data if necessary and forward READ_TO_OWN_OWNER to processor and generate an “DONE_INTERVENTION” with data response and a “DATA_ONLY” to originator as in Section 3.3.3. If final response is “RETRY” generate an “ERROR” response
READ_TO_OWN_HOME	DKILL_HOME	Generate “ERROR” response
READ_TO_OWN_HOME	DKILL_SHARER	If outstanding request, wait for all expected responses. If final response is “DONE” generate an “ERROR” response (we own the coherence granule and should never see a DKILL). If final response is “RETRY” generate a “DONE” response and continue the READ_TO_OWN_HOME. If no outstanding request generate a “DONE” response.
READ_TO_OWN_HOME	CASTOUT	Generate “ERROR” response
READ_TO_OWN_HOME	TLBIE	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
READ_TO_OWN_HOME	TLBSYNC	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
READ_TO_OWN_HOME	IKILL_HOME	Generate “ERROR” response
READ_TO_OWN_HOME	IKILL_SHARER	No collision, forward to processor then generate “DONE” response (software must maintain instruction cache coherence)

Table 7-4. Address Collision Resolution for READ_TO_OWN_HOME (Continued)

Outstanding Request	Incoming Request	Resolution
READ_TO_OWN_HOME	FLUSH	If outstanding request, wait for all expected responses. If final response is “DONE”, return data if necessary and forward FLUSH to processor and generate a “DONE” with data response as in Section 3.3.9. If final response is “RETRY” generate an “ERROR” response (we didn’t own the data and we lost at home memory) If no outstanding request generate an “ERROR” response (we didn’t own the data).
READ_TO_OWN_HOME	IO_READ_HOME	Generate “ERROR” response
READ_TO_OWN_HOME	IO_READ_OWNER	If outstanding request, wait for all expected responses. If final response is “DONE”, return data if necessary and forward IO_READ_OWNER to processor then generate a “DONE” with data response, etc. as in Section 3.3.10. If final response is “RETRY” generate an “ERROR” response (we didn’t own the data and we lost at home memory) If no outstanding request generate an “NOT_OWNER” response.

7.6 Resolving an Outstanding READ_TO_OWN_OWNER Transaction

Table 7-5 describes the address collision resolution for an incoming transaction that collides with an outstanding READ_TO_OWN_OWNER transaction.

Table 7-5. Address Collision Resolution for READ_TO_OWN_OWNER

Outstanding Request	Incoming Request	Resolution
READ_TO_OWN_OWNER	READ_HOME	Generate “RETRY” response
READ_TO_OWN_OWNER	IREAD_HOME	Generate “RETRY” response
READ_TO_OWN_OWNER	READ_OWNER	Generate “ERROR” response
READ_TO_OWN_OWNER	READ_TO_OWN_HOME	Generate “RETRY” response
READ_TO_OWN_OWNER	READ_TO_OWN_OWNER	Generate “ERROR” response
READ_TO_OWN_OWNER	DKILL_HOME	Generate “RETRY” response
READ_TO_OWN_OWNER	DKILL_SHARER	Generate “ERROR” response
READ_TO_OWN_OWNER	CASTOUT	No collision, update directory state, generate “DONE” response (CASTOUT bypasses address collision detection)
READ_TO_OWN_OWNER	TLBIE	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
READ_TO_OWN_OWNER	TLBSYNC	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
READ_TO_OWN_OWNER	IKILL_HOME	No collision, forward to processor, send IKILL_SHARER to all participants except requestor (software must maintain instruction cache coherence)
READ_TO_OWN_OWNER	IKILL_SHARER	Generate “ERROR” response
READ_TO_OWN_OWNER	FLUSH	Generate “RETRY” response
READ_TO_OWN_OWNER	IO_READ_HOME	Generate “RETRY” response
READ_TO_OWN_OWNER	IO_READ_OWNER	Generate “ERROR” response

7.7 Resolving an Outstanding DKILL_HOME Transaction

Table 7-6 describes the address collision resolution for an incoming transaction that collides with an outstanding DKILL_HOME transaction.

Table 7-6. Address Collision Resolution for DKILL_HOME

Outstanding Request	Incoming Request	Resolution
DKILL_HOME	READ_HOME	Generate “ERROR” response
DKILL_HOME	IREAD_HOME	Generate “ERROR” response
DKILL_HOME	READ_OWNER	If outstanding request, wait for all expected responses. If final response is “DONE”, return data if necessary and forward READ_OWNER to processor and generate a “DONE_INTERVENTION” with data response and a “DATA_ONLY” to originator as in Section 3.3.1. If final response is “RETRY” generate an “ERROR” response (we didn’t own the data and we lost at home memory) If no outstanding request generate an “ERROR” response (we didn’t own the data).
DKILL_HOME	READ_TO_OWN_HOME	Generate “ERROR” response
DKILL_HOME	READ_TO_OWN_OWNER	If outstanding request, wait for all expected responses. If final response is “DONE” forward READ_TO_OWN_OWNER to processor and generate a “DONE_INTERVENTION” with data response and a “DATA_ONLY” to originator as in Section 3.3.3. If final response is “RETRY” generate an “ERROR” response (we didn’t own the data and we lost at home memory) If no outstanding request generate an “ERROR” response (we didn’t own the data).
DKILL_HOME	DKILL_HOME	Generate “ERROR” response
DKILL_HOME	DKILL_SHARER	If outstanding request, wait for all expected responses. If final response is “DONE” generate an “ERROR” response (we should never see a DKILL_SHARER if we own the coherence granule). If final response is “RETRY” cancel the data cache invalidate at the processor and forward DKILL_SHARER to processor then generate a “DONE” response If no outstanding request, cancel the data cache invalidate at the processor and forward DKILL_SHARER to processor then generate a “DONE” response.
DKILL_HOME	CASTOUT	Generate “ERROR” response (cache paradox, can’t have a SHARED granule also MODIFIED in another processing element)
DKILL_HOME	TLBIE	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
DKILL_HOME	TLBSYNC	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
DKILL_HOME	IKILL_HOME	Generate “ERROR” response
DKILL_HOME	IKILL_SHARER	No collision, forward to processor then generate “DONE” response (software must maintain instruction cache coherence)

Table 7-6. Address Collision Resolution for DKILL_HOME (Continued)

Outstanding Request	Incoming Request	Resolution
DKILL_HOME	FLUSH	Generate “ERROR” response
DKILL_HOME	IO_READ_HOME	Generate “ERROR” response
DKILL_HOME	IO_READ_OWNER	If outstanding request, wait for all expected responses. If final response is “DONE” forward IO_READ_OWNER to processor then generate a “DONE” with data response, etc. as in Section 3.3.10. If final response is “RETRY” generate an “ERROR” response (we didn’t own the data and we lost at home memory) If no outstanding request generate an “ERROR” response (we didn’t own the data).

7.8 Resolving an Outstanding DKILL_SHARER Transaction

Table 7-7 describes the address collision resolution for an incoming transaction that collides with an outstanding DKILL_SHARER transaction.

Table 7-7. Address Collision Resolution for DKILL_SHARER

Outstanding Request	Incoming Request	Resolution
DKILL_SHARER	READ_HOME	Generate “RETRY” response
DKILL_SHARER	IREAD_HOME	Generate “RETRY” response
DKILL_SHARER	READ_OWNER	Generate “ERROR” response
DKILL_SHARER	READ_TO_OWN_HOME	Generate “RETRY” response
DKILL_SHARER	READ_TO_OWN_OWNER	Generate “ERROR” response
DKILL_SHARER	DKILL_HOME	Generate “RETRY” response
DKILL_SHARER	DKILL_SHARER	Generate “ERROR” response
DKILL_SHARER	CASTOUT	Generate “ERROR” response (cache paradox, can’t have a SHARED granule also MODIFIED in another processing element)
DKILL_SHARER	TLBIE	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
DKILL_SHARER	TLBSYNC	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
DKILL_SHARER	IKILL_HOME	No collision, forward to processor, send IKILL_SHARER to all participants except requestor (software must maintain instruction cache coherence)
DKILL_SHARER	IKILL_SHARER	Generate “ERROR” response
DKILL_SHARER	FLUSH	Generate “RETRY” response
DKILL_SHARER	IO_READ_HOME	If processing element is HOME: generate a “RETRY” response If processing element is not HOME: If outstanding request, wait for all expected responses. If final response is “DONE” forward IO_READ to processor then generate a “DONE” with data response, etc. as in Section 3.3.10. If final response is “RETRY” generate an “ERROR” response (we didn’t own the data and we lost at home memory) If no outstanding request generate an “ERROR” response (we didn’t own the data).
DKILL_SHARER	IO_READ_OWNER	Generate “ERROR” response

7.9 Resolving an Outstanding IKILL_HOME Transaction

Table 7-8 describes the address collision resolution for an incoming transaction that collides with an outstanding IKILL_HOME transaction.

Table 7-8. Address Collision Resolution for IKILL_HOME

Outstanding Request	Incoming Request	Resolution
IKILL_HOME	READ_HOME	Generate “ERROR” response
IKILL_HOME	IREAD_HOME	Generate “ERROR” response
IKILL_HOME	READ_OWNER	No collision, process normally
IKILL_HOME	READ_TO_OWN_HOME	Generate “ERROR” response
IKILL_HOME	READ_TO_OWN_OWNER	No collision, process normally
IKILL_HOME	DKILL_HOME	Generate “ERROR” response
IKILL_HOME	DKILL_SHARER	No collision, process normally
IKILL_HOME	CASTOUT	No collision, process normally
IKILL_HOME	TLBIE	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
IKILL_HOME	TLBSYNC	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
IKILL_HOME	IKILL_HOME	Generate “ERROR” response
IKILL_HOME	IKILL_SHARER	No collision, forward to processor then generate “DONE” response (software must maintain instruction cache coherence)
IKILL_HOME	FLUSH	Generate “ERROR” response
IKILL_HOME	IO_READ_HOME	Generate “ERROR” response
IKILL_HOME	IO_READ_OWNER	No collision, process normally

7.10 Resolving an Outstanding IKILL_SHARER Transaction

Table 7-9 describes the address collision resolution for an incoming transaction that collides with an outstanding IKILL_SHARER transaction.

Table 7-9. Address Collision Resolution for IKILL_SHARER

Outstanding Request	Incoming Request	Resolution
IKILL_SHARER	READ_HOME	No collision, process normally
IKILL_SHARER	IREAD_HOME	No collision, process normally
IKILL_SHARER	READ_OWNER	Generate “ERROR” response
IKILL_SHARER	READ_TO_OWN_HOME	No collision, process normally
IKILL_SHARER	READ_TO_OWN_OWNER	Generate “ERROR” response
IKILL_SHARER	DKILL_HOME	No collision, process normally
IKILL_SHARER	DKILL_SHARER	Generate “ERROR” response
IKILL_SHARER	CASTOUT	No collision, process normally
IKILL_SHARER	TLBIE	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
IKILL_SHARER	TLBSYNC	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
IKILL_SHARER	IKILL_HOME	No collision, forward to processor, send IKILL_SHARER to all participants except requestor (software must maintain instruction cache coherence)
IKILL_SHARER	IKILL_SHARER	Generate “ERROR” response
IKILL_SHARER	FLUSH	No collision, process normally
IKILL_SHARER	IO_READ_HOME	If processing element is HOME: generate a “RETRY” response If processing element is not HOME: If outstanding request, wait for all expected responses. If final response is “DONE” forward IO_READ to processor then generate a “DONE” with data response, etc. as in Section 3.3.10. If final response is “RETRY” generate an “ERROR” response (we didn’t own the data and we lost at home memory) If no outstanding request generate an “ERROR” response (we didn’t own the data).
IKILL_SHARER	IO_READ_OWNER	Generate “ERROR” response

7.11 Resolving an Outstanding CASTOUT Transaction

Table 7-10 describes the address collision resolution for an incoming transaction that collides with an outstanding CASTOUT transaction.

Table 7-10. Address Collision Resolution for CASTOUT

Outstanding Request	Incoming Request	Resolution
CASTOUT	READ_HOME	Generate “ERROR” response
CASTOUT	IREAD_HOME	Generate “ERROR” response
CASTOUT	READ_OWNER	Generate “RETRY” response; the CASTOUT will bypass address collision at home memory and modify the directory state
CASTOUT	READ_TO_OWN_HOME	Generate “ERROR” response
CASTOUT	READ_TO_OWN_OWNER	Generate “RETRY” response; the CASTOUT will bypass address collision at home memory and modify the directory state
CASTOUT	DKILL_HOME	Generate “ERROR” response
CASTOUT	DKILL_SHARER	Generate “ERROR” response
CASTOUT	CASTOUT	Generate “ERROR” response
CASTOUT	TLBIE	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
CASTOUT	TLBSYNC	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
CASTOUT	IKILL_HOME	Generate “ERROR” response
CASTOUT	IKILL_SHARER	No collision, forward to processor then generate “DONE” response (software must maintain instruction cache coherence)
CASTOUT	FLUSH	Generate “ERROR” response
CASTOUT	IO_READ_HOME	Generate “ERROR” response
CASTOUT	IO_READ_OWNER	Generate “RETRY” response; the CASTOUT will bypass address collision at home memory and modify the directory state

7.12 Resolving an Outstanding TLBIE or TLBSYNC Transaction

Table 7-11 describes the address collision resolution for an incoming transaction that collides with an outstanding TLBIE or TLBSYNC transaction.

Table 7-11. Address Collision Resolution for Software Coherence Operations

Outstanding Request	Incoming Request	Resolution
TLBIE, TLBSYNC	ANY	No collision, process request as described in Chapter 6, “Communication Protocols”

7.13 Resolving an Outstanding FLUSH Transaction

The flush operation has two distinct versions. The first is for processing elements that participate in the coherence protocol such as a processor and it’s associated agent, which may also have a local I/O device. The second is for processing elements that do not participate in the coherence protocols such as a pure I/O device that does not have a corresponding bit in the directory sharing mask. Table 7-12 describes the address collision resolution for an incoming transaction that collides with an outstanding participant FLUSH transaction.

Table 7-12. Address Collision Resolution for Participant FLUSH

Outstanding Request	Incoming Request	Resolution
FLUSH	READ_HOME	Generate “ERROR” response
FLUSH	IREAD_HOME	Generate “ERROR” response
FLUSH	READ_OWNER	Generate “NOT_OWNER” response (we are not allowed to issue FLUSH to an owned coherence granule - should be a CASTOUT)
FLUSH	READ_TO_OWN_HOME	Generate “ERROR” response
FLUSH	READ_TO_OWN_OWNER	Generate “NOT_OWNER” response (we are not allowed to issue FLUSH to an owned coherence granule - should be a CASTOUT)
FLUSH	DKILL_HOME	Generate “ERROR” response
FLUSH	DKILL_SHARER	If outstanding request, wait for all expected responses. If final response is “DONE” generate an “ERROR” response (we should never see a DKILL_SHARER if we own the coherence granule). If final response is “RETRY” cancel the flush at the processor and forward DKILL_SHARER to processor then generate a “DONE” response If no outstanding request, cancel the data cache invalidate at the processor and forward DKILL_SHARER to processor then generate a “DONE” response.
FLUSH	CASTOUT	Generate “ERROR” response
FLUSH	TLBIE	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
FLUSH	TLBSYNC	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
FLUSH	IKILL_HOME	Generate “ERROR” response
FLUSH	IKILL_SHARER	No collision, forward to processor then generate “DONE” response (software must maintain instruction cache coherence)
FLUSH	FLUSH	Generate “ERROR” response
FLUSH	IO_READ_HOME	Generate “ERROR” response
FLUSH	IO_READ_OWNER	Generate “NOT_OWNER” response (we are not allowed to issue FLUSH to an owned coherence granule - should be a CASTOUT)

Table 7-13 describes the address collision resolution for an incoming transaction that collides with an outstanding non-participant FLUSH transaction.

Table 7-13. Address Collision Resolution for Non-participant FLUSH

Outstanding Request	Incoming Request	Resolution
FLUSH	READ_HOME	Generate “ERROR” response (should never receive coherent operation)
FLUSH	IREAD_HOME	Generate “ERROR” response (should never receive coherent operation)
FLUSH	READ_OWNER	Generate “ERROR” response (should never receive coherent operation)
FLUSH	READ_TO_OWN_HOME	Generate “ERROR” response (should never receive coherent operation)
FLUSH	READ_TO_OWN_OWNER	Generate “ERROR” response (should never receive coherent operation)
FLUSH	DKILL_HOME	Generate “ERROR” response (should never receive coherent operation)
FLUSH	DKILL_SHARER	Generate “ERROR” response (should never receive coherent operation)
FLUSH	CASTOUT	Generate “ERROR” response (should never receive coherent operation)
FLUSH	TLBIE	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence) - non-participant may have page table hardware.
FLUSH	TLBSYNC	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence) - non-participant may have page table hardware.
FLUSH	IKILL_HOME	Generate “ERROR” response
FLUSH	IKILL_SHARER	No collision, forward to processor then generate “DONE” response (software must maintain instruction cache coherence) - non-participant may have software coherence.
FLUSH	FLUSH	Generate “ERROR” response (should never receive coherent operation)
FLUSH	IO_READ_HOME	Generate “ERROR” response (should never receive coherent operation)
FLUSH	IO_READ_OWNER	Generate “ERROR” response (should never receive coherent operation)

7.14 Resolving an Outstanding IO_READ_HOME Transaction

The I/O read operation is used by processing elements that do not want to participate in the coherence protocol but do want to get current copies of cached data. There are two versions of this operation, one for processing elements that have both processors and I/O devices, the second for pure I/O devices that do not have a corresponding bit in the directory sharing mask. Table 7-14 describes the address collision resolution for an incoming transaction that collides with an outstanding participant IO_READ_HOME transaction.

Table 7-14. Address Collision Resolution for Participant IO_READ_HOME

Outstanding Request	Incoming Request	Resolution
IO_READ_HOME	READ_HOME	Generate “ERROR” response
IO_READ_HOME	IREAD_HOME	Generate “ERROR” response
IO_READ_HOME	READ_OWNER	Generate “NOT_OWNER” response (we don’t own the data otherwise we could have obtained a copy locally)
IO_READ_HOME	READ_TO_OWN_HOME	Generate “ERROR” response
IO_READ_HOME	READ_TO_OWN_OWNER	Generate “NOT_OWNER” response (we don’t own the data otherwise we could have obtained a copy locally)
IO_READ_HOME	DKILL_HOME	Generate “ERROR” response
IO_READ_HOME	DKILL_SHARER	If outstanding request, wait for all expected responses. If final response is “DONE”, return data if necessary and forward DKILL_SHARER to processor then generate a “DONE” response. If final response is “RETRY” forward DKILL_SHARED to processor then generate a “DONE” response If no outstanding request forward DKILL_SHARER to processor then generate a “DONE” response
IO_READ_HOME	CASTOUT	Generate “ERROR” response
IO_READ_HOME	TLBIE	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
IO_READ_HOME	TLBSYNC	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
IO_READ_HOME	IKILL_HOME	Generate “ERROR” response
IO_READ_HOME	IKILL_SHARER	No collision, forward to processor then generate “DONE” response (software must maintain instruction cache coherence)
IO_READ_HOME	FLUSH	Generate “ERROR” response
IO_READ_HOME	IO_READ_HOME	Generate “ERROR” response
IO_READ_HOME	IO_READ_OWNER	Generate “NOT_OWNER” response (we don’t own the data otherwise we could have obtained a copy locally)

Table 7-15 describes the address collision resolution for an incoming transaction that collides with an outstanding non-participant IO_READ_HOME transaction.

Table 7-15. Address Collision Resolution for Non-participant IO_READ_HOME

Outstanding Request	Incoming Request	Resolution
IO_READ_HOME	READ_HOME	Generate “ERROR” response (should never receive coherent operation)
IO_READ_HOME	IREAD_HOME	Generate “ERROR” response (should never receive coherent operation)
IO_READ_HOME	READ_OWNER	Generate “ERROR” response (should never receive coherent operation)
IO_READ_HOME	READ_TO_OWN_HOME	Generate “ERROR” response (should never receive coherent operation)
IO_READ_HOME	READ_TO_OWN_OWNER	Generate “ERROR” response (should never receive coherent operation)
IO_READ_HOME	DKILL_HOME	Generate “ERROR” response (should never receive coherent operation)
IO_READ_HOME	DKILL_SHARER	Generate “ERROR” response (should never receive coherent operation)
IO_READ_HOME	CASTOUT	Generate “ERROR” response (should never receive coherent operation)
IO_READ_HOME	TLBIE	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence) - broadcast operation and non-participant may have page table hardware.
IO_READ_HOME	TLBSYNC	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence) - broadcast operation and non-participant may have page table hardware.
IO_READ_HOME	IKILL_HOME	Generate “ERROR” response
IO_READ_HOME	IKILL_SHARER	No collision, forward to processor then generate “DONE” response (software must maintain instruction cache coherence) - broadcast operation and non-participant may have software coherence.
IO_READ_HOME	FLUSH	Generate “ERROR” response (should never receive coherent operation)
IO_READ_HOME	IO_READ_HOME	Generate “ERROR” response (should never receive coherent operation)
IO_READ_HOME	IO_READ_OWNER	Generate “ERROR” response (should never receive coherent operation)

7.15 Resolving an Outstanding IO_READ_OWNER Transaction

The I/O read operation is used by processing elements that do not want to participate in the coherence protocol but do want to get current copies of cached data. There are two versions of this operation, one for processing elements that have both processors and I/O devices, the second for pure I/O devices that do not have a corresponding bit in the directory sharing mask. Table 7-16 describes the address collision resolution for an incoming transaction that collides with an outstanding IO_READ_OWNER transaction.

Table 7-16. Address Collision Resolution for Participant IO_READ_OWNER

Outstanding Request	Incoming Request	Resolution
IO_READ_OWNER	READ_HOME	Generate “RETRY” response
IO_READ_OWNER	IREAD_HOME	Generate “RETRY” response
IO_READ_OWNER	READ_OWNER	Generate “ERROR” response
IO_READ_OWNER	READ_TO_OWN_HOME	Generate “RETRY” response
IO_READ_OWNER	READ_TO_OWN_OWNER	Generate “ERROR” response
IO_READ_OWNER	DKILL_HOME	Generate “RETRY” response
IO_READ_OWNER	DKILL_SHARER	Generate “ERROR” response
IO_READ_OWNER	CASTOUT	No collision, update directory state and memory, generate DONE response (CASTOUT bypasses address collision detection)
IO_READ_OWNER	TLBIE	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
IO_READ_OWNER	TLBSYNC	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence)
IO_READ_OWNER	IKILL_HOME	No collision, forward to processor, send IKILL_SHARER to all participants except requestor (software must maintain instruction cache coherence)
IO_READ_OWNER	IKILL_SHARER	Generate “ERROR” response
IO_READ_OWNER	FLUSH	Generate “RETRY” response
IO_READ_OWNER	IO_READ_HOME	Generate “RETRY” response
IO_READ_OWNER	IO_READ_OWNER	Generate “ERROR” response (we don’t own the data otherwise we could have obtained a copy locally)

Table 7-17 describes the address collision resolution for an incoming transaction that collides with an outstanding non-participant IO_READ_OWNER transaction.

Table 7-17. Address Collision Resolution for Non-participant IO_READ_OWNER

Outstanding Request	Incoming Request	Resolution
IO_READ_OWNER	READ_HOME	Generate “ERROR” response (should never receive coherent operation)
IO_READ_OWNER	IREAD_HOME	Generate “ERROR” response (should never receive coherent operation)
IO_READ_OWNER	READ_OWNER	Generate “ERROR” response (should never receive coherent operation)
IO_READ_OWNER	READ_TO_OWN_HOME	Generate “ERROR” response (should never receive coherent operation)
IO_READ_OWNER	READ_TO_OWN_OWNER	Generate “ERROR” response (should never receive coherent operation)
IO_READ_OWNER	DKILL_HOME	Generate “ERROR” response (should never receive coherent operation)
IO_READ_OWNER	DKILL_SHARER	Generate “ERROR” response (should never receive coherent operation)
IO_READ_OWNER	CASTOUT	Generate “ERROR” response (should never receive coherent operation)
IO_READ_OWNER	TLBIE	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence) - non-participant may have page table hardware.
IO_READ_OWNER	TLBSYNC	No collision, forward to processor then generate “DONE” response (software must maintain TLB entry coherence) - non-participant may have page table hardware.
IO_READ_OWNER	IKILL_HOME	No collision, forward to processor, send IKILL_SHARER to all participants except requestor (software must maintain instruction cache coherence)
IO_READ_OWNER	IKILL_SHARER	Generate “ERROR” response
IO_READ_OWNER	FLUSH	Generate “ERROR” response (should never receive coherent operation)
IO_READ_OWNER	IO_READ_HOME	Generate “ERROR” response (should never receive coherent operation)
IO_READ_OWNER	IO_READ_OWNER	Generate “ERROR” response (should never receive coherent operation)

Glossary of Terms and Abbreviations

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

A

Address collision. An address based conflict between two or more cache coherence operations when referencing the same coherence granule.

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

Asychronous transfer mode (**ATM**). A standard networking protocol which dynamically allocates bandwidth using a fixed-size packet.

B

Big-endian. A byte-ordering method in memory where the address n of a word corresponds to the most significant byte. In an addressed memory word, the bytes are ordered (left to right) 0, 1, 2, 3, with 0 being the most significant byte.

Block flush. An operation that returns the latest copy of a block of data from caches within the system to memory.

Bridge. A processing element that connects one computer bus to another, allowing a processing element on one bus to access an processing element on the other.

Broadcast. The concept of sending a packet to all processing elements in a system.

Bus-based snoopy protocol. A broadcast cache coherence protocol that assumes that all caches in the system are on a common bus.

C

Cache. High-speed memory containing recently accessed data and/or instructions (subset of main memory) associated with a processor.

Cache coherence. Caches are coherent if a processor performing a read from its cache is supplied with data corresponding to the most recent value written to memory or to another processor’s cache. In other words, a write operation to an address in the system is visible to all other caches in the system. Also referred to as memory coherence.

Cache coherent-non uniform memory access (**CC-NUMA**). A cache coherent system in which memory accesses have different latencies depending upon the physical location of the accessed address.

Cache paradox. A circumstance in which the caches in a system have an undefined or disallowed state for a coherence granule, for example, two caches have the same coherence granule marked “modified”.

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

Castout operation. An operation used by a processing element to relinquish its ownership of a coherence granule and return it to home memory.

Coherence domain. A logically associated group of processing elements that participate in the globally shared memory protocol and are able to maintain cache coherence among themselves.

Coherence granule. A contiguous block of data associated with an address for the purpose of guaranteeing cache coherence.

Command and status registers (CSRs). A set of registers that allows a processing element to control and determine the status of another processing element’s internal hardware.

D

Deadlock. A situation in which two processing elements that are sharing resources prevent each other from accessing the resources, resulting in a halt of system operation.

Destination. The termination point of a packet on the RapidIO interconnect, also referred to as a target.

Device. A generic participant on the RapidIO interconnect that sends or receives RapidIO transactions, also called a processing element.

Device ID. The identifier of an end point processing element connected to the RapidIO interconnect.

Direct Memory Access (**DMA**). The process of accessing memory in a device by specifying the memory address directly.

Distributed memory. System memory that is distributed throughout the system, as opposed to being centrally located.

Domain. A logically associated group of processing elements.

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

E

End point. A processing element which is the source or destination of transactions through a RapidIO fabric.

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

Exclusive. A processing element has the only cached copy of a sharable coherence granule. The exclusive state allows the processing element to modify the coherence granule without notifying the rest of the system.

F

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

Flush operation. An operation used by a processing element to return the ownership and current data of a coherence granule to home memory.

G

Globally shared memory (GSM). Cache coherent system memory that can be shared between multiple processors in a system.

H

Half-word. A two byte or 16 bit quantity, aligned on two byte boundaries.

Home memory. The physical memory corresponding to the physical address of a coherence granule.

I

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

Instruction cache. High-speed memory containing recently accessed instructions (subset of main memory) associated with a processor.

Instruction cache invalidate operation. An operation that is used if the instruction cache coherence must be maintained by software.

Instruction read operation. An operation used to obtain a globally shared copy of a coherence granule specifically for an instruction cache.

Instruction set architecture (ISA). The instruction set for a certain processor or family of processors.

Intervention. A data transfer between two processing elements that does not go through the coherence granule's home memory, but directly between the requestor of the coherence granule and the current owner.

Invalidate operation. An operation used to remove a coherence granule from caches within the coherence domain.

I/O. Input-output.

I/O read operation. An operation used by an I/O processing element to obtain a globally shared copy of a coherence granule without disturbing the coherence state of the granule.

L

Little-endian. A byte-ordering method in memory where the address n of a word corresponds to the least significant byte. In an addressed memory word, the bytes are ordered (left to right) 3, 2, 1, 0, with 3 being the most significant byte.

Local memory. Memory associated with the processing element in question.

LSB. Least significant byte.

M

Memory coherence. Memory is coherent if a processor performing a read from its cache is supplied with data corresponding to the most recent value written to memory or to another processor’s cache. In other words, a write operation to an address in the system is visible to all other caches in the system. Also referred to as cache coherence.

Memory controller. The point through which home memory is accessed.

Memory directory. A table of information associated with home memory that is used to track the location and state of coherence granules cached by coherence domain participants.

Message passing. An application programming model that allows processing elements to communicate via messages to mailboxes instead of via DMA or GSM. Message senders do not write to a memory address in the receiver.

Modified. A processing element has written to a locally cached coherence granule and so has the only valid copy of the coherence granule in the system.

Modified exclusive shared invalid (**MESI**). A standard 4 state cache coherence definition.

Modified shared invalid (**MSI**). A standard 3 state cache coherence definition.

Modified shared local (**MSL**). A standard 3 state cache coherence definition.

MSB. Most significant byte.

Multicast. The concept of sending a packet to more than one processing elements in a system.

N

Non-coherent. A transaction that does not participate in any system globally shared memory cache coherence mechanism.

O

Operation. A set of transactions between end point devices in a RapidIO system (requests and associated responses) such as a read or a write.

Ownership. A processing element has the only valid copy of a coherence granule and is responsible for returning it to home memory.

P

Packet. A set of information transmitted between devices in a RapidIO system.

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

Priority. The relative importance of a packet; in most systems a higher priority packet will be serviced or transmitted before one of lower priority.

Processing Element (**PE**). A generic participant on the RapidIO interconnect that sends or receives RapidIO transactions, also called a device.

Processor. The logic circuitry that responds to and processes the basic instructions that drive a computer.

R

Read operation. An operation used to obtain a globally shared copy of a coherence granule.

Read-for-ownership operation. An operation used to obtain ownership of a coherence granule for the purposes of performing a write operation.

Remote access. An access by a processing element to memory located in another processing element.

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

S

Shared. A processing element has a cached copy of a coherence granule that may be cached by other processing elements and is consistent with the copy in home memory.

Sharing mask. The state associated with a coherence granule in the memory directory that tracks the processing elements that are sharing the coherence granule.

Source. The origin of a packet on the RapidIO interconnect, also referred to as an initiator.

Sub-double-word. Aligned on eight byte boundaries.

Switch. A multiple port processing element that directs a packet received on one of its input ports to one of its output ports.

T

Target. The termination point of a packet on the RapidIO interconnect, also referred to as a destination.

Transaction. A specific request or response packet transmitted between end point devices in a RapidIO system.

Translation look-aside buffer (TLB). Part of a processor's memory management unit; a TLB contains a set of virtual to physical page address translations, along with a set of attributes that describe access behavior for that portion of physical memory.

W

Write-through. A cache policy that passes all write operations through the caching hierarchy directly to home memory.

Word. A four byte or 32 bit quantity, aligned on four byte boundaries.

RapidIO™ Interconnect Specification Part 6: LP-Serial Physical Layer Specification

3.2, 1/2016

RapidIO.org

Revision History

Revision	Description	Date
1.1	First release	12/17/2001
1.2	Technical changes: incorporate Rev. 1.1 errata rev. 1.1.1, errata 3	06/26/2002
1.3	Technical changes: incorporate Rev 1.2 errata 1 as applicable, the following errata showings: 03-03-00004.002, 03-07-00002.001, 03-12-00000.002, 03-12-00002.004, 04-02-00000.001, 04-05-00000.003, 04-05-00006.002 (partial), 04-05-00007.001 and the following new features showings: 02-03-0003.004, 02-06-00001.004, 04-08-00013.002, 04-09-00022.002 Converted to ISO-friendly templates	02/23/2005
2.0	Significant editorial changes Technical changes: errata showings 04-11-00031.001, 06-04-00000.003, 06-07-00001.001, 07-03-00000.002, 07-03-00001.001 new features showings 05-04-00001.005, 05-04-00003.004, new speed bin and width definitions with supporting protocol	06/14/2007
2.0.1	Very minor editorial changes	08/29/2007
2.1	Significant editorial changes Technical changes: errata showings 07-09-00000.003, 07-06-00000.010, 08-05-00001.003, 08-03-00000.001, 08-02-00000.008, 08-06-00001.004, 07-11-00001.010, 08-10-00000.003, 08-11-00001.000	MM/DD/200Y
2.2	Significant editorial changes Technical changes: errata showings 09-08-00000.005, 09-09-00000.004, 10-02-00000.001, 10-03-00000.004, 10-01-00003.006, 10-08-00000.003, 10-08-00001.005, 10-10-00000.002, 10-10-00001.004, 10-11-00002.001, 11-02-00000.002, Consolidated Comments on 11-01-00000.000	05/05/2011
3.0	Changed RTA contact information. Technical changes: Addition of 10xN serial physical layer, including: • 10.3125 Gbaud lane speed • 10.3125 Gbaud electrical specifications • 64b/67b encoding: codewords, ordered sequences, and IDLE3 • Increase ackID size for IDLE3 to 12 bits • Control symbol encoding • CRC32 link level error checking • New per-port register block format, with new/modified registers • Modified per-Lane Registers • Specific link initialization state machines to support initialization of 10.3125 Gbaud links • Asymmetric operation of 10.3125 Gbaud links Changed input/output error recovery protocol to enable faster recovery. Allowed Packet Accepted control symbols to acknowledge multiple packets. Added time synchronization support, including control symbol definition, register block definition and control symbol protocol. Added support for larger packets as created by addition of Dev32 and 64b/67b encoding scheme. Completed numerous other editorial changes to improve readability and clarify intent.	10/11/2013

Revision	Description	Date
3.1	Numerous typographical errors corrected and specification clarifications made. Technical changes: Resolution of errata against the 3.0 specification. Addition of MECS Time Synchronization support and registers Addition of Structurally Asymmetric Links support and registers Addition of Pseudo Random Binary Sequence (PRBS) test support and registers Clarified alignment field treatment in CRC24 computation. Clarified that in IDLE3 the link-request control symbol shall start in Lane 0. Added statement regarding transmit emphasis register control to section 4.13.2 Added statement regarding transmit emphasis register control to section 5.18 Added definition of LR_initialize, governing beginning transmit emphasis settings, to section 5.19.2.	09/18/2014
3.2	Changes as required to add 12.5 Gbaud line rate to specification Changes required to resolve errata: Errata 10 Change to Retrain/Xmt_Width_Control State Machine Errata 11 Corrections to Typical Data Flow Diagrams Errata 12 Transmit Emphasis Timeout Control Clarification Errata 14 Seed Control Word Bit Ordering Clarification Errata 15 Descrambler Seed Ordered Sequence Spacing Clarification	01/28/2016

Table of Contents

Chapter 1 Overview

1.1 Introduction ... 31
1.2 Contents ... 32
1.3 Terminology ... 34
1.4 Conventions ... 35

Chapter 2 Packets

2.1 Introduction ... 37
2.2 Packet Field Definitions ... 37
2.3 Packet Format ... 39
2.4 Packet Protection ... 39
2.4.1 Packet CRC Operation ... 40
2.4.2 CRC-16 Code ... 42
2.5 Maximum Packet Size ... 44

Chapter 3 Control Symbols

3.1 Introduction ... 45
3.2 Control Symbol Field Definitions ... 47
3.3 Control Symbol Format ... 47
3.4 Stype0 Control Symbols ... 49
3.4.1 Packet-Accepted Control Symbol ... 52
3.4.2 Packet-Retry Control Symbol ... 52
3.4.3 Packet-Not-Accepted Control Symbol ... 52
3.4.4 Timestamp Control Symbol ... 54
3.4.5 Status Control Symbol ... 54
3.4.6 VC-Status Control Symbol ... 55
3.4.7 Link-Response Control Symbol ... 56
3.4.8 Loop-Response Control Symbol ... 57
3.5 Stype1 Control Symbols ... 58
3.5.1 Start-of-Packet Control Symbol ... 59
3.5.2 Stomp Control Symbol ... 60
3.5.3 End-of-Packet Control Symbol ... 60
3.5.4 Restart-From-Retry Control Symbol ... 61
3.5.5 Link-Request Control Symbol ... 61
3.5.5.1 Reset-port Command ... 62
3.5.5.2 Reset-Device Command ... 63
3.5.5.3 Port-status Command ... 64
3.5.6 Timing Control Symbols ... 64
3.5.6.1 Multicast-Event Control Symbol ... 64

Table of Contents

3.5.6.2 Secondary Multicast-Event Control Symbol 64
3.5.6.3 Loop-Timing Control Symbol 65
3.6 Control Symbol Protection 65
3.6.1 CRC-5 Code 65
3.6.2 CRC-5 Parallel Code Generation 66
3.6.3 CRC-13 Code 67
3.6.4 CRC-13 Parallel Code Generation 67
3.6.5 CRC-24 Code 68
3.6.6 CRC-24 Parallel Code Generation 69

Chapter 4 8b/10b PCS and PMA Layers

4.1 Introduction 71
4.2 PCS Layer Functions 71
4.3 PMA Layer Functions 72
4.4 Definitions 72
4.5 8b/10b Transmission Code 73
4.5.1 Character and Code-Group Notation 74
4.5.2 Running Disparity 75
4.5.3 Running Disparity Rules 75
4.5.4 8b/10b Encoding 76
4.5.5 Transmission Order 76
4.5.6 8b/10b Decoding 77
4.5.7 Special Characters and Columns 86
4.5.7.1 Packet Delimiter Control Symbol (/PD/) 87
4.5.7.2 Start of Control Symbol (/SC/) 87
4.5.7.3 Idle (/I/) 87
4.5.7.4 Sync (/K/) 87
4.5.7.5 Skip (/R/) 87
4.5.7.6 Align (/A/) 87
4.5.7.7 Mark (/M/) 87
4.5.7.8 Illegal 87
4.5.8 Effect of Single Bit Code-Group Errors 88
4.6 LP-Serial Link Widths 88
4.7 Idle Sequence 89
4.7.1 Clock Compensation Sequence 90
4.7.2 Idle Sequence 1 (IDLE1) 91
4.7.3 Idle Sequence 1 Generation 92
4.7.4 Idle Sequence 2 (IDLE2) 93
4.7.4.1 Idle Frame 94
4.7.4.1.1 IDLE Sequence 2 Random Data Field 95
4.7.4.1.2 IDLE Sequence 2 CS Field Marker 96
4.7.4.1.3 IDLE2 Command and Status Field (CS field) 98
4.7.4.1.4 IDLE2 CS Field Use 100
4.7.5 Idle Sequence Selection 102
4.8 Scrambling 103

Table of Contents

4.8.1 Scrambling Rules 104
4.8.2 Descrambler Synchronization 106
4.8.3 Descrambler Synchronization Verification 107
4.9 1x Mode Transmission Rules 108
4.9.1 1x Ports 108
4.9.2 Nx Ports Operating in 1x Mode 110
4.10 Nx Link Striping and Transmission Rules 111
4.11 Retimers and Repeaters 113
4.11.1 Retimers 114
4.11.2 Repeaters 114
4.12 Port Initialization 115
4.12.1 1x Mode Initialization 115
4.12.2 1x/Nx Mode Initialization 115
4.12.3 Baud Rate Discovery 116
4.12.4 State Machines 118
4.12.4.1 State Machine Conventions, Functions and Variables 118
4.12.4.1.1 State Machine Conventions 118
4.12.4.1.2 State Machine Functions 118
4.12.4.1.3 State Machine Variables 119
4.12.4.2 Lane Synchronization State Machine 123
4.12.4.3 Lane Alignment State Machine 127
4.12.4.4 1x/2x Mode Detect State Machine 129
4.12.4.5 1x Mode Initialization State Machine 130
4.12.4.6 1x/Nx Mode Initialization State Machine for N = 4, 8, 16 131
4.12.4.7 1x/2x Mode Initialization State Machine 137
4.12.4.8 1x/Mx/Nx Mode Initialization State Machines 140
4.12.4.8.1 1x/2x/Nx Initialization State Machine 140
4.12.4.8.2 1x/Mx/Nx Initialization State Machine (N > M > 2) 144
4.13 Structurally Asymmetric Links 147
4.13.1 Definitions 147
4.13.2 Structurally Asymmetric Link Operation 147
4.14 Pseudo Random Binary Sequence Testing 149

Chapter 5 64b/67b PCS and PMA Layers

5.1 Introduction 153
5.2 PCS Layer Functions 153
5.3 PMA Layer Functions 154
5.4 Definitions 154
5.5 64b/67b Transmission Code 156
5.5.1 Codeword Format 156
5.5.2 Data Codeword 157
5.5.3 Control Codeword 157
5.5.3.1 Skip-Marker Control Codeword 158
5.5.3.2 Lane-Check Control Codeword 159
5.5.3.3 Descrambler Seed Control Codeword 160

Table of Contents

5.5.3.4 Skip Control Codeword 160
5.5.3.5 Status/Control Control Codeword 160
5.5.3.6 CSB Control Codeword 165
5.5.3.7 CSE Control Codeword 165
5.5.3.8 CSEB Control Codeword 166
5.5.4 Scrambling 167
5.5.4.1 Scrambling Rules 167
5.5.4.2 Descrambler Synchronization 169
5.5.5 Selective Codeword Inversion 170
5.5.5.1 Selective Codeword Inversion Rules 170
5.5.6 Lane Check Calculation 171
5.5.7 Transmission Order 174
5.6 Packet Transmission Rules 174
5.6.1 Link CRC-32 Code 175
5.7 Packet Delimiting and Alignment 177
5.7.1 Packet Start Delimiter 177
5.7.2 Packet Termination Delimiters 177
5.8 Control Symbol Transmission Rules 178
5.9 Ordered Sequences 178
5.9.1 Seed Ordered Sequence 179
5.9.2 Status/Control Ordered Sequence 180
5.9.3 Skip Ordered Sequence 180
5.10 Idle Sequence 181
5.10.1 Idle Sequence 3 (IDLE3) 182
5.10.2 Idle Sequence 3 Generation 183
5.11 Adaptive Equalization 185
5.11.1 Lane Training/Retraining 186
5.11.2 Ports Operating at 10.3125 and 12.5 Gbaud 187
5.11.2.1 Long run 10.3125 and 12.5 Gbaud training 187
5.11.2.2 Short run 10.3125 and 12.5 Gbaud training 187
5.11.2.3 10.3125 and 12.5 Gbaud retraining 189
5.12 LP-Serial Link Widths 189
5.13 Transmission Rules 190
5.13.1 Order of Operation 190
5.13.2 1x Ports 190
5.13.3 Nx Ports Operating in 1x Mode 191
5.13.4 Kx Link Striping and Transmission Rules 192
5.14 Effect of Transmission Errors and Error Detection 194
5.15 Retimers and Repeaters 195
5.15.1 Retimers 195
5.15.2 Repeaters 196
5.16 Port Initialization 196
5.16.1 1x Mode Initialization 197
5.16.2 1x/Nx Mode Initialization 197
5.16.3 Baud Rate Discovery 197
5.17 Asymmetric Operation 199

Table of Contents

5.17.1 Port Transmission Width ... 200
5.17.1.1 Port transmission width commands ... 200
5.17.1.1.1 Transmit width port command protocol ... 200
5.17.1.2 Port transmission width requests ... 202
5.17.2 Port Receive Width ... 202
5.17.2.1 Receive Width Link Command Protocol ... 203
5.18 Structurally Asymmetric Links ... 204
5.19 State Machines ... 205
5.19.1 State Machine Conventions, Functions and Variables ... 205
5.19.1.1 State Machine Conventions ... 205
5.19.1.2 State Machine Functions ... 206
5.19.1.3 State Machine Variables ... 207
5.19.2 Frame_Lock State Machine ... 224
5.19.3 Lane Training State Machines ... 224
5.19.3.1 Long run Lane_Training State Machine ... 225
5.19.3.2 Short run Lane_Training state machine ... 230
5.19.4 Codeword Lock State Machine ... 232
5.19.5 Lane Synchronization State Machine ... 234
5.19.5.1 Entering Silence ... 235
5.19.5.1.1 Transmitter procedure ... 236
5.19.5.1.2 Receiver procedure ... 236
5.19.6 Lane Alignment State Machine ... 237
5.19.7 Port Initialization State Machine ... 239
5.19.7.1 1x/2x/Nx Initialization State Machine ... 240
5.19.8 Retrain/Transmit_Width_Control State Machine ... 247
5.19.9 Transmit Width State Machines ... 250
5.19.9.1 Transmit_Width_Cmd State Machine ... 250
5.19.9.2 Transmit_Width state machine ... 251
5.19.10 Receive Width State Machines ... 257
5.19.10.1 Receive_Width_Cmd State Machine ... 257
5.19.10.2 Receive_Width State Machine ... 258
5.20 Pseudo Random Binary Sequence Testing ... 262

Chapter 6 LP-Serial Protocol

6.1 Introduction ... 265
6.2 Packet Exchange Protocol ... 265
6.3 Traffic types ... 266
6.4 Virtual Channels ... 267
6.4.1 Virtual channel 0 (VC0) ... 267
6.4.2 Virtual Channels 1-8 (VC1-8) ... 267
6.4.3 Virtual Channel Utilization ... 268
6.5 Control Symbols ... 268
6.5.1 Control Symbol Selection ... 268
6.5.2 Control Symbol Delimiting ... 269
6.5.3 Control Symbol Use ... 269

Table of Contents

6.5.3.1 Link Initialization ... 269
6.5.3.2 Buffer Status Maintenance ... 271
6.5.3.3 Embedded Control Symbols ... 271
6.5.3.4 Timing Control Symbols ... 272
6.5.3.4.1 Multicast-Event Control Symbols ... 272
6.5.3.4.2 Loop-Timing Request ... 273
6.5.3.5 Time Synchronization Protocol ... 274
6.5.3.5.1 Setting and Reading a Timestamp Generator ... 275
6.5.3.5.2 Calibrating Transmission Delay ... 278
6.5.3.5.3 Regular Timestamp Generator Re-synchronization ... 280
6.5.3.5.4 Timestamp Generator Synchronization Control Symbol Jitter ... 281
6.5.3.6 MECS Time Synchronization Protocol ... 282
6.5.3.6.1 (S)MECS Master Operation ... 282
6.5.3.6.2 MECS Slave Operation ... 283
6.6 Packets ... 284
6.6.1 Packet Delimiting ... 284
6.6.1.1 Packet Start ... 285
6.6.1.2 Packet Termination ... 285
6.6.2 Acknowledgment Identifier ... 285
6.6.3 Packet Priority and Transaction Request Flows ... 286
6.7 Link Maintenance Protocol ... 288
6.8 Packet Transmission Protocol ... 289
6.9 Flow Control ... 291
6.9.1 Receiver-Controlled Flow Control ... 291
6.9.1.1 Reliable Traffic VC Receivers ... 292
6.9.1.2 Continuous Traffic VC Receivers ... 292
6.9.1.3 Single VC Retry Protocol ... 293
6.9.1.4 Input Retry-Stopped Recovery Process ... 294
6.9.1.5 Output Retry-Stopped Recovery Process ... 295
6.9.2 Transmitter-Controlled Flow Control ... 295
6.9.2.1 Receive Buffer Management ... 297
6.9.2.2 Effective Number of Free Receive Buffers ... 298
6.9.2.3 Speculative Packet Transmission ... 300
6.9.3 Flow Control Mode Negotiation ... 300
6.10 Canceling Packets ... 300
6.11 Transaction and Packet Delivery Ordering Rules ... 302
6.12 Deadlock Avoidance ... 303
6.13 Error Detection and Recovery ... 306
6.13.1 Lost Packet Detection ... 306
6.13.2 Link Behavior Under Error ... 307
6.13.2.1 Recoverable Errors ... 308
6.13.2.2 Idle Sequence Errors ... 308
6.13.2.2.1 IDLE1 Sequence Errors ... 308
6.13.2.2.2 IDLE2 Sequence Errors ... 309
6.13.2.2.3 IDLE3 Sequence Errors ... 309
6.13.2.3 Control Symbol Errors ... 310

Table of Contents

6.13.2.3.1 Link Protocol Violations 310
6.13.2.3.2 Corrupted Control symbols 311
6.13.2.4 Packet Errors........ 311
6.13.2.5 Link Timeout 312
6.13.2.6 Input Error-Stopped Recovery Process 312
6.13.2.7 Output Error-Stopped Recovery Process........ 314
6.14 Power Management 315

Chapter 7 LP-Serial Registers

7.1 Introduction........ 317
7.2 Register Map........ 317
7.3 Reserved Register, Bit and Bit Field Value Behavior 318
7.4 Capability Registers (CARs) 320
7.4.1 Processing Element Features CAR........ 320
7.5 LP-Serial Extended Feature Blocks........ 321
7.5.1 Generic Endpoint Devices 321
7.5.2 Generic Endpoint Devices, Software-assisted Error Recovery Option........ 321
7.5.3 Generic Endpoint Free Devices 321
7.5.4 Generic Endpoint Free Devices, Software-assisted Error Recovery Option.. 322
7.5.5 Register Map - I 322
7.5.6 Register Map - II........ 324
7.6 LP-Serial Command and Status Registers (CSRs) 328
7.6.1 LP-Serial Register Block Header........ 328
7.6.2 Port Link Timeout Control CSR........ 329
7.6.3 Port Response Timeout Control CSR 330
7.6.4 Port General Control CSR 331
7.6.5 Port n Link Maintenance Request CSRs........ 332
7.6.6 Port n Link Maintenance Response CSRs 333
7.6.7 Port n Local ackID CSRs........ 334
7.6.8 Port n Initialization Status CSRs 335
7.6.9 Port n Control 2 CSRs 336
7.6.10 Port n Error and Status CSRs........ 340
7.6.11 Port n Control CSRs 342
7.6.12 Port n Outbound ackID CSRs........ 345
7.6.13 Port n Inbound ackID CSRs........ 346
7.6.14 Port n Power Management CSRs........ 347
7.6.15 Port n Latency Optimization CSRs........ 349
7.6.16 Port n Link Timers Control CSRs 351
7.6.17 Port n Link Timers Control 2 CSRs 353
7.6.18 Port n Link Timers Control 3 CSRs 354
7.7 LP-Serial Lane Extended Features Block........ 355
7.7.1 Register Map........ 355
7.7.2 LP-Serial Lane Command and Status Registers (CSRs) 357
7.7.2.1 LP-Serial Register Block Header........ 357
7.7.2.2 Lane n Status 0 CSRs 358

Table of Contents

7.7.2.3 Lane n Status 1 CSRs ... 360
7.7.2.4 Lane n Status 2 CSRs ... 364
7.7.2.5 Lane n Status 3 CSRs ... 366
7.7.2.6 Implementation Specific CSRs ... 368
7.7.2.6.1 Lane n Status 2 CSR ... 368
7.7.2.6.2 Lane n Status 3 CSR ... 368
7.7.2.6.3 Lane n Status 4 CSR ... 368
7.7.2.6.4 Lane n Status 5 CSR ... 368
7.7.2.6.5 Lane n Status 6 CSR ... 368
7.7.2.6.6 Lane n Status 7 CSR ... 368
7.8 Virtual Channel Extended Features Block ... 369
7.8.1 Register Map ... 369
7.8.2 Virtual Channel Control Block Registers ... 370
7.8.2.1 VC Register Block Header ... 370
7.8.2.2 Port n VC Control and Status Registers ... 371
7.8.2.3 Port n VC0 BW Allocation Registers ... 373
7.8.2.4 Port n VCx BW Allocation Registers ... 375
7.9 Timestamp Generation Extension Block ... 376
7.9.1 Timestamp Generation Extension Block Header ... 378
7.9.2 Timestamp CAR ... 379
7.9.3 Timestamp Generator Status CSR ... 380
7.9.4 MECS Tick Interval CSR ... 381
7.9.5 MECS Next Timestamp MSW CSR ... 382
7.9.6 MECS Next Timestamp LSW CSR ... 383
7.9.7 Timestamp Generator MSW CSR ... 384
7.9.8 Timestamp Generator LSW CSR ... 384
7.9.9 Port n Timestamp 0 MSW CSRs ... 385
7.9.10 Port n Timestamp 0 LSW CSRs ... 385
7.9.11 Port n Timestamp 1 MSW CSRs ... 386
7.9.12 Port n Timestamp 1 LSW CSRs ... 386
7.9.13 Port n Timestamp Generator Synchronization CSRs ... 387
7.9.14 Port n Auto Update Counter CSRs ... 389
7.9.15 Port n Timestamp Synchronization Command CSRs ... 390
7.9.16 Port n Timestamp Synchronization Status CSRs ... 391
7.9.17 Port n Timestamp Offset CSRs ... 392
7.10 Miscellaneous Physical Layer Extension Block ... 393
7.10.1 Miscellaneous Physical Layer Extension Block Header ... 394
7.10.2 Miscellaneous Physical Layer CAR ... 395
7.10.3 Port n Reinit Control CSR ... 396
7.10.4 Port n SAL Control and Status CSR ... 397
7.10.5 Port n SMECS Control CSR ... 398
7.10.6 Port n PRBS Control CSR ... 399
7.10.7 Port n PRBS Lane Control CSR ... 401
7.10.8 Port n PRBS Status 0 CSR ... 402
7.10.9 Port n PRBS Status 1 CSR ... 405
7.10.10 Port n PRBS Locked Time CSR ... 408

Table of Contents

7.10.11 Port n PRBS Seed CSR 409

Chapter 8 Signal Descriptions

8.1 Introduction 411
8.2 Signal Definitions 411
8.3 Serial RapidIO Interface Diagrams 411

Chapter 9 Common Electrical Specifications for less than 6.5 Gbaud LP-Serial Links

9.1 Introduction 413
9.2 References 414
9.3 Abbreviations 415
9.4 Definitions 417
9.4.1 Definition of Amplitude and Swing 420
9.4.2 Transmitter (Near-End) Template 421
9.4.3 Receiver (Far-End) Template 424
9.4.3.1 Level I Receiver Template 424
9.4.3.2 Level II Receiver Template 426
9.4.4 Definition of Skew and Relative Wander 427
9.4.5 Total Wander Mask 428
9.4.6 Relative Wander Mask 429
9.4.7 Random Jitter Mask 429
9.4.8 Defined Test Patterns 429
9.4.9 Reference Model 430
9.5 Common Electrical Specification 430
9.5.1 Introduction 430
9.5.2 Data Patterns 431
9.5.3 Signal Levels 431
9.5.4 Bit Error Ratio 432
9.5.4.1 Level I Bit Error Ratio 432
9.5.4.2 Level II Bit Error Ratio 432
9.5.5 Ground Differences 432
9.5.6 Cross Talk 432
9.5.7 Transmitter Test Load 433
9.5.8 Transmitter Lane-to-Lane Skew 433
9.5.9 Receiver Input Lane-to-Lane Skew 433
9.5.10 Transmitter Short Circuit Current 433
9.5.11 Differential Resistance and Return Loss, Transmitter and Receiver 434
9.5.12 Baud Rate Tolerance 434
9.5.13 Termination and DC Blocking 435
9.6 Pulse Response Channel Modelling 435
9.6.1 Generating a Pulse Response 435
9.6.2 Basic Pulse Response Definitions 436
9.6.3 Transmitter Pulse Definition 437
9.6.4 Receiver Pulse Response 438

Table of Contents

9.6.5 Crosstalk Pulse Response 439
9.6.6 Decision Feedback Equalizer 439
9.6.7 Time Continuous Transverse Filter 440
9.6.7.1 Time Continuous Zero-Pole Equalizer Adaption 440
9.6.8 Time Continuous Zero/Pole 441
9.6.9 Degrees of Freedom 441
9.6.9.1 Receiver Sample Point 441
9.6.9.2 Transmit Emphasis 441
9.7 Jitter Modelling 441
9.7.1 High Frequency Jitter vs. Wander 441
9.7.2 Total Wander vs. Relative Wander 442
9.7.3 Correlated vs. Uncorrelated Jitter 442
9.7.4 Jitter Distributions 443
9.7.4.1 Unbounded and Bounded Gaussian Distribution 443
9.7.4.2 Bounded Gaussian Distribution 443
9.7.4.3 High Probability Jitter 444
9.7.4.4 Total Jitter 444
9.7.4.5 Probability Distribution Function vs. Cumulative Distribution Function .. 444
9.7.4.6 BathTub Curves 445
9.7.4.7 Specification of GJ and HPJ 447
9.7.4.8 Example of Bounded Gaussian 448
9.7.5 Statistical Eye Methodology 448
9.7.5.1 Derivation of Cursors and Calculation of PDF 448
9.7.5.2 Inclusion of Sampling Jitter 451
9.7.5.3 Generation of Statistical Eye 452

Chapter 10 1.25 Gbaud, 2.5 Gbaud, and 3.125 Gbaud LP-Serial Links

10.1 Level I Application Goals 455
10.2 Equalization 456
10.3 Explanatory Note on Level I Transmitter and Receiver Specifications 456
10.4 Level I Electrical Specification 457
10.4.1 Level I Short Run Transmitter Characteristics 457
10.4.1.1 Level I SR Transmitter Test Load 458
10.4.1.2 Level I SR Transmitter Baud Rate 458
10.4.1.3 Level I SR Transmitter Amplitude and Swing 458
10.4.1.4 Level I SR Transmitter Rise and Fall Times 458
10.4.1.5 Level I SR Transmitter Differential Pair Skew 459
10.4.1.6 Level I SR Transmitter Output Resistance and Return Loss 459
10.4.1.7 Level I SR Transmitter Lane-to-Lane Skew 459
10.4.1.8 Level I SR Transmitter Short Circuit Current 459
10.4.1.9 Level I SR Transmitter Template and Jitter 459
10.4.2 Level I Long Run Transmitter Characteristics 460
10.4.2.1 Level I LR Transmitter Test Load 461
10.4.2.2 Level I LR Transmitter Baud Rate 461
10.4.2.3 Level I LR Transmitter Amplitude and Swing 461

Table of Contents

10.4.2.4 Level I LR Transmitter Rise and Fall Times 461
10.4.2.5 Level I LR Transmitter Differential Pair Skew 462
10.4.2.6 Level I LR Transmitter Output Resistance and Return Loss 462
10.4.2.7 Level I LR Transmitter Lane-to-Lane Skew 462
10.4.2.8 Level I LR Transmitter Short Circuit Current 462
10.4.2.9 Level I LR Transmitter Template and Jitter 462
10.4.3 Level I Receiver Specifications 463
10.4.3.1 Level I Receiver Input Baud Rate 464
10.4.3.2 Level I Receiver Reference Input Signals 464
10.4.3.3 Level I Receiver Input Signal Amplitude 464
10.4.3.4 Level I Receiver Absolute Input Voltage 465
10.4.3.5 Level I Receiver Input Common Mode Impedance 465
10.4.3.6 Level I Receiver Input Lane-to-Lane Skew 465
10.4.3.7 Level I Receiver Input Resistance and Return Loss 465
10.4.3.8 Level I Receiver Input Jitter Tolerance 466
10.5 Level I Measurement and Test Requirements 467
10.5.1 Level I Transmitter Measurements 467
10.5.1.1 Level I Eye Template Measurements 467
10.5.1.2 Level I Jitter Test Measurements 468
10.5.1.3 Level I Transmit Jitter Load 468
10.5.2 Level I Receiver Jitter Tolerance 468

Chapter 11 5 Gbaud and 6.25 Gbaud LP-Serial Links

11.1 Level II Application Goals 469
11.1.1 Common to Level II Short run, Medium run and Long run 469
11.1.2 Application Goals for Level II Short Run 470
11.1.3 Application Goals for Level II Medium Run 470
11.1.4 Application Goals for Long Run 470
11.1.5 Explanatory Note on Transmitter and Receiver Specifications 471
11.2 Equalization 471
11.3 Link Compliance Methodology 472
11.3.1 Overview 472
11.3.2 Reference Models 472
11.3.3 Channel Compliance 473
11.3.4 Transmitter Compliance 474
11.3.5 Receiver Compliance 474
11.4 Level II Short Run Interface - General Requirements 475
11.4.1 Jitter and Inter-operability Methodology 475
11.4.1.1 Level II SR Defined Test Patterns 476
11.4.1.2 Level II SR Channel Compliance 476
11.4.1.3 Level II SR Transmitter Inter-operability 477
11.4.1.4 Level II SR Receiver Inter-operability 477
11.4.2 Level II SR Electrical Characteristics 478
11.4.2.1 Level II SR Transmitter Characteristics 478
11.4.2.1.1 Level II SR Transmitter Test Load 480

Table of Contents

11.4.2.1.2 Level II SR Transmitter Baud Rate 480
11.4.2.1.3 Level II SR Transmitter Amplitude and Swing 480
11.4.2.1.4 Level II SR Transmitter Rise and Fall Times 480
11.4.2.1.5 Level II SR Transmitter Differential Pair Skew 480
11.4.2.1.6 Level II SR Transmitter Output Resistance and Return Loss 480
11.4.2.1.7 Level II SR Transmitter Lane-to-Lane Skew 480
11.4.2.1.8 Level II SR Transmitter Template and Jitter 481
11.4.2.2 Level II SR Receiver Characteristics 481
11.4.2.2.1 Level II SR Receiver Input Baud Rate 483
11.4.2.2.2 Level II SR Receiver Reference Input Signals 483
11.4.2.2.3 Level II SR Receiver Input Signal Amplitude 483
11.4.2.2.4 Level II SR Receiver Absolute Input Voltage 483
11.4.2.2.5 Level II SR Receiver Input Common Mode Impedance 484
11.4.2.2.6 Level II SR Receiver Input Lane-to-Lane Skew 484
11.4.2.2.7 Level II SR Receiver Input Resistance and Return Loss 484
11.4.2.2.8 Level II SR Receiver Input Jitter Tolerance 484
11.4.2.3 Level II SR Link and Jitter Budgets 485
11.4.3 Level II SR StatEye.org Template 486
11.5 Level II Long Run Interface General Requirements 488
11.5.1 Long Run Jitter and Inter-operability Methodology 488
11.5.1.1 Level II LR Channel Compliance 488
11.5.1.2 Level II LR Transmitter Inter-operability 490
11.5.1.3 Level II LR Receiver Inter-operability 490
11.5.2 Level II LR Interface Electrical Characteristics 491
11.5.2.1 Level II LR Transmitter Characteristics 491
11.5.2.1.1 Level II LR Transmitter Test Load 493
11.5.2.1.2 Level II LR Transmitter Baud Rate 493
11.5.2.1.3 Level II LR Transmitter Amplitude and Swing 493
11.5.2.1.4 Level II LR Transmitter Rise and Fall Times 493
11.5.2.1.5 Level II LR Transmitter Differential Pair Skew 493
11.5.2.1.6 Level II LR Transmitter Output Resistance and Return Loss 493
11.5.2.1.7 Level II LR Transmitter Lane-to-Lane Skew 494
11.5.2.1.8 Level II LR Transmitter Short Circuit Current 494
11.5.2.1.9 Level II LR Transmitter Template and Jitter 494
11.5.2.2 Level II LR Receiver Characteristics 494
11.5.2.2.1 Level II LR Receiver Input Baud Rate 495
11.5.2.2.2 Level II LR Receiver Reference Input Signals 495
11.5.2.2.3 Level II LR Receiver Input Signal Amplitude 495
11.5.2.2.4 Level II LR Receiver Absolute Input Voltage 496
11.5.2.2.5 Level II LR Receiver Input Common Mode Impedance 496
11.5.2.2.6 Level II LR Receiver Input Lane-to-Lane Skew 496
11.5.2.2.7 Level II LR Receiver Input Resistance and Return Loss 496
11.5.2.2.8 Level II LR Receiver Jitter Tolerance 497
11.5.3 Level II LR Link and Jitter Budgets 497
11.5.4 Level II LR StatEye.org Template 498
11.6 Level II Medium Run Interface General Requirements 500

Table of Contents

11.6.1 Medium Run Jitter and Inter-operability Methodology 500
11.6.1.1 Level II Medium Run Channel Compliance 500
11.6.1.2 Level II MR Transmitter Inter-operability 501
11.6.1.3 Medium Receiver Inter-operability 502
11.6.2 Level II MR Interface Electrical Characteristics 502
11.6.2.1 Level II MR Transmitter Characteristics 502
11.6.2.1.1 Level II MR Transmitter Test Load 504
11.6.2.1.2 Level II MR Transmitter Baud Rate 504
11.6.2.1.3 Level II MR Transmitter Amplitude and Swing 504
11.6.2.1.4 Level II MR Transmitter Rise and Fall Times 504
11.6.2.1.5 Level II MR Transmitter Differential Pair Skew 504
11.6.2.1.6 Level II MR Transmitter Output Resistance and Return Loss 504
11.6.2.1.7 Level II MR Transmitter Lane-to-Lane Skew 505
11.6.2.1.8 Level II MR Transmitter Short Circuit Current 505
11.6.2.1.9 Level II MR Transmitter Template and Jitter 505
11.6.2.2 Level II MR Receiver Characteristics 505
11.6.2.2.1 Level II MR Receiver Input Baud Rate 506
11.6.2.2.2 Level II MR Receiver Reference Input Signals 506
11.6.2.2.3 Level II MR Receiver Input Signal Amplitude 506
11.6.2.2.4 Level II MR Receiver Absolute Input Voltage 507
11.6.2.2.5 Level II MR Receiver Input Common Mode Impedance 507
11.6.2.2.6 Level II MR Receiver Input Lane-to-Lane Skew 507
11.6.2.2.7 Level II MR Receiver Input Resistance and Return Loss 507
11.6.2.2.8 Level II MR Receiver Jitter Tolerance 508
11.6.3 Level II MR Link and Jitter Budgets 508
11.6.4 Level II MR StatEye.org Template 509
11.7 Level II Measurement and Test Requirements 511
11.7.1 High Frequency Transmit Jitter Measurement 511
11.7.1.1 BERT Implementation 511
11.7.1.2 Spectrum Analyzer and Oscilloscope Methodology 513
11.7.1.2.1 Band Limited Unbounded Gaussian Noise 513
11.7.1.2.2 Band Limited 60 Second Total Jitter Measurements 513
11.7.1.2.3 Uncorrelated High Probability Jitter 514
11.7.1.2.4 Total High Probability Jitter 515
11.7.2 Total Transmit Wander Measurement 515
11.7.3 Relative Transmit Wander Measurement 516
11.7.4 Jitter Tolerance 517
11.7.4.1 Jitter Tolerance with Relative Wander Lab Setup 517
11.7.4.1.1 General 518
11.7.4.1.2 Synchronization 518
11.7.4.1.3 Jitter 519
11.7.4.1.4 Amplitude 519
11.7.4.2 Jitter Tolerance with no Relative Wander Lab Setup 520
11.7.4.3 Jitter Tolerance with Defined ISI and no Relative Wander 520
11.7.4.4 Jitter Transfer 521
11.7.4.5 Network Analysis Measurement 522

Table of Contents

11.7.4.6 Eye Mask Measurement Setup 524

Chapter 12 Electrical Specification for 10.3125 and 12.5 Gbaud LP-Serial Links

12.1 References 525
12.2 Level III Application Goals 525
12.2.1 Common to Level III Short run and Long run 525
12.2.2 Application Goals for Level III Short Run 526
12.2.3 Application Goals for Long Run 526
12.3 Equalization 527
12.3.1 Receiver 527
12.3.2 Transmitter 527
12.4 Level III Electrical Specification 527
12.4.1 Level III Short Run 527
12.4.2 Level III Long Run 528
12.4.3 Level III Transmitter Lane-to-Lane Skew 528
12.4.4 Receiver Input Lane-to-Lane Skew 528
12.4.5 Electrical IDLE 528

Annex A Transmission Line Theory and Channel Information (Informative)

A.1 Transmission Lines Theory 529
A.2 Impedance Matching 529
A.3 Impedance Definition Details 530
A.4 Density considerations 532
A.5 Common-Mode Impedance and Return Loss 533
A.6 Crosstalk Considerations 534
A.7 Equation Based Channel Loss by Curve Fit 535

Annex B BER Adjustment Methodology (Informative)

B.1 Extrapolation of Correlated Bounded Gaussian Jitter to low BERs 537
B.1.1 Bathtub Measurements 537
B.2 Confidence Level of Errors Measurement 537
B.3 Eye Mask Adjustment for Sampling Oscilloscopes 538
B.3.1 Theory 539
B.3.2 Usage 541

Annex C Interface Management (Informative)

C.1 Introduction 543
C.2 Packet Retry Mechanism 543
C.2.1 Input port retry recovery state machine 543
C.2.2 Output port retry recovery state machine 545
C.3 Error Recovery 547

Table of Contents

C.3.1 Input port error recovery state machine 547
C.3.2 Output port error recovery state machine 548
C.3.3 Changes in Error Recovery Behavior for CT 552

Annex D Critical Resource Performance Limits (Informative)

D.1 IDLE1 and IDLE2 553
D.2 IDLE3 558

Annex E Manufacturability and Testability (Informative)

Annex F Multiple Port Configuration Example (Informative)

F.1 Introduction........ 567
F.2 System with Different Port Width Capabilities 567

Annex G MECS Time Synchronization (Informative)

G.1 Introduction........ 569
G.2 Detection of Missing MECS........ 569
G.3 MECS and SMECS Redundant Operation 569
G.4 Detection of (S)MECS Source Failure 570

Table of Contents

List of Figures

2-1 Packet Format 39
2-2 Packet Alignment....39
2-3 Error Coverage of First 16 Bits of Packet Header 40
2-4 Unpadded Packet of Length 80 Bytes or Less....41
2-5 Padded Packet of Length 80 Bytes or Less....41
2-6 Unpadded Packet of Length Greater than 80 Bytes....41
2-7 Padded Packet of Length Greater than 80 Bytes 42
2-8 CRC Generation Pipeline....43
3-1 Control Symbol 24 Format 47
3-2 Control Symbol 48 Format 48
3-3 Control Symbol 64 Format 48
3-4 CRC-5 Implementation....67
4-1 Character Notation Example (D25.3) 74
4-2 Code-Group Notation Example (/D25.3/) 75
4-3 Lane Encoding, Serialization, Deserialization, and Decoding Process 77
4-4 Example of a Pseudo-Random Idle Code-Group Generator 93
4-5 Idle Sequence 2 Idle Frame 94
4-6 5-tap Transversal Filter....100
4-7 Example of CS Field CMD, ACK, NACK Handshake 102
4-8 Scrambling Sequence Generator....105
4-9 1x Mode Control Symbol 24 Encoding and Transmission Order 109
4-10 1x Mode Packet Encoding and Transmission Order 109
4-11 1x Typical Data Flow with Control Symbol 24....110
4-12 Typical 4x Data Flow with Control Symbol 24....113
4-13 Baudrate_Discovery state machine (Informational) 117
4-14 Lane_Synchronization State Machine 126
4-15 Lane_Alignment State Machine 128
4-16 1x/2x_Mode_Detect State Machine....130
4-17 1x_Initialization State Machine 131
4-18 1x/Nx_Initialization State Machine for N = 4, 8, 16 135
4-19 Alternate 1x/4x_Initialization State Machine 137
4-20 1x/2x_Initialization State Machine....139
4-21 1x/2x/Nx_Initialization State Machine 142
4-22 1x/Mx/Nx_Initialization State Machine for N > M > 2....145
5-1 64b/67b codeword format....156
5-2 64b/67b Data codeword format 157
5-3 General 64b/67b Control Codeword Format 157
5-4 Skip-Marker Control Codeword Format....159
5-5 Lane-Check Control Codeword 159
5-6 Descrambler Seed Control Codeword Format....160
5-7 Skip Control Codeword Format....160

List of Figures

5-8 Status/Control Control Codeword Format....161
5-9 CSB Control Codeword Format....165
5-10 Logical Layout of CSB Control Codeword....165
5-11 CSE Control Codeword Format....166
5-12 Logical Layout of CSE Control Codeword....166
5-13 CSEB Control Codeword Format....166
5-14 Scrambling Sequence Generator....168
5-15 Example of Calculation for Bit 1 of BIP-23....173
5-16 Lane Encoding, Serialization, Deserialization, and Decoding Process....174
5-17 Link CRC-32 Generation Pipeline....177
5-18 Example of a Pseudo-Random Idle Codeword Generator....185
5-19 1x Typical Data Flow with Control Symbol 64....191
5-20 Typical 4x Data Flow with Control Symbol 64....193
5-21 Baudrate_Discovery state machine (Informational)....199
5-22 Long run Lane_Training state machine (lane k) Part 1 of 3....227
5-23 Long run Lane_Training state machine (lane k) Part 2 of 3....228
5-24 Long run Lane_Training state machine (lane k) Part 3 of 3....229
5-25 Short run Lane_Training state machine for lane k Part 1 of 2....230
5-26 Short run Lane_Training state machine for lane k Part 2 of 2....231
5-27 Lane k Codeword_Lock state machine....233
5-28 Lane k Lane_Synchronization state machine....235
5-29 N-lane Lane_Alignment State Machine (Part 1 of 2)....238
5-30 N-lane Lane_Alignment State Machine (Part 2 of 2)....239
5-31 1x/2x/Nx Port_Initialization State Machine, Part 1 of 3....242
5-32 1x/2x/Nx Port_Initialization state machine, Part 2 of 3....243
5-33 1x/2x/Nx Port_Initialization state machine, Part 3 of 3....244
5-34 Retrain/Xmt_Width_Control state machine....249
5-35 Transmit_Width_Cmd state machine....251
5-36 Transmit Width (XMT_Width) State Machine Part 1 of 4....253
5-37 Transmit Width (XMT_Width) State Machine Part 2 of 4....254
5-38 Transmit Width (XMT_Width) State Machine Part 3 of 4....255
5-39 Transmit Width (XMT_Width) State Machine Part 4 of 4, K, L, N ≥ 4....256
5-40 Receive_Width_Cmd state machine....257
5-41 Receive_Width (RCV_Width) State Machine, Part 1 of 4....259
5-42 Receive_Width (RCV_Width) State Machine, Part 2 of 4....260
5-43 Receive_Width (RCV_Width) State Machine, Part 3 of 4....261
5-44 Receive_Width (RCV_Width) State Machine, Part 4 of 4, K, L, N ≥ 4....262
6-1 Example Transaction with Acknowledgment....266
6-2 Time Synchronization with Synchronous Link Partners....278
6-3 Asymmetry Computation....279
6-4 Single VC Mode Receiver-Controlled Flow Control....294
6-5 Single VC Mode Transmitter-Controlled Flow Control....297
8-1 RapidIO 1x Device to 1x Device Interface Diagram....412
8-2 RapidIO Nx Device to Nx Device Interface Diagram....412
8-3 RapidIO Nx Device to 1x Device Interface Diagram....412
9-1 Definition of Transmitter Amplitude and Swing....420

List of Figures

9-2 Transition Symbol Transmit Eye Mask 422
9-3 Transition and Steady State Symbol Eye Mask 423
9-4 Level I Receiver Input Mask 425
9-5 Receiver Input Mask 426
9-6 Skew and Relative Wander Between in Band Signals 427
9-7 Total Wander of a Signal 428
9-8 Total Wander Mask 428
9-9 Relative Wander Mask 429
9-10 Random Jitter Spectrum 429
9-11 Reference Model 430
9-12 Transmitter and Input Differential Return Loss 434
9-13 Termination Example 435
9-14 Graphical Representation of Receiver Pulse 436
9-15 Transmit Pulse 437
9-16 Transmitter FIR Filter Function 438
9-17 Receiver Pulse Definition 439
9-18 Crosstalk Pulse Definition 439
9-19 Decision Feedback Equalizer 440
9-20 Feed Forward Filter 440
9-21 Generation of Total and Relative Wander 442
9-22 Jitter Probability Density Functions 443
9-23 Example of Total Jitter PDF 445
9-24 Example of Total Jitter CDF 445
9-25 Bathtub Definition 447
9-26 Example of Bounded Gaussian 448
9-27 Statistics of Pulse Response Cursor 449
9-28 Variation of the c0 Sampling Time 450
9-29 Varying the Receiver Sampling Point 451
9-30 Generation of the Data Eye and Bathtub 453
9-31 Statistical Eye 454
10-1 Single Frequency Sinusoidal Jitter Limits 466
11-1 OIF Reference Model 473
11-2 Transmitter Compliance Setup 474
11-3 Receiver Compliance Setup 475
11-4 BERT with Golden PLL 511
11-5 Spectral Measurement Setup 513
11-6 Single Side Band Relative Power Spectrum for Phase Modulated Signal 514
11-7 Transmit Wander Lab Setup 515
11-8 Relative Wander Lab Setup 516
11-9 Jitter Tolerance with Relative Wander Lab Setup 518
11-10 Jitter Tolerance with no Relative Wander 520
11-11 Jitter Tolerance with Defined ISI 521
11-12 Jitter Transfer Lab Setup 522
11-13 S-parameter Port Definitions 522
11-14 Mask Measurement with Golden PLL 524
A-1 Transmission Line as 2-port 531

List of Figures

A-2 Network Terminations 531
A-3 Measurement of Zodd, Zeven 532
A-4 Minimization of Crosstalk at IC Pins 534
A-5 Minimization of Crosstalk At Connector Pins....535
A-6 Minimization of Crosstalk Over Backplane 535
A-7 Equations Based Channel Loss Curves 536
B-1 Example Data Mask....539
B-2 Example Data Mask....540
B-3 Cumulative Distribution Function of Maximum Amplitude 541
C-1 Input Port Retry Recovery State Machine 544
C-2 Output Port Retry Recovery State Machine 545
C-3 Input Port Error Recovery State Machine....547
C-4 Output Port Error Recovery State Machine....549
F-1 Example system with asymmetric port-width capabilities 568

List of Tables

1-1 Baud Rate Class support per IDLE/Control Symbol type 34
2-1 Packet Field Definitions....................37
2-2 Use of VC, PRIO and CRF Fields 38
2-3 Parallel CRC-16 Equations....................42
2-4 Maximum Packet Size 44
3-1 Control Symbol Field Definitions....................47
3-2 Stype0 Control Symbol 24 and Control Symbol 48 Encoding....................49
3-3 Stype0 Control Symbol 64 Encoding 49
3-4 Stype0 Parameter Definitions 50
3-5 Packet-Accepted Control Symbol field usage and values. 52
3-6 Packet-Retry Control Symbol field usage and values. 52
3-7 Packet-Not-Accepted Control Symbol field usage and values. 53
3-8 Cause Field Definition 53
3-9 Timestamp Control Symbol field usage and values. 54
3-10 Status Control Symbol field usage and values....................54
3-11 VC-Status Control Symbol field usage and values....................55
3-12 VCID Definition 55
3-13 Link-Response Control Symbol field usage and values. 56
3-14 Port_status Field Definitions for Control Symbol 24 and Control Symbol 48....................56
3-15 Port_status Field Definitions for Control Symbol 64 57
3-16 Control Symbol 64 Loop-Response Control Symbol field usage and values....................58
3-17 Stype1 Control Symbol 24 and Control Symbol 48 Encoding....................58
3-18 Stype1 Control Symbol 64 Encoding 59
3-19 Start-of-Packet Control Symbol field usage and values. 60
3-20 Stomp Control Symbol field usage and values....................60
3-21 End-of-Packet Control Symbol field usage and values. 60
3-22 Restart-From-Retry Control Symbol field usage and values....................61
3-23 Link-Request Control Symbol field usage and values....................61
3-24 Timing Control Symbol field usage and values....................64
3-25 Parallel CRC-5 Equations....................66
3-26 Parallel CRC-13 Equations....................67
3-27 Parallel CRC-24 Equations....................69
4-1 Data Character Encodings 78
4-2 Special Character Encodings 85
4-3 Special Characters and Columns 86
4-4 Code-Group Corruption Caused by Single Bit Errors 88
4-5 Active Port Width Field Encodings....................97
4-6 Lane Number Field Encodings 97
4-7 Command and Status Field Encodings 98
4-8 CS Field 8/10 Bit Encodings 99
4-9 Scrambler Initialization Values 105
4-10 Dcounter Definition 119

List of Tables

4-11 lane_sync “Flicker” Probability....124
4-12 Structurally Asymmetric Link Tx/Rx Width Behaviors....148
5-1 Control Codeword function encoding....158
5-2 Lane_check field content....159
5-3 Status_control field content....161
5-4 Scrambler Initialization Values....169
5-5 BIP-23 Calculation....172
5-6 Parallel Link CRC-32 Equations....175
5-7 Seed ordered sequence....179
5-8 Status/Control ordered sequence....180
5-9 Skip ordered sequence....181
5-10 Codeword Corruption Caused by Bit Errors....194
5-11 Transmit width port command....201
5-12 Transmit width port command status....201
5-13 Structurally Asymmetric Link Tx/Rx Width Behaviors....204
5-14 Reset value for variable from Status/Control control codewords....223
5-15 Effects of lane_sync or codeword_lock de-assertion....224
6-1 Additional VC Combinations....268
6-2 Control Symbol Support for TSG Master and Slave Devices....275
6-3 Sequence and Format of Control Symbol 24 Timestamp Control Symbols....276
6-4 Sequence and Format of Control Symbol 48 Timestamp Control Symbols....276
6-5 Sequence and Format of Control Symbol 64 Timestamp Control Symbols....277
6-6 VC0 Transaction Request Flow to Priority Mapping....287
6-7 VC0 Transaction Request Flow to Priority and Critical Request Flow Mapping....287
6-8 Flow IDs for VCs....288
7-1 LP-Serial Register Map....318
7-2 Configuration Space Reserved Access Behavior....318
7-3 Bit Settings for Processing Element Features CAR....320
7-4 LP-Serial Register Map - I....323
7-5 LP-Serial Register Map - II....325
7-6 Bit Settings for LP-Serial Register Block Header....328
7-7 Bit Settings for Port Link Timeout Control CSR....329
7-8 Bit Settings for Port Response Timeout Control CSR....330
7-9 Bit Settings for Port General Control CSR, Generic Endpoint Devices....331
7-10 Bit Settings for General Port Control CSR, Generic Endpoint Free Device....331
7-11 Bit Settings for Port n Link Maintenance Request CSRs....332
7-12 Bit Settings for Port n Link Maintenance Response CSRs....333
7-13 Bit Settings for Port n Local ackID Status CSRs....334
7-14 Bit Settings for Port n Initialization Status CSRs....335
7-15 Bit Settings for Port n Control 2 CSRs....336
7-16 Bit Settings for Port n Error and Status CSRs....340
7-17 Bit Settings for Port n Control CSRs....342
7-18 Bit Settings for Port n Outbound ackID Status CSRs....345
7-19 Bit Settings for Port n Inbound ackID CSRs....346
7-20 Bit Settings for Port n Power Management CSRs....347
7-21 Bit Settings for Port n Latency Optimization CSRs....349

List of Tables

7-22 Bit Settings for Port n Link Timers Control CSRs 351
7-23 Bit Settings for Port n Link Timers Control 2 CSRs 353
7-24 Bit Settings for Port n Link Timers Control 3 CSRs 354
7-25 LP-Serial Lane Register Map 355
7-26 Bit Settings for LP-Serial Register Block Header 357
7-27 Bit Settings for Lane n Status 0 CSRs 358
7-28 Bit Settings for Lane n Status 1 CSRs 360
7-29 Bit Settings for Lane n Status 2 CSRs 364
7-30 Bit Settings for Lane n Status 3 CSRs 366
7-31 Virtual Channel Registers 369
7-32 Bit Settings for VC Register Block Header 370
7-33 Port n VC Control and Status Registers......371
7-34 Port n VC0 BW Allocation CSRs......373
7-35 BW Allocation Register Bit Values......374
7-36 Port n VCx BW Allocation CSRs......375
7-37 Timestamp Generation Extension Block 376
7-38 Bit Settings for Timestamp Generation Extension Block Header 378
7-39 Bit Settings for Timestamp CAR......379
7-40 Bit Settings for Timestamp Generator Status CSR......380
7-41 Bit Settings for MECS Tick Interval CSR......381
7-42 Bit Settings for MECS Next Timestamp MSW CSR 382
7-43 Bit Settings for MECS Next Timestamp LSW CSR 383
7-44 Bit Settings for Timestamp Generator MSW CSR......384
7-45 Bit Settings for Timestamp Generator LSW CSR 384
7-46 Bit Settings for Port n Timestamp 0 MSW CSRs......385
7-47 Bit Settings for Port n Timestamp 0 LSW CSRs......385
7-48 Bit Settings for Port n Timestamp 1 MSW CSRs......386
7-49 Bit Settings for Port n Timestamp 0 LSW CSRs......386
7-50 Bit Settings for Port n Timestamp Generator Synchronization CSRs 387
7-51 Bit Settings for Port n Auto Update Counter CSRs......389
7-52 Bit Settings for Port n Timestamp Synchronization Command CSRs 390
7-53 Bit Settings for Port n Timestamp Synchronization Status CSRs 391
7-54 Bit Settings for Port n Timestamp Offset CSRs 392
7-55 Miscellaneous Physical Layer Extension Block......393
7-56 Bit Settings for Miscellaneous Physical Layer Extension Block Header......394
7-57 Bit Settings for Miscellaneous Physical Layer CAR......395
7-58 Bit Settings for Port n Reinit Control CSR......396
7-59 Bit Settings for Port n SAL Control and Status CSR 397
7-60 Bit Settings for Port n SMECS Control CSR 398
7-61 Bit Settings for Port n PRBS Control CSR......399
7-62 Bit Settings for Port n PRBS Lane Control CSR......401
7-63 Bit Settings for Port n PRBS Status 0 CSR 402
7-64 Bit Settings for Port n PRBS Status 1 CSR 405
7-65 Bit Settings for Port n PRBS Locked Time CSR......408
7-66 Bit Settings for Port n PRBS Seed CSR 409
8-1 LP-Serial Signal Description 411

List of Tables

9-1 Abbreviations....415
9-2 General Definitions....417
9-3 Jitter and Wander Definitions....418
9-4 Transmitter Output Jitter Specification....423
9-5 Transmitter Eye Mask Cross Reference....424
9-6 Level I Receiver Jitter Specification....425
9-7 Receiver Eye Mask Cross Reference....426
9-8 Level II Receiver Jitter Specification....427
9-9 Definition of Load Types....432
10-1 Level I SR Transmitter AC Timing Specifications....457
10-2 Level I SR Transmitter Output Jitter Specifications....458
10-3 Level I SR Transmitter Return Loss Parameters....459
10-4 Level I SR Near-End (Tx) Template Intervals....460
10-5 Level I LR Transmitter AC Timing Specifications....460
10-6 Level I LR Transmitter Output Jitter Specifications....461
10-7 Level I LR Transmitter Return Loss Parameters....462
10-8 Level I LR Near-End (Tx) Template Intervals....463
10-9 Level I Receiver Electrical Input Specifications....463
10-10 Level I Receiver Input Jitter Tolerance Specifications....464
10-11 Level I Input Return Loss Parameters....465
10-12 Level I Single Frequency Sinusoidal Jitter Limits Knee Frequencies....466
10-13 Level I Far-End (Rx) Template Intervals....467
11-1 Reference Models....473
11-2 Level II SR Transmitter Output Electrical Specifications....478
11-3 Level II SR Transmitter Output Jitter Specifications....479
11-4 Level II SR Transmitter Return Loss Parameters....480
11-5 Level II SR Near-End (Tx) Template Intervals....481
11-6 Level II SR Receiver Electrical Input Specifications....481
11-7 Level II SR Receiver Input Jitter Tolerance Specifications....482
11-8 Level II SR Input Return Loss Parameters....484
11-9 Level II SR Far-End (Rx) Template Intervals....485
11-10 Level II SR Informative Loss, Skew and Jitter Budget....485
11-11 Level II SR High Frequency Jitter Budget....485
11-12 Level II LR Receiver Equalization Output Eye Mask....489
11-13 Level II LR Transmitter Output Electrical Specifications....492
11-14 Level II LR Transmitter Output Jitter Specifications....492
11-15 Level II LR Transmitter Return Loss Parameters....493
11-16 Level II LR Near-End Template Intervals....494
11-17 Level II LR Receiver Electrical Input Specifications....495
11-18 Level II LR Input Return Loss Parameters....496
11-19 Level II LR Informative Loss, Skew and Jitter Budget....497
11-20 Level II LR High Frequency Jitter Budget....497
11-21 Level II LR Receiver Equalization Output Eye Mask....501
11-22 Level II MR Transmitter Output Electrical Specifications....503
11-23 Level II MR Transmitter Output Jitter Specifications....503
11-24 Level II MR Transmitter Return Loss Parameters....504

List of Tables

11-25 Level II MR Near-End Template Intervals....505
11-26 Level II MR Receiver Electrical Input Specifications....506
11-27 Level II MR Input Return Loss Parameters....507
11-28 Level II MR Informative Loss, Skew and Jitter Budget....508
11-29 Level II MR High Frequency Jitter Budget....508
A-1 Curve fit Coefficients....536
C-1 Input Port Retry Recovery State Machine Transition Table....544
C-2 Output Port Retry Recovery State Machine Transition Table....546
C-3 Input Port Error Recovery State Machine Transition Table....548
C-4 Output Port Error Recovery State Machine Transition Table....549
D-1 Packet Transmission Delay Components....555
D-2 Packet Acknowledgment Delay Components....556
D-3 Packet Delays....556
D-4 Maximum Transmission Distances....557
D-5 IDLE3 Packet Transmission Delay Components....560
D-6 IDLE3 Packet Acknowledgment Delay Components....560
D-7 IDLE3 Packet Delays....561
D-8 IDLE3 Maximum Transmission Distances....562

List of Tables

Chapter 1 Overview

1.1 Introduction

The *RapidIO Part 6: LP-Serial Physical Layer Specification* addresses the Physical Layer requirements for devices utilizing an electrical serial connection medium. This specification defines a full duplex Serial Physical Layer interface (link) between devices. The links are comprised of one or more lanes, each lane being a pair of unidirectional serial signaling paths with one path in each direction. Further, it allows ganging of up to sixteen serial lanes for applications requiring higher link performance. It also defines a protocol for link management and packet transport over a link.

RapidIO systems are comprised of end point processing elements and switch processing elements. The RapidIO interconnect architecture is partitioned into a layered hierarchy of specifications which includes the Logical, Common Transport, and Physical Layers. The Logical Layer specifications define the operations and associated transactions by which end point processing elements communicate with each other. The Common Transport Layer defines how transactions are routed from one end point processing element to another through switch processing elements. The Physical Layer defines how adjacent processing elements electrically connect to each other. RapidIO packets are formed through the combination of bit fields defined in the Logical, Common Transport, and Physical Layer specifications.

The RapidIO LP-Serial specification defines a protocol for packet delivery between serial RapidIO devices including packet and control symbol transmission, flow control, error management, and other device to device functions. A particular device may not implement all of the mode selectable features found in this document. See the appropriate user’s manual or implementation specification for specific implementation details of a device.

With the introduction of the 10.3125 and 12.5 Gbaud speeds and higher it becomes of interest to limit the coding overhead to increase the efficiency, because of this a new encoding scheme (64b/67b) is being introduction in Rev. 3.0.

The LP-Serial Physical Layer Specification has the following properties:

- Embeds the transmission clock with data using an 8b/10b or 64b/67b encoding scheme.
- Supports links with from one lane, up to sixteen ganged lanes where each lane is a pair of unidirectional serial paths with one path in each direction.

- Employs retry and error recovery protocols for link level reliability.
- Supports transmission rates of 1.25, 2.5, 3.125, 5, 6.25, 10.3125 and 12.5 Gbaud (data rates of 1, 2, 2.5, 4, 5, 9.85 and 11.94 Gbps) per lane.
- Supports division of the Physical Layer bandwidth into up to 9 virtual channels with independent flow control.
- Supports Time Synchronization across RapidIO links with several different levels of accuracy.

This specification first defines the individual elements that make up the link protocol such as packets, control symbols, and the serial bit encoding scheme. This is followed by a description of the link protocol. Finally, the control and status registers, signal descriptions, and electrical specifications are specified.

The virtual channel features are optional. This specification defines a single virtual channel mode of operation that is fully compatible with previous RapidIO specifications.

1.2 Contents

Following are the contents of the *RapidIO Part 6: LP-Serial Physical Layer Specification:*

- Chapter 1, "Overview", (this chapter) provides an overview of the specification
- Chapter 2, "Packets", defines how a RapidIO LP-Serial packet is formed by prefixing a 10-bit Physical Layer header to the combined RapidIO Transport and Logical Layer bit fields followed by an appended 16-bit CRC field.
- Chapter 3, "Control Symbols", defines the format of three control symbols (Control Symbol 24, Control Symbol 48, and Control Symbol 64) used for packet acknowledgment, link utility functions, link maintenance, packet delineation and to convey flow control information. They may be transmitted between packets and some may be embedded within a packet.
- Chapter 4, "8b/10b PCS and PMA Layers", describes the Physical Coding Sublayer (PCS) functionality as well as the Physical Media Attachment (PMA) functionality for use with Baud Rate class 1 and 2 devices. The PCS functionality includes 8b/10b encoding scheme for embedding clock with data. It also gives transmission rules for the 1x-Nx interfaces and defines the link initialization sequence for clock synchronization. Among other things, the PMA function is responsible for serializing and de-serializing the 10-bit code-groups to and from the serial bitstream(s).
- Chapter 5, "64b/67b PCS and PMA Layers", describes the Physical Coding Sublayer (PCS) functionality as well as the Physical Media Attachment (PMA) functionality for use with Baud Rate class 3 devices. The PCS functionality includes 64b/67b encoding scheme for embedding clock with data. It also gives transmission rules for the 1x-Nx interfaces and defines the

link initialization sequence for clock synchronization. Among other things, the PMA function is responsible for serializing and de-serializing the 67-bit codewords to and from the serial bitstream(s).

- Chapter 6, "LP-Serial Protocol", describes in detail how packets, control symbols, and the PCS/PMA Layers are used to implement the Physical Layer protocol. This includes topics such as link initialization, link maintenance, error detection and recovery, flow control, bandwidth division, and transaction delivery ordering.
- Chapter 7, "LP-Serial Registers", describes the Physical Layer control and status register set. By accessing these registers a processing element may query the capabilities and status and configure another LP-Serial RapidIO processing element.
- Chapter 8, "Signal Descriptions", contains the signal pin descriptions for a RapidIO LP-Serial port and shows connectivity between processing elements.
- Chapter 9, "Common Electrical Specifications for less than 6.5 Gbaud LP-Serial Links", Chapter 10, "1.25 Gbaud, 2.5 Gbaud, and 3.125 Gbaud LP-Serial Links", Chapter 11, "5 Gbaud and 6.25 Gbaud LP-Serial Links", and Chapter 12, "Electrical Specification for 10.3125 and 12.5 Gbaud LP-Serial Links" describe the electrical specifications for a RapidIO LP-Serial device.
- Annex A, “Transmission Line Theory and Channel Information (Informative)", contains a discussion to aid in applying the AC specifications to a system design.
- Annex B, “BER Adjustment Methodology (Informative)", provides recommendations for measuring link error rates.
- Annex C, "Interface Management (Informative)", contains information pertinent to interface management in a RapidIO system, including error recovery, link initialization, and packet retry state machines.
- Annex D, "Critical Resource Performance Limits (Informative)", contains a discussion on outstanding transactions and their relationship to transmission distance capability.
- Annex E, “Manufacturability and Testability (Informative)", recommends implementing to IEEE standard 1149.6 for improved manufacturing and manufacturing test.
- Annex F, “Multiple Port Configuration Example (Informative)", describes an example of a port configuration scenario.

- Annex F, “Multiple Port Configuration Example (Informative)", describes an example of a port configuration scenario.

- Annex G, “MECS Time Synchronization (Informative)", describes operational and implementation considerations for MECS/SMECS Time Synchronization.

1.3 Terminology

The following terms are used throughout this document:

- To easily relate features to the lane speed the following Baud Rate Classes are defined:
  - Baud Rate Class 1 is used for lanes running at 1.25 Gbaud, 2.5 Gbaud, 3.125 Gbaud or 5 Gbaud.
  - Baud Rate Class 2 is used for lanes running at 6.25 Gbaud.
  - Baud Rate Class 3 is used for lanes running at 10.3125 and 12.5 Gbaud.
- Control Symbol types are based on the IDLE sequences being used. The following Control Symbol types are defined:
  - Control Symbol 24 is used when running IDLE1 and was first defined in the 1.x specifications.
  - Control Symbol 48 is used when running IDLE2 and was first defined in the 2.x specifications. It was created to increase the error protection needed with the introduction of DFE in the receiver and to carry the additional ackIDs needed to support same distance as previously with full bandwidth at 6.25G link speeds.
  - Control Symbol 64 is used when running IDLE3 links. It was defined for Baud Rate Class 3 links to support inclusion in 64b/67b encoded links. Control Symbol 64 further enhance the error protection, widens the ackID space and provides additional functionality for certain control symbol encodings as detailed in Chapter 3.

The relationship between Baud Rate Class and IDLE/Control Symbol types is shown in Table 1-1.

Table 1-1. Baud Rate Class support per IDLE/Control Symbol type

	IDLE1 / Control Symbol 24	**IDLE2 / Control Symbol 48**	**IDLE3 / Control Symbol 64**
Baud Rate Class 1	Supported	Supported	
Baud Rate Class 2		Supported	
Baud Rate Class 3			Supported

For other terminology refer to the Glossary at the back of this document.

1.4 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.
x	This value is a don’t care.

Chapter 2 Packets

2.1 Introduction

This chapter specifies the LP-Serial end to end packet format and the fields that are added by LP-Serial Physical Layer. These packets are fed into and received from the PCS function explained in Chapter 4, "8b/10b PCS and PMA Layers" and Chapter 5, "64b/67b PCS and PMA Layers".

2.2 Packet Field Definitions

This section specifies the bit fields added to a packet by the LP-Serial Physical Layer. These fields are required to implement the flow control, error management, and other specified system functions of the LP-Serial Physical Layer Specification. The fields are specified in Table 2-1.

Table 2-1. Packet Field Definitions

Field	Description
ackID	The acknowledgement identifier (ackID) is the packet identifier for link-level packet acknowledgment (for more information, see Section 6.6.2, "Acknowledgment Identifier"). The length of the ackID value depends on the length of the control symbol being used on the link (for more information on the various control symbol formats, see Section Chapter 3, "Control Symbols"). When the control symbol 24 is being used, the ackID value shall be 5 bits long and shall be left justified in the ackID field (ackID[0-4]) with the right-most bit of the field (ackID[5]) set to 0b0. When the control symbol 48 is being used, the ackID value shall be 6 bits long which fills the ackID field. When the control symbol 64 is being used, the complete ackID value shall be 12 bits long with the least significant 6 bits carried in the ackID field of the packet, and the most significant 6 bits carried in the start-of-packet control symbol.
VC	The VC bit specifies the usage of the PRIO and CRF fields. When VC = 0, the PRIO and CRF fields contain the priority bits for a virtual channel 0 packet. When VC = 1 the PRIO and CRF fields contain the Virtual Channel ID for a VC 1-8 packet. See Table 2-2.
prio	Depending on the value of the VC field, PRIO specifies packet priority or contains the most significant bits of the Virtual Channel ID (VCID). See Table 2-2. See Section 6.6.3, "Packet Priority and Transaction Request Flows" for an explanation of prioritizing packets. See Section 6.4, "Virtual Channels" for an explanation of virtual channels.
CRF	Depending on the value of the VC field, CRF differentiates between virtual channel 0 flows of equal priority or contains the least significant bit of the Virtual Channel ID. If VC=0 and Critical Request Flow is not supported, this bit is reserved. See Table 2-2. See Section 6.6.3, "Packet Priority and Transaction Request Flows" for an explanation of prioritizing packets. See Section 6.4, "Virtual Channels" for an explanation of virtual channels.
CRC	Cyclic Redundancy Code used to detect transmission errors in the packet. See Section 2.4.1, "Packet CRC Operation" for details on the CRC error detection scheme.

Table 2-2 describes the use of the VC, prio, and CRF fields.

Table 2-2. Use of VC, PRIO and CRF Fields

VC	**Description**
Single VC mode:	
VC = 0	when CRF is RSVD = 0, PRIO sets packet priority as follows: 00 - lowest priority 01 - medium priority 10 - high priority 11 - highest priority
VC = 0	when CRF is supported, PRIO||CRF sets packet priority: 00 0 - lowest priority 00 1 - critical flow lowest priority 01 0 - medium priority 01 1 - critical flow medium priority 10 0 - high priority 10 1 - critical flow high priority 11 0 - highest priority 11 1 - critical flow high priority
Multiple VC Mode:	

VC = 0
 	
VC = 1
	VC||PRIO||CRF Channel
	0 XX X - VC0 (PRIO, CRF = Priority, same as single VC mode) *
	
	1 00 0 - VC1 (PRIO, CRF = VCID)
	1 00 1 - VC2
	1 01 0 - VC3
	1 01 1 - VC4
	1 10 0 - VC5
	1 10 1 - VC6
	1 11 0 - VC7
	1 11 1 - VC8
	
	* Note: VC0 is the backwards-compatibility channel

When Fewer than 8 VCs are supported (in addition to VC0)	
VC = 1	VC||PRIO||CRF Channel 1 00 X - VC1 (VC0 + 4 VCs) 1 01 X - VC3 1 10 X - VC5 1 11 X - VC7 1 0X X - VC1 (VC0 + 2VCs) 1 1X X - VC5 1 XX X - VC1 (VC0 + 1VC)

2.3 Packet Format

This section specifies the format of LP-Serial Physical Layer packets. Figure 2-1 shows the format of the LP-Serial Physical Layer packet and how the Physical Layer ackID, VC, CRF, and prio fields are prefixed at the beginning of the packet and a 16-bit CRC field is appended to the end of the packet. An additional CRC may be included within the packet (see Section 2.4.1, "Packet CRC Operation", below).

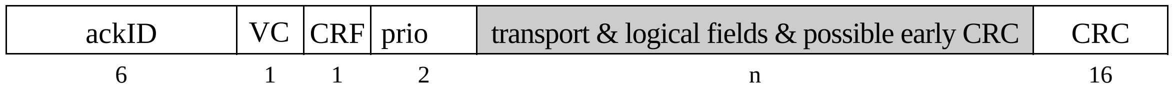


Figure 2-1. Packet Format

The unshaded fields are the fields added by the Physical Layer. The shaded field is the combined Logical and Transport Layer bits and fields that are passed to the Physical Layer (also including the possible early CRC as described in Section 2.4.1).

LP-Serial Physical Layer packets shall have a length that is an integer multiple of 32 bits. This sizing simplifies the design of port logic whose internal data paths are an integer multiple of 32 bits in width. Packets, as defined in the appropriate Logical and Transport Layer Specifications, have a length that is an integer multiple of 16 bits. This is illustrated in Figure 2-2. If the length of a packet defined by the above combination of Specifications is an odd multiple of 16 bits, a 16-bit pad whose value is 0 (0x0000) shall be appended at the end of the packet such that the resulting padded packet is an integer multiple of 32 bits in length.

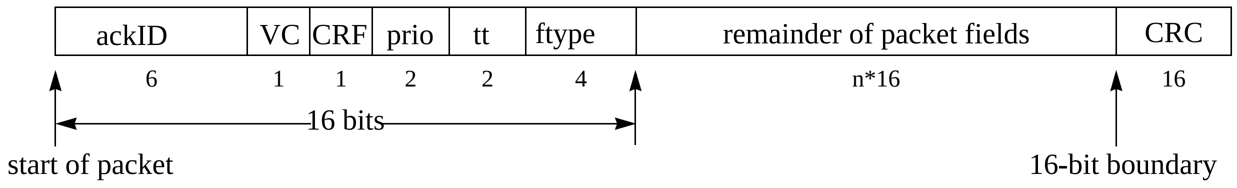


Figure 2-2. Packet Alignment

2.4 Packet Protection

A 16-bit CRC code is added to each packet by the LP-Serial Physical Layer to provide error detection. The code covers the entire packet except for the ackID field, which is considered to be zero for the CRC calculations. Figure 2-3 shows the CRC coverage for the first 16 bits of the packet which contain both the bits covered and not covered by the code.

This structure allows the ackID value to be changed on a link-by-link basis as the packet is transported across the fabric without requiring that the CRC be recomputed for each link. Since ackID values on each link are assigned sequentially for each subsequent transmitted packet, an error in the ackID field is easily detected.

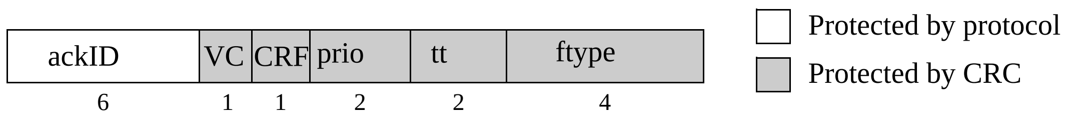


Figure 2-3. Error Coverage of First 16 Bits of Packet Header

2.4.1 Packet CRC Operation

The CRC is appended to a packet in one of two ways. For a packet whose length, exclusive of CRC, is 80 bytes or less, a single CRC is appended at the end of the logical fields. For packets whose length, exclusive of CRC, is greater than 80 bytes, a CRC is added after the first 80 bytes and a second CRC is appended at the end of the Logical Layer fields.

The second CRC value is a continuation of the first. The first CRC is included in the running calculation, meaning that the running CRC value is not 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 two bytes of packet payload for CRC checking purposes. If the CRC appended to the end of the Logical Layer fields does not cause the end of the resulting packet to align to a 32-bit boundary, a two byte pad of all logic 0s is postpended to the packet. The pad of logic 0s allows the CRC check to always be done at the 32-bit boundary. A corrupt pad may or may not cause a CRC error to be detected, depending upon the 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.

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 6.13.2, "Link Behavior Under Error", the *RapidIO Interconnect Specification* does not address the occurrence of such situations nor does it suggest a means by which a processing element would handle such situations. Instead, the mechanism for handling this situation is left to be addressed by the device manufacturers for devices that implement the functionality of early processing of packet data.

Figure 2-4 is an example of an unpadded packet of length less than or equal to 80 bytes.

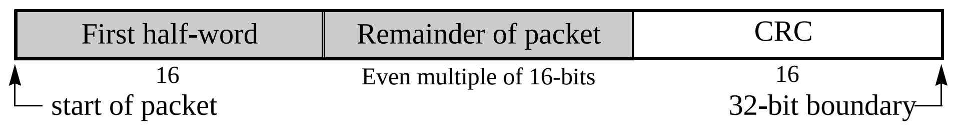


Figure 2-4. Unpadded Packet of Length 80 Bytes or Less

Figure 2-5 is an example of a padded packet of length less than or equal to 80 bytes.

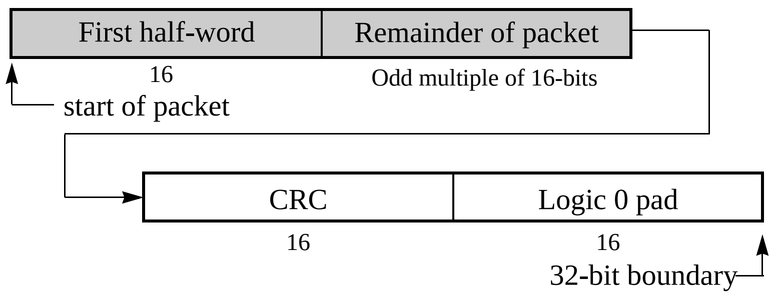


Figure 2-5. Padded Packet of Length 80 Bytes or Less

Figure 2-6 is an example of an unpadded packet of length greater than 80 bytes.

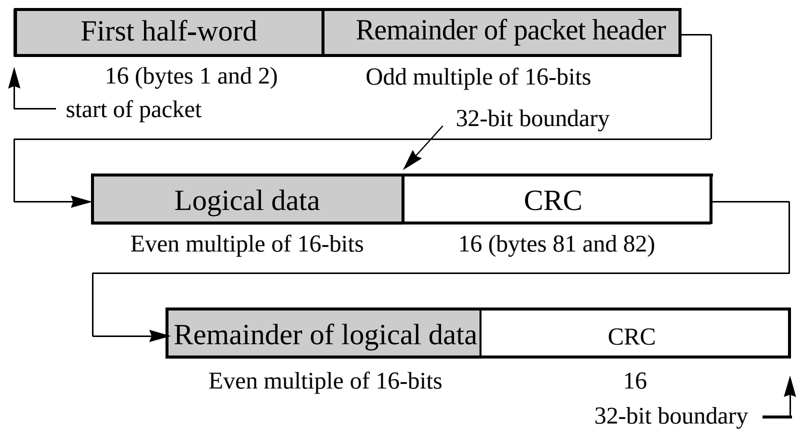


Figure 2-6. Unpadded Packet of Length Greater than 80 Bytes

Figure 2-7 is an example of a padded packet of length greater than 80 bytes.

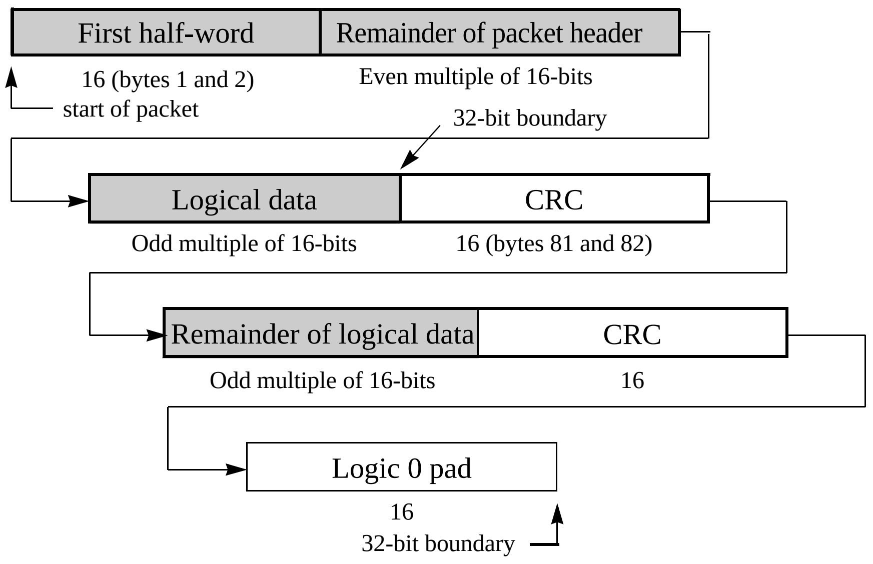


Figure 2-7. Padded Packet of Length Greater than 80 Bytes

2.4.2 CRC-16 Code

The ITU polynomial $x^{16}+x^{12}+x^5+1$ shall be used to generate the 16-bit CRC for packets. The value of the CRC shall be initialized to 0xFFFF (all logic 1s) at the beginning of each packet. For the CRC calculation, the uncovered six bits are treated as logic 0s. As an example, a 16-bit wide parallel calculation is described in the equations in Table 2-3. Equivalent implementations of other widths can be employed.

Table 2-3. Parallel CRC-16 Equations

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

Table 2-3. Parallel CRC-16 Equations (Continued)

Check Bit	e00	e01	e02	e03	e04	e05	e06	e07	e08	e09	e10	e11	e12	e13	e14	e15
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 2-8

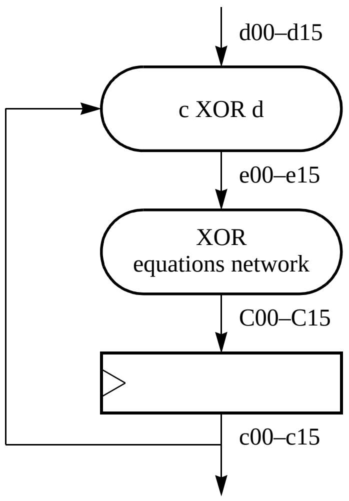


Figure 2-8. CRC Generation Pipeline

2.5 Maximum Packet Size

The RapidIO Specification does not contain an overall specification for the maximum size of a packet that a Logical Layer may pass to the Transport Layer or the Transport Layer may pass to a Physical Layer. Maximum sizes can only be determined by examining the format of each packet type at the Logical Layer and the operation of the Transport and Physical Layers.

The longest packets are those containing an operand address within the destination device, an operand size and a maximum length payload (256 bytes). Currently the largest packet format is the type 5 (write class) format, defined in the I/O Logical specification. The sizes of the components of the maximum packet are shown in more detail in Table 2-4.

Table 2-4. Maximum Packet Size

Field	Size (bytes)	Layer	Notes
Header	2	Logical, Transport, Physical	See Figure 2-2
Source ID	4	Transport	Dev32
Destination ID	4	Transport	Dev32
Trans/wrsize	1	Logical	Type 5 (write class)
srcTID	1	Logical	Type 5 (write class)
Address	8	Logical	Type 5 (write class); includes Extended_address, Address, Wdptr, and Xambs
Payload	256	Logical	Maximum data payload
CRC	4	Physical	Two CRC-16 since packet is greater than 80 bytes
Total	280		

The maximum transmitted packet size permitted by the LP-Serial specification shall be 280 bytes. This includes all packet Logical, Transport, and Physical Layer header information, data payload, and required end-to-end CRC bytes. This does not include packet delimiting control symbols or other necessary Physical Layer control information such as the IDLE3 link CRC-32.

Chapter 3 Control Symbols

3.1 Introduction

This chapter specifies RapidIO LP-Serial Physical Layer control symbols. Control symbols are the message elements used by ports connected by a LP-Serial link to manage all aspects of LP-Serial link operation. They are used for link maintenance, packet delimiting, packet acknowledgment, error reporting, and error recovery.

Three control symbols are defined. The first one is 24 bits long and is referred to as the Control Symbol 24, in previous revisions this was referred to as the “short” control symbol. The second one is 48 bits long and is referred to as the Control Symbol 48, in previous revisions this was referred to as the “long” control symbol. The third one is 64 bits long and is referred to as the Control Symbol 64.

The Control Symbol 24 was the first control symbol defined for LP-Serial links. It was designed for links operating at Baud Rate Class 1 and receivers that do not employ decision feedback equalization (DFE). It provides the functionality needed for the basic link protocol plus some extensions to the link protocol.

The Control Symbol 48 is an extension of the Control Symbol 24. It was designed for links operating at Baud Rate Class 2 and receivers employing DFE. The additional characters are required in part to provide stronger error detection for burst errors that are characteristic of receivers using DFE. The additional characters are also available to provide support for link protocol extensions beyond those supported by the Control Symbol 24.

When use of the Control Symbol 48 is supported by both ends of a LP-Serial link operating at Baud Rate Class 1, it may be used instead of the Control Symbol 24 to provide enhanced control symbol functionality. The selection between Control Symbol 24 and Control Symbol 48 usage for Baud Rate Class 1 links follows the selection of IDLE Sequence as described in Section 4.7.5, "Idle Sequence Selection". When IDLE1 is selected Control Symbol 24 shall be used and when IDLE2 is selected Control Symbol 48 shall be used. Control Symbol 48 shall be used when a LP-Serial link operates at Baud Rate Class 2.

The Control Symbol 64 is an extension of the Control Symbol 24. It was designed for links operating at Baud Rate Class 3. The additional bits are required mainly to provide stronger error detection for burst errors that are characteristic of receivers using DFE. The additional bits are also available to provide support for Baud Rate

Class 3 link protocol enhancements. Control Symbol 64 shall be used when a LP-Serial link operates at Baud Rate Class 3.

LP-Serial control symbols carry at least two independent functions. Each function is assigned one or more control symbol fields for its use. One of the fields assigned to a function specifies the primary function type. The other fields assigned to the function may, depending on the primary function type, further specify the function type, contain information required for the execution of the function, contain “supplemental information” that is not required for the execution of the function and whose value does not affect the behavior of the receiving port, or be unused. Fields that specify the function type or contain data required for the execution of the function are called “functional” fields. Fields that contain “supplemental information” are called “informational” fields. All fields are functional unless specified otherwise.

For forward compatibility, a control symbol function received by a port with an encoding in one or more of the fields assigned to the function that the port does not understand or support shall be handled as follows. If an encoding that the port does not understand or support occurs in a functional field, the control symbol function shall be ignored. If an encoding that the port does not understand or support occurs only in an informational field, the control symbol function shall be executed. In either case, no error shall be reported.

3.2 Control Symbol Field Definitions

This section describes the fields that make up the control symbols.

Table 3-1. Control Symbol Field Definitions

Field	Definition
stype0	Encoding for control symbols that use parameter0 and parameter1. The encodings are defined for Control Symbol 24 and Control Symbol 48 in Table 3-2, and for Control Symbol 64 in Table 3-3.
parameter0	Used in conjunction with stype0 encodings. For the description of parameter0 encodings, see Table 3-2 and Table 3-3.
parameter1	Used in conjunction with stype0 encodings. For the description of parameter1 encodings, see Table 3-2 and Table 3-3.
stype1	Encoding for control symbols that use the cmd field. The encodings are defined for Control Symbol 24 and Control Symbol 48 in Table 3-17, and for Control Symbol 64 in Table 3-18.
cmd	Used for Control Symbol 24 and Control Symbol 48 in conjunction with the stype1 field to define the link maintenance commands. For the cmd field descriptions, see Table 3-17.
reserved	Set to logic 0s on transmission and ignored on reception
alignment	Fixed value of 0b00. These bits are discarded when the control symbol is encoded into codewords and reinserted when it is decoded from codewords.
CRC-5	5-bit code used to detect transmission errors in Control Symbol 24. See Section 3.6, "Control Symbol Protection" for details on the CRC error detection scheme.
CRC-13	13-bit code used to detect transmission errors in Control Symbol 48. See Section 3.6, "Control Symbol Protection" for details on the CRC error detection scheme.
CRC-24	24-bit code used to detect transmission errors in Control Symbol 64. See Section 3.6, "Control Symbol Protection" for details on the CRC error detection scheme.

3.3 Control Symbol Format

This section describes the general formats of the LP-Serial control symbols. All Control Symbol 24 shall have the 24 data bit format shown in Figure 3-1.

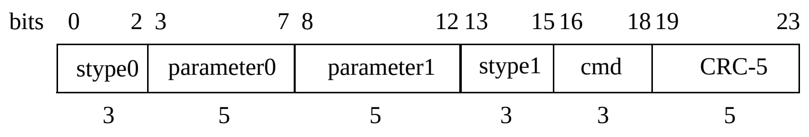


Figure 3-1. Control Symbol 24 Format

Control Symbols 24 can carry two functions, one encoded in the stype0 field and one encoded in the stype1 field. The fields parameter0 and parameter1 are used by the functions encoded in the stype0 field. The cmd field is a modifier for the functions encoded in the stype1 field.

The functions encoded in stype0 are “status” functions that convey some type of status about the port transmitting the control symbol. The functions encoded in stype1 are requests to the receiving port or transmission delimiters.

All Control Symbol 48 shall have the 48 data bit format shown in Figure 3-2.

bits	0 2	3 8	9 14	15 17	18 20	21 34	35 47
	stype0	parameter0	parameter1	stype1	cmd	reserved	CRC-13
	3	6	6	3	3	14	13

Figure 3-2. Control Symbol 48 Format

With one exception, the stype0, parameter0, parameter1, stype1, and cmd fields in the Control Symbol 48 have exactly the same function, encoding, and size as the same named fields in the Control Symbol 24. The exception is that parameter0 and parameter1 are 5-bit fields in the Control Symbol 24 and 6-bit fields in the Control Symbol 48.

All Control Symbol 64 shall have the 64 data bit format shown in Figure 3-3.

0				31	32		63
stype0[0:3]	parameter0	parameter1	stype1[0:1]	alignment	stype1[2:7]	CRC-24	alignment
4	12	12	2	2	6	24	2

Figure 3-3. Control Symbol 64 Format

The stype0, parameter0, parameter1, and stype1 fields in the Control Symbol 64 have a similar function as fields of the same name in the Control Symbol 48. The cmd field is not used for Control Symbol 64; the function it provided for Control Symbol 48 is folded onto the stype1 field. The alignment fields are used to bring the length of the unencoded control symbol up to 64 bits, the number of bits encoded into a 64b/67b codeword. The alignment fields are positioned to allow a Control Symbol 64 to be split across codewords.

Control symbols are defined with the ability to carry at least two functions so that a packet acknowledgment and a packet delimiter can be carried in the same control symbol. Packet acknowledgment and packet delimiter control symbols constitute the vast majority of control symbol traffic on a busy link. Carrying an acknowledgment (or status) and a packet delimiter whenever possible in a single control symbol allows a significant reduction in link overhead traffic and an increase in the link bandwidth available for packet transmission.

A control symbol carrying one function is referred to using the name of the function it carries. A control symbol carrying more than one function may be referred to using the name of any function that it carries. For example, a control symbol with stype0 set to packet-accepted and stype1 set to NOP is referred to a packet-accepted control symbol. A control symbol with stype0 set to packet-accepted and stype1 set to restart-from-retry is referred to as either a packet-accepted control symbol or a restart-from-retry control symbol depending on which name is appropriate for the context.

3.4 Stype0 Control Symbols

The encoding and function of stype0, and the information carried in parameter0 and parameter1 for each stype0 encoding, shall be as specified in Table 3-2 for Control Symbol 24 and Control Symbol 48, and as specified in Table 3-3 for Control Symbol 64.

Table 3-2. Stype0 Control Symbol 24 and Control Symbol 48 Encoding

stype0	Function	Contents of		Reference
		Parameter0	Parameter1	
0b000	packet-accepted	packet_ackID	buf_status	Section 3.4.1
0b001	packet-retry	packet_ackID	buf_status	Section 3.4.2
0b010	packet-not-accepted	arbitrary/ackID_status	cause	Section 3.4.3
0b011	timestamp	timestamp Bits 0–4	timestamp Bits 5–9	Section 3.4.4
0b100	status	ackID_status	buf_status	Section 3.4.5
0b101	VC_status	VCID	buf_status	Section 3.4.6
0b110	link-response	ackID_status	port_status	Section 3.4.7
0b111	implementation-defined *	implementation-defined	implementation-defined	—

* While implementation-defined control symbols are allowed, their use can result in inter-operability problems and is not recommended. There is no registration facility for implementation-defined control symbols. As a result, two implementations may assign different meanings to the same encoding of the Parameter0 and/or Parameter1 fields which could result in undefined and/or inconsistent behavior, data corruption, or system failure. The default state of a processing element after power-up shall disable transmission and processing of implementation specific control symbols.

Table 3-3. Stype0 Control Symbol 64 Encoding

stype0	Function	Contents of		Reference
		Parameter0	Parameter1	
0b0000	packet-accepted	packet_ackID	buf_status	Section 3.4.1
0b0001	packet-retry	packet_ackID	buf_status	Section 3.4.2
0b0010	packet-not-accepted	arbitrary/ackID_status	cause	Section 3.4.3
0b0011	timestamp	See Section 3.4.4		Section 3.4.4
0b0100	status	ackID_status	buf_status	Section 3.4.5
0b0101	VC_status	VCID	buf_status	Section 3.4.6
0b0110	link-response	ackID_status	port_status	Section 3.4.7
0b0111	implementation-defined *	implementation-defined	implementation-defined	—

Table 3-3. Stype0 Control Symbol 64 Encoding

stype0	Function	Contents of		Reference
		Parameter0	Parameter1	
0b1000	reserved	—	—	—
0b1001	reserved	—	—	—
0b1010	reserved	—	—	—
0b1011	loop-response	device_delay	0x000	Section 3.4.8
0b1100	reserved	—	—	—
0b1101	VoQ-backpressure	as defined in Part 12		Part 12
0b1110	reserved	—	—	—
0b1111	reserved	—	—	—

* While implementation-defined control symbols are allowed, their use can result in inter-operability problems and is not recommended. There is no registration facility for implementation-defined control symbols. As a result, two implementations may assign different meanings to the same encoding of the Parameter0 and/or Parameter1 fields that could result in undefined and/or inconsistent behavior, data corruption, or system failure.

The packet-accepted, packet-retry and packet-not-accepted control symbols are collectively referred to as “packet acknowledgment” control symbols.

“Status” is the default stype0 encoding, and is used when a control symbol does not convey another stype0 function.

Table 3-4 defines the parameters valid for stype0 control symbols and that are used for more than one value of stype0.

Table 3-4. Stype0 Parameter Definitions

Parameter	Definition
packet_ackID	The ackID of the packet being acknowledged or the ackID of the packet that caused the retry condition.
ackID_status	The value of the ackID field expected in the next packet the port receives. This value is 1 greater than the ackID of the last packet accepted by the port exclusive of CT mode packets accepted after the port entered an Input-stopped state. For example, a value of 0x01 (Control Symbol 24), 0x01 (Control Symbol 48) or 0x001 (Control Symbol 64) indicates that the ackID of the last packet accepted by the port exclusive of CT mode packets accepted after the port entered an Input-stopped state was 0 and that the port is expecting to receive a packet with an ackID field value of 1.

Table 3-4. Stype0 Parameter Definitions

Parameter	Definition
buf_status	The number of maximum length packet buffers the port has available for packet reception on the specified virtual channel (VC) at the time the control symbol containing the field is generated. The value of the buf_status field in a packet-accepted control symbol is inclusive of the receive buffer consumption of the packet being accepted. The field is used in transmitter controlled flow control to control the rate at which packets are transmitted to prevent loss of packets at the receiver due to a lack of packet buffers. **For Control Symbol 24:** Value 0–29: The encoded value is the number of maximum sized packet buffers the port has available for reception on the specified VC. The value 0, for example, signifies that the port has no packet buffers available for the specified VC (thus is not able to accept any new packets for that VC). Value 30: The value 30 indicates that the port has at least 30 maximum length packet buffers available for reception on the specified VC. Value 31: The port has an undefined number of maximum sized packet buffers available for packet reception, and relies on retry for flow control. **For Control Symbol 48:** Value 0–61: The encoded value is the number of maximum sized packet buffers the port has available for reception on the specified VC. The value 0, for example, signifies that the port has no packet buffers available for the specified VC (thus is not able to accept any new packets for that VC). Value 62: The value 62 indicates that the port has at least 62 maximum length packet buffers available for reception on the specified VC. Value 63: The port has an undefined number of maximum sized packet buffers available for packet reception, and relies on retry for flow control. **For Control Symbol 64:** Value 0–4093: The encoded value is the number of maximum sized packet buffers the port has available for reception on the specified VC. The value 0, for example, signifies that the port has no packet buffers available for the specified VC (thus is not able to accept any new packets for that VC). Value 4094: The value 4094 indicates that the port has at least 4094 maximum length packet buffers available for reception on the specified VC. Value 4095: The port has an undefined number of maximum sized packet buffers available for packet reception, and relies on retry for flow control.
Timestamp	A time value, sent as a loop-response or as part of a timestamp update.

NOTE:

The following sections describes various control symbols. Since control symbols can contain one or more functions, the fields that are applicable to each control symbol function is shown in the tables.

3.4.1 Packet-Accepted Control Symbol

The packet-accepted control symbol indicates that the port sending the control symbol has taken responsibility for sending the packet or packets to its final destination and that resources allocated to the packet or packets by the port receiving the control symbol can be released. This control symbol shall be generated only after the entire packet or packets has been received and found to be free of detectable errors. The packet-accepted control symbol field usage and values are displayed in Table 3-5.

Table 3-5. Packet-Accepted Control Symbol field usage and values.

Format	stype0	Parameter0	Parameter1
Control Symbol 24	0b000	packet_ackID	buf_status
Control Symbol 48	0b000	packet_ackID	buf_status
Control Symbol 64	0b0000	packet_ackID	buf_status

The buf_status value in the control symbol is for the VC of the packet being accepted. Since the VC of the packet is not carried in the control symbol, the port receiving the control symbol must re-associate the ackID in the packet_ackID field with the VC of the accepted packet to determine the VC to which the buf_status applies.

3.4.2 Packet-Retry Control Symbol

A packet-retry control symbol indicates that the port sending the control symbol was not able to accept the packet due to some temporary resource conflict such as insufficient buffering and the packet must be retransmitted. The control symbol field usage and values are displayed in Table 3-6.

Table 3-6. Packet-Retry Control Symbol field usage and values.

Format	stype0	Parameter0	Parameter1
Control Symbol 24	0b001	packet_ackID	buf_status
Control Symbol 48	0b001	packet_ackID	buf_status
Control Symbol 64	0b0001	packet_ackID	buf_status

The packet-retry control symbol shall be used in single VC mode. Packet retry is replaced with error recovery when multiple VCs are active. See Section 6.9, "Flow Control", for more information.

The buf_status shall be for VC0 since retries are only supported for single VC.

3.4.3 Packet-Not-Accepted Control Symbol

The packet-not-accepted control symbol indicates that the port sending the control symbol has either detected an error in the received character stream or, when operating in multiple VC mode, has insufficient buffer resources and as a result may have rejected a packet or control symbol. The control symbol contains an “arbitrary/ackID_status” field and a “cause” field. The control symbol field usage and values are displayed in Table 3-7.

Table 3-7. Packet-Not-Accepted Control Symbol field usage and values.

Format	stype0	Parameter0	Parameter1
Control Symbol 24	0b010	arbitrary/ackID_status	cause
Control Symbol 48	0b010	arbitrary/ackID_status	0b0, cause
Control Symbol 64	0b0010	arbitrary/ackID_status	0b000_0000, cause

The “arbitrary/ackID_status” field may be an arbitrary value, or may be the ackID_Status, indicating the ackID of the next packet expected by the link partner, depending on the capabilities and configuration of the device. For more information, refer to Section 7.6.15, "Port n Latency Optimization CSRs".

The “cause” field is used to provide information about the type of error that was detected for diagnostics and debug use. The content of the cause field is informational only.

The contents of both the arbitrary/ackID_status field and the cause field are informational only, unless bit 9 of the Port n Latency Optimization CSR register is set within both the transmitting and receiving port's configuration space. If both ports support “Error Recovery with ackID in PNA Enabled”, then the contents of the Parameter0 and Parameter1 fields are functional for packet-not-accepted control symbols.

The cause field shall be encoded as specified in Table 3-8 which lists a number of common faults and their encodings. If the port issuing the control symbol is not able to specify the fault, or the fault is not one of those listed in the table, the general error encoding shall be used.

Table 3-8. Cause Field Definition

Cause	Definition
0b00000	Reserved
0b00001	Received a packet with an unexpected ackID
0b00010	Received a control symbol with bad CRC
0b00011	Non-maintenance packet reception is stopped
0b00100	Received a packet with bad CRC
0b00101	Received an invalid character or codeword, or valid but illegal character
0b00110	Packet not accepted due to lack of resources
0b00111	Loss of descrambler sync
0b01000 - 0b11110	Reserved
0b11111	General error

3.4.4 Timestamp Control Symbol

Timestamp control symbols are used to set the timestamp generator value of the link partner with a high degree of accuracy, and for links operating at Baud Rate Class 1 or Baud Rate Class 2 as a response to a loop-timing request (loop-response). Timestamp control symbols contain 10 bits of a time value that is spread across the parameter0 and parameter1 fields. The control symbol field usage and values are displayed in Table 3-9. The use of timestamp control symbols is described in Section 6.5.3.5, "Time Synchronization Protocol".

Table 3-9. Timestamp Control Symbol field usage and values.

Format	stype0	Usage	Parameter0	Parameter1
Control Symbol 24	0b011	Timestamp	Start bit, end bit, most significant 3 bits of timestamp byte value	Least significant 5 bits of timestamp byte value
		Loop-response	Delay[0:4]	Delay[5:9]
Control Symbol 48	0b011	Timestamp	0b0, Start bit, end bit, most significant 3 bits of timestamp byte value	0b0, Least significant 5 bits of timestamp byte value
		Loop-response	Delay[0:5]	Delay[6:11]
Control Symbol 64	0b0011	Timestamp	See Table 6-5	See Table 6-5

3.4.5 Status Control Symbol

The status control symbol indicates receive status information about the port sending the control symbol. The control symbol contains the ackID_status and the buf_status fields. The ackID_status field allows the receiving port to determine if it and the sending port are in sync with respect to the next ackID value the sending port expects to receive. The ackID_status field is informational. The buf_status field indicates to the receiving port the number of maximum length packet buffers the sending port has available for reception on VC0 as defined in Table 3-4.

“Status” is the default stype0 encoding and is used when the control symbol does not convey another stype0 function.

The status control symbol field usage and values are displayed in Table 3-10.

Table 3-10. Status Control Symbol field usage and values.

Format	stype0	Parameter0	Parameter1
Control Symbol 24	0b100	ackID_status	buf_status
Control Symbol 48	0b100	ackID_status	buf_status
Control Symbol 64	0b0100	ackID_status	buf_status

3.4.6 VC-Status Control Symbol

The VC-status control symbol indicates to the receiving port the available buffer space that the sending port has available for packet reception on the virtual channel (VC) specified in the control symbol. The VC-status control symbol is used only for virtual channels 1 through 8 (VC1 through VC8) and may be transmitted only when the specified VC is implemented and enabled. (The status control symbol described in Section 3.4.5, "Status Control Symbol" provides this function for VC0.)

The VCID field specifies the VC to which the control symbol applies. VCID is a 3-bit field that is right justified in the Parameter0 field of the control symbol. The remaining bits of the parameter0 field are reserved, set to 0 on transmission and ignored on reception. The buf_status field indicates to the receiving port the number of maximum length packet buffers the sending port has available for reception on the specified VC as defined in Table 3-4.

The VC-status control symbol may be transmitted at any time and should be transmitted whenever the number of maximum length packet buffers available for reception on a VC has changed and has not been otherwise communicated to the connected port.

The VC-status control symbol field usage and values are displayed in Table 3-11.

Table 3-11. VC-Status Control Symbol field usage and values.

Format	stype0	Parameter0	Parameter1
Control Symbol 24	0b101	0b00, VCID	buf_status
Control Symbol 48	0b101	0b000, VCID	buf_status
Control Symbol 64	0b0101	0b0_0000_0000, VCID	buf_status

The encoding of the VCID field is specified in Table 3-12. The VCID corresponds to the VCID in the Physical Layer format as described in Chapter 2, "Packets".

Table 3-12. VCID Definition

8 Optional VCs Active	
VCID	**Definition**
0b000	VC1
0b001	VC2
0b010	VC3
0b011	VC4
0b100	VC5
0b101	VC6
0b110	VC7
0b111	VC8

4 Optional VCs Active	
VCID	**Definition**
0b00x	VC1
0b01x	VC3
0b10x	VC5
0b11x	VC7

2 Optional VCs Active	
VCID	**Definition**
0b0xx	VC1
0b1xx	VC5

1 Optional VC Active	
VCID	**Definition**
0bxxx	VC1

Active VCs are in addition to VC0

Formats for 4, 2, and 1 active VCs are shown in the three right hand columns of the table. When using fewer than 8 VCs, bits in the VCID are ignored starting from the LSB, consistent with the bit usage in the packet format. For example, with one optional VC active, all bit patterns in the VCID are interpreted as pertaining to VC1.

3.4.7 Link-Response Control Symbol

The link-response control symbol is used by a port to respond to a link-request control symbol (Section 3.5.5) as described in Section 6.7, "Link Maintenance Protocol". The status reported in the port_status field shall be the status of the port at the time the associated port-status link-request control symbol was received. The port_status field shall be treated as “information only” when a link-response control symbol is received. For backwards compatibility with 1.x revisions of this specification, when operating with lane speeds of less than 3.5 Gbaud, the port_status field shall only use one of the following values: 0b00010, 0b00100, 0b00101 or 0b10000 even if other values are defined in the specification. The link-response control symbol field usage and values are displayed in Table 3-13.

Table 3-13. Link-Response Control Symbol field usage and values.

Format	stype0	Parameter0	Parameter1
Control Symbol 24	0b110	ackID_status	port_status
Control Symbol 48	0b110	ackID_status	0b0, port_status
Control Symbol 64	0b0110	ackID_status	port_status

For Control Symbol 24 and Control Symbol 48, the encoding of the link-response control symbol port_status field shall be as defined in Table 3-14.

Table 3-14. Port_status Field Definitions for Control Symbol 24 and Control Symbol 48

Port_status	Status	Description
0b00000 - 0b00001	—	Reserved
0b00010	Error	The port has encountered an unrecoverable error and is unable to accept packets.
0b00011	—	Reserved
0b00100	Retry-stopped	The port has retried a packet and is waiting in the input retry-stopped state to be restarted.
0b00101	Error-stopped	The port has encountered a transmission error and is waiting in the input error-stopped state to be restarted.
0b00110 - 0b01111	—	Reserved
0b10000	OK	The port is accepting packets
0b10001 - 0b11111	—	Reserved

The Control Symbol 64 encoding of the link-response control symbol port_status field shall be as defined in Table 3-15.

Table 3-15. Port_status Field Definitions for Control Symbol 64

Port_status bit number	Description
0	Reserved
1–2	Input Port Status 0b00 - No input error condition exists 0b01 - Port n Error and Status CSR “Input retry-stopped” status bit is asserted 0b10 - Port n Error and Status CSR “Input Error-Stopped” status bit is asserted 0b11 - Implementation specific Input Port Fatal Error condition The values are encoded in increasing order of priority. 0b00 is the lowest priority. When multiple conditions exist simultaneously the highest priority condition shall be encoded.
3	Input Port Enabled This bit shall be set if all of the following conditions are true, otherwise this bit shall be cleared: - The Port n Control CSR Input Port Enabled bit is set. - All implementation specific bits allow Physical Layer packet acceptance.
4	Reserved
5–6	Output Port Status 0b00 - No output error condition exists 0b01 - Port n Error and Status CSR “Output retry-stopped” bit is asserted 0b10 - Port n Error and Status CSR “Output Error-Stopped” status bit is asserted 0b11 - Output Port Fatal Error condition The values are encoded in increasing order of priority. 0b00 is the lowest priority. When multiple conditions exist simultaneously the highest priority condition shall be encoded.
7	Output Port Enabled This bit shall be set if both of the following conditions are true, otherwise this bit shall be cleared: - The Port n Control CSR Output Port Enabled bit is set. - All implementation specific bits allow Physical Layer packet acceptance.
8	Port-Write Pending The port has encountered a condition which required it to initiate a Maintenance Port-write operation.
9–11	Reserved

3.4.8 Loop-Response Control Symbol

The loop-response control symbol is used by ports operating at Control Symbol 64 to respond to a Loop-timing Request control symbol. The loop-response control symbol carries a single 12-bit value, Delay, which represents the number of nanoseconds between the time the loop-timing request was received by the link partner, and the time the loop-response was generated. A Delay value of all 1s (0xFFF) indicates that the amount of delay exceeded 4094 nanoseconds. For more information, refer to Section 6.5.3.5, "Time Synchronization Protocol". The loop-response control symbol field usage and values are displayed in Table 3-13.

Table 3-16. Control Symbol 64 Loop-Response Control Symbol field usage and values.

Format	stype0	Parameter0	Parameter1
Control Symbol 64	0b1011	Delay bits 0–11	Reserved

3.5 Stype1 Control Symbols

The encoding of stype1 and the function of the cmd field are defined in Table 3-17 for Control Symbol 24 and Control Symbol 48, and the encoding of stype1 is defined in Table 3-18 for Control Symbol 64.

Table 3-17. Stype1 Control Symbol 24 and Control Symbol 48 Encoding

stype1 (3 bits)	stype1 Function	cmd	cmd Function	Packet Delimiter	Reference
0b000	Start-of-packet	0b000	Start-of-packet	Yes	Section 3.5.1
		0b001–0b111	Reserved	No	
0b001	Stomp	0b000	Stomp	Yes	Section 3.5.2
		0b001–0b111	Reserved	No	
0b010	End-of-packet	0b000	End-of-packet	Yes	Section 3.5.3
		0b001–0b111	Reserved	No	
0b011	Restart-from-retry	0b000	Restart-from-retry	*	Section 3.5.4
		0b001–0b111	Reserved	No	
0b100	Link-request	0b000–0b001	Reserved	No	-
		0b010	Reset-port	*	Section 3.5.6
		0b011	Reset-device	*	Section 3.5.5.1
		0b100	Port-status	*	Section 3.5.5.3
		0b101–0b111	Reserved	No	-
0b101	Timing	0b000	Multicast-event	No	Section 3.5.6.1
		0b001	Secondary Multicast-event		Section 3.5.6.2
		0b010	Reserved		
		0b011	Loop-Timing Request		Section 3.5.6.3
		0b100–0b111	Reserved		
0b110	Reserved	0b000–0b111	Reserved	No	-
0b111	NOP (Ignore) **	0b000	NOP (Ignore) **	No	-
		0b001–0b111	Reserved	No	

Note: * denotes that restart-from-retry and link-request control symbols may only be packet delimiters if a packet is in progress.

Note: ** NOP (Ignore) is not defined as a control symbol, but is the default value when the control symbol does not convey another stype1 function.

Table 3-18. Stype1 Control Symbol 64 Encoding

stype1 (8 bits)	Function	Packet Delimiter	Reference
0x00–0x07	Reserved		-
0x08	Stomp	yes	Section 3.5.2
0x09–0x0F	Reserved		-
0x10	End-of-packet-unpadded	yes	Section 3.5.3
0x11	End-of-packet-padded	yes	Section 3.5.3
0x12–0x17	Reserved		-
0x18	Restart-from-retry	*	Section 3.5.4
0x19–0x21	Reserved		-
0x22	Link-request/Reset-port	*	Section 3.5.6
0x23	Link-request/Reset-device	*	Section 3.5.5.1
0x24	Link-request/Port-status	*	Section 3.5.5.3
0x25–0x27	Reserved		-
0x28	Multicast-event	No	Section 3.5.6.1
0x29	Secondary Multicast-event	No	Section 3.5.6.2
x2A	Reserved		-
0x2B	Loop-timing-request	No	Section 3.5.6.3
0x2C–0x37	Reserved		-
0x38	NOP (Ignore) **	No	-
0x39–0x7F	Reserved		-
0b10, ackID[0:5]	Start-of-packet-unpadded	yes	Section 3.5.1
0b11, ackID[0:5]	Start-of-packet-padded	yes	Section 3.5.1

Note: * denotes that restart-from-retry and link-request control symbols may only be packet delimiters if a packet is in progress.

Note: ** NOP (Ignore) is not defined as a control symbol, but is the default value when the control symbol does not convey another stype1 function.

NOTE:

The following sections describe various control symbols. Since control symbols can contain one or more functions, the fields that are applicable to each control symbol function are shown in the respective tables.

3.5.1 Start-of-Packet Control Symbol

The start-of-packet control symbol is used to delimit the beginning of a packet. The control symbol field usage and values are displayed in Table 3-19.

Table 3-19. Start-of-Packet Control Symbol field usage and values.

Format	stype1	cmd	Function
Control Symbol 24 Control Symbol 48	0b000	0b000	Start-of-packet
Control Symbol 64	0b10, ackID[0:5]	N/A	Start-of-packet-unpadded
	0b11, ackID[0:5]	N/A	Start-of-packet-padded

The Control Symbol 64 start-of-packet control symbol has two variants – start-of-packet-unpadded and start-of-packet-padded – to indicate if a previous packet has padding appended to achieve a total length that is a multiple of 8 bytes. The start-of-packet-unpadded control symbol shall be used for cases where the start-of-packet does not terminate a previous packet or where the start-of-packet terminates a packet that was not padded. The start-of-packet-padded control symbol shall be used when the start-of-packet terminates a packet that was padded to multiple of 8 bytes. It is needed to differentiate between padded and non-padded packets so devices like switches that do not completely decode the packet can separate link overhead from the packet.

For Control Symbol 64, the stype1[2:7] bits contain the most significant 6 bits of the packet ackID.

3.5.2 Stomp Control Symbol

The stomp control symbol is used to cancel a partially transmitted packet. The protocol for packet cancellation is specified in Section 6.10, "Canceling Packets". The stomp control symbol field usage and values are displayed in Table 3-20.

Table 3-20. Stomp Control Symbol field usage and values.

Format	stype1	cmd
Control Symbol 24 Control Symbol 48	0b001	0b000
Control Symbol 64	0x08	N/A

3.5.3 End-of-Packet Control Symbol

The end-of-packet control symbol is used to delimit the end of a packet. The control symbol field usage and values are displayed in Table 3-21.

Table 3-21. End-of-Packet Control Symbol field usage and values.

Format	stype1	cmd	Function
Control Symbol 24 Control Symbol 48	0b010	0b000	End-of-packet
Control Symbol 64	0x10	N/A	End-of-packet-unpadded
	0x11	N/A	End-of-packet-padded

The Control Symbol 64 end-of-packet control symbol has two variants – end-of-packet-unpadded and end-of-packet-padded – to indicate if a previous packet has padding appended to achieve a total length that is a multiple of 8 bytes. The end-of-packet-unpadded control symbol shall be used when the end-of-packet terminates a packet that was not padded. The end-of-packet-padded control symbol shall be used when the end-of-packet terminates a packet that was padded to multiple of 8 bytes. It is needed to differentiate between padded and non-padded packets so devices like switches that do not completely decode the packet can separate link overhead from the packet.

3.5.4 Restart-From-Retry Control Symbol

This control symbol is used to mark the beginning of packet retransmission, so that the receiver knows when to start accepting packets after the receiver has requested a packet to be retried. The restart-from-retry control symbol cancels a current packet and may also be transmitted on an idle link.

The control symbol field usage and values are displayed in Table 3-22.

Table 3-22. Restart-From-Retry Control Symbol field usage and values.

Format	stype1	cmd
Control Symbol 24 Control Symbol 48	0b011	0b000
Control Symbol 64	0x18	N/A

3.5.5 Link-Request Control Symbol

A link-request control symbol is used by a port to either issue a command to the connected port or request its input port status. A link-request control symbol always cancels a packet whose transmission is in progress and can also be sent between packets. Under error conditions, a link-request/port-status control symbol acts as a link-request/restart-from-error control symbol as described in Section 6.7, "Link Maintenance Protocol".

The control symbol field usage and values are displayed in Table 3-23.

Table 3-23. Link-Request Control Symbol field usage and values.

Format	stype1	cmd	Function	Reference
Control Symbol 24 Control Symbol 48	0b100	0b000–0b001	Reserved	
	0b100	0b010	Reset-port	Section 3.5.5.1
	0b100	0b011	Reset-device	Section 3.5.5.2
	0b100	0b100	Port-status	Section 3.5.5.3
	0b100	0b101–0b111	Reserved	

Table 3-23. Link-Request Control Symbol field usage and values.

Format	stype1	cmd	Function	Reference
Control Symbol 64	0x22	N/A	Reset-port	Section 3.5.5.1
	0x23	N/A	Reset-device	Section 3.5.5.2
	0x24	N/A	Port-status	Section 3.5.5.3

3.5.5.1 Reset-port Command

A reset-port command is intended to allow packet exchange to resume after an unrecoverable link error condition has been detected and system software has handled this condition. Examples of the use of reset-port are link recovery after a field replaceable unit has been inserted, and when one link partner has failed and/or has been reset but not the other.

Scenarios that require a reset-port command for recovery also require packet discard to prevent packets which are undeliverable due to the unrecoverable link error condition from creating a cascade congestion failure of the entire system. Packet discard mechanisms that are part of the *RapidIO Part 8: Error Management/Hot Swap Extensions Specification* may be activated by the unrecoverable link error condition. Implementation specific packet discard mechanisms may also be activated by the unrecoverable link error condition. System recovery from packet discard is vendor specific, and outside the scope of this specification.

A device that receives a reset-port command shall perform the following:

- Disable transmission of implementation specific control symbols and, for links operating with IDLE3, implementation specific control codewords.
- Reset all ackID tracking logic for packets received, transmitted, and unacknowledged to a state consistent with a power-up reset.
- Clear all input-error, output-error, input-retry, output-retry, and port error states.
- Clear the tracking of link-request/port-status control symbol requests received or transmitted.
- Reset the port’s initialization state machine.
- Deactivate the packet discard mechanisms.
- Update the status of register values based on the above changes.
- Retain the values of registers that are not affected by the above changes.

The reset-port command shall not generate a link-response control symbol.

The timing relationship between deactivation of packet discard mechanisms and the arrival of packets may not be deterministic in all systems. For this reason, no assumptions shall be made about the effect of a reset-port command on packet storage for transmission or reception. The effect of a reset-port command on packet storage is implementation specific behavior and outside the scope of this specification.

Note that transmission and reception of a reset-port request may trigger additional functionality defined in *Part 8: Error Management/Hot Swap Extensions Specification*.

After a port transmits a reset-port request, if the port's initialization state machine(Section 5.19.7) transitions to the SILENT state within one link-response timeout period, the port shall behave as if it has received a reset-port request. The timeout period shall be tracked only for the most recently transmitted reset-port command.

Due to the undefined reliability of system designs it is necessary to put a safety lockout on the reset-port function of the link-request control symbol. A port receiving a reset-port command in a link-request control symbol shall not perform the reset-port function unless it has received four reset-port commands in a row without any other intervening packets or control symbols, except status control symbols. Such a sequence is known as a reset-port request. Reset-port requests are intended to prevent spurious reset-port commands from inadvertently resetting a port.

When issuing a reset with four consecutive reset-port commands, care must be taken to account for all effects associated with the reset event. For more information, see the *RapidIO Part 8: Error Management/Hot Swap Extensions Specification*.

3.5.5.2 Reset-Device Command

The reset-device command causes the receiving device to go through its reset or power-up sequence. All state machines and the configuration registers reset to the original power-up states. Note that the device power-up state shall disable transmission of implementation specific control symbols and, for links operating with IDLE3, implementation specific control codewords. The reset-device command does not generate a link-response control symbol.

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

When issuing a reset with four consecutive reset-device commands, care must be taken to account for all effects associated with the reset event. For more information, see the *RapidIO Part 8: Error Management/Hot Swap Extensions Specification*.

3.5.5.3 Port-status[1] Command

The port-status command requests the receiving port to return a link-response containing the ackID value it expects to next receive 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 port-status command. The implementation of flushing the output port is device specific and may result in either discarding the contents of the receive buffers or sending the control symbols on the link. The receiver then responds with a link-response control symbol.

3.5.6 Timing Control Symbols

Timing control symbols are related to communication of events and time within a system. Unlike other control symbols, timing control symbols can trigger activity on other links of a device.

The timing control symbol field usage and values are displayed in Table 3-24.

Table 3-24. Timing Control Symbol field usage and values.

Format	stype1	cmd	Function	Reference
Control Symbol 24 Control Symbol 48	0b101	0b000	Multicast-event	Section 3.5.6.1
	0b101	0b001	Secondary Multicast-event	Section 3.5.6.2
	0b101	0b010	Reserved	-
	0b101	0b011	Loop-Timing Request	Section 3.5.6.3
	0b101	0b100–0b111	Reserved	-
Control Symbol 64	0x28	N/A	Multicast-event	Section 3.5.6.1
	0x29	N/A	Secondary Multicast-event	Section 3.5.6.2
	0x2B	N/A	Loop-Timing Request	Section 3.5.6.3

3.5.6.1 Multicast-Event Control Symbol

The multicast-event control symbol allows the occurrence of a user-defined system event to be multicast throughout a system. The multicast-event control symbol differs from other control symbols in that it carries information not related to the link carrying the control symbol. For more information on Multicast-Events, see Section 6.5.3.4.1, "Multicast-Event Control Symbols".

3.5.6.2 Secondary Multicast-Event Control Symbol

Secondary Multicast-Event Control Symbol support is optional. The secondary multicast-event control symbol allows two discrete sources of multicast events to

[1]Note that Port-Status was known as Input-Status in this specification for revisions prior to 3.0.

exist within a system. The secondary multicast-event control symbol differs from other control symbols in that it carries information not related to the link carrying the control symbol. For more information on Multicast-Events, see Section 6.5.3.4.1, "Multicast-Event Control Symbols".

3.5.6.3 Loop-Timing Control Symbol

The loop-timing control symbol requests the receiver to send a loop-response control symbol in order to determine the transmission delay from the transmitting link partner to the receiving link partner. For information on the use of the loop-timing control symbol, refer to Section 6.5.3.5, "Time Synchronization Protocol".

A processing element shall support transmitting a loop-timing request when the Timestamp Master Supported bit of the Timestamp CAR (Section 7.9.2) is 1. A processing element shall support receiving a loop-timing request when the Timestamp Slave Supported bit of the Timestamp CAR is 1.

3.6 Control Symbol Protection

Control symbol error detection is provided by a cyclic redundancy check (CRC) code.

A 5-bit CRC is used for the Control Symbol 24. It provides detection of a single burst error of 5 bits or less in the 24 data bits of the 8b/10b decoded control symbol. A single 5 bit burst error is the longest burst error that can be caused by a single bit transmission error at the 8b/10b code-group level.

A 13-bit CRC is used for the Control Symbol 48. It provides detection of any set of errors in the 48 data bits of the 8b/10b decoded control symbol that can be caused by a burst error on one lane of 11 bits or less at the 8b/10b code-group level. An 11-bit error at the code-group level can corrupt at most two code-groups.

A 24-bit CRC is used for the Control Symbol 64. It provides detection of a single error burst of up to 24 bits and any odd number of bit errors. It can also detect up to 7 single bit errors across the control symbol. Further protection is provided at the codeword level as described in Chapter 5, "64b/67b PCS and PMA Layers".

3.6.1 CRC-5 Code

The ITU polynomial $x^5+x^4+x^2+1$ shall be used to generate the 5-bit CRC for Control Symbol 24.

The 5-bit CRC shall be computed over 20 bits comprised of control symbol bits 0 through 18 plus a 20th bit that is appended after bit 18 of the control symbol. The added bit shall be set to logic 0 (0b0). The 20th bit is added in order to provide maximum implementation flexibility for all types of designs. The CRC shall be computed beginning with control symbol bit 0. Before the CRC is computed, the CRC shall be set to all 1s (0b11111).

The CRC check bits c[0:4] occupy Control Symbol 24 bits [19:23] respectively.

The 5-bit CRC shall be generated by each transmitter and verified by each receiver using the Control Symbol 24.

3.6.2 CRC-5 Parallel Code Generation

Since it is often more efficient to implement a parallel CRC algorithm rather than a serial, examples of the equations for a complete, 19-bit single-stage parallel implementation are shown in shown in Table 3-25. Since only a single stage is used, the effect of both setting the initial CRC to all 1s (0b11111) and a 20th bit set to logic 0 (0b0) have been included in the equations.

In Table 3-25, an “x” means that the data input should be an input to the Exclusive-OR necessary to compute that particular bit of the CRC. A “!x”, means that bit 18 being applied to the CRC circuit must be inverted.

Table 3-25. Parallel CRC-5 Equations

Control Symbol	**CRC Checksum Bits**				
Data for CRC	**C0**	**C1**	**C2**	**C3**	**C4**
D18	x	!x	!x	!x	x
D17		x		x	x
D16	x		x	x	
D15	x	x			x
D14			x	x	x
D13		x	x	x	
D12	x	x	x		
D11		x	x		x
D10	x	x		x	
D9					x
D8				x	
D7			x		
D6		x			
D5	x				
D4	x		x		x
D3	x	x	x	x	x
D2		x		x	x
D1	x		x	x	
D0	x	x			x

Figure 3-4 shows the 19-bits that the CRC covers and how they should be applied to the circuit. As seen in Figure 3-4, bits are labeled with 0 on the left and 18 on the right. Bit 0, from the stype0 field, would apply to D0 in Table 3-25 and bit 18, from the cmd field, would apply to D18 in Table 3-25. Once completed, the 5-bit CRC is appended to the control symbol.

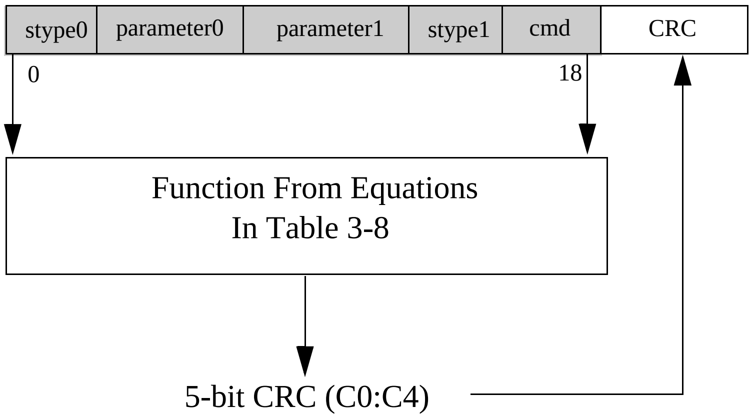


Figure 3-4. CRC-5 Implementation

3.6.3 CRC-13 Code

The polynomial $x^{13}+x^{10}+x^{8}+x^{5}+x^{2}+1$ shall be used to generate the 13-bit CRC for Control Symbol 48.

The 13-bit CRC shall be computed over control symbol bits 0 through 34 beginning with control symbol bit 0. Before the 13-bit CRC is computed, the CRC shall be set to all 0s (0b0_0000_0000_0000).

The CRC check bits c[0:12] shall occupy Control Symbol 48 bits [35:47] respectively.

The 13-bit CRC shall be generated by each transmitter and verified by each receiver using the Control Symbol 48.

3.6.4 CRC-13 Parallel Code Generation

For the CRC-13 parallel code generation, the equations are shown in Table 3-26, using rules as for the CRC-5 parallel generation.

Table 3-26. Parallel CRC-13 Equations

Control Symbol	**CRC Checksum Bits**												
Data For CRC	**C0**	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**	**C8**	**C9**	**C 10**	**C 11**	**C 12**
D34			x		x			x			x		x

Control Symbol	CRC Checksum Bits												
D33		x		x			x			x		x	
D32	x		x			x			x		x		
D31		x	x							x	x		x
D30	x	x							x	x		x	
D29	x		x		x				x				x
D28		x	x	x	x						x	x	x
D27	x	x	x	x						x	x	x	
D26	x	x			x			x	x	x			x
D25	x		x	x	x		x		x		x	x	x
D24		x		x	x	x				x		x	x
D23	x		x	x	x				x		x	x	
D22		x		x	x					x			x
D21	x		x	x					x			x	
D20		x			x								x
D19	x			x								x	
D18					x			x					x
D17				x			x					x	
D16			x			x					x		
D15		x			x					x			
D14	x			x					x				
D13					x						x		x
D12				x						x		x	
D11			x						x		x		
D10		x						x		x			
D9	x						x		x				
D8			x		x	x					x		x
D7		x		x	x					x		x	
D6	x		x	x					x		x		
D5		x			x					x	x		x
D4	x			x					x	x		x	
D3					x				x				x
D2				x				x				x	
D1			x				x				x		
D0		x				x				x			

3.6.5 CRC-24 Code

The following polynomial shall be used to generate the 24-bit CRC for Control Symbol 64:

$$x^{24}+x^{22}+x^{20}+x^{19}+x^{18}+x^{16}+x^{14}+x^{13}+x^{11}+x^{10}+x^{8}+x^{7}+x^{6}+x^{3}+x+1$$

This polynomial factors into:

$$(x^{11}+x^9+x^8+x^7+x^6+x^3+1)(x^{11}+x^9+x^8+x^7+x^5+x^3+x^2+x+1)(x+1)(x+1)$$

The 24-bit CRC shall be computed over control symbol bits 0 through 37 beginning with control symbol bit 0. Before the CRC is computed, the CRC shall be set to all1s (0b1111_1111_1111_1111_1111_1111). Note that the alignment field shall be treated as 0 by the encode and decode functions, as described in Table 3-1

The CRC check bits, c[0:23], occupy Control Symbol 64 bits, [38:61], respectively.

The 24-bit CRC shall be generated by each transmitter and verified by each receiver using the Control Symbol 64.

3.6.6 CRC-24 Parallel Code Generation

For the CRC-24 parallel code generation, the equations are shown in Table 3-27, using rules as for the CRC-5 parallel generation.

Table 3-27. Parallel CRC-24 Equations

Control Symbol	CRC Checksum Bits																							
Data For CRC	**C 0**	**C 1**	**C 2**	**C 3**	**C 4**	**C 5**	**C 6**	**C 7**	**C 8**	**C 9**	**C 10**	**C 11**	**C 12**	**C 13**	**C 14**	**C 15**	**C 16**	**C 17**	**C 18**	**C 19**	**C 20**	**C 21**	**C 22**	**C 23**
D37		x		!x	!x	!x		x		!x	x		x	!x		!x	x	!x			!x		!x	x
D36	x		x	x	x		!x		!x	x		x	x		x	x	x			!x		!x	x	
D35			x		x				x	x		x	x		x		x	x	!x			x	x	x
D34		x		x				x	x		x	x		x		x	x	x			x	x	x	
D33	x		x				x	x		x	x		x		x	x	x			x	x	x		
D32				x	x		x	x	x		x	x	x		x		x	x	x	x			x	x
D31			x	x		x	x	x		x	x	x		x		x	x	x	x			x	x	
D30		x	x		x	x	x		x	x	x		x		x	x	x	x			x	x		
D29	x	x		x	x	x		x	x	x		x		x	x	x	x			x	x			
D28	x	x	x			x	x		x	x					x		x	x	x	x	x		x	x
D27	x			x					x	x	x		x						x	x	x	x		x
D26		x	x	x	x	x			x		x	x	x	x		x	x		x	x				x
D25	x	x	x	x	x			x		x	x	x	x		x	x		x	x				x	
D24	x		x		x	x	x	x	x			x	x		x	x					x	x	x	x
D23							x			x		x	x		x	x	x	x		x		x		x
D22						x			x		x	x		x	x	x	x		x		x		x	
D21					x			x		x	x		x	x	x	x		x		x		x		
D20				x			x		x	x		x	x	x	x		x		x		x			
D19			x			x		x	x		x	x	x	x		x		x		x				
D18		x			x		x	x		x	x	x	x		x		x		x					
D17	x			x		x	x		x	x	x	x		x		x		x						

Control Symbol	**CRC Checksum Bits**																							
D16		x	x	x					x					x	x	x		x			x		x	x
D15	x	x	x					x					x	x	x		x			x		x	x	
D14	x			x	x	x	x	x		x	x	x					x	x	x			x	x	x
D13		x	x				x	x	x				x	x								x		x
D12	x	x				x	x	x				x	x								x		x	
D11	x	x		x			x	x		x		x	x	x		x	x	x		x	x	x	x	x
D10	x	x	x	x	x		x	x	x	x		x		x	x			x	x	x		x		x
D9	x		x		x		x		x	x						x			x					x
D8					x				x	x	x		x	x	x	x	x				x			x
D7				x				x	x	x		x	x	x	x	x				x			x	
D6			x				x	x	x		x	x	x	x	x				x			x		
D5		x				x	x	x		x	x	x	x	x				x			x			
D4	x				x	x	x		x	x	x	x	x				x			x				
D3		x							x			x	x	x			x	x	x		x		x	x
D2	x							x			x	x	x			x	x	x		x		x	x	
D1		x		x	x	x	x	x				x	x	x	x			x	x			x	x	x
D0	x		x	x	x	x	x				x	x	x	x			x	x			x	x	x	

Chapter 4 8b/10b PCS and PMA Layers

4.1 Introduction

This chapter specifies the functions provided by the Physical Coding Sublayer (PCS) and Physical Media Attachment (PMA) sublayer used for 8b/10b encoded links. (The PCS and PMA terminology is adopted from IEEE 802.3). The topics include character representation, scrambling, lane striping, 8b/10b encoding, serialization of the data stream, code-groups, columns, link transmission rules, idle sequences, and link initialization. The 8b/10b PCS and PMA Layers shall be used by links operating at Baud Rate Class 1 or Baud Rate Class 2.

The concept of lanes is used to describe the width of a LP-Serial link. A lane is a single unidirectional signal path between two LP-Serial ports. Five widths are defined for LP-Serial links, 1, 2, 4, 8 and 16 lanes per direction. A link with N lanes in each direction is referred to as a Nx link, e.g. a link with 4 lanes in each direction is referred to as a 4x link.

4.2 PCS Layer Functions

The Physical Coding Sublayer (PCS) function is responsible for idle sequence generation, lane striping, scrambling and encoding for transmission and decoding, lane alignment, descrambling and destriping on reception. The PCS uses an 8b/10b encoding for transmission over the link.

The PCS also provides mechanisms for determining the operational mode of the port as Nx or 1x operation, and means to detect link states. It provides for clock difference tolerance between the sender and receiver without requiring flow control.

The PCS performs the following transmit functions:

- Dequeues packets and delimited control symbols awaiting transmission as a character stream.
- Scrambles packet and control symbol data if required.
- Stripes the transmit character stream across the available lanes.
- Generates the idle sequence and inserts it into the transmit character stream for each lane when no packets or delimited control symbols are available for transmission.

- Encodes the character stream of each lane independently into 10-bit parallel code-groups.
- Passes the resulting 10-bit parallel code-groups to the PMA.

The PCS performs the following receive functions:

- Decodes the received stream of 10-bit parallel code-groups for each lane independently into characters.
- Marks characters decoded from invalid code-groups as invalid.
- If the link is using more than one lane, aligns the character streams to eliminate the skew between the lanes and reassembles (destripes) the character stream from each lane into a single character stream.
- Descrambles packet and control symbol data if required.
- Delivers the decoded character stream of packets and delimited control symbols to the higher layers.

4.3 PMA Layer Functions

The Physical Medium Attachment (PMA) Layer is responsible for serializing and de-serializing 10-bit parallel code-groups to and from a serial bitstream on a lane-by-lane basis. Upon receiving data, the PMA function provides alignment of the received bitstream to 10-bit code-group boundaries, independently on a lane-by-lane basis. It then provides a continuous stream of 10-bit code-groups to the PCS, one stream for each lane. The 10-bit code-groups are not observable by layers higher than the PCS.

If a LP-Serial port supports either baud rate discovery or adaptive equalization, these functions are also performed in the PMA Layer.

4.4 Definitions

Definitions of terms used in this specification are provided below.

1x mode: A LP-Serial port mode of operation in which the port transmits on a single lane and receives on a single lane.

Byte: An 8-bit unit of information. Each bit of a byte has the value 0 or 1.

Character: A 9-bit entity comprised of an information byte and a control bit that indicates whether the information byte contains data or control information. The control bit has the value D or K indicating that the information byte contains respectively data or control information.

Code-group: A 10-bit entity that is the result of 8b/10b encoding a character.

Column: The group of N characters that are transmitted at nominally the same time by a LP-Serial port operating in Nx mode.

Comma: A 7-bit pattern, unique to certain 8b/10b special code-groups, that is used by a receiver to determine code-group boundaries. See more in Section 4.5.7.4, "Sync (/K/)" and Table 4-2.

D-character: A character whose control bit has the value “D”. Also referred to as a data character.

Destriping: This process reverses the operation done during striping of data across multiple lanes. The method used on a link operating in Nx mode to collect and merge the data across the N lanes received simultaneously and form a single character stream. For each direction of the link, the character stream is merged across the lanes, on a character-by-character basis, beginning with lane 0, continuing in incrementing lane number order across the lanes, and wrapping back to lane 0 for character N.

Idle sequence: The sequence of characters (code-groups after 8b/10b encoding) that is transmitted by a port on each of its active output lanes when the port is not transmitting a packet or control symbol. The idle sequence allows the receiver to maintain bit synchronization, code-group alignment and, if applicable, adaptive equalization settings in between packets and control symbols.

K-character: A character whose control bit has the value “K”. Also referred to as a special character.

Lane: A single unidirectional signal path, typically a differential pair, between two LP-Serial ports.

Lane Alignment: The process of eliminating the skew between the lanes of a LP-Serial link operating in Nx mode such that the characters transmitted as a column by the sender are output by the alignment process of receiver as a column. Without lane alignment, the characters transmitted as a column might be scattered across several columns output by the receiver. The alignment process uses the columns of “A” special characters transmitted as part of the idle sequence.

Nx mode: A LP-Serial port mode of operation in which the port both transmits and receives on multiple lanes. A LP-Serial port operating in Nx mode transmits on N lanes and receives on N lanes where N has a value greater then 1. The transmit data stream is distributed across the N transmit lanes and the receive data stream is distributed across the N receive lanes.

Nx port: A LP-Serial port that supports links with up to a maximum of N lanes in each direction.

Striping: The method used on a link operating in Nx mode to distribute data across the N lanes simultaneously. For each direction of the link, the character stream is *striped* across the lanes, on a character-by-character basis, beginning with lane 0, continuing in incrementing lane number order across the lanes, and wrapping back to lane 0 for character N.

4.5 8b/10b Transmission Code

The 8b/10b transmission code used by the PCS encodes 9-bit characters (8 bits of information and a control bit) into 10-bit code-groups for transmission and reverses the process on reception. Encodings are defined for 256 data characters and 12 special characters.

The code-groups comprising the 8b/10b code have either an equal number of ones and zeros (balanced) or the number of ones differs from the number of zeros by two (unbalanced). This eases the task of maintaining 0/1 balance. The selection of code-groups also guarantees a minimum of three transitions, 0 to 1 or 1 to 0, within each code-group. For encoding, unbalanced code-groups are grouped in pairs with one member of the pair having more ones than zeros and the other member of the pair having more zeros than ones. This allows the encoder, when selecting an unbalanced code-group, to select a code-group unbalanced toward ones or unbalanced toward zeros, depending on which is required to maintain the 0/1 balance of the encoder output code-group stream.

The 8b/10b code has the following properties.

- Sufficient bit transition density (3 to 8 transitions per code-group) to allow clock recovery by the receiver.
- Special code-groups that are used for establishing the receiver synchronization to the 10-bit code-group boundaries, delimiting control symbols and maintaining receiver bit and code-group boundary synchronization.
- 0/1 balanced. (can be AC coupled)
- Detection of all single and some multiple-bit errors.

4.5.1 Character and Code-Group Notation

The description of 8b/10b encoding and decoding uses the following notation for characters, code-group and their bits.

The information bits ([0-7]) of an unencoded character are denoted with the letters “A” through “H” where the letter “H” denotes the most significant information bit (RapidIO bit 0) and the letter “A” denotes the least significant information bit (RapidIO bit 7). This is shown in Figure 4-1.

Each data character has a representation of the form Dx.y where x is the decimal value of the least significant 5 information bits EDCBA, and y is the decimal value of the most significant 3 information bits HGF as shown in Figure 4-1. Each special character has a similar representation of the form Kx.y.

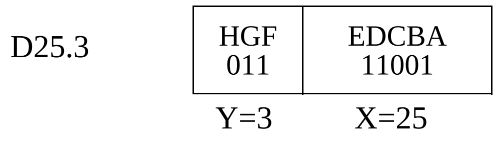


Figure 4-1. Character Notation Example (D25.3)

The output of the 8b/10b encoding process is a 10-bit code-group. The bits of a code-group are denoted with the letters “a” through “j”. The bits of a code-group are all of equal significance, there is no most significant or least significant bit. The ordering of the code-group bits is shown in Figure 4-2.

The code-groups corresponding to the data character Dx.y is denoted by /Dx.y/. The code-groups corresponding to the special character Kx.y is denoted by /Kx.y/.

	abcdei	fghj
/D25.3/	100110	1100

Figure 4-2. Code-Group Notation Example (/D25.3/)

4.5.2 Running Disparity

The 8b/10b encoding and decoding functions use a binary variable called running disparity. The variable can have a value of either positive (RD+) or negative (RD-). The encoder and decoder each have a running disparity variable for each lane which are all independent of each other.

The primary use of running disparity in the encoding process is to keep track of whether the decoder has output more ones or more zeros. The current value of encoder running disparity is used to select the which unbalanced code-group will be used when the encoding for a character requires a choice between two unbalanced code-groups.

Another use of running disparity in the decoding process is to detect errors. Given a value of decoder running disparity, only (256 + 12) = 268 of the 1024 possible code-group values have defined decodings. The remaining 756 possible code-group values have no defined decoding and represent errors, either in that code-group or in an earlier code-group.

4.5.3 Running Disparity Rules

After power-up and before the port is operational, both the transmitter (encoder) and receiver (decoder) must establish current values of running disparity.

The transmitter shall use a negative value as the initial value for the running disparity for each lane.

The receiver may use either a negative or positive initial value of running disparity for each lane.

The following algorithm shall be used for calculating the running disparity for each lane. In the encoder, the algorithm operates on the code-group that has just been generated by the encoder. In the receiver, the algorithm operates on the received code-group that has just been decoded by the decoder.

Each code-group is divided to two sub-blocks as shown in Figure 4-2, where the first six bits (abcdei) form one sub-block (6-bit sub-block) and the second four bits (fghj) form a second sub-block (4-bit sub-block). Running disparity at the beginning of the 6-bit sub-block is the running disparity at the end of the preceding code-group. Running disparity at the beginning of the 4-bit sub-block is the running disparity at the end of the preceding 6-bit sub-block. Running disparity at the end of the code-group is the running disparity at the end of the 4-bit sub-block.

The sub-block running disparity shall be calculated as follows:

1. The running disparity is positive at the end of any sub-block if the sub-block contains more 1s than 0s. It is also positive at the end of a 4-bit sub-block if the sub-block has the value 0b0011 and at the end of a 6-bit sub-block if the sub-block has the value 0b000111.
2. The running disparity is negative at the end of any sub-block if the sub-block contains more 0s than 1s. It is also negative at the end of a 4-bit sub-block if the sub-block has the value 0b1100 and at the end of a 6-bit sub-block if the sub-block has the value 0b111000.
3. In all other cases, the value of the running disparity at the end of the sub-block is running disparity at the beginning of the sub-block (the running disparity is unchanged).

4.5.4 8b/10b Encoding

The 8b/10b encoding function encodes 9-bit characters into 10-bit code-groups.

The encodings for the 256 data characters (Dx.y) are specified in Table 4-1. The encodings for the 12 special characters (Kx.y) are specified in Table 4-2. Both tables have two columns of encodings, one marked RD- and one marked RD+. When encoding a character, the code-group in the RD- column is selected if the current value of encoder running disparity is negative and the code-group in the RD+ column is selected if the current value of encoder running disparity is positive.

Data characters (Dx.y) shall be encoded according to Table 4-1 and the current value of encoder running disparity. Special characters (Kx.y) shall be encoded according to Table 4-2 and the current value of encoder running disparity. After each character is encoded, the resulting code-group shall be used by the encoder to update the running disparity according to the rules in Section 4.5.3, "Running Disparity Rules".

4.5.5 Transmission Order

The parallel 10-bit code-group output of the encoder shall be serialized and transmitted with bit “a” transmitted first and a bit ordering of “abcdeifghj”. This is shown in Figure 4-3.

Figure 4-3 gives an overview of a character passing through the encoding, serializing, transmission, deserializing, and decoding processes. The left side of the figure shows the transmit process of encoding a character stream using 8b/10b encoding and the 10-bit serialization. The right side shows the reverse process of the receiver deserializing and using 8b/10b decoding on the received code-groups.

The dotted line shows the functional separation between the PCS, that provides 10-bit code-groups, and the PMA Layer that serializes the code-groups.

The drawing also shows on the receive side the bits of a special character containing the comma pattern that is used by the receiver to establish 10-bit code-boundary synchronization.

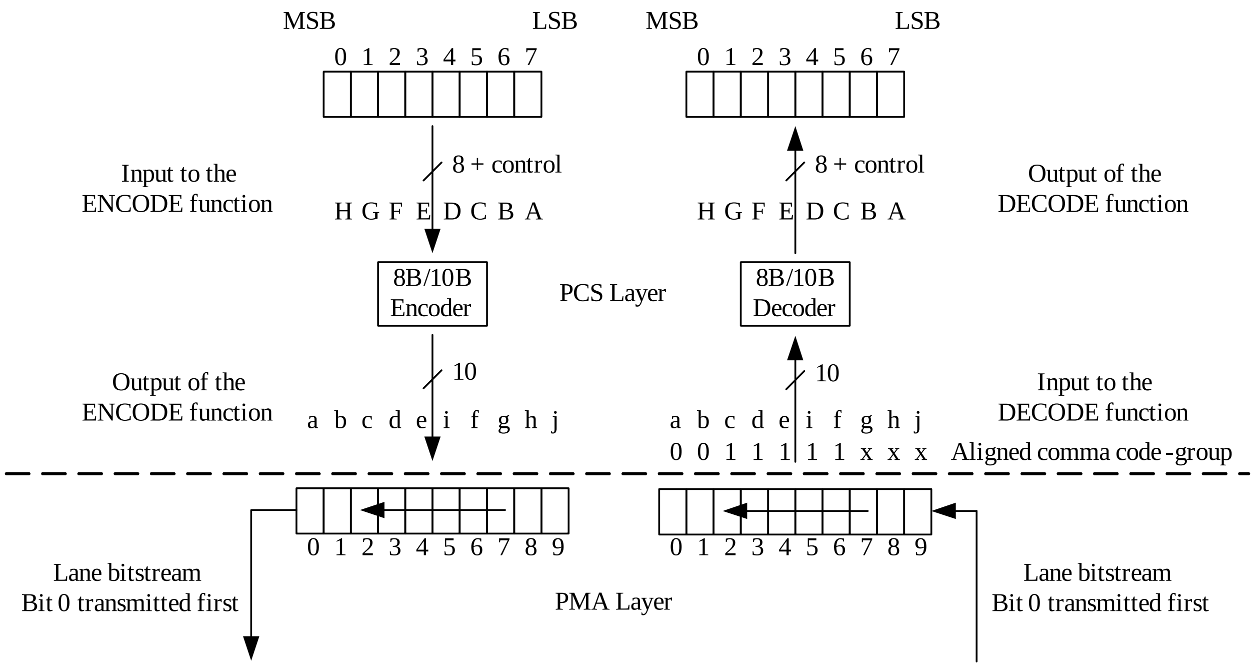


Figure 4-3. Lane Encoding, Serialization, Deserialization, and Decoding Process

4.5.6 8b/10b Decoding

The 8b/10b decoding function decodes received 10-bit code-groups into 9-bit characters and detects and reports received code-groups that have no defined decoding due to one or more transmission errors.

The decoding function uses Table 4-1, Table 4-2 and the current value of the decoder running disparity. To decode a received code-group, the decoder shall select the RD- column of Table 4-1 and Table 4-2 if the current value of the decoder running disparity is negative or shall select the RD+ column if the value is positive. The decoder shall then compare the received code-group with the code-groups in the selected column of both tables. If a match is found in one of the tables, the code-group is defined to be a “valid” code-group and is decoded to the associated character. If no match is found, the code-group is defined to be an “invalid” code-group and is decoded to a character that is flagged in some manner as INVALID. After each code-group is decoded, the decoded code-group shall be used by the decoder to update the decoder running disparity according to the rules in Section 4.5.3, "Running Disparity Rules".

Table 4-1. Data Character Encodings

Character Name	Character Value (hex)	Character Bits HGF EDCBA	Current RD –	Current RD +
			abcdei fghj	abcdei fghj
D0.0	00	000 00000	100111 0100	011000 1011
D1.0	01	000 00001	011101 0100	100010 1011
D2.0	02	000 00010	101101 0100	010010 1011
D3.0	03	000 00011	110001 1011	110001 0100
D4.0	04	000 00100	110101 0100	001010 1011
D5.0	05	000 00101	101001 1011	101001 0100
D6.0	06	000 00110	011001 1011	011001 0100
D7.0	07	000 00111	111000 1011	000111 0100
D8.0	08	000 01000	111001 0100	000110 1011
D9.0	09	000 01001	100101 1011	100101 0100
D10.0	0A	000 01010	010101 1011	010101 0100
D11.0	0B	000 01011	110100 1011	110100 0100
D12.0	0C	000 01100	001101 1011	001101 0100
D13.0	0D	000 01101	101100 1011	101100 0100
D14.0	0E	000 01110	011100 1011	011100 0100
D15.0	0F	000 01111	010111 0100	101000 1011
D16.0	10	000 10000	011011 0100	100100 1011
D17.0	11	000 10001	100011 1011	100011 0100
D18.0	12	000 10010	010011 1011	010011 0100
D19.0	13	000 10011	110010 1011	110010 0100
D20.0	14	000 10100	001011 1011	001011 0100
D21.0	15	000 10101	101010 1011	101010 0100
D22.0	16	000 10110	011010 1011	011010 0100
D23.0	17	000 10111	111010 0100	000101 1011
D24.0	18	000 11000	110011 0100	001100 1011
D25.0	19	000 11001	100110 1011	100110 0100
D26.0	1A	000 11010	010110 1011	010110 0100
D27.0	1B	000 11011	110110 0100	001001 1011
D28.0	1C	000 11100	001110 1011	001110 0100
D29.0	1D	000 11101	101110 0100	010001 1011
D30.0	1E	000 11110	011110 0100	100001 1011
D31.0	1F	000 11111	101011 0100	010100 1011
D0.1	20	001 00000	100111 1001	011000 1001

Table 4-1. Data Character Encodings

Character Name	Character Value (hex)	Character Bits HGF EDCBA	Current RD –	Current RD +
			abcdei fghj	abcdei fghj
D1.1	21	001 00001	011101 1001	100010 1001
D2.1	22	001 00010	101101 1001	010010 1001
D3.1	23	001 00011	110001 1001	110001 1001
D4.1	24	001 00100	110101 1001	001010 1001
D5.1	25	001 00101	101001 1001	101001 1001
D6.1	26	001 00110	011001 1001	011001 1001
D7.1	27	001 00111	111000 1001	000111 1001
D8.1	28	001 01000	111001 1001	000110 1001
D9.1	29	001 01001	100101 1001	100101 1001
D10.1	2A	001 01010	010101 1001	010101 1001
D11.1	2B	001 01011	110100 1001	110100 1001
D12.1	2C	001 01100	001101 1001	001101 1001
D13.1	2D	001 01101	101100 1001	101100 1001
D14.1	2E	001 01110	011100 1001	011100 1001
D15.1	2F	001 01111	010111 1001	101000 1001
D16.1	30	001 10000	011011 1001	100100 1001
D17.1	31	001 10001	100011 1001	100011 1001
D18.1	32	001 10010	010011 1001	010011 1001
D19.1	33	001 10011	110010 1001	110010 1001
D20.1	34	001 10100	001011 1001	001011 1001
D21.1	35	001 10101	101010 1001	101010 1001
D22.1	36	001 10110	011010 1001	011010 1001
D23.1	37	001 10111	111010 1001	000101 1001
D24.1	38	001 11000	110011 1001	001100 1001
D25.1	39	001 11001	100110 1001	100110 1001
D26.1	3A	001 11010	010110 1001	010110 1001
D27.1	3B	001 11011	110110 1001	001001 1001
D28.1	3C	001 11100	001110 1001	001110 1001
D29.1	3D	001 11101	101110 1001	010001 1001
D30.1	3E	001 11110	011110 1001	100001 1001
D31.1	3F	001 11111	101011 1001	010100 1001
D0.2	40	010 00000	100111 0101	011000 0101
D1.2	41	010 00001	011101 0101	100010 0101
D2.2	42	010 00010	101101 0101	010010 0101
D3.2	43	010 00011	110001 0101	110001 0101
D4.2	44	010 00100	110101 0101	001010 0101

Table 4-1. Data Character Encodings

Character Name	Character Value (hex)	Character Bits HGF EDCBA	Current RD –	Current RD +
			abcdei fghj	abcdei fghj
D5.2	45	010 00101	101001 0101	101001 0101
D6.2	46	010 00110	011001 0101	011001 0101
D7.2	47	010 00111	111000 0101	000111 0101
D8.2	48	010 01000	111001 0101	000110 0101
D9.2	49	010 01001	100101 0101	100101 0101
D10.2	4A	010 01010	010101 0101	010101 0101
D11.2	4B	010 01011	110100 0101	110100 0101
D12.2	4C	010 01100	001101 0101	001101 0101
D13.2	4D	010 01101	101100 0101	101100 0101
D14.2	4E	010 01110	011100 0101	011100 0101
D15.2	4F	010 01111	010111 0101	101000 0101
D16.2	50	010 10000	011011 0101	100100 0101
D17.2	51	010 10001	100011 0101	100011 0101
D18.2	52	010 10010	010011 0101	010011 0101
D19.2	53	010 10011	110010 0101	110010 0101
D20.2	54	010 10100	001011 0101	001011 0101
D21.2	55	010 10101	101010 0101	101010 0101
D22.2	56	010 10110	011010 0101	011010 0101
D23.2	57	010 10111	111010 0101	000101 0101
D24.2	58	010 11000	110011 0101	001100 0101
D25.2	59	010 11001	100110 0101	100110 0101
D26.2	5A	010 11010	010110 0101	010110 0101
D27.2	5B	010 11011	110110 0101	001001 0101
D28.2	5C	010 11100	001110 0101	001110 0101
D29.2	5D	010 11101	101110 0101	010001 0101
D30.2	5E	010 11110	011110 0101	100001 0101
D31.2	5F	010 11111	101011 0101	010100 0101
D0.3	60	011 00000	100111 0011	011000 1100
D1.3	61	011 00001	011101 0011	100010 1100
D2.3	62	011 00010	101101 0011	010010 1100
D3.3	63	011 00011	110001 1100	110001 0011
D4.3	64	011 00100	110101 0011	001010 1100
D5.3	65	011 00101	101001 1100	101001 0011
D6.3	66	011 00110	011001 1100	011001 0011
D7.3	67	011 00111	111000 1100	000111 0011
D8.3	68	011 01000	111001 0011	000110 1100

Table 4-1. Data Character Encodings

Character Name	Character Value (hex)	Character Bits HGF EDCBA	Current RD –	Current RD +
			abcdei fghj	abcdei fghj
D9.3	69	011 01001	100101 1100	100101 0011
D10.3	6A	011 01010	010101 1100	010101 0011
D11.3	6B	011 01011	110100 1100	110100 0011
D12.3	6C	011 01100	001101 1100	001101 0011
D13.3	6D	011 01101	101100 1100	101100 0011
D14.3	6E	011 01110	011100 1100	011100 0011
D15.3	6F	011 01111	010111 0011	101000 1100
D16.3	70	011 10000	011011 0011	100100 1100
D17.3	71	011 10001	100011 1100	100011 0011
D18.3	72	011 10010	010011 1100	010011 0011
D19.3	73	011 10011	110010 1100	110010 0011
D20.3	74	011 10100	001011 1100	001011 0011
D21.3	75	011 10101	101010 1100	101010 0011
D22.3	76	011 10110	011010 1100	011010 0011
D23.3	77	011 10111	111010 0011	000101 1100
D24.3	78	011 11000	110011 0011	001100 1100
D25.3	79	011 11001	100110 1100	100110 0011
D26.3	7A	011 11010	010110 1100	010110 0011
D27.3	7B	011 11011	110110 0011	001001 1100
D28.3	7C	011 11100	001110 1100	001110 0011
D29.3	7D	011 11101	101110 0011	010001 1100
D30.3	7E	011 11110	011110 0011	100001 1100
D31.3	7F	011 11111	101011 0011	010100 1100
D0.4	80	100 00000	100111 0010	011000 1101
D1.4	81	100 00001	011101 0010	100010 1101
D2.4	82	100 00010	101101 0010	010010 1101
D3.4	83	100 00011	110001 1101	110001 0010
D4.4	84	100 00100	110101 0010	001010 1101
D5.4	85	100 00101	101001 1101	101001 0010
D6.4	86	100 00110	011001 1101	011001 0010
D7.4	87	100 00111	111000 1101	000111 0010
D8.4	88	100 01000	111001 0010	000110 1101
D9.4	89	100 01001	100101 1101	100101 0010
D10.4	8A	100 01010	010101 1101	010101 0010
D11.4	8B	100 01011	110100 1101	110100 0010
D12.4	8C	100 01100	001101 1101	001101 0010

Table 4-1. Data Character Encodings

Character Name	Character Value (hex)	Character Bits HGF EDCBA	Current RD –	Current RD +
			abcdei fghj	abcdei fghj
D13.4	8D	100 01101	101100 1101	101100 0010
D14.4	8E	100 01110	011100 1101	011100 0010
D15.4	8F	100 01111	010111 0010	101000 1101
D16.4	90	100 10000	011011 0010	100100 1101
D17.4	91	100 10001	100011 1101	100011 0010
D18.4	92	100 10010	010011 1101	010011 0010
D19.4	93	100 10011	110010 1101	110010 0010
D20.4	94	100 10100	001011 1101	001011 0010
D21.4	95	100 10101	101010 1101	101010 0010
D22.4	96	100 10110	011010 1101	011010 0010
D23.4	97	100 10111	111010 0010	000101 1101
D24.4	98	100 11000	110011 0010	001100 1101
D25.4	99	100 11001	100110 1101	100110 0010
D26.4	9A	100 11010	010110 1101	010110 0010
D27.4	9B	100 11011	110110 0010	001001 1101
D28.4	9C	100 11100	001110 1101	001110 0010
D29.4	9D	100 11101	101110 0010	010001 1101
D30.4	9E	100 11110	011110 0010	100001 1101
D31.4	9F	100 11111	101011 0010	010100 1101
D0.5	A0	101 00000	100111 1010	011000 1010
D1.5	A1	101 00001	011101 1010	100010 1010
D2.5	A2	101 00010	101101 1010	010010 1010
D3.5	A3	101 00011	110001 1010	110001 1010
D4.5	A4	101 00100	110101 1010	001010 1010
D5.5	A5	101 00101	101001 1010	101001 1010
D6.5	A6	101 00110	011001 1010	011001 1010
D7.5	A7	101 00111	111000 1010	000111 1010
D8.5	A8	101 01000	111001 1010	000110 1010
D9.5	A9	101 01001	100101 1010	100101 1010
D10.5	AA	101 01010	010101 1010	010101 1010
D11.5	AB	101 01011	110100 1010	110100 1010
D12.5	AC	101 01100	001101 1010	001101 1010
D13.5	AD	101 01101	101100 1010	101100 1010
D14.5	AE	101 01110	011100 1010	011100 1010
D15.5	AF	101 01111	010111 1010	101000 1010
D16.5	B0	101 10000	011011 1010	100100 1010

Table 4-1. Data Character Encodings

Character Name	Character Value (hex)	Character Bits HGF EDCBA	Current RD –	Current RD +
			abcdei fghj	abcdei fghj
D17.5	B1	101 10001	100011 1010	100011 1010
D18.5	B2	101 10010	010011 1010	010011 1010
D19.5	B3	101 10011	110010 1010	110010 1010
D20.5	B4	101 10100	001011 1010	001011 1010
D21.5	B5	101 10101	101010 1010	101010 1010
D22.5	B6	101 10110	011010 1010	011010 1010
D23.5	B7	101 10111	111010 1010	000101 1010
D24.5	B8	101 11000	110011 1010	001100 1010
D25.5	B9	101 11001	100110 1010	100110 1010
D26.5	BA	101 11010	010110 1010	010110 1010
D27.5	BB	101 11011	110110 1010	001001 1010
D28.5	BC	101 11100	001110 1010	001110 1010
D29.5	BD	101 11101	101110 1010	010001 1010
D30.5	BE	101 11110	011110 1010	100001 1010
D31.5	BF	101 11111	101011 1010	010100 1010
D0.6	C0	110 00000	100111 0110	011000 0110
D1.6	C1	110 00001	011101 0110	100010 0110
D2.6	C2	110 00010	101101 0110	010010 0110
D3.6	C3	110 00011	110001 0110	110001 0110
D4.6	C4	110 00100	110101 0110	001010 0110
D5.6	C5	110 00101	101001 0110	101001 0110
D6.6	C6	110 00110	011001 0110	011001 0110
D7.6	C7	110 00111	111000 0110	000111 0110
D8.6	C8	110 01000	111001 0110	000110 0110
D9.6	C9	110 01001	100101 0110	100101 0110
D10.6	CA	110 01010	010101 0110	010101 0110
D11.6	CB	110 01011	110100 0110	110100 0110
D12.6	CC	110 01100	001101 0110	001101 0110
D13.6	CD	110 01101	101100 0110	101100 0110
D14.6	CE	110 01110	011100 0110	011100 0110
D15.6	CF	110 01111	010111 0110	101000 0110
D16.6	D0	110 10000	011011 0110	100100 0110
D17.6	D1	110 10001	100011 0110	100011 0110
D18.6	D2	110 10010	010011 0110	010011 0110
D19.6	D3	110 10011	110010 0110	110010 0110
D20.6	D4	110 10100	001011 0110	001011 0110

Table 4-1. Data Character Encodings

Character Name	Character Value (hex)	Character Bits HGF EDCBA	Current RD –	Current RD +
			abcdei fghj	abcdei fghj
D21.6	D5	110 10101	101010 0110	101010 0110
D22.6	D6	110 10110	011010 0110	011010 0110
D23.6	D7	110 10111	111010 0110	000101 0110
D24.6	D8	110 11000	110011 0110	001100 0110
D25.6	D9	110 11001	100110 0110	100110 0110
D26.6	DA	110 11010	010110 0110	010110 0110
D27.6	DB	110 11011	110110 0110	001001 0110
D28.6	DC	110 11100	001110 0110	001110 0110
D29.6	DD	110 11101	101110 0110	010001 0110
D30.6	DE	110 11110	011110 0110	100001 0110
D31.6	DF	110 11111	101011 0110	010100 0110
D0.7	E0	111 00000	100111 0001	011000 1110
D1.7	E1	111 00001	011101 0001	100010 1110
D2.7	E2	111 00010	101101 0001	010010 1110
D3.7	E3	111 00011	110001 1110	110001 0001
D4.7	E4	111 00100	110101 0001	001010 1110
D5.7	E5	111 00101	101001 1110	101001 0001
D6.7	E6	111 00110	011001 1110	011001 0001
D7.7	E7	111 00111	111000 1110	000111 0001
D8.7	E8	111 01000	111001 0001	000110 1110
D9.7	E9	111 01001	100101 1110	100101 0001
D10.7	EA	111 01010	010101 1110	010101 0001
D11.7	EB	111 01011	110100 1110	110100 1000
D12.7	EC	111 01100	001101 1110	001101 0001
D13.7	ED	111 01101	101100 1110	101100 1000
D14.7	EE	111 01110	011100 1110	011100 1000
D15.7	EF	111 01111	010111 0001	101000 1110
D16.7	F0	111 10000	011011 0001	100100 1110
D17.7	F1	111 10001	100011 0111	100011 0001
D18.7	F2	111 10010	010011 0111	010011 0001
D19.7	F3	111 10011	110010 1110	110010 0001
D20.7	F4	111 10100	001011 0111	001011 0001
D21.7	F5	111 10101	101010 1110	101010 0001
D22.7	F6	111 10110	011010 1110	011010 0001
D23.7	F7	111 10111	111010 0001	000101 1110
D24.7	F8	111 11000	110011 0001	001100 1110

Table 4-1. Data Character Encodings

Character Name	Character Value (hex)	Character Bits HGF EDCBA	Current RD –	Current RD +
			abcdei fghj	abcdei fghj
D25.7	F9	111 11001	100110 1110	100110 0001
D26.7	FA	111 11010	010110 1110	010110 0001
D27.7	FB	111 11011	110110 0001	001001 1110
D28.7	FC	111 11100	001110 1110	001110 0001
D29.7	FD	111 11101	101110 0001	010001 1110
D30.7	FE	111 11110	011110 0001	100001 1110
D31.7	FF	111 11111	101011 0001	010100 1110

Table 4-2. Special Character Encodings

Character Name	Character Value (hex)	Character Bits HGF EDCBA	Current RD –	Current RD +	Notes
			abcdei fghj	abcdei fghj	
K28.0	1C	000 11100	001111 0100	110000 1011	
K28.1	3C	001 11100	001111 1001	110000 0110	2,3
K28.2	5C	010 11100	001111 0101	110000 1010	1
K28.3	7C	011 11100	001111 0011	110000 1100	
K28.4	9C	100 11100	001111 0010	110000 1101	1
K28.5	BC	101 11100	001111 1010	110000 0101	2
K28.6	DC	110 11100	001111 0110	110000 1001	1
K28.7	FC	111 11100	001111 1000	110000 0111	1,2
K23.7	F7	111 10111	111010 1000	000101 0111	1
K27.7	FB	111 11011	110110 1000	001001 0111	
K29.7	FD	111 11101	101110 1000	010001 0111	
K30.7	FE	111 11110	011110 1000	100001 0111	1
Notes 1. Reserved code-group. 2. The code-group contain a comma. 3. A Reserved code-group for Idle Sequence 1					

The “comma” is an important element of 8b/10b encoding. A comma is a pattern of 7 bits that is used by receivers to acquire code-group boundary alignment. Two commas patterns are defined, 0b0011111 (comma+) and 0b1100000 (comma-). The pattern occurs in bits **abcdeif** of the special characters K28.1, K28.5 and K28.7. Within the code-group set, it is a singular bit pattern, which, in the absence of transmission errors, cannot appear in any other location of a code-group and cannot be generated across the boundaries of any two adjacent code-groups with the following exception:

> The /K28.7/ special code-group when followed by any of the data code-groups /D3.y/, /D11.y/, /D12.y/, /D19.y/, /D20.y/, /D28.y/, or /K28.y/, where y is an integer in the range 0 through 7, may (depending on the value of running disparity) cause a comma to be generated across the boundary of the two code-groups. A comma that is generated across the boundary between two adjacent code-groups may cause the receiver to change the 10-bit code-group alignment. As a result, the /K28.7/ special code-group may be used for test and diagnostic purposes only.

4.5.7 Special Characters and Columns

Table 4-3 defines the special characters and columns of special characters used by LP-Serial links. Special characters are used for the following functions:

1. Alignment to code-group (10-bit) boundaries on lane-by-lane basis.
2. Alignment of the receive data stream across N lanes.
3. Marking the start of the IDLE2 CS field
4. Clock rate compensation between receiver and transmitter.
5. Control symbol delimiting.

Table 4-3. Special Characters and Columns

Code-Group/Column Designation	Code-Group/Column Use	Number of Code-groups	Encoding
/PD/	Packet_Delimiter Control Symbol	1	/K28.3/
/SC/	Start_of_Control_Symbol	1	/K28.0/
/K/	Sync	1	/K28.5/
/R/	Skip	1	/K29.7/
/A/	Align	1	/K27.7/
/M/	Mark	1	/K28.1/
/**I**/	Idle	1	
||K||	Sync column	N	a column of /K28.5/
||R||	Skip column	N	a column of /K29.7/
||A||	Align column	N	a column of /K27.7/
||M|	Mark column	N	a column of /K28.1/
||**I**||	Idle column	N	a column of Idle

4.5.7.1 Packet Delimiter Control Symbol (/PD/)

PD and /PD/ are aliases for respectively the K28.3 character and the /K28.3/ code-group which are used to delimit a control symbol that contains a packet delimiter.

4.5.7.2 Start of Control Symbol (/SC/)

SC and /SC/ are aliases for respectively the K28.0 character and the /K28.0/ code-group which are used to delimit a control symbol that does not contain a packet delimiter.

4.5.7.3 Idle (/I/)

I and /I/ are aliases for respectively any of the idle sequence characters and idle sequence code-groups.

4.5.7.4 Sync (/K/)

K and /K/ are aliases for respectively the K28.5 character and the /K28.5/ code-group which are used in idle sequences to provide the receiver with the information it requires to achieve and maintain bit and 10-bit code-group boundary synchronization. /K28.5/ was selected as the Sync character as it contains the comma pattern in bits **abcdeif** which is required to locate the code-group boundaries and it provides the maximum number of transitions in bits **ghj**.

4.5.7.5 Skip (/R/)

R and /R/ are aliases for respectively the K29.7 character and the /K29.7/ code-group which are used in the idle sequences and in the clock compensation sequence.

4.5.7.6 Align (/A/)

A and /A/ are aliases for respectively the K27.7 character and the /K27.7/ code-group which are used in idle sequences and for lane alignment on links operating in Nx mode.

4.5.7.7 Mark (/M/)

M and /M/ are aliases for respectively the K28.1 character and the /K28.1/ code-group which are used in Idle Sequence 2 to provide the receiver with the information it requires to achieve and maintain 10-bit code-group boundary synchronization and to mark the location of the Idle frame CS field.

4.5.7.8 Illegal

A special character and its associated code-group that is defined by the 8b/10b code, but not specified for use by the LP-Serial protocol are declared to be an “illegal” character and “illegal” code-group respectively. The special characters K23.7, K28.2, K28.4, K28.6, K28.7 and K30.7 are illegal characters, and if a link is operating with Idle Sequence 1, K28.1 is also an illegal character.

4.5.8 Effect of Single Bit Code-Group Errors

Except in receivers using decision feedback equalization (DFE), single bit code-group errors will be the dominant code-group error by many orders of magnitude. It is therefore useful to know the variety of code-group corruptions that can be caused by a single bit error.

Table 4-4 lists all possible code-group corruptions that can be caused by a single-bit error. The notation /X/ => /Y/ means that the code-group for the character X has been corrupted by a single-bit error into the code-group for the character Y. If the corruption results in a code-group that is invalid for the current receiver running disparity, the notation /X/ => /INVALID/ is used. The table provides the information required to deterministically detect all isolated single bit transmission errors on links operating with idle sequence 1 and Control Symbol 24.

Table 4-4. Code-Group Corruption Caused by Single Bit Errors

Corruption	Detection on links using idle sequence 1 and Control Symbol 24
/SC/ => /INVALID/	Detectable as an error when decoding the code-group. When this error occurs within a packet, it is indistinguishable from a /Dx.y/ => /INVALID/. When this error occurs outside of a packet, the type of error can be inferred from whether the /INVALID/ is followed by the three /Dx.y/ that comprise the control symbol data.
/PD/ => /INVALID/	Detectable as an error when decoding the code-group. When this error occurs within a packet, it is indistinguishable from a /Dx.y/ => /INVALID/. When this error occurs outside of a packet, the type of error can be inferred from whether the /INVALID/ is followed by the three /Dx.y/ that comprise the control symbol data.
/A/, /K/ or /R/ => /Dx.y/	Detectable as an error as /Dx.y/ is illegal outside of a packet or control symbol and /A/, /K/ and /R/ are illegal within a packet or control symbol.
/A/, /K/ or /R/ => /INVALID/	Detectable as an error when decoding the code-group.
/Dx.y/ => /A/, /K/ or /R/	Detectable as an error as /A/, /K/ and /R/ are illegal within a packet or control symbol and /Dx.y/ is illegal outside of a packet or control symbol.
/Dx.y/ => /INVALID/	Detectable as an error when decoding the code-group.
/Dx.y/ => /Du.v/	Detectable as an error by the packet or control symbol CRC. The error will also result in a subsequent unerrored code-group being decoded as INVALID, but that resulting INVALID code-group may occur an arbitrary number of code-groups after the errored code-group.

4.6 LP-Serial Link Widths

LP-Serial links may have 1, 2, 4, 8, or 16 lanes per direction. All LP-Serial ports shall support operation on links with one lane per direction (1x mode) and may optionally support operation over links with 2, 4, 8 and/or 16 lanes per direction (respectively 2x mode, 4x mode, 8x mode and 16x mode). For example, a port that supports operation over 8 lanes per direction (8x mode) must also support operation over one lane per direction (1x mode) and may optionally also support operation over 2 and/or 4 lanes per direction (2x mode and/or 4x mode). The requirement that all LP-Serial ports support 1x mode is to ensure that any pair of LP-Serial ports that are capable of operating at the same baud rate also support a common link width over which they can always communicate with each other.

LP-Serial ports that support operation over two or more lanes per direction shall support 1x mode operation over two of those lanes, lane 0 and lane R (the redundancy lane). If the port supports operation over at most two lanes per direction (2x mode), lane R shall be lane 1. If the port supports operation over more than two lanes, lane R shall be lane 2. Requiring ports that support operation over links with two or more lanes per direction to also support 1x mode over two lanes per direction provides a redundant fallback capability that allows communication over the link at reduced bandwidth in the presence of lane failure, regardless of the lane that fails.

4.7 Idle Sequence

An idle sequence is a sequence of characters, code-groups after 8b/10b encoding, that is transmitted by a LP-Serial port on each of its active output lanes when the port is not initialized and, when the port is initialized, there is nothing else to transmit. At a minimum, an idle sequence provides the information required by a LP-Serial receiver to acquire and retain bit, code-group and lane alignment and contains clock compensation sequences.

Two idles sequences are defined, Idle Sequence 1, which is referred to as IDLE1, and Idle sequence 2, which is referred to as IDLE2. Both sequences contain the /K/, /A/ and /R/ special code-groups that are required respectively for establishing code-group and lane alignment in the LP-Serial receiver and providing clock compensation.

IDLE1 was the first idle sequence defined for LP-Serial links and is unchanged from the IDLE specified in Rev. 1.3 of this specification. It is based on and is very similar to the idle sequence used by XAUI, an interconnect that is defined in Clause 47 of IEEE Standard 802.3. IDLE1 was designed for LP-Serial Baud Rate Class 1 links and transmitters and receivers that do not use adaptive equalization. IDLE1 provides only the minimum idle sequence functionality.

IDLE2 was designed for LP-Serial Baud Rate Class 2 links and transmitters and receivers using adaptive equalization. In addition to the minimum idle sequence functionality, IDLE2 provides link width, lane identification and lane polarity information, randomized data for equalizer training and a command and status channel for receiver control of the transmit equalizer.

When idle is transmitted by a LP-Serial port, an idle sequence shall be transmitted on each of the port’s active output lanes. Ports operating in Nx mode shall not stripe the idle sequence across the active lanes; there is an idle sequence for each of the N lanes.

An uninitialized LP-Serial port (state variable port_initialized not asserted) shall continuously transmit an idle sequence on all active output lanes. An initialized LP-Serial port (state variable port_initialized asserted) shall transmit an idle sequence on each of its active output lanes when there is nothing else to transmit. An idle sequence may not be inserted in a packet or control symbol. An initialized LP-Serial port that becomes uninitialized while transmitting a packet or control symbol may transmit several code-groups per lane of the packet and/or control symbol before beginning the transmission of an idle sequence.

On links operating in 1x mode, the first code-group of the idle sequence shall immediately follow the last code-group of the preceding control symbol. When a link is operating in Nx mode, the first column of N idle code-groups shall immediately follow the column containing the last code-groups of the preceding control symbol.

4.7.1 Clock Compensation Sequence

The “clock compensation sequence” is four character sequence comprised of a K special character immediately followed by three R special characters (K,R,R,R). Clock compensation sequences are transmitted as part of idle sequences.

A port shall transmit a clock compensation sequence on each of its active output lanes at least once every 5000 characters transmitted per lane by the port. When a clock compensation sequence is transmitted, the entire 4 character sequence shall be transmitted. When transmitted by a port operating in Nx mode, the clock compensation sequence shall be transmitted in parallel on all N lanes resulting in the column sequence ||K||R||R||R||.

Since a packet or delimited control symbol may not be interrupted by an idle sequence, it is recommended that a port transmit a clock compensation sequence on each of its active output lanes at least once every 4096 characters transmitted per lane by the port. This requirement implies that the flow of packets and delimited control symbols available from the upper layers can be interrupted long enough to transmit an idle sequence containing a clock compensation sequence.

The compensation sequence allows retimers (discussed in Section 4.11) to compensate for up to a +/- 200 ppm difference between input bit rate and output bit rate. Both rates have a +/-100 ppm tolerance. It may also be used to allow the input side of an end point port to compensate for up to a +/-200 ppm difference between the input bit rate and the bit rate of the device core which may be running off a different clock. This is done by dropping or adding an /R/ immediately following a

/K/ in 1x mode or an ||R|| immediately following a ||K|| in Nx mode as needed to avoid overrun or underrun.

4.7.2 Idle Sequence 1 (IDLE1)

Idle Sequence 1 is a sequence of the special characters A, K and R. The sequence is 8b/10b encoded before transmission, yielding a sequence of the special code-groups /A/, /K/ and /R/ that is transmitted on the link.

The IDLE1 sequence shall comply with the following requirements:

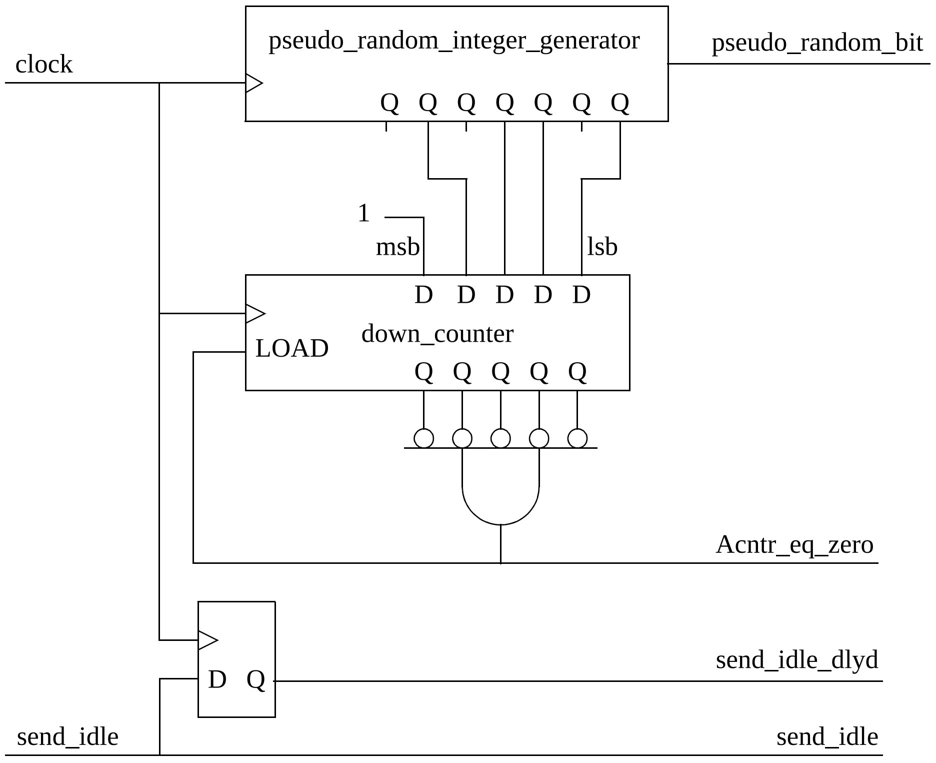
1. Each instance of an IDLE1 sequence shall begin with the K special character.
2. The second, third and fourth characters of each IDLE1 sequence may be the R special character. This allows the first four characters of an IDLE1 sequence to be K,R,R,R, the “clock compensation sequence”.
3. Except when generating the clock compensation sequence, all characters following the first character of an IDLE1 shall be a randomly selected sequence of A, K and R special characters that is based on a pseudo-random sequence generator of 7th degree or greater and subject to minimum and maximum requirements on the spacing of the A special characters. The pseudo-random selection of characters in the idle sequence results in a sequence code-groups whose spectrum has no discrete lines which helps control the EMI of long idle sequences.
4. The number of non-A special characters between A special characters within an IDLE1 sequence shall be no less than 16 and no more than 31. The number shall be pseudo-randomly selected based on a pseudo-random sequence generator of 7th degree of greater. Ideally, the number of non-A characters separating A characters should be uniformly distributed across the range of 16 through 31. However, the IDLE1 spectrum appears to be relatively insensitive to the actual distribution.
5. The requirement on the number of characters between successive A special characters should be maintained between successive IDLE1 sequences to ensure that two successive A special characters are always separated by at least 16 non-A characters.
6. Except when transmitting a clock compensation sequence, an IDLE1 sequence may be of any length and may be terminated after any code-group.
7. Each instance of IDLE1 shall be a new IDLE1 sequence that is unrelated to any previous IDLE1 sequence. Once transmission of an IDLE1 sequence has begun, the sequence may only be terminated. It may not be interrupted or stalled and then continued later.
8. When a port transmitting IDLE1 is operating in Nx mode, the port shall transmit the identical sequence of A, K and R special characters in parallel on each of the N lanes and the N idle sequences shall be aligned across the lanes such that the initial /K/ of the N sequences shall all occur in the same column

and the last code-group of the N sequences shall all occur in the same column. As a result, the IDLE1 sequence will appear as a sequence of the columns ||K||, ||R|| and ||A|| at the transmitter output.

4.7.3 Idle Sequence 1 Generation

A primitive polynomial of at least 7th degree is recommended as the generating polynomial for the pseudo-random sequence that is used in the generation of the idle sequence. The polynomials $x^7 + x^6 + 1$ and $x^7 + x^3 + 1$ are examples of primitive 7th degree polynomials which may be used as generator polynomials. The pseudo-random sequence generator is clocked (generates a new pseudo-random sequence value) once per idle sequence code-group (column). Four of the pseudo-random sequence generator state bits may be selected to generate the pseudo-random value for /A/ spacing. The selection of the state bits and their weighting has a significant effect of the distribution of values for /A/ spacing. Any other state bit or logical function of state bits may be selected as the /K/ vs. /R/ selector.

Figure 4-4 shows an example circuit illustrating how this may be done. The clock ticks whenever a code-group or column is transmitted. Send_idle is asserted whenever an idle sequence begins. The equations indicate the states in which to transmit the indicated idle code-group, except when the compensation sequence is being transmitted. Any equivalent method is acceptable.



send_K = send_idle & (!send_idle_dlyd | send_idle_dlyd & !Acntr_eq_zero & pseudo_random_bit)
send_A = send_idle & send_idle_dlyd & Acntr_eq_zero
send_R = send_idle & send_idle_dlyd & !Acntr_eq_zero & !pseudo_random_bit

Figure 4-4. Example of a Pseudo-Random Idle Code-Group Generator

4.7.4 Idle Sequence 2 (IDLE2)

IDLE 2 is a sequence of data characters and the special characters A, K, M and R. The character sequence is 8b/10b encoded before transmission, yielding a sequence of data code-groups and the special /A/, /K/, /M/ and /R/ code-groups that are transmitted on the link.

The IDLE sequence 2 shall be comprised of a continuous sequence of idle frames and clock compensation sequences. Subject to the following requirements, the exact order of idle frames and clock compensation sequences in an IDLE 2 sequence is implementation dependent.

1. The minimum clock compensation sequence density (clock compensation sequences per characters transmitted per lane) shall comply with the requirements specified in Section 4.7.1, "Clock Compensation Sequence".
2. Each clock compensation sequence shall be followed by an idle frame.
3. Each idle frame shall be followed by either a clock compensation sequence or another idle frame.

4. When a port is operating in Nx mode, the sequence of clock compensation sequences and idle frames shall be the same for all N lanes.

After a port using IDLE2 is initialized (the port initialization state variable port_initialized is asserted), the port may terminate an IDLE2 sequence after any character of an idle frame to transmit a control symbol or a SYNC sequence immediately followed by a link-request control symbol subject to the following requirements:

1. Each M special character transmitted that is part of the idle frame random data field shall be followed by a minimum of four (4) random data field random data characters.
2. The sequence of four (4) M special characters at the beginning of a CS field marker shall not be truncated.
3. A port operating in Nx mode shall terminate an IDLE2 sequence at exactly the same character position in the sequence for each of the N lanes.

Each instance of IDLE2 shall be a new IDLE2 sequence that is unrelated to any previous IDLE2 sequence. Once transmission of an IDLE2 sequence has begun, the sequence may only be terminated. It may not be interrupted or stalled and then continued later.

When a port transmitting IDLE2 is operating in Nx mode, the port shall transmit IDLE2 sequences in parallel on each of the N lanes. The sequences will be similar, but not identical because the information carried in the CS Field Marker will differ from lane to lane and the information carried in the CS Field may also differ from lane to lane. The IDLE2 sequences transmitted on each of the N lanes shall be aligned across the lanes such that the first character of the N idle sequences shall all occur in the same column and the last character of the N idle sequences shall all occur in the same column. As a result, the IDLE2 sequence will appear at the transmitter output as a sequence of the columns ||K||, ||R||, ||M|| and ||A|| and columns containing only data code-groups.

4.7.4.1 Idle Frame

Each idle frame shall be composed of three parts, a random data field, a command and status (CS) field marker and an encoded CS field as shown in Figure 4-5.

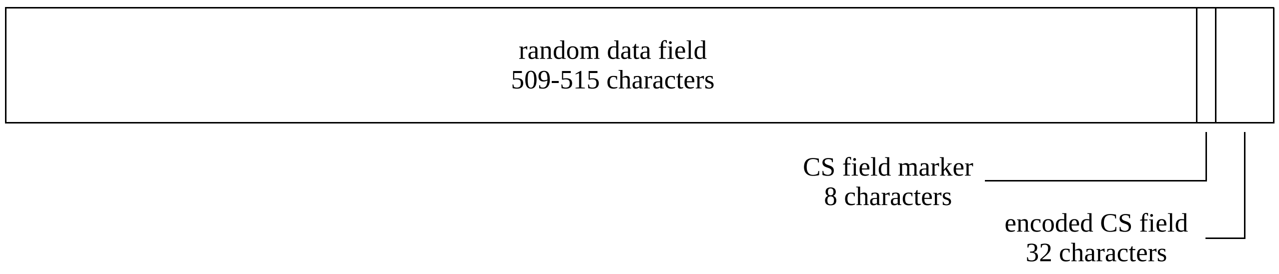


Figure 4-5. Idle Sequence 2 Idle Frame

4.7.4.1.1 IDLE Sequence 2 Random Data Field

The IDLE2 random data field shall contain pseudo-random data characters and the A and M special characters. The total length of the random data field shall be no less than 509 and no more than 515 characters. The idle field shall comply with the following requirements.

1. Unless otherwise specified, the characters comprising the random data field shall be pseudo-random data characters.
2. The random data field of an idle frame that immediately follows a clock compensation sequence shall begin with a M special character. Otherwise, the random data field of an idle frame shall begin with a pseudo-random data character.
3. Unless otherwise specified, the pseudo-random data characters in the random data field shall occur in contiguous sequences of not less than 16 and no more than 31 pseudo-random characters. The length of each contiguous sequence shall be pseudo-randomly selected. The lengths of the contiguous sequences should be uniformly distributed across the range of 16 to 31 characters. Adjacent contiguous sequences shall be separated by a single A or M special character. Each separator shall be pseudo-randomly selected. The probability of selecting the A or M special character for a given separator should be equal. The last four (4) characters of the random data field shall be pseudo-random data characters. The length of the first contiguous sequence of pseudo-random characters in the random data field shall be no less than 16 and no more than 35 characters. The length of the last contiguous sequence of pseudo-random characters in the random data field shall be no less than 4 and no more than 35 characters.
4. Each random data field that is transmitted on a given lane of a link shall be generated by first generating a prototype random data field using the above rules, but with a D0.0 character in the place of each pseudo-random data character, and then scrambling the prototype random data field with the transmit scrambler for that lane. The scrambling shall be done in exactly the same manner as packet and control symbol data characters are scrambled. The scrambler, the scrambling method and the scrambling rules are specified in Section 4.8.1, "Scrambling Rules".
5. When a port is operating in Nx mode, the location A or M special characters in a random data field shall be identical for all N lanes. If the k^{th} character of a random data field transmitted on lane 0 is an A (M) special character, the k^{th} character of the random data fields transmitted on lanes 1 through N-1 is also an A (M) special character.

Generating the random data field pseudo-random data characters by scrambling D0.0 characters results in the output serial random data bit stream being the scrambling sequence. This allows the receiver to recover the descrambler seed from the received idle frame random data field. It also allows the receiver to verify that the lane descramblers are synchronized to the incoming data stream. If a lane descrambler is correctly synchronized, the pseudo-random data characters in the idle frame random data field will all descramble to D0.0 characters.

4.7.4.1.2 IDLE Sequence 2 CS Field Marker

The CS field marker indicates the beginning of the command and status (CS) field and provides information about the link polarity, link width and lane numbering.

The CS field marker shall be the 8 character sequence

M, M, M, M, D21.5, Dx.y, D21.5, $\overline{\text{Dx.y}}$

where

x, the least significant 5 bits of Dx.y, encodes lane_number[0-4], the number of the lane within the port,

y, the most significant 3 bits of Dx.y, encodes active_link_width[0-2], the active width of the port and

$\overline{\text{Dx.y}}$ is the bit wise complement of Dx.y.

As shown above, the CS frame marker characters shall be transmitted from left to right. The first character transmitted is M, the last character transmitted is $\overline{\text{Dx.y}}$.

The “M, M, M, M” sequence that begins the CS field marker is unique and is used to locate the start of the CS data field. The sequence occurs only between the Idle Sequence 2 idle frame random data and CS fields. It never occurs in control symbols or packet data and can not be created by an isolated burst error of 11 bits or less at the code-group level.

The character D21.5 provides lane polarity indication. The 8b/10b encoding of D21.5 is independent of running disparity. If the lane polarity is inverted, the character will decode as D10.2.

If the decoding of the D21.5 characters in the CS field marker is used to detect lane polarity inversion, then consideration shall be given to Section 4.7.4 that allows a CS field marker to be terminated at any point after the initial four M characters of the marker. D21.5 or D10.2 characters can occur in the control symbol and/or packet immediately following the truncated CS field marker in such a pattern that it falsely appears that lane polarity is inverted. Therefore, it is recommended that lane polarity checking mechanisms, if present, should test only correctly formed CS Field Markers, require a consistent indication of lane polarity over multiple CS field markers, and make the lane polarity decision as early in the link initialization process as possible. The lane polarity determination for any of a port’s lanes shall not be changed when the state machine variable port_initialized is asserted.

The active_port_width field shall be encoded as specified in Table 4-5.

Table 4-5. Active Port Width Field Encodings

y	active_link_width[0-2]	Link mode	Notes
0	0b000	1x	
1	0b001	2x	
2	0b010	4x	
3	0b011	8x	
4	0b100	16x	
5	0b101	1x on lanes 0, 1 and 2	3
6	0b110	1x on both lanes 0 and 1	1
7	0b111	1x on both lanes 0 and 2	2

Notes
1. Used when a 2x port is operating in 1x mode.
2. Used when a 4x, 8x, or 16x port is operating in 1x mode.
3. Used when a 1x/2x/Nx port is operating in 1x mode. Some early implementations may report this mode as an active_link_width of 0b110 or 0b111 instead of 0b101.

The lane_number field shall be encoded as specified in Table 4-6.

Table 4-6. Lane Number Field Encodings

x	lane_number[0-4]	lane number
0	0b00000	0
1	0b00001	1
2	0b00010	2
3	0b00011	3
4	0b00100	4
5	0b00101	5
6	0b00110	6
7	0b00111	7
8	0b01000	8
9	0b01001	9
10	0b01010	10
11	0b01011	11
12	0b01100	12
13	0b01101	13
14	0b01110	14
15	0b01111	15
16-31	0b10000 - 0b11111	Reserved

A CS field marker whose first four characters are not all M special characters, fifth and seventh characters are not both D21.5 or D10.2 or sixth and eight character are not the bit wise complements of each other shall be determined to be corrupted. A received CS field marker that is determined to be truncated and/or corrupted shall be ignored and discarded. Any error detected in a truncated and/or corrupted CS field marker that is determined to be the result of a transmission error and not the result of truncation, such as an “invalid” or “illegal” character, shall be reported as an input error.

4.7.4.1.3 IDLE2 Command and Status Field (CS field)

The CS field allows a port to provide certain status information about itself to the connected port and to control the transmit emphasis settings of the connected port if the connected port supports adaptive transmit emphasis.

The CS field shall have 32 information bits, cs_field[0-31], and 32 check bits, cs_field[32-63]. The check bits cs_field[32-63] shall be the bit wise complement of the information bits cs_field[0-31] respectively.

The CS field bits are defined in Table 4-7.

Table 4-7. Command and Status Field Encodings

CS_field bit(s)	Definition
0	CMD - Command This bit indicates to the connected port when an emphasis update command is present 0b0 - no request present 0b1 - request present
1	Implementation defined
2	Receiver trained When the lane receiver controls any transmit or receive adaptive equalization, this bit indicates whether or not all adaptive equalizers controlled by the lane receiver are trained 0b0 - One or more adaptive equalizers are controlled by the lane receiver and at least one of those adaptive equalizers is not trained 0b1 - The lane receiver controls no adaptive equalizers or all of the adaptive equalizers controlled by the receiver are trained
3	Data scrambling/descrambling enabled This bit indicates whether control symbol and packet data characters are being scrambled before transmission and descrambled upon reception This bit indicates whether or not the transmitter is scrambling control symbol and packet data characters. 0b0: scrambling/descrambling disabled 0b1: scrambling/descrambling enabled
4-5	Tap(-1) status - Transmit emphasis tap(-1) status These bits indicate the status of transmit emphasis tap(-1). 0b00: not implemented 0b01: at minimum emphasis 0b10: at maximum emphasis 0b11: at intermediate emphasis setting
6-7	Tap(+1) status - Transmit emphasis tap(+1) status. These bits indicate the status of transmit emphasis tap(+1). 0b00: not implemented 0b01: at minimum emphasis 0b10: at maximum emphasis 0b11: at intermediate emphasis setting

Table 4-7. Command and Status Field Encodings

CS_field bit(s)	Definition
8-23	Reserved
24-25	Tap(-1) Command - Transmit emphasis tap(-1) update command This bit is used in conjunction with the “CMD” bit to change or retain the emphasis setting of tap(-1). 0b00: hold 0b01: decrease emphasis by one step 0b10: increase emphasis by one step 0b11: reserved
26-27	Tap(+1) Command - Transmit emphasis tap(+1) update command This bit is used in conjunction with the “CMD” bit to change or retain the emphasis setting of tap(+1). 0b00: hold 0b01: decrease emphasis by one step 0b10: increase emphasis by one step 0b11: reserved
28	Reset emphasis This bit is used in conjunction with the “CMD” bit to force the transmit emphasis settings in the connected transmitter to no emphasis 0b0: Ignore 0b1: Reset all transmit emphasis taps to no emphasis
29	Preset emphasis This bit is used in conjunction with the “CMD” bit to force the transmit emphasis settings in the connected transmitter to initial or preset values 0b0: Ignore 0b1: Set all transmit emphasis setting to their preset values.
30	ACK This bit indicates when a transmit emphasis update command from the connected port has been accepted. 0b0: command not accepted 0b1: command accepted
31	NACK This bit indicates when a transmit emphasis update command from the connected port has been refused. 0b0: command not refused 0b1: command refused

The 64 cs_field bits shall be encoded in pairs as specified in Table 4-8.

Table 4-8. CS Field 8/10 Bit Encodings

CS_field[n,n+1] n even	Encoding
0,0	D7.3
0,1	D24.3
1,0	D30.3
1,1	D24.7

This encoding has the property that after 8b/10b encoding, the resulting transmit signal has a minimum run length of 2 except at the boundary between code-groups when a /D24.7/ is immediately followed by a /D30.3/. The minimum run length of 2 reduces the effective bandwidth of the transmitted signal which improves the reliability of transmission over an unequalized or partially equalized lane.

The characters encoding the CS channel shall be transmitted in the order of the bits they encode beginning with the character encoding CS field bits [0,1] and ending with the character encoding bits [62-63].

A CS field whose bits [32-63] are not the bit wise complement of bits [0-31] respectively shall be determined to be corrupted. A received CS field that is determined to be truncated and/or corrupted shall be ignored and discarded. Any error detected in a truncated and/or corrupted CS field that is determined to be the result of a transmission error and not the result of truncation, such as an “invalid” or “illegal” character, shall be reported as an input error.

4.7.4.1.4 IDLE2 CS Field Use

The transmit emphasis status and update commands supported by the CS Field are based on a reference model for the transmitter emphasis network that is a transversal filter with K taps with baud period tap spacing. A 5-tap transversal filter is shown in Figure 4-6. The filter taps are named according to their position relative to the “main” tap which is designated tap(0). As the signal propagates through the filter, taps that are reached by the signal before it reaches the main tap are designated with negative integers. Taps that are reached by the signal after it has passed the main tap are designated with positive integers. For example, the tap immediately before the main tap is designated tap(-1), the tap immediately following the main tap is designated tap(+1) and the second tap after the main tap is designated tap(+2). The output signal of a transversal filter is formed by multiplying the voltage of each tap by a tap coefficient and summing the products together. The coefficient for tap(n) is designated k_n. The main tap, tap(0), has the property that its coefficient (k_0) is always positive. When all emphasis is disabled, the main tap coefficient is 1 and all of the other tap coefficients are 0.

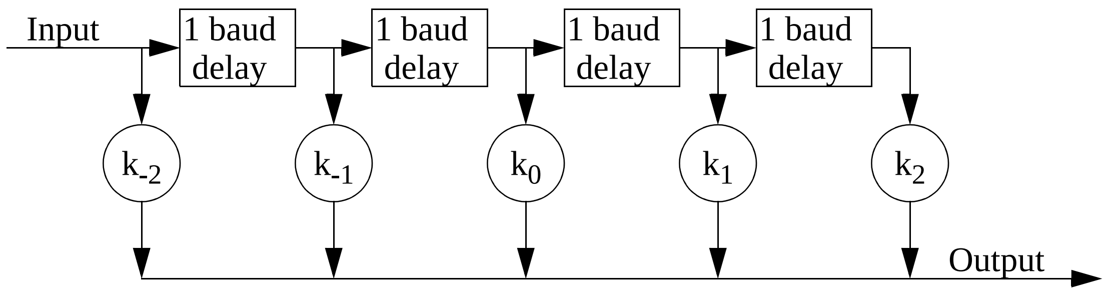


Figure 4-6. 5-tap Transversal Filter

The structure of the transmit emphasis transversal filter in a given port is conveyed to the connected port by the Tap(n) status fields in the CS fields transmitted by the port.

The intended use for transmit emphasis is to allow at least partial compensation for the transmission losses of links implemented with differential printed circuit board (PCB) trace pairs which increase with increasing frequency. Compensation is achieved by emphasizing the higher frequency portion of the transmit spectrum before transmission. A transversal filter for this purpose typically has two or three taps. The two tap filter has a main tap, tap(0), and either a tap(-1) or a tap(+1). The three tap filter has a main tap and both a tap(-1) and a tap(+1). When adjusted for transmit emphasis, the coefficients of tap(-1) and tap(+1) will be negative with emphasis increasing as the coefficients become more negative.

The CS fields exchanged between connected LP-Serial ports provides a command and acknowledgement path that allows a LP-Serial receiver to control the transmit emphasis of the connected transmitter. The issuing and acknowledgement of transmit emphasis commands is control by a handshake that uses the CS field signals CMD, ACK and NACK.

A receiver may issue the following commands. Only one of these commands may be issued at a time.

reset emphasis

preset emphasis

modify the emphasis provided by tap(-1), if tap(-1) is implemented

modify the emphasis of tap(+1), if tap(+1) is implemented

CS field commands shall be issued and acknowledged using the following rules. References to specific command bits and to the CMD bit refer to the specific command bits and the CMD bit in CS fields transmitted by the port issuing the command. References to the ACK and NACK bits refer to the ACK and NACK bits in CS fields received from the connected port. An example of this handshake is shown in Figure 4-7.

Specific command bits may be changed only when the ACK and NACK bits are both de-asserted and the CMD bit is either de-asserted or transitioning from de-asserted to asserted.

Once the CMD bit is asserted, the connected port will either assert ACK after accepting and executing the command or assert NACK if the command cannot be executed. The assertion of ACK or NACK shall occur no more than 250usec after the assertion of CMD. ACK and NACK shall never be asserted at the same time.

Once ACK or NACK is asserted in a CS field received by the port issuing the command, the CMD bit is de-asserted.

ACK or NACK, whichever is asserted, shall be de-asserted within 250usec of receipt of a CS field with the CMD bit deasserted.

If, for any reason, the connected port fails to assert ACK or NACK the assertion of CMD within the timeout period configured in Port n Link Timers Control CSRs Emphasis Command Timeout field, CMD shall be deasserted. Once deasserted, CMD shall remain deasserted for at least the timeout period configured in the Emphasis Command Timeout field before being reasserted..

A CS field command to increase the emphasis of tap(n) by one step shall cause the tap(n) coefficient to be made more negative by one step. A command to decrease the emphasis of tap(n) by one step shall cause the tap(n) coefficient to be made more positive by one step. The transmit emphasis step sizes are implementation dependant. The adjustment of the tap(n) coefficient value may result in the coefficient value of one or more of the other taps to be modified by the transmitter to maintain certain specifications such as the minimum or maximum transmit amplitude.

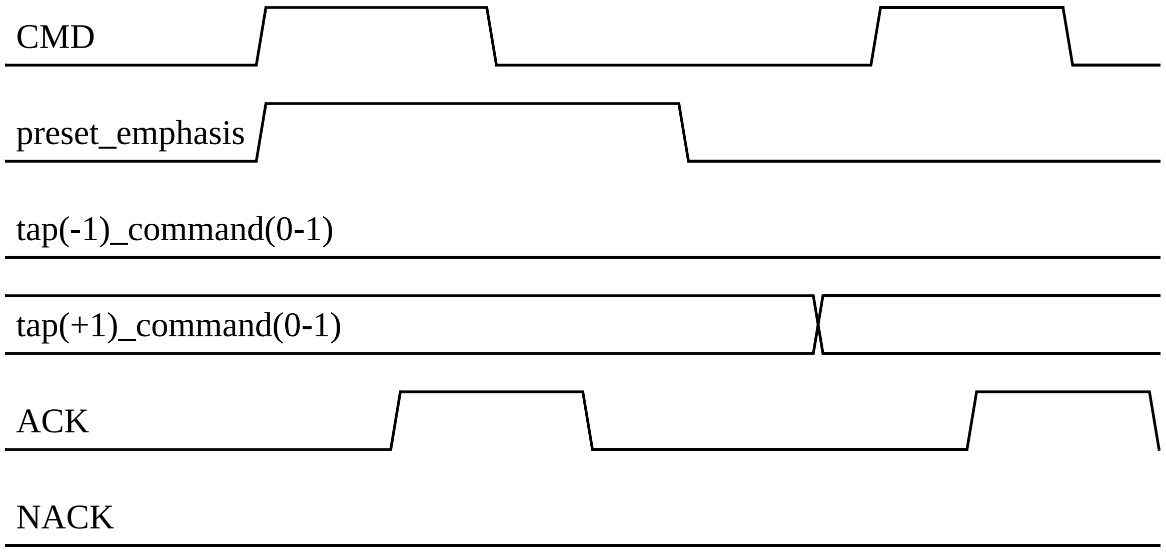


Figure 4-7. Example of CS Field CMD, ACK, NACK Handshake

4.7.5 Idle Sequence Selection

LP-Serial Baud Rate Class 2 links shall always use the IDLE2 sequence. LP-Serial Baud Rate Class 1 links shall support use of the IDLE1 sequence and may support use of the IDLE2 sequence.

If a LP-Serial port is operating at Baud Rate Class 1 and both ports on the link support both IDLE1 and IDLE2, the port shall determine which idle sequence to use on the link by using the following algorithm during the port initialization process.

> If a LP-Serial port is operating at Baud Rate Class 1, supports the IDLE2 sequence and its configuration allows it to use the IDLE2 sequence, the port shall transmit the IDLE2 sequence when it enters the SEEK state of the port initialization process. (The port initialization process is specified in Section 4.12.) Otherwise, a LP-Serial port operating at Baud Rate Class 1 shall transmit the IDLE1 sequence when entering the SEEK state and shall use the IDLE1 on the link until the port reenters the SEEK state.
>
> A LP-Serial port transmitting the IDLE2 sequence shall monitor the idle sequence it is receiving from the connected port. The port shall determine the idle sequence being received from the connected port using a lane for which lane_sync is asserted. The techniques and algorithms used by a port supporting both IDLE1 and IDLE2 to determine which idle sequence it is receiving are implementation specific and outside the scope of this specification.
>
> If the LP-Serial port that is transmitting the IDLE2 sequence receives IDLE2 from the connected port, IDLE2 shall be the idle sequence used on the link until the port reenters the SEEK state. If the port receives IDLE1 from the connected port, the port shall switch to transmitting IDLE1 and IDLE1 shall be the idle sequence used on the link until the port reenters the SEEK state.

There are restrictions on the type of equalizers and, if any of the equalization is adaptive, on the adaptive equalizer training algorithms that can be used by ports operating at Baud Rate Class 1. These restrictions are specified in Section 10.2, "Equalization" and Section 11.2, "Equalization".

4.8 Scrambling

Scrambling smooths the spectrum of a port’s transmit signal and reduces the spectrum’s peak values. This is most important when long strings of the same character or of a repeating character sequence are transmitted. The result is a reduction in the amount of electromagnetic interference (EMI) generated by the link and easier design of adaptive equalizer training algorithms. Scrambling of packet and control symbol data characters is used only on links operating with idle sequence 2 (IDLE2). It is not used on links operating with idle sequence 1 (IDLE1) for backwards compatibility with early revisions of this specification.

4.8.1 Scrambling Rules

The use of control symbol and packet data character scrambling on a LP-Serial link is determined by the idle sequence being used on the link.

> If the idle sequence selection process specified in Section 4.7.5 has selected idle sequence 1 (IDLE1) for use on the link, no characters shall be scrambled before transmission on the link.
>
> If the idle sequence selection process has selected idle sequence 2 (IDLE2), control symbol and packet data characters shall be scrambled by the transmitter before transmission on the link and descrambled in the receiver upon reception. (The per lane scramblers are also used to generate the pseudo-random data characters in the IDLE2 random data field as specified in Section 4.7.4.1.1). Special characters, CS field marker data characters, and CS field data characters shall not be scrambled before transmission.

Scrambling and descrambling of control symbol and packet data characters can be disabled for test purposes by setting the Data scrambling disable bit in the Port *n* Control 2 CSR. Scrambling and descrambling of control symbol and packet data characters shall not be disabled for normal link operation. Setting the Data scrambling disable bit does not disable the use of the lane scramblers for the generation of pseudo-random data characters for the IDLE2 random data field. (See Section 7.6.9, "Port n Control 2 CSRs").

Scrambling and descrambling shall be done at the lane level. Nx ports shall have a transmit scrambling and receive descrambling function for each of the N lanes. In the transmitter, scrambling shall occur before 8b/10b encoding, and if the port is operating in Nx mode, after lane striping. In the receiver, descrambling shall occur after 8b/10b decoding, and if the port is operating in Nx mode, before lane destriping.

The polynomial $x^{17}+x^{8}+1$ shall be used to generate the pseudo-random sequences that are used for scrambling and descrambling. This polynomial is not primitive, but when the sequence generator is initialized to all 1s or other appropriate values, the polynomial produces a sequence with a repeat length of 35,805 bits. The bit serial output of the pseudo-random sequence generator shall be taken from the output of the register holding x^{17}. The pseudo random sequence generator is shown in Figure 4-8.

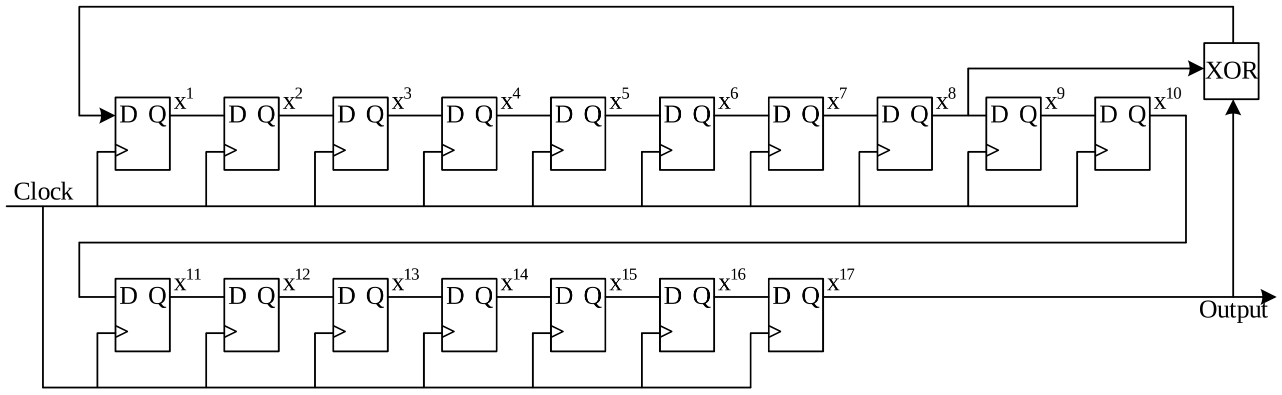


Figure 4-8. Scrambling Sequence Generator

Control symbol and packet data characters shall be scrambled and descrambled by XORing the bits of each character with the output of the pseudo-random sequence generator. The bits of each data character are scrambled/descrambled in order of decreasing significance. The most significant bit (bit 0) is scrambled/descrambled first, the least significant bit (bit 7) is scrambled/descrambled last.

The transmitter and receiver scrambling sequence generators shall step during all characters except R special characters. This is to prevent loss of sync between transmit and receive scramblers when an /R/ or ||R|| is added or removed by a retimer.

To minimize any correlation between lanes when a port is transmitting on multiple lanes, the scrambling sequence applied to a given output lane of the port shall be offset from the scrambling sequence applied to any other output lane of the port by at least 64 bits. If separate scrambling sequence generators are used for each lane, the offset requirement can be achieved by initializing the scramblers to the values specified in Table 4-9, which provide an offset of 64.

Table 4-9. Scrambler Initialization Values

Lane	Initialization value [x^1-x^{17}]
0	0b1111 1111 1111 1111 1
1	0b1111 1111 0000 0110 1
2	0b0000 0000 1000 0110 1
3	0b0000 0110 0111 1010 0
4	0b1000 0000 1011 1001 0
5	0b1111 1010 1000 0111 0
6	0b0100 0011 1001 1011 1
7	0b1100 0100 1010 0101 0
8	0b0101 1111 0100 1001 0
9	0b1111 1010 0111 1001 1

Table 4-9. Scrambler Initialization Values

Lane	Initialization value $[x^1\text{-}x^{17}]$
10	0b1011 0011 0111 0010 1
11	0b1100 1010 1011 0011 0
12	0b1011 1000 0101 0011 1
13	0b0000 1011 0110 1111 0
14	0b0101 1000 1001 1010 1
15	0b0011 0111 1010 1000 1

4.8.2 Descrambler Synchronization

Since the pseudo-random data characters of the random data field of the idle sequence 2 idle frame are generated by scrambling D0.0 characters, the pseudo-random characters of the random data field contain the pseudo-random sequence used by the transmitter to scramble control symbol and packet data characters. A sequence of at least four (4) contiguous pseudo-random data characters immediately follow each M special character in the random data field.

Each lane descrambler shall synchronize itself to the scrambled data stream it is receiving by using the scrambling sequence extracted from the pseudo-random data characters received by the lane to re-initialize the state of the descrambler.

After a lane descrambler has been re-initialized, the next two descrambler sync tests, which are defined in Section 4.8.3, shall be used to verify descrambler synchronization. If the result of both lane descrambler sync tests is “pass”, the descrambler shall be determined to be “in sync”. Otherwise, the lane descrambler shall be determined to be “out of sync” and the resynchronization process shall be repeated.

To ensure that a port that may have lost descrambler sync is able to recover descrambler sync before it is sent a link maintenance protocol link-request control symbol, a LP-Serial port that is operating with IDLE2 shall transmit a SYNC sequence (described below) before transmitting any link-request control symbol. The SYNC sequence shall be transmitted in parallel on each of the N active lanes of a link operating in Nx mode and shall immediately precede the link-request control symbol. If the link is operating in 1x mode, the last character of the SYNC sequence is immediately followed by the first character of the link-request. If the link is operating in Nx mode, the last column of the SYNC sequence is immediately followed by the column containing the first characters of the link-request.

The SYNC sequence shall be comprised of four contiguous repetitions of a five character sequence that begins with a M special character immediately followed by 4 pseudo-random data characters, i.e. the SYNC sequence is MDDDD MDDDD MDDDD MDDDD. The pseudo-random data characters shall be generated in the same way as the pseudo-random data characters in the random data field of the IDLE2 idle frame are generated. The SYNC sequence will appear as four repetitions of ||M||D||D||D||D|| on a link operating in Nx mode.

4.8.3 Descrambler Synchronization Verification

Each active lane of a LP-Serial port that is descrambling received control symbol and packet data characters shall, with the one exception stated below, perform a descrambler synchronization state check (descrambler sync check) whenever a descrambler sync check trigger event is detected in the received character stream of the lane.

A descrambler sync check trigger event is defined as the occurrence of one of the following character sequences in the received character stream of an active lane.

1. A single K, M or R special character that is not part of a contiguous sequence of K, M and/or R special characters.
2. A contiguous sequence of K and/or R special characters possibly followed by a M special character.

The descrambler sync check shall consist of inspecting the descrambled values of the four contiguous characters following the trigger sequence. These four characters are designated the descrambler sync “check field”. The characters comprising the check field shall be determined as follows.

> The check field for the first type of trigger event shall be the four characters immediately following the K, M or R special character.
>
> The check field for the second type of trigger event that does not end with a M special character shall be the four characters immediately following the contiguous sequence of K and/or R special characters.
>
> The check field for the second type of trigger event that ends with a M special character shall be the four characters immediately following the M special character.

When the descrambler is in sync and in the absence of transmission errors, the “check field” will contain four data characters that are all D0.0s after descrambling.

The exception to the rule stated above that each descrambler sync check trigger sequence shall cause the receiving lane to execute a descrambler sync check is when the descrambler check trigger sequence begins in the four character check field of a previous trigger sequence. When this occurs, the trigger sequence shall not trigger a descrambler sync check. For example, the RM in the sequence KRXRMDDDD, where X is neither a K nor R, shall not trigger a descrambler sync check as it begins in the four character check field used by the descrambler sync check triggered by the KR sequence.

If the descrambled value of each of the four characters in a check field is D0.0, the result of the descrambler sync test shall be “pass”. Otherwise, the result of the descrambler sync test shall be “fail” and the descrambler shall be determined to be “out of sync”.

A sync test can fail because of either a loss of descrambler sync or a data transmission error(s) in either the sync trigger sequence or the check field.

If a descrambler sync test fails, the port shall immediately enter the Input Error-stopped state if it is not already in that state and resynchronize the descrambler. All control symbols and packet received while a lane descrambler is out of sync shall be ignored and discarded. The cause field in the packet-not-accepted control symbol issued by the port on entering the Input Error-stopped state due to a sync check failure shall indicate “loss of descrambler sync”.

4.9 1x Mode Transmission Rules

4.9.1 1x Ports

A 1x LP-Serial port shall 8b/10b encode and transmit the character stream of delimited control symbols and packets received from the upper layers in the order the characters were received from the upper layers. When neither control symbols nor packets are available from the upper layers for transmission, an idle sequence shall be fed to the input of the 8b/10b encoder for encoding and transmission.

On reception, the code-group stream is 8b/10b decoded and the resulting character stream of error free delimited control symbols and packets shall be passed to the upper layers in the order the characters were received from the link.

If the link is operating with idle sequence 2, control symbol and packet data characters shall be scrambled before transmission and descrambled after reception as specified in Section 4.8.

Figure 4-9 shows the encoding and transmission order for a Control Symbol 24 transmitted over a LP-Serial link operating in 1x mode.

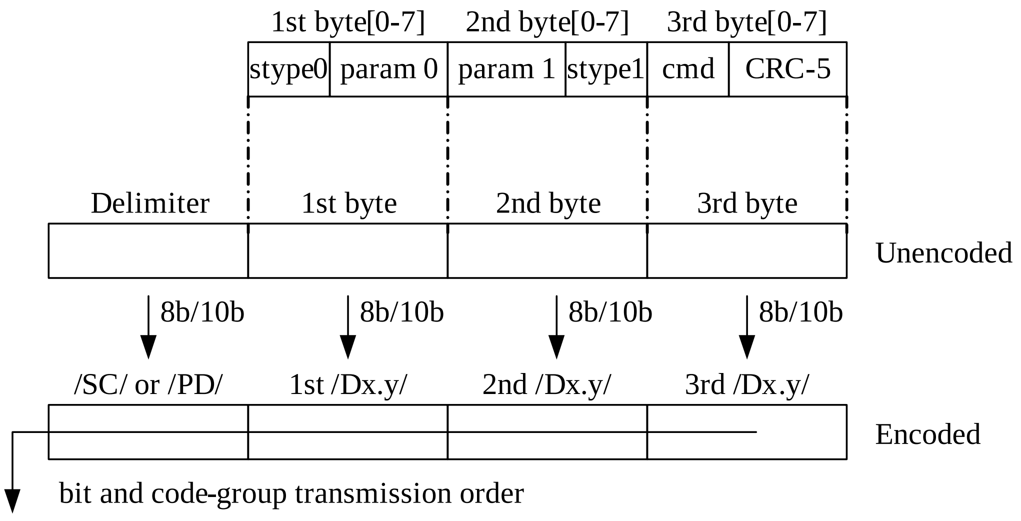


Figure 4-9. 1x Mode Control Symbol 24 Encoding and Transmission Order

Figure 4-10 shows the encoding and transmission order for a packet transmitted over a 1x LP-Serial link.

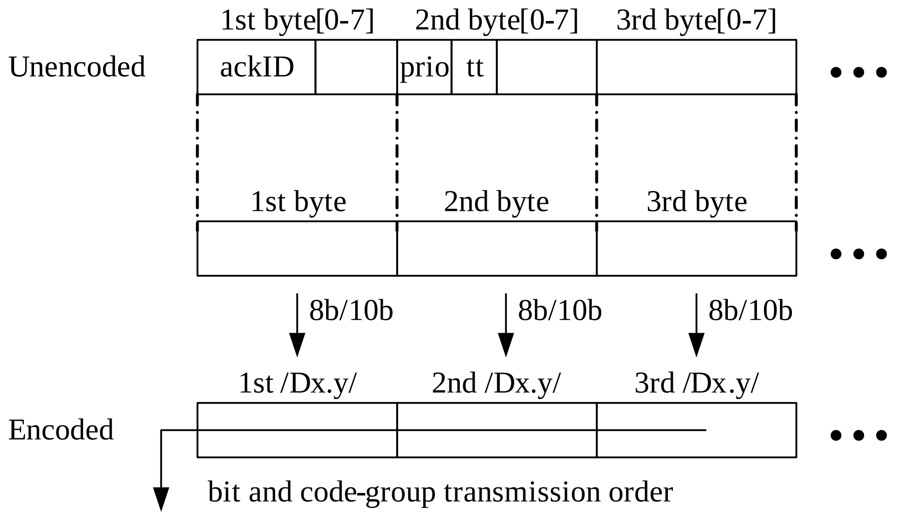


Figure 4-10. 1x Mode Packet Encoding and Transmission Order

Figure 4-11 shows an example of idle sequence 1, Control Symbol 24 and packet transmission on a 1x LP-Serial link.

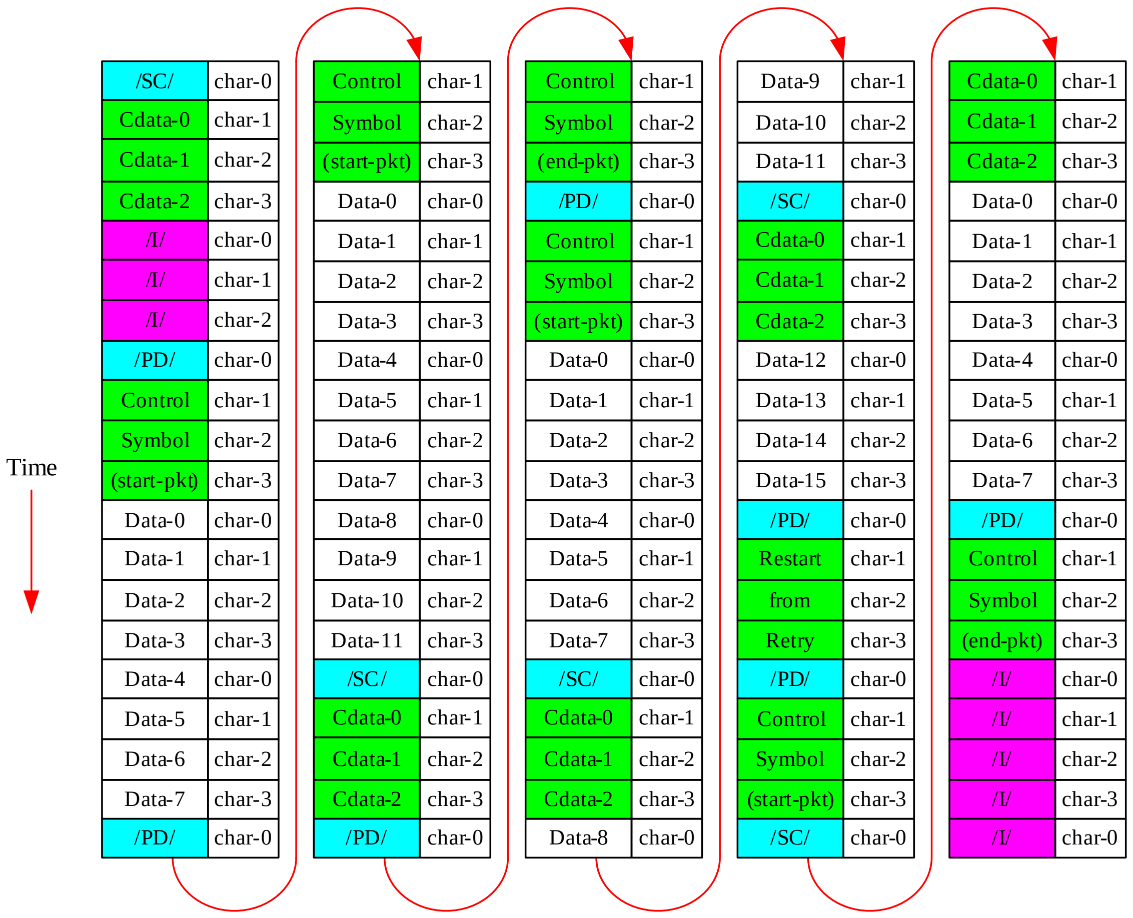


Figure 4-11. 1x Typical Data Flow with Control Symbol 24

4.9.2 Nx Ports Operating in 1x Mode

When a Nx port is operating in 1x mode, the character stream of delimited control symbols and packets received from the upper layers shall be fed in parallel to both lanes 0 and R for encoding and transmission in the order the characters were received from the upper layers. (The character stream is not striped across the lanes before encoding as is done when operating in Nx mode.) When neither delimited control symbols nor packets are available from the upper layers for transmission, an idle sequence shall be fed in parallel to both lane 0 and lane R for 8b/10b encoding and transmission on lanes 0 and R.

On reception, the code-group stream from either lane 0 or R shall be selected according to the state of the 1x/Nx_Initialization state machine (Section 4.12.4.5), decoded and the error free delimited control symbols and packets passed to the upper layers.

When a port that optionally supports and is enabled for both 2x mode and a wider Nx mode is operating in 1x, the port shall support both lanes 1 and 2 as redundancy lanes. The port shall transmit the 1x mode data stream on lanes 0, 1 and 2 and attempt to receive 1x mode data stream on lanes 0, 1 and 2. The port shall select between using the data received on lane 0 or the data received on the redundancy lane which may be either lane 1 or lane 2 depending on the connected port. Unless forced to use the redundancy lane, the port shall use the data stream received on lane 0 if it is available. The 1x/2x/Nx_Initialization state machine specified in Section 4.12.4.8.1 shall be used for a port supporting both 2x and a wider Nx mode to comply with the above requirements.

If the link is operating with idle sequence 2, control symbol and packet data characters shall be scrambled before transmission and descrambled after reception as specified in Section 4.8.

Once a Nx port is initialized to a 1x mode, the port may elect to disable the output driver of the lane which was not selected for reception by the initialization state machine of the connected port. Since the ports connected by the link may not be receiving on the same lane (one port could be receiving on lane 0 and the other port receiving on lane R), the connected port must be interrogated to determine which lane can be output disabled. It is recommended that the mechanism for disabling the output driver be under software control.

4.10 Nx Link Striping and Transmission Rules

A LP-Serial port operating in Nx mode shall stripe the character stream of delimited control symbols and packets received from the upper layers across the N active output lanes in the order the characters were received from the upper layers. Each lane shall then 8b/10b encode and transmit the characters assigned to it. When neither control symbols nor packets are available from the upper layers for transmission, an idle sequence shall be fed to each of the N lanes for 8b/10b encoding and transmission.

Packets and delimited control symbols shall be striped across the N active lanes beginning with lane 0. The first character of each packet, or delimited control symbol, shall be placed in lane K where K modulo 4 = 0. The second character shall be placed in lane (K + 1), and the nth character shall be placed in lane (K + (n - 1)) which wraps around to lane 0 when (K + (n - 1)) modulo N = 0.

The lengths of control symbols and packets in the LP-Serial Physical Layer are positive integer multiples of 4 characters. As a result, when N, the width of the link, is greater than 4, occasions will occur when there are not enough packets and/or control symbols available for transmission to fill a column. For example, lanes 0-3 of a link operating in 8x mode contain a delimited Control Symbol 24 or the last 4 characters of a delimited Control Symbol 48, but there is nothing available to put in lanes 4-7. When this occurs, all remaining characters in the column shall be filled (padded) with pseudo-random data characters. The first pseudo-random data pad character shall occur in a lane whose lane_number modulo 4 = 0. The number of pseudo-random data pad characters in a column shall be a positive integer multiple of 4. If the link is operating with idle sequence 2, the pseudo-random data characters shall be generated by using the lane scramblers to scramble D0.0 characters. With the exception stated in Section 6.6.1.2, padding characters shall not be inserted between packet delimiting control symbols and the packet(s) they delimit.

After striping, each of the N streams of characters shall be independently 8b/10b encoded and transmitted.

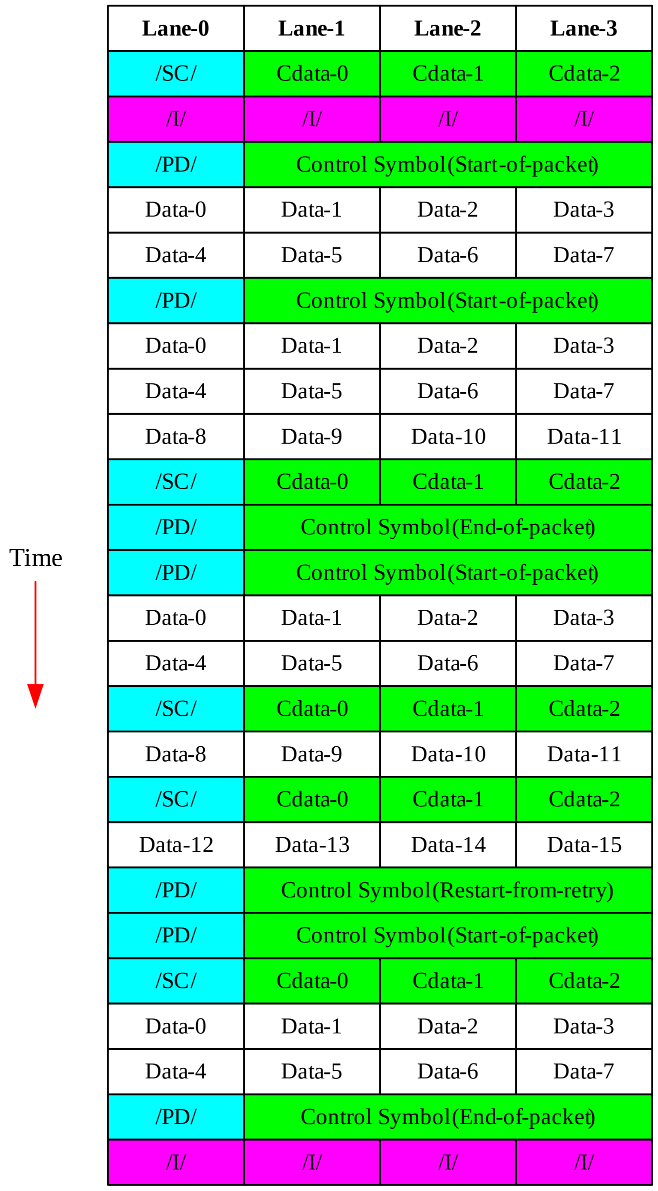
On reception, each lane shall be 8b/10b decoded.

If the link is operating with idle sequence 2, control symbol and packet data characters shall be scrambled before transmission and descrambled after reception as specified in Section 4.8.

After decoding, the N lanes shall be aligned. The ||A|| columns transmitted as part of an idle sequence provide the information needed to perform alignment. After alignment, the columns are destriped into a single character stream and passed to the upper layers.

The lane alignment process eliminates the skew between lanes so that after destriping, the ordering of characters in the received character stream is the same as the ordering of characters before striping and transmission. Since the minimum number of non ||A|| columns between ||A|| columns is 16, the maximum lane skew that can be unambiguously corrected is the time it takes to transmit 7 code-groups on a lane.

Figure 4-12 shows an example of Idle Sequence 1, Control Symbol 24 and packet transmission on a 4x link.



Lane-0	Lane-1	Lane-2	Lane-3
/SC/	Cdata-0	Cdata-1	Cdata-2
/I/	/I/	/I/	/I/
/PD/	Control Symbol(Start-of-packet)		
Data-0	Data-1	Data-2	Data-3
Data-4	Data-5	Data-6	Data-7
/PD/	Control Symbol(Start-of-packet)		
Data-0	Data-1	Data-2	Data-3
Data-4	Data-5	Data-6	Data-7
Data-8	Data-9	Data-10	Data-11
/SC/	Cdata-0	Cdata-1	Cdata-2
/PD/	Control Symbol(End-of-packet)		
/PD/	Control Symbol(Start-of-packet)		
Data-0	Data-1	Data-2	Data-3
Data-4	Data-5	Data-6	Data-7
/SC/	Cdata-0	Cdata-1	Cdata-2
Data-8	Data-9	Data-10	Data-11
/SC/	Cdata-0	Cdata-1	Cdata-2
Data-12	Data-13	Data-14	Data-15
/PD/	Control Symbol(Restart-from-retry)		
/PD/	Control Symbol(Start-of-packet)		
/SC/	Cdata-0	Cdata-1	Cdata-2
Data-0	Data-1	Data-2	Data-3
Data-4	Data-5	Data-6	Data-7
/PD/	Control Symbol(End-of-packet)		
/I/	/I/	/I/	/I/

Figure 4-12. Typical 4x Data Flow with Control Symbol 24

4.11 Retimers and Repeaters

The LP-Serial Specification allows “retimers” and “repeaters”. Retimers amplify a weakened signal, but do not transfer jitter to the next segment because they use a local transmit clock. Repeaters also amplify a weakened signal, but transfer jitter to the next segment because they use a transmit clock derived from the received data stream. Retimers allow greater distances between end points at the cost of additional latency. Repeaters support less distance between end points than retimers and only add a small amount of latency.

4.11.1 Retimers

A retimer shall comply with all applicable AC specifications found in Chapter 9, "Common Electrical Specifications for less than 6.5 Gbaud LP-Serial Links", Chapter 10, "1.25 Gbaud, 2.5 Gbaud, and 3.125 Gbaud LP-Serial Links", and Chapter 11, "5 Gbaud and 6.25 Gbaud LP-Serial Links". This includes resetting the jitter budget thus extending the transmission distance for the link. The retimer repeats the received code-groups after performing code-group synchronization and serializes the bitstream again on transmission, based on a local clock reference. Up to two retimers are allowed between two end nodes.

A retimer is not RapidIO protocol-aware or addressable in any way. The only awareness a retimer has is to the synchronization on the /K/ code-group and the function of /R/ insertion and removal. A retimer may insert up to one /R/ code-group immediately following a /K/ code-group, or remove one /R/ code-group that immediately follows a /K/ code-group. Since the /R/ code-group is disparity neutral, its insertion or deletion does not affect the running disparity.

A N-lane retimer must perform lane synchronization and deskew, in exactly the same way a RapidIO device implementing the LP-Serial Physical Layer does when synchronizing inputs during initialization and startup. A Nx mode retimer will synchronize and align all lanes that are driven to it. Therefore, such a retimer allows for the degradation of an input Nx link to a 1x link on either lane 0 or R. If any link drops out, the retimer must merely continue to pass the active links, monitoring for the compensation sequence and otherwise passing through whatever code-groups appear on its inputs. A retimer may optionally not drive any outputs whose corresponding inputs are not active.

Any insertion or removal of a /R/ code-groups in a N-lane retimer must be done on a full column. A retimer may retime links operating at the same width only (i.e. cannot connect a link operating at 1x to a link operating at Nx). A retimer may connect a 1x link to a Nx link that is operating in 1x mode. Retimers perform clock tolerance compensation between the receive and transmit clock. The transmit clock is usually derived from a local reference.

Retimers do not check for code violations. Code-groups received on one port are transmitted on the other regardless of code violations or running disparity errors.

4.11.2 Repeaters

A repeater is used to amplify the signal, but does not retime the signal, and therefore can add additional jitter to the signal. It does not compensate for clock rate variation. The repeater repeats the received code-groups as the bits are received by sampling the incoming bits with a clock derived from the bit stream, and then retransmitting them based on that clock. Repeaters may be used with Nx links but lane-to-lane skew may be amplified. Repeaters do not interpret or alter the bit stream in any way.

4.12 Port Initialization

This section specifies the port initialization process. The process includes detecting the presence of a partner at the other end of the link (a link partner), establishing bit synchronization and code-group boundary alignment and if present, adjusting any adaptive equalizers. The process also includes determining if the connected port supports an Nx mode in addition to 1x mode and selecting 1x or Nx mode operation, then, if 1x mode is selected, selecting lane 0 or lane R (the redundancy lane, lane 1 for 2x ports and lane 2 for 4x, 8x or 16x ports) for link reception.

Port initialization may optionally include baud rate discovery.

The initialization process is controlled by several state machines. The number and type of state machines depends on whether the port supports only 1x mode (a 1x port) or supports both 1x and one or more Nx modes (a 1x/Nx port). In either case, there is a primary state machine and one or more secondary state machines. The use of multiple state machines results in a simpler overall design. As might be expected, the initialization process for a 1x port is simpler than and is a subset of the initialization process for a 1x/Nx port.

The port initialization process supports an optional test mode that allows ports that support more than one multi-lane mode of operation to enable and monitor the operation of the inactive lanes when the port is operating at less than maximum width. The performance of inactive lanes can be monitored only if the inactive lanes are connected to and supported by the connected port and the test mode is implemented and enabled in both ports. The test mode is enabled with the “Enable inactive lanes” bit defined in Section 7.6.9. The initiation, implementation and interpretation of tests conducted using this test mode is outside of this specification.

The initialization process for 1x, 1x/Nx ports, and ports supporting 1x mode and multiple Nx modes is both described in text and specified with state machine diagrams. **In the case of conflict between the text and a state machine diagram, the state machine diagram takes precedence.**

4.12.1 1x Mode Initialization

The initialization process for ports that support only 1x mode shall be controlled by two state machines, 1x_Initialization and Lane_Synchronization. 1x_Initialization is the primary state machine and Lane_Synchronization is the secondary state machine. The operation of these state machines is described and specified in Section 4.12.4.4 and Section 4.12.4.2 respectively.

4.12.2 1x/Nx Mode Initialization

The initialization process for ports that support both 1x and a Nx mode is controlled by a primary state machine and four or more secondary state machines. The primary state machine is the 1x/Nx_Initialization state machine. Lane_Synchronization[0] through Lane_Synchronization[N-1] (one for each of the N lanes), Lane_Alignment (one for each supported Nx mode) and 1x/2x_Mode_Detect (used only in the 1x/2x_Initialization state machine) are the secondary state machines. The operation of the secondary state machines is described and specified in Section 4.12.4.2 through Section 4.12.4.4 respectively.

The 1x/Nx_Initialization state machine provides a degree of LP-Serial link width auto-negotiation. The goal of the auto-negotiation is to ensure that any connected combination of 1x, 1x/2x, 1x/4x, 1x/8x or 1x/16x LP-Serial ports that are configured in some manner to operate at the same baud rate will automatically find a link width over which they can communicate. For example if a 1x/4x port is connected to a 1x/8x port, they will auto-negotiate to operate in 1x mode. If however the 1x/8x port optionally also supports 4x mode (making it a 1x/4x/8x port) and its 1x/Nx_Initialization state machine has been modified as shown in Figure 4-22 to be a 1x/4x/8x_Initialization state machine, then the ports will auto-negotiate to operate in 4x mode.

In most configurations, the auto-negotiation also ensures that a pair of connected multi-lane LP-Serial ports configured in some manner to operate at the same baud rate will find a link width over which they can communicate in the presence of a lane failure. For example, if two 1x/4x ports are connected and lane 0 is broken in one direction, the ports will auto-negotiate to operate in 1x mode using lane 0 in the direction that lane 0 is operational and lane 2 in the direction that lane 0 is broken. This feature works only for pairs of ports that support the same redundancy lane. It does not work when a 1x/2x port is connected to a 1x/4x or wider port.

4.12.3 Baud Rate Discovery

Baud rate discovery is optional. If implemented, baud rate discovery occurs during the SEEK state of the 1x_Initialization and 1x/Nx_Initialization state machines. Ports that implement baud rate discovery shall use the following algorithm.

1. When the port enters the SEEK state, it begins transmitting an idle sequence on lane 0 and, if the port supports a Nx mode, on lane R, the 1x mode redundancy lane. The idle sequence shall be transmitted at the highest lane baud rate that is supported by the port and that is enabled for use.
2. The port shall then look for an inbound signal on lane 0 or lane R of the link from a connected port. The method of detecting the presence of an inbound signal from a connected port is implementation specific and outside the scope of this specification.
3. Once an inbound signal is detected, the port shall determine the baud rate of the signal. The method of detecting the baud rate of the signal is implementation specific and outside the scope of this specification.
4. If the baud rate of the inbound signal is the same as the baud rate at which the port is transmitting, the link shall operate at that per lane baud rate until the port reenters the SEEK state and the baud rate discovery process is complete.

5. If the baud rate on the inbound signal is less than the baud rate of the idle sequence being transmitted by the port, the port shall reduce the baud rate at which it is transmitting to the next lowest baud rate that it supports and that is enabled for use and go to step 2.
6. If the baud rate on the inbound signal is greater than the baud rate of the idle sequence being transmitted by the port, the port shall continue transmitting at the current baud rate go to step 2.

An informational state diagram for the Baudrate_Discovery state machine is shown in Figure 4-13.

The techniques and algorithms used to compare the baud rates of the signals being transmitted and received are implementation specific and beyond the scope of this specification.

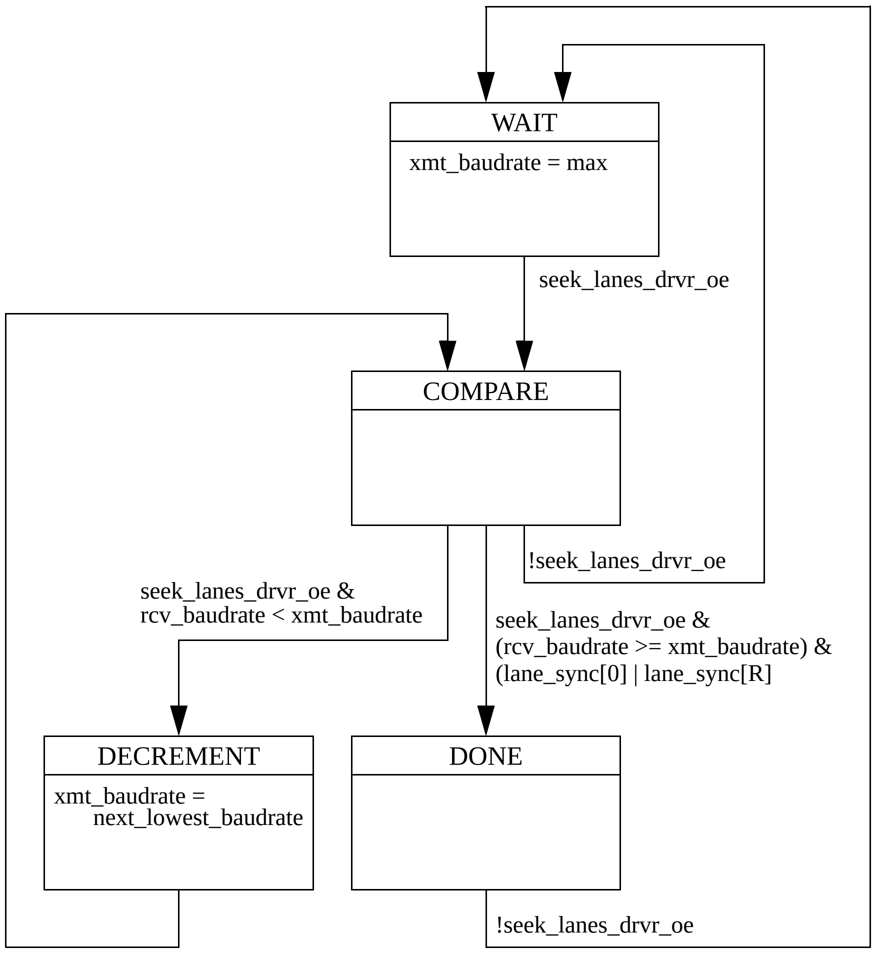


Figure 4-13. Baudrate_Discovery state machine (Informational)

4.12.4 State Machines

4.12.4.1 State Machine Conventions, Functions and Variables

4.12.4.1.1 State Machine Conventions

The conventions used in state machine specification are as follows.

A state machine state is persistent until an exit condition occurs.

A state machine variable that is listed in the body of a state but is not part of an assignment statement is asserted for the duration of that state only.

A state machine variable that is assigned a value in the body of a state retains that value until assigned a new value in another state.

A state machine function that is listed in the body of a state is executed once during the state.

A state machine variable is asserted when its value is 1 and de-asserted when it value is 0.

Except when otherwise directed by parentheses, the order of precedence of logic operations when evaluating a logic expression is, in order of decreasing precedence, negation/compliment (!) followed by intersection (&) and union (|).

Logic expressions within paired parentheses are evaluated before the rest of a logic expression is evaluated with the operations within the innermost pair of parentheses evaluated first.

4.12.4.1.2 State Machine Functions

The functions used in the state machines are defined as follows.

change( )

Asserted when the variable on which it operates changes state.

next_code_group( )

Gets the next 10 bit code-group for the lane when it becomes available.

next_Ncolumn( )

Gets the next column of N code-groups or characters, as appropriate, from lanes 0 to N-1 when it becomes available.

4.12.4.1.3 State Machine Variables

The variables used in the state machines are defined as follows.

1x_mode_delimiter

Asserted when a column of two characters from lanes 0 and 1 contains two SC or two PD special characters. Otherwise de-asserted.

1x_mode_detected

Asserted by the 1x/2x_Mode_Detect state machine when it determines that the link receiver input signals on lanes 0 and 1 are in 1x mode. Otherwise, de-asserted.

2x_mode_delimiter

Asserted when a column of two characters from lanes 0 and 1 contains one SC or PD special character and one data character. Otherwise de-asserted.

||A||

Asserted when the current column contains all /A/s. Otherwise de-asserted.

Acounter

A counter used in the Lane Alignment state machine to count received alignment columns (||A||s).

align_error

Asserted when the current column contains at least one /A/, but not all /A/s. Otherwise, de-asserted.

/COMMA/

If Idle Sequence 1 is being used on the link to which the port is connected, asserted when the current code-group is /K28.5/. Otherwise, de-asserted.

If Idle Sequence 2 is being used on the link to which the port is connected, asserted when the current code-group is either /K28.1/ or /K28.5/. Otherwise, de-asserted.

Dcounter

A 2-bit synchronous saturating up/down counter with the behavior specified in Table 4-10. The counter is used in the 1x/2x_Mode_Detect state machine.

Table 4-10. Dcounter Definition

Counter Value	(count_up,count_down)			
	0,0	0,1	1,0	1,1
0x0	0x0	0x0	0x1	0x0
0x1	0x1	0x0	0x2	0x1

Table 4-10. Dcounter Definition

Counter Value	(count_up,count_down)			
	0,0	0,1	1,0	1,1
0x2	0x2	0x1	0x3	0x2
0x3	0x3	0x2	0x3	0x3

disc_tmr_done (discovery timer done)

Asserted when disc_tmr_en has been continuously asserted for 28 +/- 4 msec and the state machine is in the DISCOVERY or a RECOVERY state. The assertion of disc_tmr_done causes disc_tmr_en to be de-asserted. When the state machine is in a state other than the DISCOVERY or a RECOVERY state, disc_tmr_done is de-asserted.

disc_tmr_en (discovery timer enable)

When asserted, the discovery timer (disc_tmr) runs. When de-asserted, the discovery timer is reset to and maintains its initial value.

force_1x_mode

Asserted when all Nx (multi-lane) modes are disabled. When asserted, forces the 1x/Nx Initialization state machine to use 1x mode.

force_laneR

When force_1x_mode is asserted, force_laneR controls whether lane 0 or lane R, the redundancy lane, is preferred for 1x mode reception. If force_laneR is asserted, lane R is the preferred lane. If force_laneR is deasserted, lane 0 is the preferred lane. If the preferred lane is functional, it is selected by the port initialization state machine for 1x mode reception. If the preferred lane is not functional, the non-preferred lane, if functional, is selected for 1x mode reception.

If force_1x_mode is not asserted, the state of force_laneR has no effect on the initialization state machine.

force_reinit

When asserted, forces the port Initialization state machine to re-initialize. The signal is set under software control and is cleared by the Initialization state machine.

Icounter

Counter used in the Lane_Synchronization state machine to count INVALID received code-groups. There is one Icounter for each lane in a Nx mode receiver.

idle_selected

When asserted, indicates that the IDLE sequence for use on the link has been selected by the Idle Sequence Selection process specified in Section 4.7.5.

If the port supports only one IDLE sequence at the current baud rate, the bit is always asserted.

If the port supports multiple IDLE sequences at the current baud rate, the bit is de-asserted when the Initialization state machine is in the SILENT state and is otherwise controlled by the Idle Sequence Selection process. The Idle Sequence Selection process runs when the Initialization state machine is in the SEEK state and lane_sync has been asserted for lane 0, 1 and/or 2. The bit is asserted when the Idle Sequence Selection process completes.

/INVALID/

When asserted, /INVALID/ indicates that the current code-group is an invalid code-group.

Kcounter

Counter used in the Lane_Synchronization state machine to count received code-groups that contain a comma pattern. There is one Kcounter for each lane in a Nx mode receiver.

lane_ready[n]

lane_ready[n] = lane_sync[n] & lane_trained[n]

lane_sync

Asserted by the Lane_Synchronization state machine when it determines that the lane it is monitoring is in bit synchronization and code-group boundary alignment. Otherwise de-asserted.

lane_sync[n]

The lane_sync signal for lane n.

lane_trained[n]

De-asserted when the local lane[n] receiver controls adaptive equalization in the receiver and/or the connected lane[n] transmitter and the training of the equalization in either the lane[n] receiver or the connected lane[n] transmitter has not been completed. Otherwise, asserted.

lane0_drvr_oe

When asserted, the output driver for lanes 0 is enabled

lanes01_drvr_oe

When asserted, the output drivers for lanes 0 and 1 are enabled

lanes02_drvr_oe

When asserted, the output drivers for lanes 0 and 2 are enabled lanes13_drvr_oe

When asserted, the output drivers for lanes 1 and 3 are enabled

Mcounter

Mcounter is used in the Lane_Alignment state machine to count columns received that contain at least one /A/, but not all /A/s.

N_lanes_aligned

Asserted by the Lane_Alignment state machine when it determines that lanes 0 through N-1 are in sync and aligned.

N_lanes_drvr_oe

The output enable for the lanes 0 through N - 1.

N_lanes_ready

N_lanes_ready = N_lanes_aligned & lane_ready[0] & ... & lane_ready[N-1]

N_lane_sync

Indicates when lanes 0 through N-1 of a receiver operating in Nx mode are in bit synchronization and code-group boundary alignment.

N_lane_sync = lane_sync[0] & ... & lane_sync[N-1]

Nx_mode

Asserted when the port is initialized and operating in Nx mode

port_initialized

When asserted, port_initialized indicates that the port is initialized. Otherwise the port is not initialized. The state of port_initialized affects what the port may transmit on and accept from the link.

receive_lane1

In a 2x port that is initialized and is operating in 1x mode (2x_mode de-asserted), receive_lane1 indicates which lane the port has selected for input. When asserted, the port input is taken from lane 1. When de-asserted the port input is taken from lane 0. When the port is operating in 2x mode (2x_mode asserted), receive_lane1 is undefined and shall be ignored.

receive_lane2

In a Nx port that is initialized and is operating in 1x mode (Nx_mode de-asserted for all N > 2), receive_lane2 indicates which lane the port has selected for input. When asserted, the port input is taken from lane 2. When de-asserted the port input is taken from lane 0. When the port is operating in Nx mode (some Nx_mode asserted), receive_lane2 is undefined and shall be ignored.

seek_lanes_drvr_oe

The output enable for the lane 0 and the lane R output drivers of a 1x/Nx port.

signal_detect

Asserted when a lane receiver is enabled and a signal meeting an implementation defined criteria is present at the input of the receiver. The use of signal_detect is implementation dependent. It may be continuously asserted or it may be used to require that some implementation defined additional condition be met before the Lane_Synchronization state machine is allowed to exit the NO_SYNC state. Signal_detect might for example be used to ensure that the input signal to a lane receiver meet some minimum AC input power requirement to prevent the receiver from locking on to crosstalk.

silence_tmr_done

Asserted when silence_tmr_en has been continuously asserted for 120 +/- 40 µs and the state machine is in the SILENT state. The assertion of silence_tmr_done causes silence_tmr_en to be de-asserted. When the state machine is not in the SILENT state, silence_tmr_done is de-asserted.

silence_tmr_en

When asserted, the silence_tmr runs. When de-asserted, the silence_tmr is reset to and maintains its initial value.

/VALID/

When asserted, /VALID/ indicates that the current code-group is a valid code-group given the current running disparity.

Vcounter

Vcounter is used in the Lane_Synchronization state machine to count VALID received code-groups. There is one Vcounter for each lane in a Nx mode receiver.

4.12.4.2 Lane Synchronization State Machine

The Lane_Synchronization state machine monitors the bit synchronization and code-group boundary alignment for a lane receiver. A port that supports only 1x mode (1x port) has one Lane_Synchronization state machine. A port that supports Nx mode has N Lane_Synchronization state machines, one for each lane (Lane_Synchronization[0] through Lane_Synchronization[N-1]).

The Lane_Synchronization state machine is specified in Figure 4-14

The state machine determines the bit synchronization and code-group boundary alignment state of a lane receiver by monitoring the received code-groups and looking for code-groups containing the “comma” pattern, other valid code-groups and invalid code-groups. The “comma” pattern is the bit sequence that is used to establish code-group boundary alignment. When a lane is error free the “comma” pattern occurs only in the /K28.1/ and /K28.5/ code-groups. Several counters are used to provide hysteresis so that occasional bit errors do not cause spurious lane_sync state changes.

The state machine does not specify how bit synchronization and code-group boundary alignment is to be achieved. The methods used by a lane receiver to achieve bit synchronization and code-group boundary alignment are implementation dependent. However, an isolated single bit or burst error shall not cause the code-group boundary alignment mechanism to change alignment. For example, a single bit or burst error that results in a “comma” pattern across a code-group boundary shall not cause the code-group boundary alignment mechanism to change alignment.

The state machine starts in the NO_SYNC state and sets the variables Kcounter[n], Vcounter[n], and lane_sync[n] to 0 (lane n is out of code-group boundary sync). It then looks for a /COMMA/ code-group. When it finds one and the signal signal_detect[n] is asserted, the machine moves to the NO_SYNC_1 state.

The NO_SYNC_1 state in combination with the NO_SYNC_2 and NO_SYNC_3 states looks for the reception of 127 /COMMA/ and Vmin /VALID/ code-groups without any intervening /INVALID/ code-groups. When this condition is achieved, state machine goes to state SYNC. If an intervening /INVALID/ code-group is detected, the machine goes back to the NO_SYNC state.

The values of 127 and Vmin are selected such that it is highly unlikely that SYNC would be falsely reported and that the bit error rate (BER) is low enough that it is highly unlikely that once asserted, lane_sync will “flicker” ON and OFF while the training of the receiver timing recovery and any adaptive equalization is completed. Vmin shall have a minimum value of 0 and is implementation dependent. When Vmin = 0, the behavior of this Lane_Synchronization state machine is identical to that of the Lane_Synchronization state machine specified in Rev. 1.3 of this specification.

Table 4-11 shows the approximate maximum probability of lane_sync “flicker” for some values of Vmin and over the BER range of $1*10^{-2}$ to $1*10^{-12}$. It is recommended that Vmin be at least 2^{12} - 1.

Table 4-11. lane_sync “Flicker” Probability

Vmin	Approximate maximum probability of lane_sync flicker
0	0.24
2^{12} - 1	0.021
2^{13} - 1	0.011

Table 4-11. lane_sync "Flicker" Probability

Vmin	Approximate maximum probability of lane_sync flicker
2^{14} - 1	0.0056
2^{15} - 1	0.0028
2^{16} - 1	0.0014

When Vmin = 0 and IDLE1 is being received, something more than 256 code-groups must be received after the first /COMMA/ to achieve the 128 /COMMA/ code-groups without error criteria to transition to the SYNC state because the /COMMA/ code-group comprises slightly less than half of the code-groups in the IDLE1 sequence.

When Vmin = 0 and IDLE2 is being received, something more than 9 Idle Frames must be received after the first /COMMA/ to achieve the 127 /COMMA/ code-groups without error to transition to the SYNC state because there are on average about 14 /COMMA/ code-groups per Idle Frame.

In the SYNC state, the machine sets the variable lane_sync[n] to 1 (lane n is in code-group boundary sync), sets the variable Icounter[n] to 0 and begins looking for /INVALID/ code-groups. If an /INVALID/ code-group is detected, the machine goes to state SYNC_1.

The SYNC_1 state in combination with the SYNC_2, SYNC_3, and SYNC_4 states looks for 255 consecutive /VALID/ code-groups without any /INVALID/ code-groups. When 255 /VALID/ symbols are received, the Icounter[n] value is decremented in the transition through the SYNC_4 state. If it does not, it increments Icounter[n]. If Icounter[n] is decremented back to 0, the state machine returns to the SYNC state. If Icounter[n] is incremented to Imax, the state machine goes to the NO_SYNC state and starts over. Imax is an integer and shall have a value of 3 or greater for receivers not using DFE and a value of 4 or greater for receivers using DFE. This algorithm tolerates isolated single bit or burst errors in that an isolated single bit or burst error will not cause the machine to change the variable lane_sync[n] from 1 to 0 (in sync to out of sync).

A single bit error at the code-group level can cause two INVALID characters to be reported, one due to a corrupted code-group and one due to corrupted running disparity with causes a subsequent code-group to be reported as INVALID. A burst error no longer than 11 bits in length can cause three INVALID characters to be reported, two due to two corrupted code-groups and one due to corrupted running disparity which causes a subsequent code-group to be reported as INVALID.

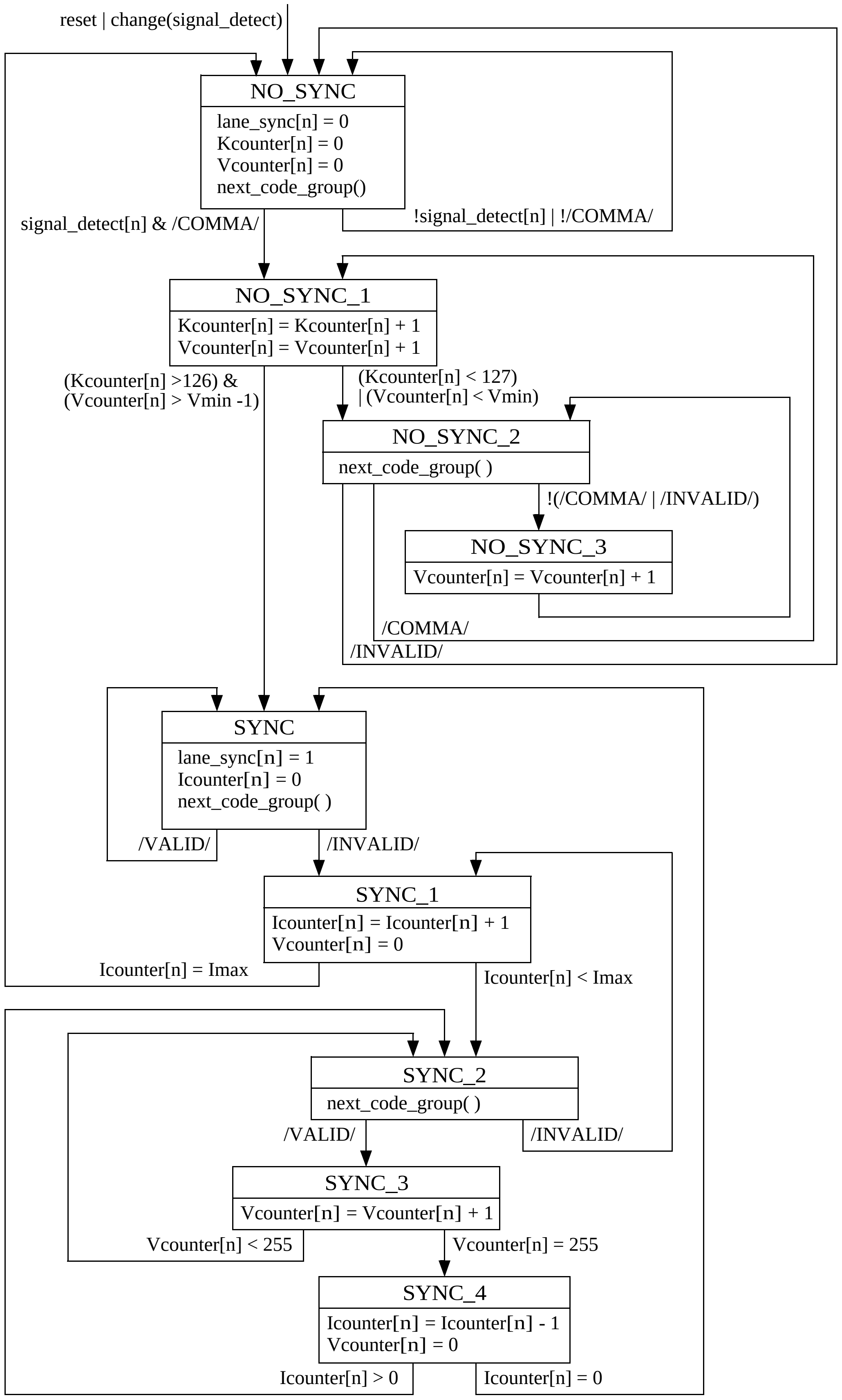


Figure 4-14. Lane_Synchronization State Machine

4.12.4.3 Lane Alignment State Machine

The Lane_Alignment state machine monitors the alignment of the output of the N lane receivers in a port operating in Nx mode. A port supporting one or more multi-lane modes has one Lane_Alignment state machine for each supported Nx mode. A port supporting only 1x mode does not have a Lane_Alignment state machine. Lane alignment is required in a Nx port receiver to compensate for unequal propagation delays through the N lanes.

The Lane_Alignment state machine is specified in Figure 4-15.

The state machine determines the alignment state by monitoring the N lanes for columns containing all /A/s (||A||), columns containing at least one but not all /A/s and columns containing no /A/s. Several counters are used to provide hysteresis so that isolated single bit or burst errors do not cause spurious lanes_aligned state changes.

The state machine does not specify how lane alignment is to be achieved. The methods used by a port receiver to achieve lane alignment are implementation dependent. However, isolated single bit or burst errors shall not cause the lane alignment mechanism to change lane alignment. For example, an isolated single bit or burst error that results in a column that contains at least one /A/ but not all /A/s shall not cause the lane alignment mechanism to change the lane alignment.

The state machine starts in the NOT_ALIGNED state where the variables Acounter and N_lanes_aligned are set to 0 (all N lanes are not aligned). The machine then waits for all N lanes to achieve code-group boundary alignment (N_lanes_sync asserted) and the reception of an ||A|| (a column of all /A/s). When this occurs, the machine goes to NOT_ALIGNED_1 state.

The NOT_ALIGNED_1 state in combination with the NOT_ALIGNED_2 state looks for the reception of four ||A||s without the intervening reception of a misaligned column (a column with at least one /A/ but not all /A/s which causes the signal align_error to be asserted). When this occurs, the machine goes to the ALIGNED state. If an intervening misaligned column is received, the machine goes back to the NOT_ALIGNED state.

In the ALIGNED state, the machine sets the variable N_lanes_aligned to 1 (all N lanes are aligned) and the variable Mcounter to 0 and looks for a misaligned column (align_error asserted). If a misaligned column is detected, the machine goes to the ALIGNED_1 state.

The ALIGNED_1 state in combination with the ALIGNED_2 and ALIGNED_3 states look for the reception of four ||A||s without the intervening reception of more than Mmax - 1 additional misaligned columns. If this condition occurs, the state machine returns to the ALIGNED state. If Mmax - 1 additional intervening misaligned columns occurs, the machine goes to the NOT_ALIGNED state and starts over. Mmax is an integer and shall have a value of 2 or greater for receivers not using DFE and a value of 3 or greater for receivers using DFE.

This algorithm tolerates an isolated single bit or burst error in that such an error will not cause the machine to change the variable N_lanes_aligned from 1 to 0 (in lane alignment to out of lane alignment).

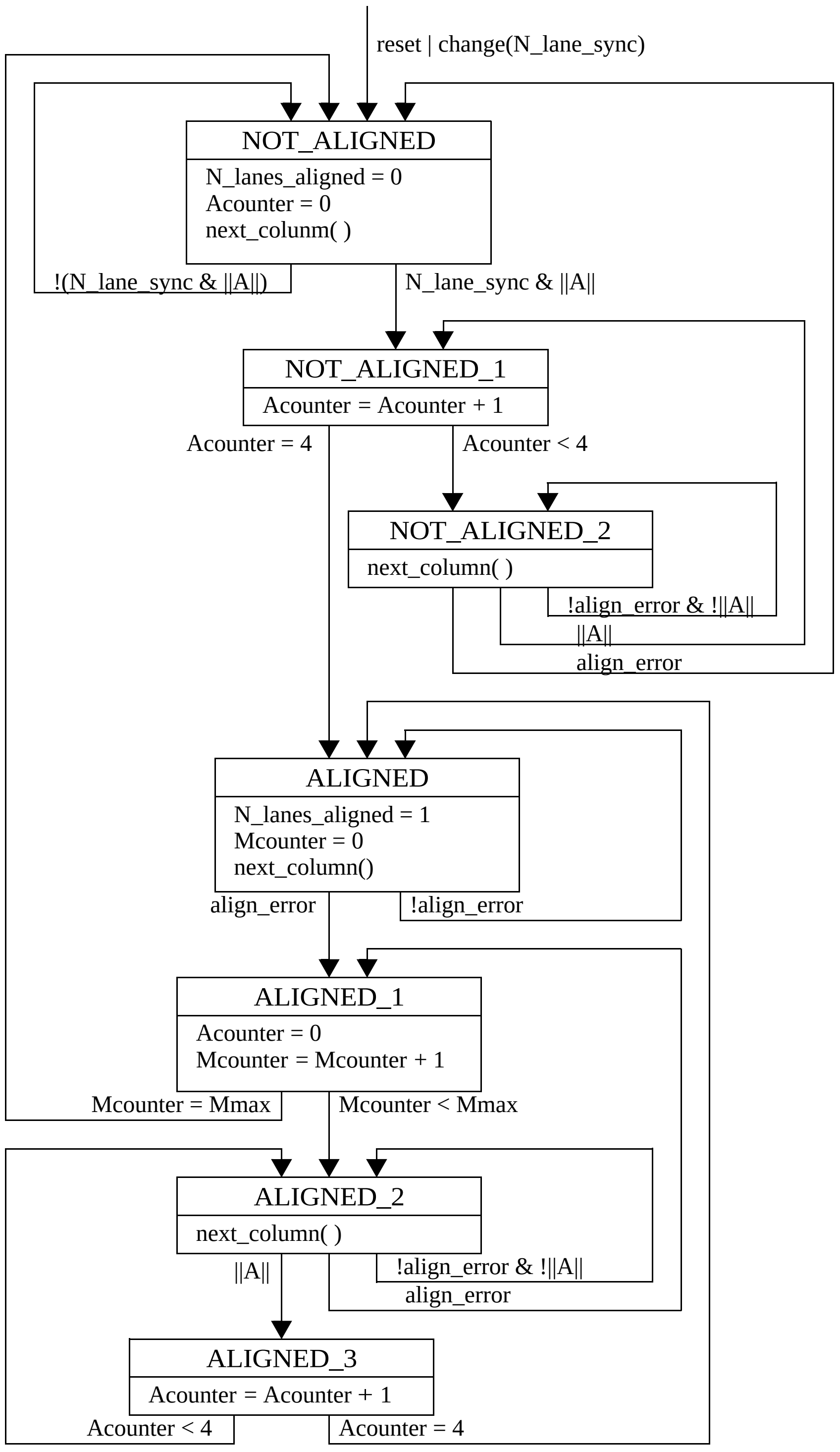


Figure 4-15. Lane_Alignment State Machine

4.12.4.4 1x/2x Mode Detect State Machine

The 1x/2x_Mode_Detect state machine monitors the columns formed from aligned characters received on lanes 0 and 1 of a port that supports 2x mode. When such a port is receiving an input signal on only lanes 0 and 1, the state machine is used to determine whether the connected port is transmitting in 1x mode or in 2x mode. A port that supports 2x mode shall have one 1x/2x_Mode_Detect state machine.

The 1x/2x_Mode_Detect state machine is specified in Figure 4-16.

Except for the case of N = 2, a 1x/Nx receiver can tell whether the connected port is operating in 1x mode or Nx mode by observing the number of active lanes it is receiving (the number of lanes for which lane_sync[n] is asserted). This follows from the fact that a 1x/Nx port operating in 1x mode transmits only on lanes 0 and R. In the case of N = 2, the port transmits on both lanes regardless of whether it is operating in 1x or 2x mode making it impossible for a 1x/2x receiver to determine the mode of the connected port based on the number of active lanes it is receiving. The 1x/2x_Mode_Detect state machine provides mode detection for the 1x/2x receiver.

The 1x/2x_Mode_Detect state machine enters the INITIALIZE state whenever the port is reset or the state of 2_lanes_aligned changes state. The machine initializes the 1x_mode_detected and Dcounter variables (the connected port is initially assumed to be operating in 2x mode) and waits for the lanes to become aligned. Once the two lanes are aligned, the machine goes to the GET_COLUMN state to get the next available column.

In the GET_COLUMN state, each column is examined as it becomes available to determine whether it contains any control symbol delimiter special characters (SC or PD characters). If no SC or PD characters are found, no action is taken and the state machine remains in the GET_COLUMN state. If the column contains a single SC or PD special character, the column is determined to be a 2x mode delimiter and the state machine enters the 2x_DELIMITER state. If the column contains a two SC or two PD special characters, the column is determined to be a 1x mode delimiter and the state machine enters the 1x_DELIMITER state.

In the 1x_DELIMITER state, the Dcounter is decremented by 1 and the value of the Dcounter is tested. If the Dcounter is > 0, the state machine goes to the GET_COLUMN state. If the Dcounter is = 0, the state machine goes to the SET_1x_MODE state where 1x_mode_detected is set to 1. The state machine then goes to the GET_COLUMN state.

In the 2x_DELIMITER state, the Dcounter is incremented by 1 and the value of the Dcounter is tested. If the Dcounter is < 3, the state machine goes to the GET_COLUMN state. If the Dcounter is = 3, the state machine goes to the SET_2x_MODE state where 1x_mode_detected is set to 0. The state machine then goes to the GET_COLUMN state.

The Dcounter is used to prevent transmission errors from erroneously changing the state of 1x_mode_detected.

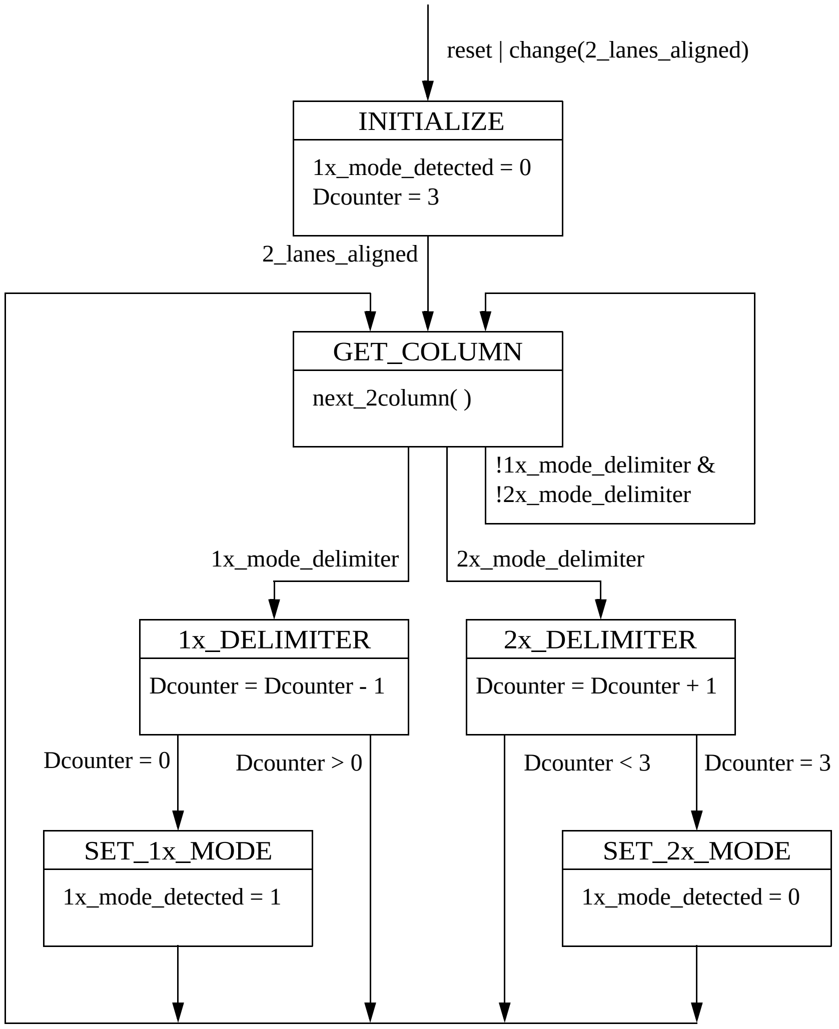


Figure 4-16. 1x/2x_Mode_Detect State Machine

4.12.4.5 1x Mode Initialization State Machine

The 1x_Initialization state machine specified in this section shall be used by ports that support only 1x mode (1x ports). The state machine is specified in Figure 4-17.

The machine starts in the SILENT state. The link output driver is disabled to force the link partner to initialize regardless of its current state. The duration of the SILENT state is controlled by the silence_tmr. The duration must be long enough to ensure that the link partner detects the silence (as a loss of lane_sync) and is forced to initialize but short enough that it is readily distinguished from a link break. When the silent interval is complete, the SEEK state is entered.

In the SEEK state, the link output driver is enabled, an idle sequence is transmitted, and the port waits for lane_ready to be asserted indicating the presence of a link partner. While lane_ready as defined indicates the bit and code-group boundary alignment state of the link receiver, it is used by the state machine to indicate the presence of a link partner. When lane_ready and idle_selected are both asserted, the 1X_MODE state is entered.

The input signal force_reinit allows the port to force link initialization at any time.

The variable port_initialized is asserted only in the 1X_MODE state.

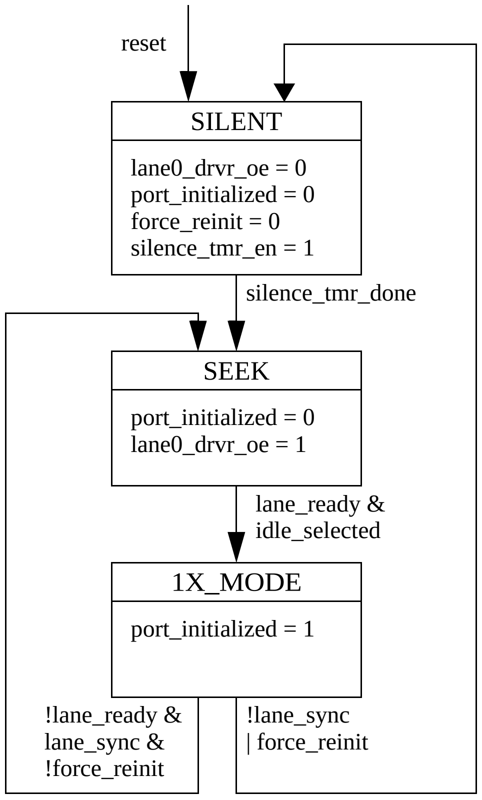


Figure 4-17. 1x_Initialization State Machine

4.12.4.6 1x/Nx Mode Initialization State Machine for N = 4, 8, 16

The 1x/Nx_Initialization state machines specified in this section shall be used by ports that support both 1x mode and an Nx mode (1x/Nx ports) for N = 4, 8 or 16. The initialization state machine for 1x/2x ports is specified in Section 4.12.4.7. 1x/8x and 1x/16x ports shall use the 1x/Nx_Initialization state machine specified in Figure 4-18. 1x/4x ports should use the 1x/Nx_Initialization state machine specified in Figure 4-18, but may use the 1x/4x_Initialization state machine specified in Figure 4-19. The 1x/4x_Initialization state machine of Figure 4-19 shall not be used in new designs.

The 1x/Nx_Initialization state machine controls port initialization and determines when the port is initialized. The state machine also controls whether the port receiver operates in 1x or Nx mode and in 1x mode whether lane 0 or lane 2, the 1x mode redundancy lane, is selected for control symbol and packet reception.

The 1x/Nx_Initialization state machine starts in SILENT state. All N lane output drivers are disabled to force the link partner to re-initialize regardless of its current state. The duration of the SILENT state is controlled by the silence_tmr. The duration must be long enough to ensure that the link partner detects the silence (as a loss of lane_sync) and is forced to re-initialize. When the silent interval is complete, the state machine enters the SEEK state.

In the SEEK state, a 1x/Nx port transmits an idle sequence on lanes 0 and 2 (the other output drivers remain disabled to save power) and waits for an indication that a link partner is present. While lane_sync as defined indicates the bit and code-group boundary alignment state of a lane receiver, it is used by the state machine to indicate the presence of a link partner. A link partner is declared to be present when either lane_sync[0] or lane_sync[2] is asserted. The assertion of idle_selected and either lane_sync[0] or lane_sync[2] causes the state machine to enter the DISCOVERY state.

In the DISCOVERY state, the port enables the output drivers for all N lanes and transmits an idle sequence on all N lanes if Nx mode is enabled. The discovery timer (disc_tmr) is started. The discovery timer allows time for the link partner to enter its DISCOVERY state and if Nx mode is enabled in the link partner, for all N local lane receivers to acquire bit synchronization and code-group boundary alignment and to complete the training of any adaptive equalization that is present and for all N lanes to be aligned.

While waiting for the end of the discovery period (disc_tmr_en asserted but disc_tmr_done de-asserted), if Nx_mode is enabled, all N lanes become ready and lane alignment is achieved (N_lanes_ready asserted), the machine enters the Nx_MODE state. If force_1x_mode is asserted (Nx_mode_enabled is deasserted), force_laneR is not asserted and lane 0 becomes ready (lane_ready[0] asserted), the machine enters the 1x_MODE_LANE0 state. If both force_1x_mode and force_laneR are asserted and lane 2 becomes ready (lane_ready[2] asserted), the machine enters the 1x_MODE_LANE2 state.

At the end of the discovery period (disc_tmr_done asserted), if the state machine has not entered the Nx_mode or one of the 1x modes and at least one of lane 0 or lane 2 is ready, the machine will enter one of the 1x mode states. If lane 0 is ready and either force_1x_mode and force_laneR are asserted but lane 2 is not ready or Nx mode is enabled but N_lanes_ready is deasserted, the machine enters the 1X_MODE_LANE0 state. If lane 2 is ready, lane 0 is not ready and either force_1x_mode is asserted and force_laneR is not asserted or neither force_1x_mode nor N_lanes_ready are asserted, the machine enters the 1X_MODE_LANE2 state. If neither lane_ready[0] nor lane_ready[2] is asserted, the machine enters the SILENT state and restarts the port initialization process.

If lane synchronization for both lane 0 and lane R is lost (both lane_sync[0] and lane_sync[2] de-asserted) during the DISCOVERY state, the state machine enters the SILENT state and restarts the port initialization process.

When in the Nx_MODE state, port_initialized is asserted. If N_lanes_ready is lost (N_lanes_ready de-asserted), the state machine transitions to either the SILENT state if both lane_sync[0] and lane_sync[2] are de-asserted or the DISCOVERY state if either lane_sync[0] or lane_sync[2] is asserted. This allows a 1x/Nx port in the Nx_MODE state to recover to Nx_MODE if N_lanes_ready was de-asserted due to multi-bit reception error or the need to retrain some of the adaptive equalization, but also allows the port to switch to 1x mode if the port is no longer able to receive in Nx mode or if the connected 1x/Nx port is not able to receive in Nx mode and has switched to 1x mode.

When in the 1x_MODE_LANE0 state, port_initialized is asserted. If lane_ready[0] is de-asserted but lane_sync[0] is still asserted, the machine transitions to the 1x_RECOVERY state to attempt recovery to the 1x_MODE_LANE0 state. If lane_sync[0] is de-asserted the state machine enters the SILENT state.

When in the 1x_MODE_LANE2 state, port_initialized is asserted. If lane_ready[2] is de-asserted but lane_sync[2] is still asserted, the machine transitions to the 1x_RECOVERY state to attempt recovery to the 1x_MODE_LANER state. If lane_sync[2] is de-asserted, the state machine enters the SILENT state.

When the 1x_RECOVERY state is entered, the discovery timer (disc_tmr_en asserted) is started. The port reenters the 1x_MODE_LANE0 state if lane_ready[0] is reasserted and the port was in the 1x_MODE_LANE0 state immediately before entering the 1x_RECOVERY state. The port reenters the 1x_MODE_LANE2 state if lane_ready[2] is reasserted and the port was in the 1x_MODE_LANE2 state immediately before entering the 1x_RECOVERY state. If both lane_sync[0] and lane_sync[2] are lost (both lane_sync[0] and lane_sync[R] de-asserted), the SILENT state in entered. To prevent that state machine from possibly being stuck in the 1x_RECOVERY state, if the appropriate lane_ready[ ] is not asserted before the discovery time is up (disc_tmr_done asserted), the SILENT state is entered.

The state machine does not support recovery from a 1x mode state to Nx_MODE or the other 1x mode without going through the SILENT state.

The input signals force_1x_mode and force_laneR allow the state of the machine to be forced during initialization into 1x mode, and in 1x mode to be forced to receive on lane 2.

The input signal force_reinit allows the port to force port n link re-initialization at any time.

The variable port_initialized is asserted only in the 1x_MODE_LANE0, 1x_MODE_LANE2 and Nx_MODE states.

NOTE:

> The name and specified function of the state machine variable N_lanes_drvr_oe is potentially confusing. As specified, its assertion causes the drivers for all N lanes of an Nx link to be output enabled.

However, N_lanes_drvr_oe is only asserted when the state machine variable lanes02_drvr_oe is also asserted. (The assertion of lanes02_drvr_oe causes the drivers for lane 0 and 2 to be output enabled). As a consequence, the net effect of the assertion or de-assertion of N_lanes_drvr_oe is that the drivers of all of the N lanes except the lanes 0 and 2 are output enabled or disabled respectively. The operation of an implementation that uses lanes02_drvr_oe as the output enable for the seek lane drivers and N_lanes_drvr_oe as the output enable for the remaining N-2 lanes will be operationally indistinguishable from an implementation that uses (lanes02_drvr_oe OR N_lanes_drvr_oe) as the output enable for the seek lane drivers and N_lanes_drvr_oe as the output enable for the remaining N-2 lanes.

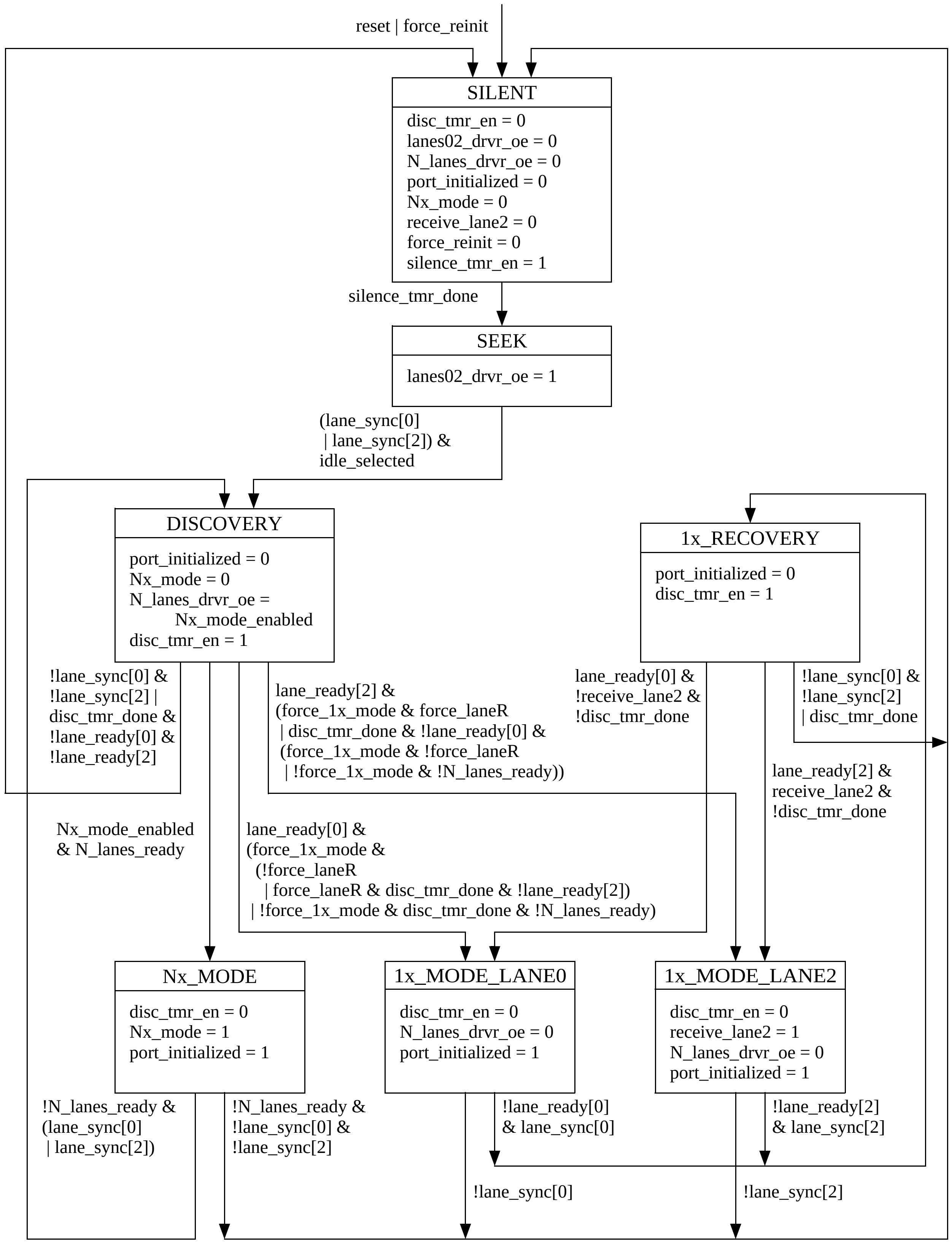

reset | force_reinit
SILENT
disc_tmr_en = 0
lanes02_drvr_oe = 0
N_lanes_drvr_oe = 0
port_initialized = 0
Nx_mode = 0
receive_lane2 = 0
force_reinit = 0
silence_tmr_en = 1
silence_tmr_done
SEEK
lanes02_drvr_oe = 1
(lane_sync[0] | lane_sync[2]) & idle_selected
DISCOVERY
port_initialized = 0
Nx_mode = 0
N_lanes_drvr_oe = Nx_mode_enabled
disc_tmr_en = 1
1x_RECOVERY
port_initialized = 0
disc_tmr_en = 1
!lane_sync[0] & !lane_sync[2] | disc_tmr_done & !lane_ready[0] & !lane_ready[2]
lane_ready[2] & (force_1x_mode & force_laneR | disc_tmr_done & !lane_ready[0] & (force_1x_mode & !force_laneR | !force_1x_mode & !N_lanes_ready))
lane_ready[0] & !receive_lane2 & !disc_tmr_done
!lane_sync[0] & !lane_sync[2] | disc_tmr_done
lane_ready[2] & receive_lane2 & !disc_tmr_done
Nx_mode_enabled & N_lanes_ready
lane_ready[0] & (force_1x_mode & (!force_laneR | force_laneR & disc_tmr_done & !lane_ready[2]) | !force_1x_mode & disc_tmr_done & !N_lanes_ready)
Nx_MODE
disc_tmr_en = 0
Nx_mode = 1
port_initialized = 1
1x_MODE_LANE0
disc_tmr_en = 0
N_lanes_drvr_oe = 0
port_initialized = 1
1x_MODE_LANE2
disc_tmr_en = 0
receive_lane2 = 1
N_lanes_drvr_oe = 0
port_initialized = 1
!N_lanes_ready & (lane_sync[0] | lane_sync[2])
!N_lanes_ready & !lane_sync[0] & !lane_sync[2]
!lane_ready[0] & lane_sync[0]
!lane_ready[2] & lane_sync[2]
!lane_sync[0]
!lane_sync[2]

Figure 4-18. 1x/Nx_Initialization State Machine for N = 4, 8, 16

The following Initialization state machine may be used for 1x/4x ports that support only the IDLE1 idle sequence. The only difference between the 1x/Nx Initialization state machine of Figure 4-18 and the 1x/4x_Initialization state machine of Figure 4-19 is that the 1x/4x_Initialization machine does not have the 1x_RECOVERY state. As a consequence, the machines have different behavior when force_1x_mode is asserted. Unlike the 1x/Nx machine, the 1x/4x machine does not have a bias for the 1x_MODE_LANE0 state when force_1x_mode is not asserted.

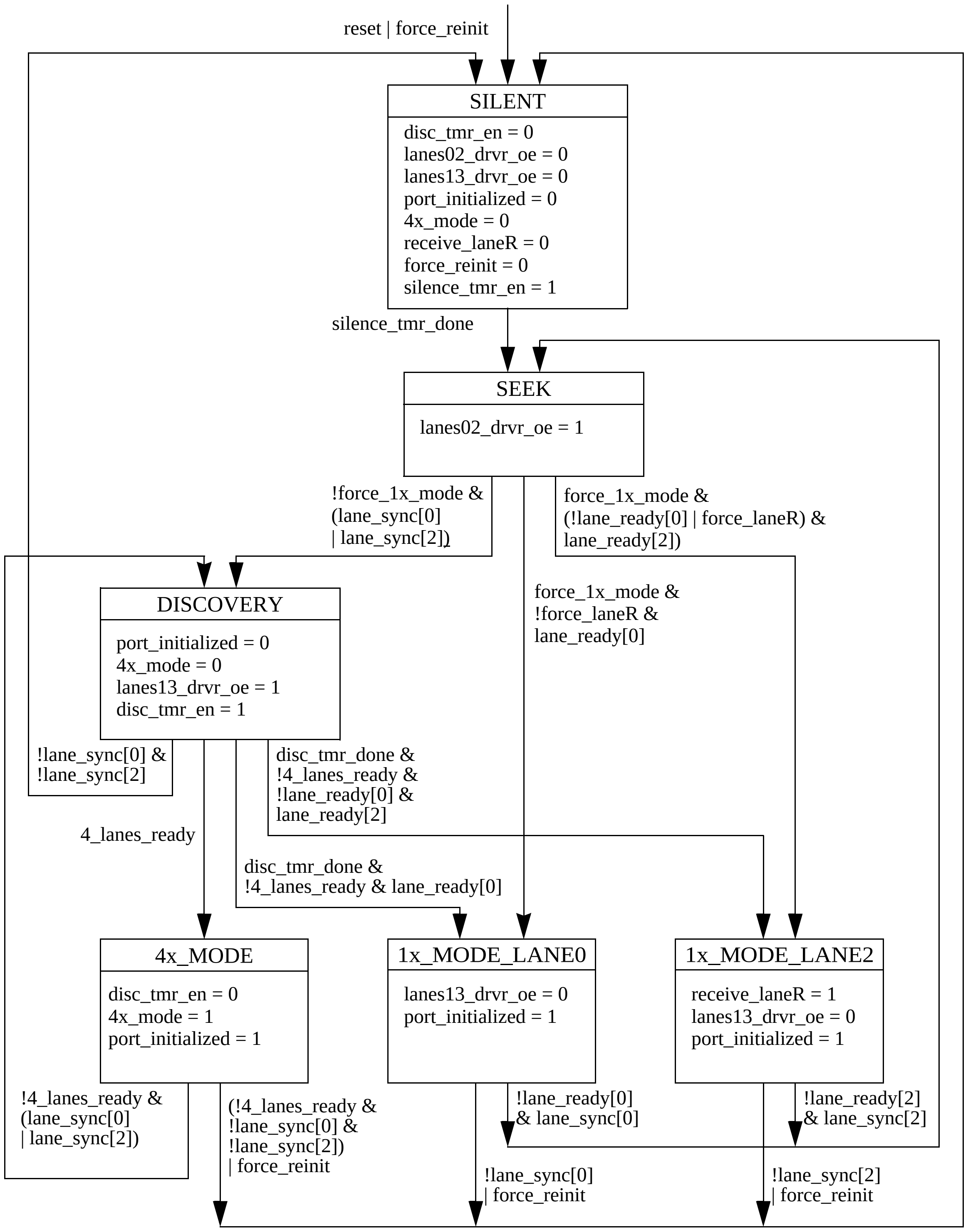

reset | force_reinit
SILENT
disc_tmr_en = 0
lanes02_drvr_oe = 0
lanes13_drvr_oe = 0
port_initialized = 0
4x_mode = 0
receive_laneR = 0
force_reinit = 0
silence_tmr_en = 1
silence_tmr_done
SEEK
lanes02_drvr_oe = 1
!force_1x_mode & (lane_sync[0] | lane_sync[2])
force_1x_mode & (!lane_ready[0] | force_laneR) & lane_ready[2])
force_1x_mode & !force_laneR & lane_ready[0]
DISCOVERY
port_initialized = 0
4x_mode = 0
lanes13_drvr_oe = 1
disc_tmr_en = 1
!lane_sync[0] & !lane_sync[2]
disc_tmr_done & !4_lanes_ready & !lane_ready[0] & lane_ready[2]
4_lanes_ready
disc_tmr_done & !4_lanes_ready & lane_ready[0]
4x_MODE
disc_tmr_en = 0
4x_mode = 1
port_initialized = 1
1x_MODE_LANE0
lanes13_drvr_oe = 0
port_initialized = 1
1x_MODE_LANE2
receive_laneR = 1
lanes13_drvr_oe = 0
port_initialized = 1
!4_lanes_ready & (lane_sync[0] | lane_sync[2])
(!4_lanes_ready & !lane_sync[0] & !lane_sync[2]) | force_reinit
!lane_ready[0] & lane_sync[0]
!lane_ready[2] & lane_sync[2]
!lane_sync[0] | force_reinit
!lane_sync[2] | force_reinit

Figure 4-19. Alternate 1x/4x_Initialization State Machine

4.12.4.7 1x/2x Mode Initialization State Machine

The 1x/2x_Initialization state machine specified in this section shall be used by 1x/2x ports. Except for the method it uses to decide whether to operate in 1x or 2x mode and the use of lane 1 as the redundancy lane, this state machine is identical to the 1x/Nx_Initialization state machine specified in Figure 4-18 with N = 2.

Ports that support more than 2 lanes disable all lanes except lanes 0 and R when operating in 1x mode. This allows the Initialization state machine for a port supporting more than 2 lanes to use the number of active lanes the port is receiving to determine whether to operate in 1x or Nx mode. 1x/2x ports transmit on both lanes regardless of whether they are operating in 1x or 2x mode. As a result, 1x/2x ports need a mechanism other than the number of active lanes being received to determine whether to operate in 1x or 2x mode. The 1x/2x_Mode_Detect state machine specified in Section 4.12.4.4 provides this mechanism.

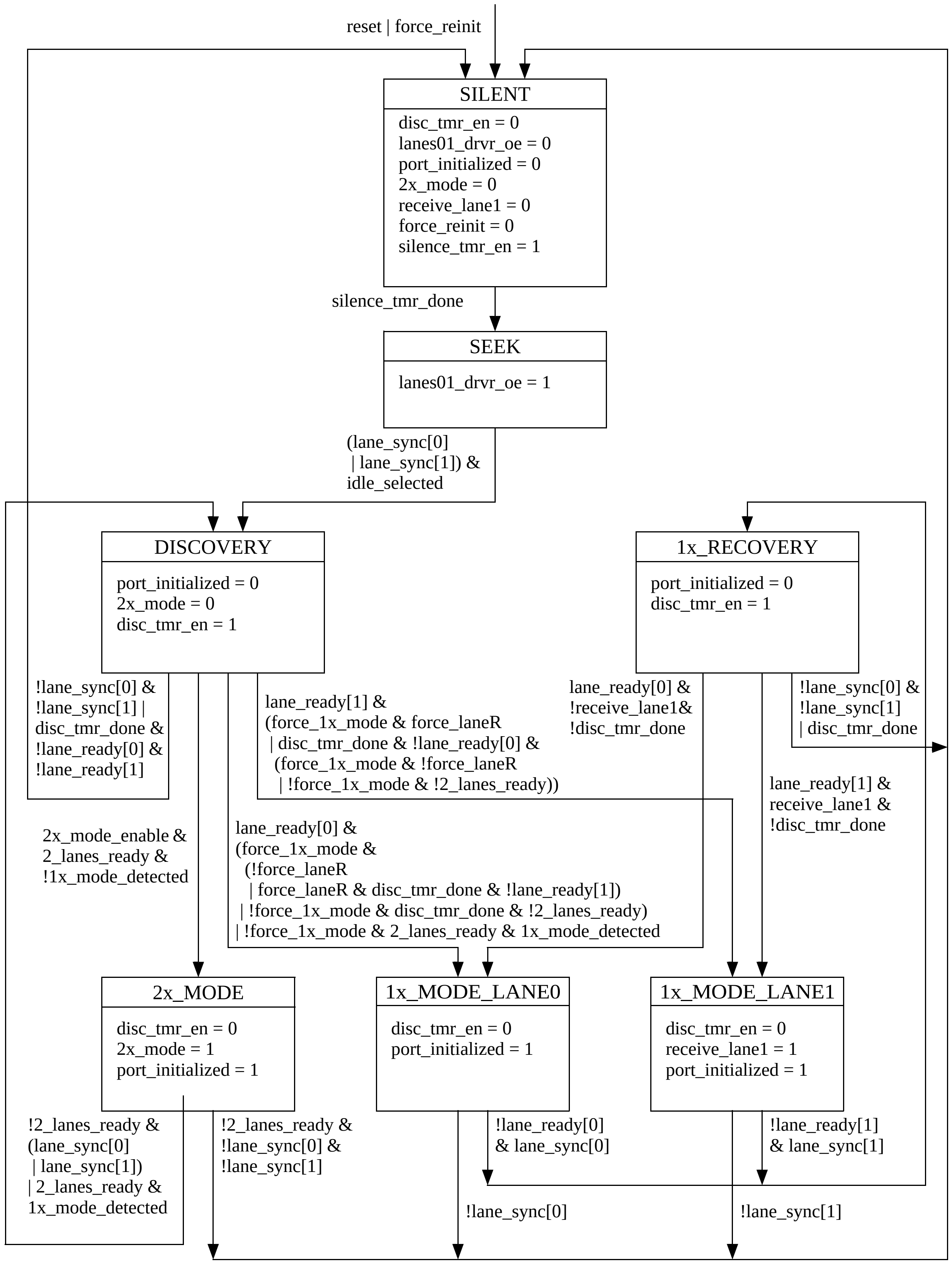

reset | force_reinit
SILENT
disc_tmr_en = 0
lanes01_drvr_oe = 0
port_initialized = 0
2x_mode = 0
receive_lane1 = 0
force_reinit = 0
silence_tmr_en = 1
silence_tmr_done
SEEK
lanes01_drvr_oe = 1
(lane_sync[0] | lane_sync[1]) & idle_selected
DISCOVERY
port_initialized = 0
2x_mode = 0
disc_tmr_en = 1
1x_RECOVERY
port_initialized = 0
disc_tmr_en = 1
!lane_sync[0] & !lane_sync[1] | disc_tmr_done & !lane_ready[0] & !lane_ready[1]
lane_ready[1] & (force_1x_mode & force_laneR | disc_tmr_done & !lane_ready[0] & (force_1x_mode & !force_laneR | !force_1x_mode & !2_lanes_ready))
lane_ready[0] & !receive_lane1& !disc_tmr_done
!lane_sync[0] & !lane_sync[1] | disc_tmr_done
lane_ready[1] & receive_lane1 & !disc_tmr_done
2x_mode_enable & 2_lanes_ready & !1x_mode_detected
lane_ready[0] & (force_1x_mode & (!force_laneR | force_laneR & disc_tmr_done & !lane_ready[1]) | !force_1x_mode & disc_tmr_done & !2_lanes_ready) | !force_1x_mode & 2_lanes_ready & 1x_mode_detected
2x_MODE
disc_tmr_en = 0
2x_mode = 1
port_initialized = 1
1x_MODE_LANE0
disc_tmr_en = 0
port_initialized = 1
1x_MODE_LANE1
disc_tmr_en = 0
receive_lane1 = 1
port_initialized = 1
!2_lanes_ready & (lane_sync[0] | lane_sync[1]) | 2_lanes_ready & 1x_mode_detected
!2_lanes_ready & !lane_sync[0] & !lane_sync[1]
!lane_ready[0] & lane_sync[0]
!lane_ready[1] & lane_sync[1]
!lane_sync[0]
!lane_sync[1]

Figure 4-20. 1x/2x_Initialization State Machine

4.12.4.8 1x/Mx/Nx Mode Initialization State Machines

A Nx port may optionally support more than one multi-lane mode of operation. For example, an 8x port may support 4x mode in addition to the 8x mode and 1x modes. A port supporting more than one multi-lane mode is referred to as a 1x/Mx/ .... /Nx port where 1 < M < ... < N.

The initialization state machine for a port that supports multiple multi-lane modes of operation requires two or three additional states for each additional supported mode of multi-lane operation.

Like the 1x/Nx_Initialization state machine, the 1x/Mx/Nx_Initialization state machines support link width negotiation. The negotiation algorithm implemented by the state machine attempts to select the greatest link width supported by both ports of a connected port pair. However, once a link width is selected, a wider link width can be selected only if the state machine enters the SILENT state which restarts the selection algorithm.

4.12.4.8.1 1x/2x/Nx Initialization State Machine

The 1x/2x/Nx_Initialization state machine is specified in Figure 4-21 and shall be used by 1x/2x/Nx ports. Because the redundancy lane, lane R, differs for a 1x/2x port and a 1x/Nx port (N = 4, 8 or 16), the Initialization state machine for a 1x/2x/Nx port is the most complicated of the possible 1x/Mx/Nx_Initialization state machines.

The 1x/2x/Nx_Initialization state machine has three more states than a 1x/Nx_Initialization state machine, the 2x_MODE, 2x_RECOVERY and the 1x_MODE_LANE1 states

The operation of the 1x/2x/Nx_Initialization state machine is essentially the same as that of a 1x/2x_Initialization state machine for the 1x and 2x modes operation and that of a 1x/Nx_Initialization state machine for Nx mode operation. The differences between the 1x/2x/Nx_Initialization state machine and the 1x/2x_Initialization and 1x/Nx_Initialization state machines are as follows.

In the SEEK state, the lanes whose drivers are output enabled depend on the modes that are enabled. Lanes 0 and 1 are output enabled if the 2x mode is enabled. Lanes 0 and 2 are output enabled if the Nx mode is enabled or the 2x mode is disabled. And if both modes are enabled, lanes 0, 1 and 2 are output enabled. The state machine enters the DISCOVERY state when lane_sync is asserted for lanes 0, 1 or 2.

In the DISCOVERY state, the lane selection priority for 1x mode is lane 0 first, lane 2 second and lane 1 third. This priority is to bias the selection to lane 0 and to ensure that lane 2, not lane 1, is selected when 4x mode or wider is enabled in the connected port.

In the 2x_MODE state, the state machine transitions to the 2x_RECOVERY state if 1x_mode_detected is asserted. The state machine goes to the 2x_RECOVERY state rather than directly to the 1x_MODE_LANE0 state so that the port_initialized bit is de-asserted indicating that the port is no longer in the normal operational state and that the link must be re-initialized before packet transmission can be resumed. Once in the 2x_RECOVERY state, the state machine then transitions to the 1x_MODE_LANE0 state if both 2_lanes_ready and 1x_mode_detected are still asserted.

The 2x_RECOVERY state is used to prevent the port from recovering to Nx mode once 2x mode has been selected.

In the 1x_MODE_LANE2 state, the state machine is allowed to transition to the 1x_MODE_LANE1 *state* via the 1x_RECOVERY state in the event that the connected port is a 1x/2x/Nx port and the connected port switches to 2x_MODE.

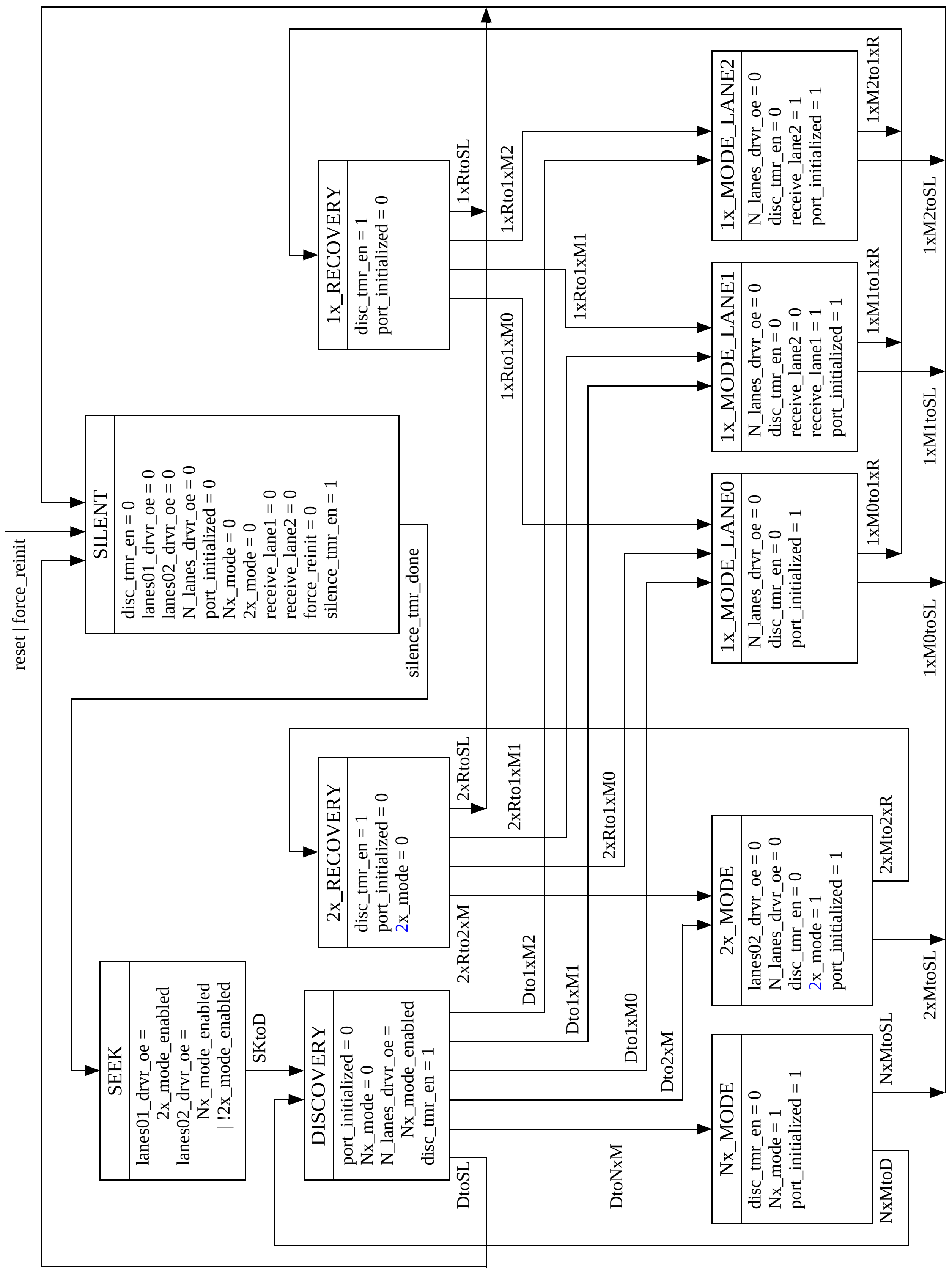


Figure 4-21. 1x/2x/Nx_Initialization State Machine

The variables that are local to the 1x/2x/Nx_Initialization state machine shown in Figure 4-21 are defined as follows.

```
1xM0to1xR = !lane_ready[0] & lane_sync[0]

1xM0toSL = !lane_sync[0]

1xM1to1xR = !lane_ready[1] & lane_sync[1]

1xM1toSL = !lane_sync[1]

1xM2to1xR = !lane_ready[2] & (lane_sync[1] | lane_sync[2])

1xM2toSL = !lane_sync[2] & !lane_sync[1]

1xRto1xM0 = !disc_tmr_done & !receive_lane1 & !receive_lane2 & lane_ready[0]

1xR to1xM1 = !disc_tmr_done &
                (receive_lane1 | receive_lane2 & !lane_ready[2]) & lane_ready[1]

1xRto1xM2 = !disc_tmr_done & receive_lane2 & lane_ready[2]

1xRtoSL = !lane_sync[0] & !lane_sync[1] & !lane_sync[2]
                | disc_tmr_done

2xMto2xR = !2_lanes_ready & (lane_sync[0] | lane_sync[1])
                | 2_lanes_ready & 1x_mode_detected

2xMtoSL = !lane_sync[0] & !lane_sync[1]

2xRto1xM0 = disc_tmr_done & !2_lanes_ready & lane_ready[0]
                | 2_lanes_ready & 1x_mode_detected

2xRto1xM1 = disc_tmr_done & !2_lanes_ready & !lane_ready[0] & lane_ready[1]

2xRto2xM = 2_lanes_ready & !1x_mode_detected

2xRtoSL = !lane_sync[0] & !lane_sync[1]
                | disc_tmr_done & !lane_ready[0] & !lane_ready[1]

Dto1xM0 = lane_ready[0] &
                (  force_1x_mode &
                   (!force_laneR
                    | force_laneR & disc_tmr_done & !lane_ready[1] & !lane_ready[2]
                   )
                 | !force_1x_mode & disc_tmr_done &
                   (!Nx_mode_enabled | !N_lanes_ready) &
                   (!2x_mode_enabled | !2_lanes_ready)
                )
```

```
Dto1xM1 = disc_tmr_done & lane_ready[1] & !lane_ready[2] &
                    (  force_1x_mode &
                       (force_laneR | !force_laneR & disc_tmr_done & !lane_ready[0])
                     | !force_1x_mode & !lane_ready[0] &
                       (!Nx_mode_enabled | !N_lanes_ready) &
                       (!2x_mode_enabled | !2_lanes_ready)
                    )

Dto1xM2 = lane_ready[2] &
                    (  force_1x_mode &
                       (force_laneR | !force_laneR & disc_tmr_done & !lane_ready[0])
                     | !force_1x_mode & disc_tmr_done & !lane_ready[0] &
                        (!Nx_mode_enabled | !N_lanes_ready) &
                        (!2x_mode_enabled | !2_lanes_ready)
                    )

Dto2xM = 2x_mode_enabled & 2_lanes_ready &
                    (!Nx_mode_enabled | disc_tmr_done & !N_lanes_ready)

DtoNxM = Nx_mode_enabled & N_lanes_ready

DtoSL = !lane_sync[0] & !lane_sync[1] & !lane_sync[2]
                    | disc_tmr_done & !lane_ready[0] & !lane_ready[1] & !lane_ready[2]

NxMtoD = !N_lanes_ready & (lane_sync[0] | lane_sync[2])

NxMtoSL = !lane_sync[0] & !lane_sync[2]

SKtoD = (lane_sync[0] | lane_sync[1] | lane_sync[2]) & idle_selected
```

4.12.4.8.2 1x/Mx/Nx Initialization State Machine (N > M > 2)

The 1x/Mx/Nx_Initialization state machine for N > M > 2 is specified in Figure 4-22 and shall be used by 1x/Mx/Nx ports.

The 1x/Nx/Nx_Initialization state machine has two more states than a 1x/Nx_Initialization state machine, the Mx_MODE and Mx_RECOVERY states, but one less state than the 1x/2x/Nx_Initialization state machine, the 1x_MODE_LANE1 state. Its operation is most similar to that of the 1x/2x/Nx_Initialization state machine, but is less complex as the redundancy lane R is the same for all N and M > 2.

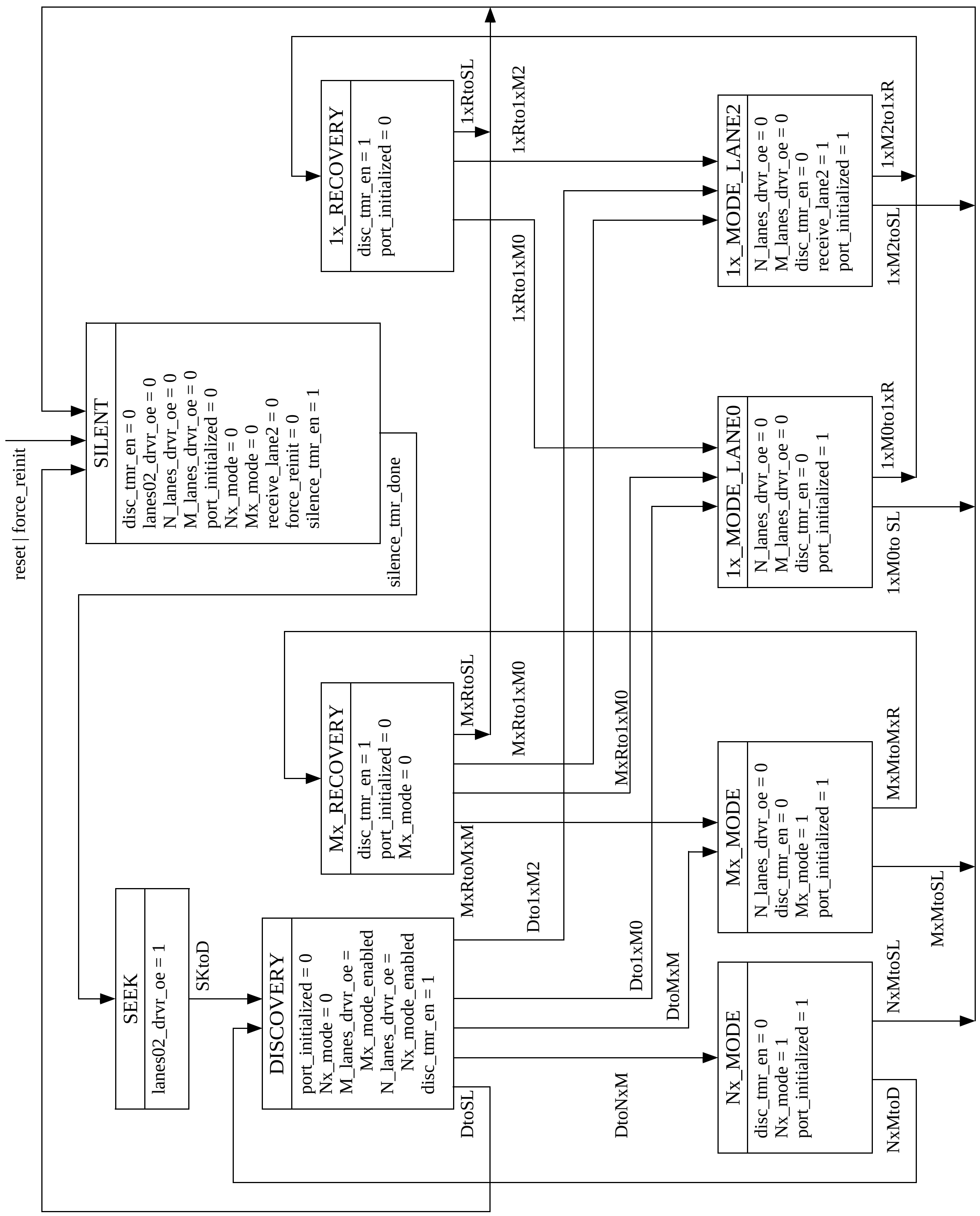


Figure 4-22. 1x/Mx/Nx_Initialization State Machine for N > M > 2

The variables that are local to the 1x/Mx/Nx_Initialization state machine shown in Figure 4-22 are defined as follows.

```
1xM0to1xR = !lane_ready[0] & lane_sync[0]

1xM0toSL = !lane_sync[0]

1xM2to1xR = !lane_ready[2] & lane_sync[2]

1xM2toSL = !lane_sync[2]

1xRto1xM0 = !disc_tmr_done & !receive_lane2 & lane_ready[0]

1xRto1xM2 = !disc_tmr_done & receive_lane2 & lane_ready[2]

1xRtoSL = !lane_sync[0] & !lane_sync[2]
                | disc_tmr_done

Dto1xM0 = lane_ready[0] &
                (  force_1x_mode &
                   (!force_laneR | force_laneR & disc_tmr_done & !lane_ready[2])
                  | !force_1x_mode & disc_tmr_done &
                    (!Nx_mode_enabled | !N_lanes_ready) &
                    (!Mx_mode_enabled | !M_lanes_ready)
                )

Dto1xM2 = lane_ready[2] &
                (  force_1x_mode &
                   (force_laneR | !force_laneR & disc_tmr_done & !lane_ready[0])
                  | !force_1x_mode & disc_tmr_done & !lane_ready[0] &
                     (!Nx_mode_enabled | !N_lanes_ready) &
                     (!Mx_mode_enabled | !M_lanes_ready)
                 )

DtoMxM = Mx_mode_enabled & M_lanes_ready &
                (!Nx_mode_enabled | disc_tmr_done & !N_lanes_ready)

DtoNxM = Nx_mode_enabled & N_lanes_ready

DtoSL = !lane_sync[0] & !lane_sync[2]
                | disc_tmr_done & !lane_ready[0] & !lane_ready[2]

MxMtoMxR = !M_lanes_ready & (lane_sync[0] | lane_sync[2])

MxMtoSL = !lane_sync[0] & !lane_sync[2]

MxRto1xM0 = disc_tmr_done & !M_lanes_ready & lane_ready[0]

MxRto1xM2 = disc_tmr_done & !M_lanes_ready & !lane_ready[0] & lane_ready[2]

MxRtoMxM = !disc_tmr_done & M_lanes_ready

MxRtoSL = !lane_sync[0] & !lane_sync[2]
                | disc_tmr_done & !lane_ready[0] & !lane_ready[2]

NxMtoD = !N_lanes_ready & (lane_sync[0] | lane_sync[2])

NxMtoSL = !lane_sync[0] & !lane_sync[2]
```

SKtoD = (lane_sync[0] | lane_sync[2]) & idle_selected

4.13 Structurally Asymmetric Links

Many power-sensitive applications have traffic patterns where the data flow in one direction of a link is always far greater than the other direction. Structurally asymmetric links (SAL) optimize transmitter and receiver designs by removing the unneeded unidirectional serial signaling paths.

SAL support is optional.

4.13.1 Definitions

Far Link Partner: The processing element that must be accessed over the Structurally Asymmetric Link.

Near Link Partner: The processing element that can be accessed without using the Structurally Asymmetric Link.

4.13.2 Structurally Asymmetric Link Operation

The procedure for configuring Structurally Asymmetric Link Operation is as follows:

1. Disable all Asymmetric Mode support on both link partners by writing zero to the “Asymmetric Modes Enabled” field in the Port n Power Management CSRs.
2. Configure the Far Link Partner registers: Port n Reinit Control CSR and Port n SAL Control and Status CSR.
3. Configure the Near Link Partner: Port n Reinit Control CSR and Port n SAL Control and Status CSR.

   After this step, the Port n SAL Control and Status CSR “SAL RX Width” field of each link partner shall match the other link partner's Port n SAL Control and Status CSR “SAL TX Width” field.

   Note that if the programmed configuration does not allow the links to successfully initialize in both directions, the link will recover by repeatedly decrementing the Silence Count field of the Port n Reinit Control CSR until Structurally Asymmetric Mode is disabled and the link reverts to redundant 1x operation.
4. Write 1 to the “Pulse Force Reinit” field in the Port n Reinit Control CSR.
5. Wait sufficient time for the link to reinitialize, as defined in Section 4.12, "Port Initialization".
6. Check the operational width of the Near Link Partner and Far Link Partner to confirm that the link is operating in Structurally Asymmetric Mode.

Structurally Asymmetric Link mode shall be attempted when SAL_Enabled is asserted. Structurally Asymmetric Link Operation mode shall be disabled when SAL_Enabled is deasserted.

Behavioral requirements for SAL RX Width and SAL TX Width field values are specified below in terms of which lanes are enabled for transmission and reception, what data is transmitted on each lane, and which lanes are enabled for reception. When SAL RX Width or SAL TX Width values are not 0b0000, the link partners shall not process received IDLE2 transmit emphasis commands (IDLE2 "ACK" and "NACK" fields shall be 0b0), and shall not send IDLE2 transmit emphasis commands ("Tap(+1) Command" and "Transmit emphasis tap(-1)" fields shall be 0b00).

Table 4-12. Structurally Asymmetric Link Tx/Rx Width Behaviors

SAL RX Width	SAL TX Width	Description
0b0000 (No Override)	0b0000 (No Override)	No effect on receive or transmit width.
0b0001 (1x, lane 0)	0b0001 (1x, lane 0. Disable lanes 1, 2, and 3)	Transmitter shall transmit a valid 1x bit stream on lane 0. Transmitter shall ensure that the link partner cannot detect a valid bit stream on lanes 1, 2, and 3. Receiver shall enable reception on lane 0 only. Receiver and transmitter shall operate as a 1x port.
0b0010 (1x, lane 1)	0b0010 (1x, lane 1. Disable lanes 0, 2, and 3)	Transmitter shall transmit a valid 1x bit stream on lane 1. Transmitter shall ensure that the link partner cannot detect a valid bit stream on lanes 0, 2, and 3. Receiver shall enable reception on lane 1 only. Receiver and transmitter shall operate as a 1x port.
0b0011 (1x, lane 2)	0b0011 (1x, lane 2. Disable lanes 0, 1, and 3)	Transmitter shall transmit a valid 1x bit stream on lane 2. Transmitter shall ensure that the link partner cannot detect a valid bit stream on lanes 0, 1, and 3. Receiver shall enable reception on lane 2 only. Receiver and transmitter shall operate as a 1x port.
0b0100 (1x, lane 3)	0b0100 (1x, lane 3. Transmit Lane 0 compliant data on lane 3. Disable Lanes 0, 1, and 2)	Transmitter shall transmit a valid 1x lane 0 bit stream on lane 3. Transmitter shall ensure that the link partner cannot detect a valid bit stream on lanes 0, 1, and 2. Receiver shall behave as if data received on lane 3 was actually received on lane 0. Receiver and transmitter shall operate as a 1x port.
0b0101 (2x, lanes 0 & 1. Lanes 2 and 3 are not used)	0b0101 (2x, lanes 0 & 1. Disable lanes 2 and 3)	Transmitter shall send valid 2x mode bit streams on lanes 0 and 1. Transmitter shall ensure that the link partner cannot detect a valid bit stream on lanes 2 and 3. Transmitter shall operate as a 2x port. Receiver shall operate as a 2x/1x port.

Table 4-12. Structurally Asymmetric Link Tx/Rx Width Behaviors

SAL RX Width	SAL TX Width	Description
0b0110 (2x, lanes 2 & 3)	0b0110 (2x, lanes 2 & 3. Transmit lane 0 and 1 2x compliant data streams on lanes 2 and 3. Disable transmission on lanes 0 and 1.)	Transmitter shall send a valid 2x mode bit stream, as composed for lane 0, on lane 2. Transmitter shall send a valid 2x mode bit stream, as composed for lane 1, on lane 3. Transmitter shall ensure that the link partner cannot detect a valid bit stream on lanes 0 and 1. Transmitter shall operate as a 2x port. Receiver shall behave as if the data received on lane 2 was actually received on lane 0, and as if the data received on lane 3 was actually received on lane 1. Receiver shall operate as a 2x/1x port.
0b0111 (4x, lanes 0-3)	0b0111 (4x, lanes 0-3)	Transmitter shall operate as a 4x port. Receiver shall operate as a 4x/1x port.
0b1000 (8x, lanes 0-7)	0b1000 (8x, lanes 0-7)	Transmitter shall operate as an 8x port. Receiver shall operate as an 8x/1x port.
0b1001 (16x)	0b1001 (16x)	Transmitter shall operate as a 16x/1x port. Receiver shall operate as a 16x/1x port.
0b1010- 0b1011 (Implementation specific)	0b1010- 0b1011 (Implementation specific)	Implementation specific behavior.
0b1100- 0b1111 (Reserved)	0b1100- 0b1111 (Reserved)	Reserved.

It is strongly recommended that devices which support structurally asymmetric links operating at Baud Rate Class 2 speeds implement register control of the transmit emphasis coefficient set.

4.14 Pseudo Random Binary Sequence Testing

Serial interfaces require Pseudo Random Binary Sequence (PRBS) generation and checking capabilities for qualifying and testing devices. This section specifies the PRBS generation and checking capabilities of RapidIO devices for in-field diagnostics. The capabilities defined are sufficient to perform diagnostics without the use of external test equipment. It is possible to use the capabilities defined in this section for qualifying devices.

Support for PRBS generation and checking is optional.

A link that is under test is not available for packet or control symbol exchange. If PRBS testing begins while packets are in flight the operation of the link is implementation specific, since PRBS testing may require an extended period of time.

Since a link cannot be used for packet or control symbol exchange while under test, the programming model for PRBS testing assumes that register access to one end of the link will be interrupted while the PRBS test is active. The end of the link that is not accessible during the test is known as the “far” end of the link. The “near” end of the link is the “far” end's link partner.

The following points define the operation of a PRBS test.

1. Program the following values on the far end of the link, in any order:
   - Port n Reinit Control CSR “Silence Count” to a non-zero value
   - Port n PRBS Control CSR “PRBS Pattern Selection”, “PRBS Lock Interval Threshold” and “PRBS Test Interval” values
   - Port n PRBS Lane Control CSR “PRBS Transmit Lane Control” and “PRBS Receive Lane Control” values
2. Repeat step 1 for the near end of the link.
3. Set the Port n Reinit Control CSR “Pulse Force-Reinit” field on the near end of the link to trigger Silence detection by the far end.
4. A PRBS test shall be performed if silence is detected and all of the following are true:
   - Port n Reinit Control CSR “Silence Count” value is not 0
   - Port n PRBS Control CSR “PRBS Pattern Selection” value is not 0
5. The Port n PRBS Control CSR “PRBS Active” bit shall be set at the start of a PRBS test. The bit shall remain asserted for the interval programmed in the PRBS Test Interval field. The bit shall be cleared when the PRBS Test Interval has completed. While PRBS Active is set, all receive lanes for the port shall pass data marked as “error” to the ports state machines.
6. All PRBS status values shall be cleared to 0 whenever the PRBS Active bit transitions from 0 to 1, including the following registers/fields:
   - Port n PRBS Control CSR “PRBS Completed”
   - Port n PRBS Status 0 CSR
   - Port n PRBS Status 1 CSR
   - Port n PRBS Locked Time CSR
7. The PRBS Completed bit shall remain cleared for the interval programmed in the PRBS Test Interval field. The PRBS Completed bit shall be set when the PRBS Test Interval has completed.
8. While PRBS Active is set, the port shall transmit the selected PRBS sequence on all lanes enabled in the Port n PRBS Lane Control CSR “PRBS Transmit Lane Control” field. Lanes that are disabled in the Port n PRBS Lane Control CSR “PRBS Transmit Lane Control” field shall be electrically idle.
9. While PRBS Active is set, the port shall check the selected PRBS sequence on all lanes enabled in the Port n PRBS Lane Control CSR “PRBS Receive Lane Control” field. The checking algorithm shall be as follows:
   - The Port n PRBS Status 1 CSR “Lane x PRBS Lock Status” shall be set if

checking for the lane is enabled and the received PRBS sequence has matched the predicted PRBS sequence for at least the PRBS Lock Interval Threshold.

- The “Lane x PRBS Error Count” shall be incremented by 1 if the received PRBS sequence does not match the predicted PRBS sequence and the “Lane x PRBS Lock Status” field is set. When the receive equalization method used by a device can cause a single error to be replicated as a burst of errors, the checking algorithm shall ensure that the “Lane x PRBS Error Count” shall be incremented by 1 for each burst of errors. An example of such a receive equalization method is DFE.

10. While PRBS Active is set, the Port n PRBS Locked Time CSR “All PRBS Locked Time” field shall be incremented by 1 whenever a period equal to the currently programed Discovery Timer period has expired, and the “Lane x PRBS Lock Status” field is set for all lanes enabled in the Port n PRBS Lane Control CSR “PRBS Receive Lane Control” field.

Chapter 5 64b/67b PCS and PMA Layers

5.1 Introduction

This chapter specifies the functions provided by the Physical Coding Sublayer (PCS) and Physical Media Attachment (PMA) sublayer used for 64b/67b encoded links. (The PCS and PMA terminology is adopted from IEEE 802.3). The topics include character representation, scrambling, lane striping, 64b/67b encoding, serialization of the data stream, codewords, columns, link transmission rules, idle sequences, and link initialization. The 64b/67b PCS and PMA Layers shall be supported by links operating at Baud Rate Class 3.

The concept of lanes is used to describe the width of a LP-Serial link. A lane is a single unidirectional signal path between two LP-Serial ports. Five widths are defined for LP-Serial links, 1, 2, 4, 8 and 16 lanes per direction. A link with N lanes in each direction is referred to as a Nx link, e.g. a link with 4 lanes in each direction is referred to as a 4x link.

5.2 PCS Layer Functions

The Physical Coding Sublayer (PCS) function is responsible for idle sequence generation, lane striping, scrambling and encoding for transmission and decoding, lane alignment, descrambling and destriping on reception. The PCS uses a 64b/67b encoding for transmission over the link.

The PCS also provides mechanisms for determining the operational mode of the port as Nx or 1x operation, and means to detect link states. It provides for clock difference tolerance between the sender and receiver without requiring flow control.

The PCS performs the following transmit functions:

- Adds link CRC-32 and padding as needed.
- Dequeues packets and control symbols awaiting transmission as a character stream.
- Stripes the transmit character stream across the available lanes.
- Scrambles outgoing data stream.
- Generates the idle sequence and inserts it into the transmit character stream for each lane when no packets or control symbols are available for transmission.

- Encodes the character stream of each lane independently into 67-bit parallel codewords.
- Passes the resulting 67-bit parallel codewords to the PMA.

The PCS performs the following receive functions:

- Decodes the received stream of 67-bit parallel codewords for each lane independently into characters.
- Marks characters decoded from errored codewords as invalid.
- If the link is using more than one lane, aligns the character streams to eliminate the skew between the lanes and reassembles (destripes) the character stream from each lane into a single character stream.
- Descrambles incoming data stream.
- Delivers the decoded character stream of packets and control symbols to the higher layers.
- Removes link CRC-32 and padding as needed.

5.3 PMA Layer Functions

The Physical Medium Attachment (PMA) Layer is responsible for serializing/de-serializing 67-bit parallel codewords to/from a serial bitstream on a lane-by-lane basis. Upon receiving data, the PMA function provides alignment of the received bitstream to 67-bit codeword boundaries, independently on a lane-by-lane basis. It then provides a continuous stream of 67-bit codewords to the PCS, one stream for each lane. The 67-bit codewords are not observable by layers higher than the PCS.

If a LP-Serial port supports either baud rate discovery or adaptive equalization, these functions are also performed in the PMA Layer.

5.4 Definitions

Definitions of terms used in this specification are provided below.

1x mode: An LP-Serial port mode of operation in which the port transmits on a single lane or receives on a single lane.

1x port: An LP-Serial port that supports a link with only one lane in each direction.

Asymmetric mode: An LP-Serial port mode of operation in which the number of lanes the port transmits on is independent from the number of lanes the port receives on.

Block: An entity of 64 bits of data with additional control to indicate the type of information carried in the 64-bit.

Byte: An 8-bit unit of information. Each bit of a byte has the value 0 or 1. The bits of a byte are numbered 0 through 7 with bit 0 being the most significant bit (msb).

Character: A 9-bit entity comprised of an information byte and a control bit that indicates whether the information byte contains data or control information. A byte is defined to contain data if it is part of a packet, is padding or idle bytes. A byte is defined to contain control information if it is part of a control symbol.

Codeword: A 67-bit entity that is the result of 64b/67b encoding of a block.

Codeword disparity: The number of “1”s in a codeword minus the number of “0”s in the codeword.

Column: The group of N codewords that are transmitted at nominally the same time by a LP-Serial port operating in Nx mode.

Destriping: The method used on a link operating in Nx mode to collect and merge the data across the N lanes received simultaneously and form a single block stream. This process reverses the operation done during striping of data across multiple lanes. For each direction of the link, the block stream is merged across the lanes, on a block-by-block basis, beginning with lane 0, continuing in incrementing lane number order across the lanes, and wrapping back to lane 0 for block N.

Differential Manchester Encoding (DME): A line code in which data and clock signals are combined to form a single 2-level self-synchronizing data stream. It is a differential encoding, using the presence or absence of transitions to indicate logical value. The DME scheme used in this specification is specified in Clause 72.6.10.2.2 of the *IEEE Standard 802.3-2008 (Part 5)*.

Disparity: The number of “1”s in an arbitrary block of binary data minus the number of “0”s in that block of data.

Idle sequence: The sequence of codewords that is transmitted by a port on each of its active output lanes when the port is not transmitting a packet or control symbol. The idle sequence allows the receiver to maintain bit synchronization, codeword alignment and, if applicable, adaptive equalization settings between packets and control symbols.

Lane: A single unidirectional signal path, typically a differential pair, between two LP-Serial ports.

Lane Alignment: The process of eliminating the skew between the lanes of a LP-Serial link operating in Nx mode such that the codewords transmitted as a column by the sender are output by the alignment process of the receiver as a column. Without lane alignment, the codewords transmitted as a column might be scattered across several columns output by the receiver.

Nx mode: A LP-Serial port mode of operation in which the port both transmits or receives on multiple lanes. A LP-Serial port operating in Nx mode transmits on N lanes and receives on N lanes where N has a value greater than 1. The transmit data stream is distributed across the N transmit lanes and the receive data stream is distributed across the N receive lanes.

Nx port: A LP-Serial port that supports a link with up to a maximum of N lanes in each direction.

Ordered Sequence: A sequence of two or more control codewords with fixed ordering.

Running Disparity: The running disparity of the signal transmitted over a lane is defined as the sum of the disparities of all of the codewords transmitted over the lane since the transmitting port exited the SILENT state.

Striping: The method used on a link operating in Nx mode to distribute data across the N lanes simultaneously. For each direction of the link, the block stream is *striped* across the lanes, on a block-by-block basis, beginning with lane 0, continuing in incrementing lane number order across the lanes, and wrapping back to lane 0 for block N.

5.5 64b/67b Transmission Code

The 64b/67b transmission code used by the PCS encodes 64-bit blocks of data and/or control information into 67-bit codewords for transmission and reverses the process on reception. There are two types of codewords: “data” codewords and “control” codewords. Data codewords encode 64 bits of data. Control codewords encode 64 bits of control information or some combination of data and control information.

Codewords are scrambled to statistically achieve an acceptable transition density for baud rate recovery in the receiver. Codewords are selectively inverted based on the running disparity and codeword disparity to ensure that the transmitted signal on each lane is DC balanced within +/- 66 1’s or 0’s at all times..

5.5.1 Codeword Format

The codeword is comprised of an inverted bit, a pair of bits marking the beginning and type of the codeword and a 64-bit data_field. The basic format of the codeword is shown in Table 5-1.

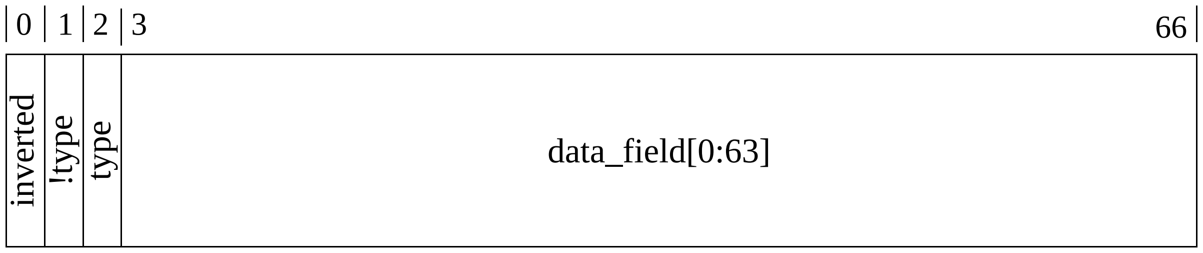


Figure 5-1. 64b/67b codeword format

The inverted bit indicates whether the data_field has been inverted to control the running disparity of the transmitted signal:

0b0 - data_field[0:63] has not been inverted.

0b1 - data_field[0:63] has been inverted.

The type bit indicates the type of codeword:

0b0: Control, the codeword encodes a block that contains control information and may contains data information.

0b1: Data, the codeword encodes a block that contains only data information.

The !type bit is the complement of the type bit.

The transition between the !type and type bits indicates a fixed offset from the beginning of the codeword, for use in codeword lock state machine in Section 5.19.4.

The format and content of the data_field depends on the information encoded in the codeword.

Codewords shall be transmitted from left to right, bit 0 to bit 66 starting with the inverted bit and progressing to data_field[63].

5.5.2 Data Codeword

The format of a data codeword (type bit = 0b1) shall be as shown in Figure 5-2.

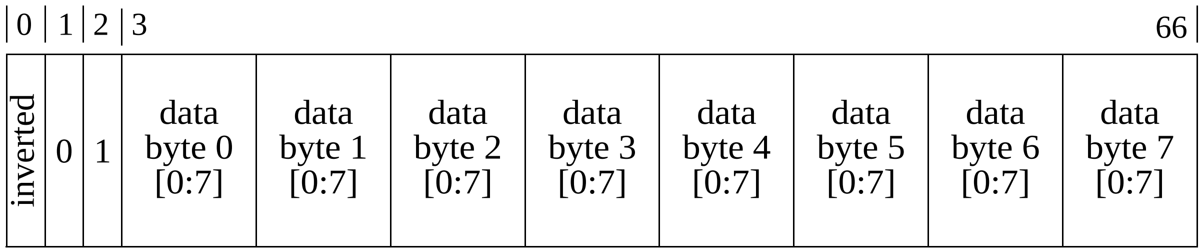


Figure 5-2. 64b/67b Data codeword format

5.5.3 Control Codeword

The format of a control codeword (type bit = 0b0) depends on the information the codeword encodes. The 2 bits at location [30:31] of a control codeword data_field are a cc_type field that specifies the contents and format of data_field[0:29,32:63]. The general format of a control codeword shall be as shown in Figure 5-3.

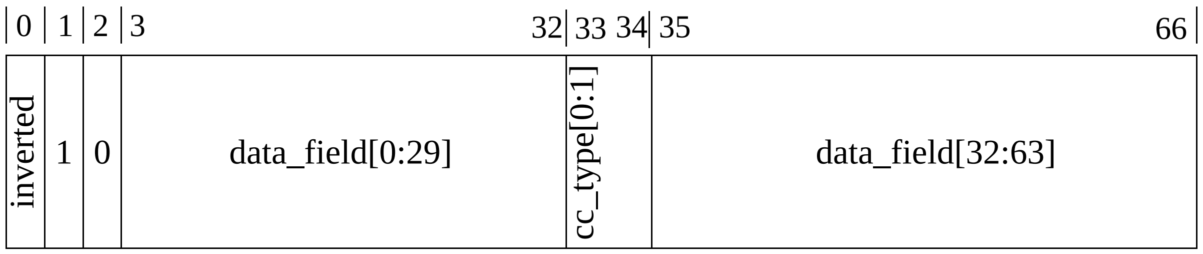


Figure 5-3. General 64b/67b Control Codeword Format

The encoding of the control codeword functions are shown in Table 5-1

Table 5-1. Control Codeword function encoding

cc_type[0:1]	data_field[32:35]	Name	Description
0b00	0b0000 - 0b0011	Implementation Specific	Reserved for implementation specific purposes. The default power-up state of a processing element shall disable transmission and processing of implementation specific control codewords.
	0b0100 - 0b1010	Reserved	
	0b1011	Skip-Marker	A fixed value 67-bit control codeword used to mark the beginning of the Skip ordered sequence. The data field of the codeword has a disparity of 0.
	0b1100	Lane Check	Used to monitor lane bit error rate.
	0b1101	Descrambler Seed	The descrambler seed is used to initialize and/or check the state of the per lane descrambler.
	0b1110	Skip	A fixed value 67-bit codeword that can be added or removed from a Skip ordered sequence for clock compensation. The data field of the codeword has a disparity of 0.
	0b1111	Status/Control	The data_field contains the status/control data field.
0b01	see description	Control Symbol Begin (CSB)	data_field[0:63] contains Control Symbol[0:29] || 0b01 || 4 data characters
0b10	see description	Control Symbol End (CSE)	Data_field[0:63] contains Control Symbol[32:61] || 0b10 || 4 data characters
0b11	see description	Control Symbol End and Begin (CSEB)	Data_field[0:63] contains Control SymbolA[32:61] || 0b11 || Control SymbolB[0:29] || 0b00

Control codewords can be further sub-divided into two categories: Symbol Bearing and Non-Symbol Bearing. Symbol Bearing control codewords include: CSB, CSE and CSEB. Non-Symbol Bearing control codewords include: Skip-Marker, Lane Check, Descrambler Seed, Skip and Status/Control.

5.5.3.1 Skip-Marker Control Codeword

The Skip-Marker control codeword is used together with the Skip control codeword to provide clock compensation. The format of the Skip-Marker control codeword shall be as shown in Figure 5-4. The codeword data field has a disparity of 0 and a Hamming distance of 32 from the Skip control codeword. The codeword shall be transmitted only as part of a Skip ordered sequence. For more information, refer to Section 5.9.3, "Skip Ordered Sequence".

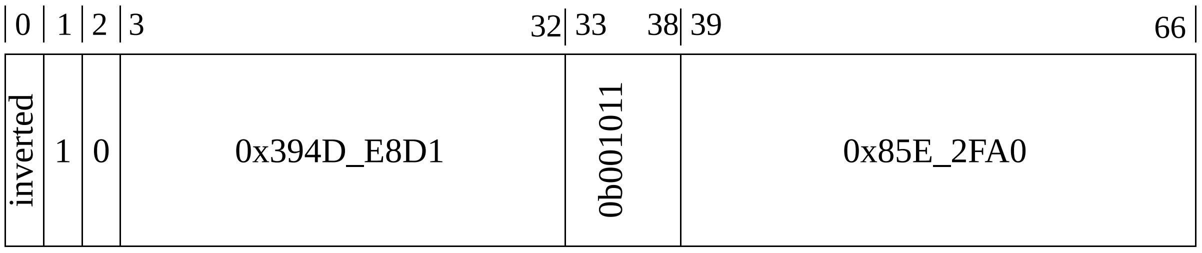


Figure 5-4. Skip-Marker Control Codeword Format

5.5.3.2 Lane-Check Control Codeword

The Lane-Check control codeword is used to monitor the BER of the lanes in the link. The Lane-Check control codeword is only intended to be used for bit error rate estimation and shall not influence or trigger error recovery. Its format shall be as shown in Figure 5-5. The codeword shall be transmitted only as a part of the Skip ordered sequence. For more information, refer to Section 5.9.3, "Skip Ordered Sequence".

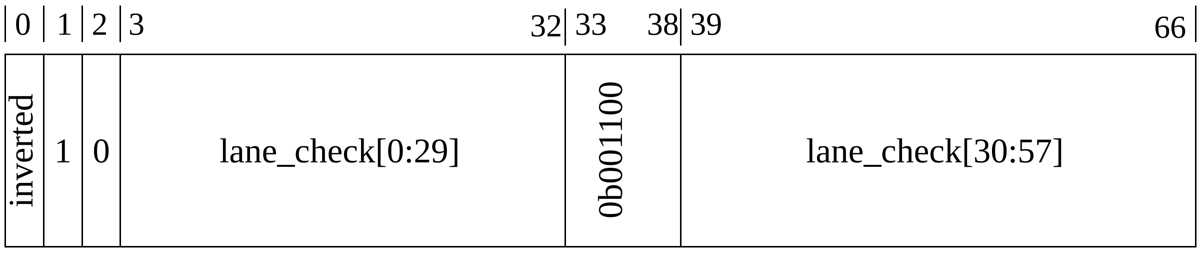


Figure 5-5. Lane-Check Control Codeword

The content of the Lane-Check control codeword lane check value field shall be as specified in Table 5-2.

Table 5-2. Lane_check field content

Location	Bit(s)	Definition
0-22	23 bits	The BIP-23 field contains the result of a bit interleaved parity calculation. Each bit in the BIP-23 field is an even parity calculation over all of the previous specified bits of a given lane since the previous Lane-Check control codeword, but not including the current Lane-Check control codeword, any Skip-Marker control codeword, or Skip control codeword. The Lane Check calculation is described in Section 5.5.6.
23-34	12 bits	Fixed value of 0b1011_0101_0101. Combined with the rest of the data_field of the Lane Check control codeword this results in a codeword with a data field disparity of 0.
35-57	23 bits	Bit-wise inversion of BIP-23, also referred to as the iBIP-23 field.

5.5.3.3 Descrambler Seed Control Codeword

The format of the Descrambler Seed control codeword shall be as shown in Figure 5-6. Seed codewords transmitted on lane k of a link shall contain the value of the lane k transmit scrambler (seed) which would have been used to scramble the Descrambler Seed Control Codeword if it was a codeword that is scrambled before transmission. The lane k transmit scrambler state (seed) is used to set or check the state of the lane k descrambler in the connected receiver at the time that the Seed codeword would have been descrambled. The Descrambler Seed control codeword is transmitted only as part of a Seed ordered sequence. For more information, refer to Section 5.9.1, "Seed Ordered Sequence".

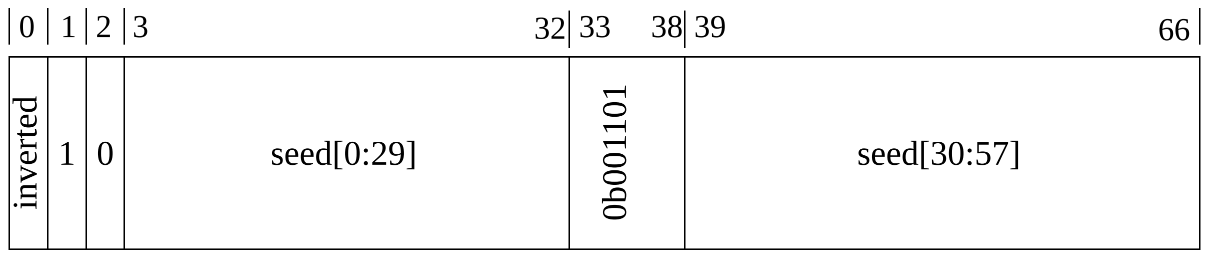


Figure 5-6. Descrambler Seed Control Codeword Format

5.5.3.4 Skip Control Codeword

The Skip control codeword is used together with the Skip-Marker control codeword to provide clock compensation. The format of the Skip control codeword shall be as shown in Figure 5-7. The codeword data field has a disparity of 0 and a Hamming distance of 32 from the Skip-marker control codeword. The codeword shall be transmitted only as part of a Skip ordered sequence. For more information, refer to Section 5.9.3, "Skip Ordered Sequence".

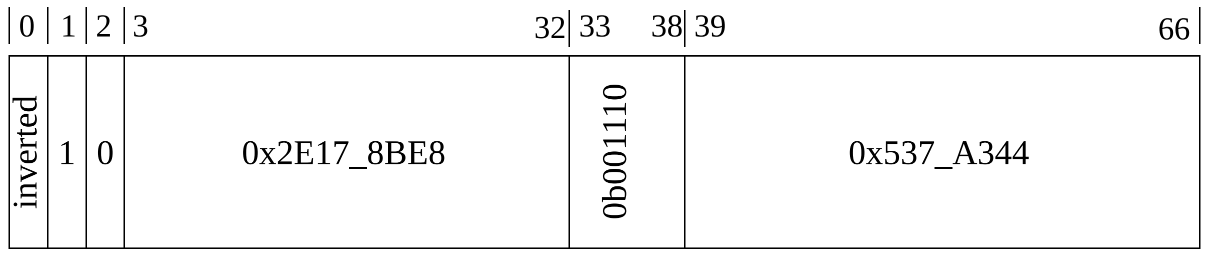


Figure 5-7. Skip Control Codeword Format

5.5.3.5 Status/Control Control Codeword

The Status/Control control codeword is use to communicate various link level information between two link partners, this includes link training control, link initialization and asymmetric link width control. The format of the Status/Control control codeword shall be as shown in Figure 5-8. The codeword shall be transmitted only as part of a Status/Control ordered sequence. For more information, refer to Section 5.9.2, "Status/Control Ordered Sequence".

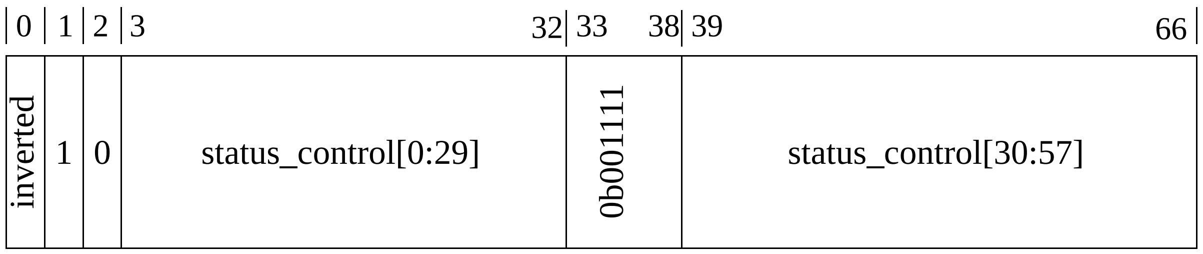


Figure 5-8. Status/Control Control Codeword Format

The content of the status_control field shall be as specified in Table 5-3

Table 5-3. Status_control field content

Location	Bit(s)	Scope	Definition
0-7	8 bits	Port[1]	Port number. The number of the port within the device to which the lane is assigned.
8-11	4 bits	Lane[3]	Lane number The number of the lane within the port to which the lane is assigned.
12	1 bit	Port[1]	Remote training support Indicates whether the port supports control of per lane transmit equalization by the lane receivers in the connected port. 0b0 - The port does not support control of its transmit equalization by the connected port. 0b1 - The port supports control of its transmit equalization by the connected port.
13	1 bit	Port[1]	Retraining enabled Indicates whether the port is allowed to enter retraining mode, based on the register value of the “10G Retraining Enable” field described in Section 7.6.9. 0b0 - Retraining mode is not enabled. 0b1 - Retraining mode is enabled.
14	1 bit	Port[1]	Asymmetric mode enabled Indicates whether the port is allowed to enter asymmetric mode, based on the register value of the “Asymmetric modes enabled” field described in Section 7.6.14. 0b0 - Asymmetric mode is not enabled. 0b1 - Asymmetric mode is enabled.
15	1 bit	Port[1]	Port initialized Indicates the initialization status of the port. The value and meaning of this bit transmitted on all lanes of a port shall be the same as that of the port’s state machine variable port_initialized.
16	1 bit	Port[1]	Transmit 1x mode Indicates when the port is transmitting in 1x symmetric mode. 0b0 - The port is not transmitting in 1x mode. The state machine variable max_width != 1x. 0b1 - The port is transmitting in 1x symmetric mode. The state machine variable max_width = 1x.

Table 5-3. Status_control field content

Location	Bit(s)	Scope	Definition
17-19	3 bits	Port[1]	Receive width The width at which the port is currently receiving control symbols and packets (destriping width) 0b000 - None 0b001 - 1x mode, lane 0 0b010 - 2x mode 0b011 - 4x mode 0b100 - 8x mode 0b101 - 16x mode 0b110 - 1x mode, lane 1 0b111 - 1x mode, lane 2 The receive width field shall retain the value it held prior to the Port Initialization State Machine entering the 1x_RECOVERY, 2x_RECOVERY, or Nx_RECOVERY states for the duration of those recovery states.
20-22	3-bits	Asym. Port[2]	Receive lanes ready The value of the field shall indicate the lanes being received by the port as indicated by the lanes for which lane_ready is asserted, lanes beyond max_width shall not be considered ready for this purpose 0b000 - No lanes ready 0b001 - lane_ready[0] 0b010 - lane_ready[0] & lane_ready[1] 0b011 - lane_ready[0] & lane_ready[1] & ... & lane_ready[3] 0b100 - lane_ready[0] & lane_ready[1] & ... & lane_ready[7] 0b101 - lane_ready[0] & lane_ready[1] & ... & lane_ready[15] 0b110 - 0b111 - reserved
23	1 bit	Lane[3]	Receive lane ready The value and meaning of this bit transmitted on lane k shall be the same as that of the lane's state machine variable lane_ready[k]
24	1 bit	Lane[3]	Lane trained Indicates the training status of the lane. The value and meaning of this bit transmitted on lane k shall be the same as that of the port's state machine variable lane_trained[k]
25-27	3 bits	Asym. Port[2]	Receive width command The port receiving the command shall attempt to switch to the receive width specified in the command received on lane 0. 0b000 - hold current receive width 0b001 - receive in 1x mode 0b010 - receive in 2x mode 0b011 - receive in 4x mode 0b100 - receive in 8x mode 0b101 - receive in 16x mode 0b110-0b111 - reserved
28	1 bit	Asym. Port[2]	Receive width command ACK 0b0 - No command status 0b1 - Command executed
29	1 bit	Asym. Port[2]	Receive width command NACK 0b0 - No command status 0b1 - Command not executed

Table 5-3. Status_control field content

Location	Bit(s)	Scope	Definition
30-32	3 bits	Asym. Port[2]	Transmit width request A request that the port receiving this field change its transmit width to the width specified in the request. This field, in conjunction with the "Transmit width request pending" bit transmitted by the port receiving the transmit width request, is used to send, acknowledge, and control the flow of transmit width requests across the link. The receiver shall only see the transmit width request on lane 0 as a valid request. 0b000 - no request (hold current transmit width) 0b001 - request transmit 1x mode 0b010 - request transmit 2x mode 0b011 - request transmit 4x mode 0b100 - request transmit 8x mode 0b101 - request transmit 16x mode 0b110-0b111 - reserved
33	1 bit	Asym. Port[2]	Transmit width request pending This bit is used by a port to acknowledge the receipt of a transmit width request. This bit, in conjunction with the "Transmit width request" field transmitted by the connected port, is used to acknowledge and control the flow of transmit width requests across the link. 0b0 - No request pending 0b1 - Request pending
34	1 bit	Asym. Port[2]	Transmit Status/Control ordered sequences Indicates the required rate of Status/Control ordered sequences on a link. The value and meaning of this bit transmitted on a lane shall be the same as that of the lanes state machine variable xmt_sc_seq.
35-38	4 bits	Lane[3]	Transmit equalizer tap When the transmit equalizer command is tap specific, this field contains the number of the equalizer tap to which the tap specific command shall be applied. The tap number is encoded as a signed 2's complement 4-bit integer. 0b0000 - Tap 0 0b0001 - Tap +1 0b0010 - Tap +2 0b0011 - Tap +3 0b0100 - Tap +4 0b0101 - Tap +5 0b0110 - Tap +6 0b0111 - Tap +7 0b1000 - Tap -8 0b1001 - Tap -7 0b1010 - Tap -6 0b1011 - Tap -5 0b1100 - Tap -4 0b1101 - Tap -3 0b1110 - Tap -2 0b1111 - Tap -1 When the transmit equalizer update command is not tap specific, the field shall have the value 0b0000 and shall be ignored.

Table 5-3. Status_control field content

Location	Bit(s)	Scope	Definition
39-41	3 bits	Lane[3]	Transmit equalizer command 0b000 - Hold/No command 0b001 - Decrement (make more negative by one step) the coefficient of the specified tap. 0b010 - Increment (make more positive by one step) the coefficient of the specified tap. 0b011-0b100 - Reserved 0b101- Initialize - Set the tap coefficients to their INITIALIZE state as defined Clause 72.6.10.4.2 of IEEE Standard 802.3-2008 (part 5). 0b110 - Preset coefficients - Set the coefficient of tap 0 to its maximum value and the coefficients of all other taps to 0 as specified in Clause 72.6.10.4.1 of IEEE Standard 802.3-2008 (part 5). 0b111 - Indicate specified tap implementation status. When Transmit equalizer command are 0b001, 0b010 or 0b111; the Transmit equalizer tap value shall contain the value of the Tap; for other commands the Transmit equalizer tap value shall be 0b0000
42-44	3 bits	Lane[3]	Transmit equalizer status 0b000 - Not updated - No command is pending or the status of the current command has not been determined. 0b001 - Updated - The tap specific command has been executed and the tap is at neither its minimum nor maximum value. 0b010 - Minimum - Either the tap specified tap decrement command has been executed and the tap is now at its minimum value or the specified tap was already at its minimum value. 0b011 - Maximum - Either the tap specific tap increment command has been executed and the tap is now at its maximum value or the specified tap was already at it maximum value. 0b100 - Preset or Initialize command executed. 0b101 - Reserved. 0b110 - Specified tap not implemented. 0b111 - Specified tap implemented.
45	1 bit	Port[1]	Retrain grant When the Status/Control control codeword is formed, the value of this bit shall be the same as the value of the port’s state machine variable retrain_grnt.
46	1 bit	Port[1]	Retrain ready When the Status/Control control codeword is formed, the value of this bit shall be the same as the value of the port’s state machine variable retrain_ready.
47	1 bit	Port[1]	Retraining When the Status/Control control codeword is formed, the value of this bit shall be the same as the value of the port’s state machine variable retraining.
48	1 bit	Port[1]	Port Entering Silence 0b0 - The port is transmitting normally. 0b1 - All lanes of the port are going to enter the Silence state.
49	1 bit	Lane[3]	Lane Entering Silence 0b0 - The lane is transmitting normally. 0b1 - The lane is going to enter the Silence state based on asymmetric mode operation, based on port width downgrade in symmetric mode or other events that makes the lane enter silence i.e. after a keep alive event.
50-57	8 bits	-	Reserved

[1] The “Port” scope means that the transmitting port shall transmit the same value on all lanes and that the receiving port shall only use the values received on lane 0 or lane R if operating in redundant mode.

[2] The “Asym. Port” scope means that the transmitting port shall transmit the same value on all lanes and that the receiving port shall only use the values received on lane 0. The value transmitted is only used when the link is operating in asymmetric mode as defined in Section 5.17, "Asymmetric Operation"

[3] The “Lane” scope means that the transmitted value is lane specific.

5.5.3.6 CSB Control Codeword

The format of the CSB control codeword shall be as shown in Figure 5-9.

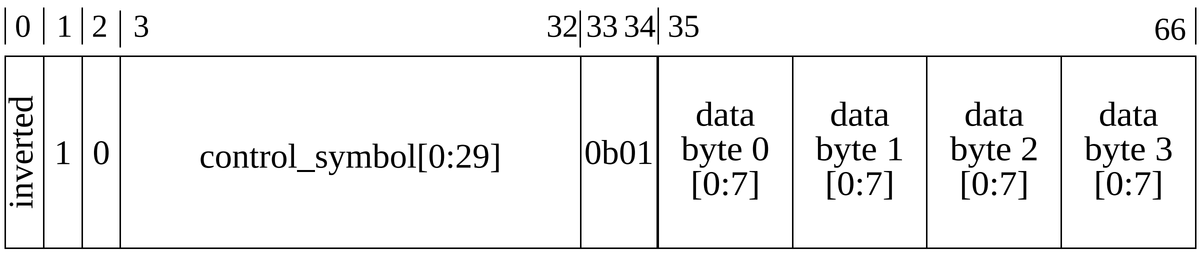


Figure 5-9. CSB Control Codeword Format

The CSB control codeword encodes 8 sequential bytes beginning with 4 data bytes followed by 4 bytes containing the first 32 bits of a control symbol, control_symbol[0:31], as shown in Figure 5-10. The 8 bytes (64 bits) are encoded by first performing a 32-bit rotation on the bits being encoded, inserting the rotated 64 bits into the control codeword, and then setting the control codeword cc_type[0:1] to 0b01, which overwrites the 2-bit alignment field of the control symbol. Notice how the logical layout differ from the actual codeword layout by showing the logical ordering of the data with oldest data showing to the left.

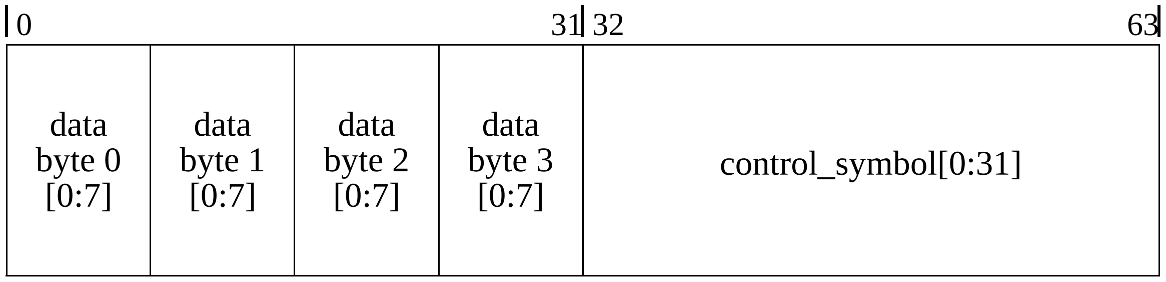


Figure 5-10. Logical Layout of CSB Control Codeword

When there is no data for encoding in codeword bits [35:66], the bits shall be loaded with bytes of 0x00, which when scrambled become pseudo-random data bytes.

5.5.3.7 CSE Control Codeword

The format of the CSE control codeword shall be as shown in Figure 5-11.

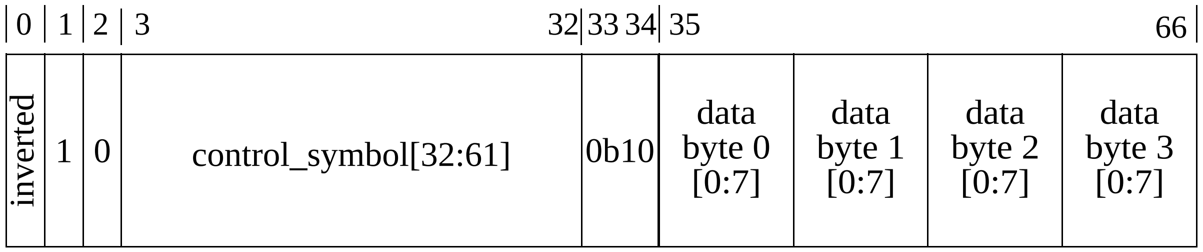


Figure 5-11. CSE Control Codeword Format

The CSE control codeword encodes 8 sequential bytes beginning with 4 bytes containing the second 32 bits of a control symbol, control_symbol[32:63], followed by 4 data bytes as shown in Figure 5-12. The 8 bytes (64 bits) are encoded by inserting the 64 bits directly into the control codeword data_field and then setting control codeword cc_type[0:1] to 0b10 which overwrites the 2-bit alignment field of the control symbol.

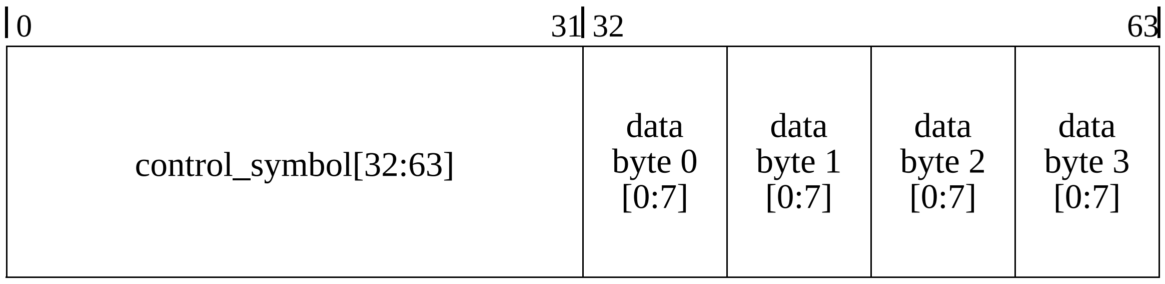


Figure 5-12. Logical Layout of CSE Control Codeword

When there is no data for encoding in codeword bits [35:66], the bits shall be loaded with bytes of 0x00, which when scrambled become pseudo-random data bytes.

5.5.3.8 CSEB Control Codeword

The format of the CSEB control codeword shall be as shown in Figure 5-13.

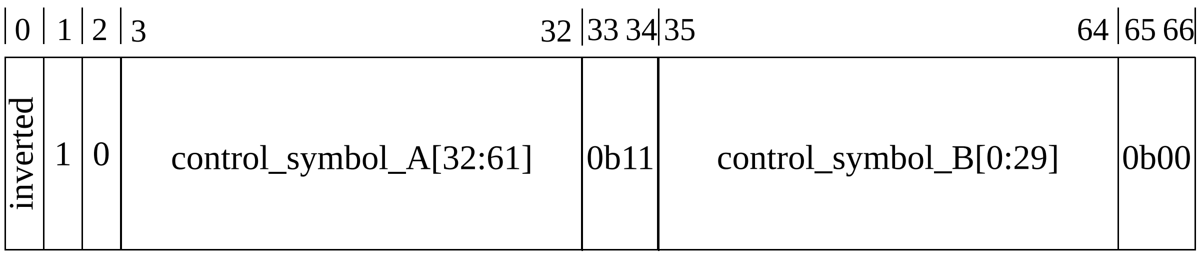


Figure 5-13. CSEB Control Codeword Format

The CSEB control codeword encodes 8 sequential bytes beginning with 4 bytes containing the last 32 bits of a control symbol (control_symbol_A) followed by 4 bytes containing the first 32 bits of the immediately following control symbol (control_symbol_B). The control symbol alignment fields are overwritten with the cc_type (2'b11) in the case of control_symbol_A and with zeros (2'b00) in the case of control_symbol_B.

5.5.4 Scrambling

Scrambling smooths the spectrum of a port’s transmit signal and reduces the spectrum’s peak values. This is most important when long strings of the same character or of a repeating character sequence are transmitted. The result is a reduction in the amount of electromagnetic interference (EMI) generated by the link and easier design of adaptive equalizer training algorithms.

5.5.4.1 Scrambling Rules

A portion of all data codewords and of some control codewords are scrambled before transmission on an LP-Serial link. Bits [0:2] of a codeword (inverted, !type and type) shall never be scrambled.

Scrambling and descrambling shall be done on a per lane basis. At any specific time, each of the lanes scramblers shall have a different state.

Scramblers and descramblers shall step and generate 64 bits of scrambling sequence for every codeword except Skip control codewords. Scramblers and descramblers shall neither step nor generate any scrambling sequence bits for Skip control codewords.

Codewords shall be scrambled according to the following rules:

Codeword bits [3:66] of all data codewords shall be scrambled.

Codeword bits [3:32] and [35:66] of all control codewords with codeword bits[33:34] != 0b00 shall be scrambled.

Control codewords with codeword bits[33:34] = 0b00 shall not be scrambled.

Therefore the CSB, CSE and CSEB control codewords shall be scrambled and all other control codeword types shall not be scrambled.

The codeword data_field shall be scrambled from left to right beginning with codeword bit [3] and ending with codeword bit [66]. When scrambling a control codeword with codeword bits[33:34] != 0b00, the scrambler bits that would be used to scramble codeword bits[33:34] shall be ignored and not used. When a control codeword with codeword bits[33:34] = 0b00 is encountered, all 64 scrambler bits shall be ignored and not used. The scrambler shall still step 64 bits for each codeword except for Skip control codewords, even if only some or none of the 64 bits are used for scrambling.

A pseudo-random sequence generated by a Fibonacci (external) form linear feedback shift register (LFSR) generator using the primal generating polynomial $x^{58}+x^{39}+1$ shall be used for scrambling. The output of the scrambler shall be the output of the register holding x^{58}, the oldest and most significant state bit.

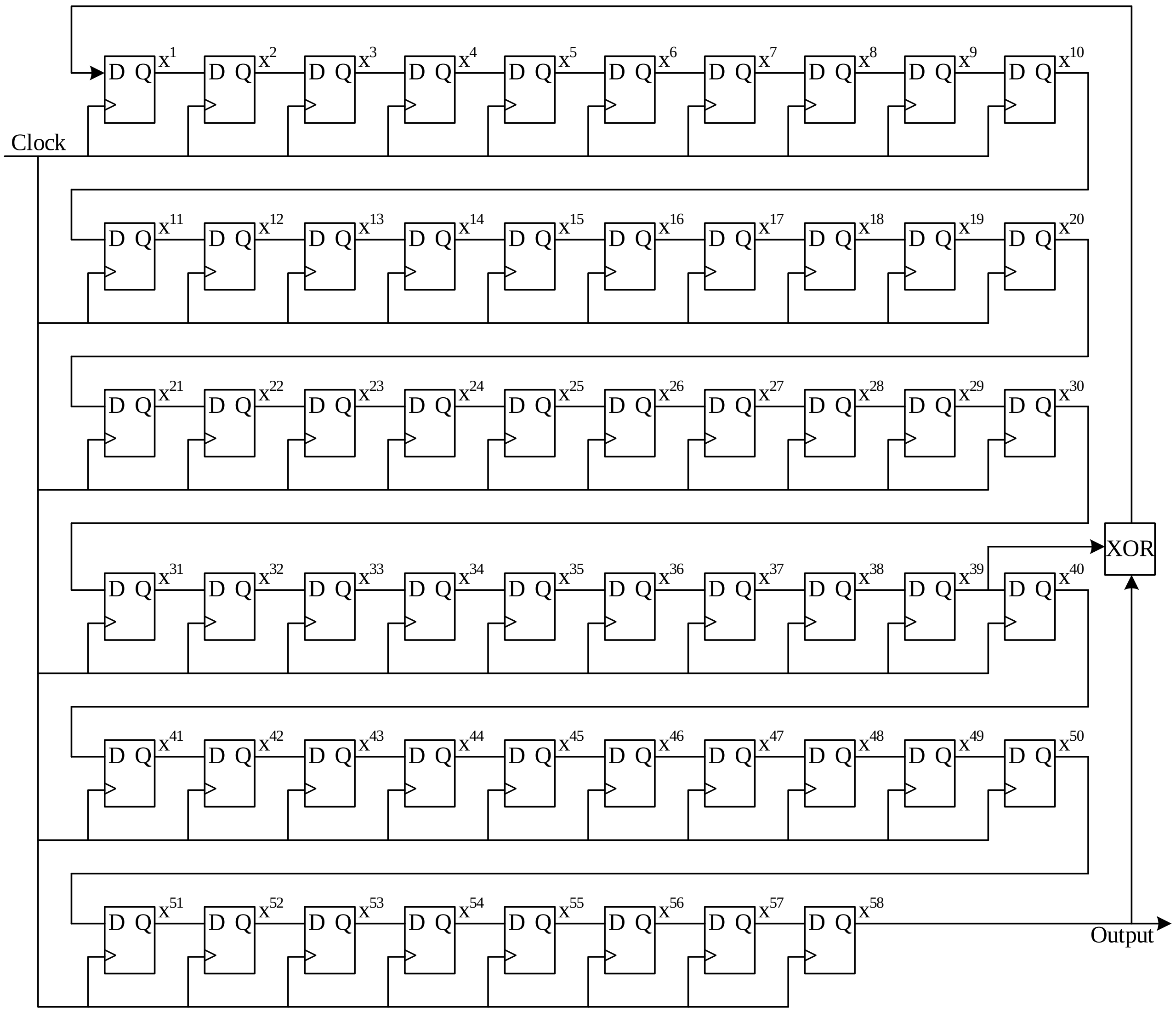

Clock
D Q
x^{1} x^{2} x^{3} x^{4} x^{5} x^{6} x^{7} x^{8} x^{9} x^{10}
x^{11} x^{12} x^{13} x^{14} x^{15} x^{16} x^{17} x^{18} x^{19} x^{20}
x^{21} x^{22} x^{23} x^{24} x^{25} x^{26} x^{27} x^{28} x^{29} x^{30}
x^{31} x^{32} x^{33} x^{34} x^{35} x^{36} x^{37} x^{38} x^{39} x^{40}
x^{41} x^{42} x^{43} x^{44} x^{45} x^{46} x^{47} x^{48} x^{49} x^{50}
x^{51} x^{52} x^{53} x^{54} x^{55} x^{56} x^{57} x^{58}
XOR
Output

Figure 5-14. Scrambling Sequence Generator

To minimize any correlation between lanes when a port is transmitting on multiple lanes, the scrambling sequence applied to a given output lane of the port shall be offset from the scrambling sequence applied to any other output lane of the port by at least 512 bits. If separate scrambling sequence generators are used for each lane, the offset requirement can be achieved by initializing the scramblers to the values specified in Table 5-4, which provide an offset of 512.

Table 5-4. Scrambler Initialization Values

Lane	Initialization value $[x^1-x^{58}]$
0	0x1ec_f564_79a8_b120
1	0x1a1_af7d_7264_5f9e
2	0x2ef_2b62_302b_d094
3	0x14a_da90_3a26_68aa
4	0x1fe_d572_55e7_da1d
5	0x283_8ff2_c69c_3618
6	0x3bc_111d_3429_3ece
7	0x1c0_3994_44ae_4a2b
8	0x09f_ebc2_faee_77fb
9	0x239_7200_3b8e_9cff
10	0x00a_45db_c14e_f218
11	0x36d_3a42_6876_e9c4
12	0x2c1_a537_55d7_8dea
13	0x2b9_e833_dd9d_6b34
14	0x1cb_c090_ab7f_79b3
15	0x26f_aa25_7342_3ae5

5.5.4.2 Descrambler Synchronization

Each lane descrambler shall synchronize itself to the data stream it is receiving by using Seed ordered sequences. For more information about Seed ordered sequences, refer to Section 5.9.1, "Seed Ordered Sequence".

The first Descrambler Seed control codeword of the Seed ordered sequence shall be used to re-initialize the state of the descrambler. Refer to 5.5.3.3 Descrambler Seed Control Codeword for the mapping of seed[0:57] to the descrambler coefficients. A Descrambler Seed control codeword shall be determined to be the first in a Seed ordered sequence if the preceding codeword is not a Descrambler Seed control codeword. Based on this definition, only the first Seed ordered sequence of a sequence of consecutive Seed ordered sequences will trigger descrambler re-initialization.

After a lane descrambler has been re-initialized, the second Descrambler Seed control codeword of the Seed ordered sequence shall be used to verify descrambler synchronization. The descrambler verification is done by comparing the received seed value from the Descrambler Seed control codeword with the current seed value of the descrambler and if they match then the descrambler is determined to be “in sync”; otherwise, the descrambler shall be determined to be “out of sync”.

A sync test can fail because of either a loss of descrambler sync or a data transmission error(s) in either of the codewords of the Seed ordered sequence.

If a descrambler sync test fails while receive_enable is asserted, an initialized port shall immediately enter the Input Error-stopped state if it is not already in that state and resynchronize the descrambler. An uninitialized port shall ignore scrambler sync failures. For more information about error recovery processes, refer to Section 6.13, "Error Detection and Recovery". All control symbols and packets received while the lane descrambler of a 1x link or the lane descrambler of any lane carrying control symbols and packets in a multi-lane mode is out of sync shall be ignored and discarded. The cause field in the packet-not-accepted control symbol issued by the port on entering the Input Error-stopped state due to a sync check failure shall indicate “loss of descrambler sync”.

To ensure that a port that may have lost descrambler sync can recover descrambler sync before it is sent a link maintenance protocol link-request control symbol, a LP-Serial port that is operating with IDLE3 shall transmit a Seed ordered sequence before every transmitted link-request control symbol. For reset-device or reset-port where four link-request control symbols are transmitted, each of the four link-request control symbols shall be preceded by a Seed ordered sequence. The Seed ordered sequence shall be transmitted in parallel on each of the N active lanes of a link operating in Nx mode, and shall immediately precede the link-request control symbol. If the link is operating in 1x mode, the last codeword of the Seed ordered sequence is immediately followed by the first codeword of the link-request. If the link is operating in Nx mode, the last column of the Seed ordered sequence is immediately followed by the column containing the codewords of the link-request.

5.5.5 Selective Codeword Inversion

Selective codeword inversion is used to bound the running disparity of the signal transmitted over each lane of a LP-Serial link that uses 64b/67b encoding.

5.5.5.1 Selective Codeword Inversion Rules

Selective codeword inversion shall be applied to the signal transmitted over each lane of an LP-Serial link according to the following rules.

1. The transmitter shall start with 0 as the initial value for the running disparity calculation for each lane.
2. After each codeword is formed and if appropriate, scrambled, compute the disparity of the resulting codeword.
3. If the signs of the codeword disparity and the running disparity of the lane over which the codeword will be transmitted are different, the codeword shall be transmitted as is without inversion. The running disparity at the end of the codeword shall be the running disparity at the beginning of the codeword plus the disparity of the codeword.

4. If the signs of the codeword disparity and the running disparity of the lane over which the codeword will be transmitted are the same, invert bits [0,3:66] of the codeword and transmit the resulting codeword. The running disparity at the end of the codeword shall be the running disparity at the beginning of the codeword minus the disparity of the codeword before inversion.
5. At the receiver, invert bits [0,3:66] of each received codeword if codeword bit[0] = 0b1 (the codeword was inverted before transmission).

5.5.6 Lane Check Calculation

A lane BIP-23 field is carried in each Lane Check control codeword. This allows an accurate and fast measure of the bit error ratio of a specific lane. This information is used to update error counters; however, no state machines use this information.

Each Lane Check control codeword has two Bit Interleaved Parity (BIP) fields, BIP-23 and iBIP-23. iBIP-23 is a bit-wise inversion of BIP-23 to simplify error detection and to maintain a data field disparity value of 0. The BIP-23 field contains the result of a BIP calculation. Each bit in the BIP-23 field is an even parity calculation over a set of specified bits from each codeword on a given lane, as specified in Table 5-5. The first codeword in the transmitters and receivers BIP calculation shall be a Lane Check control codeword, modified such that the BIP field is all 0. Note that the iBIP field of the Lane Check control codeword is kept unchanged. The BIP calculation shall exclude Skip-Marker and Skip control codewords. The BIP calculation shall be done on non-inverted codewords. On the transmit side the codeword values in the BIP calculation shall not have selective codeword inversion applied. On the receive side, the BIP calculation shall use the uninverted, original values of codewords that had selective codeword inversion applied for transmission.

The Lane Check control codeword is used to implement a parity check over intervals of codewords that begin with a Lane Check control codeword, known as the 'start' Lane Check, and end with the next Lane Check control codeword known as the 'finish' Lane Check. The 'finish' Lane Check for the preceding codeword sequence is the 'start' Lane Check for the next codeword sequence. When a 'finish' Lane Check control codeword is received, the BIP-23 value calculated by the receiver shall be checked against the BIP-23 field of the 'finish' Lane Check control codeword. If the two values are different, the receiver’s Lane n Status 0 CSRs “8b/10b decoding errors” field shall be incremented by 1. Note that the receivers calculated BIP-23 value used in the comparison shall exclude the 'finish' Lane Check control codeword. The checking procedure is sufficient to detect errors in the 'start' Lane Check non-BIP-23 fields, the codeword sequence, and the 'finish' Lane Check BIP-23 field. Bit errors in one 'start' to 'finish' Lane Check interval do not influence bit error detection in subsequent 'start' to 'finish' Lane Check intervals.

Table 5-5 shows the contribution of the bits from the 67-bit codeword to each BIP-23 bit. As an example, BIP-23 bit 1 is generated by XORing bits 0, 22, and 45 from all previous 67-bit codewords starting with the last Lane Check control codeword. BIP-23 bit 0 and bit 22 include one less bit from each 67-bit codeword.

An example BIP calculation is displayed in Figure 5-15. Codewords starting from the Lane Check control codeword up until the last codeword before a Skip-marker are included in the BIP-23 calculation. The BIP-23 field of the 'start' Lane Check control codeword is all zeros in the calculation.

Table 5-5. BIP-23 Calculation

BIP-23 bit number	Assigned 67-bit word bits
0	21, 44
1	0, 22, 45
2	1, 23, 46
3	2, 24, 47
4	3, 25, 48
5	4, 26, 49
6	5, 27, 50
7	6, 28, 51
8	7, 29, 52
9	8, 30, 53
10	9, 31, 54
11	10, 32, 55
12	11, 33, 56
13	12, 34, 57
14	13, 35, 58
15	14, 36, 59
16	15, 37, 60
17	16, 38, 61
18	17, 39, 62
19	18, 40, 63
20	19, 41, 64
21	20, 42, 65
22	43, 66

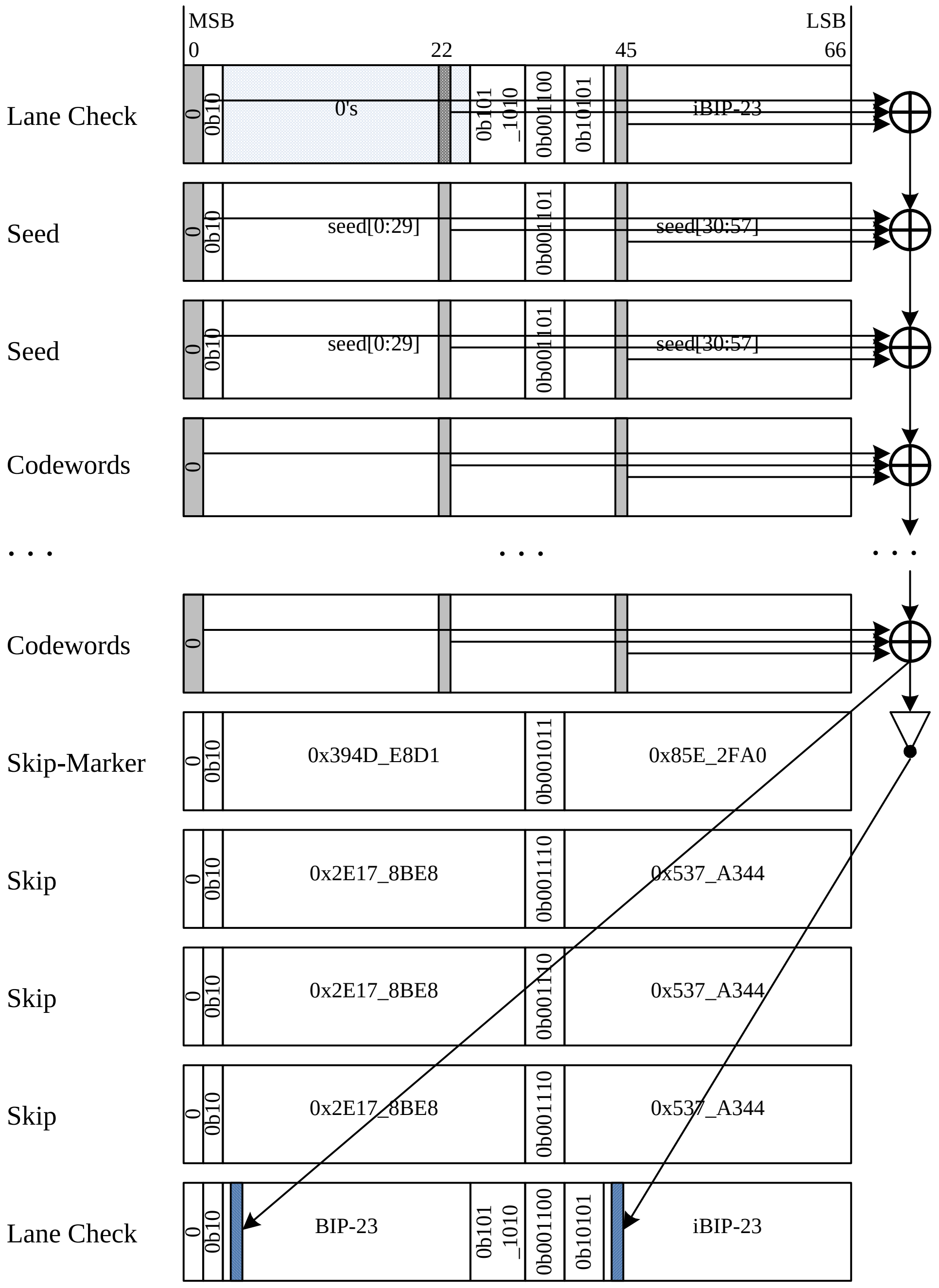


Figure 5-15. Example of Calculation for Bit 1 of BIP-23

The BIP-23 is calculated over the fully encoded, scrambled, but not selectively inverted codewords.

When calculating the BIP-23 value for the first Lane Check control codeword to be transmitted on a link after silence the value does not matter and the BIP-23 can either be set to a fixed value or be calculated over and an unspecified number of codewords preceding the Skip ordered sequence containing the Lane Check control codeword.

The BIP-23 value of the first error free received Lane Check control codeword that is recognized by the receiver after achieving codeword lock shall be not be checked.

5.5.7 Transmission Order

The parallel 67-bit codeword output of the encoder shall be serialized and transmitted with bit 0 transmitted first and a sequential bit ordering towards bit 66. This is shown in Figure 5-16.

Figure 5-16 gives an overview of a set of characters passing through the encoding, serializing, transmission, deserializing, and decoding processes. The left side of the figure shows the transmit process of encoding a character stream using 64b/67b encoding and the 67-bit serialization. The right side shows the reverse process of the receiver deserializing and using 64b/67b decoding on the received codewords.

The dotted line shows the functional separation between the PCS, that provides 67-bit codewords, and the PMA Layer that serializes the codewords.

The drawing also shows on the receive side the bits of the type field containing the pattern that is used by the receiver to establish 67-bit codeword boundary synchronization.

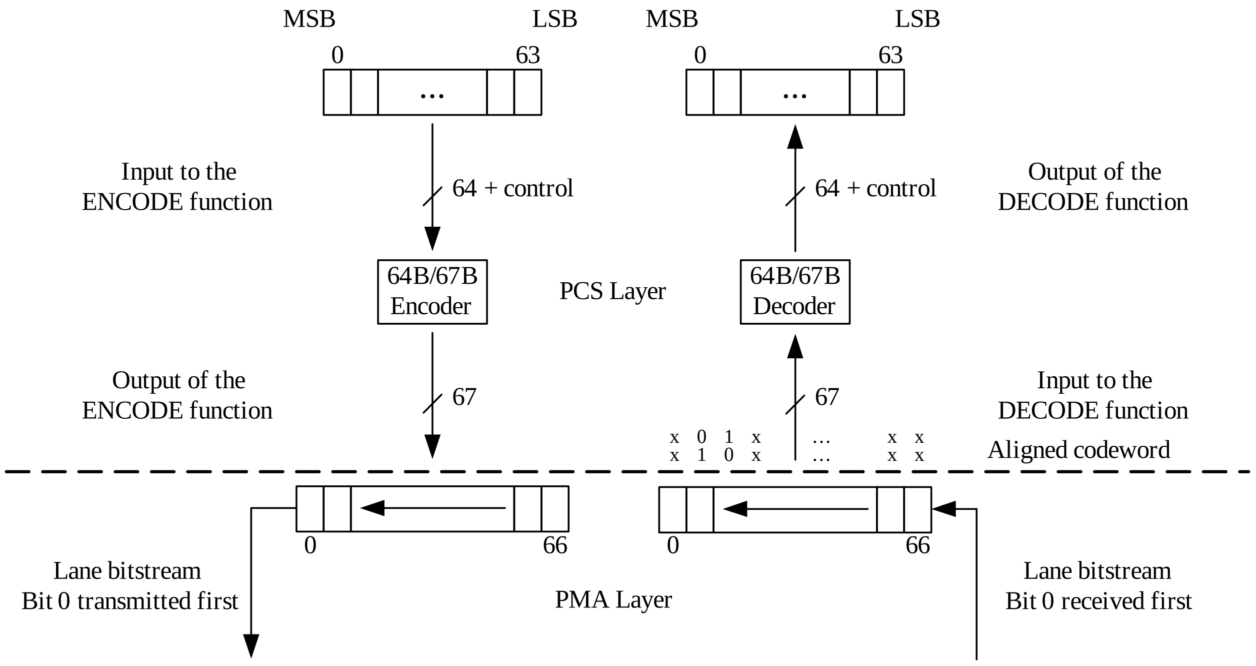

MSB
LSB
0
63
...
MSB
LSB
0
63
...
Input to the ENCODE function
64 + control
64 + control
Output of the DECODE function
64B/67B Encoder
PCS Layer
64B/67B Decoder
Output of the ENCODE function
67
67
Input to the DECODE function
x 0 1 x ... x x
x 1 0 x ... x x
Aligned codeword
0
66
0
66
Lane bitstream Bit 0 transmitted first
PMA Layer
Lane bitstream Bit 0 received first

Figure 5-16. Lane Encoding, Serialization, Deserialization, and Decoding Process

5.6 Packet Transmission Rules

The packet format as defined in Chapter 2, "Packets" shall be augmented by an additional 32-bit link CRC-32 for transport over a link for which 64b/67b encoding is employed. The link CRC-32 shall be generated by the transmitting port and shall be used to check for packet corruption by the receiving port after which the link CRC-32 shall be discarded. The link CRC-32 shall be computed over the packet data, excluding the ackID field, including the CRC-16 and if present the additional embedded CRC-16 and the 16-bit pad. The link CRC-32 shall use the polynomial specified in IEEE 802.3 - 2008 (Section 1) clause 3.2.9. The link CRC-32 shall be computed as described in Section 5.6.1.

The length of packets shall be an integer multiple of 8 bytes. The length includes the link CRC-32. Packets that are not an integer multiple of 8 bytes in length shall be padded with 4 bytes of 0x00 such that the padded length is an integer multiple of 8 bytes. The padding bytes of 0x00 shall be placed after the link CRC-32.

The padding bytes allow the CRC-32 check to be performed on an 8 byte boundary. Corrupt padding bytes may or may not cause a CRC-32 error to be detected, depending upon the implementation. Corruption of the 2 pad bytes inserted after the final CRC-16 of a packet shall cause a CRC-32 error to be detected.

The maximum length of a packet shall be 288 bytes: the 280 byte maximum packet length calculated in Section 2.5 plus 4 bytes for the additional CRC-32. (288 bytes is an integer multiple of 8 bytes.)

Packets whose transmission is terminated before the end of the packet shall be terminated at an 8 byte boundary relative to the beginning of the packet except in error recovery cases. Receivers shall not assume that a packet whose transmission is terminated before the end of the packet includes the link CRC-32.

5.6.1 Link CRC-32 Code

The IEEE 802.3 - 2008 (Section 1) clause 3.2.9 polynomial:

$$X^{32}+X^{26}+X^{23}+X^{22}+X^{16}+X^{12}+X^{11}+X^{10}+X^{8}+X^{7}+X^{5}+X^{4}+X^{2}+X+1$$

shall be used to generate the link CRC-32 for packets. The value of the link CRC-32 shall be initialized to 0xFFFF_FFFF (all logic 1s) at the beginning of each packet. For the link CRC-32 calculation, the six ackID bits are treated as logic 0s. As an example, a 32-bit wide parallel calculation is described in the equations in Table 5-6. Equivalent implementations of other widths can be employed.

Table 5-6. Parallel Link CRC-32 Equations

Check Bit	e00	e01	e02	e03	e04	e05	e06	e07	e08	e09	e10	e11	e12	e13	e14	e15	e16	e17	e18	e19	e20	e21	e22	e23	e24	e25	e26	e27	e28	e29	e30	e31
C00	x	x	x	x	x		x	x	x								x				x		x	x			x					
C01		x	x	x	x	x		x	x	x								x				x		x	x			x				
C02	x		x	x	x	x	x		x	x	x								x				x		x	x			x			
C03		x		x	x	x	x	x		x	x	x								x				x		x	x			x		
C04			x		x	x	x	x	x		x	x	x								x				x		x	x			x	
C05	x			x		x	x	x	x	x		x	x	x								x				x		x	x			x

Table 5-6. Parallel Link CRC-32 Equations (Continued)

Check Bit	e00	e01	e02	e03	e04	e05	e06	e07	e08	e09	e10	e11	e12	e13	e14	e15	e16	e17	e18	e19	e20	e21	e22	e23	e24	e25	e26	e27	e28	e29	e30	e31
C06	x		x	x						x	x		x	x	x		x				x			x					x	x		
C07		x		x	x						x	x		x	x	x		x				x			x					x	x	
C08	x		x		x	x						x	x		x	x	x		x				x			x					x	x
C09	x		x		x	x		x	x				x	x		x		x		x	x		x									x
C10	x		x		x	x		x		x				x	x				x			x	x				x					
C11		x		x		x	x		x		x				x	x				x			x	x				x				
C12			x		x		x	x		x		x				x	x				x			x	x				x			
C13	x			x		x		x	x		x		x				x	x				x			x	x				x		
C14	x	x			x		x		x	x		x		x				x	x				x			x	x				x	
C15		x	x			x		x		x	x		x		x				x	x				x			x	x				x
C16		x			x			x			x	x		x		x	x			x			x	x	x		x	x	x			
C17			x			x			x			x	x		x		x	x			x			x	x	x		x	x	x		
C18	x			x			x			x			x	x		x		x	x			x			x	x	x		x	x	x	
C19	x	x			x			x			x			x	x		x		x	x			x			x	x	x		x	x	x
C20	x			x	x	x	x	x				x			x	x	x	x		x			x					x	x		x	x
C21	x		x	x		x							x			x		x	x				x				x		x	x		x
C22			x					x	x					x					x	x	x		x				x	x		x	x	
C23	x			x					x	x					x					x	x	x		x				x	x		x	x
C24			x	x			x	x	x	x	x					x	x					x		x	x		x		x	x		x
C25		x	x				x			x	x	x						x			x			x	x	x	x	x		x	x	
C26			x	x				x			x	x	x						x			x			x	x	x	x	x		x	x
C27	x	x	x				x	x				x	x	x			x			x	x			x		x		x	x	x		x
C28	x				x		x						x	x	x		x	x				x	x	x	x				x	x	x	
C29	x	x				x		x						x	x	x		x	x				x	x	x	x				x	x	x
C30				x	x			x							x	x			x	x	x		x		x	x					x	x
C31	x	x	x	x		x	x	x								x				x		x	x			x						x

where:

C00–C31	contents of the new check symbol

e00–e31	contents of the intermediate value symbol
	e00 = d00 XOR c00
	e01 = d01 XOR c01
	through
	e31 = d31 XOR c31

d00–d31	contents of the next 32 bits of the packet
c00–c31	contents of the previous check symbol
	assuming the pipeline described in Figure 5-17

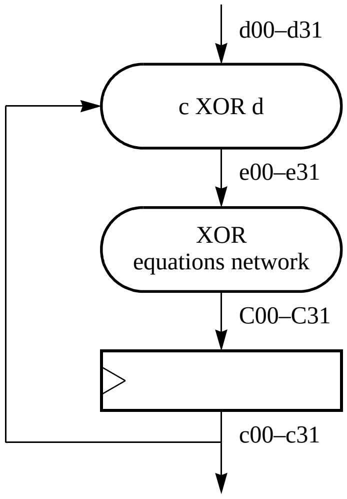


Figure 5-17. Link CRC-32 Generation Pipeline

5.7 Packet Delimiting and Alignment

Packets shall be delimited for transmission by two control symbols, a “start of packet delimiter” and an “end of packet delimiter”. The control symbol containing the start of packet delimiter shall immediately precede the first byte of the packet or the first byte of an embedded control symbol. With one exception stated below, the control symbol containing the end of packet delimiter shall immediately follow the last byte of the packet or the last byte of an embedded control symbol.

5.7.1 Packet Start Delimiter

The beginning of packet shall be delimited by a start-of-packet control symbol.

After 64b/67b encoding, the last half of the start-of-packet control symbol shall share a CSE control codeword with the first 4 data bytes of the packet, or a CSEB control codeword with the first half of an embedded control symbol.

5.7.2 Packet Termination Delimiters

A packet shall be terminated in one of the following ways.

The end of a complete packet is delimited with

an end-of-packet control symbol or

a start-of-packet control symbol that also marks the beginning of the next packet.

The packet is canceled by

a restart-from-retry control symbol,

a stomp control symbol or

any link-request control symbol.

After 64b/67b encoding, and with one exception stated below, the first half of the packet terminating delimiter control symbol shall share a CSB control codeword with the last 4 data bytes of the packet, or a CSEB control codeword with the last half of an embedded control symbol.

If a packet is canceled with a link-request control symbol, a Seed ordered sequence shall be transmitted between the end of the packet and the link-request control symbol <u>on all lanes</u>. Bytes of 0x00 shall be used to pad the space between the end of the packet and the beginning of the Seed ordered sequence. The link-request control symbol shall immediately follow the Seed ordered sequence. The link-request control symbol shall begin transmission in Lane 0 of a multi-lane port. Since the link-request control symbol also functions as the “restart-from-error” control symbol, the transmission of the Seed ordered sequence is needed to allow the receiver's descrambler(s) to recover synchronization with the input data stream(s) in the case the receiving port has lost descrambler sync.

5.8 Control Symbol Transmission Rules

Links using the 64b/67b line code shall use Control Symbol 64 as defined in Section 3.3. Each control symbol shall be encoded using a pair of contiguous control codewords such that half of the control symbol is in each of the two control codewords. Isolated control symbols shall be encoded using a CSB control codeword followed by a CSE control codeword. A sequence of n contiguous control symbols shall be encoded with one CSB control codeword followed in order by n-1 CSEB control codewords and one CSE control codeword.

Control symbols embedded in a packet shall align to an 8-byte boundary relative to the beginning of the packet.

5.9 Ordered Sequences

To facilitate error detection, the Seed, Status/Control, Lane Check, Skip Marker and Skip control codewords shall be transmitted only in “ordered sequences”. Each ordered sequence is comprised of a sequence of two or more control codewords with fixed ordering and with sufficient known content and redundancy to detect corruptions in a received ordered sequence.

When an ordered sequence is transmitted on a link direction operating in a multi-lane mode, the ordered sequence shall be transmitted in parallel on all active lanes with the sequence beginning in the same column on all active lanes and ending in the same column on all active lanes. The result being that when transmitted, the ordered sequence appears on the link as columns of codewords, one column for each codeword in the sequence. Ordered sequences shall not be striped. While the same ordered sequence of codewords is sent in parallel on all active lanes, the values carried in the codewords of the sequence can differ from lane to lane.

The reception of an incorrectly formed or corrupted ordered sequence on any active receive lane shall be handled by the receiver as an input error.

With one exception defined in Section 5.7.2, control codewords with a CC_type value of 0 shall not be transmitted within delimited packets. Control codewords with a CC_type value of 0 shall not interrupt control symbol transmission, as defined in Section 5.8. Control codewords with a CC_type of 0 shall always be transmitted as columns, where every codeword in the column has the same value in data_field[32:35].

For forward compatibility and robustness, a column of control codewords with a CC_type of 0 and data_field[32:35] value that the port does not understand shall be handled as follows. A column of control codewords with a reserved data_field[32:35] value shall not be processed further, and shall not cause an error to be detected. A column of control codewords with an unsupported implementation specific data_field[32:35] value shall not be processed further, and shall not cause an error to be detected. A column of control codewords with a supported implementation specific data_field[32:35] value shall not be processed further while processing of implementation specific control codewords is disabled.

5.9.1 Seed Ordered Sequence

The Seed ordered sequence shall be comprised of two sequential Descrambler Seed control codewords. Sending the seed in successive codewords allows the descrambler to be initialized with the first codeword and then checked with the second codeword. If either codeword is corrupted, the type or format bits of the codewords will not match or the seed in the second codeword will not match the seed generated by the descrambler from the seed in the first codeword and allowing the corruption to be easily detected.

Table 5-7. Seed ordered sequence

Seed ordered sequence
Seed control codeword
Seed control codeword

The Seed ordered sequence shall be transmitted before each link-request control symbol. As part of the IDLE3 sequence the Seed ordered sequence shall be transmitted at least once for every 52 codewords transmitted per lane. For more information on the idle sequence, refer to Section 5.10, "Idle Sequence".

5.9.2 Status/Control Ordered Sequence

The Status/Control ordered sequence shall be comprised of two sequential Status/Control control codewords. The content of the two Status/Control control codewords in a Status/Control ordered sequence transmitted on a specific lane shall be identical. The sequential transmission of two identical Status/Control control codewords per lane allows corruption in either of the two words to be easily detected by simple comparison.

Table 5-8. Status/Control ordered sequence

Status/Control sequence
Status/Control control codeword
Status/Control control codeword

A Status/Control ordered sequence shall be considered valid only if the two consecutive Status/Control control codewords are identical and the variable lane_sync[k] is asserted, where k is the lane the codewords are received on.

When a link is operating in a multi-lane mode, the Status/Control ordered sequence shall be used by the receiver to align the active lanes.

Before the output enables of the transmitter are deasserted, the IDLE3 sequence shall be transmitted for a period of time which allows 8 Status/Control ordered sequences to be sent with the “Port Entering Silence” and “Lane Entering Silence” indications set according to what triggered the output enables to be deasserted. After the first Status/Control ordered sequence that signals Entering Silence the transmitter shall not be transmitting longer than a period of 512 codewords, which is sufficient to transmit more than the required 8 Status/Control ordered sequences. Implementations should complete packets which are currently in transmission before starting transmission of the IDLE3 sequence. For more information on the idle sequence, refer to Section 5.10, "Idle Sequence".

The Status/Control ordered sequence shall be transmitted at least once for every 256 codewords transmitted per lane when operating in asymmetric mode and the variable xmt_sc_seq is set (See Section 5.19.1.3). As part of the IDLE3 sequence the Status/Control ordered sequence shall be transmitted at least once for every 49 codewords transmitted per lane. Under no circumstances shall the Status/Control ordered sequence be transmitted more often than once for every 18 codewords transmitted per lane.

5.9.3 Skip Ordered Sequence

When transmitted, the Skip ordered sequence shall be comprised of a Skip-marker control codeword immediately followed by three Skip control codewords, then followed in order by a Lane Check control code and a Seed ordered sequence. The transmitted Skip ordered sequence is shown in Table 5-9.

Table 5-9. Skip ordered sequence

Skip ordered sequence
Skip-Marker control codeword
Skip control codeword
Skip control codeword
Skip control codeword
Lane Check control codeword
Seed control codeword
Seed control codeword

When received, a Skip ordered sequence shall be comprised of a Skip-marker control codeword followed in order by one or more Skip control codewords, a Lane Check control codeword and a Seed ordered sequence. Any deviation from this order indicates that an error has occurred. The Seed ordered sequence shall be used to verify, and if necessary, to reset the descrambler synchronization.

The Skip ordered sequence is used for clock compensation. A retimer may add one Skip control codeword or delete one Skip codeword from a Skip ordered sequence to compensate for the difference between its input and output baud rates. If a retimer adds a Skip codeword to the sequence, it shall add the codeword immediately after the Skip-Marker codeword. On links operating in a multilane mode, Skip codewords shall be added or deleted in columns.

A port shall transmit a Skip ordered sequence on each of its active output lanes at least once for every 5000 codewords transmitted per lane by the port. Since a packet or delimited control symbol may not be interrupted by an ordered sequence, it is recommended that a port transmit a Skip ordered sequence on each of its active output lanes at least once for every 4096 codewords transmitted per lane by the port.

5.10 Idle Sequence

The idle sequence defined for 64b/67b encoded links is referred to as IDLE3. The IDLE3 sequence is a sequence of codewords transmitted by a LP-Serial port on each of its active output lanes when the port is not initialized, and when the port is initialized and there are no packets or control symbols to transmit. The IDLE3 sequence enables a LP-Serial receiver to acquire and retain bit, codeword and lane alignment, as well as supporting clock compensation.

When idle is transmitted by a LP-Serial port, an idle sequence shall be transmitted on each of the port’s active output lanes. Ports operating in Nx mode shall not stripe the idle sequence across the active lanes; there is an idle sequence for each of the N lanes.

An uninitialized LP-Serial port (state variable port_initialized not asserted) shall continuously transmit an idle sequence on all active output lanes. An initialized LP-Serial port (state variable port_initialized asserted) shall transmit an idle sequence on each of its active output lanes when there is nothing else to transmit. An idle sequence may not be inserted in a packet or control symbol. An initialized LP-Serial port that becomes uninitialized while transmitting a packet or control symbol may transmit several codewords per lane of packet and/or control symbol before beginning the transmission of an idle sequence.

On links operating in 1x mode, the first codeword of the idle sequence shall immediately follow the last codeword of the preceding control symbol. When a link is operating in Nx mode, the first column of N idle codewords shall immediately follow the column containing the last codeword of the preceding control symbol.

5.10.1 Idle Sequence 3 (IDLE3)

The IDLE3 Sequence shall be a continuous sequence of “ordered sequences” and data codewords containing pseudo-random data. Data codewords containing pseudo-random data will be referred to as “pseudo-random data codewords”. The exact sequence of “ordered sequences” and pseudo-random data codewords comprising a specific IDLE3 sequence is implementation dependent.

The IDLE3 sequence shall be generated according to the following rules.

1. Pseudo-random data codewords shall be generated by first forming data codewords filled with bytes of 0x00 and then scrambling those data codewords with the transmitter’s per lane scrambler(s).
2. An ordered sequence, once begun, shall be transmitted in its entirety.
3. When IDLE3 sequence is being transmitted:

    A Status/Control ordered sequence shall be transmitted once every 18 to 53 codewords transmitted per lane.

    A Seed ordered sequence shall be transmitted at least once every 53 codewords transmitted per lane.

    The Seed ordered sequences transmitted as part of Skip ordered sequences can be counted as part of the Seed ordered sequences that are transmitted to meet the minimum Seed ordered sequence transmission rate.

    The spacing between Status/Control ordered sequences should be pseudo-random to minimize peaks in the spectrum of the transmitted signal.

4. If a port is transmitting in 1x mode:

The IDLE3 sequence may begin with a pseudo-random data codeword or any ordered sequence.

An ordered sequence may begin at any codeword boundary that is not interior to another ordered sequence.

The IDLE3 sequence may be terminated after the last codeword of an ordered sequence or after any data codeword.

5. If a port is transmitting in a multi-lane mode:

   The IDLE3 sequence begins at a column boundary

   An IDLE3 sequence shall be transmitted in parallel on all active lanes

   The sequence of ordered sequences and pseudo-random data codewords shall be exactly the same for all active lanes.

   The IDLE3 sequence and each ordered sequence in the IDLE3 sequence shall begin in the same column for all active lanes and shall end in the same column for all active lanes, i.e. the IDLE3 sequence and all ordered sequences in the IDLE3 sequence are aligned across the active lanes.

   The IDLE3 sequence may begin with a pseudo-random data codeword or any ordered sequence, subject to the following restriction on Status/Control ordered sequence spacing.

   Status/Control ordered sequences shall be separated by at least 16 non-Status/Control codeword columns, regardless of whether the last Status/Control ordered sequence was part of this IDLE3 sequence or a previous IDLE3 sequence. (This requirement is to ensure that Status/Control ordered sequence columns that are used for lane alignment are separated by a minimum of 16 codeword columns.)

   An ordered sequence may begin at any codeword column boundary that is not interior to another ordered sequence.

   The IDLE3 sequence may be terminated after the last codeword of an ordered sequence or after any pseudo-random data codeword.

5.10.2 Idle Sequence 3 Generation

A primitive polynomial of at least 7th degree is recommended as the generating polynomial for the pseudo-random sequence that is used in the generation of the idle sequence. The polynomials $x^7 + x^6 + 1$ and $x^7 + x^3 + 1$ are examples of primitive 7th degree polynomials which may be used as generator polynomials. The pseudo-random sequence generator is clocked (generates a new pseudo-random sequence value) once per idle sequence codeword (column). Five of the pseudo-random sequence generator state bits may be selected to generate the pseudo-random value for Status/Control ordered sequence spacing. The selection of the state bits and their weighting has an impact on the distribution of values for Status/Control ordered sequence spacing.

One way to achieve the random spacing requirements from Section 5.9.1 is to repeatedly send one of the following sequences depending on the need to transmit Skip ordered sequences:

1. Sequence starting with a Seed ordered sequence:

   One Seed ordered sequence

   A pseudo-random number between 14 and 45 of Data codewords

   A Status/Control ordered sequence

2. Sequence starting with a Skip ordered sequence:

   One Skip ordered sequence

   A pseudo-random number between 9 and 40 of Data codewords

   A Status/Control ordered sequence

The above sequences provide the required spacing of 16 to 47 codewords between Status/Control ordered sequences. Transmission of Skip ordered sequences should be minimized, as completion of the seven codeword Skip sequence delays the start of packet transmission.

It should be kept in mind that transmitting Skip ordered sequences too often can impact link efficiency if packets arrive for transmission when a Skip ordered sequence is being transmitted. This is because the Skip ordered sequence is seven codewords long and it has to be completed before packet transmission can start.

Figure 5-18 shows an example circuit illustrating how this may be done. The clock ticks whenever a codeword or column is transmitted. Send_idle is asserted whenever an idle sequence begins and stays asserted until the idle sequence ends. The equations for start_sc, start_skip and start_seed indicate the states in which to start the transmission of either Status/Control, Skip or Seed ordered sequences respectively. The example circuit will provide a pseudo random number between 17 and 32 of codewords before the first Status/Control ordered sequence, and a pseudo-random number between 16 and 47 of codeword between any of the following Status/Control ordered sequences. Any equivalent method is acceptable.

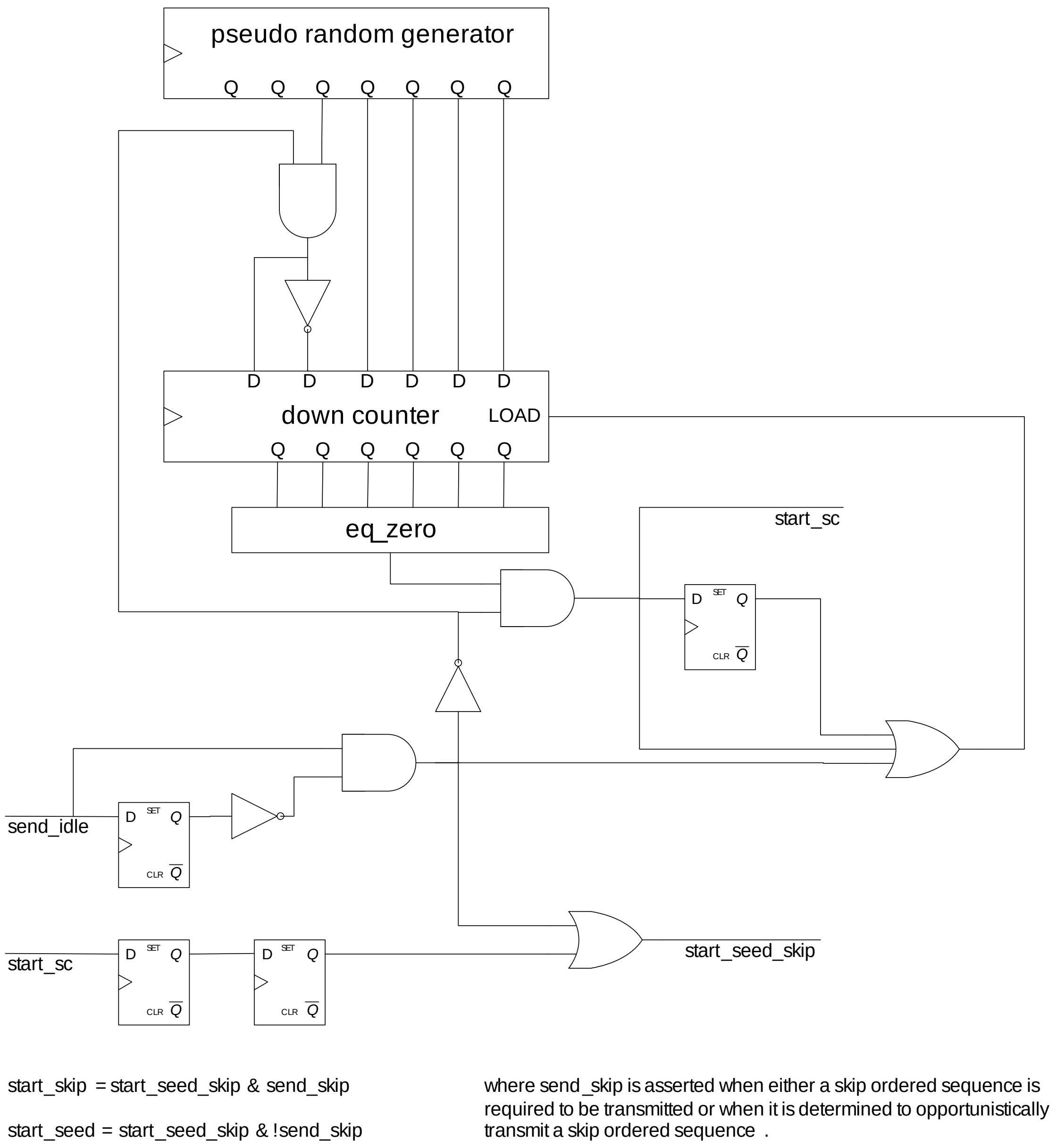

pseudo random generator
Q Q Q Q Q Q Q
D D D D D D
down counter
LOAD
Q Q Q Q Q Q
eq_zero
start_sc
D SET Q
CLR Q
send_idle
start_sc
start_seed_skip
start_skip = start_seed_skip & send_skip
start_seed = start_seed_skip & !send_skip
where send_skip is asserted when either a skip ordered sequence is required to be transmitted or when it is determined to opportunistically transmit a skip ordered sequence .

Figure 5-18. Example of a Pseudo-Random Idle Codeword Generator

5.11 Adaptive Equalization

At baud rates of 10 Gbaud and higher, the transmission characteristics of channels from a few centimeters to 1 meter long vary so much that per lane adaptive equalization is required to achieve reliable communication over the full range of channel lengths. Adaptive equalization can be located in the lane transmitter, the lane receiver, or both. Channels whose length does not exceed a few tens of centimeters may require only fixed or manually adjusted equalization for reliable communication.

Training of per lane adaptive receive equalization is controlled by the lane receiver. Reliable training of adaptive receive equalization requires the transmission by the connected lane transmitter of a signal suitable for training. The mechanism and algorithms used to train the adaptive receive equalization are implementation specific and beyond the scope of this specification.

Training of per lane adaptive transmit equalization is also controlled by the connected lane receiver, or some mechanism that has access to measurements of the quality of the signal received by the lane receiver after the signal has been processed by any receive equalization present in the receiver and that has control of the adaptive transmit equalizer settings. Control of the adaptive transmit equalizer setting by the connected receiver requires a method for the lane receiver to send adjustment commands to, and obtain status from, the adaptive equalization in the connected lane transmitter. For LP-Serial links operating with IDLE3, the adjustment commands and status are carried in-band in the per lane training signals transmitted by the connected ports. For interoperability, the training signal, the format of transmit equalizer training commands and status, and the transmit equalizer structure, need to be standardized. The mechanism and algorithms used to train the adaptive transmit equalization are implementation specific and beyond the scope of this specification.

Note that similar standardization is defined for IDLE2 in Section 4.7.4.1.4, "IDLE2 CS Field Use".

5.11.1 Lane Training/Retraining

Two modes are specified for adjustment of per lane adaptive equalization, training and retraining.

Training mode is used when a link is initially brought up (starting from the Port_Initialization state machine SILENT state). It is also used when a port encounters a problem from which it cannot easily recover and therefore attempts recovery by completely reinitializing the link. In training mode, the per lane adaptive equalization is trained with no assumed knowledge of the lane’s characteristics.

Retraining mode is used when the adaptive equalization has been initially trained and some equalization adjustment is needed to correct for unacceptable amounts of drift over time in the characteristics of the lane transmitters, the channels, and/or the lane receivers. This mode is provided to allow a port to “fine-tune” the adaptive equalization settings of the lanes it is receiving in less time than would be required to train the lanes from scratch. Retraining uses the IDLE signal as the training signal and starts with the current equalizer settings.

5.11.2 Ports Operating at 10.3125 and 12.5 Gbaud

Two sets of electrical specifications are specified for LP-Serial links operating at 10.3125 and 12.5 Gbaud, a short run set and a long run set. LP-Serial ports that support the long run electrical specification are referred to as “long run” ports. LP-Serial ports that support only the short run electrical specification are referred to as “short run” ports.

Long run ports shall support both long run and short run electrical specifications and both long run training as specified in Section 5.11.2.1, and short run training as specified in Section 5.11.2.2. Short run ports may support short run training. Ports that support training shall support retraining.

5.11.2.1 Long run 10.3125 and 12.5 Gbaud training

Long run ports shall support adaptive equalizer training using the training frame structure, DME encoding of the control channel and the protocol specified in Clauses 72.6.10.1 through 72.6.10.2, and their sub-clauses, the Frame lock and Coefficient update state machines specified in Clause 72.6.10.4, and the related variables defined in Clause 72.6.10.3 and the transmitter output waveform and waveform requirements specified in Clauses 72.7.1.10 and 72.7.1.11 of IEEE Standard 802.3-2008 (Part 5) for 10GBASE-KR. Training using this training frame structure and protocol, these state machines and transmitter output waveform requirements is referred to as “DME training”.

The transmitter output waveform specified in Clause 72.1.1.11 requires a 3-tap transversal transmit equalizer, or its equivalent, to meet the long-run transmitter output waveform specifications. A 3-tap transversal filter is shown in Figure 72-11 of Clause 72.7.1.10 of IEEE Standard 802.3-2008 (Part 5). As specified in IEEE Standard 302.3-2008, the training frame structure supports control of 3 transmit equalizer taps. Using the same allocation of four bits per tap, two bits for command and two bits for status, reserved bits in the training frame Control Channel allow expansion to control a total of 7 transmit equalizer taps.

The DME training signal is mostly 10.3125 or 12.5 Gbaud pseudo-random data and is suitable for training any adaptive receive equalization present in the receiver.

Implementations shall implement a timeout on DME transmit emphasis requests. The timeout shall be controlled by Port n Link Timers Control CSRs Emphasis Command Timeout field.

5.11.2.2 Short run 10.3125 and 12.5 Gbaud training

The short run electrical specification is taken from Annex 83A (XLAUI/CAUI) of IEEE Standard 802.3-2008. It requires a 2-tap transversal transmit equalizer, or its equivalent, to meet the short run transmitter output waveform specifications. But it does not require that the short run transmit equalizer’s tap settings be adjustable –

there is no method specified for control of the transmit equalizer’s tap setting by the connected lane receiver.

LP-Serial short run 10.3125 and 12.5 Gbaud lane transmitters may implement adaptive transmit equalization. Short run ports that implement adaptive transmit equalization shall support adjustment of each lane’s adaptive transmit equalizer’s settings by the connected lane receiver.

The short run training signal shall be IDLE3. Since IDLE3 is composed of 64b/67b data codewords containing pseudo-random data and ordered sequences, each comprised of multiple 64b/67b control codewords, short run training is also referred as “codeword training” or “CW training”.

CW training commands shall be carried in the “Transmit equalizer command” and “Transmit equalizer tap” fields of Status/Control control codewords. The “CW training” transmit equalizer commands are a super-set of the coefficient update commands provided in the “Coefficient update” field of the long run DME training frame with the exception that only one tap-specific coefficient update command can be issued at a time. The super-set approach was done to ease the design on the adaptive equalizer training mechanism by minimizing the differences between the “DME” and “CW” coefficient update command sets.

When the “Transmit equalizer command” is tap specific, the tap number shall be specified in the “Transmit equalizer tap” field; otherwise, the “Transmit equalizer tap” field shall be set to 0x0 on transmission and ignored on reception. The Transmit equalizer tap” fields support a total of 16 transmit equalizer taps (-8 to +7) to allow the use of CW training at baud rates greater than 10.3125 and 12.5 Gbaud that may require a transmit equalizer with more than three taps.

CW training command status shall be carried in the “Transmit equalizer status” field of Status/Control control codewords. The CW training command status values are a super-set of the coefficient update status values in the status report field on the DME training frame again with the exception that the status for only one tap-specific command can reported at a time. The super-set approach was done to ease the design on the adaptive equalizer training mechanism by minimizing the differences between the “DME” and “CW” coefficient update status sets.

CW training shall use the following handshake protocol:

1. A transmit equalizer command shall be considered asserted when the value of the “Transmit equalizer command” field is different from “hold”; otherwise, no transmit equalizer command is asserted.
2. The “Transmit equalizer tap” field shall be considered an extension of the “Transmit equalizer command” field. The “Transmit equalizer tap” field shall have the appropriate value and the value shall not change while a “Transmit equalizer command” is asserted.
3. The assertion of a “Transmit equalizer command” shall occur only when the value of “Transmit equalizer status” is “not_updated”.

4. Once a “Transmit equalizer command” is asserted, it shall remain asserted and unchanged in value until the value of “Transmit equalizer status” is different from “not_updated” or the command has been asserted for the timeout period configured in Port n Link Timers Control CSRs Emphasis Command Timeout field. At that point, the command shall be de-asserted within 5 usec of whichever of the two events occurred first. If the command timed out, the command shall be deasserted for the timeout period configured in the Emphasis Command Timeout field.
5. Once a “Transmit equalizer status” value other than “not_updated” is asserted, it shall remain asserted until the value of “Transmit equalizer command” returns to “hold”.

5.11.2.3 10.3125 and 12.5 Gbaud retraining

Retraining shall use the same mechanisms and protocol as specified in Section 5.11.2.2 for short run training.

The need for retraining a trained lane is indicated by the assertion of the lane’s lane_degraded signal. The lane_degraded signal for lane k shall be generated by the mechanism that controls the settings of the lane k adaptive equalization. It is assumed that this mechanism can monitor the quality of the signal received by the lane receiver after the signal has been processed by any receive equalization present in the receiver. If no such metric is available during normal data reception, the lane_degraded signals may be permanently de-asserted.

Retraining is enabled by the “10G Retraining enable” bit in the Port n Control 2 CSRs. Retraining shall occur only when the 10G Retraining enable bit is asserted and one or more of the lanes asserting lane_trained are also asserting lane_degraded. When this condition occurs, all lanes asserting lane_trained shall be retrained regardless of whether or not they are asserting lane_degraded. Retraining all of the trained lanes at once minimizes the number of times the link must be taken down for retraining.

To avoid interaction between retraining and changing the transmission width of one direction of a link when the link is operating in asymmetric mode, retraining and transmission width changes shall be serialized so that only one such operation can occur at a time.

5.12 LP-Serial Link Widths

LP-Serial links may have 1, 2, 4, 8, or 16 lanes per direction. All LP-Serial ports shall support operation on links with one lane per direction (1x mode) and may optionally support operation over links with 2, 4, 8 and/or 16 lanes per direction (respectively 2x mode, 4x mode, 8x mode and 16x mode). For example, a port that supports operation over 8 lanes per direction (8x mode) must also support operation over one lane per direction (1x mode) and may optionally also support operation over 2 and/or 4 lanes per direction (2x mode and/or 4x mode). The requirement that all LP-Serial ports support 1x mode is to ensure that any pair of LP-Serial ports that are capable of operating at the same baud rate also support a common link width over which they can always communicate with each other.

LP-Serial ports that support operation over two or more lanes per direction shall support 1x mode operation over two of those lanes, lane 0 and lane R (the redundancy lane). If the port supports operation over at most two lanes per direction (2x mode), lane R shall be lane 1. If the port supports operation over more than two lanes, lane R shall be lane 2. Requiring ports that support operation over links with two or more lanes per direction to also support 1x mode over two lanes per direction provides a redundant fallback capability that allows communication over the link at reduced bandwidth in the presence of lane failure, regardless of the lane that fails.

5.13 Transmission Rules

5.13.1 Order of Operation

The sequence of codewords containing packets and control symbols transmitted over a 64b/67b encoded LP-Serial link shall be as if they had been 64b/67b encoded, striped (if an Nx link), scrambled and selectively codeword inverted in that order regardless of the order in which these operation were actually performed.

5.13.2 1x Ports

A 1x LP-Serial port shall 64b/67b encode and transmit the character stream of control symbols and packets received from the upper layers in the order the characters were received from the upper layers. When neither control symbols nor packets are available from the upper layers for transmission, an idle sequence shall be fed to the input of the 64b/67b encoder for encoding and transmission.

On reception, the codeword stream is 64b/67b decoded and the resulting character stream of error-free control symbols and packets shall be passed to the upper layers in the order the characters were received from the link.

The data stream shall be scrambled before transmission and descrambled after reception as specified in Section 5.5.4.

Figure 5-19 shows an example of IDLE3 sequence, Control Symbol 64 and packet transmission on a 1x LP-Serial link.

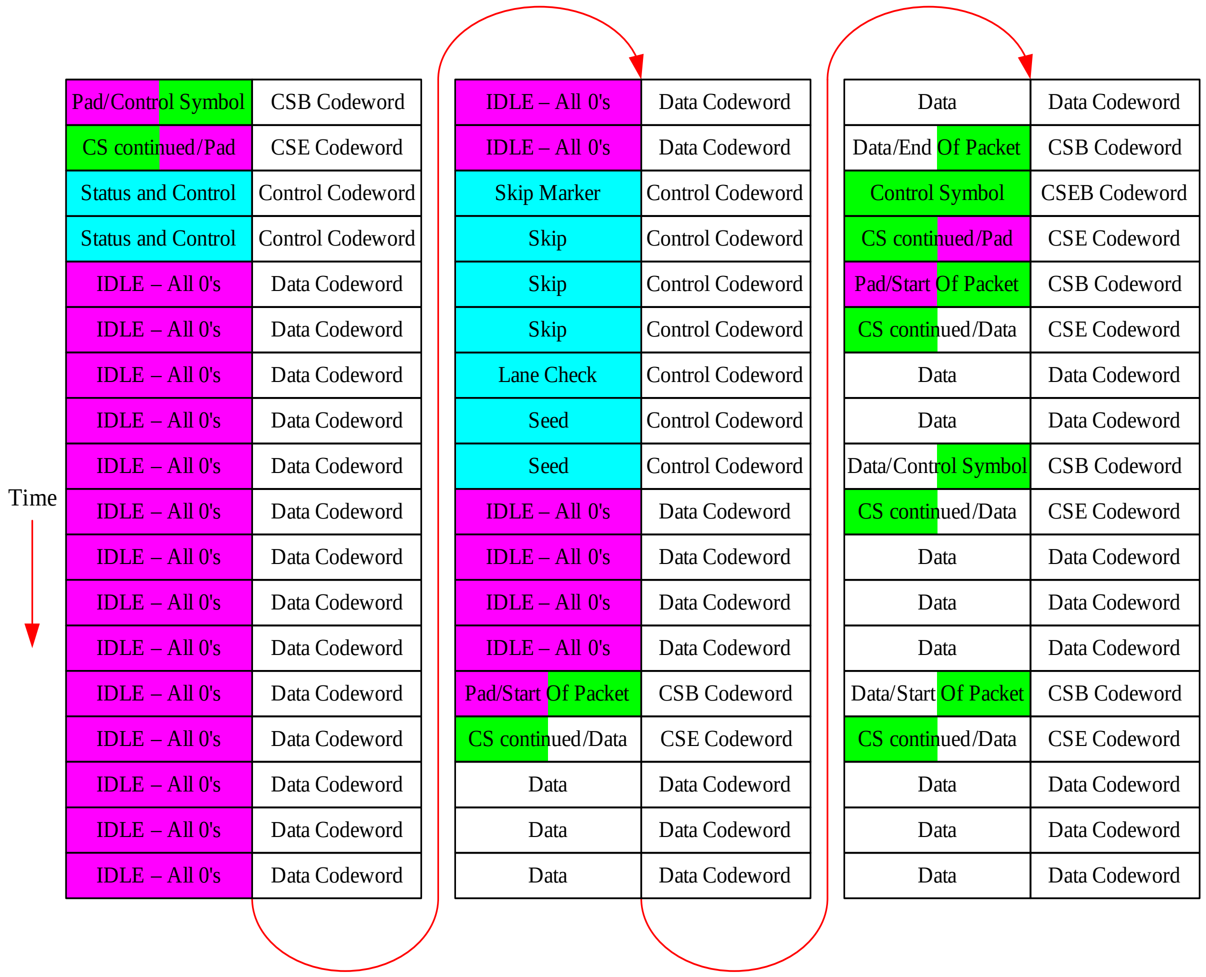


Figure 5-19. 1x Typical Data Flow with Control Symbol 64

5.13.3 Nx Ports Operating in 1x Mode

When a Nx port is operating in 1x mode, the character stream of control symbols and packets received from the upper layers shall be fed in parallel to both lanes 0 and R for encoding and transmission in the order the characters were received from the upper layers. (The character stream is not striped across the lanes before encoding as is done when operating in Nx mode.) When neither control symbols nor packets are available from the upper layers for transmission, an idle sequence shall be fed in parallel to both lane 0 and lane R for 64b/67b encoding and transmission on lanes 0 and R.

On reception, the codeword stream from either lane 0 or R shall be selected according to the state of the 1x/Nx Port_Initialization state machine (Section 5.19.7), decoded and the error-free control symbols and packets passed to the upper layers.

When a port that optionally supports and is enabled for both 2x mode and a wider Nx mode is operating in 1x, the port shall support both lanes 1 and 2 as redundancy lanes. The port shall transmit the 1x mode data stream on lanes 0, 1 and 2 and attempt to receive 1x mode data stream on lanes 0, 1 and 2. The port shall select between using the data received on lane 0 or the data received on the redundancy lane which may be either lane 1 or lane 2 depending on the connected port. Unless forced to use the redundancy lane, the port shall use the data stream received on lane 0 if it is available. The 1x/2x/Nx Port_Initialization state machine specified in Section 5.19.7.1 shall be used for a port supporting both 2x and a wider Nx mode to comply with the above requirements.

Packet data characters shall be scrambled before transmission and descrambled after reception as specified in Section 5.5.4.

Once a Nx port is initialized to a 1x mode, the port may elect to disable the output driver of the lanes which was not selected for reception by the initialization state machine of the connected port. Since the ports connected by the link may not be receiving on the same lane (one port could be receiving on lane 0 and the other port receiving on lane R), the information in the “Receive Width” field of received Status/Control Control Codeword can be used to determine which lanes can be output disabled. It is recommended that the mechanism for disabling the output driver be under software control.

5.13.4 Kx Link Striping and Transmission Rules

Transmitters operating in Kx multi-lane mode shall stripe control symbols and packets received from the upper layers across the K active output lanes in the order the characters were received from the upper layers. Here Kx is used as the active width of the transmitter as opposed to the Nx initial width of the port negotiated at the time of link initialization. Each lane shall then 64b/67b encode and transmit the codewords assigned to it. When neither control symbols nor packets are available from the upper layers for transmission, an idle sequence shall be fed to each of the K lanes for 64b/67b encoding and transmission.

Packets and control symbols shall be striped across the K active lanes beginning with lane 0. The order of striping shall be lane 0 through K-1 in order of increasing lane number and then repeating beginning again with lane 0.

If part way through a column, no more packets or control symbols are available for transmission, the column shall be filled (padded) with data codewords containing bytes of 0x00 until either a control symbol or packet becomes available for transmission or the end of the column is reached. The data codewords of 0x00 bytes become data codewords of pseudo-random data after scrambling by the transmitter’s lane scrambler(s).

The first control symbol after an IDLE3 sequence shall start at the beginning of a column.

After striping, each of the K streams of characters shall be independently 64b/67b encoded and transmitted.

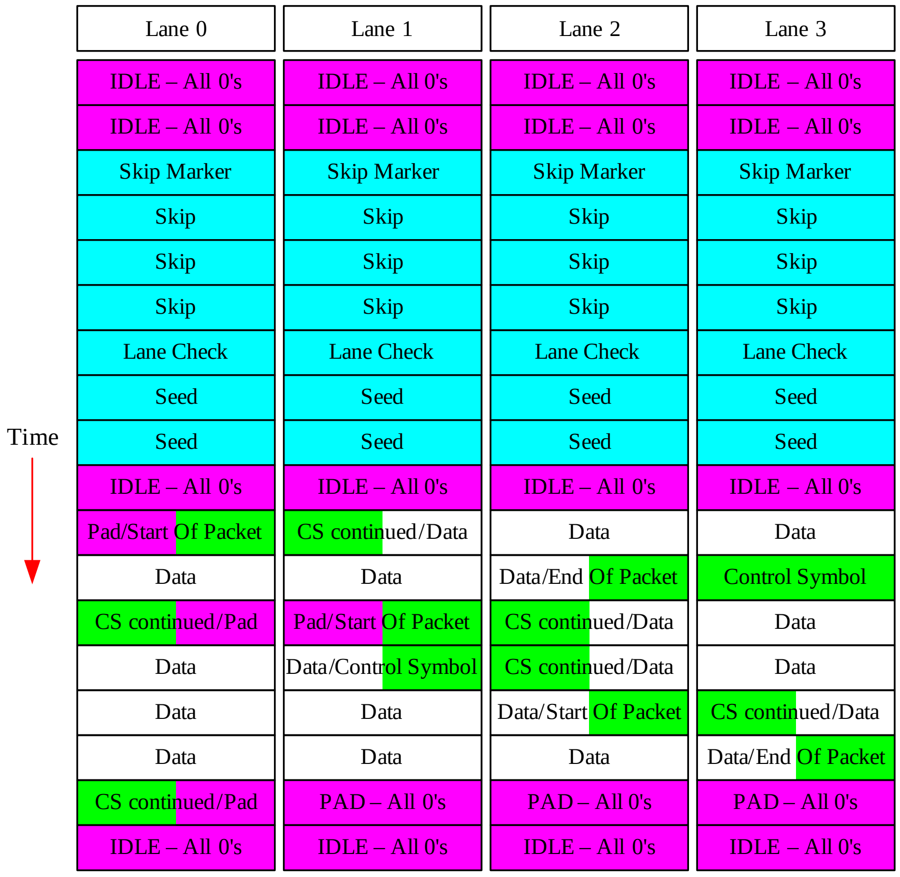
On reception, each lane shall be 64b/67b decoded.

Data characters shall be scrambled before transmission and descrambled after reception as specified in Section 5.5.4.

After decoding, the K lanes shall be aligned. The Status/Control control codewords transmitted as part of an idle sequence provide the information needed to perform alignment. After alignment, the columns are destriped into a single character stream and passed to the upper layers.

The lane alignment process eliminates the skew between lanes so that after destriping, the ordering of characters in the received character stream is the same as the ordering of characters before striping and transmission. Since the minimum number of non Status/Control codewords between Status/Control control codewords is 16, the maximum lane skew that can be unambiguously corrected is the time it takes to transmit 7 codewords on a lane.

Figure 5-20 shows an example of IDLE3 sequence, Control Symbol 64 and packet transmission on a 4x link.



Lane 0	Lane 1	Lane 2	Lane 3
IDLE – All 0's	IDLE – All 0's	IDLE – All 0's	IDLE – All 0's
IDLE – All 0's	IDLE – All 0's	IDLE – All 0's	IDLE – All 0's
Skip Marker	Skip Marker	Skip Marker	Skip Marker
Skip	Skip	Skip	Skip
Skip	Skip	Skip	Skip
Skip	Skip	Skip	Skip
Lane Check	Lane Check	Lane Check	Lane Check
Seed	Seed	Seed	Seed
Seed	Seed	Seed	Seed
IDLE – All 0's	IDLE – All 0's	IDLE – All 0's	IDLE – All 0's
Pad/Start Of Packet	CS continued/Data	Data	Data
Data	Data	Data/End Of Packet	Control Symbol
CS continued/Pad	Pad/Start Of Packet	CS continued/Data	Data
Data	Data/Control Symbol	CS continued/Data	Data
Data	Data	Data/Start Of Packet	CS continued/Data
Data	Data	Data	Data/End Of Packet
CS continued/Pad	PAD – All 0's	PAD – All 0's	PAD – All 0's
IDLE – All 0's	IDLE – All 0's	IDLE – All 0's	IDLE – All 0's

Figure 5-20. Typical 4x Data Flow with Control Symbol 64

5.14 Effect of Transmission Errors and Error Detection

Table 5-10 lists all possible codeword corruptions that can be caused by either single bit errors or burst errors. The notation /X/ => /Y/ means that the codeword of type /X/ has been corrupted by an error into the codeword of type Y. If the corruption results in a codeword that has an invalid type field, the notation /X/ => /INVALID/ is used. The table provides the information required to detect all isolated transmission errors on links operating with idle sequence 3.

Table 5-10. Codeword Corruption Caused by Bit Errors

Corruption	Description	Detection
/Control Codeword/, /Data Codeword/ => /INVALID/	Codeword corruption resulting in invalid type field	Detectable as an error when decoding the codeword.
/Descrambler Seed Control Codeword/, /Status/Control Control Codeword/, /Lane Check Control Codeword/, /Skip-Marker Control Codeword/ or /Skip Control Codeword/ => /Different Control Codeword/	Non-Symbol Bearing control codeword corrupted to different control codeword	Detectable as an error when validating the ordered sequence. If the errored codeword is the first of an ordered sequence it may be detected as multiple errors.
/Descrambler Seed Control Codeword/, /Status/Control Control Codeword/, /Lane Check Control Codeword/, /Skip-Marker Control Codeword/ or /Skip Control Codeword/ => /Data Codeword/	Non-Symbol Bearing control codeword corrupted to data codeword.	Detectable as an error when validating the ordered sequence. Additionally, it may be detectable by the fact that the Data Codeword is not expected at this position on the link.
/CSB Control Codeword/, /CSEB Control Codeword/ or /CSE Control Codeword/ => /Different Control Codeword/	Symbol Bearing control codeword corrupted to different control codeword.	Detectable as an error when validating the sequence of Control Codewords.
/CSB Control Codeword/, /CSEB Control Codeword/ or /CSE Control Codeword/ => /Data Codeword/	Symbol Bearing control codeword corrupted to data codeword.	Detectable as an error when validating the sequence of Control Codewords.
/Data Codeword/ => /CSB Control Codeword/, /CSEB Control Codeword/ or /CSE Control Codeword/	Data codeword corrupted to Symbol Bearing control codeword.	Detectable as an error when validating the sequence of Control Codewords.
/Data Codeword/ => /Descrambler Seed Control Codeword/, /Status/Control Control Codeword/, /Lane Check Control Codeword/, /Skip-Marker Control Codeword/ or /Skip Control Codeword/	Data codeword corrupted to Non-Symbol Bearing control codeword.	Detectable as an error when validating the ordered sequence. Additionally, this may be detectable as packet CRC error if the Data Codeword is part of a packet.
/Control Codeword/, /Data Codeword/ => /Reserved Control Codeword/	Any codeword corrupted to a reserved control codeword.	Detectable as an error when validating sequence of Control Codewords, or detectable as an error when validating an ordered sequence. Additionally, this may be detectable as packet CRC error if the Data Codeword is part of a packet.

5.15 Retimers and Repeaters

The LP-Serial Specification allows “retimers” and “repeaters”. Retimers amplify a weakened signal, but do not transfer jitter to the next segment because they use a local transmit clock. Repeaters also amplify a weakened signal, but transfer jitter to the next segment because they use a transmit clock derived from the received data stream. Retimers allow greater distances between end points at the cost of additional latency. Repeaters support less distance between end points than retimers and only add a small amount of latency.

5.15.1 Retimers

A retimer shall comply with all applicable AC specifications found in Chapter 12, "Electrical Specification for 10.3125 and 12.5 Gbaud LP-Serial Links". Retimers shall reset the jitter budget thus extending the transmission distance for the link. A RapidIO link shall support a maximum of two retimers.

A retimer is aware of the PMA encoding of RapidIO. The retimer is not otherwise required to be aware of the RapidIO protocol. The retimer has no registers that can be accessed from RapidIO. A retimer shall perform codeword synchronization and selective codeword inversion on the received bit stream, and shall repeat the received codewords after serializing the bitstream and performing selective codeword inversion again on transmission.

Retimers may use a transmit clock derived from a local reference. Retimers shall perform clock tolerance compensation between the received bit stream and transmitted bit stream. A retimer is aware of the Skip ordered sequence and the function of Skip codeword insertion and removal. A retimer may insert up to one Skip codeword immediately following a Skip Marker codeword, or remove one Skip codeword that immediately follows a Skip Marker codeword. Insertion or removal of a Skip codeword can affect the running disparity of the lane, so the retimer shall implement selective codeword inversion. Any insertion or removal of Skip codewords in a N-lane retimer shall be done on a full column.

A N-lane retimer shall perform lane synchronization and deskew, in exactly the same way a RapidIO device implementing the LP-Serial Physical Layer does when synchronizing inputs during initialization and startup. A Nx mode retimer shall synchronize and align all lanes that are driven to it. Therefore, such a retimer shall support the degradation of an input Nx link to a 1x link on either lane 0 or R. If any link drops out, the retimer shall continue to pass the active links, monitoring for the compensation sequence and otherwise passing through whatever codewords appear on its inputs. A retimer may optionally not drive any outputs whose corresponding inputs are not active.

A retimer shall only retime links operating at the same width (i.e. cannot connect a link operating at 1x to a link operating at Nx). A retimer may connect a 1x link to a Nx link that is operating in 1x mode.

Retimers do not check for code violations. Codewords received on one port are transmitted on the other regardless of code violations.

5.15.2 Repeaters

A repeater is used to amplify the signal, but does not retime the signal, and therefore can add additional jitter to the signal. It does not compensate for clock rate variation. The repeater repeats the received codewords as the bits are received by sampling the incoming bits with a clock derived from the bit stream, and then retransmitting them based on that clock. Repeaters may be used with Nx links but lane-to-lane skew may be amplified. Repeaters do not interpret or alter the bit stream in any way.

5.16 Port Initialization

This section specifies the port initialization process. The process includes detecting the presence of a partner at the other end of the link (a link partner), establishing bit synchronization and codeword boundary alignment and if present, adjusting any adaptive equalizers. The process also includes determining if the connected port supports an Nx mode in addition to 1x mode and selecting 1x or Nx mode operation, then, if 1x mode is selected, selecting lane 0 or lane R (the redundancy lane, lane 1 for 2x ports and lane 2 for 4x, 8x or 16x ports) for link reception.

Port initialization may optionally include baud rate discovery.

The initialization process is controlled by several state machines. The number and type of state machines depends on whether the port supports only 1x mode (a 1x port) or supports both 1x and one or more Nx modes (a 1x/Nx port). In either case, there is a primary state machine and one or more secondary state machines. The use of multiple state machines results in a simpler overall design. As might be expected, the initialization process for a 1x port is simpler than and is a subset of the initialization process for a 1x/Nx port.

The port initialization process supports an optional test mode that allows ports that support more than one multi-lane mode of operation to enable and monitor the operation of the inactive lanes when the port is operating at less than maximum width. The performance of inactive lanes can be monitored only if the inactive lanes are connected to and supported by the connected port and the test mode is implemented and enabled in both ports. The test mode is enabled with the “Enable inactive lanes” bit defined in Section 7.6.9. The initiation, implementation and interpretation of tests conducted using this test mode are beyond the scope of this specification.

The initialization process for 1x, 1x/Nx ports, and ports supporting 1x mode and multiple Nx modes is both described in text and specified with state machine diagrams. **In the case of conflict between the text and a state machine diagram, the state machine diagram takes precedence.**

5.16.1 1x Mode Initialization

The initialization process for ports that support only 1x mode shall be controlled by two state machines, 1x_Initialization and Lane_Synchronization. 1x_Initialization is the primary state machine and Lane_Synchronization is the secondary state machine. The operation of these state machines is described and specified in Section 5.19.7.1 and Section 5.19.4 respectively.

5.16.2 1x/Nx Mode Initialization

The initialization process for ports that support both 1x and a Nx mode is controlled by a primary state machine and five or more secondary state machines. The primary state machine is the 1x/Nx_Initialization state machine. Lane_Synchronization[0] through Lane_Synchronization[N-1] (one for each of the N lanes), Codeword_Lock[0] through Codeword_Lock [N-1], and Lane_Alignment (one for each supported Nx mode) are the secondary state machines. The operation of the secondary state machines is described in Section 5.19.4 through Section 5.19.6 respectively.

The 1x/Nx_Initialization state machine provides a degree of LP-Serial link width auto-negotiation. The goal of the auto-negotiation is to ensure that any connected combination of 1x, 1x/2x, 1x/4x, 1x/8x or 1x/16x LP-Serial ports that are configured in some manner to operate at the same baud rate will automatically find a link width over which they can communicate. For example if a 1x/4x port is connected to a 1x/8x port, they will auto-negotiate to operate in 1x mode. If however the 1x/8x port optionally also supports 4x mode (making it a 1x/4x/8x port), then the ports will auto-negotiate to operate in 4x mode.

In most configurations, the auto-negotiation also ensures that a pair of connected multi-lane LP-Serial ports configured in some manner to operate at the same baud rate will find a link width over which they can communicate in the presence of a lane failure. For example, if two 1x/4x ports are connected and lane 0 is broken in one direction, the ports will auto-negotiate to operate in 1x mode using lane 0 in the direction that lane 0 is operational and lane 2 in the direction that lane 0 is broken. This feature works only for pairs of ports that support the same redundancy lane. It does not work when a 1x/2x port is connected to a 1x/4x or wider port.

5.16.3 Baud Rate Discovery

Baud rate discovery is optional. If implemented, baud rate discovery occurs during the SEEK state of the Port Initialization state machine. Ports that implement baud rate discovery shall use the following algorithm.

1. When the port enters the SEEK state, it begins transmitting an idle sequence on lane 0 and, if the port supports a Nx mode, on lane R, the 1x mode redundancy lane. The idle sequence shall be transmitted at the highest lane baud rate that is supported by the port and that is enabled for use.

2. The port shall then look for an inbound signal on lane 0 or lane R of the link from a connected port. The method of detecting the presence of an inbound signal from a connected port is implementation specific and outside the scope of this specification.
3. Once an inbound signal is detected, the port shall determine the baud rate of the signal. The method of detecting the baud rate of the signal is implementation specific and outside the scope of this specification.
4. If the baud rate of the inbound signal is the same as the baud rate at which the port is transmitting, the link shall operate at that per lane baud rate until the port reenters the SEEK state and the baud rate discovery process is complete.
5. If the baud rate on the inbound signal is less than the baud rate of the idle sequence being transmitted by the port, the port shall reduce the baud rate at which it is transmitting to the next lowest baud rate that it supports and that is enabled for use and go to step 2.
6. If the baud rate on the inbound signal is greater than the baud rate of the idle sequence being transmitted by the port, the port shall continue transmitting at the current baud rate go to step 2.

An informational state diagram for the Baudrate_Discovery state machine is shown in Figure 5-21.

The techniques and algorithms used to compare the baud rates of the signals being transmitted and received are implementation specific and beyond the scope of this specification.

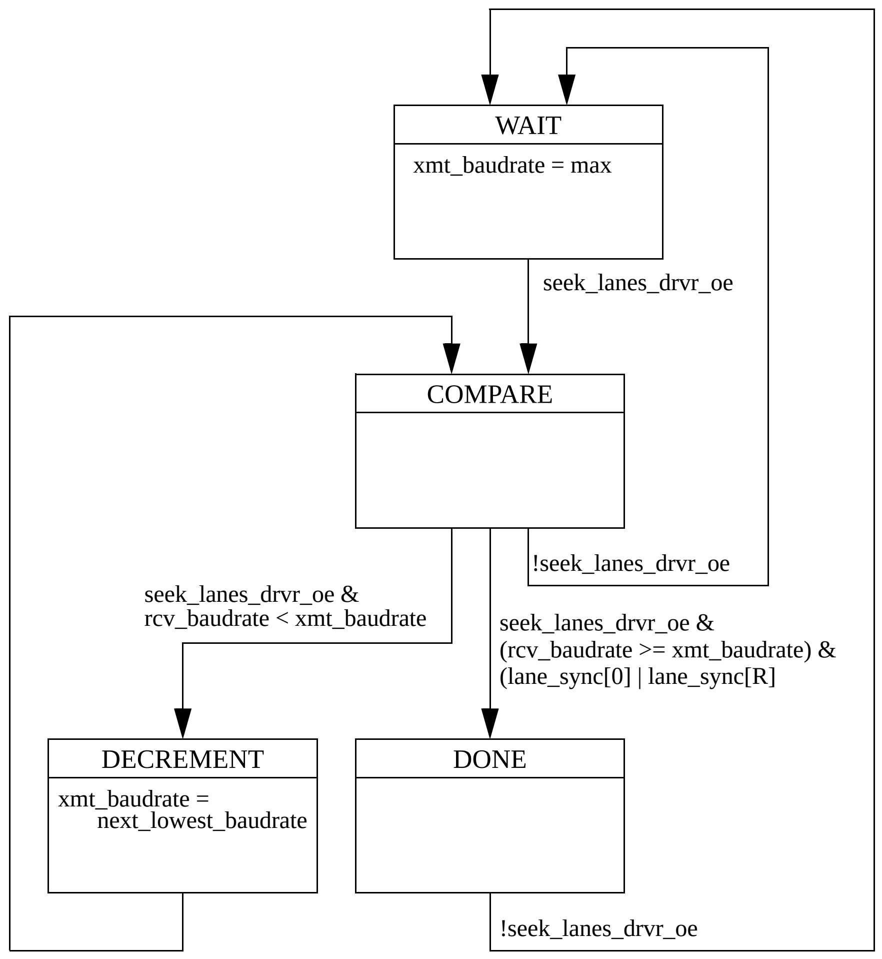

WAIT
xmt_baudrate = max
seek_lanes_drvr_oe
COMPARE
!seek_lanes_drvr_oe
seek_lanes_drvr_oe & rcv_baudrate < xmt_baudrate
seek_lanes_drvr_oe & (rcv_baudrate >= xmt_baudrate) & (lane_sync[0] | lane_sync[R]
DECREMENT
xmt_baudrate = next_lowest_baudrate
DONE
!seek_lanes_drvr_oe

Figure 5-21. Baudrate_Discovery state machine (Informational)

5.17 Asymmetric Operation

“Asymmetric” operation of an LP-Serial link is when the link operates with more lanes actively carrying control symbols and packets in one direction than the other. Support for asymmetric operation is optional. Asymmetric operation allows the directional bandwidth of a link and the power consumption of the ports connected by the link to be tailored to the performance requirements for the link.

All pairs of connected LP-Serial ports initialize to the widest symmetric lane width that is enabled and operational in both of the connected ports. Once both ports of a connected pair have completed port initialization (port_initialized asserted in both ports), the ports shall enter asymmetric mode only if both ports support asymmetric mode, asymmetric mode is enabled in both ports, and both ports have initialized to the same multi-lane width. Ports that initialize to a 1x mode shall not enter asymmetric mode.

When both ports connected by a link are in asymmetric mode, the link is referred to as being in asymmetric mode. An LP-Serial link shall not operate asymmetrically unless the link is in asymmetric mode. Once a link is in asymmetric mode, the width of each direction of transmission can be changed independently.

When in asymmetric mode, link directions shall operate only in width modes that are enabled in both of the connected ports and shall not operate in width modes that are wider than the symmetric width of the link at the time the connected ports completed port initialization and entered asymmetric mode.

Being in asymmetric mode shall not prevent a link from operating symmetrically. For example, consider an LP-Serial link connecting two 1x/4x ports with 4x mode enabled in both ports. When in asymmetric mode, this link can operate in 4x mode in both direction, 4x mode in one direction and 1x in the other direction, or 1x mode in both directions.

5.17.1 Port Transmission Width

5.17.1.1 Port transmission width commands

When a link operating with IDLE3 is in asymmetric mode, the transmission width for a specific direction of the link shall be under control of the port transmitting in that direction. In asymmetric mode, the transmission width of a port is changed by issuing a “transmit width port command” to the port.

There can be multiple sources for the transmit width port commands issued to a specific port. If multiple sources are present, a mechanism shall be provided to prioritize and serialize the transmit width port commands such that at most only one command is active at a specific time. Such a mechanism is implementation specific and beyond the scope of this specification.

A “transmit width port command” that is received by a port that is not in asymmetric mode shall be negatively acknowledged (NACKed) and discarded.

A transmit width port command that directs the port to transmit at a width that is not enabled in the port, or that is greater than the width to which the port initialized, shall be negatively acknowledged (NACKed) and discarded.

The time limit of 250 usec used on the following sections shall be controlled by the “Transmit Width Command Timeout” field from the Port n Link Timers Control 3 CSRs. Note that the two ports connected should use the same value in their “Transmit Width Command Timer”.

5.17.1.1.1 Transmit width port command protocol

Transmit width port commands shall be presented to a port in a 3-bit “Transmit width port command” field encoded as specified in Table 5-11. This command is written into the Port n Power Management CSRs bits 16-18.

Table 5-11. Transmit width port command

Transmit width port command [0:2]	Definition
0b000	Hold - no command
0b001	Transmit in 1x mode
0b010	Transmit in 2x mode
0b011	Transmit in 4x mode
0b100	Transmit in 8x mode
0b101	Transmit in 16x mode
0b110	Reserved
b0111	Reserved

A port shall respond to a transmit width port command through a 2-bit “Transmit width port command status” field encoded as specified in Table 5-12. This response can be read from the Port n Power Management CSRs bits 19-20.

Table 5-12. Transmit width port command status

Transmit width port command status [0:1]	Definition
0b00	No status
0b01	ACK - the command has been successfully executed
0b10	NACK - the command has for some reason not been executed and is rejected
0b11	Reserved

Transmit width port commands shall be presented to and acknowledged by a port using the following handshake protocol:

1. A transmit width port command shall be considered asserted if the Transmit width port command field has a value other than “Hold”; otherwise, no transmit width port command is asserted.
2. The assertion of a transmit width port command shall occur only when the Transmit width port command status field has the value “no status”.
3. Once asserted, a transmit width port command shall remain asserted and unchanged until either the command has been acknowledged (the value of the Transmit width port command status field is “ACK” or “NACK”), or the command has been asserted continuously for 250 usec. The command shall then be de-asserted within 250 usec of whichever event occurs first.
4. A port shall respond to a transmit width port command by changing the value of the Transmit width port command status field to “ACK” or “NACK” within 250 usec of the assertion of the command.

5. A port shall return the value of the Transmit width command status field to "no status" within 250 usec of the de-assertion of a transmit width port command.

5.17.1.2 Port transmission width requests

Either port of an LP-Serial link operating in asymmetric mode with IDLE3 can request that the connected port change its transmit width. "Transmission width requests" shall be sent to the connected port in the "Transmit width request" field of Status/Control control codewords transmitted by the port. "Transmit width requests" shall be acknowledged using the "Transmit width request pending" bit in Status/Control control codeword transmitted by the port receiving the request. "As its name implies, a "Transmit width request" is just a request. It is not a command. A port receiving a "Transmission width request" can either honor, or ignore and discard, a request after acknowledging the request.

Transmit width requests and acknowledgements shall use the following handshake protocol. The protocol applies to the "Transmit width request" field in Status/Control control codewords transmitted by one port (the requesting port) of the pair of ports connected by an LP-Serial link and the "Transmit width request pending" bit in Status/Control control codewords transmitted by the other port (the requested port) of the connected pair.

1. A transmit width request shall be considered asserted if the "Transmit width request" field has a value other than "Hold".
2. The assertion of a transmit width request shall occur only when the "Transmit width request pending" bit is de-asserted.
3. Once asserted, a transmit width request shall remain asserted and unchanged until either the request has been acknowledged (the "Transmit width request pending" bit is asserted) or the request has been asserted continuously for 250 usec. At which point the request shall then be de-asserted within 250 usec of whichever event occurred first.
4. A port shall respond to a transmit width request by asserting the "Transmit width request pending" bit within 250 usec of the assertion of the request.
5. The port receiving the transmit width request shall de-assert its "Transmit width request pending" bit within 250 usec of the de-assertion of the request. Up to this limit, the responding port can delay the de-assertion of the Transmit width request pending bit to control the rate of transmit width requests.

5.17.2 Port Receive Width

The receive width of a port operating in asymmetric mode with IDLE3 shall be controlled by "receive width link commands" received by the port from the connected port. Receive width link commands shall be transported across the link in the "Receive width command" field of Status/Control control codewords. Receive width acknowledgements shall be communicated using the “Receive width command ACK” and “Receive width command NACK” bits of Status/Control control codewords.

Receive width link commands shall be issued by a port to the connected port when the port has received an executable “transmit width port command”. No more than one receive width link command shall be active at a specific time.

When in asymmetric mode, a port shall receive only in width modes that are enabled in the port and that are no wider that the symmetric width of the link at the time the port completed port initialization and entered asymmetric mode. Receive width link commands to receive in other width modes shall be negatively acknowledged (NACKed) and discarded.

The time limit of 62.5 usec used on the following subsection shall be controlled by the “Receive Width Command Timeout” field from the Port n Link Timers Control 3 CSRs. Note that the two ports connected should use the same value in their “Receive Width Command Timer”.

5.17.2.1 Receive Width Link Command Protocol

Receive width link commands shall use the following handshake protocol:

1. A Receive width link command shall be considered asserted if the Receive width link command field has a value other than “Hold”; otherwise, no receive width link command is asserted.
2. The assertion of a receive width link command shall occur only when the “Receive width command ACK” and the “Receive width command NACK” bits are de-asserted.
3. Once asserted, a receive width link command shall remain asserted and unchanged until either the command has been acknowledged (either “Receive width command ACK” or “Receive width command NACK” is asserted) or the command has been asserted continuously for 62.5 usec. The command shall then be de-asserted within 62.5 usec of whichever event occurred first.
4. A port shall respond to a receive width link command by asserting “Receive width command ACK” or “Receive width command NACK” within 62.5 usec of the assertion of the command.
5. A port shall de-assert “Receive width command ACK” and “Receive width command NACK” within 62.5 usec of either the de-assertion of a receive width link command or the continuous assertion of “ACK” or “NACK” for 62.5 usec, whichever occurs first.

5.18 Structurally Asymmetric Links

Structurally asymmetric link (SAL) operation, as defined in section 4.13, "Structurally Asymmetric Links", may be supported by 64b/67b encoded links.

Behavioral requirements for SAL RX Width and SAL TX Width field values in the Port n SAL Control and Status CSR for 64b/67b encoded links are specified below in terms of which lanes are enabled for transmission and reception, what data is transmitted on each lane, and which lanes are enabled for reception. When SAL RX Width or SAL TX Width values are non-zero, DME_mode[k] shall be deasserted for all k lanes associated with the port, and retrain_en shall be deasserted.

Table 5-13. Structurally Asymmetric Link Tx/Rx Width Behaviors

SAL RX Width	SAL TX Width	Description
0b0000 (No Override)	0b0000 (No Override)	No effect on receive or transmit width.
0b0001 (1x, lane 0)	0b0001 (1x, lane 0. Disable lanes 1, 2, and 3)	Transmitter shall transmit a valid 1x IDLE3 bit stream on lane 0. Transmitter shall ensure that the link partner cannot detect a valid bit stream on lanes 1, 2, and 3. Receiver shall enable reception on lane 0 only. Receiver and transmitter shall operate as a 1x port.
0b0010 (1x, lane 1)	0b0010 (1x, lane 1. Disable lanes 0, 2, and 3)	Transmitter shall transmit a valid 1x IDLE3 bit stream on lane 1. Transmitter shall ensure that the link partner cannot detect a valid bit stream on lanes 0, 2, and 3. Receiver shall enable reception on lane 1 only. Receiver and transmitter shall operate as a 1x port.
0b0011 (1x, lane 2)	0b0011 (1x, lane 2. Disable lanes 0, 1, and 3)	Transmitter shall transmit a valid 1x IDLE3 bit stream on lane 2. Transmitter shall ensure that the link partner cannot detect a valid bit stream on lanes 0, 1, and 3. Receiver shall enable reception on lane 2 only. Receiver and transmitter shall operate as a 1x port.
0b0100 (1x, lane 3)	0b0100 (1x, lane 3. Transmit Lane 0 compliant data on lane 3. Disable lanes 0, 1, and 2)	Transmitter shall transmit a valid 1x lane 0 IDLE3 bit stream on lane 3. Transmitter shall ensure that the link partner cannot detect a valid bit stream on lanes 0, 1, and 2. Receiver shall behave as if data received on lane 3 was actually received on lane 0. Receiver and transmitter shall operate as a 1x port.
0b0101 (2x, lanes 0 & 1. Lanes 2 and 3 are not used)	0b0101 (2x, lanes 0 & 1. Disable lanes 2 and 3)	Transmitter shall send valid 2x mode IDLE3 bit streams on lanes 0 and 1. Transmitter shall ensure that the link partner cannot detect a valid bit stream on lanes 2 and 3. Transmitter shall operate as a 2x port. Receiver shall operate as a 2x/1x port.

Table 5-13. Structurally Asymmetric Link Tx/Rx Width Behaviors

SAL RX Width	SAL TX Width	Description
0b0110 (2x, lanes 2 & 3)	0b0110 (2x, lanes 2 & 3. Transmit lane 0 and 1 2x compliant data streams on lanes 2 and 3. Disable transmission on lanes 0 and 1.)	Transmitter shall send a valid 2x mode IDLE3 bit stream, as composed for lane 0, on lane 2. Transmitter shall send a valid 2x mode IDLE3 bit stream, as composed for lane 1, on lane 3. Transmitter shall ensure that the link partner cannot detect a valid bit stream on lanes 0 and 1. Transmitter shall operate as a 2x port. Receiver shall behave as if the data received on lane 2 was actually received on lane 0, and as if the data received on lane 3 was actually received on lane 1. Receiver shall operate as a 2x/1x port.
0b0111 (4x, lanes 0-3)	0b0111 (4x, lanes 0-3)	Transmitter shall operate as a 4x port. Receiver shall operate as a 4x/1x port.
0b1000 (8x, lanes 0-7)	0b1000 (8x, lanes 0-7)	Transmitter shall operate as an 8x port. Receiver shall operate as an 8x/1x port.
0b1001 (16x)	0b1001 (16x)	Transmitter shall operate as a 16x port. Receiver shall operate as a 16x port.
0b1010- 0b1011 (Implementation specific)	0b1010- 0b1011 (Implementation specific)	Implementation specific behavior.
0b1100- 0b1111 (Reserved)	0b1100- 0b1111 (Reserved)	Reserved.

It is strongly recommended that devices which support structurally asymmetric links operating at Baud Rate Class 3 speeds implement register control of the transmit emphasis coefficient set.

5.19 State Machines

In the following sections, state machines are specified using state diagrams and described using text. In the case of conflict(s) between the descriptive text and a state machine diagram, the state machine diagram takes precedence.

While only the detailed state machine diagram for the 1x/2x/Nx state machine is shown in Section 5.19.7, "Port Initialization State Machine", the initialization state machine may alternatively support any of the combinations of widths as described for lower data rates in Section 4.12.4.

5.19.1 State Machine Conventions, Functions and Variables

5.19.1.1 State Machine Conventions

The conventions used in state machine specification are as follows:

A state machine state is persistent until an exit condition occurs. If no exit conditions are specified, the exit is unconditional.

A state machine variable that is listed in the body of a state but is not part of an assignment statement is asserted for the duration of that state only.

A state machine variable that is assigned a value in the body of a state retains that value until assigned a new value in another state.

A state machine function that is listed in the body of a state is executed once during the state.

A state machine variable is asserted when its value is 1 and de-asserted when it value is 0.

Except when otherwise directed by parentheses, the order of precedence of logic operations when evaluating a logic expression is, in order of decreasing precedence, negation/compliment (!) followed by AND (&), exclusive-OR (^) and OR (|).

Logic expressions within paired parentheses are evaluated before the rest of a logic expression is evaluated with the operations within the innermost pair of parentheses evaluated first.

5.19.1.2 State Machine Functions

State machine functions shall, with one exception, be executed to completion once their execution has begun. The exception is that their execution shall be terminated when an overriding condition such as “reset” or “reinitialize” is asserted.

The functions used in the state machines are defined as follows:

change(operand)

Asserted when the operand changes its state or value.

check_dscrmblr_sync()

This function checks the descrambler status based on the last received codeword. If the last received codeword was Seed control codeword and the seed value matches the current state of the descrambler then the dscrmblr_sync variable shall be asserted and the dscrmblr_error shall be de-asserted. If the last received codeword was either a Seed control codeword with a seed value that does not match the current state of the descrambler, or it was not a Seed control codeword that terminated a Seed ordered sequence unexpectedly (odd number of consecutive Seed control codewords), then the dscrmblr_sync variable shall be de-asserted and the dscrmblr_error shall be asserted. Both dscrmblr_sync and descrmblr_error shall retain their value until the next Seed is received. Dscrmblr_sync and descrmbler_error shall be deasserted on loss of lane synchronization.

clear(variable)

Clear a variable to its inactive state.

next_codeword( )

Get the next 67-bit codeword for the lane when it becomes available.

next_Ncolumn( )

Get the next column of N codewords from lanes 0 to N-1 when it becomes available.

set(variable)

Set a variable to its active state.

set_xmt_equalizer(k,coefficient set)

Set the port’s lane k transmit equalizer settings to the values specified for “coefficient set”.

slip_codeword_alignment( )

Move the serial to 67-bit parallel conversion boundaries one bit earlier or one bit later, but always in the same direction in the input serial data stream.

transmit_sc_sequences(n)

Transmit n Status/Control ordered sequences while maintaining the minimum specified separation of the sequences.

wait(wait_time)

Wait for a time equal to wait_time +/- 10%. The value of wait_time shall be expressed as a number and a defined unit of time. For example, 100 UI, 18 training frames, 37 codewords, and 22 msec are all acceptable values for wait_time.

5.19.1.3 State Machine Variables

The variables used in the state machines are defined as follows:

16_lanes_drvr_oe

When asserted, the drivers for lanes 0 through 15 shall be output enabled. When de-asserted, shall have no effect on which lane drivers are output enabled.

2x_mode_enabled, 4x_mode_enabled,
8x_mode_enabled, 16x_mode_enabled

Each of these variable is asserted when operation in that particular mode is enabled; otherwise, de-asserted.

2x_mode_supported, 4x_mode_supported,

8x_mode_supported, 16x_mode_supported

Each of these variable is asserted when operation in that particular mode is supported by the implementation; otherwise, de-asserted.

4_lanes_drvr_oe

When asserted, the drivers for lanes 0 through 3 shall be output enabled. When de-asserted, shall have no effect on which lane drivers are output enabled.

8_lanes_drvr_oe

When asserted, the drivers for lanes 0 through 7 shall be output enabled. When de-asserted, shall have no effect on which lane drivers are output enabled.

Acounter

A counter used in the N_Lane_Alignment state machine to count received pairs of consecutive columns that contain Status/Control control codewords in every row of both columns.

asym_mode_en

The variable shall be asserted if any of the bits in the “Asymmetric modes enabled” field of Port n Power Management CSRs is set; otherwise, de-asserted.

```
bad_rcv_width_cmd = !rcv_width_link_cmd_ack & !rcv_width_link_cmd_nack &
                    (!asym_mode & (from_sc_rcv_width_link_cmd != “hold”)
                     | (from_sc_rcv_width_link_cmd = “2x mode”) &
                       !asym_2x_mode_enabled
                     | (from_sc_rcv_width_link_cmd = “4x mode”) &
                       (!asym_4x_mode_enabled | (max_width < “4x”))
                     | (from_sc_rcv_width_link_cmd = “8x mode”) &
                       (!asym_8x_mode_enabled | (max_width < “8x”))
                     | (from_sc_rcv_width_link_cmd = “16x mode”) &
                       (!asym_16x_mode_enabled | (max_width < “16x”))
                     | (from_sc_rcv_width_link_cmd = “reserved”)
                    )

bad_xmt_width_cmd = !xmt_width_port_cmd_ack & !xmt_width_port_cmd_nack &
                    (!asym_mode & (xmt_width_port_cmd != “hold”)
                     | (xmt_width_port_cmd = “2x mode”) &
                           !asym_2x_mode_enabled
                     | (xmt_width_port_cmd = “4x mode”) &
                           (!asym_4x_mode_enabled | (max_width < “4x”))
                     | (xmt_width_port_cmd = “8x mode”) &
                           (!asym_8x_mode_enabled | (max_width < “8x”))
                     | (xmt_width_port_cmd = “16x mode”) &
                           (!asym_16x_mode_enabled | (max_width < “16x”))
                     | (xmt_width_port_cmd = “reserved”)
                    )
```

codeword_lock[k]

Asserted when the lane k Codeword_Lock state machine determines that the lane k receiver is in bit synchronization and 64b/67b codeword boundary alignment with its input serial data stream. Otherwise, deasserted.

CWcounter[k]

Counter used in the lane k Codeword_Lock state machine to count the number of codewords.

disc_tmr_done (discovery timer done)

Asserted when disc_tmr has run continuously asserted for the interval configured in the Discovery Completion Timer field of the Port n Link Timers Control 2 CSRs for the port. De-asserted when disc_tmr_en is de-asserted; otherwise, de-asserted.

disc_tmr_en (discovery timer enable)

When asserted, the disc_tmr runs continuously. When de-asserted, the disc_tmr shall be reset to and retain its initial (default) value.

De-asserted automatically when the state machine exits the state in which the disc_tmr was enabled (disc_tmr_en was asserted).

DME_mode[k]

When asserted, lane k shall generate DME training frames for transmission. When de-asserted, lane k shall generate 64b/67b codewords for transmission. DME_mode[k] is affected by structurally asymmetric link operation. For more information, see Section "Structurally Asymmetric Links" on page -204.

dme_wait_tmr_done[k]

Asserted when the dme_wait_tmr[k] has run for a time equivalent to 252 DME frames (or 4 times the value in the DME Wait Timer field of the Port n Link Timers Control CSR).

dme_wait_tmr_en[k]

When asserted, allows the dme_wait_tmr to run. When de-asserted, the dme_wait_tmr[k] shall be reset to and retain its initial (default) value. The dme_wait_tmr_en[k] shall be de-asserted when either dme_wait_tmr_done[k] is asserted or the state machine exits the state in which the dme_wait_tmr[k] was enabled (dme_wait_tmr_en[k] was asserted), whichever occurs first.

drvr_oe[k]

Asserted when the lane k driver is output enabled; otherwise, de-asserted. When de-asserted the transmitter shall enter Electrical IDLE as defined in Section 12.4.5.

The value is calculated based on the values of some or all of 4_lanes_drvr_oe, 8_lanes_drvr_oe, 16_lanes_drvr_oe, lane0_drvr_oe, lanes01_drvr_oe, lanes02_drvr_oe and force_drvr_oe[k], dependent on the value of k.

DScounter[k]

Counter used in the lane k Lane_Synchronization state machine to count the number of descrambler Seed control codewords with a seed value that match the current state of the descrambler.

dscrmblr_error

Asserted if the last receive codeword was a Seed control codeword with a seed value the does not match the current state of the descrambler, or if a sequence of an odd number of consecutive Seed control codewords are received.

dscrmblr_sync

Asserted if the last receive codeword was a Seed control codeword with a seed value that match the current state of the descrambler.

end_asym_mode

When asserted, causes the Port_Initialization state machine to exit the ASYM_MODE state and enter the SILENT state.

force_1x_mode

Forces a port that supports one or more multilane modes to use 1x mode. When asserted, all Nx (multi-lane) modes are disabled.

This variable is derived from the Port Width Override field of the Port n Control CSRs.

force_drvr_oe[k] (force driver output enable for lane k)

When asserted, the output enable for the lane k driver shall be asserted. When de-asserted, the state of the output enable for the lane k driver is controlled by other variables.

force_laneR

When force_1x_mode is asserted, force_laneR controls whether lane 0 or lane R, the redundancy lane, is preferred for 1x mode reception. If force_laneR is asserted, lane R is the preferred lane. If force_laneR is de-asserted, lane 0 is the preferred lane. If the preferred lane is functional, it is selected by the port initialization state machine for 1x mode reception. If the preferred lane is not functional, the non-preferred lane, if functional, is selected for 1x mode reception.

If force_1x_mode is not asserted, the state of force_laneR has no effect on the initialization state machine.

This variable is derived from the Port Width Override field of the Port n Control CSRs.

force_no_lock[k]

When asserted, forces the Codeword_Lock state machine to re-initialize. The signal is set and cleared by the Lane_Synchronization state machine.

force_reinit

When asserted, forces the Port_Initialization state machines to re-initialize. The signal is set under software control through the Port Width Override field of the Port n Control CSRs and is cleared by the Port_Initialization state machine.

frame_lock[k]

The frame_lock variable is identical to the frame_lock variable defined in IEEE 802.3-2008 Section 72.6.10.3. The variable frame_lock[k] is frame-lock variable for lane k.

from_dme_rcvr_ready[k]

The value of this variable is updated based on the value of the “Receiver ready” bit in the most recent training frame received on lane k from its link partner. The training frame is defined in IEEE 802.3 Section 72.6.10.2.

De-asserted in the UNTRAINED state of the lane training state machine. The value of from_dme_rcvr_ready[k] shall not be set until no fewer than three consecutive training frames have been received with the receiver ready bit asserted. From_dme_rcvr_ready[k] shall be de-asserted if a single training frame has been received with the receiver ready bit de-asserted.

from_sc_asym_mode_en

The variable shall have the same value as the “Asymmetric mode enabled” bit in the most recent error-free Status/Control control codeword received on lane 0 from its link partner.

from_sc_initialized (partner initialized)

The value of this variable shall be the value of the “Port initialized” bit in the most recent error-free Status/Control control codeword received by the port from its link partner.

from_sc_lane_ready[k] (link partner lane k ready)

The value this variable shall be the value of the “Receive lane ready” field in the most recent error-free Status/Control control codeword received on lane k from its link partner.

from_sc_lane_silence[k]

The variable shall have the same value as the “Lane Entering Silence” bit in the most recent error-free Status/Control control codeword received on lane k from its link partner.

from_sc_lane_trained[k] (link partner lane k receiver trained)

The value of this variable shall be the value of the “Lane trained” bit in the most recent error-free Status/Control control codeword received on lane k from its link partner.

from_sc_port_silence

The variable shall be asserted when receiving an error-free Status/Control ordered sequence where any one of the “Port Entering Silence” bits from lanes less than max_width is asserted, or de-asserted if none of the “Port Entering Silence” bits from lanes less than max_width is asserted.

from_sc_rcv_lanes_ready (link partner receiver lanes ready)

The value this variable shall be the value of the “Receive lanes ready” field in the most recent error-free Status/Control control codeword received by the port from its link partner.

from_sc_rcv_width (link partner receive width)

The value this variable shall be the value of the “Receive width” field in the most recent error-free Status/Control control codeword received by the port from its link partner.

from_sc_rcv_width_link_cmd (received receive width link command)

Contains the value of the “Receive width command” field of the most recently received error-free Status/Control control codeword received by the port from its link partner. If the variables lane_sync[0] is de-asserted, from_sc_rcv_width_link_cmd shall be set to the value “hold”.

from_sc_rcv_width_link_cmd_ack
(link partner receive width link command acknowledge)

The value this variable shall be the value of the “Receive width command ACK” field in the most recent error-free Status/Control control codeword received by the port from its link partner.

from_sc_rcv_width_link_cmd_nack
(link partner receive width link command negative acknowledge)

The value this variable shall be the value of the “Receive width command NACK” field in the most recent error-free Status/Control control codeword received by the port from its link partner.

from_sc_retrain_en (partner retraining enable)

The value of this variable shall be the value of the “Retraining enabled” bit in the most recent error-free Status/Control control codeword received by the port from its link partner.

from_sc_retrain_grnt (partner retrain grant)

The value of this variable shall be the value of the “Retrain grant” bit in the most recent error-free Status/Control control codeword received by the port from its link partner.

from_sc_retrain_ready (partner retrain ready)

The value of this variable shall be the value of the “Retrain ready” bit in the most recent error-free Status/Control control codeword received by the port from its link partner.

from_sc_retraining

The value of this variable shall be the same as the value of the “Retraining” bit in the most recent error-free Status/Control control codeword received by the port from the link.

from_sc_xmt_1x_mode (partner transmitting in 1x mode)

This variable shall have the same value and meaning as the “Transmit 1x mode” bit in the most recent error-free Status/Control control codeword received by the port from the link.

IVcounter[k]

Counter used in the lane k Codeword_Lock state machine to count the number of invalid sync headers detected.

keep_alive

A periodic signal that has two timers associated with it. The first timer is the keep alive period that determines how often keep_alive goes active, this timer is programmable to values in the range of 10 msec to 10 sec. The second timer is the keep alive active time that determines how long keep_alive is active before going inactive, this timer is programmable to values in the range of 2 usec to 125 usec. The signal is used to ensure that the transmitter of each trained lane is output enabled periodically to allow the connected lane receiver to track changes due to temperature drift and any other slow moving changes, and thereby keeping the lane receiver sufficiently trained to be quickly operational when needed.

lane_degraded[k]

Asserted when the adaptive equalization for lane k has previously been successfully trained (lane_trained[k] asserted) and subsequently the characteristics of lane k have changed enough that it is determined that the transmission quality of the lane has degraded to the point that the adaptive equalization needs to be retrained.

De-asserted when lane_trained[k] is de-asserted or when lane_trained[k] is asserted and the retraining of the lane k adaptive equalization has been successfully completed.

The criteria for determining when characteristics of a previously trained lane have changed enough that the adaptive equalization requires retraining is implementation specific and beyond the scope of this specification. A possible criteria for the assertion of lane_degraded[k] is that the lane k BER has become greater than $1*10^{-12}$.

lane_ready[k]

lane_ready[k] = lane_sync[k] & lane_trained[k] & !lane_retraining[k]

lane_retraining[k]

Asserted when the adaptive equalization controlled by the lane k receiver is retraining; otherwise, de-asserted.

lane_sync[k]

Asserted when the lane k Lane_Synchronization state machine determines that the lane k receiver is in bit synchronization and 64b/67b codeword boundary alignment with its input serial data stream; otherwise, de-asserted.

lane_trained[k]

De-asserted when the Port_Initialization state machine is in the SILENT state.

De-asserted if any of the adaptive equalization that is controlled by the lane k receiver has not been successfully trained.

De-asserted if retraining of lane k fails.

Asserted in response to the train_lane[k] being asserted if the lane k receiver controls no adaptive equalization, which includes any adaptive receive equalization in the lane k receiver and any adaptive transmit equalization in the connected lane k transmitter controlled by the lane k receiver.

Asserted when all of the adaptive equalization that is controlled by the lane k receiver has been successfully trained. Once asserted, lane_trained[k] remains asserted until one of the above de-assertion criteria is met.

lane0_drvr_oe

When asserted, the output driver for lane 0 shall be enabled

lanes01_drvr_oe

When asserted, the output drivers for lanes 0 and 1 shall be enabled.

lanes02_drvr_oe

When asserted, the output drivers for lanes 0 and 2 shall be enabled.

lost_valid_cs_reception

Asserted when receive_enable has been continuously asserted for the last 2048 columns and no valid control symbol or Status/Control Ordered Sequence has been received during that time.

De-asserted when the Codeword Lock state machine enters the NO_LOCK state.

Note that this variable is asserted on a port based event and de-asserted on a lane based event.

LR_initialize

Initial transmit emphasis coefficient set for Long Reach operation. The LR_initialize coefficient set shall be compliant with IEEE Standard 802.3-2008 (Part 5), Clause 72.6.10.4.2 Training.

max_width

Indicates the symmetric width of the link when the port was initialized. It is also the maximum width of either direction of the link when the link is operating in asymmetric mode.

Mcounter

A counter used in the Lane_Alignment state machine to count received pairs of consecutive columns received that contain a Status/Control control codeword in at least one row of the first column, but that do not contain Status/Control control codewords in every row of both columns.

N_lanes_aligned

Asserted by the Lane_Alignment state machine when it determines that lanes 0 through N-1 are in sync and aligned.

N_lanes_drvr_oe

When asserted, the output drivers for lanes 0 through N - 1 are enabled.

N_lanes_ready

N_lanes_ready = N_lanes_aligned & lane_ready[0] & ... & lane_ready[N-1]

N_lane_sync

Asserted when lanes 0 through N-1 of a receiver operating are in bit synchronization and codeword boundary alignment; otherwise, de-asserted.

N_lane_sync = lane_sync[0] & ... & lane_sync[N-1] no_sc_Ncolumn

Asserted if none of the codewords in the Ncolumn returned most recently by the next_Ncolumn() function are Status/Control control codewords; otherwise, de-asserted.

part_sc_Ncolumn

Asserted if some, but not all of the codewords in the Ncolumn returned most recently by the next_Ncolumn() function are Status/Control control codewords; otherwise, de-asserted.

port_initialized

Asserted when the port successfully completes the port initialization process and remains asserted until the Port_Initialization state machine re-enters the SILENT state; otherwise, de-asserted.

PIsm_state (Port_Initialization state machine state)

The current state of the port's Port_Initialization state machine.

rcv_width (receive width)

A three bit field indicating the width mode at which the port is currently receiving control symbols and packets from the link. Also the source of the value placed in the "Receive width" field of Status/Control control codewords transmitted by the port.

The current receive width shall be encoded as follows:

0b000 - None; the port has not completed initialization
0b001 - 1x mode
0b010 - 2x mode
0b011 - 4x mode
0b100 - 8x mode
0b101 - 16x mode
0b110 - 1x mode, lane 1
0b111 - 1x mode, lane 2

The rcv_width variable shall retain the value it held prior to the Port Initialization State Machine entering the 1x_RECOVERY, 2x_RECOVERY or Nx_RECOVERY states for the duration of those recovery states.

rcv_width_link_cmd (receive width link command)

The variable contains the value placed in the "Receive width command" field of Status/Control control codewords transmitted on the link by the port.

rcv_width_link_cmd_ack (receive width link command acknowledge)

When asserted, indicates that the current receive width link command was successfully executed. De-asserted when the value "Receive width command" field of Status/Control control codeword most recently received by the port is “hold”; otherwise de-asserted.

The value of this bit is the value of the “Receive width command ACK” field of Status/Control control codewords transmitted by the port.

rcv_width_link_cmd_nack (receive width link command negative acknowledge)

When asserted, indicates that the current receive width link command was not executed and the receive width is unchanged. De-asserted when the value of the “Receive width command” field in Status/Control control codeword most recently received by the port is “hold”; otherwise de-asserted.

The value of this bit is the value of the “Receive width command NACK” field of Status/Control control codewords transmitted by the port.

rcv_width_tmr_done (receive width timer done)

Asserted when the rcv_width_tmr has run for 62.5 usec +/- 34%. De-asserted when the state machine exits the state in which rcv_width_tmr_en was asserted.

rcv_width_tmr_en (receive width timer enable)

When asserted, the rcv_width_tmr is enable to run. De-asserted when either the rcv_width_tmr_done is asserted or the state machine exits the state in which rcv_width_tmr_en was asserted. When de-asserted, the timer is set to and held at its initial (default) value.

receive_enable

When asserted, and port_initialized and link_initialized are also asserted, the port can accept control symbols and packets from the link. When de-asserted, control symbols and packets received from the link shall be ignored and discarded.

Used to enable/disable the reception of control symbols and packets when port_initialized and link_initialized are asserted.

receive_enable = receive_enable_pi & receive_enable_rw

receive_enable_pi

A local receive enable control used in the Port_Initialization state machine.

receive_enable_rw

A local receive enable control used in the Receive_Width state machine.

receive_lane1

Asserted when a port is operating in 1x mode and the port is either receiving the 1x mode data stream from lane 1 or has entered the 1x_RECOVERY state from the 1x_MODE_LANE1 state; otherwise, de-asserted.

receive_lane2

Asserted when a port is operating in 1x mode and the port is either receiving the 1x mode data stream from lane 2 or has entered the 1x_RECOVERY state from the 1x_MODE_LANE2 state; otherwise, de-asserted.

recovery_retrain

Variable used in the Port_Initialization state machine to prevent the recovery period to be extended more than once when allowing for retraining.

recovery_tmr_done

Asserted when the recovery_tmr has run for 62.5 msec +/- 34%. De-asserted when the recovery_tmr_en is de-asserted.

recovery_tmr_en (recovery timer enable)

When asserted, allows the recovery_tmr to run. When de-asserted, the recovery_tmr shall be reset to and retain its initial (default) value.

recovery_tmr_en shall be de-asserted when either recovery_tmr_done is asserted or the state machine exits the state in which the recovery_tmr was enabled (recovery_tmr_en was asserted), whichever occurs first.

retrain

Controls when a pending retrain operation can be executed. When asserted, the pending retraining operation is allowed to begin execution. When de-asserted, a pending retrain operation shall wait.

retrain_en (retrain enable)

The variable shall have the same value and meaning as the Port n Control 2 CSRs field “10G Retraining Enable” bit. Retrain_en is affected by structurally asymmetric link operation. For more information, see Section "Structurally Asymmetric Links" on page -204.

retrain_fail[k]

When asserted, indicates that an adaptive equalization retrain failure has occurred on receive lane k; otherwise, de-asserted.

retrain_grnt (retrain grant)

An output of the Retrain/Transmit_Width_Control. Asserted when a pending retraining request has won permission to proceed.

retrain_lane[k]

When asserted, the lane k adaptive equalization training mechanism shall attempt to retrain all lane k adaptive equalization controlled by the lane k receiver. When de-asserted, lane k retraining, if in progress, shall be terminated within 1 msec.

retrain_pending

```
retrain_pending = ((lane_degraded[0] | lane_degraded[1] | ... |
                   lane_degraded[max_width-1]) & (max_width > 1) |
                  (max_width = 1) &
                   (lane_degraded[0] & (!receive_lane1 & !receive_lane2) |
                    lane_degraded[1] & receive_lane1 |
                    lane_degraded[2] & receive_lane2)) &
                 retrain_en & from_sc_retrain_en &
                 port_initializaed & from_sc_initialized
```

retrain_ready

Asserted when the port is ready to begin retraining; otherwise de-asserted.

retrain_tmr_done

Asserted when the retrain_tmr has run for 62.5 milliseconds +/- 34%. De-asserted when the retrain_tmr_en is de-asserted.

retrain_tmr_en

When asserted, the retrain_tmr runs continuously. When de-asserted, the retrain_tmr is reset to and maintains its initial (default) value.

retraining

Indicates when one or more of the port's lanes are retraining.

```
retraining = (lane_retraining[0] | ... | lane_retraining[max_width-1]) &
              (max_width > 1) |
             (max_width =1) &
              (lane_retraining[0] |
               lane_retraining[1] |
               lane_retraining[2])
```

sc_Ncolumn

Asserted if all of the codewords in the Ncolumn returned most recently by the next_Ncolumn() function are Status/Control control codewords; otherwise, de-asserted.

seek_lanes_drvr_oe

The output enable for the lane 0 and the lane R output drivers of a 1x/Nx port.

SH_transition

Asserted when bits [1:2] (sync header) of the codeword being tested are complements of one another.

signal_detect

Asserted when a lane receiver is enabled and a signal meeting an implementation defined criteria is present at the input of the receiver. The use of signal_detect is implementation dependent. It can be continuously asserted or used to require that some implementation defined additional condition be met before the Lane_Synchronization state machine is allowed to exit the NO_SYNC state. Signal_detect might for example be used to ensure that the input signal to a lane receiver meet some minimum AC input power requirement to prevent the receiver from locking on to crosstalk.

silence_tmr_done

Asserted when silence_tmr_en has been continuously asserted for 120 +/- 40 usec and the state machine is in the SILENT state. The assertion of silence_tmr_done causes silence_tmr_en to be de-asserted. When the state machine is not in the SILENT state, silence_tmr_done is de-asserted.

silence_tmr_en

When asserted, the silence_tmr runs. When de-asserted, the silence_tmr is reset to and maintains its initial value.

train_lane[k]

When asserted, causes all adaptive equalization controlled by the lane k receiver to be trained. When de-asserted, training of all adaptive equalization controlled by the lane k receiver shall be terminated and the training mechanism returned to its idle state within 1 msec.

train_tmr_done[k] (train timer done)

Asserted when the lane k train timer has run for the interval configured in the DME Training Completion Timer field or the CW Training Completion Timer field of the Port n Link Timers Control CSRs for the port, depending on what type of training is active. De-asserted when train_tmr_en[k] is de-asserted.

train_tmr_en[k] (train timer enable)

When asserted, the lane k train timer shall run continuously. When de-asserted, train timer shall reset to and maintain its initial (default) value.

training_fail[k]

When asserted, indicates that an adaptive equalization training failure has occurred on receive lane k since the bit was last read; otherwise, de-asserted.

transmit_enable

When asserted, allows the port to transmit control symbols and packets. When de-asserted, the transmission of control symbols and packet shall be terminated at a natural control symbol/packet boundary and remain terminated until transmit_enable is again asserted.

The value of transmit_enable is controlled by the Port_Initilization, Transmit_Width and Retrain/Xmt_Width_Control state machines. If either one of the state machines de-asserts their local transmit enable then the transmit_enable is de-asserted.

transmit_enable = transmit_enable_pi & transmit_enable_tw &
    transmit_enable_rtwc

transmit_enable_pi

A local transmit enable control used in the Port_Initialization state machine.

transmit_enable_rtwc

A local transmit enable control used in the Retrain/Xmt_Width_Control state machine.

transmit_enable_tw

A local transmit enable control used in the Transmit_Width state machine. Control symbols and packets may be transmitted when transmit_enable_tw is asserted. When transmit_enable_tw is deasserted, the port shall complete transmission of packets or control symbols in progress and then stop transmitting further packets or control symbols until transmit_enable_tw is asserted. Completion of in-progress packet and control symbol transmission shall be signaled by the assertion of the xmting_idle variable.

Vcounter[k]

Counter used in the lane k Codeword_Lock state machine to count the number of valid sync headers detected.

xmt_sc_seq (transmit Status/Control ordered sequences)

When asserted, the port shall transmit a minimum of one Status/Control ordered sequence per 256 codewords transmitted per lane. When de-asserted, the port and the connected port shall transmit Status/Control ordered sequences at the rate(s) required by other portions of this specification.

The value of xmt_sc_seq is replicated in the Transmit Status/Control ordered sequences field of the Status/Control control codewords transmitted on the link by the port.

xmt_width (transmit width)

Indicates the width mode at which the port is currently transmitting. The current transmit width of the port can be encoded as follows:

0b000 - None
0b001 - 1x mode
0b010 - 2x mode

0b011 - 4x mode
0b100 - 8x mode
0b101 - 16x mod
0b110 - Reserved
0b111 - Reserved

xmt_width_grnt (transmit width grant)

Controls when a pending transmit width command can be executed. When asserted, the pending transmit width command is allowed to begin execution; otherwise, a pending transmit width command shall wait.

xmt_width_cmd_pending (transmit width command pending)

= 1x_mode_xmt_cmd | 2x_mode_xmt_cmd | 4x_mode_xmt_cmd |
  8x_mode_xmt_cmd | 16x_mode_xmt_cmd

xmt_width_port_cmd (transmit width port command)

A command issued to a port by software or some mechanism within the device containing the port ordering the port to transmit in a specific width mode. The variable has the value “hold” when no command is present. This variable is used in conjunction with the variables xmt_width_port_cmd_ack and xmt_width_port_cmd_nack to control the flow of transmit width port commands.

The value of xmt_width_port_cmd may change from “hold” to another value only when xmt_width_port_cmd_ack and xmt_width_port_cmd_nack are both de-asserted. When the value of xmt_width_port_cmd is other than “hold”, it shall retain that value until either xmt_width_port_cmd_ack or xmt_width_port_cmd_nack is asserted, at which point the value of xmt_width_port_cmd shall change to “hold”.

When there are multiple sources of transmit width port commands, the prioritizing and multiplexing of commands from the multiple sources is implementation specific and beyond the scope of this specification.

xmt_width_port_cmd_ack (transmit width port command acknowledge)

Asserted when the pending xmt_width_port_cmd is different from “hold” and has been executed. Once asserted, it remains asserted until the xmt_width_port_cmd is set to “hold”, at which point the variable is de-asserted; otherwise, de-asserted.

xmt_width_port_cmd_nack (transmit width port command negative acknowledge)

Asserted when the pending xmt_width_port_cmd is different from “hold”, and has for some reason, not been executed. Once asserted, it remains asserted until the xmt_width_port_cmd is set to “hold”, at which point the variable is de-asserted; otherwise, de-asserted.

The non-execution may be due to the requested mode not being enabled or an execution failure.

xmt_width_tmr_done (transmit width timer done)

Asserted when the xmt_width_tmr has run continuously for 250 +/- 85 usec. De-asserted when xmt_width_tmr_en is de-asserted.

xmt_width_tmr_en (transmit width timer enable)

When asserted the xmt_width_tmr runs continuously. When de-asserted, the xmt_width_tmr is reset to and retains its initial (default) value.

xmting_idle (transmitting idle)

Asserted when the port has stopped transmitting control symbols and packets in response to the deassertion of transmit_enable_tw. The port shall transmit only the IDLE3 sequence when xmting_idle is asserted. Xmting_idle is deasserted when transmit_enable_tw is asserted.

The variables that get set to a value based on the most recent error-free Status/Control control codeword received by the port from its link partner shall be reset to the values in table when the Port_Initialization state machine is in the SILENT state. An Status/Control control codeword shall only be determined to be error-free when it is part of a valid Status/Control ordered sequence received from the link partner.

Table 5-14. Reset value for variable from Status/Control control codewords

Variable	reset value
from_sc_asym_mode_en	0b0
from_sc_initialized	0b0
from_sc_lane_ready[k]	0b0
from_sc_lane_silence[k]	0b0
from_sc_lane_trained[k]	0b0
from_sc_port_silence	0b0
from_sc_rcv_lanes_ready	0b0
from_sc_rcv_width	0b000
from_sc_rcv_width_link_cmd	0b000
from_sc_rcv_width_link_cmd_ack	0b0
from_sc_rcv_width_link_cmd_nack	0b0
from_sc_retrain_en	0b0
from_sc_retrain_grnt	0b0
from_sc_retrain_ready	0b0
from_sc_retraining	0b0
from_sc_xmt_1x_mode	0b0

When lane_sync or codeword_lock is de-asserted, the value of the variables derived from the Status/Control control codewords can no longer be determined accurately. The values of those variables shall behave as defined in table Table 5-15 when lane_sync or codeword_lock is de-asserted to ensure correct operation of state machines.

Table 5-15. Effects of lane_sync or codeword_lock de-assertion

Variable	Reset
from_sc_asym_mode_en	No
from_sc_initialized	No
from_sc_lane_ready[k]	Yes
from_sc_lane_silence[k]	Yes
from_sc_lane_trained[k]	Yes
from_sc_port_silence	Yes
from_sc_rcv_lanes_ready	No
from_sc_rcv_width	No
from_sc_rcv_width_link_cmd	No
from_sc_rcv_width_link_cmd_ack	No
from_sc_rcv_width_link_cmd_nack	No
from_sc_retrain_en	No
from_sc_retrain_grnt	No
from_sc_retrain_ready	No
from_sc_retraining	No
from_sc_xmt_1x_mode	No

In addition to the above, the value of Status/Control control codeword bit 34 (Transmit Status/Control ordered sequences) is used to control the transmission rate of Status/Control ordered sequences within the IDLE3 sequence. When lane_sync or codeword_lock is de-asserted, the lane shall continue to transmit Status/Control ordered sequences at the rate requested in the last correctly received Status/Control ordered sequence.

5.19.2 Frame_Lock State Machine

The recovery of DME training frame boundaries in a lane receiver shall be controlled and monitored by the Frame_Lock state machine. There shall be one Frame_Lock state machine for each lane receiver.

The Frame_Lock state machine shall be the Frame Lock state machine specified by Clause 72.6.10.4.1 of IEEE 802.3 2008 Part 5.

5.19.3 Lane Training State Machines

Two Lane_Training state machines are defined, a long run Lane_Training state machine for ports supporting the long run electrical specification (long run ports), and a short run Lane_Training state machine for ports supporting only the short run electrical specification (short run ports).

A Lane_Training state machine controls the training and retraining of all per lane adaptive equalization that is controlled by the receive end of the lane. It does not control the actual adjustment of adaptive equalizer settings which is done by implementation specific mechanisms that are beyond the scope of this specification.

The Lane_Training state machines provide two “modes” of operation, “training” mode and “retraining” mode as described in Section 5.11.1.

When the training state machines use i.e. set_xmt_equalizer(k,“LR_initialize”), it is intended to indicate that a coefficient set in this case for long run initialization is loaded into the transmitter equalizer settings. The coefficient set can be implemented as a static set of values or a set of values that is controlled through registers or by the adaptive equalization algorithm. Management of the coefficient set is outside the scope of this specification.

5.19.3.1 Long run Lane_Training State Machine

The long run Lane_Training state machine is specified in Figure 5-22 through Figure 5-24. There shall be a long run Lane_Training state machine for each lane receiver of a long run port.

The state machine supports both DME and CW training so that long run port can train adaptive transmit equalization in either short run or long run ports.

The long run Lane_Training state machine is forced into the UNTRAINED state when the Port Initialization state machine is in the SILENT state and prepares for training in DME mode. The state machine for lane k then waits for lane k to be output enabled and for either frame_lock[k] or lane_sync[k], but not both, to be asserted. If lane k is output enabled and frame_lock[k] is asserted, the state machine determines that it is connected to a long run port and training will be in DME mode. If lane k is output enabled and lane_sync[k] is asserted, the state machine determines that is connected to a short reach port and switches to CW training mode.

In both DME and CW mode, the state machine begins training by starting the training timer (train_tmr_en[k] asserted). The training timer runs continuously during both the DME and CW training processes so that if a failure occurs at any stage of either processes, the failure can be detected and the state machine can recover rather than becoming stuck due to the failure. If a failure occurs in either the DME or CW training process, training_fail[k] is set and the lane k state machine returns to the UNTRAINED state.

The state machine checks that lane_trained[k] is de-asserted, orders the training mechanism to start training (train_lane[k] asserted) and waits for the training of lane k to complete both locally (lane_trained[k] asserted) and in the connected port (from_sc_lane_trained[k] asserted).

Once training is complete in CW mode, the state machine enters the TRAINED state. In DME mode, the port waits until the number of additional DME frames specified in “DME Wait Timer” field of Port n Link Timers Control CSRs are transmitted, to ensure that the link partner sees that the port has completed training before entering the TRANIED state and switching from DME frame to codeword transmission.

Once in the TRAINED state, the state machine periodically output enables lane k, if the lane is not currently being output enabled due to the current asymmetric width, so that the connected lane k receiver can track changes due to temperature drift and any other slow moving changes, and thereby keeping the lane receiver sufficiently trained to be quickly operational when needed. The on/off duty cycle is determined by the duty cycle of keep_alive. A lane that is output enabled only by keep_alive shall transmit the IDLE3 sequence.

Retraining occurs when the variable “retrain” is asserted by the Retrain/Transmit_Width state machine. The state machine starts the retraining timer (retrain_tmr_en asserted) and verifies that lane_retraining[k] is de-asserted. It then orders the training mechanism to retrain the lane by asserting retrain_lane[k]. The state machine then waits for the retraining of lane k to complete both locally and in the connected receiver. At which point, retraining is completed and the state machine returns to the TRAINED state.

The retrain timer runs continuously during the retraining processes so that if a failure occurs at any stage of the processes, the failure can be detected and the state machine can recover rather than becoming stuck due to the failure. If a failure occurs during the retraining process, retrain_fail[k] is set and the lane k state machine stays in the RETRAIN_FAIL state until it is forced into the UNTRAINED state by the Port Initialization state machine entering SILENT state.

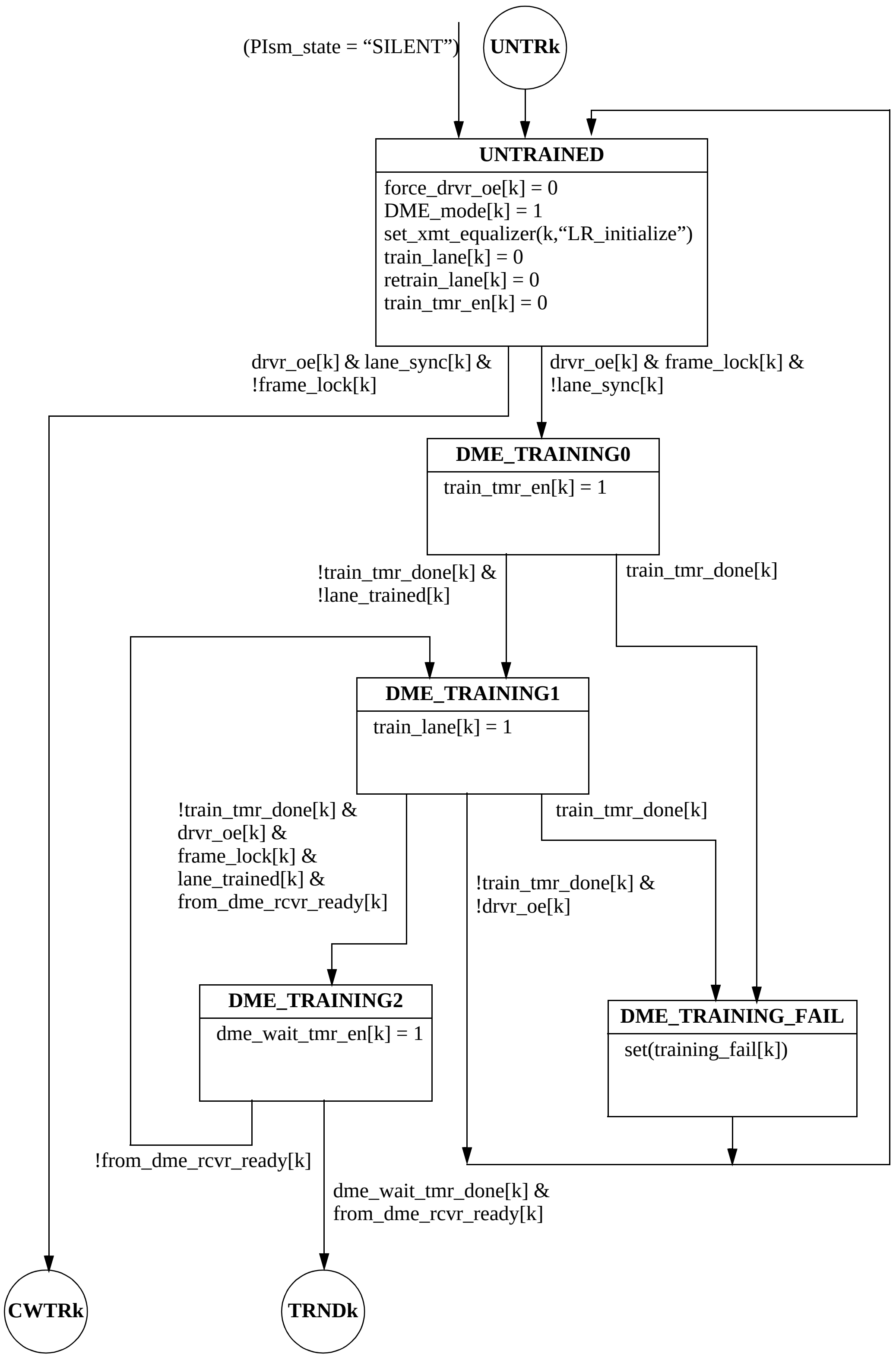


Figure 5-22. Long run Lane_Training state machine (lane k) Part 1 of 3

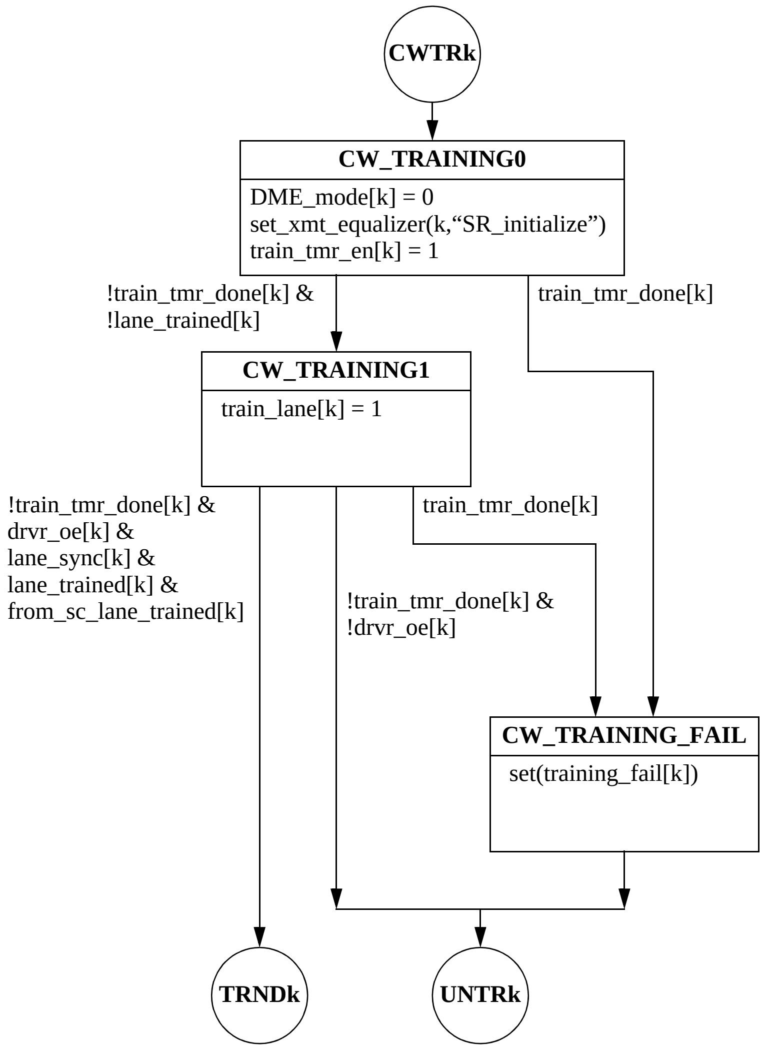


Figure 5-23. Long run Lane_Training state machine (lane k) Part 2 of 3

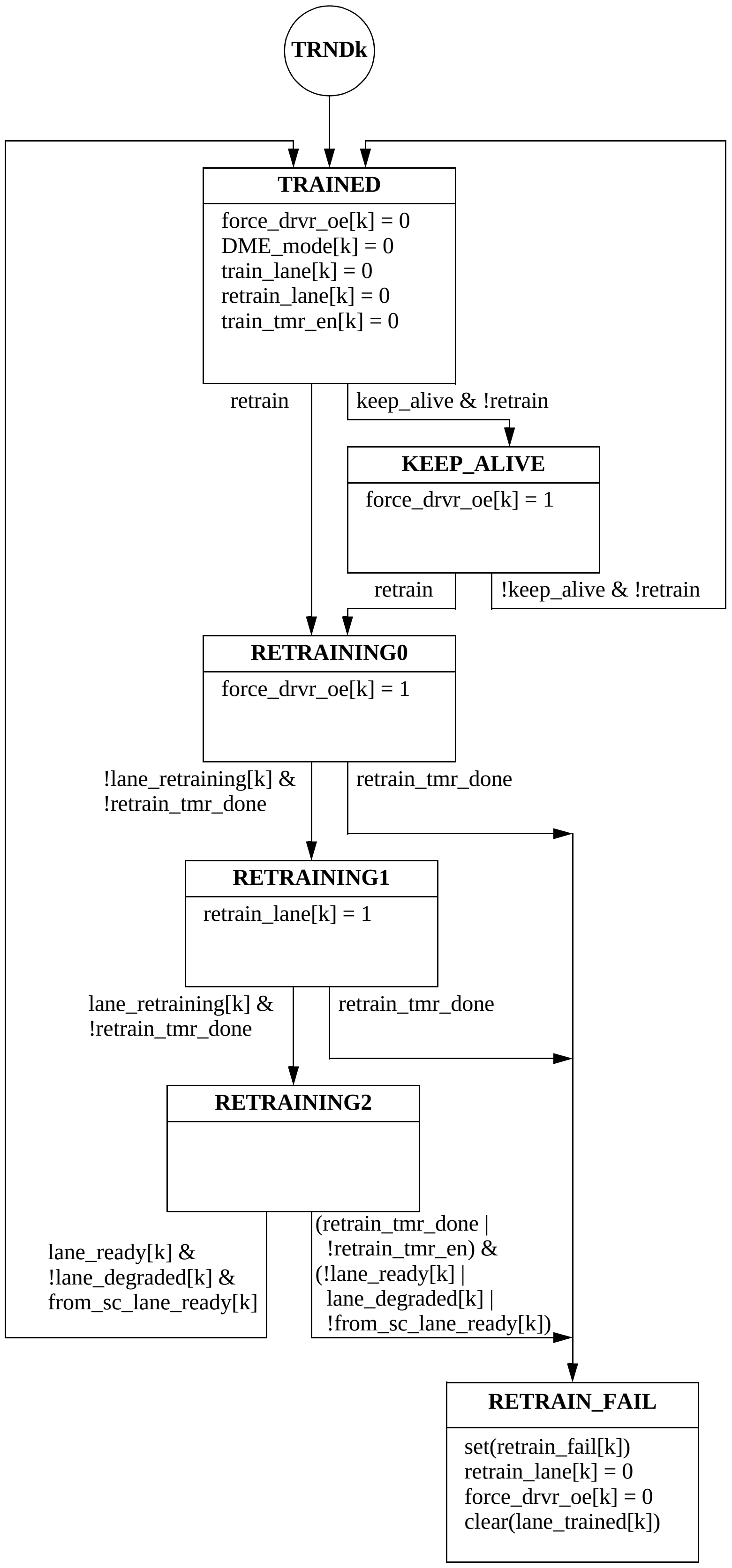


Figure 5-24. Long run Lane_Training state machine (lane k) Part 3 of 3

5.19.3.2 Short run Lane_Training state machine

The short run Lane_Training state machine is specified in Figure 5-25 through Figure 5-26. There shall be a short run Lane_Training state machine for each lane receiver of a short run port.

The short run Lane_Training state machine is essentially the long run Lane_training state machine with DME training removed.

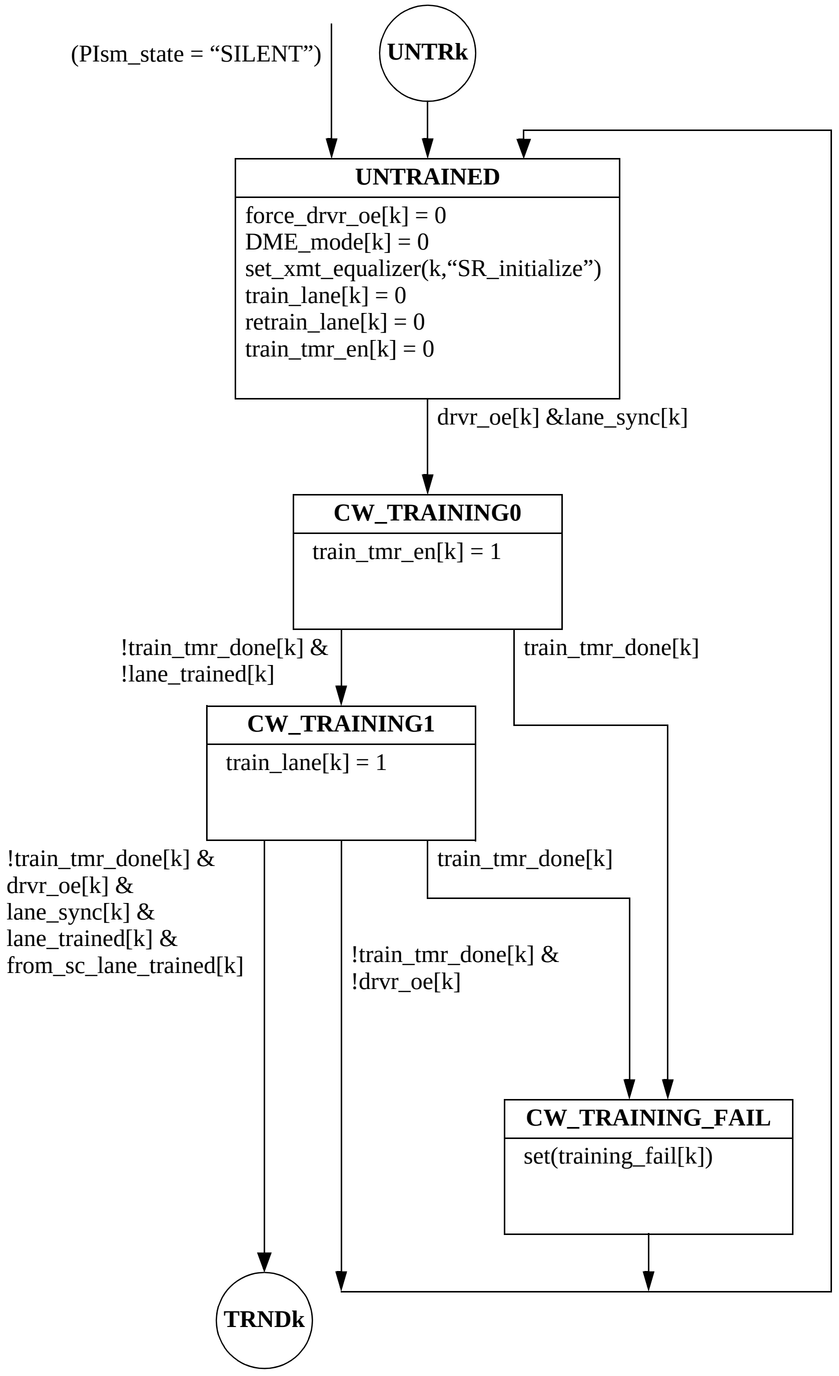


Figure 5-25. Short run Lane_Training state machine for lane k Part 1 of 2

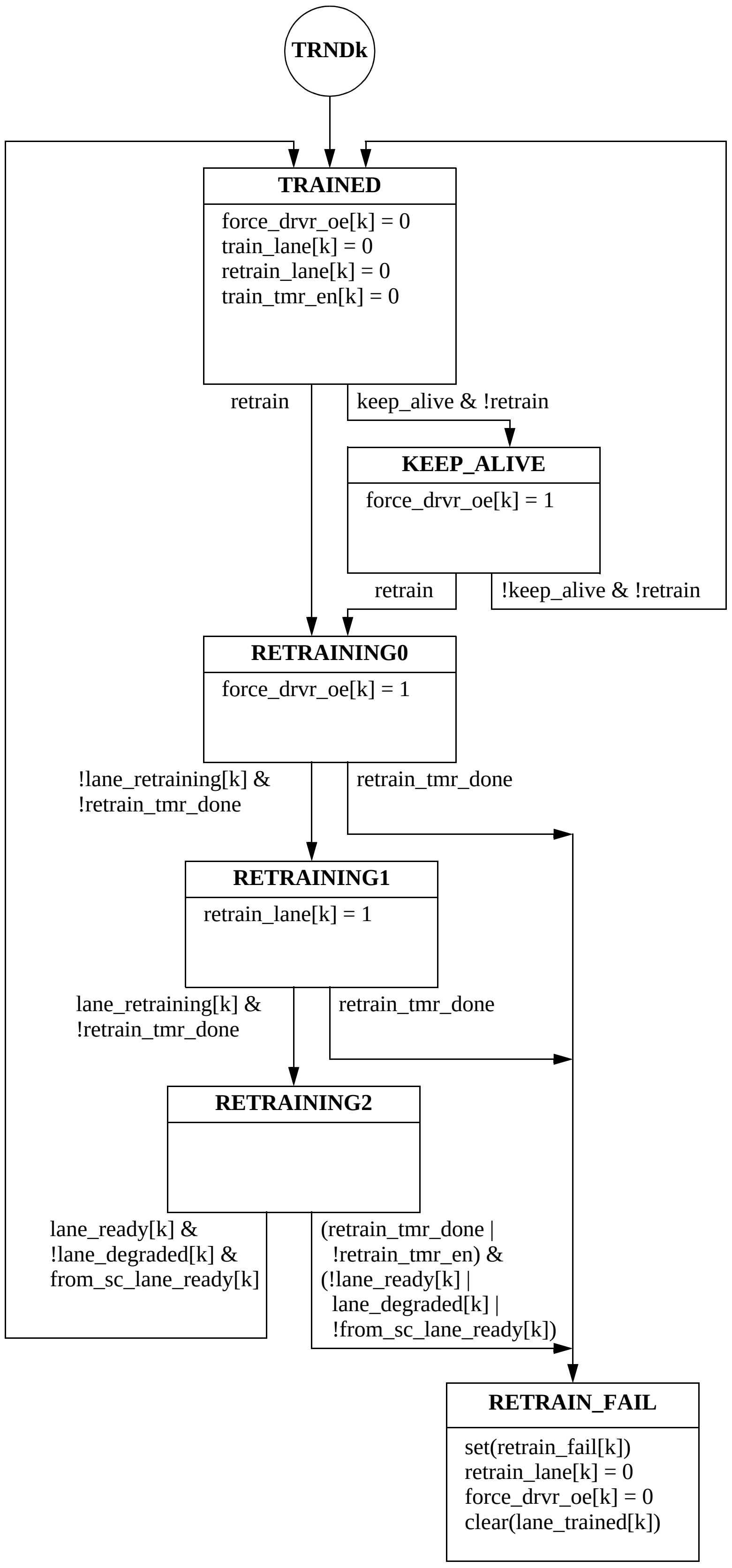


Figure 5-26. Short run Lane_Training state machine for lane k Part 2 of 2

5.19.4 Codeword Lock State Machine

Codeword boundary recovery in a lane receiver is controlled and monitored by the Codeword_Lock state machine. There shall be one Codeword_Lock state machine for each lane receiver.

Codeword boundary acquisition is based on locating the transition between codeword bits [1:2] (“!type” and “type” bits) that occurs in every 64b/67b codeword. For convenience, 64b/67b codeword bits [1:2] will be called the codeword “sync header”. (This term is adopted from IEEE 802.3 - 2008 Clause 49 10GBASE-R in which the 2-bit field that marks the beginning of a 10GBASE-R 64B/66B codeword is called the “sync header”.)

The state machine begins the search for codeword boundaries in the lane receiver’s input serial data stream by testing the output of the lane receiver’s serial to 67-bit parallel converter for a valid sync header in consecutive 67-bit codewords. Codeword boundary alignment is declared (codeword_lock asserted) if 64 consecutive codewords are found each containing a valid sync header. Codeword misalignment is declared if a codeword containing an invalid sync header occurs before 64 consecutive codewords are found each containing a valid sync header. Each invalid sync header detected causes the count of consecutive codewords containing valid sync headers to be restarted from zero.

If codeword misalignment is declared during the search for codeword boundary alignment, the serial to 67-bit parallel conversion boundaries is moved one bit earlier or one bit later in the input serial data stream, but always in the same direction, and the new alignment tested. This process is repeated until correct codeword boundary alignment is achieved.

Once codeword boundary alignment is achieved (codeword_lock is asserted), the serial to 67-bit parallel conversion boundary is no longer adjusted. The occurrence of IVmax codewords containing invalid sync headers before 64 codewords with valid sync headers occurs causes the state machine to declare loss of codeword boundary alignment (codeword_lock de-asserted). At which point, the search for codeword boundary alignment begins again.

IVmax is an integer constant that specifies the value of the IVcounter at which the state machine determines that the lane has lost codeword boundary synchronization and de-asserts codeword_lock. The greater the value of IVmax, the longer it takes for codeword_lock to be de-asserted after the loss of an input signal. IVmax shall have a minimum value of 3. The recommended value of IVmax for normal operation is 3. The value may be set higher when the lane’s adaptive equalization is being trained or retrained.

The 64b/67b Codeword_Lock state machine for lane k is shown in Figure 5-27.

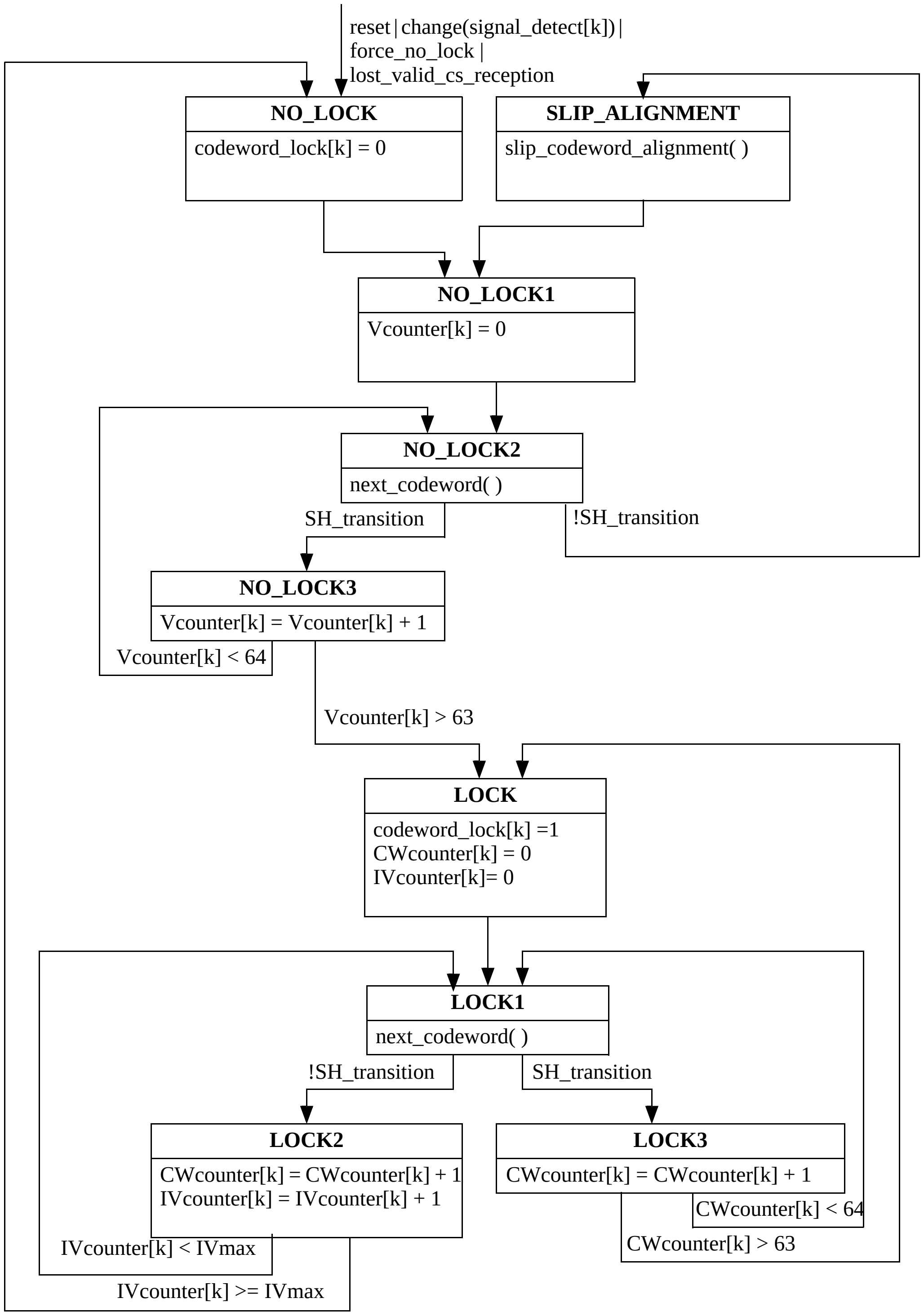


Figure 5-27. Lane k Codeword_Lock state machine

5.19.5 Lane Synchronization State Machine

After codeword boundary alignment is achieved it is needed to look at some protocol specifics to determine that the lane if fully synchronized to the incoming signal and that the incoming signal complies on a certain level to the protocol. The Descrambler Seed control codewords is used for this purpose.

After the codeword_lock[k] variable gets asserted the state machine will look for 6 Descrambler Seed control codewords that all are matching up with the internal state of the descrambler (the descrambler is in sync). When this is achieved the lane is declared to be in sync by the assertion of the lane_sync[k] variable.

The 64b/67b Lane_Synchronization state machine for lane k is shown in Figure 5-28.

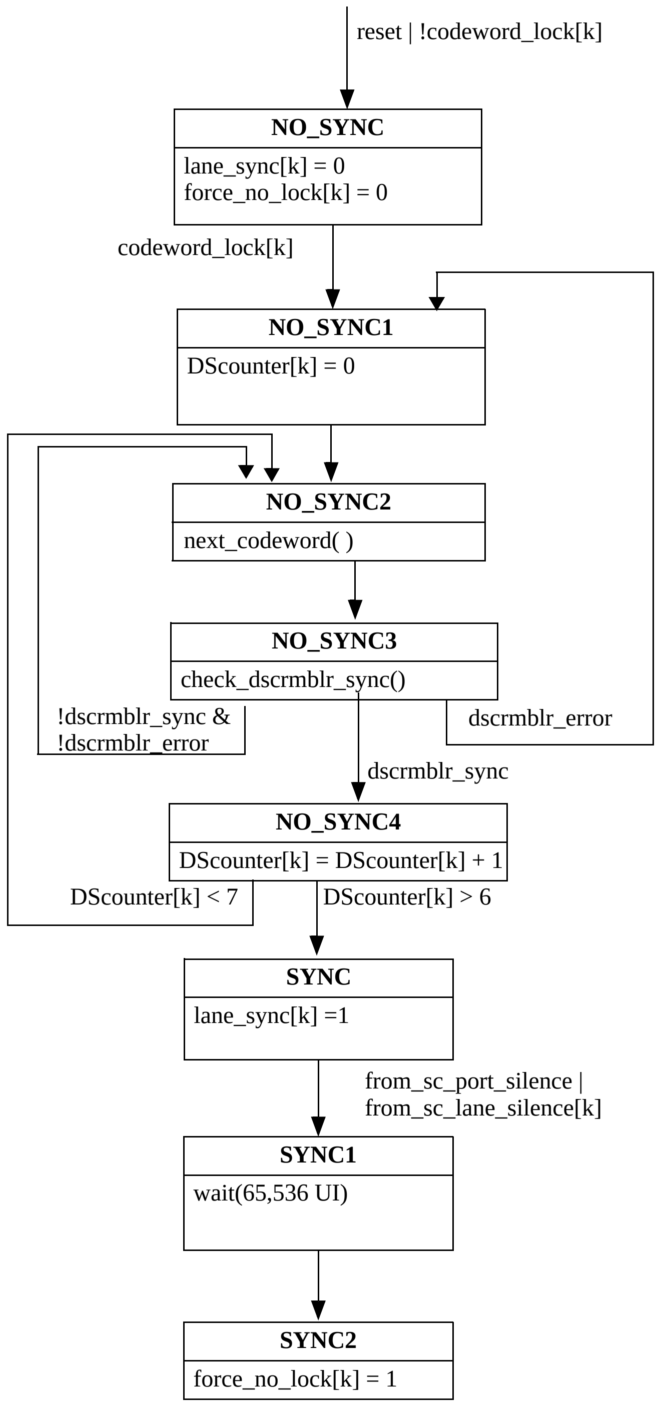


Figure 5-28. Lane k Lane_Synchronization state machine

5.19.5.1 Entering Silence

When a port or a lane is going to enter silence, meaning it is going to disable the transmitter, it is desired to do this in a controlled fashion that informs the link partner of the intent. To facilitate this there are allocated bits in the Status/Control control codeword to indicate each of these events, a lane is going to enter silence or a port is going to enter silence.

It is expected that implementations will use the signals 4_lanes_drvr_oe, 8_lanes_drvr_oe, 16_lanes_drvr_oe, lane0_drvr_oe, lanes01_drvr_oe, lanes02_drvr_oe and force_drvr_oe[k] to determine for each lane if the transmitter shall be enabled or disabled. When it is determined that the transmitter is going to be disabled it is recommended to follow the enter silence procedure described in the following. It is expected that it’s not always possible to easily control the transition to silence, i.e. a hard reset of a device can result in the transmitter being disabled immediately instead of going through the described procedure.

5.19.5.1.1 Transmitter procedure

When a port detects that a lane or port is going to enter silence it is recommended to use the following procedure when possible:

- Stop transmission of packets and control symbols to the lane(s) involved. This could be done by narrowing the port width in asymmetric mode or by stopping the flow of packets by the port via the transmit_enable signal, or by using another similar mechanism.
- After the flow of packets are stopped, set the Lane Entering Silence bit and if applicable the Port Entering Silence bit in the Status/Control control codeword transmitted to the link partner.
- Continue to send the IDLE3 sequence until either a minimum of 8 Status/Control ordered sequences have been transmitted or a maximum of 512 codewords have been transmitted.
- Disable the transmitter, and clear the drvr_oe[k] variable for the affected lanes.
- If the procedure is for a port to enter silence then enter the SILENT state of the Port Initialization state machine if its not already the current state.

5.19.5.1.2 Receiver procedure

When a lane receives the Lane Entering Silence or Port Entering Silence indication from its link partner it is recommended to use the following procedure:

- Transition from the SYNC state to the SYNC1 state of the Lane Synchronization state machine.
- If the received indicator was the Port Entering Silence then start the transmitter procedure for the port that received the Port Entering Silence indication.
- Wait for 65,536 UI, then transition to the SYNC2 state of the Lane Synchronization state machine, in which state the force_no_lock[k] variable is asserted to force the Codeword Lock state machine into its NO_LOCK state.
  The wait period is set so that it is comfortable higher than the time for the link

partner to disable the transmitter when following the procedure described in the previous section. This is to guarantee that there is not an active signal being received when entering the NO_LOCK state.

5.19.6 Lane Alignment State Machine

For a number of reasons, the lanes of a multi-lane LP-Serial link will have different propagation delays with the result that the lanes must be realigned in the receiver before the lanes can be destriped. The Lane_Alignment state machine monitors the alignment of the received lanes and determine whether the lanes are aligned. A receiver shall have a Lane_Alignment state machine for each multi-lane link width supported by the receiver as each Lane_Alignment state machine is width specific. The Lane_Alignment state machine for Nx mode is specified in Figure 5-29 and Figure 5-30. The method for achieving alignment of the N lanes is implementation specific and outside the scope of this specification.

Status/Control ordered sequences, which are transmitted in columns when a port is transmitting in a multi-lane mode, are used to acquire and monitor lane alignment in the receiver. When received, the Status/Control ordered sequences may be misaligned due to lane to lane differences in propagation delay. Status/Control control codewords may be corrupted by transmission errors, and so may be received as a different codeword. Similarly, other codewords may be corrupted by transmission errors and become Status/Control control codewords.

The Lane_Alignment state machine looks for pairs of sequential columns, the first of which contains at least one Status/Control control codeword. Once a column containing at least one Status/Control control codeword is found, that column and the immediately following column are examined as a pair for the pattern of Status/Control control codewords they contain. To limit complexity, each column is characterized as containing all, some or no Status/Control control codewords.

Using the above characterization of columns, it is at best difficult to distinguish between some cases of misalignment and of codeword corruption by a transmission error. Examination of multiple pairs of columns is used to distinguish between misalignment and corruption. Misalignment indicators repeat in multiple pairs of columns whereas corruption does not.

To simplify the handling of such ambiguity, the state machine uses different algorithms for determining when the N lanes are aligned and when alignment is lost. The state machine requires four error-free and correctly aligned Status/Control ordered sequences to determine that the N lanes are aligned. However, once the state machine has determined that the N lanes are aligned, it tolerates occasional transmission errors by requiring four indications of misalignment in a short period of time before determining that the N lanes are misaligned.

The Lane_Alignment state machine is specified in Figure 5-29 and Figure 5-30. There shall be one Lane_Alignment state machine for each multi-lane width supported by the port.

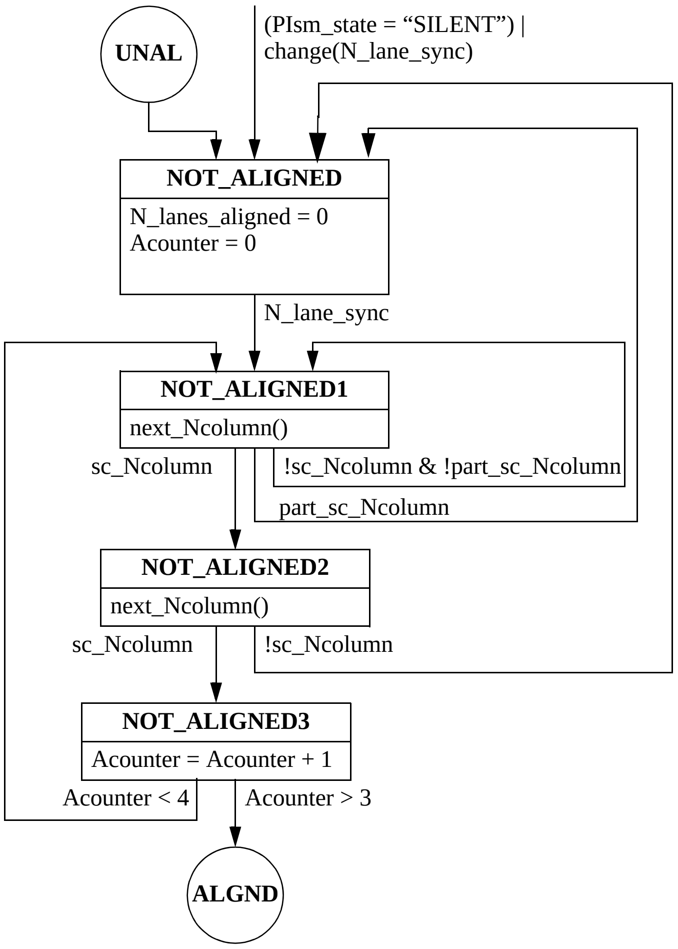


Figure 5-29. N-lane Lane_Alignment State Machine (Part 1 of 2)

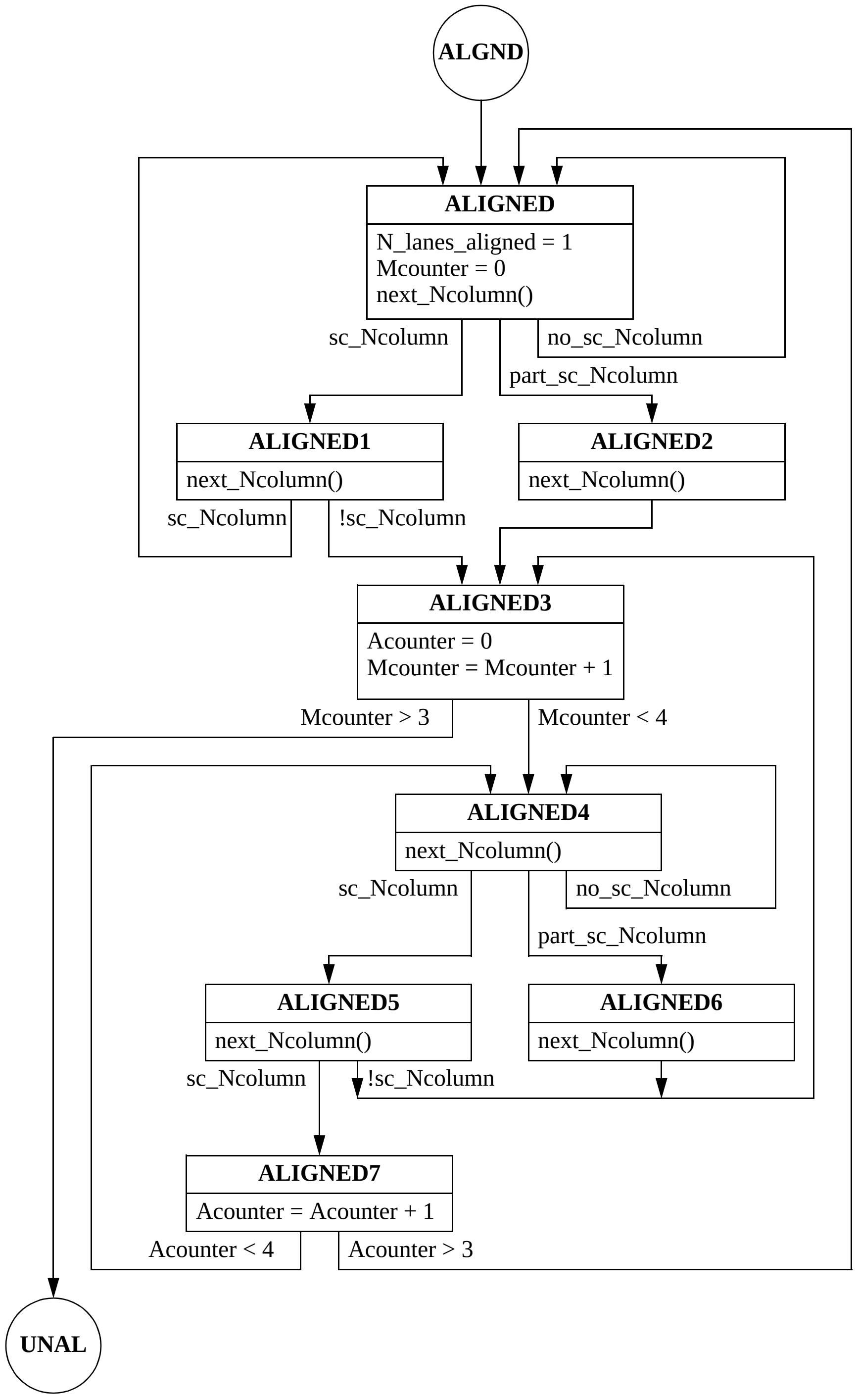


Figure 5-30. N-lane Lane_Alignment State Machine (Part 2 of 2)

5.19.7 Port Initialization State Machine

The Port_Initialization state machine is the primary state machine that controls bringing a port from reset to an operational state in which the port may transmit and receive control symbols and packets. There shall be one Port_Initialization state machine per port.

The generalized Port_Initialization state machine for 64b/67b encoded links is a modest modification of the Port_Initialization state machine for 8b/10b encoded links. The modifications include support for asymmetric operation, separate discovery and recovery timers, and delaying the entry into the 2x_MODE state to make sure that the connected port is transmitting in 2x, not 1x mode on lanes 0 and 1.

Upon entering the ASYM_MODE state, the Port_Initialization state machine passes control over transmit and receive widths to the Transmit_Width and Receive_Width state machines, respectively. If there is a failure that the Transmit_Width and/or Receive_Width state machines cannot handle, control is returned to the Port_Initialization state machine and the port is reinitialized.

There are 16 variants of the Port_initialization state machine, one supporting only 1x mode, four supporting 1x mode and one multi-lane mode, six supporting 1x mode and two multi-lane modes, four for 1x mode and 3 multi-lane modes, and one supporting all five specified link widths. State machine diagrams are defined for only the 1x/2x/Nx one of these variants. These state machine diagrams shall be used as guidance for the construction of state machines for the remaining variants.

5.19.7.1 1x/2x/Nx Initialization State Machine

The Port_Initialization state machine for a 1x/2x/Nx mode port is shown in Figure 5-31 through Figure 5-33.

The assertion of reset or force_reinit forces the state machine to enter the SILENT state regardless of the state machines current state. In SILENT, the state machine disables all lane transmitter outputs, initializes a number of variables and waits for the “silent” period to end. The silent period is used to force the lane partner, when present, to also reinitialize.

At the end of the “silent” period, the state machine enters the SEEK state and output enables the drivers for lane 0 and lane 1 and/or lane 2, depending on which of the multi-lanes modes supported by the port are enabled. This announces the port’s presence on the link. The state machine then waits for the assertion of frame_lock or lane_sync for lane 0, 1, or 2 announcing the presence of a link partner. When the presence of a link partner is detected, the state machine enters the DISCOVERY state.

In the DISCOVERY state, the state machine output enables all of the lanes required for the enabled multi-lane modes and starts the discovery timer (disc_tmr_en asserted). The state machine then monitors the lane_sync, lane_ready and the number of lanes that are being received and that can be aligned. The state machine chooses which width mode to enter based on the number of lanes being received and that are aligned. With a few exceptions, this decision is made at the end of the discovery period (disc_tmr_done asserted) to ensure that a decision is not made before all lanes being received have had a chance to be trained and aligned.

Once in a width mode, the port is declared initialized (port_initialized asserted), lanes not used by the current width mode are output disabled, and several variables characterized the width mode are set. If a lane needed for reception of the current width mode goes "not ready" (lane_ready[x] de-asserted) but at least one of the redundancy lanes (lane 0 or lane R) is still being received (lane-sync[0] or lane_sync[R] asserted), the state machine enters the recovery state for that width mode (for example, the state machine enters 2X_RECOVERY from 2X_MODE). If lane_sync is lost (lane_sync de-asserted) on both of the redundancy lanes, the state machine determines that the link partner is no longer active and enters the SILENT state to attempt to reinitialize.

A state machine in the Kx_RECOVERY state disables reception of incoming traffic, starts the recovery timer and attempts to recover to Kx_MODE. If the port has not recovered to Kx_MODE by the time the recovery time is up, the state machine attempts to recover to a narrower link width or another lane in 1x mode. If that fails the state machine enters the SILENT state. And if at any time in the recovery state lane_sync is lost on both of the redundancy lanes, the state machine immediately enters the SILENT state.

When in a multi-lane width mode state, the state machine transitions to ASYM_MODE if asymmetric mode is enabled in both ports and both ports are initialized to the same port width.

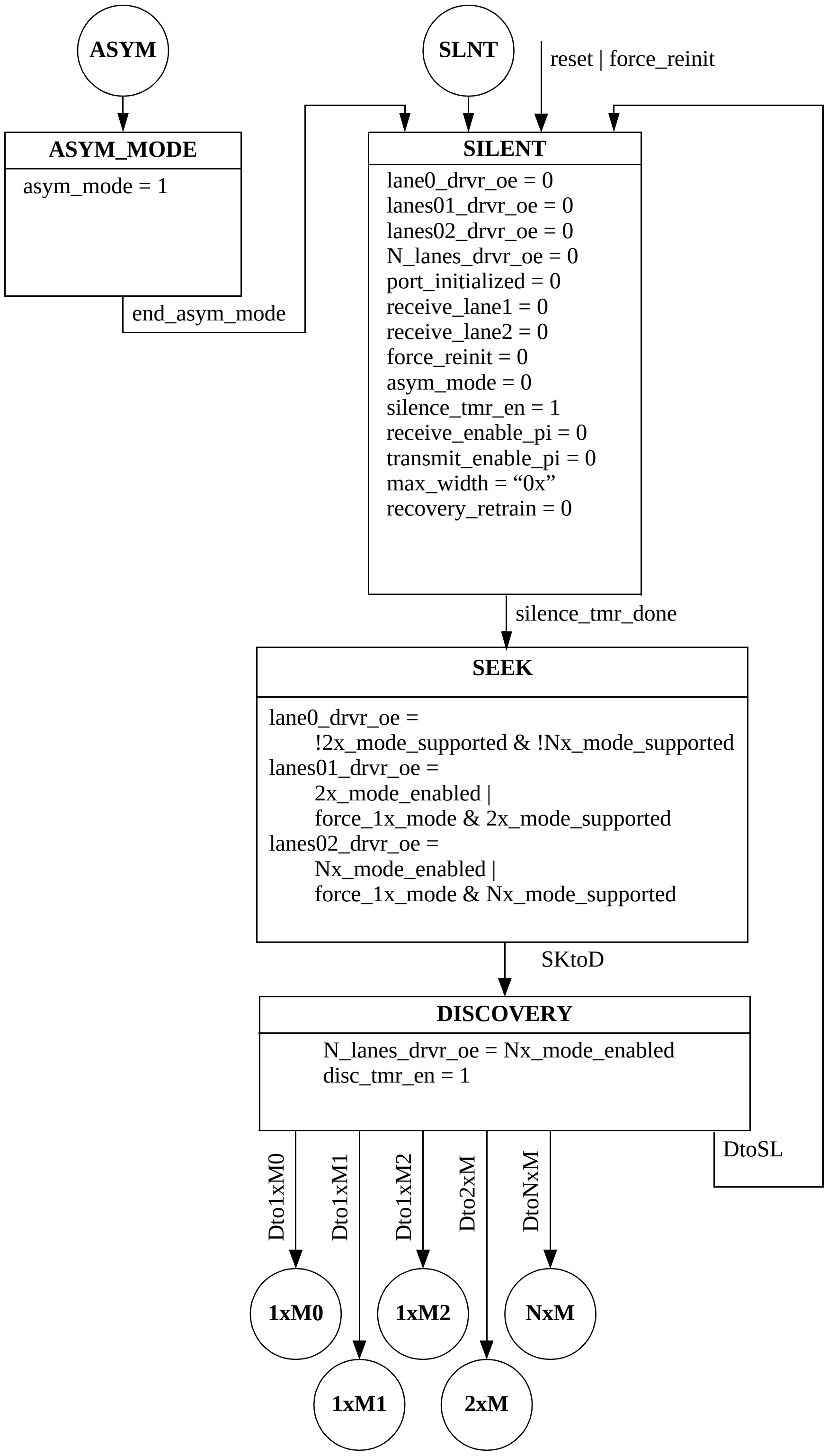


Figure 5-31. 1x/2x/Nx Port_Initialization State Machine, Part 1 of 3

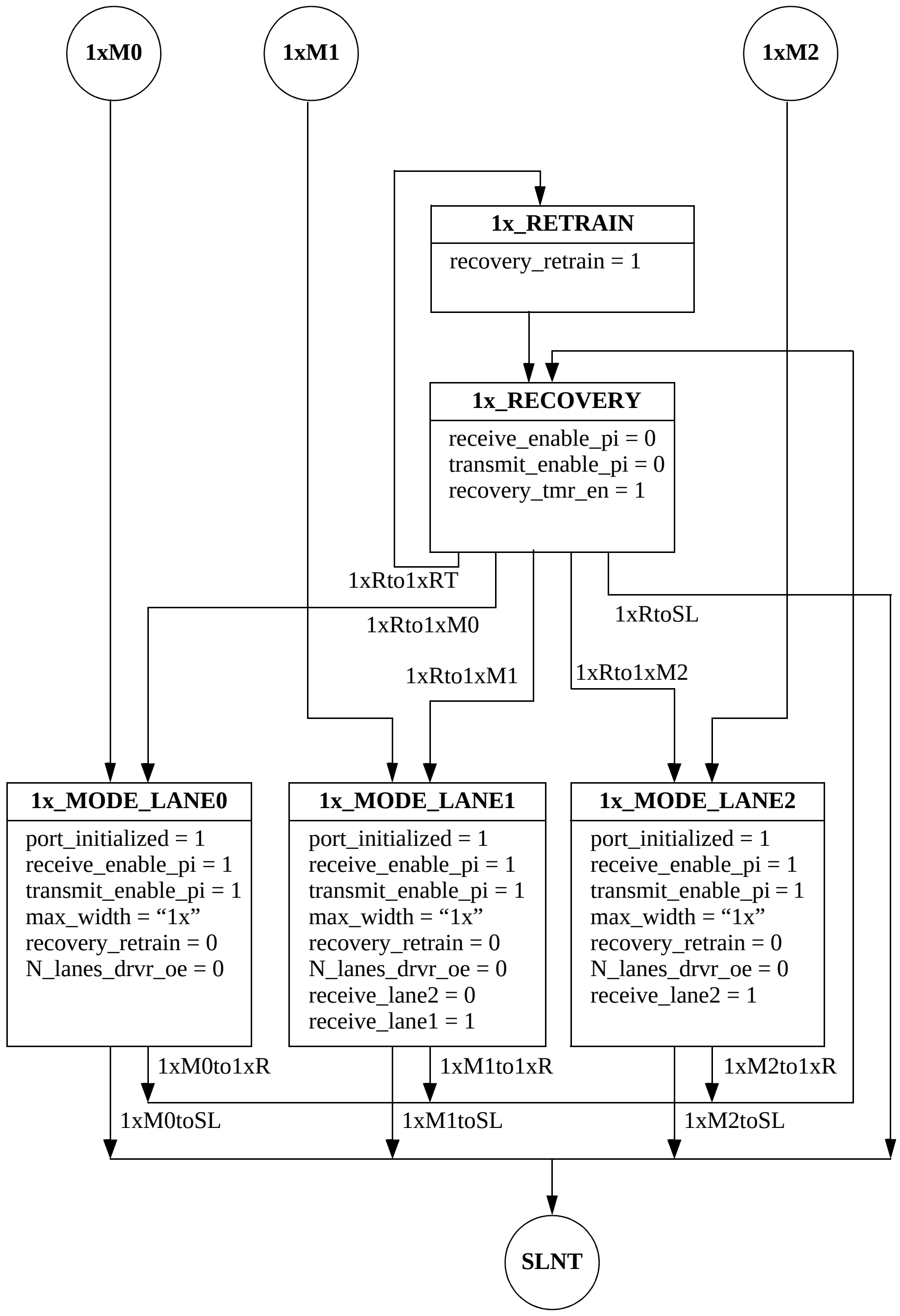


Figure 5-32. 1x/2x/Nx Port_Initialization state machine, Part 2 of 3

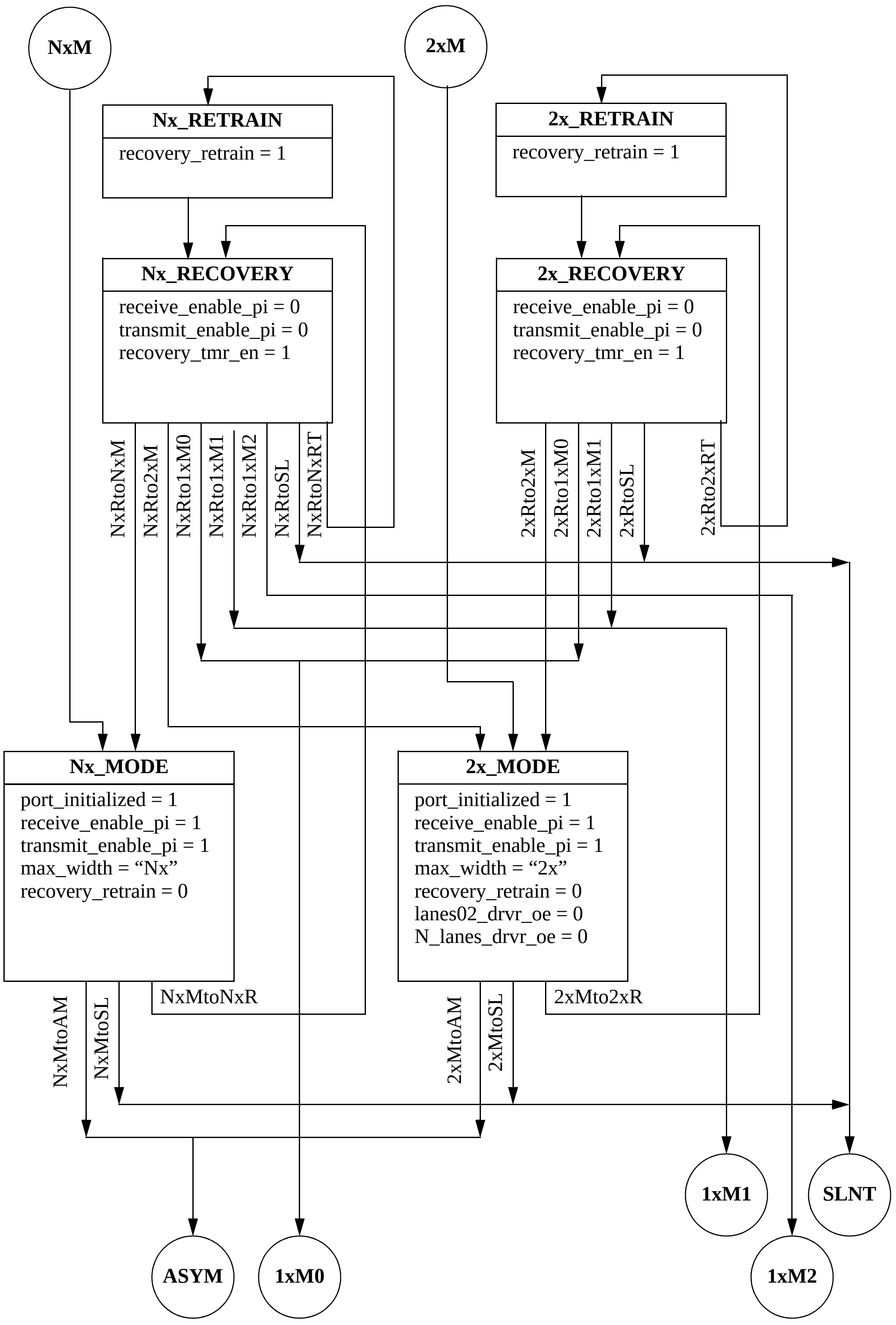


Figure 5-33. 1x/2x/Nx Port_Initialization state machine, Part 3 of 3

The equations for the state transition conditions for the 1x/2x/Nx Port_Initialization state machine are as follows:

```
1xM0to1xR = !lane_ready[0] & lane_sync[0]

1xM0toSL = !lane_sync[0]

1xM1to1xR = !lane_ready[1] & lane_sync[1]

1xM1toSL = !lane_sync[1]

1xM2to1xR = !lane_ready[2] & (lane_sync[1] | lane_sync[2])

1xM2toSL = !lane_sync[2] & !lane_sync[1]

1xRto1xM0 = !recovery_tmr_done & !receive_lane1 & !receive_lane2 &
            lane_ready[0] & (!retrain | recovery_retrain)

1xRto1xM1 = !recovery_tmr_done &
            (receive_lane1 |
             receive_lane2 & !lane_ready[2] &
               (2x_mode_enabled | force_1x_mode & 2x_mode_supported)) &
            lane_ready[1] & (!retrain | recovery_retrain)

1xRto1xM2 = !recovery_tmr_done & receive_lane2 & lane_ready[2] &
            (!retrain | recovery_retrain)

1xRto1xRT = retrain & !recovery_retrain

1xRtoSL = !lane_sync[0] &
            !(lane_sync[1] & 2x_mode_supported) &
            !(lane_sync[2] & 4x_mode_supported) | recovery_tmr_done

2xMtoAM = asym_mode_en & from_sc_asym_mode_en &
            port_initialized & from_sc_initialized &
            (from_sc_rcv_width = “2x mode”) &
            2_lanes_ready & !from_sc_xmt_1x_mode

2xMto2xR = !2_lanes_ready & (lane_sync[0] | lane_sync[1])
            | 2_lanes_ready & from_sc_xmt_1x_mode

2xMtoSL = !lane_sync[0] & !lane_sync[1]

2xRto1xM0 = (recovery_tmr_done & !2_lanes_ready & lane_ready[0] |
             2_lanes_ready & from_sc_xmt_1x_mode) &
            xmting_idle & (!retrain | recovery_retrain)

2xRto1xM1 = (recovery_tmr_done & !2_lanes_ready & !lane_ready[0] &
             lane_ready[1]) &
            xmting_idle & (!retrain | recovery_retrain)

2xRto2xM = 2_lanes_ready & !from_sc_xmt_1x_mode &
            (!retrain | recovery_retrain)

2xRto2xRT = retrain & !recovery_retrain
```

```
2xRtoSL = !lane_sync[0] & !lane_sync[1] |
                recovery_tmr_done & !lane_ready[0] & !lane_ready[1]

Dto1xM0 = lane_ready[0] &
                (force_1x_mode &
                  (!force_laneR |
                   force_laneR & disc_tmr_done &
                     !(lane_ready[1] & 2x_mode_supported) &
                     !(lane_ready[2] & Nx_mode_supported)
                  ) |
                 !force_1x_mode & disc_tmr_done  &
                  !(Nx_mode_enabled & N_lanes_ready) &
                  (!(2x_mode_enabled & 2_lanes_ready) |
                   2_lanes_ready & from_sc_xmt_1x_mode
                  ) |
                 !2x_mode_supported & !Nx_mode_supported
                )

Dto1xM1 = lane_ready[1] &
                (force_1x_mode & 2x_mode_supported &
                  !(Nx_mode_supported & lane_ready[2]) &
                  (force_laneR & (!Nx_mode_supported | disc_tmr_done & !lane_ready[2]) |
                   !force_laneR & disc_tmr_done & !lane_ready[0] &
                     !(Nx_mode_supported & lane_ready[2])) |
                 !force_1x_mode & 2x_mode_enabled &
                  !(Nx_mode_enabled & lane_ready[2]) &
                  disc_tmr_done & !lane_ready[0]
                )

Dto1xM2 = lane_ready[2] &
                (force_1x_mode & Nx_mode_supported &
                  (force_laneR | !force_laneR & disc_tmr_done & !lane_ready[0]) |
                 !force_1x_mode & Nx_mode_enabled &
                  disc_tmr_done & !lane_ready[0]
                )

Dto2xM = 2x_mode_enabled & 2_lanes_ready &
                (disc_tmr_done | !Nx_mode_enabled) &
                !from_sc_xmt_1x_mode & !(Nx_mode_enabled & N_lanes_ready)

DtoNxM = Nx_mode_enabled & N_lanes_ready

DtoSL = disc_tmr_done & !lane_ready[0] &
                !(lane_ready[1] &
                  (2x_mode_enabled | force_1x_mode & 2x_mode_supported)
                ) &
                !(lane_ready[2] &
                  (Nx_mode_enabled | force_1x_mode & Nx_mode_supported)
                )

NxMtoAM = asym_mode_en & from_sc_asym_mode_en &
                port_initialized & from_sc_initialized &
                (from_sc_rcv_width = “Nx mode”) &
                N_lanes_ready
```

```
NxMtoNxR = !N_lanes_ready & (lane_sync[0] |
                 (lane_sync[1] & 2x_mode_enabled) | lane_sync[2])

NxMtoSL = !lane_sync[0] &
                 !(lane_sync[1] & 2x_mode_enabled) &
                 !lane_sync[2]

NxRto1xM0 = recovery_tmr_done  & lane_ready[0] & !N_lanes_ready &
                 (!(2x_mode_enabled & 2_lanes_ready) |
                   2_lanes_ready & from_sc_xmt_1x_mode
                 ) & xmting_idle & (!retrain | recovery_retrain)

NxRto1xM1 = recovery_tmr_done & !lane_ready[2] & !lane_ready[0] &
                 lane_ready[1] & 2x_mode_enabled & xmting_idle &
                 (!retrain | recovery_retrain)

NxRto1xM2 = recovery_tmr_done & lane_ready[2] & !lane_ready[0] & xmting_idle &
                 (!retrain | recovery_retrain)

NxRto2xM = 2x_mode_enabled & 2_lanes_ready & recovery_tmr_done &
                 !from_sc_xmt_1x_mode & !N_lanes_ready & xmting_idle &
                 (!retrain | recovery_retrain)

NxRtoNxM = N_lanes_ready & (!retrain | recovery_retrain)

NxRtoNxRT = retrain & !recovery_retrain

NxRtoSL = !lane_sync[0] & !lane_sync[2] &
                 !(lane_sync[1] & 2x_mode_enabled) |
                 recovery_tmr_done & !lane_ready[0] & !lane_ready[2] &
                 !(lane_ready[1] & 2x_mode_enabled)

SKtoD = frame_lock[0] ^ lane_sync[0] |
                 (2x_mode_enabled | force_1x_mode & 2x_mode_supported) &
                   frame_lock[1] ^ lane_sync[1] |
                 (Nx_mode_enabled | force_1x_mode & Nx_mode_supported) &
                   frame_lock[2] ^ lane_sync[2]
```

5.19.8 Retrain/Transmit_Width_Control State Machine

The Retrain/Transmit_Width_Control state machine provides two functions:

- It serializes link retraining and transmit width change operations on a link operating in asymmetric mode to avoid any interaction between the two operations.
- It ensures that control symbol and packet transmission is suspended in both link directions while any of the link's lanes are being retrained.

These functions are combined into a single machine to minimize the total number of states required.

Retraining and transmit width change operations are serialized by requiring pending retraining and transmit width change operations to arbitrate for permission to execute, and once an operation has been granted permission to execute, not conducting another arbitration until the current operation completes execution.

Ensuring that control symbol and packet transmission is suspended in both link directions during retraining is achieved as follows:

1. The port initiating the retraining signals the connected port that it is has granted retraining permission to execute.
2. The initiating port then waits for the connected port to signal that it also granted retraining permission to execute.
3. The initiating port suspends control symbols and packets transmission and signals the connected port that it is ready to retrain.
4. The initiating port waits for the connected port to signal that its has also suspended control symbol and packet transmission and is also ready to retrain.
5. Both ports then retrain.

Once both ports have indicated they are ready to retrain, the state machine verifies that no port is currently retraining (retraining is de-asserted), orders all trained lanes to retrain (retrain asserted), verifies that retraining has begun (retraining asserted), and waits for both the port and the link partner to complete retraining. At which point, the state machine returns to the IDLE state.

Unlike the lane training operation which has a dead man timer for each lane, retraining shares a single retraining dead man timer (retrain_tmr) across all lanes. The timer is used to prevent a failure at any stage of the retraining process from hanging the state machine in some intermediate state.

All lanes that are used by the port when it achieves port_initialized status (indicated by max_width) participate in retraining. The failure of any of the lanes participating in retraining to retrain successfully within the retraining time will cause the port to downgrade the port width.

The signaling between the connected ports is through dedicated bits in the Status/Control control codewords transmitted in Status/Control ordered sequences by the ports.

The Retrain/Xmt_Width_Control state machine is specified in Figure 5-34. Each port shall have a Retrain/Xmt_Width_Control state machine.

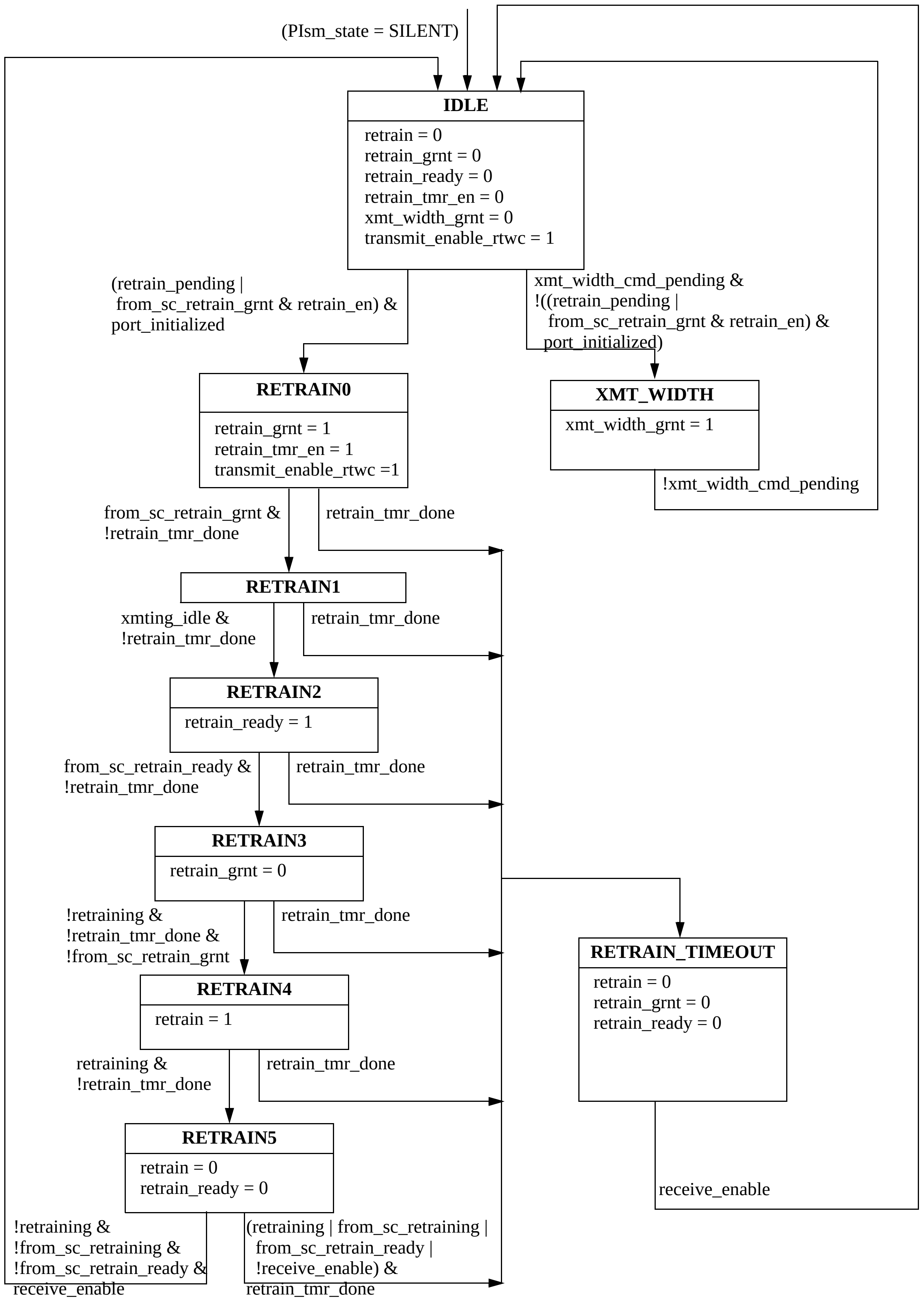

(PIsm_state = SILENT)
IDLE
retrain = 0
retrain_grnt = 0
retrain_ready = 0
retrain_tmr_en = 0
xmt_width_grnt = 0
transmit_enable_rtwc = 1
(retrain_pending | from_sc_retrain_grnt & retrain_en) & port_initialized
xmt_width_cmd_pending & !((retrain_pending | from_sc_retrain_grnt & retrain_en) & port_initialized)
RETRAIN0
retrain_grnt = 1
retrain_tmr_en = 1
transmit_enable_rtwc =1
XMT_WIDTH
xmt_width_grnt = 1
!xmt_width_cmd_pending
from_sc_retrain_grnt & !retrain_tmr_done
retrain_tmr_done
RETRAIN1
xmting_idle & !retrain_tmr_done
retrain_tmr_done
RETRAIN2
retrain_ready = 1
from_sc_retrain_ready & !retrain_tmr_done
retrain_tmr_done
RETRAIN3
retrain_grnt = 0
!retraining & !retrain_tmr_done & !from_sc_retrain_grnt
retrain_tmr_done
RETRAIN_TIMEOUT
retrain = 0
retrain_grnt = 0
retrain_ready = 0
RETRAIN4
retrain = 1
retraining & !retrain_tmr_done
retrain_tmr_done
RETRAIN5
retrain = 0
retrain_ready = 0
receive_enable
!retraining & !from_sc_retraining & !from_sc_retrain_ready & receive_enable
(retraining | from_sc_retraining | from_sc_retrain_ready | !receive_enable) & retrain_tmr_done

Figure 5-34. Retrain/Xmt_Width_Control state machine

5.19.9 Transmit Width State Machines

Transmit width commands are received by a port from possibly multiple and unspecified sources and transmit width command acknowledgements are returned to those sources. The handling of transmit width commands received by a port is shared between the Transmit_Width and the Transmit_Width_Cmd state machines. The Transmit_Width state machine handles the transmit width command if it is executable. The Transmit_Width_Cmd state machine handles the command if it is not executable and handles the final stages of the command/acknowledgement protocol for all transmit width commands.

There shall be one Transmit_Width_Cmd state machine and one Transmit_Width state machine per port.

5.19.9.1 Transmit_Width_Cmd State Machine

The Transmit_Width_Cmd state machine checks each transmit width port command received by a port for executability. A transmit width port command is not executable if the port is not in asymmetric mode, the requested width is not enabled or the requested width is greater than the maximum symmetric width of the link at the time the port was last initialized. If the command is not executable (bad_xmt_width_cmd asserted), the state machine negatively acknowledges (NACKS) the command. Regardless of whether or not a transmit width command is executable, the state machine enforces the final stages of the transmit width port command/acknowledgement protocol by keeping xmt_width_link_cmd_ack (ACK) or xmt_width_link_cmd_nack (NACK) asserted until the transmit width port command is de-asserted.

The Transmit_Width_Cmd state machine is specified in Figure 5-35.

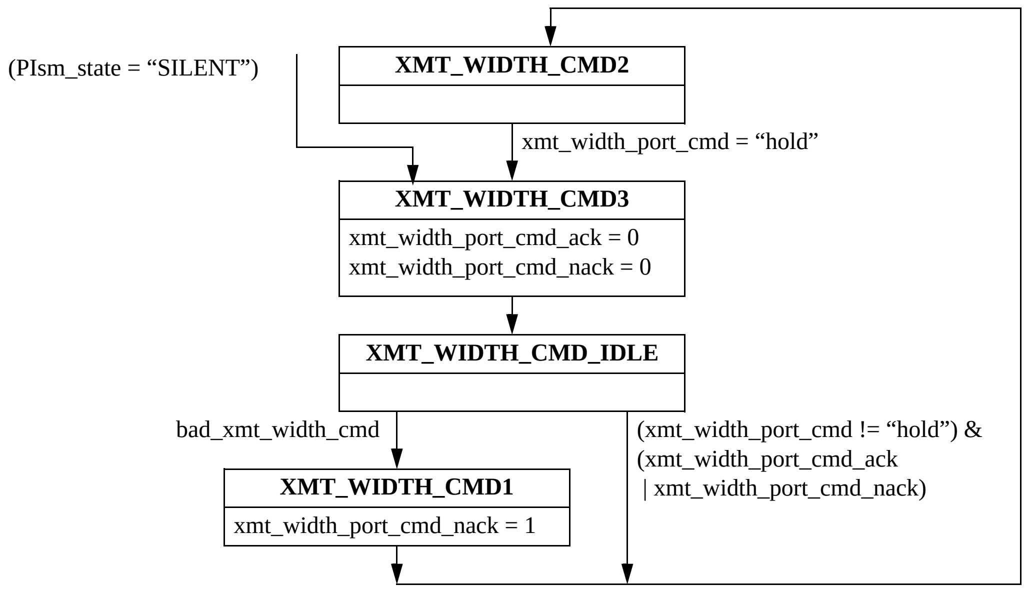

(PIsm_state = "SILENT")
XMT_WIDTH_CMD2
xmt_width_port_cmd = "hold"
XMT_WIDTH_CMD3
xmt_width_port_cmd_ack = 0
xmt_width_port_cmd_nack = 0
XMT_WIDTH_CMD_IDLE
bad_xmt_width_cmd
(xmt_width_port_cmd != "hold") &
(xmt_width_port_cmd_ack
| xmt_width_port_cmd_nack)
XMT_WIDTH_CMD1
xmt_width_port_cmd_nack = 1

Figure 5-35. Transmit_Width_Cmd state machine

5.19.9.2 Transmit_Width state machine

Control of a port's transmit width is transferred from the Port_Initialization state machine to the port's Transmit_Width state machine when the Port_Initialization state machine enters ASYM_MODE (asym_mode asserted).

When an executable transmit width port command is received by a port, the Transmit_Width state machine attempts to switch to the requested transmit width. The state machine begins by starting the transmit width timer (xmt_width_tmr_en asserted) and output enabling the drivers for the lanes needed for requested width. When transitioning from a narrower to a wider width, 64b/67b compliant data sufficient to allow the link partner to achieve frame lock and lane alignment shall be sent on the newly enabled lanes. Some examples of 64b/67b compliant data are the IDLE3 sequence, and the data pattern sent on lane 0 of the port.

The transmit width timer runs during the entire transmit width change process. If any stage of the process fails to complete before the transmit width timer times out, the transmit width port command is NACKed and the state machine either restores the port to its current transmit width or, if that does not appear to be possible, it forces the port to reinitialize (end_asym_mode is asserted). The transmit width timer is used to prevent the Transmit_Width state machine from becoming stuck part way though a transmit width change operation.

When the lanes needed for the new width become ready, the state machine halts the transmission of control symbols and packets by deasserting transmit_enable_tw and waiting for control symbol and packet transmission to end as indicated by xmting_idle.

When the transmission of control symbols and packets end, a receive width link command is sent in the “Receive width command” field of Status/Control control codewords send to the connected port to switch to the new receive width.

If the connected port ACKs the receive width link command and the receive width it reports in the “Receive width” of Status/Control control codewords that it transmits matches the requested receive width, the state machine switches to the new transmit width, changes the receive width link command to “hold”, ACKs the transmit width port command, output disables any lanes not needed for the new transmit width, and re-enables control symbol and packet transmission (transmit_enable_tw asserted).

If the connected port NACKs the receive width link command, and the receive width it reports in the “Receive width” field of Status/Control control codewords that it transmits matches the current transmit width, the state machine output disables the lanes that were output enabled for the requested transmit width, NACKs the transmit width port command, and re-enables control symbol and packet transmission at the current transmit width.

The Transmit_Width state machine is specified in Figure 5-36 through Figure 5-39. The portion of the state machine that is specific to a given receive width shall be implemented only if that width mode is supported by the port. For example, if a port supports only 1x and 4x modes, only width specific portions of the Receive_Width state machine for 1x and 4x modes shall be implemented. The width specific portions for 2x, 8x, and 16x modes shall not be implemented.

Additional variables that are local to the Transmit_Width state machine are defined as follows:

```
1x_mode_xmt_cmd = asym_mode & (xmt_width_port_cmd = “1x mode”) &
                  !xmt_width_port_cmd_ack & !xmt_width_port_cmd_nack

2x_mode_xmt_cmd = asym_mode & (xmt_width_port_cmd = “2x mode”) &
                  asym_2x_mode_enabled & !xmt_width_port_cmd_ack &
                  !xmt_width_port_cmd_nack

4x_mode_xmt_cmd = asym_mode & (xmt_width_port_cmd = “4x mode”) &
                  asym_4x_mode_enabled & (max_width >= “4x”) &
                  !xmt_width_port_cmd_ack & !xmt_width_port_cmd_nack

8x_mode_xmt_cmd = asym_mode & (xmt_width_port_cmd = “8x mode”) &
                  asym_8x_mode_enabled & (max_width >= “8x”) &
                  !xmt_width_port_cmd_ack & !xmt_width_port_cmd_nack

16x_mode_xmt_cmd = asym_mode & (xmt_width_port_cmd = “16x mode”) &
                  asym_16x_mode_enabled & (max_width >= “16x”) &
                  !xmt_width_port_cmd_ack & !xmt_width_port_cmd_nack
```

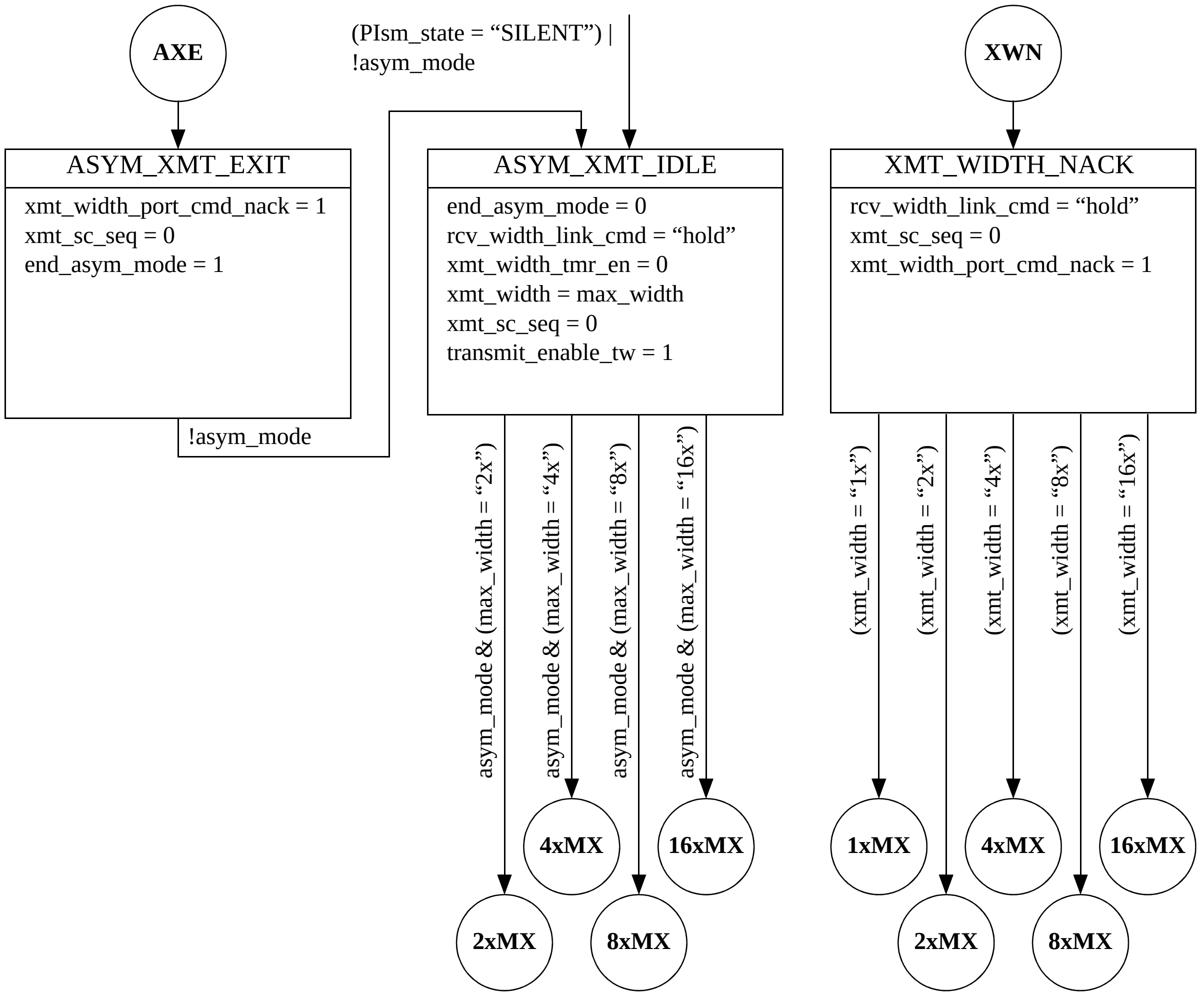


Figure 5-36. Transmit Width (XMT_Width) State Machine Part 1 of 4

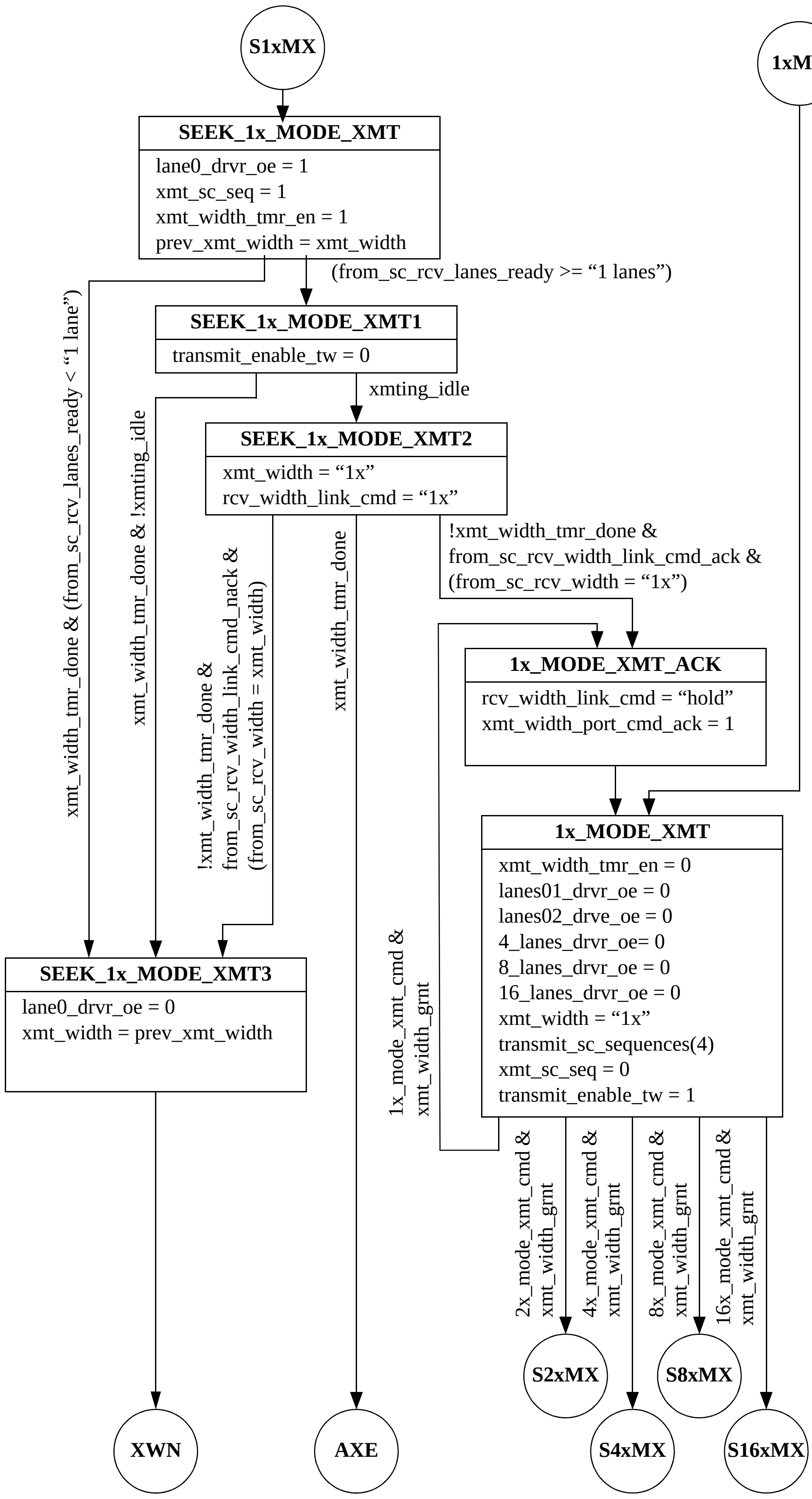

S1xMX
1xMX
SEEK_1x_MODE_XMT
lane0_drvr_oe = 1
xmt_sc_seq = 1
xmt_width_tmr_en = 1
prev_xmt_width = xmt_width
(from_sc_rcv_lanes_ready >= “1 lanes”)
xmt_width_tmr_done & (from_sc_rcv_lanes_ready < “1 lane”)
SEEK_1x_MODE_XMT1
transmit_enable_tw = 0
xmting_idle
xmt_width_tmr_done & !xmting_idle
SEEK_1x_MODE_XMT2
xmt_width = “1x”
rcv_width_link_cmd = “1x”
!xmt_width_tmr_done & from_sc_rcv_width_link_cmd_nack & (from_sc_rcv_width = xmt_width)
xmt_width_tmr_done
!xmt_width_tmr_done & from_sc_rcv_width_link_cmd_ack & (from_sc_rcv_width = “1x”)
1x_MODE_XMT_ACK
rcv_width_link_cmd = “hold”
xmt_width_port_cmd_ack = 1
1x_MODE_XMT
xmt_width_tmr_en = 0
lanes01_drvr_oe = 0
lanes02_drve_oe = 0
4_lanes_drvr_oe= 0
8_lanes_drvr_oe = 0
16_lanes_drvr_oe = 0
xmt_width = “1x”
transmit_sc_sequences(4)
xmt_sc_seq = 0
transmit_enable_tw = 1
1x_mode_xmt_cmd & xmt_width_grnt
SEEK_1x_MODE_XMT3
lane0_drvr_oe = 0
xmt_width = prev_xmt_width
2x_mode_xmt_cmd & xmt_width_grnt
4x_mode_xmt_cmd & xmt_width_grnt
8x_mode_xmt_cmd & xmt_width_grnt
16x_mode_xmt_cmd & xmt_width_grnt
S2xMX
S8xMX
XWN
AXE
S4xMX
S16xMX

Figure 5-37. Transmit Width (XMT_Width) State Machine Part 2 of 4

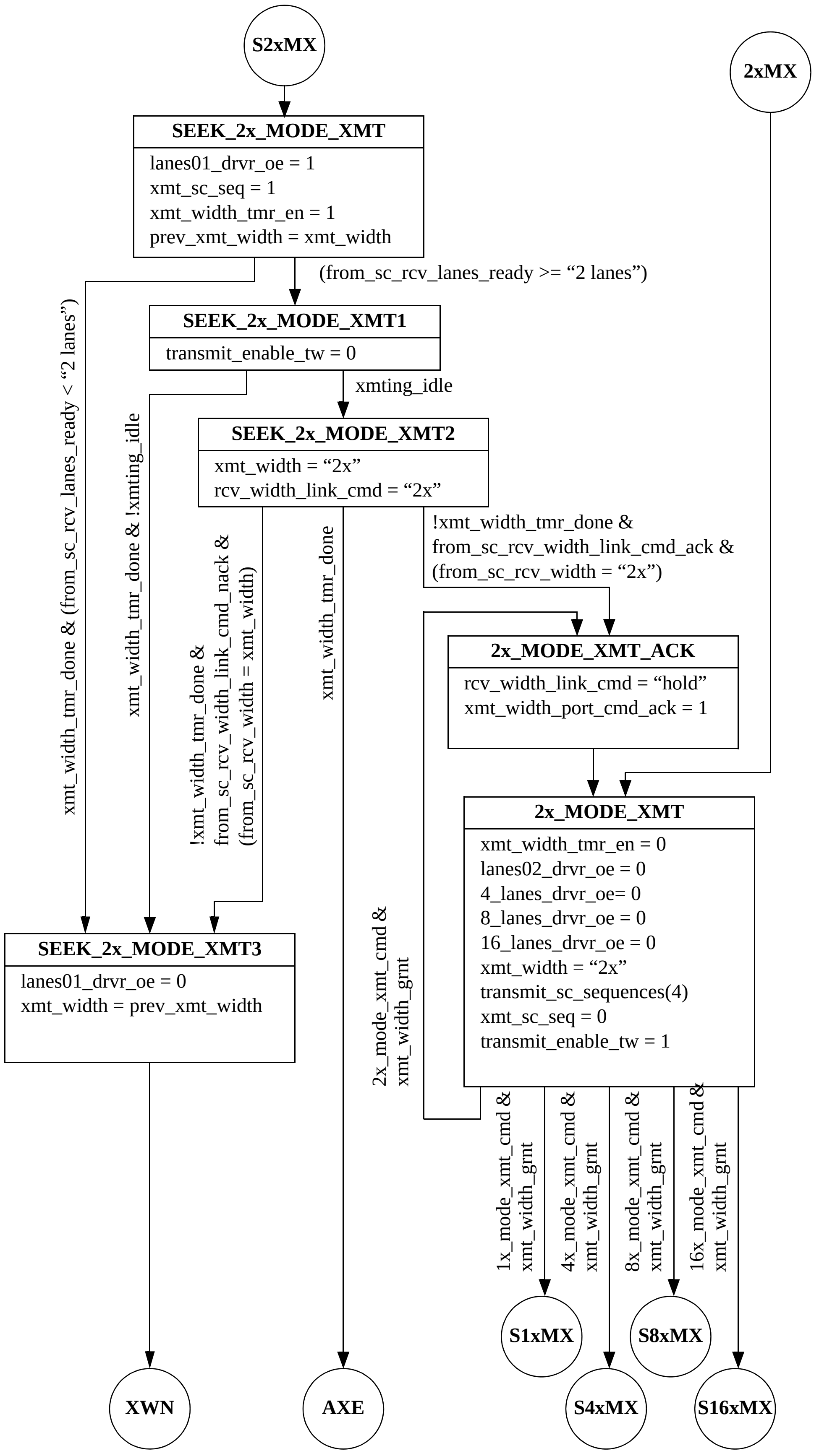


Figure 5-38. Transmit Width (XMT_Width) State Machine Part 3 of 4

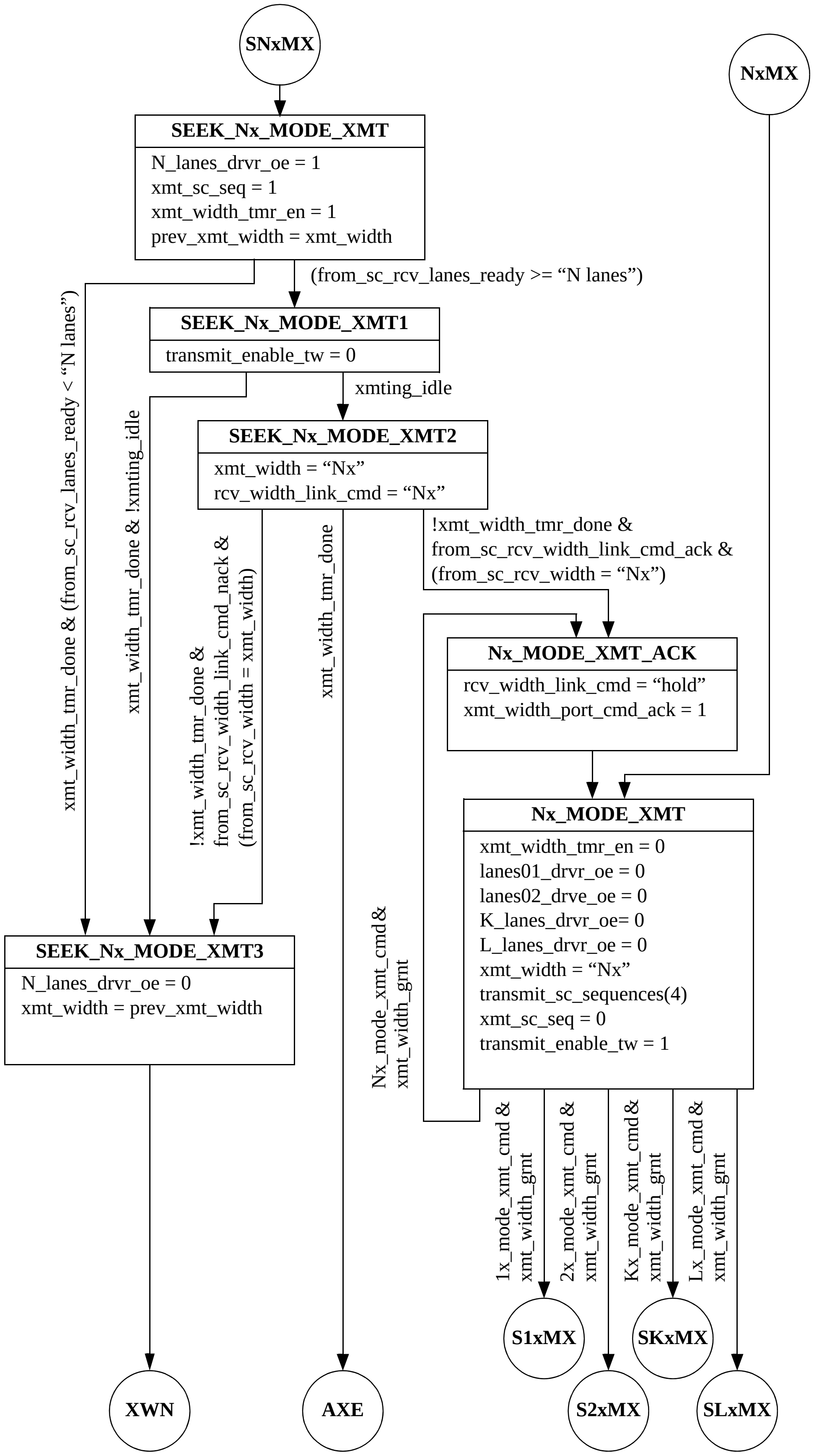


Figure 5-39. Transmit Width (XMT_Width) State Machine Part 4 of 4, K, L, N ≥ 4

5.19.10 Receive Width State Machines

Receive width link commands are received by a port in Status/Control ordered sequences transmitted by the connected port, and receive width command acknowledgements are returned in Status/Control ordered sequences. The handling of receive width link commands received by a port is shared between the Receive_Width and the Receive_Width_Cmd state machines. The Receive_Width state machine handles the receive width command if it is executable. The Receive_Width_Cmd state machine handles the command if it is not executable and handles the final stages of the command/acknowledgement protocol for all receive width link commands.

There shall be one Receive_Width_Cmd state machine and one Receive_Width state machine per port.

5.19.10.1 Receive_Width_Cmd State Machine

The Receive_Width_Cmd state machine checks each receive width link command received by a port for executability. A receive width link command is not executable if the port is not in asymmetric mode or the requested width is not enabled. If the command is not executable (bad_rcv_width_cmd asserted), the state machine negatively acknowledges (NACKS) the command. Regardless of whether a receive width command is executable, the state machine enforces the final stages of the receive width link command/acknowledgement protocol by keeping rcv_width_link_cmd_ack (ACK) or rcv_width_link_cmd_nack (NACK) asserted until the receiver width link command is de-asserted.

The Receive_Width_Cmd state machine is shown in Figure 5-40.

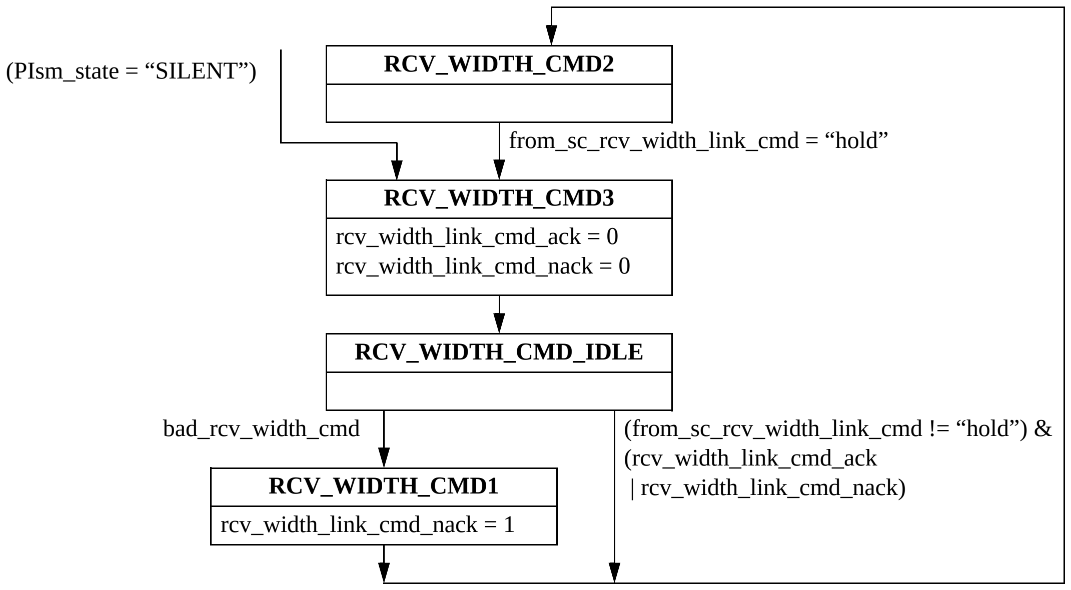

(PIsm_state = “SILENT”)
RCV_WIDTH_CMD2
from_sc_rcv_width_link_cmd = “hold”
RCV_WIDTH_CMD3
rcv_width_link_cmd_ack = 0
rcv_width_link_cmd_nack = 0
RCV_WIDTH_CMD_IDLE
bad_rcv_width_cmd
(from_sc_rcv_width_link_cmd != “hold”) & (rcv_width_link_cmd_ack | rcv_width_link_cmd_nack)
RCV_WIDTH_CMD1
rcv_width_link_cmd_nack = 1

Figure 5-40. Receive_Width_Cmd state machine

5.19.10.2 Receive_Width State Machine

Control of a port’s receive width is transferred from the Port_Initialization state machine to the Receive_Width state machine when the Port_Initialization state machine enters ASYM_MODE (asym_mode asserted).

When an executable receive width link command is received by a port, the Receive_Width state machine attempts to switch to the requested receive width. The state machine begins by starting the rcv_width_tmr (rcv_width_tmr_en asserted), stopping the reception of control symbols and packets (receive_enable de-asserted) and waits for N lanes to become ready and aligned (N_lanes_ready asserted), where N is the receive width requested in the receive width link command.

The receive width timer runs during the entire receive width change process. If any stage of the process fails to complete before the receive width timer times out, the receive width link command is NACKed and the state machine either restores the port to its current receive width or, if that does not appear to be possible, it forces the port to reinitialize (end_asym_mode is asserted). The receive width timer is used to prevent the Receive_Width state machine from becoming stuck part way though a receive width change operation.

When N lanes are ready and aligned, the state machine switches the port to the new receive width, ACKs the receive width link command, and re-enables control symbol and packet reception.

The Receive_Width state machine is specified in Figure 5-41 through Figure 5-44. The portion of the state machine specific to a given receive width shall be implemented only if that width mode is supported by the port. For example, if a port supports only 1x and 4x modes, only width specific portions of the Receive_Width state machine for 1x and 4x modes shall be implemented. The width specific portions for 2x, 8x, and 16x modes shall not be implemented.

Additional variables that are local to the Receive_Width state machine are as follows:

```
1x_mode_rcv_cmd = asym_mode & (from_sc_rcv_width_link_cmd = “1x mode”) &
                  !rcv_width_link_cmd_ack & !rcv_width_link_cmd_nack

2x_mode_rcv_cmd = asym_mode & (from_sc_rcv_width_link_cmd = “2x mode”) &
                  asym_2x_mode_enabled & !rcv_width_link_cmd_ack &
                  !rcv_width_link_cmd_nack

4x_mode_rcv_cmd = asym_mode & (from_sc_rcv_width_link_cmd = “4x mode”) &
                  asym_4x_mode_enabled & (max_width >= “4x”) &
                  !rcv_width_link_cmd_ack & !rcv_width_link_cmd_nack

8x_mode_rcv_cmd = asym_mode & (from_sc_rcv_width_link_cmd = “8x mode”) &
                  asym_8x_mode_enabled & (max_width >= “8x”) &
                  !rcv_width_link_cmd_ack & !rcv_width_link_cmd_nack
```

16x_mode_rcv_cmd = asym_mode & (from_sc_rcv_width_link_cmd = “16x mode”) &
asym_16x_mode_enabled & (max_width >= “16x”) &
!rcv_width_link_cmd_ack & !rcv_width_link_cmd_nack

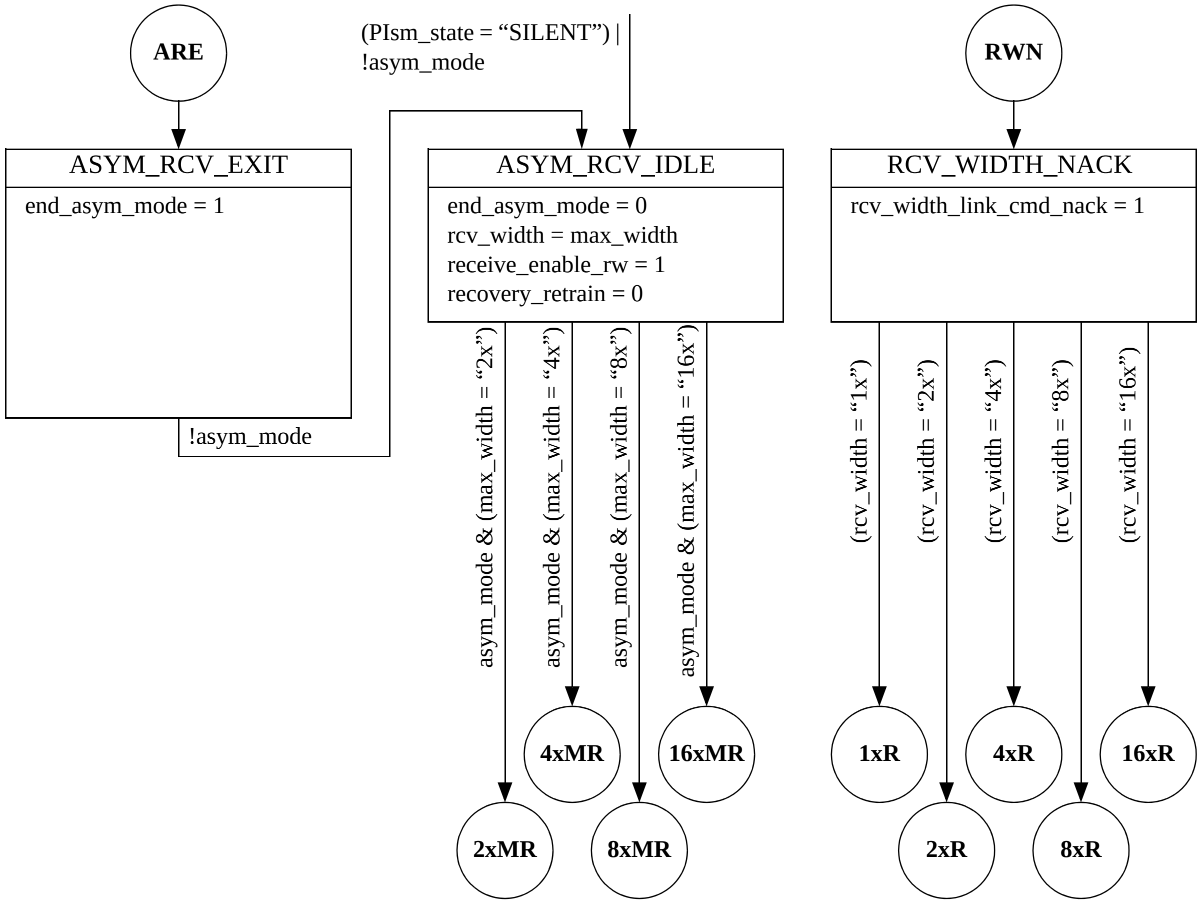


Figure 5-41. Receive_Width (RCV_Width) State Machine, Part 1 of 4

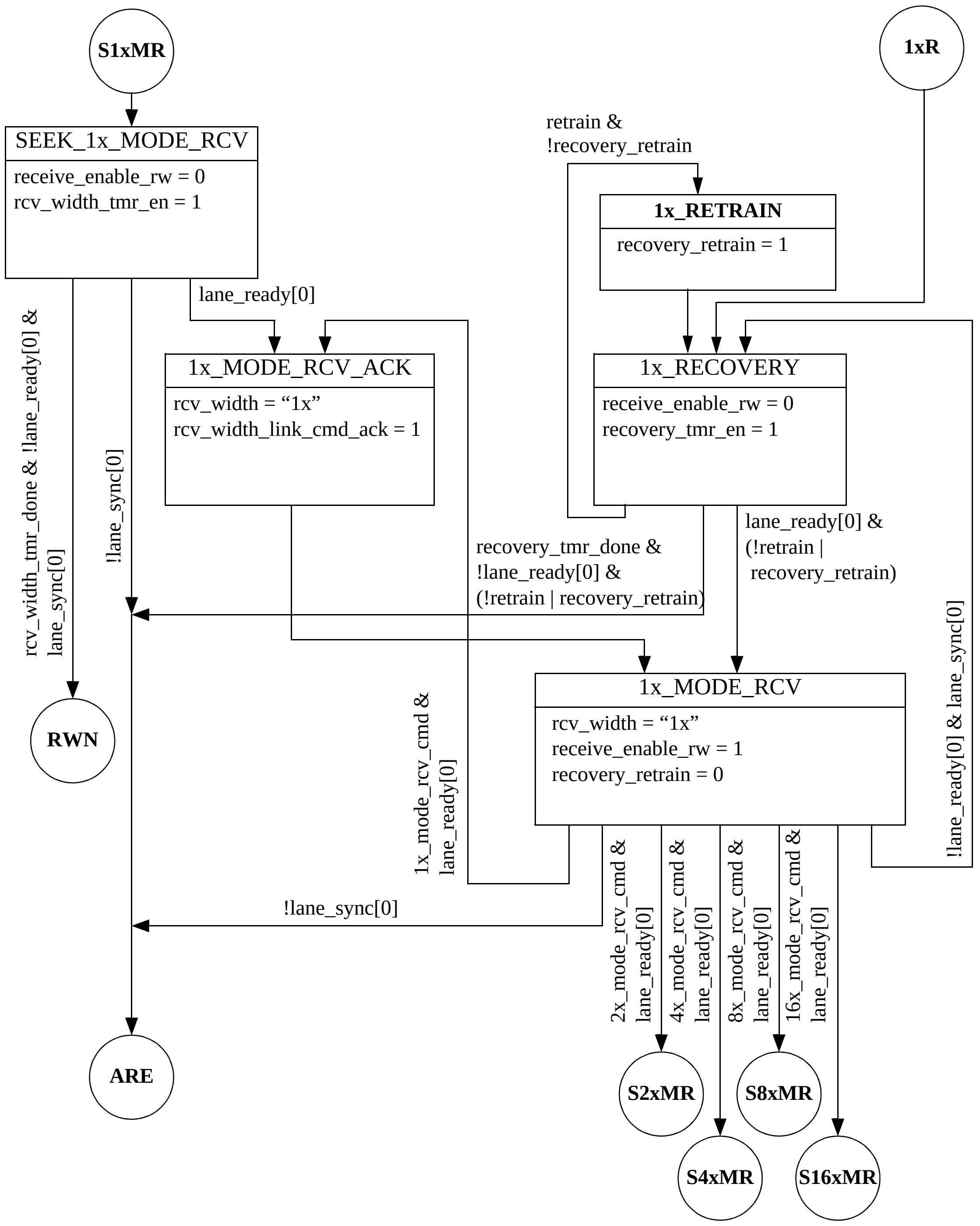


Figure 5-42. Receive_Width (RCV_Width) State Machine, Part 2 of 4

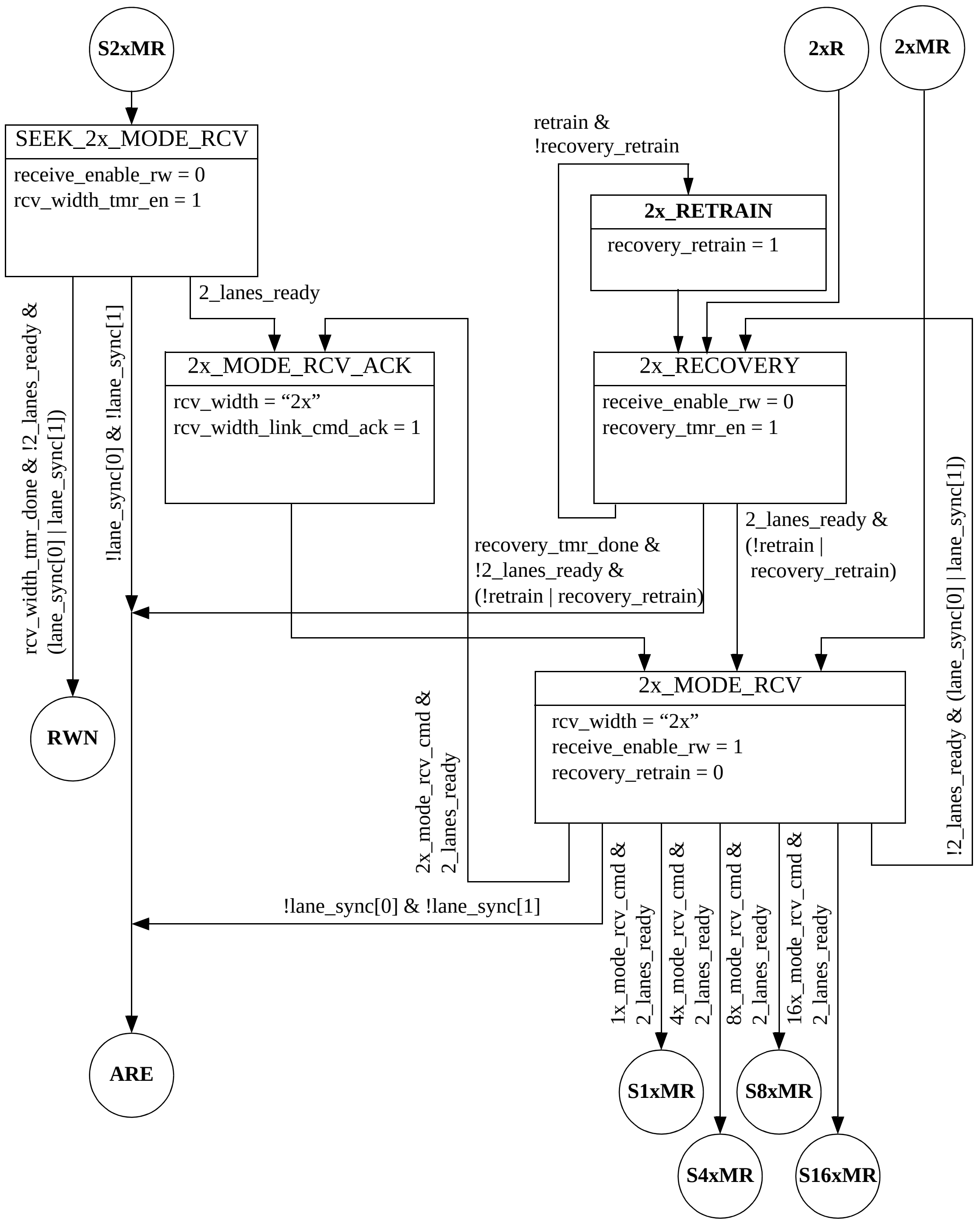

S2xMR
SEEK_2x_MODE_RCV
receive_enable_rw = 0
rcv_width_tmr_en = 1
2_lanes_ready
rcv_width_tmr_done & !2_lanes_ready & (lane_sync[0] | lane_sync[1])
!lane_sync[0] & !lane_sync[1]
2x_MODE_RCV_ACK
rcv_width = “2x”
rcv_width_link_cmd_ack = 1
2xR
2xMR
retrain & !recovery_retrain
2x_RETRAIN
recovery_retrain = 1
2x_RECOVERY
receive_enable_rw = 0
recovery_tmr_en = 1
recovery_tmr_done & !2_lanes_ready & (!retrain | recovery_retrain)
2_lanes_ready & (!retrain | recovery_retrain)
!2_lanes_ready & (lane_sync[0] | lane_sync[1])
2x_MODE_RCV
rcv_width = “2x”
receive_enable_rw = 1
recovery_retrain = 0
2x_mode_rcv_cmd & 2_lanes_ready
!lane_sync[0] & !lane_sync[1]
1x_mode_rcv_cmd & 2_lanes_ready
4x_mode_rcv_cmd & 2_lanes_ready
8x_mode_rcv_cmd & 2_lanes_ready
16x_mode_rcv_cmd & 2_lanes_ready
RWN
ARE
S1xMR
S8xMR
S4xMR
S16xMR

Figure 5-43. Receive_Width (RCV_Width) State Machine, Part 3 of 4

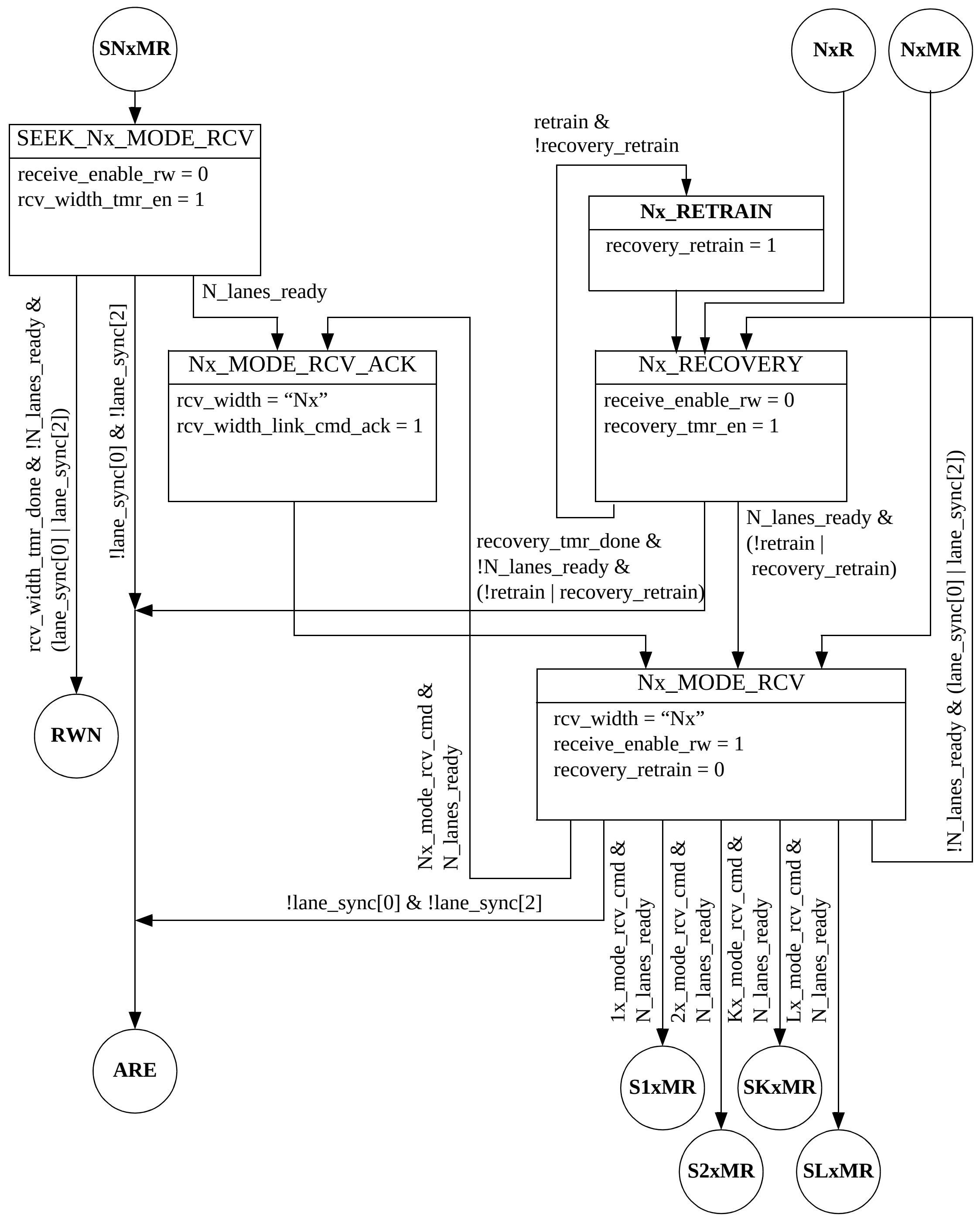


Figure 5-44. Receive_Width (RCV_Width) State Machine, Part 4 of 4, K, L, N ≥ 4

5.20 Pseudo Random Binary Sequence Testing

PRBS testing is supported for Baud Rate Class 3 operation, using the programming model described in section 4.14, "Pseudo Random Binary Sequence Testing".

While PRBS Active is set, all receive lanes for the port shall ensure that frame lock and codeword lock are deasserted. A design shall not assume that the data received from the link partner prevents frame lock and codeword lock assertion.

Chapter 6 LP-Serial Protocol

6.1 Introduction

This chapter specifies the LP-serial protocol which is the link level protocol for LP-serial links. The chapter covers traffic types, virtual channels (VCs), virtual channel queue management, packet priority, the mapping of transaction request flows onto packet priority, buffer management, and the use of control symbols in managing the delivery of packets between two devices connected by a LP-Serial link.

The protocol defines two types of traffic and provides a method for exchanging packets of each traffic type. The first type of traffic, called “reliable traffic” (RT), is the type of traffic RapidIO was originally designed to support. RT mode provides reliable delivery of packets between two devices that are connected by a RapidIO LP-Serial link. The second type of traffic, called “continuous traffic” (CT), provides unreliable delivery of packets that are “time sensitive”.

The protocol supports up to nine (9) virtual channels (VC0-VC8). Virtual Channel 0 (VC0) is always active and always operates in reliable traffic mode. It provides backward compatibility with Revision 1.3 RapidIO LP-Serial links. When only VC0 is active, a link is said to be operating in single VC mode. VCs 1-8 are optional, and if implemented, may be disabled for backward compatibility with Rev. 1.3 LP-Serial links.

6.2 Packet Exchange Protocol

As originally designed, the LP-Serial specification defines a protocol for devices connected by a LP-Serial link in which each packet transmitted by one device is acknowledged by control symbols transmitted by the other device. If a packet cannot be accepted for any reason, an acknowledgment control symbol indicates the reason and that the original packet and any transmitted subsequent packets must be resent. This behavior provides a flow control and error control mechanism between connected processing elements. This is the protocol for reliable traffic (RT).

Figure 6-1 shows an example of transporting a RT request and response packet pair across an interconnect fabric with acknowledgments between the link transmitter/receiver pairs along the way. This allows flow control and error handling to be managed between each electrically connected device pair rather than between the original source and final target of the packet. An end point device shall transmit an acknowledgment control symbol for a request packet before transmitting the response packet corresponding to that request.

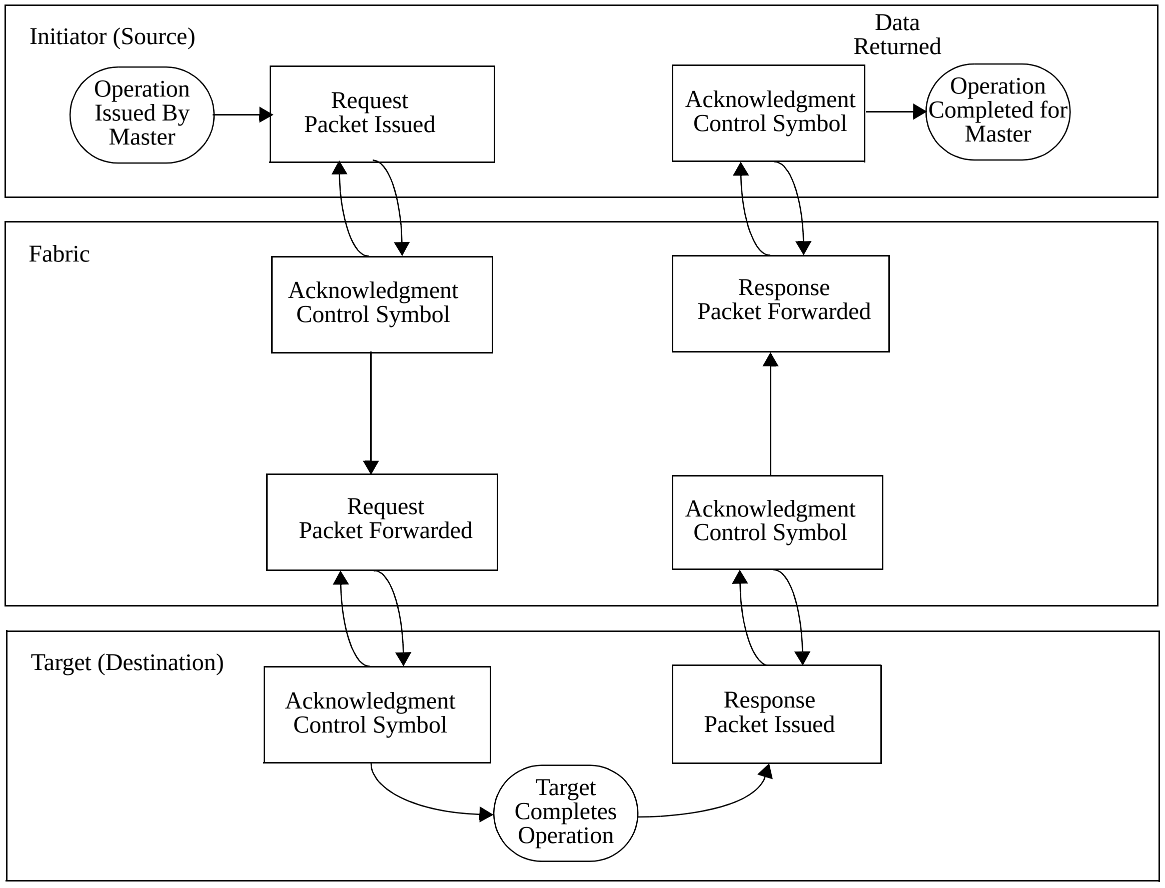

Initiator (Source)
Data Returned
Operation Issued By Master
Request Packet Issued
Acknowledgment Control Symbol
Operation Completed for Master
Fabric
Acknowledgment Control Symbol
Response Packet Forwarded
Request Packet Forwarded
Acknowledgment Control Symbol
Target (Destination)
Acknowledgment Control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

Figure 6-1. Example Transaction with Acknowledgment

The protocol for continuous traffic (CT) is very similar to the protocol for reliable traffic (RT). The primary differences are that some CT packets are not acknowledged and therefore CT packets are subject to loss due to errors or insufficient buffer resources at the receiver.

6.3 Traffic types

The LP-Serial protocol provides support for transporting two types of traffic, “reliable traffic” (RT) and “continuous traffic” (CT). Reliable Traffic is guaranteed to be lossless by using packet retransmission whenever packet corruption is detected or receiver buffer resources are overrun. Continuous Traffic is subject to packet loss when packet corruption is detected or receiver buffer resources are overrun, but does not incur any additional latency, by continuing its packet flow without retransmission of unacknowledged packets.

6.4 Virtual Channels

Virtual channels provides a mechanism that allows the bandwidth of a link to be allocated amongst various unrelated “streams” and types of traffic in a manner that ensures that each stream, or group of streams, receives a guaranteed minimum fraction of the link bandwidth. This is done by allocating one or more streams of a given traffic type to each VC and then allocating each VC a specified fraction of the link bandwidth.

The LP-Serial protocol supports up to nine (9) virtual channels (VC0-VC8).

6.4.1 Virtual channel 0 (VC0)

VC0 shall be supported by all LP-Serial ports. VC0 shall always be active, operate in RT mode and support the packet priority rules as described in Section 6.6.3, as well as supporting the packet delivery ordering rules described in Section 6.11. VC0 provides the packet transport service specified in Rev. 1.3 of this specification and is backward compatible with Rev. 1.3.

6.4.2 Virtual Channels 1-8 (VC1-8)

Support for VC1 through VC8 by LP-Serial ports is optional. Any of VC1 through VC8 that are implemented shall support operation in RT mode and may optionally support and be configured for operation in CT mode. CT VCs operate independent of each other. RT VCs operate as a “RT Group”. That is to say, when the error recovery protocol is used to recover a damaged packet, the unacknowledged packets for all VCs in RT mode are retransmitted. See Section 6.13, "Error Detection and Recovery" for more on the error recovery process of RT and CT VCs.

The number of VCs implemented is up to the implementer. VC0 is always implemented. The number of channels for VCs 1-8 may be 0, 1, 2, 4, or 8. The binary configuration allows traffic to be combined (by ignoring bits in the VC field) in a predictable manner. Implementations with fewer than the full number of VCs should ignore, but must not modify, any ignored VC bits. That way traffic can fan back out into a larger set of VCs on subsequent links. The hierarchy for combining VCs is as follows:

Table 6-1. Additional VC Combinations

8 VCs	4 VCs	2 VCs	1 VC
VC1	VC1	VC1	VC1
VC2			
VC3	VC3		
VC4			
VC5	VC5	VC5	
VC6			
VC7	VC7		
VC8			

In systems implementing one or more of VCs 1 through 8 and in which the number of VCs 1 through 8 that are implemented varies from one LP-Serial link to another, care needs to exercised in assignment of VC numbers so that the desired RT or CT characteristic of a virtual channel is maintained as the channel passes from one link to another link that implements fewer virtual channels.

6.4.3 Virtual Channel Utilization

Packets are transmitted from one or more virtual channels according to the weighted distribution of bandwidth for each channel. The weighting is such that under demand for full utilization of the link’s bandwidth, each active VC is guaranteed a certain portion of that bandwidth. This is the minimum that each VC can achieve. When the demand for bandwidth is less than the allocation for any VC, the extra bandwidth may be distributed among the other VCs giving them more than their allotment. The algorithm for scheduling traffic is up to the implementer as long as the rules (see Section 6.11, "Transaction and Packet Delivery Ordering Rules") are met.

Processing elements shall not assume any packet ordering guarantees between VCs. Packets within a VC in VCs 1 - 8 are equally weighted and must be kept in order. Only packets within VC0 have additional ordering rules based on priority.

6.5 Control Symbols

Control Symbols are the message elements used by ports connected by a LP-Serial link to manage all aspects of LP-Serial link operation. They are used for link maintenance, packet delimiting, packet acknowledgment, error reporting, and error recovery.

6.5.1 Control Symbol Selection

For a LP-Serial link running at Baud Rate Class 1 the control symbol type used on the link is determined by the idle sequence being used on the link. Idle sequence selection occurs during the port initialization process. If the link is operating with IDLE1, the Control Symbol 24 shall be used. If the link is operating with IDLE2, the Control Symbol 48 shall be used.

A LP-Serial link running at Baud Rate Class 2 shall support Control Symbol 48 and IDLE2.

A LP-Serial link running at Baud Rate Class 3 shall support Control Symbol 64 and IDLE3.

6.5.2 Control Symbol Delimiting

LP-Serial control symbols (Control Symbol 24 and Control Symbol 48) on 8b/10b encoded links are delimited for transmission by 8b/10b special characters. For 64b/67b encoded link no such delimiting exists for Control Symbol 64.

Control Symbol 24 are delimited by a single 8b/10b special character that marks the beginning of the control symbol and immediately precedes the first character of the control symbol. Since control symbol length is constant and known, an end delimiters is neither needed nor provided.

Control Symbol 48 are delimited by two 8b/10b special characters. The first special character marks the beginning of the control symbol (the start delimiter) and immediately precedes the first character of the control symbol. The second special character marks the end of the control symbol (the end delimiter) and immediately follows the last character of the control symbol. The end delimiter special character replicates the value of the start delimiter special character. The end delimiter is provided for error detection in a burst error environment.

One of two special characters is used to delimit a control symbol. If the control symbol contains a packet delimiter, the special character PD (K28.3) is used. If the control symbol does not contain a packet delimiter, the special character SC (K28.0) is used. This use of different special characters provides the receiver with an “early warning” of the content of the control symbol.

The control symbol delimiting special character(s) shall be added to the control symbol before the control symbol is passed to the PCS sublayer for 8b/10b encoding and, if applicable, lane striping.

The combination of a control symbol and its delimiting special character(s) is referred to as a “delimited control symbol”.

6.5.3 Control Symbol Use

6.5.3.1 Link Initialization

An LP-Serial port needs be initialized and the link to which it is connected also needs be initialized before the port may begin the normal transmission of packets and control symbols. The port is initialized when the port's Initialization state machine variable port_initialized is asserted. The link is initialized after the port has successfully completed the following link initialization process and entered the link_initialized state (link_initialized variable asserted).

When a port is in the port_initialized state, but not in the link_initialized state, and for ports operating with IDLE3 transmit_enable is asserted, the port shall transmit only idle sequences, status, VC-status, link-request and link-response control symbols and, if IDLE2 is the idle sequence in use on the link, SYNC sequences.

After a LP-Serial port is initialized, the port shall complete the following sequence of actions to enter the link_initialized state (normal operational state).

1. The initialized port shall transmit idle sequence and at least one status control symbol per 1024 code-groups or codewords transmitted per lane until the port has received an error free status control symbol from the connected port. The transmission of status control symbols indicates to the connected port that the port has completed initialization. The transmission of an idle sequence is required for the connected port to complete initialization.
2. After the initialized port has received an error free status control symbol from the connected port, the port shall transmit idle sequence and at least 15 additional status control symbols. This group of control symbols may be sent more rapidly than the minimum rate of one status control symbol every 1024 code-groups or codewords transmitted per lane.
3. After the initialized port has received an error free status control symbol, the port shall wait until it has received a total of seven error free status control symbols with no intervening errors. This requirement provides a degree of link verification before packets and other control symbols are transmitted.
4. If any VC other than VC0 is implemented and enabled, the port shall transmit a single VC_Status control symbol for each such VC. This initializes the flow control status for each implemented and enabled VC other than VC0.
5. The port enters the link_initialized state.

Once a port is in the link_initialized state, loss of port initialization (port_initialized variable deasserted) shall cause the port to exit the link_initialized state (link_initialized variable deasserted). The link is then uninitialized from the point of view of that port. Once the port has exited the link_initialized state, the port shall not resume the normal transmission of packets and control symbols until the port has re-entered both the port_initialized and link_initialized states.

A port that is not in the port_initialized state, or a port operating with IDLE3 in the port_initialized state when receive_enable is deasserted, shall ignore and discard any packet or control symbol that it receives from the connected port. A port that is in the port_initialized state but not in the link_initialized state shall ignore and discard any packet or any control symbol, other than status, VC-status, link-request or link-response control symbols, that it receives from the connected port.

A LP-Serial port shall not enter the Input error-stopped state or the Output error-stopped state unless the port is in the link_initialized state and, for ports operating with IDLE3, receive_enable is asserted. The loss of link initialization (the state machine link_initialized variable is deasserted) shall not cause a port already in the Input error-stopped state or the Output error-stopped state to exit either of those states.

6.5.3.2 Buffer Status Maintenance

When a LP-Serial port is in the normal operational state, it shall transmit a control symbol containing the buf_status field for VC0 at least once every 1024 code-groups or codewords transmitted per lane. To comply with this requirement, the port shall transmit a status control symbol if no other control symbol containing the buf_status field for VC0 is available for transmission.

NOTE:Note: Status Control Symbol Transmission Rates

If device X compliant to revision 2.1 or later is connected to a device Y which is compliant to a specification revision earlier than 2.1, the rate at which device X and device Y receive buffer status information will be different. This difference does not create any interoperability issues. The rate of buf_status updates shall not be checked.

When a LP-Serial port is in the normal operational state and any VC other than VC0 is active (VCs 1-8), the port shall transmit a control symbol containing the buf_status field for each active VC at least once every VC refresh period. To comply with this requirement, the port shall transmit a VC_status control symbol for each active VC, other than VC0, if no other control symbol containing the buf_status field for that VC is available for transmission during the VC refresh interval. VC_status may be transmitted at any time, triggered by changes in VC conditions according to implementation specific algorithms.

The VC refresh period can be configured through the VC Refresh Interval register field defined in Chapter 7, "LP-Serial Registers". The shortest VC refresh period is 1024 code-groups or codewords, and the longest VC refresh period required to be implemented is 1024 x 16 = 16K code-groups or codewords.

NOTE:VC Refresh Period

The VC Refresh Interval register field contains space for up to 8 bits to be used, so based on implementation, the maximum refresh period may be 256K code-groups or codewords (see Section 7.8.2.2).

6.5.3.3 Embedded Control Symbols

Any control symbol that does not contain a packet delimiter may be embedded in a packet. An embedded control symbol may contain any defined encoding of stype0 and a stype1 encoding of “Timing” or “NOP”. Control symbols with stype1 encodings of start-of-packet, end-of-packet, stomp, restart-from-retry, or link-request cannot be embedded as they would terminate the packet.

When a Control Symbol 24 or Control Symbol 48 is embedded in a packet, the delimited control symbol shall begin on a 4-character boundary of the packet. That is, the number of packet characters between the end of the delimited start-of-packet control symbol and the start of the embedded delimited control symbol shall be a non-negative integer multiple of 4.

When a Control Symbol 64 is embedded in a packet, the control symbol shall begin on a 8-byte boundary of the packet. That is, the number of packet bytes between the end of the start-of-packet control symbol and the start of the embedded control symbol shall be a non-negative integer multiple of 8. This requirement is automatically fulfilled by the codeword encoding defined in Section 5.5.

The manner and degree to which control symbol embedding is used on a link impacts both link and system performance. For example, embedding multicast-event control symbols allows their propagation delay and delay variation through switch processing elements to be minimized and is highly desirable for some multicast-event applications. Embedding packet acknowledgment control symbols reduces the delay in freeing packet buffers in the transmitting port which can increase packet throughput and reduce packet propagation delay in some situations, which can be desirable. On the other hand, embedding all packet acknowledgment control symbols rather than combining as many of them as possible with packet delimiter control symbols reduces the link bandwidth available for packet transmission and may be undesirable.

6.5.3.4 Timing Control Symbols

Timing control symbols are related to communication of events and time within a system. Unlike other control symbols, timing control symbols can trigger activity on other links of a device.

6.5.3.4.1 Multicast-Event Control Symbols

Multicast-Event Control Symbols and Secondary Multicast Event Control Symbols provide a mechanism for notifying end points that system defined events have occurred. These events can be selectively multicast through the system. For the format of the multicast-event control symbols, see Section 3.5.6. Multicast-Event Control Symbols and Secondary Multicast Event Control Symbols are generically referred to as (S)MECS.

When a switch processing element receives an MECS, the switch shall forward the MECS by issuing an identical MECS on each port that is designated by the port's Port n Control CSRs “Multicast-event Participant” field as a Multicast-Event output port. When a switch processing element receives an SMECS, the switch shall forward the SMECS by issuing an identical SMECS on each port that is designated by the port’s Port n SMECS Control CSR “Secondary Multicast-event Participant” field as a Secondary Multicast-Event output port. A switch port shall never forward an (S)MECS back to the device from which it received the (S)MECS, regardless of whether the port is designated a (Secondary) Multicast-Event Participant output port or not.

It is intended that at any given time, MECS will be sourced by a single device and SMECS will be sourced by a different device; however, the source device of each can change (in case of failover, for example). In the event that two or more (S)MECS of the same type 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 (S)MECS shall be forwarded. The others may be forwarded or discarded (device dependent). Multicast-Event Control Symbols and Secondary Multicast-Event Control Symbols shall not be combined with each other.

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). The transmission delay of Multicast-Event control symbols can be increased dramatically by Skip ordered sequences, asymmetric transmit width change and retraining. The latency impact for asymmetric transmit width change can range in the 10’s to 100’s of usec, and retraining can range in the 10’s of msec.

6.5.3.4.2 Loop-Timing Request

Support for transmission and reception of the Loop-Timing Request control symbol is optional. A processing element shall be capable of transmitting a Loop-Timing Request control symbol if the Timestamp Master Support bit of the Timestamp CAR is 1. A processing element shall be capable of receiving a Loop-Timing Request control symbol if the Timestamp Slave Support bit is 1 in the Timestamp CAR.

When a processing element transmits a Loop-Timing Request control symbol, the value of its Timestamp Generator shall be latched in the Port n Timestamp 0 MSW CSR and Port n Timestamp 0 LSW CSR. When a processing element receives a Loop-Timing Request control symbol, the processing element shall transmit a loop-response control symbol.

6.5.3.5 Time Synchronization Protocol

Support for time synchronization is optional. Time synchronization is the method of synchronizing the "sense of time" between RapidIO processing elements. The "sense of time" is embodied in a Time Stamp Generator (TSG) for each node.

A TSG consists of a single 64-bit nanosecond granularity counter. The timestamp generator is divided between two 32-bit registers:

- The Timestamp Generator LSW CSR register contains the least significant 32 bits of the TSG.
- The Timestamp Generator MSW CSR register contains the most significant 32 bits of the TSG.

A timestamp is a 64-bit value, consisting of the MSW register in the most significant bits and the LSW register in the least significant bits.

The TSG counter increments regularly using a multiple of the period of the clock that drives the TSG counter. The reference clock period chosen is implementation specific. For example, assume that the TSG counter is driven by a 312.5 MHz clock with a period of 3.2 nanoseconds. The TSG counter could increment by 16, every 16 nanoseconds. It is also valid for the counter to increment in the following pattern over a period of 16 nanoseconds: 3, 3, 3, 3, 4.

Synchronization of TSGs is supported with varying degrees of accuracy. For example:

- Synchronization of TSGs with microseconds of accuracy is required. TSGs are synchronized between link partners using specific control symbols. TSGs advance at the same frequency, +/- 100 PPM.
- Synchronization of TSGs within less than a microsecond is required. TSGs are synchronized between link partners using specific control symbols. The difference in frequency between link partners is calibrated and compensated for. Timestamp Generator master devices regularly update Timestamp Generator Slave devices.
- Synchronization of TSGs within 100 nanoseconds or less is required. All TSGs use the same clock frequency to control the rate at which time advances. The delay between link partners is calibrated using control symbols to ensure maximum accuracy. TSGs are synchronized between link partners regularly, adjusting for the delay between link partners.

The following sections discuss mechanisms that implement the above synchronization. These mechanisms may use either maintenance reads/writes or control symbols.

Table 6-2 summarizes the control symbol support required to implement the timestamp synchronization protocol based on the values of the Timestamp CAR fields. Note that for devices that can be both a TSG Master and Slave, Slave control symbol support is required in TSG Slave mode, and Master control symbol support is required when in TSG Master mode, where the TSG mode is determined by the Port Operating Mode field of the Port n Timestamp Generator Synchronization CSR.

Table 6-2. Control Symbol Support for TSG Master and Slave Devices

Control Symbol	Timestamp Master/ Slave Supported Both = 0	Timestamp Slave Supported = 1	Timestamp Master Supported = 1
Loop-Timing Request	None	Receive	Transmit
Loop-Response	None	Transmit	Receive
Timestamp Sequence	None	Receive	Transmit

6.5.3.5.1 Setting and Reading a Timestamp Generator

To set the TSG registers using Maintenance Writes, first write the TSG LSW register and then write the TSG MSW register. Software may elect to delay updating the TSG when the TSG LSW register is close to rolling over to avoid incrementing the TSG MSW register.

To read the TSG registers using Maintenance Reads, first read the TSG MSW register, then the TSG LSW register, and then the TSG MSW register again. If the value of the TSG MSW register has not changed, then the timestamp value has been read successfully. If the value of the MSW register has changed, the TSG LSW register shall be read again to compose an accurate timestamp.

Devices can support 8-byte register reads and writes, which allow the MSW and LSW registers to be read and updated simultaneously.

The TSG of a link partner can also be set using sequences of Timestamp control symbols. Support for transmission and reception of a sequence of timestamp control symbols is optional. A processing element shall support transmitting a sequence of timestamp control symbols when the Timestamp Master Supported bit of the Timestamp CAR is 1. A processing element shall support receiving a sequence of timestamp control symbols when the Timestamp Slave Supported bit of the Timestamp CAR is 1.

When links are operating with Control Symbol 24, a sequence of 8 Control Symbol 24 timestamp control symbols is sent to set the link partner’s timestamp generator value. Each Control Symbol 24 timestamp control symbol in the sequence contains two flags and eight bits of the 64-bit timestamp generator value, as shown in Table 6-3.

Table 6-3. Sequence and Format of Control Symbol 24 Timestamp Control Symbols

Control Symbol Sequence	Parameter 0 Bit 0 "Start Flag"	Parameter 0 Bit 1 "End Flag"	Parameter 0 Bits 2-4	Parameter 1 Bits 0-4
0	1	0	Timestamp [0:2]	Timestamp[3:7]
1	0	0	Timestamp [8:10]	Timestamp[11:15]
2	0	0	Timestamp [16:18]	Timestamp[19:23]
3	0	0	Timestamp [24:26]	Timestamp[27:31]
4	0	0	Timestamp [32:34]	Timestamp[35:39]
5	0	0	Timestamp [40:42]	Timestamp[43:47]
6	0	0	Timestamp [48:50]	Timestamp[51:55]
7	0	1	Timestamp [56:58]	Timestamp[59:63]

When links are operating with Control Symbol 48, a sequence of 8 Control Symbol 48 timestamp control symbols is sent to set the link partner's timestamp generator value. The format and contents of each Control Symbol 48 timestamp control symbol in the timestamp sequence is defined in Table 6-4.

Table 6-4. Sequence and Format of Control Symbol 48 Timestamp Control Symbols

Control Symbol Sequence	Parameter 0 Bit 0	Parameter 0 Bit 1 "Start Flag"	Parameter 0 Bit 2 "End Flag"	Parameter 0 Bits 3-5	Parameter 1 Bit 0	Parameter 1 Bits 1-5
0	0b0	1	0	Timestamp [0:2]	0b0	Timestamp[3:7]
1	0b0	0	0	Timestamp [8:10]	0b0	Timestamp[11:15]
2	0b0	0	0	Timestamp [16:18]	0b0	Timestamp[19:23]
3	0b0	0	0	Timestamp [24:26]	0b0	Timestamp[27:31]
4	0b0	0	0	Timestamp [32:34]	0b0	Timestamp[35:39]
5	0b0	0	0	Timestamp [40:42]	0b0	Timestamp[43:47]
6	0b0	0	0	Timestamp [48:50]	0b0	Timestamp[51:55]
7	0b0	0	1	Timestamp [56:58]	0b0	Timestamp[59:63]

When links are operating with Control Symbol 64, a sequence of 4 Control Symbol 64 timestamp control symbols is sent to set the link partner's timestamp generator value. A sequence number is used to ensure the integrity of the Timestamp control symbol sequence. The format and contents of each Control Symbol 64 timestamp control symbol in the timestamp sequence is defined in Table 6-5.

Table 6-5. Sequence and Format of Control Symbol 64 Timestamp Control Symbols

Control Symbol Sequence	Parameter 0 Bits 0-2	Parameter 0 Bits 3-4	Parameter 0 Bits 5-7	Parameter 0 Bits 8-11	Parameter 1 Bits 0-11
0	Reserved	0b00	Reserved	Timestamp [0:3]	Timestamp[4:15]
1	Reserved	0b01	Reserved	Timestamp [16:19]	Timestamp[20-31]
2	Reserved	0b10	Reserved	Timestamp [32:35]	Timestamp[36:47]
3	Reserved	0b11	Reserved	Timestamp [48:51]	Timestamp[52:63]

The timestamp value in the sequence of Timestamp control symbols shall be sent as if all 64 bits were captured when the first Timestamp control symbol was formulated. The timestamp value sent may have a nanoseconds offset added to it before transmission to account for transmission delay. The offset is a programmable value found in the Port n Timestamp Offset CSRs.

A sequence of Timestamp control symbols shall not be interrupted by any other control symbols or an IDLE sequence.

If all the Timestamp control symbols in a sequence are not received correctly, without interruption, the receiver’s TSG shall not be updated.

The receiver can adjust the timestamp value, if necessary, to reflect transmission delay due to control symbol alignment and/or the time required to receive the full sequence of Timestamp control symbols.

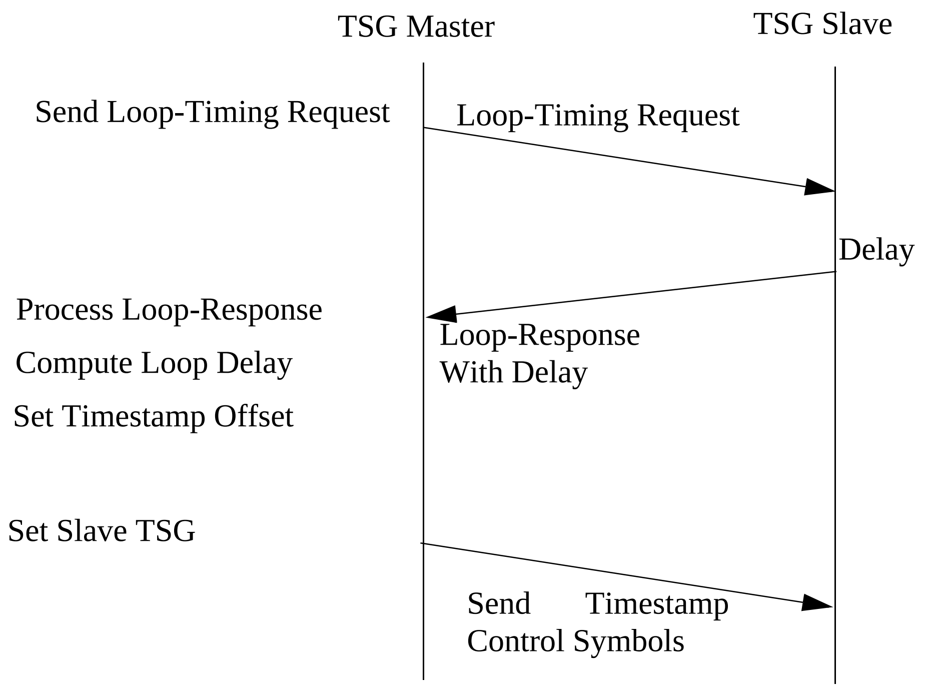
A timestamp generator shall immediately change its value to 0 when programmed to do so. A timestamp generator shall immediately change its value when programmed to a value larger than the current TSG value.

When either control symbols or maintenance packets are used to change a TSG to a value that is less than the current TSG value, the TSG value shall be held constant for the difference in time between the current TSG value and the time that was programmed. This has the effect of halting time until the new time value is reached. Implementations shall allow the current time value to be held constant for a period of 65535 nanoseconds. Setting a timestamp value more than 65535 nanoseconds in the past results in implementation specific behavior.

6.5.3.5.2 Calibrating Transmission Delay

The procedure for calibrating the transmission delay between the Master and Slave is shown in the Figure 6-2 message sequence chart.

Figure 6-2. Time Synchronization with Synchronous Link Partners



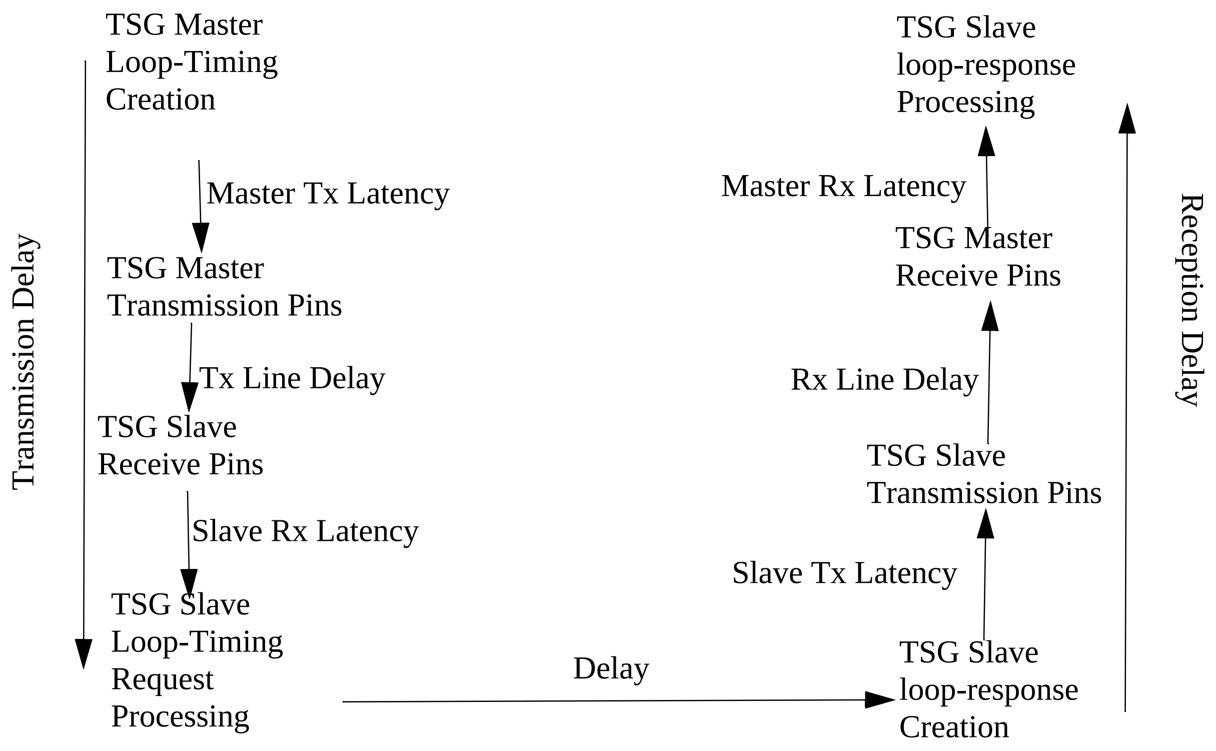
The steps are defined as follows:

1. Send Loop-Timing Request: The TSG Master sends a Loop-Timing Request control symbol to the TSG Slave by writing 0x00000003 to the Port n Timestamp Synchronization Command CSR. This latches the current TSG Master TSG value in the Port n Timestamp 0 MSW CSR and Port n Timestamp 0 LSW CSR.
2. Process Loop-Response: The TSG Master receives a Loop-response, which contains the Delay amount in the TSG Slave, for the Loop-Timing Request control symbol. This also causes the current TSG Master TSG value to be latched in the Port n Timestamp 1 MSW CSR and Port n Timestamp 1 LSW CSR.
3. Compute Loop Delay: The TSG Master computes the loop timing delay as described at the end of this section.
4. Set Timestamp Offset: The TSG Master programs its Port n Timestamp Offset register value to the Loop Delay computed in Step 3.
5. Set Slave TSG: The TSG Master sets the TSG Slaves Timestamp Generator value by writing 0x00000010 to the Port n Timestamp Synchronization Command CSR.

A loop-response for a loop-timing request must be received within the link response timeout period. If the loop-response is not received within the timeout period, then the loop-timing request shall be treated as completed. A loop-timing request shall not be retransmitted in the event of a timeout.

Computing loop delay is complicated by the need to account for transmit and receive asymmetries in the transmitter and receiver of the TSG Slave and TSG Master. These asymmetries are displayed in Figure 6-3.

Figure 6-3. Asymmetry Computation



TSG Master Loop-Timing Creation
Master Tx Latency
TSG Master Transmission Pins
Tx Line Delay
TSG Slave Receive Pins
Slave Rx Latency
TSG Slave Loop-Timing Request Processing
Transmission Delay
Delay
TSG Slave loop-response Processing
Master Rx Latency
TSG Master Receive Pins
Rx Line Delay
TSG Slave Transmission Pins
Slave Tx Latency
TSG Slave loop-response Creation
Reception Delay

The following computation uses the notation “(Condition)?Val1:Val2” to describe a function that returns “Val1” if “Condition” is true, and “Val2” if “Condition” is false.

The term “Master Tx Asymmetry” below is computed using fields in the TSG Master’s Port n Timestamp Generator Synchronization CSRs fields.

The term “Slave Rx Asymmetry” below is computed using fields in the TSG Slave’s Port n Timestamp Generator Synchronization CSRs fields.

Total Delay = Timestamp 1 - Timestamp 0 - Delay

Master Tx Asymmetry = (Asymmetry / 2) * ((Tx Has Lower Latency = 1)?-1:1)

Slave Rx Asymmetry = (Asymmetry / 2) * ((Tx Has Lower Latency = 1)?1:-1);

Transmission Delay = (Total Delay / 2) + Master Tx Asymmetry + Slave Rx Asymmetry

Note that it is not possible for the TSG Master to measure the actual transmission delay. The transmission delay computed is still inaccurate by half the difference between Tx Line Delay and Rx Line Delay. Solutions that require highly accurate TSG synchronization can minimize asymmetry between Tx Line Delay and Rx Line Delay through physical design constraints.

When links are operating with Control Symbol 24 or Control Symbol 48, a loop-response shall consist of a single Timestamp control symbol transmitted in response to a loop-timing request (Section 3.5.6.3). For Control Symbol 24 and Control Symbol 48 formats, the Timestamp carries a single Delay value that represents the number of nanoseconds between the time the loop-timing request was received by the link partner, and the time the loop-response was generated. The Delay field is 12 bits for Control Symbol 48 and Control Symbol 64, but for Control Symbol 24 a 10 bit delay field is sufficient to address the delay value. A Delay value of all 1s (0x3FF for Control Symbol 24, 0xFFF for Control Symbol 48 and Control Symbol 64) indicates that the amount of delay is too large to be encoded.

When links are operating with Control Symbol 64, a loop-response shall consist of a single Control Symbol 64 Loop-Response control symbol format defined in Section 3.4.8.

When a loop-response is received while a loop-timing request is outstanding, the current value of the Timestamp Generator shall be captured in the Port n Timestamp 1 MSW CSRs and Port n Timestamp 1 LSW CSRs, and the delay value of the loop-response control symbol shall be captured in the Port n Timestamp Synchronization Status CSRs.

A processing element shall support receiving a loop-response when the Timestamp Master Supported bit of the Timestamp CAR is 1. A processing element shall support transmitting a loop-response when the Timestamp Slave Supported bit of the Timestamp CAR is 1.

6.5.3.5.3 Regular Timestamp Generator Re-synchronization

It may be necessary to regularly update TSG values throughout the RapidIO fabric; for example, when endpoints in the system do not support Common Frequency. Two methods of automatic re-synchronization are possible:

- If a device supports both a TSG Slave port that receives timestamp updates, and TSG Master ports that transmit timestamp updates, it is easiest to automatically update the TSG Master ports link partners whenever the TSG Slave port is updated.
- If a device is the TSG Master for the entire system, the device can be configured to regularly update its link partner’s sense of time.

A TSG Master port can be configured to update its link partner whenever the TSG is updated by setting the “Auto-update Link Partner Timestamp Generators” field to 1 in the Port n Timestamp Generator Synchronization CSRs.

A TSG Master port can be configured to periodically update its link partner. The period is programmed by setting the Update Period field of the Port n Auto Update Counter CSRs. Periodic updates are enabled by setting the Update Period field to a non-zero value in the Port n Auto Update Counter CSRs.

The rate of timestamp updates is controlled by the Port n Auto Update Counter CSRs. Timestamp updates must be sent at a rate which bounds the absolute time difference between the master timestamp generator and the slave timestamp generator. For example, assume the system requires the timestamp generators to be synchronized within 100 nanoseconds of each other, and the reference clocks for the timestamp generators can differ by 200 PPM. A frequency difference of 200 PPM will create a difference of 100 nanoseconds within 500 microseconds. Therefore, the Port n Auto Update Counter CSRs may be programmed to a value of 488 (500,000 nsec/1024 nanoseconds) to ensure timestamp updates are sent at the minimum rate required for timestamp generator synchronization.

Transmission errors may corrupt a timestamp update. Timestamp updates should be sent faster than the minimum rate to ensure that the slave timestamp generator will meet system requirements when a bit error corrupts a timestamp update. The actual rate of timestamp updates depends upon the bit error rate of the system and the systems tolerance to failure.

Extending the previous example, assume the timestamp updates are sent using a 4x IDLE3 link with a bit error rate of 10^{-12}. If the Port n Auto Update Counter CSRs is programmed to 162 (triple the minimum rate) and three consecutive timestamp updates are corrupted, the slave timestamp generator could have drifted more than 100 nanoseconds from the master timestamp generator. Conservatively assuming each timestamp update requires 335 bits, corruption of three consecutive timestamp updates will happen approximately once every 20 million years. The details of these computations are found in a spreadsheet available to RapidIO.org members.

Note that the slave timestamp generator is resynchronized by the next timestamp update. At that time, the slave timestamp generator could have drifted from the master by up to 133 nanoseconds.

6.5.3.5.4 Timestamp Generator Synchronization Control Symbol Jitter

The accuracy of TSG synchronization depends on the consistency with which frequency differences and loop delay can be measured. To ensure maximum accuracy, the following points should be considered with respect to TSG Slave and TSG Master support.

The point in the design where “Loop-Timing Request”, “Loop-Response for Loop-Timing Request”, and “Timestamp” control symbols are generated should have identical latency with respect to the Timestamp Generator from the time the control symbol is formulated to the time the control symbol is transmitted.

The point in the design where “Loop-Timing Request”, “Loop-Response for Loop-Timing Request”, and “Timestamp” control symbols are processed should have identical latency with respect to the Timestamp Generator from the time the control symbol is received by the SerDes to the time the control symbol is processed.

These two points ensure that measurements can be applied consistently to frequency offset calculations and loop delay calculations.

6.5.3.6 MECS Time Synchronization Protocol

The MECS Time Synchronization Protocol is a low cost mechanism for implementing time synchronization within a system. The MECS Time Synchronization Protocol makes the following simplifying assumptions regarding system operation:

- It is possible to know, or ignore, the latency of propagating (S)MECS from a source to every endpoint that must know time in the system.
- (S)MECS can be sent periodically to update time on all endpoints. The amount of time between successive (S)MECS transmissions is known as the “tick interval”.
- The starting value for a timestamp generator can be set using maintenance write packets with a request transmit-until-receive latency that is less than the “tick interval”.

Typically, these assumptions are only valid in systems with either static or well known configurations.

MECS Time Synchronization Protocol assumes that there is at least one source of MECS within the system, known as the “MECS Master”. A source of SMECS may also exist, known as the “SMECS Master”. Nodes that update their time based on received MECS/SMECS are known as “MECS Slaves”. The MECS and SMECS Masters are responsible for periodically transmitting MECS/SMECS that will be distributed through the RapidIO fabric to all of the MECS Slaves.

6.5.3.6.1 (S)MECS Master Operation

The MECS and SMECS Masters may be endpoints or switches. The (S)MECS Masters generate (S)MECS periodically, through mechanisms defined within the standard or by some other means.

The (S)MECS Master registers are programmed as follows to periodically transmit (S)MECS:

- The Timestamp Generator MSW CSR and Timestamp Generator LSW CSR are written with the current time. The Timestamp Generator begins incrementing.
- The MECS Next Timestamp MSW CSR and MECS Next Timestamp LSW CSR are written with the time of the first MECS transmission.

- The MECS Tick Interval CSR is written with the period for transmitting MECS. This register must be written before the Timestamp Generator timestamp value exceeds the MECS Next Timestamp timestamp value. Note that the MECS Time Synchronization Role bit shall be set to 1 on the MECS Master. If the device supports SMECS, the SMECS selection field shall be set appropriately.

The value chosen for the MECS Tick Interval CSR should be an exact multiple of the clock period for the Timestamp Generator in order to minimize transmission jitter. The “tick interval” should also be an exact multiple of the clock periods of each of the MECS Slaves.

The MECS Master compares the Timestamp Generator timestamp value with the MECS Next Timestamp timestamp value. If the Timestamp Generator timestamp value is equal to or greater than the MECS Next Timestamp value, the MECS Master shall:

- Transmit an (S)MECS
- Increment the (S)MECS Next Timestamp timestamp value by the value of the MECS Tick Interval CSR Tick Interval field.

Devices that support MECS Master operation shall be capable of transmitting MECS. Devices that support SMECS Master operation shall be capable of transmitting SMECS.

It is strongly encouraged to minimize (S)MECS transmission jitter in (S)MECS Master devices.

6.5.3.6.2 MECS Slave Operation

To begin tracking time on an MECS Slave, the MECS Slave registers are programmed as follows:

- The Timestamp Generator MSW CSR and Timestamp Generator LSW CSR are cleared to 0.
- The MECS Next Timestamp MSW CSR and MECS Next Timestamp LSW CSR are programmed with the timestamp value that shall be set when the next MECS is received by the MECS Slave.
- The MECS Tick Interval CSR is programmed with the “tick interval” value. Note that the MECS Time Synchronization Role bit shall be cleared to 0 on the MECS Slave. If the device supports SMECS, the SMECS selection field shall be set appropriately. This register must be written before the first MECS is received.

Once the registers have been programmed, reception of an MECS shall cause the following two actions to be performed by an MECS Slave:

The timestamp value contained in the MECS Next Timestamp MSW CSR and MECS Next Timestamp LSW CSR is used to update the Timestamp Generator

MSW CSR and Timestamp Generator LSW CSR. The rules for updating time described in section 6.5.3.5.1, "Setting and Reading a Timestamp Generator" shall be followed to prevent time from going backwards.

The tick interval found in the MECS Tick Interval CSR shall be added to the timestamp value found in the MECS Next Timestamp MSW CSR and MECS Next Timestamp LSW CSR, and written to the MECS Next Timestamp MSW CSR and MECS Next Timestamp LSW CSR.

Just as with the timestamp update protocol defined in 6.5.3.5.3, "Regular Timestamp Generator Re-synchronization", control symbols may be corrupted due to transmission errors. The implication is that an MECS may be lost, and MECS Slave time will be out of sync with the MECS Master to a degree that exceeds system specifications. For this reason, MECS Slave implementations may need to detect that an MECS has been lost. The mechanisms for detecting and correcting MECS loss are implementation specific and outside the scope of this specification.

Devices that support MECS Time Synchronization MECS Slave operation shall support reception of Multicast Event Control Symbols. It is strongly encouraged to minimize MECS reception jitter.

MECS routing in a system is controlled by the Port n Control CSRs “Multicast-Event Participant” bit field. MECS must be configured using a tree topology to avoid reception of multiple copies of the same original MECS.

SMECS propagation is controlled by a similar bit, with similar toplogy requirements, defined in the Port n SMECS Control CSR.

Note that Annex G, “MECS Time Synchronization (Informative)" discusses operational and implementation aspects of (S)MECS time synchronization.

6.6 Packets

6.6.1 Packet Delimiting

LP-Serial packets are delimited for transmission by control symbols. Since packet length is variable, both start-of-packet and end-of-packet delimiters are required. The start-of-packet delimiter immediately precedes the first character of the packet or an embedded delimited control symbol. With the exception stated in Section 6.6.1.2, the control symbol marking the end of a packet (packet termination) immediately follows the last character of the packet or the end of an embedded delimited control symbol.

The following control symbols are used to delimit packets.

- Start-of-packet
- End-of-packet
- Stomp

- Restart-from-retry
- Any link-request

6.6.1.1 Packet Start

The beginning of a packet (packet start) shall be marked by a start-of-packet control symbol.

6.6.1.2 Packet Termination

A packet shall be terminated in one of the following ways:

- The end of a packet is marked with an end-of-packet control symbol.
- The end of a packet is marked with a start-of-packet control symbol that also marks the beginning of a new packet.
- The packet is canceled by a restart-from-retry or stomp control symbol.
- The packet is canceled by any link-request control symbol. The cancellation of a packet by a link-request control symbol is subject to the requirement of Section 4.8.2 that every link-request control symbol transmitted on a link operating with IDLE2 be immediately preceded by a SYNC sequence, or subject to the requirements of Section 5.5.4.2 that every link-request control symbol transmitted on a link operating with IDLE3 be immediately preceded by a Seed ordered sequence.

If a link-request control symbol terminates a packet on a link that is operating with IDLE2, the SYNC sequence is required to precede the link-request control symbol. The SYNC sequence does not terminate the packet. If a link-request control symbol terminates a packet on a link that is operating with IDLE3, the Seed ordered sequence is required to precede the link-request control symbol. The Seed ordered sequence does not terminate the packet. The rules for marking the link-request control symbol as packet delimiting do not change. If the link-request control symbol is canceling a possibly truncated packet and the link is operating in 1x mode, the first character of the SYNC sequence shall immediately follow the last character of the canceled packet or of an embedded control symbol. If the link is operating in Nx Mode, the rules for Nx striping and padding shall be followed as stated in Section 4.10 for links operating with IDLE1 or IDLE2, or as stated in Section 5.13 for links operating with IDLE3.

6.6.2 Acknowledgment Identifier

Each packet requires an identifier to uniquely identify its acknowledgment control symbol. This identifier, the ackID, is 5 bits long when using Control Symbol 24, 6 bits when using Control Symbol 48, and 12 bits when using Control Symbol 64. This allows up to 2^N outstanding unacknowledged request and/or response packets where N is the number of bits in the ackID field. To eliminate the ambiguity between 0 and 2^N outstanding packets, a maximum of 2^N-1 outstanding unacknowledged packets shall be allowed at any one time.

The value of ackID assigned to the first packet transmitted after a reset shall be 0. The values of ackID assigned to subsequent packets shall be in increasing numerical order, wrapping back to 0 on overflow. The ackID assigned to a packet indicates the order of the packet transmission and is independent of the virtual channel assignment of the packet.

The acknowledgment control symbols are defined in Chapter 3, "Control Symbols". When acknowledgement control symbols are received containing VC specific information (e.g., buf_status), the transmitter side of the port must re-associate that information with the correct VC based on the returned ackID.

Devices that support Control Symbol 64 shall support configuration values whereby Packet Accepted controls symbols sent and/or received acknowledge multiple packets. The configuration shall be controlled by the Port n Latency Optimization CSRs. When transmitting control symbols, devices operating with Control Symbol 24 or Control Symbol 48 shall support a default configuration in which they send one Packet Accepted control symbol for each received packet. Devices operating with Control Symbol 24 or Control Symbol 48 may optionally support a configuration in which they may transmit one Packet Accepted control symbol for multiple received packets. Devices operating with Control Symbol 64 may transmit one Packet Accepted control symbol for multiple received packets.

Devices operating with Control Symbol 24 or Control Symbol 48 may optionally support reception of Packet Accepted control symbols which acknowledge all outstanding packets up to and including the packet ackID. Devices operating with Control Symbol 64 shall support reception of Packet Accepted control symbols which acknowledge all outstanding packets up to and including the packet ackID. It shall be possible to configure devices operating with Control Symbol 64 to transmit a Packet Accepted control symbol for each received packet.

6.6.3 Packet Priority and Transaction Request Flows

Within VC0 each packet has a priority, and optionally a critical request flow, 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 RapidIO for several purposes which include transaction ordering and deadlock prevention. The critical request flow is carried in the CRF bit. It allows a flow to be designated as a critical or preferred flow with respect to other flows of the same priority. Support for critical request flows is strongly encouraged.

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 (and optionally the CRF bit) of the packet. If the CRF bit is not supported, 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 below.

Table 6-6. VC0 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

If the CRF bit is supported, the transaction request flows are mapped similarly as specified in Table 6-7. Endpoints that do not support the CRF bit treat it as reserved, setting it to logic 0 on transmit and ignoring it on receive.

Table 6-7. VC0 Transaction Request Flow to Priority and Critical Request Flow Mapping

Flow	System Priority	Request CRF Bit Setting	Request Packet Priority	Response CRF Bit Setting	Response Packet Priority
F or higher	Highest	1	2	1	3
E	Higher than A, B, C, D	0	2	0	3
D	Higher than A, B, C	1	1	1	2 or 3
C	Higher than A, B	0	1	0	2 or 3
B	Higher than A	1	0	1	1, 2, or 3
A	Lowest	0	0	0	1, 2, or 3

The mapping of transaction request flows allows a RapidIO transport fabric to maintain transaction request flow ordering without the fabric having any knowledge of transaction types or their interdependencies. This allows a RapidIO 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 and critical request flow for each path through the fabric and can maintain transaction request flow priority by never allowing a lower priority packet to pass a higher priority packet taking the same path through the fabric. In the case of congestion or some other restriction, a set CRF bit indicates that a flow of a priority can pass a flow of the same priority without the CRF bit set.

For VC0, flows identified as A - F (or higher) are synonymous with 0A - 0F, etc. Flows for VCs 1-8 (A and higher) are identified as 1A, 2A,...8A. All traffic in flows 1A-8A are transaction requests which do not require a response. Transaction requests that require responses, and their corresponding responses, must use VC0 with the appropriate priority.

Table 6-8. Flow IDs for VCs

Transaction Request Flow	VC	Transaction Request Flow	VC
1A and higher	VC1	5A and higher	VC5
2A and higher	VC2	6A and higher	VC6
3A and higher	VC3	7A and higher	VC7
4A and higher	VC4	8A and higher	VC8

6.7 Link Maintenance Protocol

The link maintenance protocol involves a request and response pair between ports connected by a LP-Serial link. For software management, the request is generated through ports in the configuration space of the sending device. An external host write of a command to the link-request register with an I/O logical specification maintenance write transaction causes a 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 a link-response control symbol if the link-request command required it to do so. The external host retrieves the link-response by polling the link-response register with I/O logical maintenance read transactions. A device with multiple RapidIO interfaces has a link-request and a link-response register pair for each corresponding RapidIO interface.

The automatic error recovery mechanism relies on the hardware generating packet-not-accepted and link-request/port-status control symbols under the transmission error conditions described in Section 6.13.2.1, "Recoverable Errors", and using the corresponding link-response information to attempt recovery.

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

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

The link-request/port-status control symbol requires a response. A port receiving a link-request/port-status control symbol returns a link-response control symbol containing two pieces of information:

- port_status
- ackID_status

The port_status indicators are described in Table 3-14 for Control Symbol 24 and Control Symbol 48 operation and in Table 3-15 for Control Symbol 64 operation.

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/port-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/port-status control symbol is received.

6.8 Packet Transmission Protocol

The LP-Serial protocol for packet transmission provides link level flow and error detection and recovery.

The protocol uses control symbols to delimit packets when they are transmitted across a LP-Serial link as specified in Section 6.6.1, "Packet Delimiting".

The link protocol uses acknowledgment to monitor packet transmission. With two exceptions, each packet transmitted across a LP-Serial link shall be acknowledged by the receiving port with a packet acknowledgment control symbol. Packets shall be acknowledged in the order in which they were transmitted (ackID order). The first exception occurs when a single packet-acknowledge control symbol acknowledges multiple packets. The second exception is when an event has occurred that caused a port to enter the Input Error-stopped state. CT mode packets accepted by a port after the port entered the Input Error-stopped state and before the port receives a link-request/port-status control symbol shall not be acknowledged.

To associate packet acknowledgment control symbols with transmitted packets, each packet shall be assigned an ackID value according to the rules of Section 6.6.2, "Acknowledgment Identifier" that is carried in the ackID field of the packet and the packet_ackID field of the associated acknowledgment control symbol. The ackID value carried by a packet indicates its order of transmission and the order in which it is acknowledged.

The LP-Serial link RT protocol uses retransmission to recover from packet transmission errors or a lack of receive buffer resources. To enable packet retransmission, a copy of each RT packet transmitted across a LP-Serial link shall be kept by the sending port until either a packet-accepted control symbol is received for the packet or the sending port determines that the packet has encountered an unrecoverable error condition.

The LP-Serial link CT protocol does not use packet retransmission. CT mode packets that are corrupted by transmission errors or that are not accepted because of a lack of receive buffer resources are discarded and lost. Therefore, a port need not retain a copy of a CT mode packet whose transmission has been completed.

The LP-Serial link protocol uses the ackID value carried in each packet to ensure that no RT mode packets are lost due to transmission errors. With one exception, a port shall accept packets from a LP-Serial link only in sequential ackID order, i.e. if the ackID value of the last packet accepted was N, the ackID value of the next packet that is accepted must be (N+1) modulo2^n where n is the number of bits in the ackID field. The exception is when an event has occurred that caused a port to enter the Input Error-stopped state. A CT mode packet received by a port after the port entered the Input Error-stopped state, and before the port receives a link-request/port-status control symbol, shall be accepted by the port without regard to the value of the packet’s ackID field if the packet is otherwise error free and there are adequate receive buffer resources to accept the packet. The value that is maintained by the port of the ackID expected in the next packet shall not be changed by the acceptance of CT packets during this period.

A LP-Serial port accepts or rejects each error free packet that it receives with the expected ackID depending on whether the port has input buffer space available for the VC and/or priority level of the packet. The use of the packet-accepted, packet-retry, packet-not-accepted and restart-from-retry control symbols and the buf_status field in packet acknowledgment control symbols to control the flow of packets across a LP-Serial link is covered in Section 6.9, "Flow Control".

The LP-Serial link protocol allows a packet that is being transmitted to be canceled at any point during its transmission. Packet cancellation is covered in Section 6.10, "Canceling Packets".

The LP-Serial link protocol provides detection and recovery processes for both transmission errors and protocol violations. The enumeration of detectable errors, the detection of errors and the associated error recovery processes are covered in Section 6.13, "Error Detection and Recovery".

In order to prevent switch processing element internal errors, such as SRAM soft bit errors, from silently corrupting a packet and the system, switch processing elements shall maintain packet error detection coverage while a packet is passing through the switch. The simplest method for maintaining packet error detection coverage is to pass the packet CRC through the switch as part of the packet. This works well for all non-maintenance packets whose CRC does not change as the packets are transported from source to destination through the fabric. Maintaining error detection coverage is more complicated for maintenance packets as their hop_count and CRC change every time they pass through a switch. However, passing the packet CRC through the switch as part of the packet does not prevent packet loss due to soft errors within the switch. Recovery from soft errors within a switch requires that each packet passing through the switch be covered by some type of error correction of adequate strength.

In order to support transaction ordering requirements of the Logical Layer Specifications, the LP-Serial protocol imposes packet delivery ordering requirements within the Physical Layer and transaction delivery ordering requirements between the Physical Layer and the Transport Layer in end point processing elements. These requirements are covered in Section 6.11, "Transaction and Packet Delivery Ordering Rules".

In order to prevent deadlock, the LP-Serial protocol imposes a set of deadlock prevention rules. These rules are covered in Section 6.12, "Deadlock Avoidance".

This specification provides both bandwidth reservation and priority based channels. Priority scheduling may or may not be included in the reservation of bandwidth. Whatever allocation of bandwidth is used for priority traffic, higher level flows will reduce the bandwidth available for lower level flows. It is possible that traffic associated with higher flow levels can starve traffic associated with lower flow levels. It is important to use the available flows properly for the transaction type, to insure the rules in Section 6.11, "Transaction and Packet Delivery Ordering Rules" and Section 6.12, "Deadlock Avoidance" are met. The actual mechanisms used to schedule traffic are beyond the scope of this specification.

6.9 Flow Control

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

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

6.9.1 Receiver-Controlled Flow Control

Receiver-controlled flow control is the simplest and basic method of flow control. In this method, the input side of a port controls the flow of packets from its link partner by accepting or rejecting 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 for the VC and 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. The port optionally acknowledges multiple packets with a single packet-accepted control symbol. Transmission of a packet-accepted control symbol informs the port's link partner that the packet (or packets) 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 (or packets) freeing buffer space in the partner.

The remaining behavior is a function of the mode of the VC.

6.9.1.1 Reliable Traffic VC Receivers

If buffer space is not available, the port rejects the packet. If multiple VCs are active, and the VC is in reliable traffic mode, the rejected packet shall be acknowledged with the packet-not-accepted control symbol. The cause field of the control symbol should be set to "packet not accepted due to lack of resources". The "arbitrary, or ackID_Status" field of the packet-not-accepted control symbol can be set to the ackID of the retried packet. In this case, the packets associated with ackIDs up to, but not including, the retried ackID are acknowledged by the packet-not-accepted control symbol. For information about control of the "arbitrary, or ackID_Status" field refer to Section 7.6.15, "Port n Latency Optimization CSRs". Reception of the packet-not-accepted control symbol causes the entire "RT Group" to go through the same process used in error recovery to resequence and retransmit the RT packets. See Section 6.13, "Error Detection and Recovery" for details.

If the port is operating in single VC mode, the port may use the Packet Retry protocol described in Section 6.9.1.3, "Single VC Retry Protocol", or it may continue to use the packet-not-accepted protocol described above.

6.9.1.2 Continuous Traffic VC Receivers

If buffer space is not available, and the VC is in CT mode, the packet is acknowledged as accepted, and the packet is discarded. This preserves the order of the normal link response and does not impact performance. Receiver based flow control for CT channels will result in packet loss due to receiver overruns depending on bandwidth and buffering conditions. See Section 6.9.2, "Transmitter-Controlled Flow Control" for transmitter based flow control options.

6.9.1.3 Single VC Retry Protocol

When operating with a single VC (VC0), the receiver may use the retry protocol for handling receiver overruns. It is a requirement that implementers include this functionality in the channel design to be backward compatible with existing RapidIO interfaces.

When a port rejects a packet, it immediately enters the Input Retry-stopped state and follows the Input Retry-stopped recovery process specified in Section 6.9.1.4, "Input Retry-Stopped Recovery Process". As part of the Input Retry-stopped recovery process, the port sends a packet-retry control symbol to its link partner indicating that the packet whose ackID is in the packet_ackID field of the control symbol and all packets subsequently transmitted by the port have been discarded by the link partner and must all be retransmitted. When the ability to acknowledge multiple packets with a single control symbol is enabled, the packet-retry control symbol shall acknowledge all packets up to, but not including, the ackID in the retry control symbol. 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.

Devices may optionally support configuration values whereby Retry control symbols sent and/or received acknowledge multiple packets. The configuration shall be controlled by the Port n Latency Optimization CSRs. When transmitting control symbols, devices operating with Control Symbol 24 or Control Symbol 48 shall support a default configuration in which the Retry control symbol does not acknowledge packets. Devices operating with Control Symbol 24 or Control Symbol 48 may optionally support a configuration in which Retry control symbols acknowledge all packets up to, but not including, the ackID_status in the Retry control symbol. Devices operating with Control Symbol 64 shall support a default configuration whereby a transmitted Retry control symbol acknowledges all packets up to, but not including, the ackID_status in the Retry control symbol.

Devices operating with Control Symbol 24 or Control Symbol 48 may optionally support reception of Retry control symbols that acknowledge all outstanding packets up to, but not including, the ackID_status. Devices operating with Control Symbol 64 shall support reception of Retry control symbols that acknowledge all outstanding packets up to, but not including, the ackID_status. It shall be possible to configure devices operating with Control Symbol 64 to transmit a Retry control symbol only when all packets up to the Retried packet have been acknowledged.

A port that receives a packet-retry control symbol immediately enters the Output Retry-stopped state and follows the Output Retry-stopped recovery process specified in Section 6.9.1.5, "Output Retry-Stopped Recovery Process". As part of the Output Retry-stopped recovery process, the port receiving the packet-retry control symbol sends a restart-from-retry control symbol which causes its link partner to exit the Input Retry-stopped state and resume packet reception. The ackID assigned to that first packet transmitted after the restart-from-retry control symbol is the ackID of the packet that was retried.

Figure 6-4 shows an example of single VC 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.

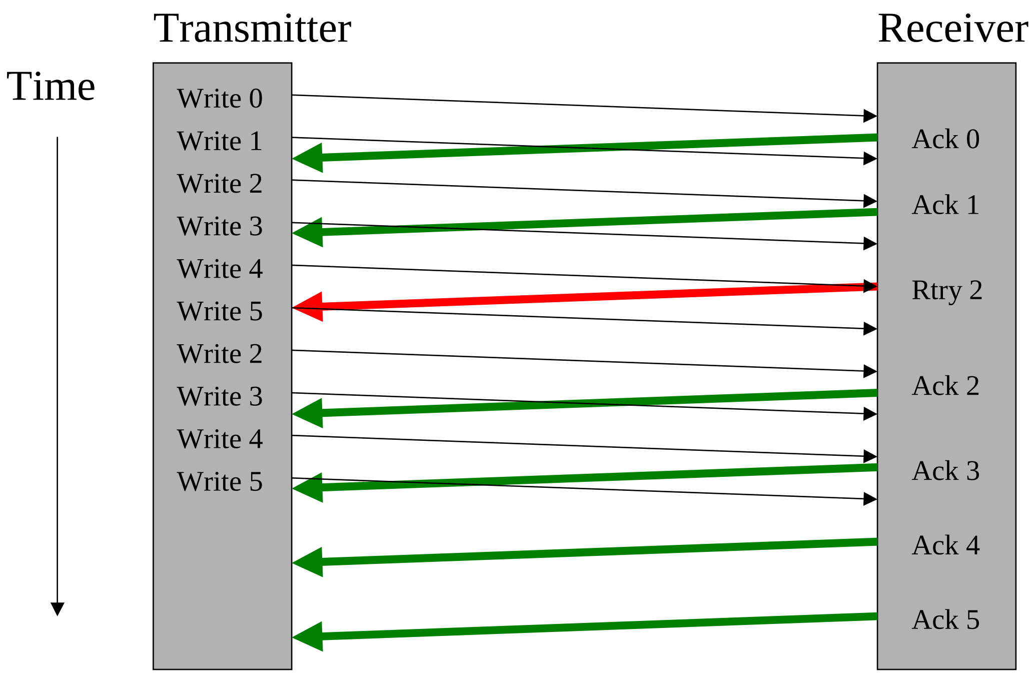


Figure 6-4. Single VC Mode Receiver-Controlled Flow Control

6.9.1.4 Input Retry-Stopped Recovery Process

When the input side of a port operating with only VC0 active (single VC mode) retries a packet, it immediately enters the Input Retry-stopped state. To recover from this state, the input side of the port takes the following actions.

- Discards the rejected or canceled packet without reporting a packet error and ignores all subsequently received packets while the port is in the Input Retry-stopped state.

- Causes the output side of the port to issue a packet-retry control symbol containing the ackID value of the retried packet in the packet_ackID field of the control symbol. (The packet-retry control symbol causes the output side of the link partner to enter the Output Retry-stopped state and send a restart-from-retry control symbol.)
- When a restart-from-retry control symbol is received, exit the Input Retry-stopped state and resume packet reception.

An example state machine with the behavior described in this section is included in Section C.2, "Packet Retry Mechanism".

6.9.1.5 Output Retry-Stopped Recovery Process

To recover from the Output Retry-stopped state, the output side of a port takes the following actions.

- Immediately stops transmitting new packets.
- Resets the link packet acknowledgment timers for all transmitted but unacknowledged packets. (This prevents the generation of spurious timeout errors.)
- Transmits a restart-from-retry control symbol.
- Backs up to the first unaccepted packet (the retried packet) which is the packet whose ackID value is specified by the packet_ackID value contained in the packet-retry control symbol. (The packet_ackID value is also the value of ackID field the port retrying the packet expects in the first packet it receives after receiving the restart-from-retry control symbol.)
- Exits the Output Retry-stopped state and resumes transmission with either the retried packet or a higher priority packet which is assigned the ackID value contained in the packet_ackID field of the packet-retry control symbol.

An example state machine with the behavior described in this section is included in Section C.2, "Packet Retry Mechanism".

6.9.2 Transmitter-Controlled Flow Control

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

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

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

A port informs its link partner when the number of free buffers available for packet reception changes. The new value of buf_status is conveyed in the buf_status field of a packet-accepted, packet-retry, status, or VC_status control symbol. Each change in the number of free buffers a port has available for packet reception need not be conveyed to the link partner. However, a port shall send a control symbol containing the buf_status field to its link partner no less often than the minimum rate specified in Section 6.5.3.2, "Buffer Status Maintenance".

When a port implements more than VC0, the value of buf_status is kept on a per VC basis by the receiving port. When a packet-accepted symbol is returned, the buf_status field is filled with the status for the specific VC that the packet was sent to. When sending buf_status asynchronously (not in response to any specific packet), the status control symbol is used for VC0, and the VC_status control symbol is used for VC’s 1-8.

A port whose link partner is operating in transmitter-control flow control mode should never receive a packet-not-accepted (or packet-retry control symbol if operating in single VC mode) from its link partner unless the port has transmitted more packets than its link partner has receive buffers, has violated the rules that all input buffers may not be filled with low priority packets or there is some fault condition. A receiver overrun is handled according to the rules in 6.9.1, "Receiver-Controlled Flow Control".

If a port, operating in single VC mode, for whose link partner is operating in transmitter-control flow control mode, receives a packet-retry control symbol, the output side of the port immediately enters the Output Retry-stopped state and follows the Output Retry-stopped recovery process specified in Section 6.9.1.5, "Output Retry-Stopped Recovery Process".

A simple example of single VC transmitter-controlled flow control is shown in Figure 6-5.

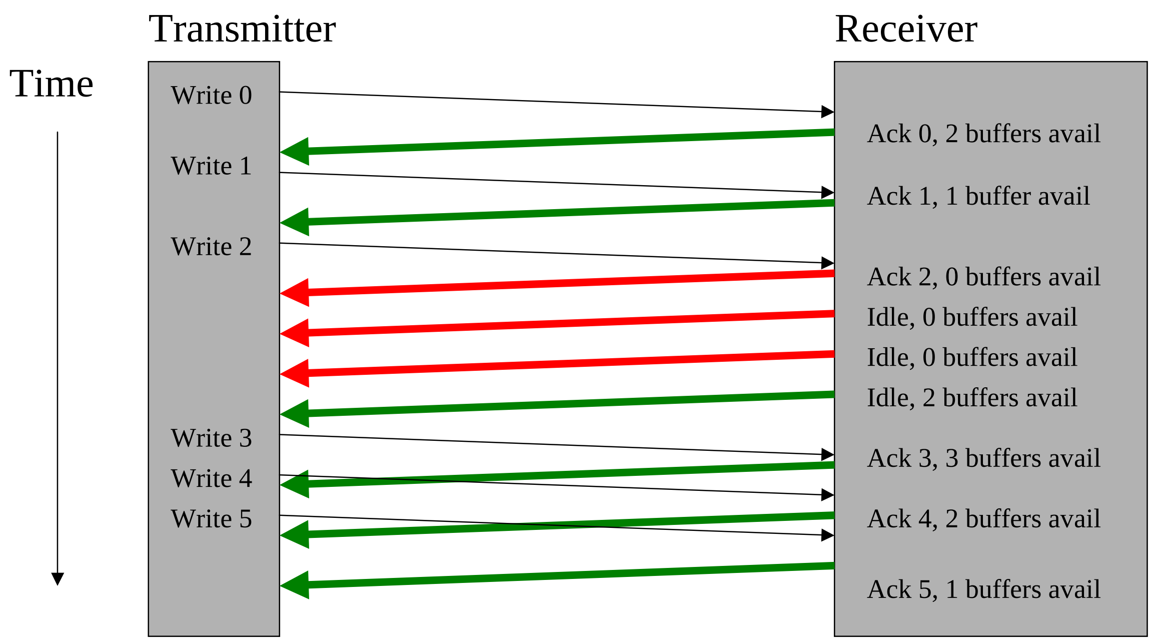


Figure 6-5. Single VC Mode Transmitter-Controlled Flow Control

6.9.2.1 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 6.12, "Deadlock Avoidance" concerning the acceptance of packets by ports.

For VC0, it is important to manage buffers in a way that reserves room for high priority packets. One possible implementation to organize the buffers is to establish watermarks and use them to progressively limit the packet priorities that can be transmitted as the effective number of free VC0 buffers in the receiver decreases. For example, VC0 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 VC0 buffers decreases. Designate the three watermarks as WM0, WM1, and WM2 where WM0 > WM1 > WM2 > 0 and employ the following rules.

If free_buffer_count0 >= WM0, all priority packets may be transmitted.

If WM0 > free_buffer_count0 >= WM1, only priority 1, 2, and 3 packets may be transmitted.

If WM1 > free_buffer_count0 >= WM2, only priority 2 and 3 packets may be transmitted.

If WM2 > free_buffer_count0, only priority 3 packets may be transmitted.

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

WM0 = 4

WM1 = 3

WM2 = 2

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

The three watermark values and the number of free buffers reported in the buf_status field of status control symbols received by the port following link initialization and before the start of packet transmission may be stored in a CSR. Since the maximum value of each of these four items is 4094 when using Control Symbol 64, each will fit in an 12-bit field and all four will not fit in a single 32-bit CSR. If the watermarks are software setable, the three watermark fields in the CSRs should be writable. For the greatest flexibility, a set of watermark registers should be provided for each port on a device.

For VCs 1-8, packets within the same VC are equal in priority and always kept in order. The free buffers in the receiver can be partitioned between VCs in any number of ways: they can be equally divided among the VCs, they can be statically partitioned based on the bandwidth allocation percentages, or they may be dynamically allocated from a larger pool. The only requirement is that once a given amount of buffers is reported by the receiver to the transmitter those buffers shall remain available for packets for that VC. Buffers may be deallocated once they are used, by removing the data, but not reporting the buffer available to that VC. At that time, the buffer may be reallocated to another VC. The specific method of buffer allocation is beyond the scope of this specification.

6.9.2.2 Effective Number of Free Receive Buffers

The number of buffers available in a link partner for packet reception on a given VC is typically less than the value of the buf_status field most recently received for that VC from the link partner. The value in the buf_status field does not account for packets that have been transmitted by the VC but not acknowledged by its link partner. The variable free_buffer_countN is defined to be the effective number of free buffers available in the link partner for packet reception on VC N. The recommended way for a port to compute and maintain these “free buffer counts” is to implement the following rules.

1. Each active VC maintains a variable “free_buffer_countVC” whose value shall be the effective number of free buffers available to that VC in the link partner for packet reception.
2. Each active VC maintains a variable “outstanding_packet_countVC” whose value is number of packets that have been transmitted on that VC, but that have not been acknowledged by its link partner.

3. After link initialization and before the start of packet transmission,

```
If {[(control_symbol = cs24) & (received_buf_status < 31)] |
        [(control_symbol = cs48) & (received_buf_status < 63)] |
        [(control_symbol = cs64) & (received_buf_status < 4095)]}
{
        flow_control_mode = transmitter;
        free_buffer_count0 = received_buf_status0;
        outstanding_packet_count0 = 0;
        for VC 1 through 8 {
            free_buffer_countVC =
                            received_VC_buffer_statusVC
            outstanding_packet_countVC = 0
        }
}
else
        flow_control_mode = receiver;
```

4. When a status or VC_Status control symbol is received by the port,

```
free_buffer_countVC =
        received_buf_statusVC - outstanding_packet_countVC;
```

5. When a packet is transmitted by the VC,

```
outstanding_packet_count VC=
                        outstanding_packet_countVC + 1
free_buffer_countVC = free_buffer_countVC - 1
```

6. When a packet-accepted control symbol is received by the port indicating that a packet has been accepted by the link partner, the buf_status field of the control symbol is re-associated with the originating VC:

```
Outstanding_packet_countVC =
                            Outstanding_packet_countVC - 1;
free_buffer_countVC =
        received_buf_statusVC - outstanding_packet_countVC;
```

7. 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_count0 = 0;
free_buffer_count0 = received_buf_status0;
```

8. 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 or a lack of buffer resources,

Outstanding_packet_countVC = 0;
free_buffer_countVC = free_buffer_count VC (remains unchanged);

9. When a link-response control symbol is received,

free_buffer_count0 = received_buf_status;

6.9.2.3 Speculative Packet Transmission

A port whose link partner is operating in transmitter-controlled flow control mode may send more packets on a given VC than the number of free buffers indicated by the link partner as being available for that VC. 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.

When the link has multiple operating VCs, speculative packet transmission may increase the CT packet loss rate and how frequently the link runs the error-recovery process.

6.9.3 Flow Control Mode Negotiation

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

The flow control mode negotiation rule is as follows:

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

If multiple VCs are used, then a port shall have either all channels in receiver based flow control or all channels in transmitter based flow control. All status and VC_status control symbols shall be consistent in their buf_status reporting in this regard.

6.10 Canceling Packets

When a port becomes aware of some condition that will require the packet it is currently transmitting to be retransmitted, the port may cancel the packet. This allows the port to avoid wasting bandwidth by not completing the transmission of a packet that the port knows must be retransmitted. Alternatively, the sending port may choose to complete transmission of the packet normally.

A port may cancel a packet if the port detects a problem with the packet as it is being transmitted or if the port receives a packet-retry or packet-not-accepted control symbol for a packet that is still being transmitted or that was previously transmitted. A packet-retry or packet-not-accepted control symbol can be transmitted by a port for a packet at any time after the port begins receiving the packet.

The sending device shall use the stomp control symbol, the restart-from-retry control symbol (in response to a packet-retry control symbol), or any link request control symbol to cancel a packet.

A port receiving a canceled packet shall drop the packet. The cancellation of a packet shall not result in the generation or report of any errors. If the packet was canceled because the sender received a packet-not-accepted control symbol, the error that caused the packet-not-accepted to be sent shall be reported in the normal manner.

The behavior of a port that receives a canceled packet depends on the control symbol that canceled the packet. A port that is not in an input stopped state (Retry-stopped or Error-stopped) while receiving the canceled packet and has not previously acknowledged the packet shall have the following behavior.

> If the packet is canceled by a link-request/port-status control symbol, the port shall drop the packet without reporting a packet error.
>
> If the packet is canceled by a restart-from-retry control symbol a protocol error has occurred and the port shall immediately enter the Input Error-stopped state and follows the Input Error-stopped recovery process specified in Section 6.13.2.6, "Input Error-Stopped Recovery Process".
>
> If the packet was canceled by other than a restart-from-retry or link-request/port-status control symbol and the port is operating in single VC mode (only VC0 is active), the port shall immediately enter the Input Retry-Stopped state and follow the Input Retry-Stopped recovery process specified in Section 6.9.1.4, "Input Retry-Stopped Recovery Process". If the packet was canceled before the packet ackID field was received by the port, the packet_ackID field of the associated packet-retry control symbol acknowledging the packet shall be set to the ackID the port expected in the canceled packet
>
> If the packet was canceled by other than a restart-from-retry or link-request/port-status control symbol and the port is operating in multiple VC mode (at least one of VC1-8 is active), the port shall immediately enter the Input Error-Stopped state and follow the Input Error-Stopped recovery process specified in Section 6.13.2.6, "Input Error-Stopped Recovery Process".

A packet whose transmission is canceled shall be considered to be an untransmitted packet.

6.11 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 forward 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, critical request flow or VC 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, same critical request flow, same VC bit, 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).**

Rules for Scheduling Among VCs:

The whole link bandwidth is evenly divided into ‘N’ portions and each portion is 1/N of the whole link bandwidth. Each VC is configured to have guaranteed bandwidth. The method among VCs is also vendor dependent, as long as it satisfies the following three rules:

1. If the total guaranteed bandwidth for all the supported VCs is more than 100%, then the received bandwidth for each supported VC cannot be guaranteed.
2. If the total guaranteed bandwidth for all the supported VCs is less than or equal to 100%, demand for more than its guaranteed bandwidth shall not cause any other VCs to receive less than their guaranteed bandwidth.
3. If one VC demands less bandwidth than its guaranteed bandwidth, the extra bandwidth may be distributed among other VCs.

If VC0 participates in the bandwidth reservation process, then all VCs will receive their expected minimum bandwidth. However, VC0 may be treated as a special case. VC0 may be treated with strict priority, getting whatever bandwidth is required when it has traffic to transport. In this condition, the remaining VCs will divide up whatever portion of bandwidth remains. If VC0’s utilization is significant, compared with the traffic on the other VCs, then the other VCs bandwidth will still be proportional to each other, but will vary as the available bandwidth is modified by VC0.

The implementer may also choose to implement some priorities within VC0 with strict priority, and schedule the rest with reserved bandwidth. This specification does not require any particular treatment as there are application cases for any of the above. Chapter 7, "LP-Serial Registers" defines a standard control register should the implementer decide to make this a programmable feature.

6.12 Deadlock Avoidance

Request transactions requiring responses shall only use VC0. The response packet shall only use VC0. The following requirements apply to prioritized traffic within VC0.

To allow a RapidIO protocol to evolve without changing the switching fabric, switch processing elements are not required, with the sole exception of ftype 8 maintenance transactions, to discern between packet types, their functions or their interdependencies. Switches, for instance, are not required to discern between packets carrying request transactions and packets carrying response transactions. As a result, it is possible for two end points, A and B to each fill all of their output buffers, the fabric connecting them and the other end point's input buffers with read requests. This would result in an input to output dependency loop in each end point in which there would be no buffer space to hold the responses necessary to complete any of the outstanding read requests.

To break input to output dependencies, end point processing elements must have the ability to issue outbound response packets even if outbound request packets awaiting transmission are congestion blocked by the connected device. Two techniques are provided to break input to output dependencies. First, a response packet (a packet carrying a response transaction) is always assigned an initial priority one priority level greater than the priority of the associated request packet (the packet carrying the associated request transaction).

This requirement is specified in Table 6-6 and Table 6-7. It breaks the dependency cycle at the request flow level. Second, the end point processing element that is the source of the response packet may additionally raise the priority of the response packet to a priority higher than the minimum required by Table 6-6 and Table 6-7 if necessary for the packet to be accepted by the connected device. This additional increase in response packet priority above the minimum required by Table 6-6 and Table 6-7 is called promotion. An end point processing element may promote a response packet only to the degree necessary for the packet to be accepted by the connected device.

The following rules define the deadlock prevention mechanism:

Deadlock Prevention Rules:

1. **A RapidIO fabric shall be dependency cycle free for all operations that do not require a response. (This rule is necessary as there are no mechanisms provided in the fabric to break dependency cycles for operations not requiring responses.)**
2. **A packet carrying a request transaction that requires a response shall not be issued at the highest priority. (This rule ensures that an end point processing element can issue a response packet at a priority higher than the priority of the associated request. This rule in combination with rule 3 are basis for the priority assignments in Table 6-6 and Table 6-7)**
3. **A packet carrying a response shall have a priority at least one priority level higher than the priority of the associated request. (This rule in combination with rule 2 are basis for the priority assignments in Table 6-6 and Table 6-7)**
4. **A switch processing element port shall accept an error-free packet of priority N if there is no packet of priority greater**

than or equal to N that was previously received by the port and is still waiting in the switch to be forwarded. (This rule has multiple implications which include but are not limited to the following. First, a switch processing element port must have at least as many maximum length packet input buffers as there are priority levels. Second, a minimum of one maximum length packet input buffer must be reserved for each priority level. A input buffer reserved for priority N might be restricted to only priority N packets or might be allowed to hold packets of priority greater than or equal to N, either approach complies with the rule.)

5. **A switch processing element port that transmits a priority N packet that is forced to retry by the connected device shall select a packet of priority greater than N, if one is available, for transmission. (This guarantees that packets of a given priority will not block higher priority packets.)**
6. **An end point processing element port shall accept an error-free packet of priority N if the port has enough space for the packet in the input buffer space of the port allocated for packets of priority N. (Lack of input buffer space is the only reason an end point may retry a packet.)**
7. **The decision of an end point processing element to accept or retry an error-free packet of priority N shall not depend on the ability of the end point to issue request packets of priority less than or equal to N from any of its ports. (This rule works in conjunction with rule 6. It prohibits a device's inability to issue packets of priority less than or equal to N, due to congestion in the connected device, from resulting in a lack of buffers to receive inbound packets of priority greater than or equal to N which in turn would result in packets of priority greater than or equal to N being forced to retry. The implications and some ways of complying with this rule are presented in the following paragraphs.)**

One implication of Rule 7 is that a port may not fill all of its buffers that can be used to hold packets awaiting transmission with packets carrying request transactions. If this situation was allowed to occur and the output was blocked due to congestion in the connected device, read transactions could not be processed (no place to put the response packet), input buffer space would become filled and all subsequent inbound request packets would be forced to retry violating Rule 7.

Another implication is that a port must have a way of preventing output blockage at priority less than or equal to N, due to congestion in the connected device, from resulting in a lack of input buffer space for inbound packets of priority greater than or equal to N. There are multiple ways of doing this.

One way is to provide a port with input buffer space for at least four maximum length packets and reserve input buffer space for higher priority packets in a manner similar to that required by Rule 4 for switches. In this case, output port blockage at priority less than or equal to N will not result in blocking inbound packets of priority greater than or equal to N as any responses packets they generate will be of priority greater than N which is not congestion blocked. The port must however have the ability to select packets of priority greater than N for transmission from the packets awaiting transmission. This approach does not require the use of response packet priority promotion.

A port can use the promotion mechanism to increase the priority of response packets until they are accepted by the connected device. This allows output buffer space containing response packets to be freed even though all request packets awaiting transmission are congestion blocked.

As an example, suppose an end point processing element has a blocked input port because all available resources are being used for a response packet that the processing element is trying to send. If the response packet is retried by the downstream processing element, raising the priority of the response packet until it is accepted allows the processing element’s input port to unblock so the system can make forward progress.

It should be noted that implementing response priority promotion in a device may help with its link partner’s input buffer congestion, not its own input buffer congestion. It should also be noted that response priority promotion may not be able to guarantee forward progress in the system unless the link partner has implemented priority based input buffer reservation.

6.13 Error Detection and Recovery

Error detection and recovery is becoming a more important issue for many systems. The LP-Serial specification provides extensive error detection and recovery by combining retry protocols with cyclic redundancy codes, the selection of delimiter special characters and response timers.

One feature of the error protection strategy is that with the sole exception of maintenance packets, the CRC value carried in a packet remains unchanged as the packet moves through the fabric. The CRC carried in a maintenance packet must be regenerated at each switch as the hop count changes.

6.13.1 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 timeout 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 timeout is not signaled. The response to this error condition is implementation dependent.

The RapidIO specifications require timeout counters for the Physical Layer, the port link timeout counters, and counters for the Logical Layer, the port response timeout counters. The interpretation of the counter values is implementation dependent, based on a number of factors including link clock rate, the internal clock rate of the device, and the desired system behavior.

The Physical Layer timeout occurs between the transmission of a packet and the receipt of an acknowledgment control symbol. This timeout interval is likely to be comparatively short because the packet and acknowledgment pair must only traverse a single link.

The Logical Layer timeout occurs between the issuance of a request packet that requires a response packet and the receipt of that response packet. This timeout 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 timeout will occur. This timeout 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 timeout 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 behavior is supported by the issuing processing element, the port response timeout implementation must look for both responses, regardless on which links they are returned.

Link reinitialization, port width changes, and link retraining can temporarily interrupt the flow of packets and control symbols in one or both directions on a link, unexpectedly increasing the time required to receive responses. For this reason, it is recommended that the port link timeout counters and the port response timeout counters should not advance when link_initialized is deasserted. For ports operating with IDLE3, port link timeout counters and the port response timeout counters should not advance if either receive_enable or transmit_enable are deasserted.

6.13.2 Link Behavior Under Error

The LP-Serial link uses error detection and retransmission to protect RT packets against loss or corruption due to transmission errors. Transmission error detection is done at the input port, and all transmission error recovery is also initiated at the input port.

The packet transmission protocol requires that each RT packet transmitted by a port be acknowledged by the receiving port and that a port retain a copy of each RT packet that it transmits until the port receives a packet-accepted control symbol acknowledgment for the packet or the sending port determines that the packet has encountered an unrecoverable error. If the receiving port detects a transmission error in a packet, the port sends a packet-not-accepted control symbol acknowledgment back to the sender indicating that the packet was corrupted as received. After a link-request/port-status and link-response control symbol exchange, the sender begins retransmission with the next packet according to the priority/bandwidth scheduling rules. The RT VCs retransmit all packets that were unacknowledged at the time of the error. CT VCs continue with the next untransmitted packet.

All RT packets corrupted in transmission are retransmitted. The number of times a packet can be retransmitted before the sending port determines that the packet has encountered an unrecoverable condition is implementation dependent.

The primary mechanism for informing the link partner of a detected error is the Packet Not Accepted control symbol. Devices may optionally support a configuration in which they transmit Packet Not Accepted control symbols that contain an ackID_status, and support resumption of packet transmission using the next expected ackID found in received Packet Not Accepted control symbols. Packet Not Accepted control symbols that contain the ackID_status shall be interpreted as acknowledging all ackIDs up to, but not including, the ackID_status value. The configuration shall be controlled by the Port n Latency Optimization CSRs.

6.13.2.1 Recoverable Errors

The following five basic types of errors are detected by a LP-Serial port:

- An idle sequence error
- A control symbol error
- A packet error
- A column padding error
- A timeout waiting for an acknowledgment or link-response control symbol

6.13.2.2 Idle Sequence Errors

The detectable idle sequence errors depend on the idle sequence being used on the link. Links operating with Control Symbol 24 use the IDLE1 sequence. Links operating with Control Symbol 48 use the IDLE2 sequence. Links operating with Control Symbol 64 use the IDLE3 sequence.

To limit input port complexity, the port is not required to determine the specific error that resulted in an idle sequence error.

6.13.2.2.1 IDLE1 Sequence Errors

The IDLE1 sequence is comprised of A, K, and R (8b/10b special) characters. If an input port detects an invalid character or any valid character other then A, K, or R in an IDLE1 sequence and the port is not in the Input Error-stopped state, the port shall immediately enter the Input Error-stopped state and follow the Input Error-stopped recovery process specified in Section 6.13.2.6, "Input Error-Stopped Recovery Process".

Following are several examples of idle sequence errors.

- A single bit transmission error can change an /A/, /K/, or /R/ code-group into a /Dx.y/ (data) code-group which is illegal in an idle sequence.
- A single bit transmission error can change an /A/, /K/, or /R/ code-group into an invalid code-group.
- A single bit transmission error can change an /SP/ or /PD/ (control symbol delimiters) into an invalid code-group.

6.13.2.2.2 IDLE2 Sequence Errors

The IDLE2 sequence is comprised of A, K, M and R special characters and data characters. If an input port detects any of the following errors in an IDLE2 sequence and the port is not in the Input Error-stopped state, the port shall immediately enter the Input Error-stopped state and follow the Input Error-stopped recovery process specified in Section 6.13.2.6, "Input Error-Stopped Recovery Process".

- An invalid character or any special character other than A, K, M or R
- After lane alignment is achieved,

  a column that contains an A, but is not all As,

  a column that contains a K, but is not all Ks,

  a column that contains a M, but is not all Ms,

  a column that contains a R, but is not all Rs or

  a column that contains a data character, but is not all data characters.

6.13.2.2.3 IDLE3 Sequence Errors

The IDLE3 sequence is comprised of control codewords and data codewords. If an input port detects any of the following errors in an IDLE3 sequence when receive_enable is asserted and the port is not in the Input Error-stopped state, the port shall immediately enter the Input Error-stopped state and follow the Input Error-stopped recovery process specified in Section 6.13.2.6, "Input Error-Stopped Recovery Process".

- An invalid codeword
- After lane alignment is achieved,

  a column that contains a control codeword, but is not all control codewords,

  a column that contains a control codeword, but is not all the same control codeword type,

  a column that contains a data codeword, but is not all data codewords.

  Incomplete or otherwise corrupted Ordered Sequences, see Table 5-10 for some possible scenarios

6.13.2.3 Control Symbol Errors

There are two types of detectable control symbol errors

- An uncorrupted control symbol that violates the link protocol
- A corrupted control symbol

6.13.2.3.1 Link Protocol Violations

The reception of a control symbol with no detected corruption that violates the link protocol shall cause the receiving port to immediately enter the appropriate Error-stopped state. Stype1 control symbol protocol errors shall cause the receiving port to immediately enter the Input Error-stopped state if not already in the Input Error-stopped state and follow the Input Error-stopped recovery process specified in Section 6.13.2.6, "Input Error-Stopped Recovery Process". Stype0 control symbol protocol errors shall cause the receiving port to immediately enter the Output Error-stopped state if not already in the Output Error-stopped state and follow the Output Error-stopped recovery process specified in Section 6.13.2.7, "Output Error-Stopped Recovery Process". If both stype0 and stype1 control symbols contain protocol errors, then the receiving port shall enter both Error-stopped states and follow both error recovery processes.

Link protocol violations include the following:

- Unexpected packet-accepted, packet-retry, or packet-not-accepted control symbol
- Packet acknowledgment control symbol with an unexpected packet_ackID value
- Link timeout while waiting for an acknowledgment or link-response control symbol
- Receipt of a packet-retry symbol when operating in multi-VC mode
- Receipt of an unsolicited Timestamp Control Symbol when the device is a timestamp Master
- Receipt of a sequence of Timestamp Control Symbols that do not conform to the sequences described in Table 6-3, Table 6-4 or Table 6-5.

The following does not constitute a protocol violation:

- Receipt of a VC_status symbol when operating in single VC mode. Unexpected VC_status symbols are discarded.

The following is an example of a link protocol violation and recovery when the Multiple Acknowledges Enabled bit in the Port n Latency Optimization CSRs is deasserted. A sender transmits RT mode packets labeled ackID 2, 3, 4, and 5. It receives acknowledgments for packets 2, 4, and 5, indicating a probable error associated with ackID 3. The sender then stops transmitting new packets and sends a link-request/port-status (restart-from-error) control symbol to the receiver. The receiver then returns a link- response control symbol indicating which packets it has received properly. These are the possible responses and the sender’s resulting behavior:

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

6.13.2.3.2 Corrupted Control symbols

The reception of a control symbol with detected corruption shall cause the receiving port to immediately enter the Input Error-stopped state if not already in the Input Error-stopped state and follow the Input Error-stopped recovery process specified in Section 6.13.2.6, "Input Error-Stopped Recovery Process".

Input ports detect the following types of control symbol corruption:

- A Control Symbol 24 or Control Symbol 48 containing invalid characters or valid but non-data characters
- A Control Symbol 64 containing invalid codewords
- A control symbol with an incorrect CRC value
- A Control Symbol 24 or Control Symbol 48 whose start delimiter (SC or PD) occurs in a lane whose lane_number mod4 != 0
- A Control Symbol 48 that does not have a end delimiter in the seventh character position after its start delimiter and with the same value as the start delimiter
- A Control Symbol 64 control codeword out of sequence or incomplete sequence of Control Symbol 64 control codewords

6.13.2.4 Packet Errors

Each packet received by a port shall be checked for the following types of errors:

- Packet with an unexpected ackID value
- Packet with an incorrect CRC value
- Packet containing invalid characters or valid non-data characters
- Packet that exceeds the maximum packet size.

With one exception, the reception of a packet with any of the above errors shall cause the receiving port to immediately enter the Input Error-stopped state if not already in the Input Error-stopped state and follow the Input Error-stopped recovery process specified in Section 6.13.2.6, "Input Error-Stopped Recovery Process". The exception occurs when the link to which the port is connected is operating with the IDLE2 or IDLE3 idle sequence, the packet in which one or more errors were detected was canceled by a link-request control symbol, and the only errors detected in the packet were the presence of one or more M special characters for IDLE2 or one or more Descrambler Seed control codewords and may cause excessive packet length. In this case, the errors detected in the packet shall be ignored and the packet handled as a canceled packet as specified in Section 6.10, "Canceling Packets".

6.13.2.5 Link Timeout

A link timeout while waiting for an acknowledgment or link-response control symbol is handled as a link protocol violation as described in Section 6.13.2.3.1, "Link Protocol Violations".

6.13.2.6 Input Error-Stopped Recovery Process

When the input side of a port detects a transmission error, it immediately enters the Input Error-stopped state. To recover from this state, the input side of the port takes the following actions:

- Record the condition(s) that caused the port to enter the Input Error-stopped state.
- If an error(s) was detected in a control symbol or packet, ignore and discard the corrupted control symbol or packet.
- Cause the output side of the port to issue a packet-not-accepted control symbol. (The packet-not-accepted control symbol causes the output side of the receiving port to enter the Output Error-stopped state and send a link-request/port-status control symbol.) If the packet-not-accepted control symbol contains the next expected ackID and transmitter based flow control is in use on the link, cause the output side of the port to transmit Status and VC_Status control symbols for all active VCs. Transmission of Status and VC_Status control symbols is optional when receiver based flow control is in use, and when the packet-not-accepted control symbol does not contain the next expected ackID. Transmission of the Status and VC_Status control symbols is subject to the following:
  - The packet-not-accepted control symbol shall be transmitted either before or after all of the status and VC-status control symbols are transmitted.
  - The status and VC-status control symbols that are transmitted shall be transmitted in the following order. If a status control symbol is transmitted it shall be transmitted first before any of the VC-status control symbols. Any VC-status control symbols that are transmitted shall be transmitted after the status control symbol and in order of increasing VCID.

    The buffer status found in the status and VC-status control symbols shall reflect buffer status up to, but not including, the packet associated with the ackID in the packet-not-accepted control symbol.
- Subsequent to the event that caused the port to enter the Input Error-stopped state and prior to the reception of a link-request/port-status control symbol,
  - discard without acknowledgement or error report all packets that are received for VCs operating in RT mode,

  - accept without acknowledgement (accept silently) all error free packets that are received for VCs operating in CT mode for which the VC specified in the packet has buffer space available and
  - discard without acknowledgement all packets that are received for VCs operating in CT mode which are not error free or for which the VC specified in the packet does not have buffer space available.
- When a link-request/port-status control symbol is received from the connected port, cause the output side of the port to transmit a link-response control symbol and if the transmitter-controlled flow control is in use on the link, to also transmit a VC_Status control symbol for each of VC1-8 that is active. The transmission of a VC_Status control symbol for each of VC1-8 that is active is optional if receiver-controlled flow control in use on the link. The input side of the port should also cause the output side of the port to transmit a status control symbol (for VC0). The input side of the port then exits the Input Error-stopped state and resumes normal packet reception. The actual transmission of the link-response, VC-status, and status control symbols may occur after the input side of the port exits the Input Error-stopped state and resumes normal packet reception.
- The transmission of the link-response, status and VC-status control symbols is subject to the following requirements.
  - The link-response control symbol shall be transmitted either before any of the status and VC-status control symbols are transmitted or after all of the status and VC-status control symbols are transmitted.
  - The status and VC-status control symbols that are transmitted shall be transmitted in the following order. If a status control symbol is transmitted it shall be transmitted first before any of the VC-status control symbols. Any VC-status control symbols that are transmitted shall be transmitted after the status control symbol and in order of increasing VCID.
  - The link-response control symbol shall not be transmitted until the input side of the port is ready to resume packet reception and either the buffer consumption of all packets received by the port before the link-request/port-status control symbol has been determined or the port can maintain the distinction after packet reception resumes between packets received before the reception of the link-request/port-status control symbol and packets received after the reception of the link-request/port-status control symbol (as the processing of packets received before the link-request/port-status control symbol differs from the processing of packets received after the link-request/port-status control symbol).
  - The status or VC-status control symbol for a VC operating in RT mode shall indicate the number of receive buffers available for that VC inclusive of the buffer consumption of all packets received and accepted by the port for that VC before the event that caused the port to enter the Input Error-stopped state.
  - The VC-status control symbol for a VC operating in CT mode shall indicate the number of receive buffers available for that VC inclusive of the buffer

consumption of all packets received and accepted by the port for that VC before the link-request/port-status control symbol was received.

– The status and VC-status control symbols shall be transmitted before any packet acknowledgment control symbols are transmitted for packets received after the link-request/port-status control symbol was received.

An example state machine with the behavior described in this section is included in Section C.3, "Error Recovery".

6.13.2.7 Output Error-Stopped Recovery Process

To recover from the Output Error-stopped state, the output side of a port takes the following actions.

- Immediately stops transmitting new packets.
- Resets the link packet acknowledgment timers for all transmitted but unacknowledged packets. (This prevents the generation of spurious timeout errors.)
- Transmits a link-request/port-status (restart-from-error) control symbol. (The link-request/port-status control symbol causes the connected port to transmit a link-response control symbol that contains the port_status and ackID_status of the input side of the port. The ackID_status field contains the ackID value that is expected in the next packet that the connected port receives.)
- If the optional ability to perform error recovery with the ackID in the packet-not-accepted control symbol is enabled, and receipt of a Packet Not Accepted control symbol was the cause of entering the Output Error-Stopped state, then the port exits the output error-stopped state. VCs operating in RT mode back up to the first unaccepted packet in each VC. VCs operating in CT mode silently assume the unacknowledged packets were accepted and adjust their state accordingly.
- If the optional ability to perform error recovery with the ackID in the packet-not-accepted control symbol is disabled, or the port entered the Output Error-Stopped state for a reason other than receipt of a packet not accepted control symbol, the port waits until the link-response is received, VCs operating in RT mode back up to the first unaccepted packet in each VC. VCs operating in CT mode silently assume the unacknowledged packets were accepted and adjust their state accordingly.
- The port exits the Output Error-stopped state and resumes transmission with the next RT or CT packet according to the bandwidth allocation algorithm using the ackID value contained in the link-response control symbol.
- If the ability to perform error recovery using the ackID in the packet-not-accepted control symbol is enabled, and receipt of a Packet Not Accepted control symbol was the cause of previously entering the Output Error-Stopped state, then receipt of a link-response shall complete the

outstanding link-request/port-status control symbol, allowing another link-request/port-status control symbol to be transmitted. The contents of the link-response control symbol shall be treated as informational in this case.

An example state machine with the behavior described in this section is included in Section C.3, "Error Recovery".

6.14 Power Management

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

Chapter 7 LP-Serial Registers

7.1 Introduction

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 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.

There are four types of LP-Serial devices, an endpoint device, an endpoint device with additional software recovery registers, an endpoint free (or switch) device, and an endpoint free device with additional software recovery registers. Each has a different set of CSRs, specified in Section 7.5.1, Section 7.5.2, Section 7.5.3, and Section 7.5.4, respectively. All four device types have the same CARs, specified in Section 7.4.

Devices supporting Virtual Channels contain an additional register block for configuring VC support for each port. That block is added on after the above register block using a separate EF_PTR, as described in Section 7.8.

7.2 Register Map

These registers utilize the Extended Features blocks and can be accessed using *RapidIO Part 1: 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 LP-Serial physical features block may exist in any position in the Extended Features data structure and may exist in any portion of the Extended Features Space in the register address map for the 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 7-1. LP-Serial Register Map

Configuration Space Byte Offset	Register Name
0x0-F	Reserved
0x10-13	Processing Element Features CAR
0x14-0xFF	Reserved
0x100–FFFF	Extended Features Space
0x10000–FFFFFF	Implementation-defined Space

7.3 Reserved Register, Bit and Bit Field Value Behavior

Table 7-2 describes the required behavior for accesses to reserved register bits and reserved registers for the RapidIO register space,

Table 7-2. Configuration Space Reserved Access Behavior

Byte Offset	Space Name	Item	Initiator behavior	Target behavior
0x0–3F	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–FF	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 7-2. Configuration Space Reserved Access Behavior (Continued)

Byte Offset	Space Name	Item	Initiator behavior	Target behavior
0x100–FFFF	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–FFFFFF	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.

When a writable bit field is set to a reserved value, device behavior is implementation specific.

7.4 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 7-2 for the required behavior for accesses to reserved registers and register bits.

CARs are big-endian with bit 0 the most significant bit.

7.4.1 Processing Element Features CAR (Configuration Space Offset 0x10)

The processing element features CAR identifies the major functionality provided by the processing element. The bit settings are shown in Table 7-3.

Table 7-3. Bit Settings for Processing Element Features CAR

Bits	Name	Description
0–3	—	Reserved
4	Multiport	This bit shall be implemented by devices that support Baud Rate Class 2 or Baud Rate Class 3, but is optional for devices that do not support those baud rate classes. If this bit is not implemented it is Reserved. If this bit is implemented, the Switch Port Information CAR at Configuration Space Offset 0x14 (see *RapidIO Part 1: I/O Logical Specification*) shall be implemented regardless of the state of bit 3 of the Processing Element Features CAR. Indicates whether the PE implements multiple external RapidIO ports 0b0 - PE does not implement multiple external RapidIO ports 0b1 - PE implements multiple external RapidIO ports
5–24	—	Reserved
25	Implementation-defined	Implementation-defined
26	CRF Support	PE supports the Critical Request Flow (CRF) indicator 0b0 - Critical Request Flow is not supported 0b1 - Critical Request Flow is supported
27–31	—	Reserved

7.5 LP-Serial Extended Feature Blocks

This section describes the LP-Serial Extended Features Blocks. There is a separate Extended Features block for each of the following types of processing elements.

Generic endpoint devices

Generic endpoint devices, software assisted error recovery option

Generic endpoint free devices

Generic endpoint free devices, software assisted error recovery option

All registers in the LP-Serial Extended Feature Blocks are 32 bits in length and aligned to 32 bit boundaries. The details of the registers used are specified in Section 7.6.

7.5.1 Generic Endpoint Devices

This section specifies the LP-Serial registers comprising the LP-Serial Extended Features Block for a generic endpoint device. This Extended Features register block is assigned Extended Features block ID=0x0001 for devices using Register Map - I, and Extended Features block ID=0x0011 for devices using Register Map - II.

Table 7-4 and Table 7-5 show the register maps of the RapidIO LP-Serial Extended Features Block for different device classes. The register maps specify the registers that comprise these Extended Features Blocks.

7.5.2 Generic Endpoint Devices, Software-assisted Error Recovery Option

This section specifies the LP-Serial registers comprising the LP-Serial Extended Features Block for a generic endpoint 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 and for device debug. This Extended Features register block is assigned Extended Features block ID=0x0002 for devices using Register Map - I, and Extended Features block ID=0x0012 for devices using Register Map - II.

Table 7-4 and Table 7-5 show the register maps of the RapidIO LP-Serial Extended Features Block for different device classes. The register maps specify the registers that comprise these Extended Features Blocks.

7.5.3 Generic Endpoint Free Devices

This section specifies the LP-Serial registers comprising the LP-Serial Extended Features Block for a generic device that does not contain endpoint functionality (i.e.

switches). This Extended Features register block uses extended features block ID=0x0003 for devices using Register Map - I, and Extended Features block ID=0x0013 for devices using Register Map - II.

Table 7-4 and Table 7-5 show the register maps of the RapidIO LP-Serial Extended Features Block for different device classes. The register maps specify the registers that comprise these Extended Features Blocks.

7.5.4 Generic Endpoint Free Devices, Software-assisted Error Recovery Option

This section specifies the LP-Serial registers comprising the LP-Serial Extended Features Block for a generic device that does not contain endpoint functionality but that does support software assisted error recovery. Typically these devices are switches. 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 and for device debug. This Extended Features register block is assigned Extended Features block ID=0x0009 for devices using Register Map - I, and Extended Features block ID=0x0019 for devices using Register Map - II.

Table 7-4 and Table 7-5 show the register maps of the RapidIO LP-Serial Extended Features Block for different device classes. The register maps specify the registers that comprise these Extended Features Blocks.

7.5.5 Register Map - I

Table 7-4 defines the register maps that may be used for devices which only support IDLE1 or IDLE2 operation. An “X” in a column indicates that the register shall be implemented for the indicated Extended Features Block ID. An “O” in a column indicates that the register may optionally be implemented for the indicated Extended Features Block ID.

The Extended Features Block IDs 0x0001, 0x0002, 0x0003, and 0x0009 are deprecated in favor of Register Map - II defined for IDLE3 devices defined in “Section 7.5.6, Register Map - II” on page 324.

The structure and use of the individual registers comprising Register Map - I is specified in Section 7.6.

The required behavior for accesses to reserved registers and register bits is specified in Table 7-2.

The Block Byte Offset is the offset relative to the 16-bit Extended Features Pointer (EF_PTR) that points to the beginning of the block. The address of a byte in the block is calculated by adding the block byte offset to EF_PTR that points to the beginning of the block. This is denoted as [EF_PTR+xx], where xx is the block byte offset in hexadecimal.

This register map is currently only defined for devices with up to 16 RapidIO ports, but can be extended or shortened as 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 + 0xBC]. Register map offset [EF_PTR + 0xC0] can be used for another Extended Features block.

Table 7-4. LP-Serial Register Map - I

	Block Byte Offset	Register Name	Extended Features Block ID			
			Endpoint		Endpoint Free	
			0x0001	0x0002	0x0003	0x0009
General	0x0	LP-Serial Register Block Header	X	X	X	X
	0x4–1C	Reserved	-			
	0x20	Port Link Timeout Control CSR	X	X	X	X
	0x24	Port Response Timeout Control CSR	X	X	-	-
	0x28–38	Reserved	-			
	0x3C	Port General Control CSR	X	X	X	X
Port 0	0x40	Port 0 Link Maintenance Request CSR	-	X	-	X
	0x44	Port 0 Link Maintenance Response CSR	-	X	-	X
	0x48	Port 0 Local ackID Status CSR	-	X	-	X
	0x4C	Port 0 Initialization Status CSR	O	O	O	O
	0x50	Reserved	-			
	0x54	Port 0 Control 2 CSR	X	X	X	X
	0x58	Port 0 Error and Status CSR	X	X	X	X
	0x5C	Port 0 Control CSR	X	X	X	X
Port 1	0x60	Port 1 Link Maintenance Request CSR	-	X	-	X
	0x64	Port 1 Link Maintenance Response CSR	-	X	-	X
	0x68	Port 1 Local ackID Status CSR	-	X	-	X
	0x6C	Port 1 Initialization Status CSR	O	O	O	O
	0x70	Reserved	-			
	0x74	Port 1 Control 2 CSR	X	X	X	X
	0x78	Port 1 Error and Status CSR	X	X	X	X
	0x7C	Port 1 Control CSR	X	X	X	X
Ports 2-14	0x80–218	Assigned to Port 2-14 CSRs				

Table 7-4. LP-Serial Register Map (Continued)- I

	Block Byte Offset	Register Name	Extended Features Block ID			
			Endpoint		Endpoint Free	
			0x0001	0x0002	0x0003	0x0009
Port 15	0x220	Port 15 Link Maintenance Request CSR	-	X	-	X
	0x224	Port 15 Link Maintenance Response CSR	-	X	-	X
	0x228	Port 15 Local ackID Status CSR	-	X	-	X
	0x22C	Port 15 Initialization Status CSR	O	O	O	O
	0x230	Reserved	-			
	0x234	Port 15 Control 2 CSR	X	X	X	X
	0x238	Port 15 Error and Status CSR	X	X	X	X
	0x23C	Port 15 Control CSR	X	X	X	X

7.5.6 Register Map - II

Table 7-5 defines the register maps that shall be used for devices which support IDLE3 operation. These register maps may be used for devices which only support IDLE1 and/or IDLE2 operation. The four right-most columns indicate register implementation requirements for the Extended Features Block IDs. An “X” in a column indicates that the register shall be implemented for the indicated Extended Features Block ID. An “O” in a column indicates that the register may optionally be implemented for the indicated Extended Features Block ID.

The structure and use of the individual registers comprising Register Map - II is specified in Section 7.6.

The required behavior for accesses to reserved registers and register bits is specified in Table 7-2.

The Block Byte Offset is the offset relative to the 16-bit Extended Features Pointer (EF_PTR) that points to the beginning of the block. The address of a byte in the block is calculated by adding the block byte offset to EF_PTR that points to the beginning of the block. This is denoted as [EF_PTR+xx], where xx is the block byte offset in hexadecimal.

This register map is currently only defined for devices with up to 16 RapidIO ports, but can be extended or shortened as 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 + 0x13C]. Register map offset [EF_PTR + 0x140] can be used for another Extended Features block.

Table 7-5. LP-Serial Register Map - II

	Block Byte Offset	Register Name	Extended Features Header ID			
			Endpoint		Endpoint Free	
			0x0011	0x0012	0x0013	0x0019
General	0x0	LP-Serial Register Block Header	X	X	X	X
	0x4–1C	Reserved	-			
	0x20	Port Link Timeout Control CSR	X	X	X	X
	0x24	Port Response Timeout Control CSR	X	X	-	-
	0x28–38	Reserved	-			
	0x3C	Port General Control CSR	X	X	X	X
Port 0	0x40	Port 0 Link Maintenance Request CSR	-	X	-	X
	0x44	Port 0 Link Maintenance Response CSR	-	X	-	X
	0x48	Reserved				
	0x4C	Port 0 Initialization Status CSR	O	O	O	O
	0x50	Reserved	-			
	0x54	Port 0 Control 2 CSR	X	X	X	X
	0x58	Port 0 Error and Status CSR	X	X	X	X
	0x5C	Port 0 Control CSR	X	X	X	X
	0x60	Port 0 Outbound ackID CSR	-	X	-	X
	0x64	Port 0 Inbound ackID CSR	-	X	-	X
	0x68	Port 0 Power Management CSR[1]	O	O	O	O
	0x6C	Port 0 Latency Optimization CSR	X	X	X	X
	0x70	Port 0 Link Timers Control CSR	X	X	X	X
	0x74	Port 0 Link Timers Control 2 CSR	X	X	X	X
	0x78	Port 0 Link Timers Control 3 CSR[1]	X	X	X	X
	0x7C	Reserved	-			

Table 7-5. LP-Serial Register Map - II

	Block Byte Offset	Register Name	Extended Features Header ID			
			Endpoint		Endpoint Free	
			0x0011	0x0012	0x0013	0x0019
Port 1	0x80	Port 1 Link Maintenance Request CSR	-	X	-	X
	0x84	Port 1 Link Maintenance Response CSR	-	X	-	X
	0x88	Reserved	-			
	0x8C	Port 1 Initialization Status CSR	O	O	O	O
	0x90	Reserved	-			
	0x94	Port 1 Control 2 CSR	X	X	X	X
	0x98	Port 1 Error and Status CSR	X	X	X	X
	0x9C	Port 1 Control CSR	X	X	X	X
	0xA0	Port 1 Outbound ackID CSR	-	X	-	X
	0xA4	Port 1 Inbound ackID CSR	-	X	-	X
	0xA8	Port 1 Power Management CSR[1]	O	O	O	O
	0xAC	Port 1 Latency Optimization CSR	X	X	X	X
	0xB0	Port 1 Link Timers Control CSR	X	X	X	X
	0xB4	Port 1 Link Timers Control 2 CSR	X	X	X	X
	0xB8	Port 1 Link Timers Control 3 CSR[1]	X	X	X	X
	0xBC	Reserved	-			
Ports 2-14	0xC0–3FC	Assigned to Port 2-14 CSRs				

Table 7-5. LP-Serial Register Map - II

	Block Byte Offset	Register Name	Extended Features Header ID			
			Endpoint		Endpoint Free	
			0x0011	0x0012	0x0013	0x0019
Port 15	0x400	Port 15 Link Maintenance Request CSR	-	X	-	X
	0x404	Port 15 Link Maintenance Response CSR	-	X	-	X
	0x408	Reserved	-			
	0x40C	Port 15 Initialization Status CSR	O	O	O	O
	0x410	Reserved	-			
	0x414	Port 15 Control 2 CSR	X	X	X	X
	0x418	Port 15 Error and Status CSR	X	X	X	X
	0x41C	Port 15 Control CSR	X	X	X	X
	0x420	Port 15 Outbound ackID CSR	-	X	-	X
	0x424	Port 15 Inbound ackID CSR	-	X	-	X
	0x428	Port 15 Power Management CSR[1]	O	O	O	O
	0x42C	Port 15 Latency Optimization CSR	X	X	X	X
	0x430	Port 15 Link Timers Control CSR	X	X	X	X
	0x434	Port 15 Link Timers Control 2 CSR	X	X	X	X
	0x438	Port 15 Link Timers Control 3 CSR[1]	X	X	X	X
	0x43C	Reserved	-			

[1]These registers are reserved and not in use for devices that only support IDLE1 and/or IDLE2 operation.

7.6 LP-Serial Command and Status Registers (CSRs)

All Command and Status registers are 32 bits in length and are aligned to 32 bit boundaries. All CSRs are accessed as 4 byte entities. CSRs are big endian with bit 0 the most significant bit.

Refer to Table 7-2 for the required behavior for accesses to reserved register bits.

For each register the Block Offset is listed, some registers have a different offset or spacing between offsets. Block Offsets that are applicable for Register Map - I are prefixed with “RM-I”, and Block Offsets that are applicable for Register Map - II are prefixed with “RM-II”. If no prefix is used the Block Offset is the same for all Baud Rate Classes.

7.6.1 LP-Serial Register Block Header (Block Offset 0x0)

The LP-Serial register block header register contains the EF_PTR to the next extended features block and the EF_ID that identifies LP-Serial Extended Feature Block for which this is the register block header.

Table 7-6. Bit Settings for LP-Serial Register Block Header

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

7.6.2 Port Link Timeout Control CSR (Block Offset 0x20)

The port link timeout control register contains the timeout timer value for all ports on a device. This timeout 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 timeout interval, and represents between 3 and 6 seconds.

Table 7-7. Bit Settings for Port Link Timeout Control CSR

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

7.6.3 Port Response Timeout Control CSR (Block Offset 0x24)

The port response timeout control register contains the timeout timer count for all ports on a device. This timeout is for sending a request packet to receiving the corresponding response packet. The reset value is the maximum timeout interval, and represents between 3 and 6 seconds.

Table 7-8. Bit Settings for Port Response Timeout Control CSR

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

7.6.4 Port General Control CSR (Block Offset 0x3C)

The bits accessible through the Port General Control CSR are bits that apply to all ports in a device. There is a single copy of each such bit per device. These bits are also accessible through the Port General Control CSR of any other Physical Layers implemented on a device.

The structure and bit definitions of the Port General Control CSR depend on whether or not the device contains an endpoint. The register bit definitions for a generic endpoint device with or without the software assisted error recovery option are specified in Table 7-9.

Table 7-9. Bit Settings for Port General Control CSR, Generic Endpoint Devices

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

The register bit definitions for a generic endpoint free device with or without the software assisted error recovery option are specified in Table 7-10.

Table 7-10. Bit Settings for General Port Control CSR, Generic Endpoint Free Device

Bit	Name	Reset Value	Description
0-1	—		Reserved
2	Discovered	see footnote[1]	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 Discovered reset value is implementation dependent

7.6.5 Port *n* Link Maintenance Request CSRs (RM-I Block Offsets 0x40, 60, ... , 220) (RM-II Block Offsets 0x40, 80, ... , 400)

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 7-11. Bit Settings for Port *n* Link Maintenance Request CSRs

Bit	Name	Reset Value	Description
0–28	—		Reserved
29-31	Command	0b000	This field controls the contents of the cmd field sent in the link-request control symbol for Control Symbol 24 and Control Symbol 48. This field controls the least significant 3 bits of the link-request Stype1 field for Control Symbol 64. If read this field returns the last written value.

7.6.6 Port *n* Link Maintenance Response CSRs (RM-I Block Offsets 0x44, 64, ... , 224) (RM-II Block Offsets 0x44, 84, ... , 404)

The port link maintenance response registers are accessible both by a local processor and an external device. A read to this register returns the status received in a link-response control symbol. The ackID_status, port_status, and port_status_cs64 fields are defined in Table 3-4, Table 3-14, and Table 3-15, respectively. This register is read-only.

Table 7-12. 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.
1–2	—		Reserved
3-14	port_status_cs64	0x000	Reserved for Control Symbol 24 and Control Symbol 48 Port status field from Control Symbol 64
15-26	ackID_status	0x000	ackID status field from the link-response control symbol. Bits 22 to 26 are valid for Control Symbol 24. Bits 21 to 26 are valid for Control Symbol 48. All bits are valid for Control Symbol 64.
27-31	port_status	0b00000	Reserved for Control Symbol 64 port_status field from the link-response control symbol, Control Symbol 24 and Control Symbol 48.

7.6.7 Port *n* Local ackID CSRs (RM-I Block Offsets 0x48, 68, ... , 228)

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 output and input sides of the ports.

This register is only applicable for Control Symbol 24 and Control Symbol 48. Equivalent functionality is provided for Control Symbol 64 through the Port n Outbound ackID CSRs and the Port n Inbound ackID CSRs

Table 7-13. Bit Settings for Port *n* Local ackID Status CSRs

Bit	Name	Reset Value	Description
0	Clr_outstanding_ackIDs	0b0	Writing 0b1 to this bit causes all outstanding unacknowledged packets to be discarded. This bit should only be written when trying to recover a failed link. This bit is always logic 0 when read.
1	—		Reserved
2-7	Inbound_ackID	0b000000	Input port next expected ackID value. Bit 2 is only valid for Control Symbol 48.
8-17	—		Reserved
18-23	Outstanding_ackID	0b000000	Output port unacknowledged ackID status. Next expected acknowledge control symbol ackID field that indicates the ackID value expected in the next received acknowledge control symbol. Bit 18 is only valid for Control Symbol 48.
24-25	—		Reserved
26-31	Outbound_ackID	0b000000	Output port next transmitted ackID value. Software writing this value can force retransmission of outstanding unacknowledged packets in order to manually implement error recovery. Bit 26 is only valid for Control Symbol 48.

7.6.8 Port *n* Initialization Status CSRs (RM-I Block Offsets 0x4C, 6C, ... , 22C) (RM-II Block Offsets 0x4C, 8C, ... , 40C)

The Port *n* Initialization Status CSRs are used to inform software of the progress of various state machines for all baud rates. Implementation of these registers shall be optional. All bits and bit fields in these register shall be as defined in Table 7-14. Unless otherwise specified, the bits and bit fields of these registers shall be read only.

Table 7-14. Bit Settings for Port *n* Initialization Status CSRs

Bit	Name	Reset Value	Description
0-4	Lane Alignment	Impl. Spec.	State of the lane alignment state machine. This field shall be 0 if the port can only operate in 1x mode. The encoding of this value is implementation specific.
5	—		Reserved
6-9	1x/2x Mode Detection	Impl. Spec.	State of the 1x/2x Mode Detection state machine. This field shall be 0 if the port can only operate in the 1x mode. The encoding of this value is implementation specific.
10	—		Reserved
11-15	Port Initialization State Machine	Impl. Spec.	State of the 1x/2x/4x/8x/16x port initialization state machine. This field shall be 0 if the port can only operate in 1x mode. The encoding of this value is implementation specific.
16-19	Received status control symbols	0b0000	Count of the number of consecutive error-free Status control symbols received. This counter shall not increment once the “link initialized” state has been achieved.
20	—		Reserved
21-27	Transmitted status control symbols	0b0000000	Count of the number of Status control symbols transmitted. This counter shall continue to increment until the “link initialized” state has been achieved. If the port can determine the status of the link partner through the contents of the IDLE2 or IDLE3 sequence, this counter shall also continue to increment until the link partner indicates that it has achieved the “port initialized” state.
28-31	—		Reserved

7.6.9 Port *n* Control 2 CSRs (RM-I Block Offset 0x54, 74, ... , 234) (RM-II Block Offset 0x54, 94, ... , 414)

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

Table 7-15. Bit Settings for Port *n* Control 2 CSRs

Bit	Name	Reset Value	Description
0-3	Selected Baudrate	0b0000	Indicates the baudrate at which the initialized port is operating. (read only) 0b0000 - no rate selected 0b0001 - 1.25 Gbaud 0b0010 - 2.5 Gbaud 0b0011 - 3.125 Gbaud 0b0100 - 5.0 Gbaud 0b0101 - 6.25 Gbaud 0b0110 - 10.3125 Gbaud 0b0111 - 12.5 Gbaud 0b0111–0b1111 - Reserved
4	Baudrate Discovery Support	see footnote[1]	Indicates whether automatic baudrate discovery is supported (read-only) 0b0 - Automatic baudrate discovery not supported 0b1 - Automatic baudrate discovery supported
5	Baudrate Discovery Enable	see footnote[2]	Controls whether automatic baudrate discovery is enabled 0b0 - Automatic baudrate discovery disabled 0b1 - Automatic baudrate discovery enable The port shall not allow this bit to be set unless it supports baudrate discovery.
6	1.25 Gbaud Support	see footnote[1]	Indicates whether port operation at 1.25 Gbaud is supported (read only) 0b0 - 1.25 Gbaud operation not supported 0b1 - 1.25 Gbaud operation supported
7	1.25 Gbaud Enable	see footnote[2]	Controls whether port operation at 1.25 Gbaud is enabled 0b0 - 1.25 Gbaud operation disabled 0b1 - 1.25 Gbaud operation enabled The port shall not allow this bit to be set unless it supports 1.25 Gbaud.
8	2.5 Gbaud Support	see footnote[1]	Indicates whether port operation at 2.5 Gbaud is supported (read only) 0b0 - 2.5 Gbaud operation not supported 0b1 - 2.5 Gbaud operation supported
9	2.5 Gbaud Enable	see footnote[2]	Controls whether port operation at 2.5 Gbaud is enabled 0b0 - 2.5 Gbaud operation disabled 0b1 - 2.5 Gbaud operation enabled The port shall not allow this bit to be set unless it supports 2.5 Gbaud.
10	3.125 Gbaud Support	see footnote[1]	Indicates whether port operation at 3.125 Gbaud is supported (read only) 0b0 - 3.125 Gbaud operation not supported 0b1 - 3.125 Gbaud operation supported
11	3.125 Gbaud Enable	see footnote[2]	Controls whether port operation at 3.125 Gbaud is enabled 0b0 - 3.125 Gbaud operation disabled 0b1 - 3.125 Gbaud operation enabled The port shall not allow this bit to be set unless it supports 3.125 Gbaud.

Table 7-15. Bit Settings for Port *n* Control 2 CSRs

Bit	Name	Reset Value	Description
12	5.0 Gbaud Support	see footnote[1]	Indicates whether port operation at 5.0 Gbaud is supported (read only) 0b0 - 5.0 Gbaud operation not supported 0b1 - 5.0 Gbaud operation supported
13	5.0 Gbaud Enable	see footnote[2]	Controls whether port operation at 5.0 Gbaud is enabled 0b0 - 5.0 Gbaud operation disabled 0b1 - 5.0 Gbaud operation enabled The port shall not allow this bit to be set unless it supports 5.0 Gbaud.
14	6.25 Gbaud Support	see footnote[1]	Indicates whether port operation at 6.25 Gbaud is supported (read only) 0b0 - 6.25 Gbaud operation not supported 0b1 - 6.25 Gbaud operation supported
15	6.25 Gbaud Enable	see footnote[2]	Controls whether port operation at 6.25 Gbaud is enabled 0b0 - 6.25 Gbaud operation disabled 0b1 - 6.25 Gbaud operation enabled The port shall not allow this bit to be set unless it supports 6.25 Gbaud
16	10.3125 Gbaud Support	see footnote[1]	Indicates whether port operation at 10.3125 Gbaud is supported (read only) 0b0 - 10.3125 Gbaud operation not supported 0b1 - 10.3125 Gbaud operation supported
17	10.3125 Gbaud Enable	see footnote[2]	Controls whether port operation at 10.3125 Gbaud is enabled 0b0 - 10.3125 Gbaud operation disabled 0b1 - 10.3125 Gbaud operation enabled The port shall not allow this bit to be set unless it supports 10.3125 Gbaud.
18	12.5 Gbaud Support	see footnote[1]	Indicates whether port operation at 12.5 Gbaud is supported (read only) 0b0 - 12.5 Gbaud operation not supported 0b1 - 12.5 Gbaud operation supported
19	12.5 Gbaud Enable	see footnote[2]	Controls whether port operation at 12.5 Gbaud is enabled 0b0 - 12.5 Gbaud operation disabled 0b1 - 12.5 Gbaud operation enabled The port shall not allow this bit to be set unless it supports 12.5 Gbaud.
20-26	—		Reserved
27	10G Retraining Enable	see footnote[2]	Controls the behavior for lane retraining for a port: 0b0 - Lane retraining is disabled 0b1 - Retrain all lanes if any have “degraded” status

Table 7-15. Bit Settings for Port *n* Control 2 CSRs

Bit	Name	Reset Value	Description
28	Enable Inactive Lanes	0b0	Use of the test mode enabled by the implementation of this bit to monitor the behavior of the inactive lanes requires that this bit must be set in both ports and that all link width modes wider than the desired Mx mode must be disabled in the Port n Control CSR of both ports. Failure to meet these requirements will result in unspecified link behavior. (Modes wider than the desired Mx mode must be disabled so that the Initialization state machine ignores the asserted lane_sync signals from the lanes with forced output enables and does not attempt to enter a mode wider than Mx). When a 1x/Nx or 1x/Mx/Nx port is operating in 1x mode where 1<M<N and N = 4, 8 or 16, lanes 0 and 2 are the active lanes and lane 1 and lanes 3 through N-1 are the inactive lanes. Note that, to be consistent with the previous paragraph, a 1x/2x/Nx port operating in 1x mode must have 2x mode disabled. When a 1x/Mx/Nx port is operating in Mx mode where 1<M<N and N = 4, 8 or 16, lanes 0 through M-1 are the active lanes and lanes M through N-1 are the inactive lanes. The test mode enabled by the implementation of this bit only allows the testing of the inactive lanes that are supported by both of the connected ports. For example, if a 1x/4x/8x port is connected to a 1x/4x/16x port and the link is operating in 4x mode, only lanes 4 though 7 can be monitored using this test mode. The implementation of this bit is optional. When implemented, this bit allows software to force the lanes of the port that are not currently being used to carry traffic, the “inactive lanes”, to be enabled for testing while the “active lanes” continue to carry traffic. If this bit is not implemented it is reserved. If implemented, this bit shall not be asserted when the port is connected to a link that includes retimers as defined in Section 4.11.1, "Retimers". 0b0: The output enables of all of the lanes controlled by the port are controlled solely by the port’s Initialization state machine 0b1: The port’s receivers for the inactive lanes are enabled. The port’s drivers for the inactive lanes are output enabled if and only if the port’s Initialization state machine is not in the SILENT or SEEK state. A continuous IDLE sequence of the same type as is in use on the active lanes shall be transmitted on the inactive lanes when their transmitters are output enabled. The IDLE sequences transmitted on the inactive lanes shall comply with all rules for that type of IDLE sequence including alignment across the inactive lanes, but they are not required to use the same bit sequences or be aligned in any way relative to the IDLE sequences transmitted on the active lanes. If IDLE2 is being used on the active lanes of the port, the inactive lanes of the port shall report their lane number and port width in the CS Field Marker and handle commands carried in the CS Field as if they were active lanes. If IDLE3 is being used on the active lanes of the port, the inactive lanes of the port shall report their lane number and port width in the Status/Control Ordered Sequence and handle commands carried in the Status/Control Ordered Sequence as if they were active lanes.

Table 7-15. Bit Settings for Port *n* Control 2 CSRs

Bit	Name	Reset Value	Description
29	Data scrambling disable	0b0	Read-write 0b0: transmit scrambler and receive descrambler are enabled. 0b1: The transmit scrambler and receive descrambler are disabled for control symbol and packet data characters. Control symbol and packet data characters are neither scrambled in the transmitter before transmission nor descrambled in the receiver upon reception. The transmit scrambler remains enabled for the generation of pseudo-random data characters for the IDLE2 random data field. This bit is for test use only and shall not be asserted during normal operation. For IDLE3 links, a transition of this bit from 0 to 1 shall cause the transmitter to set the link partner's descrambler seed to 0, and set the local transmitter's scrambler seed to 0, for all lanes. For IDLE3 links, a transition of this bit from 1 to 0 shall cause the transmitter to set the link partner's descrambler seed to a random value for each lane, and set the local transmitter's scrambler seed to match what the link partner was set to. The random value shall not be 0 for any lane. An example set of IDLE3 scrambler values is presented in Table 5-4. It shall be noted that there is a fundamental difference between IDLE2 scrambler disable and IDLE3 scrambler disable,. The disabling of the IDLE2 scrambler need to be configured in both ports of a link and affects both directions together, whereas for IDLE3 the disabling the scrambler is controlled independently in each direction from the transmitters port.
30	Remote Transmit Emphasis Control Support	see footnote[3]	Indicates whether the port is able to transmit commands to control the transmit emphasis in the connected port 0b0 - The port does not support transmit emphasis adjustment in the connected port 0b1 - The port supports transmit emphasis adjustment in the connected port
31	Remote Transmit Emphasis Control Enable	see footnote[4]	Controls whether the port may adjust the transmit emphasis in the connected port 0b0 - Remote transmit emphasis control is disabled 0b1 - Remote transmit emphasis control is enabled The port shall not let this bit be set unless remote transmit emphasis control is supported and the link to which the port is connected is using IDLE2 or IDLE3.

[1]The reset value is implementation dependent

[2]The reset value is 0b1 if feature is supported, otherwise 0b0

[3]The Remote Transmit Emphasis Control Support reset value is implementation dependent

[4]The Remote Transmit Emphasis Control Enable reset value is implementation dependent

7.6.10 Port *n* Error and Status CSRs (RM-I Block Offset 0x58, 78, ... , 238) (RM-II Block Offset 0x58, 98, ... , 418)

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

Table 7-16. Bit Settings for Port *n* Error and Status CSRs

Bit	Name	Reset Value	Description
0	Idle Sequence 2 Support	see footnote[1]	Indicates whether the port supports idle sequence 2 for Baud Rate Class 1. 0b0 - idle sequence 2 not supported for Baud Rate Class 1. 0b1 - idle sequence 2 supported for Baud Rate Class 1.
1	Idle Sequence 2 Enable	see footnote[2]	Controls whether idle sequence 2 is enabled for Baud Rate Class 1. 0b0 - idle sequence 2 disabled for Baud Rate Class 1. 0b1 - idle sequence 2 enabled for Baud Rate Class 1. The port shall not allow this bit to be set unless idle sequence 2 is supported and shall not allow this bit to be cleared if only idle sequence 2 is supported.
2-3	Idle Sequence	see footnote[1]	Indicates which IDLE sequence is active. 0b00 - idle sequence 1 is active. 0b01 - reserved. 0b10 - idle sequence 2 is active. 0b11 - idle sequence 3 is active.
4	Flow Control Mode	0b0	Indicates which flow control mode is active (read only). 0b0 - receiver-controlled flow control is active. 0b1 - transmitter-controlled flow control is active.
5-10	—		Reserved
11	Output Retry-encountered	0b0	Output port has encountered a retry condition.This bit is set when bit 13 is set. Once set, remains set until written with a logic 1 to clear.
12	Output Retried	0b0	Output port has received a packet-retry control symbol and can not make forward progress. This bit is set when bit 13 is set and is cleared when a packet-accepted or a packet-not-accepted control symbol is received (read-only).
13	Output Retry-stopped	0b0	Output port has received a packet-retry control symbol and is in the “output retry-stopped” state (read-only).
14	Output Error-encountered	0b0	Output port has encountered (and possibly recovered from) a transmission error. This bit is set when bit 15 is set. Once set, remains set until written with a logic 1 to clear.
15	Output Error-stopped	0b0	Output is in the “output error-stopped” state (read-only).
16-20	—		Reserved
21	Input Retry-stopped	0b0	Input port is in the “input retry-stopped” state (read-only).
22	Input Error-encountered	0b0	Input port has encountered (and possibly recovered from) a transmission error. This bit is set when bit 23 is set. Once set, remains set until written with a logic 1 to clear.
23	Input Error-stopped	0b0	Input port is in the “input error-stopped” state (read-only).
24-25	—		Reserved

Table 7-16. Bit Settings for Port *n* Error and Status CSRs

Bit	Name	Reset Value	Description
26	Port-write Disabled	0b0	The port-write disable field shall control whether the port can trigger port-write transmission and interrupt assertion for the device. 0b0 - Events for this port shall trigger port-write transmission and interrupt assertion for as long as the port-write pending bit is set. 0b1 - Events for this port shall not trigger port-write transmission and interrupt assertion regardless of the state of the port-write pending bit.
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 Unavailable	see footnote[3]	Indicates whether or not the port is available (read only). The port's resources may have been merged with another port to support wider links. 0b0 - The port is available for use. 0b1 - The port is not available for use.
29	Port Error	0b0	Input or output port has encountered an error from which hardware was unable to recover. Once set, remains set until written with a logic 1 to clear.
30	Port OK	0b0	The input and output ports are initialized and the port is exchanging error-free control symbols with the attached device (read-only).
31	Port Uninitialized	0b1	Input and output ports are not initialized. This bit and bit 30 are mutually exclusive (read-only).

[1]The reset value is implementation dependent

[2]The reset value is 0b1 if feature is supported, otherwise 0b0

[3]The Port Unavailable reset value is implementation dependent

7.6.11 Port *n* Control CSRs (RM-I Block Offsets 0x5C, 7C, ... , 23C) (RM-II Block Offsets 0x5C, 9C, ... , 41C)

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

Table 7-17. Bit Settings for Port *n* Control CSRs

Bit	Name	Reset Value	Description
0-1	Port Width Support	see footnote[1]	Indicates port width modes supported by the port (read-only). This field is used in conjunction with the Extended Port Width Support field of this register. The bits of these two fields collectively indicate the port width modes supported by the port in addition to 1x mode which is supported by all ports Bit 0: 0b0 - 2x mode not supported 0b1 - 2x mode supported Bit 1: 0b0 - 4x mode not supported 0b1 - 4x mode supported
2-4	Initialized Port Width	see footnote[2]	Indicates the width of the link after port initialization when the port is operating in symmetric mode. When the port is operating in asymmetric mode, indicates the maximum width of the link after port initialization. (read only) 0b000 - 1 lane (1x mode) 0b001 - 1 lane (1x mode) receiving on lane R 0b010 - 4 lanes (4x mode) 0b011 - 2 lanes (2x mode) 0b100 - 8 lanes (8x mode) 0b101 - 16 lanes (16x mode) 0b110 - 0b111 - Reserved

Bit	Name	Reset Value	Description
5 - 7	Port Width Override	0b000	Soft port configuration to control the width modes available for port initialization. The bits in this field are used and defined in conjunction with the bits of the Extended Port Width Override field (bits 16-17). When bit [5] = 0b0, bits 16-17 are Reserved When bit [5] = 0b1, bit 6 controls the enabling of 4x mode, bit 7 controls the enabling of 2x mode, bit 16 controls the enabling of 8x mode and bit 17 controls the enabling of 16x mode. Port n Control CSR bits [5-7,16-17] 0b000xx - All lanes widths supported by the port are enabled 0b001xx - Reserved 0b010xx - Force 1x mode, lane R not forced 0b011xx - Force 1x mode, force lane R 0b10000 - Implementation specific behavior 0b10001 - 16x mode enabled; 2x, 4x and 8x modes disabled 0b10010 - 8x mode enabled; 2x, 4x and 16x modes disabled 0b10011 - 8x and 16x modes enabled; 2x and 4x modes disabled 0b10100 - 2x mode enabled; 4x, 8x and 16x modes disabled 0b10101 - 2x and 16x modes enabled; 4x and 8x modes disabled 0b10110 - 2x and 8x modes enabled; 4x and 16x modes disabled 0b10111 - 2x, 8x and 16x modes enabled; 4x mode disabled 0b11000 - 4x mode enabled; 2x, 8x and 16x modes disabled 0b11001 - 4x and 16x modes enabled; 2x and 8x modes disabled 0b11010 - 4x and 8x modes enabled; 2x and 16x modes disabled 0b11011 - 4x, 8x and 16x modes enabled; 2x mode disabled 0b11100 - 2x and 4x modes enabled; 8x and 16x modes disabled 0b11101- 2x, 4x and 16x modes enabled; 8x mode disabled 0b11110 - 2x, 4x and 8x modes enabled; 16x mode disabled 0b11111- 2x, 4x, 8x and 16 x modes enabled The port shall not allow the enabling of a link width mode that is not supported by the port. A change in the value of the Port Width Override or Extended Port Width Override field shall cause the port to re-initialize using the new field value(s).
8	Port Disable	0b0	Port disable: 0b0 - port receivers/drivers are enabled 0b1 - port receivers/drivers are disabled and are unable to receive/transmit any packets or control symbols
9	Output Port Enable	see footnote[3]	Output port transmit enable: 0b0 - port is stopped and not enabled to issue any packets except to route or respond to I/O logical MAINTENANCE packets. Control symbols are not affected and are sent normally. This is the recommended state after device reset. 0b1 - port is enabled to issue any packets

Bit	Name	Reset Value	Description
10	Input Port Enable	see footnote[4]	Input port receive enable: 0b0 - port is stopped and only enabled to route or respond I/O logical MAINTENANCE packets. 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. This is the recommended state after device reset. 0b1 - port is enabled to respond to any packet
11	Error Checking Disable	0b0	This bit disables all RapidIO transmission error checking 0b0 - Error checking and recovery is enabled 0b1 - Error checking and recovery is disabled Device behavior when error checking and recovery is disabled and an error condition occurs is undefined
12	Multicast-event Participant	see footnote[5]	Send incoming Multicast-event control symbols to this port (multiple port devices only)
13	—		Reserved
14	Enumeration Boundary	see footnote[6]	An enumeration boundary aware system enumeration algorithm shall honor this flag. The algorithm, on either the ingress or the egress port, shall not enumerate past a port with this bit set. This provides for software enforced enumeration domains within the RapidIO fabric.
15	—		Reserved
16-17	Extended Port Width Override	0b00	Extended soft port configuration to control the width modes available for port initialization. The bits in this field are used and defined in conjunction with the bits in the Port Width Override field. See the Description of the Port Width Override field for the specification of these bits.
18-19	Extended Port Width Support	see footnote[7]	Indicates additional port width modes supported by the port (read-only). This field is used in conjunction with the Port Width Support field of this register. The bits of these two fields collectively indicate the port width modes supported by the port in addition to 1x mode which is supported by all ports Bit 18: 0b0 - 8x mode not supported 0b1 - 8x mode supported Bit 19: 0b0 - 16x mode not supported 0b1 - 16x mode supported
20-27	Implementation-defined		Implementation-defined
28-30	—		Reserved
31	Port Type		This indicates the port type (read only) 0b0 - Reserved 0b1 - Serial port

[1]The Port Width reset value is implementation dependent

[2]The Initialized Port Width reset value is implementation dependent

[3]The Output Port Enable reset value is implementation dependent

[4]The Input Port Enable reset value is implementation dependent

[5]The Multicast-event Participant reset value is implementation dependent

[6]The Enumeration Boundary reset value is implementation dependent; provision shall be made to allow the reset value to be configurable if this feature is supported

[7]The Extended Port Width Support reset value is implementation dependent

7.6.12 Port *n* Outbound ackID CSRs (RM-II Block Offsets 0x60, 0xA0, ... , 0x420)

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 the output side of the port at the time the read was initiated, and before a maintenance response, if any, was generated.

Table 7-18. Bit Settings for Port *n* Outbound ackID Status CSRs

Bit	Name	Reset Value	Description
0	Clr_outstanding_ackIDs	0b0	Writing 0b1 to this bit causes all outstanding unacknowledged packets to be discarded. This bit should only be written when trying to recover a failed link. This bit is always logic 0 when read.
1-7	—	All 0’s	Reserved
8-19	Outstanding_ackID	0x000	Output port unacknowledged ackID status. Next expected acknowledge control symbol ackID field that indicates the ackID value expected in the next received acknowledge control symbol. Bits 15-19 are valid for Control Symbol 24. Bits 14-19 are valid for Control Symbol 48. All bits are valid for Control Symbol 64.
20-31	Outbound_ackID	0x000	Output port next transmitted ackID value. Software writing this value can force retransmission of outstanding unacknowledged packets in order to manually implement error recovery. Bits 27-31 are valid for Control Symbol 24. Bits 26-31 are valid for Control Symbol 48. All bits are valid for Control Symbol 64.

7.6.13 Port *n* Inbound ackID CSRs (RM-II Block Offsets 0x64, 0xA4, ... , 0x424)

The port link local Inbound ackID status registers are accessible both by a local processor and an external device. These registers are required for devices that support Control Symbol 64. A read to these registers returns the local inbound ackID status for the input side for Control Symbol 64 at the time the read was initiated.

Table 7-19. Bit Settings for Port *n* Inbound ackID CSRs

Bit	Name	Reset Value	Description
0-19	—	All 0’s	Reserved
20-31	Inbound_ackID	0x000	Input port next expected ackID value. Bits 27-31 are valid for Control Symbol 24. Bits 26-31 are valid for Control Symbol 48. All bits are valid for Control Symbol 64.

7.6.14 Port *n* Power Management CSRs (RM-II Block Offsets 0x68, A8, ... , 428)

The Port *n* Power Management CSRs are used to control power management through the use of asymmetric links. Unless otherwise specified, the reset values of all fields are implementation specific. All bits and bit fields in this register shall be as defined in Table 7-20. Unless otherwise specified, the bits and bit fields of this register shall be read/write.

The fields in this register is used to control the asymmetric width from software, refer to the description of asymmetric mode operation described in Section 5.17, "Asymmetric Operation".

This register shall be implemented for devices which support asymmetric operation.

Table 7-20. Bit Settings for Port *n* Power Management CSRs

Bit	Name	Reset Value	Description
0-4	Asymmetric modes supported	Impl. Spec	Indicates the asymmetric widths that are supported by a port. 0b1xxxx - 1x mode receive and transmit 0bx1xxx - 2x mode receive and transmit 0bxx1xx - 4x mode receive and transmit 0bxxx1x - 8x mode receive and transmit 0bxxxx1 - 16x mode receive and transmit This field shall be 0b00000 for ports that only support 1x operation. This field shall indicate support for port widths that are supported and enabled according to the Port n Control CSR bit fields. This field is read-only.
5-9	Asymmetric modes enabled	Impl. Spec.	Indicates the asymmetric widths that are enabled for a port. 0b1xxxx - 1x mode receive and transmit 0bx1xxx - 2x mode receive and transmit 0bxx1xx - 4x mode receive and transmit 0bxxx1x - 8x mode receive and transmit 0bxxxx1 - 16x mode receive and transmit Implementations shall only allow bits in this field to be set if the corresponding bit in the “Asymmetric Modes Supported” field is set.
10-12	Transmit width status	0b000	Indicates the operating width of the transmitter. 0b000 - none 0b001 - 1x mode transmit 0b010 - 2x mode transmit 0b011 - 4x mode transmit 0b100 - 8x mode transmit 0b101 - 16x mode transmit 0b110 - 0b111 - Reserved When Receive asymmetric mode status is 0b000, the Transmit asymmetric mode status shall be 0b000. This field is read-only.

Table 7-20. Bit Settings for Port *n* Power Management CSRs (Continued)

Bit	Name	Reset Value	Description
13-15	Receive width status	0b000	Indicates the operating width of the receiver. 0b000 - none. 0b001 - 1x mode receive 0b010 - 2x mode receive 0b011 - 4x mode receive 0b100 - 8x mode receive 0b101 - 16x mode receive 0b110–0b111 - Reserved When Transmit asymmetric mode status is 0b000, the receive asymmetric mode status shall be 0b000. This field is read-only.
16-18	Change my transmit width	0b000	This is a request to change the local transmitter asymmetric mode. 0b000 - No change 0b001 - 1x mode transmit 0b010 - 2x mode transmit 0b011 - 4x mode transmit 0b100 - 8x mode transmit 0b101 - 16x mode transmit 0b110–0b111 - Reserved The value of this field shall always be 0b000 when read. Writing this field with a value other than zero while the Status of My Transmit Width Change field is 0b01 shall result in implementation specific behavior. Requesting a transmitter width that is not supported or disabled in the local transmitter shall result in implementation specific behavior.
19-20	Status of My Transmit Width Change	0b00	This field gives the status of the last requested change to the local transmitter width. 0b00 - No status 0b01 - ACK - the command has been successfully executed 0b10 - NACK - the command has for some reason not been executed and is rejected 0b11 - Reserved This field is read-only.
21-23	Change Link Partner Transmit Width	0b000	This is a request to change the link partners transmitter asymmetric mode. 0b000 - No change 0b001 - 1x mode transmit 0b010 - 2x mode transmit 0b011 - 4x mode transmit 0b100 - 8x mode transmit 0b101 - 16x mode transmit 0b110–0b111 - Reserved The value of this field shall always be 0b000 when read. Writing this field with a value other than zero while the Status of Link Partner Transmit Width Change field is 0b01 shall result in implementation specific behavior.
24-25	Status of Link Partner Transmit Width Change	0b00	This field gives the status of the last requested change to the link partner’s transmitter width. 0b00 - No request outstanding/request completed 0b01 - In progress 0b10 - Failed due to timeout 0b11 - Reserved This field is read-only.
26-31	—		Reserved

7.6.15 Port *n* Latency Optimization CSRs (RM-II Block Offset 0x6C, AC, ... , 42C)

These registers indicate the capabilities of a device to reduce latency through optional protocol enhancements, and control whether these capabilities are enabled. All bits and bit fields in these registers shall be as defined in Table 7-21. Unless otherwise specified, the bits and bit fields of these registers shall be read/write.

Table 7-21. Bit Settings for Port *n* Latency Optimization CSRs

Bit	Name	Reset Value	Description
0	Multiple Acknowledges Supported	see footnote[1]	Indicates whether the port supports reception of Packet Accepted, Packet Not Accepted, and Retry control symbols which acknowledge multiple outstanding ackIDs. 0b0 - A control symbol shall always acknowledge one ackID 0b1 - A control symbol shall acknowledge multiple outstanding ackIDs. This bit shall be read-only.
1	Error Recovery with ackID in PNA Supported	see footnote[1]	Indicates whether the port can use the ackID value optionally found in a Packet Not Accepted control symbol to start transmitting packets before receipt of a link-response control symbol. 0b0 - The port cannot use the ackID value in a Packet Not Accepted control symbol 0b1 - The port can use the ackID value in a Packet Not Accepted control symbol This bit shall be read-only.
2	TX AckID_Status in PNA Supported	see footnote[1]	Indicates whether the port places the ackID of the next expected packet in the “arbitrary, or ackID_status” field of a Packet Not Accepted control symbol. If transmitter based flow control is in use on the link and this bit is set, the port also transmits Status and VC_Status control symbols when a Packet Not Accepted control symbol is sent. 0b0 - The Packet Not Accepted “arbitrary/ackID_status” field contains arbitrary values. 0b1 - The Packet Not Accepted “arbitrary/ackID_status” field contains the ackID of the next expected packet. This bit shall be set if the “Error Recovery with ackID in PNA Supported” field is set. This bit shall be read-only.
3-7	—		Reserved
8	Multiple Acknowledges Enabled	see footnote[2]	Controls whether the port shall accept and optionally transmit Packet Accepted, Packet Not Accepted, and Retry control symbols which acknowledge multiple ackIDs. 0b0 - A Packet Accepted control symbol shall always acknowledge one ackID 0b1 - A Packet Accepted control symbol shall acknowledge all ackIDs up to and including the ackID found in the Packet Accepted control symbol. If the Multiple Acknowledges Supported field is clear, this field shall be reserved. When this bit is set, the port may transmit Packet Accepted, Packet Not Accepted and Retry control symbols which acknowledge multiple ackIDs.

Table 7-21. Bit Settings for Port *n* Latency Optimization CSRs

Bit	Name	Reset Value	Description
9	Error Recovery with ackID in PNA Enabled	see footnote[2]	Controls when the port shall use the ackID value found in a Packet Not Accepted control symbol to start transmitting packets before receipt of a link-response control symbol. 0b0 - The port shall not use the ackID value in a Packet Not Accepted control symbol 0b1 - The port shall use the ackID value in received Packet Not Accepted control symbols. The port shall transmit the ackID value in a Packet Not Accepted control symbol. If the Error Recovery with ackID in PNA Supported field is clear, this field shall be reserved.
10-31	—		Reserved

[1]The value of this field shall be1 for links operating with Control Symbol 64. The value of this field is implementation specific for ports which are operating with Control Symbol 24 or Control Symbol 48.

[2]The reset value shall be 1 for links operating with Control Symbol 64. The reset value shall be 0 for links operating with Control Symbol 24 or Control Symbol 48.

7.6.16 Port *n* Link Timers Control CSRs (RM-II Block Offsets 0x70, 0xB0, ... , 0x430)

The Port *n* Link Timers Control CSRs are used to control timers related to link training operation. All bit fields in this register shall be as defined in Table 7-22. Unless otherwise specified, the bits of this register shall be readable and writable.

Table 7-22. Bit Settings for Port *n* Link Timers Control CSRs

Bit	Name	Reset Value	Description
0-7	DME Training Completion Timer	See Description	Controls the length of time allowed for DME training to complete. The Maximum Period for this timeout shall be one second +/- 34%. The programmed period for this timeout is computed by: (DME Training Completion Timer) * (Maximum Period/256). For purposes of interoperability, the default timeout period must be more accurate than one second +/- 34%. The reset value of this timer is the implementation specific value which results in a DME Training Completion timeout period that is: - at least 500 milliseconds and - is as close to 500 milliseconds as possible A value of 0 shall disable this timer. NOTE: The Maximum Period of this timeout is specified loosely (+/- 34%) to allow implementation flexibility and innovation. The reset value of the timeout is specified more tightly (+ 0 to 1/256%) to ensure consistent, interoperable behavior during link initialization. This register field is reserved when the port is operating with IDLE1 or IDLE2.
8-15	DME Wait_Timer	0x3F	Controls the number of DME training frames transmitted after the link partner has indicated that its receiver is trained. This value is encoded as the number of training frames to send, divided by 4. The default value shall cause transmission of 252 training frames. The maximum value shall cause transmission of 1020 training frames. A value of 0 shall cause DME training frames to be transmitted continuously until the DME Training Completion Timer expires. This register field is reserved when the port is operating with IDLE1 or IDLE2.

Table 7-22. Bit Settings for Port *n* Link Timers Control CSRs

Bit	Name	Reset Value	Description
16-23	CW Training Completion Timer	See Description	Controls the length of time allowed for training to complete for codeword training when operating with IDLE3 and CS Field training when operating with IDLE2. This timer shall have the same Maximum Period and reset value as the DME Training Completion Timer when operating with IDLE3. The Maximum Period and reset value of this field shall be implementation specific when operating with IDLE2 as such a values have not been defined in the standard. The programmed period for this timeout is computed by: (CW Training Completion Timer) * (Maximum Period/256). A value of 0 shall disable this timer. This register field is reserved when the port is operating with IDLE1.
24-31	Emphasis Command Timeout	0xFF	Controls the length of time allowed for transmit emphasis command to be acknowledged during DME training, CW training, CS Field training, and retraining. The Maximum Period for this timeout shall be 256 microseconds +/- 34%. The programmed period for this timeout is computed by: (Emphasis Command Timeout) * (Maximum Period/256). A value of 0 shall disable this timer. The reset value of this timer is the implementation specific value which results in a Emphasis Command timeout period that is: - at least 100 microseconds and - is as close to 100 microseconds as possible A value of 0 shall disable this timer. NOTE: The Maximum Period of this timeout is specified loosely (+/- 34%) to allow implementation flexibility and innovation. The reset value of the timeout is specified more tightly (+ 0 to 1/256%) to ensure consistent, interoperable behavior during link initialization. This register field is reserved when the port is operating with IDLE1.

7.6.17 Port *n* Link Timers Control 2 CSRs (RM-II Block Offsets 0x74, 0xB4, ... , 0x434)

The Port *n* Link Timers Control 2 CSRs are used to control timers related to link retraining operation and link initialization. All bit fields in this register shall be as defined in Table 7-23. Unless otherwise specified, the bits of this register shall be readable and writable.

Table 7-23. Bit Settings for Port *n* Link Timers Control 2 CSRs

Bit	Name	Reset Value	Description
0-7	Retraining Completion Timer	0xFE	Controls the length of time allowed for retraining a lane once the lane is determined to be operating in a degraded state. The Maximum Period for this timeout is 62.5 milliseconds, +/- 34%. The programmed period for this timeout is computed by: (Retraining Completion Timer) * (Maximum Period/256). A value of 0 shall disable this timer. The value of this timer shall be programmed to be less than the Recovery Timer. This register field is reserved when the port is operating with IDLE1 or IDLE2.
8-15	Discovery Completion Timer	See Description	Controls the length of time allowed for Discovery for multi-lane ports. This timer shall have the same Maximum Period as the DME Training Completion Timer. The programmed period for this timeout is computed by: (Discovery Completion Timer) * (Maximum Period/256). When operation with IDLE3 the reset value of this field shall be computed by adding 1 to the reset value of the DME Training Completion Timer. When operating with IDLE2 that implements the CW Training Completion Timer the reset value of this field shall be computed by adding 1 to the reset value of the CW Training Completion Timer. When operating with IDLE1 or with IDLE2 that does not implement the CW Training Completion Timer, the reset value shall be matching the requirement of a 28 +/- 4 msec discovery time. A value of 0 shall disable this timer. The value of this timer shall be programmed to be larger than both the DME Training Completion Timer and the CW Training Completion Timer when operating with IDLE3 or with IDLE2 that implements the CW Training Completion Timer.
16-23	Recovery Timer	See Description	Controls the length of time the Port_Initialization state machines and the Receive_Width state machine are allowed to remain in the 1x_RECOVERY, 2x_RECOVERY, or Nx_RECOVERY states. The Maximum Period for this timeout is 62.5 milliseconds, +/- 34%. The programmed period for this timeout is computed by: (Recovery Timer) * (Maximum Period/256). When operating with IDLE3 the reset value shall be 0xFF. When operating with IDLE1 or IDLE2 the reset value shall match the requirement of a 28 +/- 4 msec recovery time. A value of 0 shall disable this timer.
24-31	—		Reserved

7.6.18 Port *n* Link Timers Control 3 CSRs (RM-II Block Offsets 0x78, 0xB8, ... , 0x438)

The Port *n* Link Timers Control 3 CSRs are used to control timers related to IDLE3 asymmetric mode operation. All bit fields in this register shall be as defined in Table 7-24. Unless otherwise specified, the bits of this register shall be readable and writable.

This register shall be implemented for devices which support asymmetric operation.

Table 7-24. Bit Settings for Port *n* Link Timers Control 3 CSRs

Bit	Name	Reset Value	Description
0-7	Transmit Width Command Timeout	0xFF	Controls the length of time allowed for a Transmit Width Command change to complete. The Maximum Period for this timeout is 250 microseconds, +/- 34%. The programmed period for this timeout is computed by: (Transmit Width Command Timeout) * (Maximum Period/256). A value of 0 shall disable this timer.
8-15	Receive Width Command Timeout	0x40	Controls the length of time allowed for a Receive Width Command change to complete. The Maximum Period for this timeout is 250 microseconds, +/- 34%. The programmed period for this timeout is computed by: (Receive Width Command Timeout) * (Maximum Period/256). A value of 0 shall disable this timer.
16-21	Keep-alive Transmission Period	0x01	Controls the length of time a lane shall transmit to keep the link partner SerDes alive on lanes that are not in use in asymmetric mode. The Maximum Period for transmission is 125 microseconds, +/- 34%. The programmed period for transmission is computed by: (Keep-alive Transmission Period) * (Maximum Period/64). A value of 0 results in implementation specific behavior.
22-31	Keep-alive Transmission Interval	0x3FF	Controls the length of time between Keep-alive Transmission Periods for lanes that are not in use when a port is operating in asymmetric mode. The Maximum Period for this timeout is 10 seconds, +/- 34%. The programmed period for this timeout is computed by: (Keep-alive Transmission Interval) * (Maximum Period/1024). A value of 0 shall disable this timeout. When the timeout is disabled, no Keep-Alive transmissions are performed.

7.7 LP-Serial Lane Extended Features Block

This section specifies the LP-Serial Lane Extended Features Block. All registers in this block are 32 bits in length, aligned to a 32-bit (4-byte) boundary and accessed as 4 byte entities, although some processing elements may optionally allow larger accesses. This Extended Features register block is assigned Extended Features block ID=0x000D.

7.7.1 Register Map

Table 7-25 shows the register map of the RapidIO LP-Serial Lane Extended Features Block. The register map specifies the registers that comprise this Extended Features Block.

The Block Offset is the offset relative to the 16-bit Extended Features Pointer (EF_PTR) that points to the beginning of the block. The address of a byte in the block is calculated by adding the block byte offset to EF_PTR that points to the beginning of the block. This is denoted as [EF_PTR+xx] where xx is the block byte offset in hexadecimal.

This register map is currently only defined for devices with up to 32 LP-Serial lanes, but can be extended or shortened if more or less lane definitions are required for a device. For example, a device with four LP-Serial lanes is only required to use register map space corresponding to offsets [EF_PTR + 0x00] through [EF_PTR + 0x8C]. Register map offset [EF_PTR + 0x90] can be used for another Extended Features block.

Table 7-25. LP-Serial Lane Register Map

	Block Byte Offset	Register Name
General	0x0	LP-Serial Lane Register Block Header
	0x4–C	Reserved
Lane 0	0x10	Lane 0 Status 0 CSR
	0x14	Lane 0 Status 1 CSR
	0x18	Lane 0 Status 2 CSR
	0x1C	Lane 0 Status 3 CSR
	0x20	Lane 0 Status 4 CSR
	0x24	Lane 0 Status 5 CSR
	0x28	Lane 0 Status 6 CSR
	0x2C	Lane 0 Status 7 CSR

Table 7-25. LP-Serial Lane Register Map

	Block Byte Offset	Register Name
Lane 1	0x30	Lane 1 Status 0 CSR
	0x34	Lane 1 Status 1 CSR
	0x38	Lane 1 Status 2 CSR
	0x3C	Lane 1 Status 3 CSR
	0x40	Lane 1 Status 4 CSR
	0x44	Lane 1 Status 5 CSR
	0x48	Lane 1 Status 6 CSR
	0x4C	Lane 1 Status 7 CSR
Lanes 2-30	0x50–3EC	Registers for lanes 2-30
Lane 31	0x3F0	Lane 31 Status 0 CSR
	0x3F4	Lane 31 Status 1 CSR
	0x3F8	Lane 31 Status 2 CSR
	0x3FC	Lane 31 Status 3 CSR
	0x400	Lane 31 Status 4 CSR
	0x404	Lane 31 Status 5 CSR
	0x408	Lane 31 Status 6 CSR
	0x40C	Lane 31 Status 7 CSR

The structure and use of the registers comprising the LP-Serial Lane Extended Features Block is specified in Section 7.7.2.

The required behavior for accesses to reserved registers and register bits is specified in Table 7-2.

7.7.2 LP-Serial Lane Command and Status Registers (CSRs)

7.7.2.1 LP-Serial Register Block Header (Block Offset 0x0)

The LP-Serial register block header register contains the EF_PTR to the next extended features block and the EF_ID that identifies LP-Serial Lane Extended Feature Block for which this is the register block header.

Table 7-26. Bit Settings for LP-Serial Register Block Header

Bit	Name	Reset Value	Description
0-15	EF_PTR		Hard wired pointer to the next block in the data structure, if one exists
16-31	EF_ID	0x000D	Hard wired Extended Features Block ID

7.7.2.2 Lane *n* Status 0 CSRs (Block Offsets 0x10, 30, ... , 3F0)

This register shall always be implemented. It contains status information about the local lane transceiver, i.e. the lane *n* transceiver in the device implementing this register. Unless otherwise specified, all bits in this register are read-only.

Table 7-27. Bit Settings for Lane *n* Status 0 CSRs

Bit	Name	Reset Value	Description
0-7	Port Number		The number of the port within the device to which the lane is assigned
8-11	Lane Number		The number of the lane within the port to which the lane is assigned
12	Transmitter type		Transmitter type 0b0 - short run 0b1 - long run
13	Transmitter mode		Transmitter operating mode 0b0 - short run 0b1 - long run
14-15	Receiver type		Receiver type 0b00 - short run 0b01 - medium run 0b10 - long run 0b11 - Reserved The encoding for medium run shall be reserved when operating at Baud Rate Class 3.
16	Receiver input inverted		This bit indicates whether the lane receiver has detected that the polarity of its input signal is inverted and has inverted its receiver input to correct the polarity. 0b0 - receiver input not inverted 0b1 - receiver input inverted
17	Receiver trained		When the lane receiver controls any transmit or receive adaptive equalization, this bit indicates whether or not all adaptive equalizers controlled by the lane receiver are trained. If the lane supports the IDLE2 sequence, the value of this bit shall be the same as the value in the “Receiver trained” bit in the CS Field transmitted by the lane. 0b0 - One or more adaptive equalizers are controlled by the lane receiver and at least one of those adaptive equalizers is not trained 0b1 - The lane receiver controls no adaptive equalizers or all of the adaptive equalizers controlled by the lane receiver are trained
18	Receiver lane sync		This bit indicates the state of the lane’s lane_sync signal. 0b0: lane_sync FALSE 0b1: lane_sync TRUE
19	Receiver lane ready		This bit indicates the state of the lane’s lane_ready signal 0b0 - lane_ready FALSE 0b1 - lane_ready TRUE

Bit	Name	Reset Value	Description
20-23	8b/10b decoding errors	0x0	For Baud Rate Class 1 and 2 operation, this field shall indicate the number of 8b/10b decoding errors that have been detected for this lane since this register was last read. For Baud Rate Class 3 operation, this field shall indicate the number of bit interleaved parity (lane check) failures. The field shall be reset to 0x0 when the register is read. 0x0: No 8b/10b decoding errors have been detected since this register was last read. 0x1: One 8b/10b decoding error has been detected since this register was last read. 0x2: Two 8b/10b decoding errors have been detected since this register was last read. ... 0xD: Thirteen 8b/10b decoding errors have been detected since this register was last read. 0xE: Fourteen 8b/10b decoding errors have been detected since this register was last read. 0xF: At least fifteen 8b/10b decoding errors have been detected since this register was last read.
24	Lane_sync state change	0b0	Indicates whether the lane_sync signal for this lane has changed state since the bit was last read. This bit is reset to 0b0 when the register is read. This bit provides an indication of the burstiness of the transmission errors detected by the lane receiver. 0b0 - The state of lane_sync has not changed since this register was last read 0b1 - The state of lane_sync has changed since this register was last read
25	lane_trained state change	0b0	Indicates whether the lane_trained signal for this lane has changed state since the bit was last read. This bit is reset to 0b0 when the register is read. A change in state of lane_trained indicates that the training state of the adaptive equalization under the control of this receiver has changed. Frequent changes of the training state suggest a problem with the lane. 0b0 - The state of lane_trained has not changed since this register was last read 0b1 - The state of lane_trained has changed since this register was last read
26-27	—		Reserved
28	Status 1 CSR implemented		This bit indicates whether or not the Status 1 CSR is implemented for this lane 0b0 - The Status 1 CSR is not implemented for this lane 0b1 - The Status 1 CSR is implemented for this lane This field shall be 0b1 when Baud Rate Class 3 is supported.
29-31	Status 2-7 CSRs implemented		This field indicates the number of implementation specific Status 2-7 CSRs that are implemented for this lane 0b000 - None of the Status 2-7 CSRs are implemented for this lane 0b001 - The Status 2 CSR is implemented for this lane 0b010 - The Status 2 and 3 CSRs are implemented for this lane 0b011 - The Status 2 through 4 CSRs are implemented for this lane 0b100 - The Status 2 through 5 CSRs are implemented for this lane 0b101 - The Status 2 through 6 CSRs are implemented for this lane 0b110 - The Status 2 through 7 CSRs are implemented for this lane 0b111 - Reserved This field shall have a value of 0b010 or greater when Baud Rate Class 3 is supported.

7.7.2.3 Lane *n* Status 1 CSRs (Block Offsets 0x14, 34, ... , 3F4)

The Lane *n* Status 1 CSRs shall be implemented if the lane supports the IDLE2 or IDLE3 sequence.

When the lane is operating with IDLE2, this register contains information about the connected port that is collected from the CS markers and CS fields of the IDLE2 sequence received by the local lane *n* receiver. Only information from error free CS markers and CS fields shall be reported in this register.

When the lane is operating with IDLE3, this register contains information regarding the lanes' initialization and electrical status. For fields which rely on information from received Status/Control ordered sequence, their value shall only be updated based on error-free Status/Control ordered sequences.

Unless otherwise specified, all bits in these registers are read-only.

Table 7-28. Bit Settings for Lane *n* Status 1 CSRs

Bit	Name	Reset Value	Description
0	IDLE received	0b0	This bit indicates whether valid information has been received by the lane since the bit was last reset. Information is accepted from a IDLE2 Control and Status Field or Field Marker, or a valid IDLE3 Status/Control Ordered Sequence. The bit is R/W. This bit can be reset by writing the bit with the value 0b1. Writing the bit with the value 0b0 does not change the value of the bit. 0b0 - No information has been received since the bit was last reset 0b1 - An information has been received at some time since the bit was last reset
1	IDLE information current	0b0	This bit indicates whether the information in this register that is collected from the received IDLE sequence is current. When asserted, this bit indicates that the information is from the last IDLE2 CS Marker and CS Field, or from an IDLE3 Status Control Ordered Sequence that were received by the lane without detected errors, and that the lane's lane_sync signal has remained asserted since the last information was received. 0b0 - The IDLE information is not current 0b1 - The IDLE information is current
2	Values changed	0b1	When the lane is operating using IDLE2, this bit indicates whether the values of any of the other 31 bits in this register have changed since the register was last read. When the lane is operating using IDLE3, this bit indicates whether the values of the IDLE3 fields in this register, or if any fields in the Lane n Status 2 CSR and the Lane n Status 3 CSR have changed. This bit is reset when the register is read. 0b0 - The values have not changed 0b1 - One or more values have changed
3	Implementation defined		Implementation defined
4	IDLE2 connected port lane receiver trained		IDLE2 connected port lane receiver trained 0b0 - Receiver not trained 0b1 - Receiver trained Captured from the IDLE2 Command and Status Field “Receiver Trained” field.

Bit	Name	Reset Value	Description
5-7	IDLE2 received port width		IDLE2 received port width 0b000 - 1 lane 0b001 - 2 lanes 0b010 - 4 lanes 0b011 - 8 lanes 0b100 - 16 lanes 0b101-0b111 - Reserved Captured from the IDLE2 Command and Status Marker “Active Port Width” field.
8-11	IDLE2 lane number in connected port		The number of the lane (0-15) within the connected port 0b0000 - Lane 0 0b0001 - Lane 1 ... 0b1111 - Lane 15 Captured from the IDLE2 Command and Status Marker “Lane Number” field.
12-13	IDLE2 connected port transmit emphasis Tap(-1) status		Tap(-1) status 0b00 - Tap(-1) not implemented 0b01 - Tap(-1) at minimum emphasis 0b10 - Tap(-1) at maximum emphasis 0b11 - Tap(-1) at intermediate emphasis setting Captured from the IDLE2 Command and Status Field “Tap(-1) Status” field.
14-15	IDLE2 connected port transmit emphasis Tap(+1) status		Tap(+1) status 0b00 - Tap(+1) not implemented 0b01 - Tap(+1) at minimum emphasis 0b10 - Tap(+1) at maximum emphasis 0b11 - Tap(+1) at intermediate emphasis setting Captured from the IDLE2 Command and Status Field “Tap(+1) Status” field.
16	IDLE2 connected port scrambling/descrambling enabled		IDLE2 connected port scrambling/descrambling 0b0 - Scrambling/descrambling not enabled 0b1 - Scrambling/descrambling enabled Captured from the IDLE2 Command and Status Field “Data scrambling/descrambling enabled” field.
17	IDLE3 Loss of Signal		This bit shall be set when at least one of the following has occurred since the last time this register was read: • Receive_enable has been continuously asserted for 2048 columns and no control symbols have been received • The “signal_detected” indication is de-asserted • A Status/Control codeword was received that indicates the link partner’s transmitter is entering the silent state, or that the transmitter for this lane is disabled • Lane synchronization was lost 0b0 - The lane is receiving valid data or control codewords b01 - The lane is not receiving valid data or control codewords This field shall be reset to 0x0 when the register is read

Bit	Name	Reset Value	Description
18-20	Training Type	0b000	This field is reserved for IDLE1 links. This field indicates the type of lane training currently being performed by the lane when operating with IDLE3. The field shall be encoded as follows: 0b000 = Lane_training state machine is in the UNTRAINED state 0b001 = Long run Lane_training state machine is in a state whose name begins with “DME”. 0b010 = Lane_training state machine is in a state whose name begins with “CW_TRAIN”. 0b011 = Lane_training state machine is in a state whose name begins with “RETRAIN”. 0b100 = Lane_training state machine is in the TRAINED or KEEP_ALIVE state 0b101-0b111 are Reserved This field may be used to indicate the type of lane training currently being performed when operating with IDLE2. When used for IDLE2, this field shall be encoded as follows: 0b000 = No signal is being received from the link partner. 0b001 = Reserved 0b010 = The link is currently in a state where training is being performed or in a training error state 0b011 = The link is currently in a state where retraining is being performed or in a retraining error state 0b100 = The link has trained successfully and is not currently in a training or retraining state 0b101-0b111 = Reserved
21	IDLE3 DME Training Failed	0b0	For IDLE1 and IDLE2 operation, this field is reserved. For IDLE3 operation, this field shall indicate whether DME training has failed for this lane since this register was last read. This field shall be encoded as follows: 0b0 - No failure seen. 0b1 - DME training has failed since this register was last read. This bit shall be set when the Long Run Lane_Training State Machine enters the DME_TRAINING_FAIL state. This bit may be set for other implementation specific reasons. This field is read only. This bit is cleared when this register is read.
22	IDLE3 DME Training Completed	0b0	This field is reserved for IDLE1 and IDLE2 links. For IDLE3 operation, this field shall indicate whether DME training has completed for this lane since this register was last read. This field shall be encoded as follows: 0b0 - DME training has not completed. 0b1 - DME training has completed since this register was last read. This bit shall be set when the Long Run Lane_Training State Machine transitions from DME_TRAINING2 to the TRAINED state. This field is read only. This bit is cleared when this register is read.

Bit	Name	Reset Value	Description
23	CW Training Failed	0b0	This field is reserved for IDLE1 links. For IDLE3 operation, this field shall indicate whether CW training has failed for this lane since this register was last read. This field shall be encoded as follows: 0b0 - No failure seen. 0b1 - CW training has failed since this register was last read. This bit shall be set when the Long Run or Short Run Lane_Training State Machine enters the CW_TRAINING_FAIL state. This bit may be set for other implementation specific reasons. This field may be used to indicate lane training failure when operating with IDLE2. This field is read only. This bit is cleared when this register is read.
24	CW Training Completed	0b0	This field is reserved for IDLE1 links. For IDLE3 operation, this field shall indicate whether CW training has completed for this lane since this register was last read. This field shall be encoded as follows: 0b0 - CW training has not completed. 0b1 - CW training has completed since this register was last read. This bit shall be set when the Long Run or Short Run Lane_Training State transitions from CW_TRAINING1 to the TRAINED state. This field may be used to indicate lane training completion when operating with IDLE2. This field is read only. This bit is cleared when this register is read.
25	CW Retraining Failed	0b0	This field is reserved for IDLE1 links. For IDLE3 operation, this field shall indicate whether CW retraining has failed for this lane since this register was last read. This field shall be encoded as follows: 0b0 - No failure seen. 0b1 - CW retraining has failed since this register was last read. This bit shall be set when the Long Run or Short Run Lane_Training State Machine enters the RETRAIN_FAIL state. This bit may be set for other implementation specific reasons. This field may be used to indicate lane retraining failure when operating with IDLE2. This field is read only. This bit is cleared when this register is read.
26	CW Retraining Completed	0b0	This field is reserved for IDLE1 links. For IDLE3 operation, this field shall indicate whether CW retraining has completed for this lane since this register was last read. This field shall be encoded as follows: 0b0 - CW retraining has not completed. 0b1 - CW retraining has completed since this register was last read. This bit shall be set when the Long Run or Short Run Lane_Training State Machine transitions from RETRAINING2 to the TRAINED state. This field may be used to indicate lane retraining completion when operating with IDLE2. This field is read only. This bit is cleared when this register is read.
27-31	—		Reserved

7.7.2.4 Lane *n* Status 2 CSRs (Block Offsets 0x18, 38, ... , 3F8)

These registers shall be implemented if the lane supports IDLE3. These registers contain information from received Status/Control ordered sequences. The bits and bit fields of these registers shall be as defined in Table 7-29. Only information from error-free Status/Control ordered sequences, shall be reported in these registers. Unless otherwise specified, all bits in these registers are read-only.

Table 7-29. Bit Settings for Lane *n* Status 2 CSRs

Bit	Name	Reset Value	Description
0-7	LP Port Number	All 0's	Number of the link partner's port that is connected to this lane. It should match what is in the Lane n Status 0 CSR [Port Number] field on the link partner. Captured from the Status/Control control codeword field "Port number".
8-11	LP Lane Number	All 0's	Number of the link partner's lane connected to this lane. It should match what is in the Lane n Status 0 CSR [Lane Number] field on the link partner. Captured from the Status/Control control codeword field "Lane number".
12	LP Remote training support	0b0	Captured from the Status/Control control codeword field "Remote training support".
13	LP Retraining enabled	0b0	Captured from the Status/Control control codeword field "Retraining enabled".
14	LP Asymmetric mode enabled	0b0	The status of support for Asymmetric Operation in the link partner. 0b0 - Asymmetric mode disabled 0b1 - Asymmetric mode enabled
15	LP Port Initialized	0b0	Indicates whether the link partner's port has completed initialization. Matches the port_initialized state machine signal. 0b0 - Port in not initialized 0b1 - Port is initialized
16	LP Transmit 1x mode	0b0	Indicates when the link partner's port is transmitting in 1x symmetric mode. 0b0 - The port is not transmitting in 1x mode. The state machine variable max_width != 1x. 0b1 - The port is transmitting in 1x symmetric mode. The state machine variable max_width = 1x.
17-19	LP Receive width	0b000	The width at which the Link Partner port is currently receiving control symbols and packets. 0b000 - none 0b001 - 1x mode 0b010 - 2x mode 0b011 - 4x mode 0b100 - 8x mode 0b101 - 16x mode 0b110 - 1x mode, lane 1 0b111 - 1x mode, lane 2

Bit	Name	Reset Value	Description
20-22	LP Receive lanes ready	0b000	Indicates the lanes being received by the port for which lane_ready is asserted. 0b000 - No lanes ready 0b001 - lane_ready[0] 0b010 - lane_ready[0] & lane_ready[1] 0b011 - lane_ready[0] & lane_ready[1] &... & lane_ready[3] 0b100 - lane_ready[0] & lane_ready[1] &... & lane_ready[7] 0b101 - lane_ready[0] & lane_ready[1] &... & lane_ready[15] 0b110–0b111 - reserved
23	LP Receive lane ready	0b0	The value and meaning of this bit on lane k shall be the same as that of the link partner’s lane state machine variable lane_ready[k]
24	LP Lane trained	0b0	Indicates the training status of the link partner’s lane. The value and meaning of this bit on lane k shall be the same as that of the link partner’s port state machine variable lane_trained[k]
25-27	LP Change receiver width command	0b000	The port receiving the command shall attempt to switch to the receive width specified in the command. 0b000 - hold current receive width 0b001 - receive in 1x mode 0b010 - receive in 2x mode 0b011 - receive in 4x mode 0b100 - receive in 8x mode 0b101 - receive in 16x mode 0b110–0b111 - reserved
28	LP change receiver width command acknowledge		Receive width command ACK 0b0 - No command status 0b1 - Command executed
29	LP change receiver width command negative acknowledge		Receive width command NACK 0b0 - No command status 0b1 - Command not executed
30-31	—		Reserved

7.7.2.5 Lane *n* Status 3 CSRs (Block Offsets 0x1C, 3C, ... , 3FC)

These registers shall be implemented if the lane supports the IDLE3 sequence. These registers contain information from received Status/Control ordered sequences. The bits and bit fields of these registers shall be as defined in Table 7-30. Only information from error free Status/Control ordered sequences shall be reported in these registers. Unless otherwise specified, all bits in these registers are read-only.

Table 7-30. Bit Settings for Lane *n* Status 3 CSRs

Bit	Name	Reset Value	Description
0-2	LP Transmit width request		A request that the port receiving this field change its transmit width to the width specified in the request. 0b000 - no request (hold current transmit width) 0b001 - request transmit 1x mode 0b010 - request transmit 2x mode 0b011 - request transmit 4x mode 0b100 - request transmit 8x mode 0b101 - request transmit 16x mode 0b110–0b111 - reserved
3	LP Transmit width request pending	0b0	Indicates that the link partner has received the transmitter width request sent by this device and is processing it. 0b0 - No request pending 0b1 - Request pending
4	LP Transmit SC-sequences	0b0	Request to transmit SC-sequence at least every 256 codewords per lane. 0b0 - no additional SC-sequence transmission rate requirement 0b1 - required minimum SC-sequences transmission rate is once every 256 codewords per lane.
5-8	LP Transmit equalizer tap	0b0000	When the transmit equalizer command is tap specific, this field contains the number of the equalizer tap to which the tap specific command shall be applied. The tap number is encoded as a signed 2’s complement 4-bit integer. 0b0000 - Tap 0 0b0001 - Tap +1 0b0010 - Tap +2 0b0011 - Tap +3 0b0100 - Tap +4 0b0101 - Tap +5 0b0110 - Tap +6 0b0111 - Tap +7 0b1000 - Tap -8 0b1001 - Tap -7 0b1010 - Tap -6 0b1011 - Tap -5 0b1100 - Tap -8 0b1101 - Tap -3 0b1110 - Tap -2 0b1111 - Tap -1 When the transmit equalizer update command is not tap specific, the field shall have the value 0b0000 and shall be ignored.

Bit	Name	Reset Value	Description
9-11	LP Transmit equalizer command	0b000	0b000 - Hold/No command 0b001 - Decrement (make more negative by one step) the coefficient of the specified tap. 0b010 - Increment (make more positive by one step) the coefficient of the specified tap. 0b011-0b100 - Reserved 0b101- Initialize - Set the tap coefficients to their INITIALIZE state as defined Clause 72.6.10.4.2 of IEEE Standard 802.3-2008 (part 5). 0b110 - Preset coefficients - Set the coefficient of tap 0 to its maximum value and the coefficients of all other taps to 0 as specified in Clause 72.6.10.4.1 of IEEE Standard 802.3-2008 (part 5). 0b111 - Indicate specified tap implementation status. When Transmit equalizer command are 0b001, 0b010 or 0b111; the Transmit equalizer tap value shall contain the value of the Tap; for other commands the Transmit equalizer tap value shall be 0b0000
12-14	LP Transmit equalizer status	0b000	0b000 - Not updated - No command is pending or the status of the current command has not been determined. 0b001 - Updated - The tap specific command has been executed and the tap is at neither its minimum nor maximum value. 0b010 - Minimum - Either the tap specified tap decrement command has been executed and the tap is now at its minimum value or the specified tap was already at its minimum value. 0b011 - Maximum - Either the tap specific tap increment command has been executed and the tap is now at its maximum value or the specified tap was already at it maximum value. 0b100 - Preset or Initialize command executed. 0b101 - Reserved. 0b110 - Specified tap not implemented. 0b111 - Specified tap implemented.
15	LP Retrain grant	0b0	The value of this bit shall be the same as the value of the link partner’s port state machine variable retrain_grnt.
16	LP Retrain ready	0b0	The value of this bit shall be the same as the value of the link partner’s port state machine variable retrain_ready.
17	LP Retraining	0b0	The value of this bit shall be the same as the value of the link partner’s port state machine variable retraining.
18	LP Port Entering Silence	0b0	0b0 - The link partner’s port is transmitting normally. 0b1 - All lanes of the link partner’s port are going to enter the Silence state.
19	LP Lane Entering Silence	0b0	0b0 - The link partner’s lane is transmitting normally. 0b1 - The link partners’s lane is going to enter the Silence state based on asymmetric mode operation or based on port width downgrade in symmetric mode.
20-27	LP State control reserved	0x00	Captures bit 50-57 of the Status_control field that currently are defined as reserved.
28-31	—		Reserved

7.7.2.6 Implementation Specific CSRs

7.7.2.6.1 Lane n Status 2 CSR (Block Offsets 0x18, 38, ..., 3F8)

7.7.2.6.2 Lane n Status 3 CSR (Block Offsets 0x1C, 3C, ..., 3FC)

7.7.2.6.3 Lane n Status 4 CSR (Block Offsets 0x20, 40, ..., 400)

7.7.2.6.4 Lane n Status 5 CSR (Block Offsets 0x24, 44, ..., 404)

7.7.2.6.5 Lane n Status 6 CSR (Block Offsets 0x28, 48, ..., 408)

7.7.2.6.6 Lane n Status 7 CSR (Block Offsets 0x2C, 4C, ..., 40C)

The implementation of these registers is optional for IDLE1 and IDLE2 and when implemented their contents and format are implementation specific. The implementation of the Lane n Status 4, 5, 6 and 7 CSRs is optional for IDLE3. The registers shall be implemented in increasing numerical order beginning with the Lane *n* Status 2 CSR. For example, if only one of the registers is implemented it shall be the Status 2 CSR. If three registers are implemented they shall be the Status 2 through 4 CSRs, and if five of the registers are implemented, they shall be the Status 2 through 6 CSRs.

7.8 Virtual Channel Extended Features Block

This section describes the registers for RapidIO LP-Serial devices supporting virtual channels. This Extended Features register block is assigned Extended Features block EF_ID=0x000A.

7.8.1 Register Map

Table 7-31 shows the virtual channel register map for RapidIO LP-Serial devices. The Block Offset is the offset relative to the 16-bit Extended Features Pointer (EF_PTR) that points to the beginning of the block.

The address of a byte in the block is calculated by adding the block byte offset to EF_PTR that points to the beginning of the block. This is denoted as [EF_PTR+xx] where xx is the block byte offset in hexadecimal.

Table 7-31. Virtual Channel Registers

	Block Byte Offset	Register Name
General	0x0	VC Register Block Header
	0x4-1C	Reserved
Port 0	0x20	Port 0 VC Control and Status Register
	0x24	Port 0 VC0 BW Allocation Register
	0x28	Port 0 VC 5, VC 1 BW Allocation Register
	0x2C	Port 0 VC 7, VC 3 BW Allocation Register
	0x30	Port 0 VC 6, VC 2 BW Allocation Register
	0x34	Port 0 VC 8, VC 4 BW Allocation Register
	0x38-3C	Reserved
Port 1	0x40	Port 1 VC Control and Status Register
	0x44	Port 1 VC0 BW Allocation Register
	0x48	Port 1 VC 5, VC 1 BW Allocation Register
	0x4C	Port 1 VC 7, VC 3 BW Allocation Register
	0x50	Port 1 VC 6, VC 2 BW Allocation Register
	0x54	Port 1 VC 8, VC 4 BW Allocation Register
	0x58-5C	Reserved
Port *n*	[(0x20 * (n + 1)] to [0x20 * (n + 1) + 0x1C]	Additional Port Registers

The registers are paired according to the VCs as they are implemented. In the second example, with VCs Supported 0x01, the upper portion (VC5 portion) of the register would be non-functioning.

NOTE:

There are no provisions in this specification to provide for dynamic reconfiguration of the VCs. A vendor is not prohibited from implementing dynamic reconfiguration, it is just beyond the scope of this specification. Both ends of the channel need to be configured alike, or unexpected behavior may result, also beyond the scope of this specification. The default method is to configure VC operation when the channel is quiescent either by protocol, or by holding the master enable in the disabled state.

7.8.2 Virtual Channel Control Block Registers

This section contains register descriptions that define the bandwidth allocation configuration for the virtual channels.

7.8.2.1 VC Register Block Header (Block Offset 0x0)

The LP-Serial VC register block header register contains the EF_PTR to the next extended features block and the EF_ID that identifies this as the Virtual Channel Extended Features Block.

Table 7-32. Bit Settings for VC Register Block Header

Bit	Name	Reset Value	Description
0-15	EF_PTR		Hard wired pointer to the next block in the data structure, if one exists
16-31	EF_ID	0x000A	Hard wired Extended Features Block ID

7.8.2.2 Port *n* VC Control and Status Registers (Block Offset ((port number) + 1) * 0x20))

This register is used by each port to set up VC operation.

Table 7-33. Port *n* VC Control and Status Registers

Bit	Name	Reset Value	Description
0 - 7	VC Refresh Interval	0x00	The number of 1024 code-group or codeword intervals over which the VC status must be refreshed. Refresh Interval: 0x0 - 1K code-groups or codewords, 0xF - 16K code-groups or codewords, 0xFF - 256K codegroups or codewords Implementers are required to support a maximum VC refreshing period of at least 1024 x 16 = 16K code-groups or codewords in size. The maximum possible VC refreshing period that can be supported is 1024 x 256 = 256K code-groups or codewords. Writing to this field with a value greater than the maximum supported value by the port will set the field to the maximum value supported by the port
8 - 15	CT Mode	0x00	Enables VCs to operate in CT mode beginning with VC8: 0x00 - all VCs in RT mode For 8 VCs: 0x01 - VC8 in CT mode 0x03 - VC8, VC7 in CT mode 0x07 - VC8, VC7, VC6, VC 5 in CT mode 0x0F - VC8 - VC1 in CT mode For 4 VCs: 0x01 - VC7 in CT mode 0x03 - VC7, VC5 in CT mode 0x07 - VC7, VC5, VC3, VC1 in CT mode For 2 VCs: 0x01 - VC5 in CT mode 0x03 - VC5, VC1 in CT mode For 1 VC: 0x01 - VC1 in CT mode Implementers may support CT mode on a portion of the available VCs. CT mode must be implemented in the highest VCs first to allow this simplified programming model. VCs not supporting CT operation are indicated by not allowing the programmed bits to set. Example: 8VCs enabled, VC8 and VC7 only support CT mode. Writing a 0x07 would result in a register value of 0x03 when read back.

Table 7-33. Port *n* VC Control and Status Registers

Bit	Name	Reset Value	Description
16 - 23	VCs Support	see footnote[1]	Number of Virtual Channels Supported (Read Only) 0x00 - Only VC0 is supported 0x01 - VC0, VC1 Supported 0x02 - VC0, VC1, VC5 supported 0x04 - VC0, VC1, VC3, VC5, VC7 supported 0x08 - VC0, VC1-VC8
24 - 31	VCs Enable	0x00	0x00 - Enable Only VC0 0x01 - Enable VC0, VC1 0x02 - Enable VC0, VC1, VC5 0x04 - Enable VC0, VC1, VC3, VC5, VC7 0x08 - Enable VC0, VC1-VC8 Note: Bits 24-27, and any bits associated with unimplemented VCs need not be writable, but must return 0 when read. Setting this field to a value larger than the number of VCs supported as indicated in bits 16-23 shall result in only VC0 being enabled.

[1]The VCs Supported reset value is implementation dependent

7.8.2.3 Port *n* VC0 BW Allocation Registers (Block Offset (((port number) + 1) * 0x20) + 0x04))

This register is used to enable and configure VC0's participation in the bandwidth reservation scheduling.

Table 7-34. Port *n* VC0 BW Allocation CSRs

Bit	Name	Reset Value	Description
0	VC0 Bandwidth Reservation Capable	see footnote[1]	0b0 - VC0 is strict priority, and has priority over the other VCs. It will utilize bandwidth without regard to bandwidth reservation. The bandwidth reservation algorithm will divide up what bandwidth is remaining after VC0 has no outstanding requests. 0b1 - VC0 is capable of being allocated bandwidth This bit is read only
1	VC0 BW Res Enable	0b0	0b0 - VC0 is strict priority, does not participate in bandwidth reservation 0b1 - VC0 will be allocated bandwidth according to BW Allocation Registers
2 - 7	—		Reserved
8 - 15	Bandwidth Reservation Precision	see footnote[2]	Indicates the number of bits used in the bandwidth reservation precision for all VCs in this port. (read only) 0x00 - 8 bits 0x01 - 9 bits 0x02 - 10 bits 0x04 - 11 bits 0x08 - 12 bits 0x10 - 13 bits 0x20 - 14 bits 0x40 - 15 bits 0x80 - 16 bits
16-31	Bandwidth Allocation	0x00	The contents of this register determines the minimum bandwidth reserved for this VC (see below) The bandwidth allocation value is left justified based on precision. Bits, are ignored based on the precision value: 0bnnnn_nnnn_xxxx_xxxx (8 bit precision) where 'x' represents ignored bits 0bnnnn_nnnn_nxxx_xxxx (9 bit precision) 0bnnnn_nnnn_nnnn_xxxx (12 bit precision), etc.

[1] The VC0 Bandwidth Reservation Capable reset value is implementation dependent
[2] The Bandwidth Reservation Precision reset value is implementation dependent

VC0 may or may not participate in the bandwidth reservation scheduling for the link. The required implementation is for VC0 to be strict priority. Traffic on VC0 is serviced before any of the other VCs in this mode. The remaining bandwidth is then divided according to the percentages in the bandwidth allocations. This will result in the bandwidth allocations being variable if VC0's utilization is significant when compared with the activity on the other VCs.

Optionally, VC0 may be included in the bandwidth reservation scheduling. In this case, the priorities within VC0 are serviced when VC0 is allocated bandwidth on the link. VC0 activity cannot cause the other VCs to receive less than their allocation of bandwidth.

The Bandwidth Reservation Precision field is used to indicate the granularity of bandwidth scheduling for the port. The value in this register applies to the subsequent BW Allocation Registers as well.

The value programmed in the BW Allocation Registers is a binary fraction based on the percentage of the overall total bandwidth. 100% bandwidth is represented by a value of 1.000:

Table 7-35. BW Allocation Register Bit Values

Bit / Value							
0	1	2	3	4	5	6	7
2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}
Bit / Value							
8	9	10	11	12	13	14	15
2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}	2^{-15}	2^{-16}

Example: 33% bandwidth is allocated as:

33/100 = 0.0101010101010101b, so the BW allocation register value is:
0101010101010101b, and would be rounded down to:
01010101xxxxxxxxb if 8 bit precision is being used.

The value may be programmed as is into the left justified register, with the unused bits being ignored, but that might cause some precision errors. Also, if the percentage results in a value smaller than the precision, a value of 0 could result in a VC getting no service. The precision value allows the bandwidth allocation algorithm to round up or down based on the dividing point, and to detect and round up a zero value to allocate at least a minimal increment of bandwidth.

The total of all the allocations should not exceed 100%. The result, by definition, will not be as programmed. The actual behavior will depend on the method used to schedule the activity. The implementation of the scheduler, and thus its behavior when not programmed correctly is outside the scope of this specification.

7.8.2.4 Port *n* VC*x* BW Allocation Registers (Block Offset ((((port number) + 1) * 0x20) + (offset based on VC #, see Table 7-31)))

This register is used to enable and program VCs 1-8 participation in the bandwidth reservation scheduling. Each register supports 2 VCs, ordered as described in Section 7.8.1, "Register Map".

Table 7-36. Port *n* VC*x* BW Allocation CSRs

Bit	Name	Reset Value	Description
0 - 15	Bandwidth Allocation	0x0000	The contents of this register determines the minimum bandwidth reserved for this VC (see below) The bandwidth allocation value is left justified based on precision. Bits, are ignored based on the precision value: 0bnnnn_nnnn_xxxx_xxxx (8 bit precision) where ‘x’ represents ignored bits 0bnnnn_nnnn_nxxx_xxxx (9 bit precision) 0bnnnn_nnnn_nnnn_xxxx (12 bit precision), etc.
16-31	Bandwidth Allocation	0x0000	The contents of this register determines the minimum bandwidth reserved for this VC (see below) The bandwidth allocation value is left justified based on precision. Bits, are ignored based on the precision value: 0bnnnn_nnnn_xxxx_xxxx (8 bit precision) where ‘x’ represents ignored bits 0bnnnn_nnnn_nxxx_xxxx (9 bit precision) 0bnnnn_nnnn_nnnn_xxxx (12 bit precision), etc.

In the instance where VC1 is supported, but VC5 is not, bits 0 - 15 are reserved.

The Bandwidth Allocation is as described previously for VC0.

A value of ‘0’ for bandwidth allocation results in no service being given to that VC. VCs initialize with a value of zero and remain inactive until allocated bandwidth. It is recommended that the bandwidth allocations be made before enabling the VCs, but the actual implementation is beyond the scope of this specification.

7.9 Timestamp Generation Extension Block

The Timestamp Generation Extension Block is optional. The block contains different registers depending on the Timestamp CAR field values as defined in Table 7-37. The “General” column indicates registers that shall be implemented regardless of the values of the Timestamp CAR.

The “MECS” column indicates which registers shall be implemented when the “MECS Master Supported” or “MECS Slave Supported” bit is 1. The “Time Slave” column indicates which registers shall be implemented when the “Timestamp Slave Supported” bit is 1. The “Time Master” column indicates which registers shall be implemented when the “Timestamp Master Supported” bit is 1. In all cases, an “X” in a column means that the register shall be implemented.

If the “MECS Master Supported and MECS Slave Supported”, “Timestamp Slave Supported” and “Time Master” bits are 0, then only General registers shall be implemented.

Table 7-37. Timestamp Generation Extension Block

	Block Byte Offset	Register Name	General	MECS	Time Slave	Time Master
General	0x00	Timestamp Generation Extension Block Header	X	X	X	X
	0x04	Timestamp CAR	X	X	X	X
	0x08	Timestamp Generator Status CSR	X	X	X	X
	0x0C	MECS Tick Interval CSR	-	X	-	-
	0x10	Reserved	-	-	-	-
	0x14	MECS Next Timestamp MSW CSR	-	X	-	-
	0x18	MECS Next Timestamp LSW CSR	-	X	-	-
	0x0C-1C	Reserved	-	-	-	-
	0x20-2C	Implementation Specific	-	-	-	-
	0x30	Reserved	-	-	-	-
	0x34	Timestamp Generator MSW CSR	X	X	X	X
	0x38	Timestamp Generator LSW CSR	X	X	X	X
	0x3C	Reserved	-	-	-	-

Table 7-37. Timestamp Generation Extension Block

	Block Byte Offset	Register Name	General	MECS	Time Slave	Time Master
Port 0	0x40	Reserved	-	-	-	-
	0x44	Port 0 Timestamp 0 MSW CSR	-	-	-	X
	0x48	Port 0 Timestamp 0 LSW CSR	-	-	-	X
	0x4C-50	Reserved	-	-	-	-
	0x54	Port 0 Timestamp 1 MSW CSR	-	-	-	X
	0x58	Port 0 Timestamp 1 LSW CSR	-	-	-	X
	0x5C	Reserved	-	-	-	-
	0x60	Port 0 Timestamp Generator Synchronization CSR	-	-	X	X
	0x64	Port 0 Auto Update Counter CSR	-	-	-	X
	0x68	Port 0 Timestamp Synchronization Command CSR	-	-	-	X
	0x6C	Port 0 Timestamp Synchronization Status CSR	-	-	-	X
	0x70	Port 0 Timestamp Offset CSR	-	-	-	X
	0x74-7C	Implementation Specific	-	-	-	-
Ports 1-14	0x80–3FC	Assigned to Port 1-14 CSRs				
Port 15	0x400	Reserved	-	-	-	-
	0x404	Port 15 Timestamp 0 MSW CSR				X
	0x408	Port 15 Timestamp 0 LSW CSR				X
	0x40C-410	Reserved	-	-	-	-
	0x414	Port 15 Timestamp 1 MSW CSR				X
	0x418	Port 15 Timestamp 1 LSW CSR				X
	0x41C	Reserved	-	-	-	-
	0x420	Port 15 Timestamp Generator Synchronization CSR	-	-	X	X
	0x424	Port 15 Auto Update Counter CSR	-	-	-	X
	0x428	Port 15 Timestamp Synchronization Command CSR	-	-	-	X
	0x42C	Port 15 Timestamp Synchronization Status CSR	-	-	-	X
	0x430	Port 15 Timestamp Offset CSR	-	-	-	X
	0x434-43C	Implementation Specific	-	-	-	-

7.9.1 Timestamp Generation Extension Block Header (Block Offset 0x0)

The Timestamp Generation Extension Block Header register contains the EF_PTR to the next EF_BLK and the EF_ID that identifies this as the Timestamp Generation Extension Block Header. The use and meaning of the bits and bit fields of this register shall be as specified in Table 7-38. The register is read-only.

Table 7-38. Bit Settings for Timestamp Generation Extension Block Header

Bit	Name	Reset Value	Description
0-15	EF_PTR		Hard wired pointer to the next block in the data structure, if one exists
16-31	EF_ID	0x000F	Hard wired Extended Features ID

7.9.2 Timestamp CAR (Block Offset 0x04)

This register indicates which Timestamp Synchronization capabilities the device supports. The use and meaning of the bits and bit fields of this register shall be as specified in Table 7-39. The bits and bit fields in this register are read only.

Table 7-39. Bit Settings for Timestamp CAR

Bit	Name	Reset Value	Description
0	Timestamp Slave Supported	See Footnote 1	Indicates whether the device supports operation as a Timestamp Slave 0b0 - Device does not support operation as a Timestamp Slave 0b1 - Device supports operation as a Timestamp Slave
1	Timestamp Master Supported	See Footnote 1	Indicates whether the device supports operation as a Timestamp Master 0b0 - Device does not support operation as a Timestamp Master 0b1 - Device supports operation as a Timestamp Master
2	Common Clock Frequency Supported	See Footnote 1	Indicates whether the device supports use of a common clock frequency 0b0 - Device does not support common clock frequency 0b1 - Device supports common clock frequency
3	MECS Slave Supported	See Footnote 1	Indicates whether the device supports the MECS Time Synchronization Protocol as a slave 0b0 - Device does not support reception of MECS for time updates 0b1 - Device supports reception of MECS for time updates
4	MECS Master Supported	See Footnote 1	Indicates whether the device supports transmission of MECS as a MECS Master for MECS Time Synchronization Protocol. 0b0 - Device does not support transmission of MECS for time updates 0b1 - Device supports transmission of MECS for time updates
5	SMECS Support	See Footnote 1	Indicates whether the device supports transmission and reception of SMECS. 0b0 - Device does not support transmission or reception of SMECS 0b1 - Device supports transmission and reception of SMECS This bit shall only be set if at least one of the “MECS Slave Support” and “MECS Maser Support” bits is set.
6-31	---	0x00	Reserved

1 The reset value of this field is implementation specific.

7.9.3 Timestamp Generator Status CSR (Block Offset 0x08)

This register indicates the current status of the Timestamp Generator. Note that Table 7-40 contains two columns, “All” and “Common Freq”. An “X” in the “All” column indicates bits which shall be implemented in this register. An “X” in the “Common Freq” column indicates bits which shall be implemented if the Timestamp CAR Common Clock Frequency Support bit field is 0b1.

The use and meaning of the bits and bit fields of this register shall be as specified in Table 7-40. The bits and bit fields in this register are read only unless otherwise specified.

Table 7-40. Bit Settings for Timestamp Generator Status CSR

Bit	Name	Reset Value	Description	All	Common Freq
0	Timestamp Generator Clock Locked	See Footnote 1	Indicates whether the Timestamp Generator counter is operating from a good clock source. 0b0 - Timestamp Generator is not operating with a good clock source. 0b1 - Timestamp Generator is operating with a good clock source.	X	-
1	Timestamp Generator Common Clock	See Footnote 1	Indicates whether the Timestamp Generator counter is operating based on a clock frequency which is the same as that of the link partners. 0b0 - Timestamp Generator is not operating with a common clock frequency. 0b1 - Timestamp Generator is operating with a common clock frequency.	-	X
2	Timestamp Generator Stopped	See Footnote 1	Indicates if the Timestamp Generator counter is not advancing because it is being set to an earlier time. 0b0 - Timestamp Generator is advancing 0b1 - Timestamp Generator is temporarily not advancing	X	-
3	Timestamp Generator Was Stopped	See Footnote 1	Indicates if the Timestamp Generator counter has not advanced because it has been set to an earlier time. 0b0 - Timestamp Generator has advanced continuously since this bit was last cleared 0b1 - Timestamp Generator has temporarily stopped advancing at least once since this bit was last cleared. This bit may be cleared by writing “1” to it.	X	-
4-31	---	0x00	Reserved	-	-

1 The reset value of this field is implementation specific.

7.9.4 MECS Tick Interval CSR (Block Offset 0x10)

On an (S)MECS Master, this register controls the amount of time between transmission of one (S)MECS and the next. On an (S)MECS Slave, this register controls the number of nanoseconds added to the MECS Next Timestamp MSW CSR when an (S)MECS is received. The use and meaning of the bit fields in this register shall be as specified in Table 7-41. The bit fields in this register are read/write.

Table 7-41. Bit Settings for MECS Tick Interval CSR

Bit	Name	Reset Value	Description
0	MECS Time Synchronization Role	See Footnote [1]	Controls whether a device operates as a MECS Master or MECS Slave. 0 - The device is operating as an MECS Slave 1 - The device is operating as an MECS Master If the Timestamp CAR “MECS Slave Supported” and “MECS Master Supported” bits are both set, this field shall be read/write. Otherwise, this field shall be read only.
1	SMECS Selection	See Footnote [2]	Controls whether a device is using MECS or SMECS for its MECS Time Synchronization Role 0 - The device uses MECS 1 - The device uses SMECS If the Timestamp CAR “SMECS Support” bit is set, this field shall be read/write. Otherwise, this field shall be read only and have a value of 0.
2-3	Lost TSG Sync Error Threshold	See Footnote[3]	Controls the number of MECS/SMECS “ticks” that must be lost before declaring the timestamp generator to be out of sync. The selection of MECS or SMECS arrival tracking is controlled by SMECS Selection. The criteria for detecting lost MECS/SMECS is implementation specific. This field is encoded as follows: 0b00 - Lost Tick Error Threshold is disabled 0b01 - If one tick is lost, declare the timestamp generator out of sync 0b10 - If two ticks are lost, declare the timestamp generator out of sync 0b11 - If three ticks are lost, declare the timestamp generator out of sync
4	Lost Tick Error Status	0	This field indicates if the device has detected at least one lost tick. 0 - A Lost Tick Error has not been detected 1 - A Lost Tick Error has been detected This bit must be written with 1 to be cleared. Reporting and control of reporting of this event is defined in Part 8.
5	Lost TSG Sync Error Status	0	This field indicates that the device has detected at least “Lost TSG Sync Error Threshold” consecutive ticks have been lost. 0 - A Lost TSG Sync Error has not been detected 1 - A Lost TSG Sync Error has been detected This bit must be written with 1 to be cleared. Reporting and control of reporting of this event is defined in Part 8.

Table 7-41. Bit Settings for MECS Tick Interval CSR

Bit	Name	Reset Value	Description
6-7	---	0x00	Reserved
8-31	Tick Interval	0x000000	For an (S)MECS Master, an (S)MECS shall be sent when time has advanced by this many nanoseconds. For an MECS Slave, time has advanced by this many nanoseconds whenever an (S)MECS is received. (S)MECS transmission, and (S)MECS timestamp synchronization for received MECS, is disabled when this register is 0.

[1] The reset value of this field is implementation specific.

[2] The reset value of this field is implementation specific.

[3] The reset value of this field is implementation specific.

7.9.5 MECS Next Timestamp MSW CSR (Block Offset 0x18)

On an (S)MECS Master, this register contains the time when the next (S)MECS shall be transmitted. On an (S)MECS Slave, this register contains the timestamp value used to update the Timestamp Generator MSW CSR when the next (S)MECS is received. This register is updated whenever an (S)MECS is received by an MECS Slave, or when an (S)MECS is transmitted by an (S)MECS Master. For more information on the use and operation of this register, refer to Section 6.5.3.6, "MECS Time Synchronization Protocol". The use and meaning of the bit fields of this register shall be as specified in Table 7-42. The bit fields in this register are read/write.

Table 7-42. Bit Settings for MECS Next Timestamp MSW CSR

Bit	Name	Reset Value	Description
0-31	MSW Bits	0x00000000	Most significant 32 bits for the timestamp value used to update the Timestamp Generator MSW CSR when a Multicast Event Control Symbol is received by an MECS Slave. Most significant 32 bits of the timestamp value compared with the Timestamp Generator value to determine when a Multicast Event Control Symbol must be transmitted by an MECS Master.

[1]The reset value of this field is implementation specific.

7.9.6 MECS Next Timestamp LSW CSR (Block Offset 0x1C)

On an (S)MECS Master, this register contains the time when the next (S)MECS shall be transmitted. On an (S)MECS Slave, this register contains the timestamp value used to update the Timestamp Generator LSW CSR when the next (S)MECS is received. This register is updated whenever an (S)MECS is received by an (S)MECS Slave, or when an (S)MECS is transmitted by an (S)MECS Master.

For more information on the use of this register, refer to Section 6.5.3.6, "MECS Time Synchronization Protocol". The use and meaning of the bit fields of this register shall be as specified in Table 7-43. The bit fields in this register are read/write.

Table 7-43. Bit Settings for MECS Next Timestamp LSW CSR

Bit	Name	Reset Value	Description
0-31	LSW Bits	0x00000000	Least significant 32 bits for the timestamp value used to update the Timestamp Generator LSW CSR when a Multicast Event Control Symbol is received by an MECS Slave. Least significant 32 bits of the timestamp value compared with the Timestamp Generator value to determine when a Multicast Event Control Symbol shall be transmitted by an MECS Master.

7.9.7 Timestamp Generator MSW CSR (Block Offset 0x034)

This register indicates the most significant 32 bits of the Timestamp Generator. The use and meaning of the bits and bit fields of this register shall be as specified in Table 7-44. The bits and bit fields in this register are read/write.

Table 7-44. Bit Settings for Timestamp Generator MSW CSR

Bit	Name	Reset Value	Description
0-31	MSW Bits	0x00000000	Most significant 32 bits for the timestamp generator.

7.9.8 Timestamp Generator LSW CSR (Block Offset 0x038)

This register indicates the least significant 32 bits for the Timestamp Generator. The use and meaning of the bits and bit fields of this register shall be as specified in Table 7-45. The bits and bit fields in this register are read/write.

Table 7-45. Bit Settings for Timestamp Generator LSW CSR

Bit	Name	Reset Value	Description
0-31	LSW Bits	0x00000000	Least significant 32 bits for the timestamp generator.

7.9.9 Port *n* Timestamp 0 MSW CSRs (Block Offsets 0x44, 0x84, ..., 0x404)

These registers contain the value of the Timestamp Generator MSW CSR when a Loop-Timing-Request control symbol is transmitted. The use and meaning of the bits and bit fields of these registers shall be as specified in Table 7-46. The bits and bit fields in these registers are read only.

Table 7-46. Bit Settings for Port n Timestamp 0 MSW CSRs

Bit	Name	Reset Value	Description
0-31	MSW Bits	0x00000000	Most significant 32 bits from the timestamp generator.

7.9.10 Port *n* Timestamp 0 LSW CSRs (Block Offsets 0x48, 0x88, ..., 0x408)

These registers contain the value of the Timestamp Generator LSW CSR when a Loop-Timing-Request control symbol is transmitted. The use and meaning of the bits and bit fields of these registers shall be as specified in Table 7-47. The bits and bit fields in these registers are read only.

Table 7-47. Bit Settings for Port n Timestamp 0 LSW CSRs

Bit	Name	Reset Value	Description
0-31	LSW Bits	0x00000000	Least significant 32 bits from the timestamp generator.

7.9.11 Port *n* Timestamp 1 MSW CSRs (Block Offsets 0x54, 0x94, ..., 0x414)

These registers contain the value of the Timestamp Generator MSW CSR when a Loop Response control symbol is received. The use and meaning of the bits and bit fields of these registers shall be as specified in Table 7-48. The bits and bit fields in these registers are read only.

Table 7-48. Bit Settings for Port n Timestamp 1 MSW CSRs

Bit	Name	Reset Value	Description
0-31	MSW Bits	0x00000000	Most significant 32 bits from the timestamp generator.

7.9.12 Port *n* Timestamp 1 LSW CSRs (Block Offsets 0x58, 0x98, ..., 0x418)

These registers contain the value of the Timestamp Generator LSW CSR when a Loop Response control symbol is received. The use and meaning of the bits and bit fields of these registers shall be as specified in Table 7-49. The bits and bit fields in these registers are read only.

Table 7-49. Bit Settings for Port n Timestamp 0 LSW CSRs

Bit	Name	Reset Value	Description
0-31	LSW Bits	0x00000000	Least significant 32 bits from the timestamp generator.

7.9.13 Port *n* Timestamp Generator Synchronization CSRs (Block Offsets 0x60, 0xA0, ..., 0x420)

These registers control the Timestamp Generator Synchronization capabilities that the port will accept and transmit. The columns in Table 7-50 determine which fields must be implemented, based on the bit field values of the Timestamp CAR.

The use and meaning of the bits and bit fields of these registers shall be as specified in Table 7-50. The bits and bit fields in these registers are read/write.

Table 7-50. Bit Settings for Port n Timestamp Generator Synchronization CSRs

Bit	Name	Reset Value	Description	Time Slave	Time Master	Com. Freq.
0	Accept Timestamps	0b0	Indicates whether the device will accept Timestamp Control Symbols from the link partner. 0b0 - Device will not accept Timestamp Control Symbols from the link partner. 0b1 - Device accepts Timestamp Control Symbols from the link partner.	X	-	-
1	Disable Clock Compensation Sequence	0b0	Controls whether the device will transmit Clock Compensation Sequences. 0b0 - Device transmits clock compensation sequences regularly as required 0b1 - Device does not transmit clock compensation sequences.	-	-	X
2	Auto-update Link Partner Timestamp Generators	0b0	Controls whether the device will automatically update the timestamp generator of the link partner connected to this port if the timestamp generator on this device is set. 0b0 - Do not automatically update the link partner timestamp generator 0b1 - Automatically update the link partner timestamp generator whenever the timestamp generator on this device is set.	-	X	-
3-5	---	0x00	Reserved	-	-	-
6-7	Port Operating Mode	0b00	When a port supports both time slave and master capabilities, this bit is used to control the port’s operating mode. 0b00 - Master and slave functionality disabled 0b01 - Time slave functionality enabled 0b10 - Time master functionality enabled 0b11 - Reserved	X	X	-
8-18	---	0x00	Reserved	-	-	-

Table 7-50. Bit Settings for Port n Timestamp Generator Synchronization CSRs

Bit	Name	Reset Value	Description	Time Slave	Time Master	Com. Freq.
19	Tx Has Lower Latency	See Footnote 1	Indicates whether the transmit path has lower latency than the receive path, or vice versa. This value controls how the Asymmetry field is applied to loop delay calculations. 0b0 - Tx has higher latency than Rx. 0b1 - Tx has lower latency than Rx.	X	X	-
20-31	Asymmetry	See Footnote 1	Measure of the latency difference between the receive path and transmit path of this port. The value represents the number of nanoseconds. 0x000 - No difference between transmit and receive control path latency. 0x001 - One nanosecond difference between transmit and receive control path latency ... 0xFFF - 4095 nanoseconds difference between transmit and receive control path latency.	X	X	-

1 The reset value of this field is implementation specific.

7.9.14 Port *n* Auto Update Counter CSRs (Block Offsets 0x64, 0xA4, ..., 0x424)

These registers determine how often a timestamp generator master updates the link partner's timestamp generator. This is done on a per port basis since each link partner may have different tolerances/requirements for timestamp updates. The interval allows the link partner to be updated with intervals that range from once a microsecond to once an hour.

Periodic timestamp updates shall not be sent when these registers are 0.

The use and meaning of the bits and bit fields of these registers shall be as specified in Table 7-51. The bits and bit fields in these registers are read/write.

Table 7-51. Bit Settings for Port n Auto Update Counter CSRs

Bit	Name	Reset Value	Description
0-31	Update Period	0x00000000	Time between timestamp updates. Units are 1024 nanoseconds.

7.9.15 Port *n* Timestamp Synchronization Command CSRs (Block Offsets 0x68, 0xA8, ..., 0x428)

These registers enable Loop-Timing Request control symbols to be sent to the link partner. They also allow a sequence of Timestamp Control Symbols to be sent to the link partner to set the link partner’s timestamp generator.

The use and meaning of the bits and bit fields of these registers shall be as specified in Table 7-52. The bits and bit fields in these registers are read/write.

Table 7-52. Bit Settings for Port n Timestamp Synchronization Command CSRs

Bit	Name	Reset Value	Description
0-22	---	0x00	Reserved
23	Send Zero Timestamp	0b0	A port shall transmit a sequence of Timestamp Control Symbols when this field is written with a value of 1 and the Port Operating Mode is set to 0b10 (Master Enabled). The Timestamp Control Symbols shall carry a value of zero for all timestamp generator bits.
24-26	---	0x00	Reserved
27	Send Timestamp	0b0	A port shall transmit a sequence of Timestamp Control Symbols when this field is written with a value of 1 and the Port Operating Mode is set to 0b10 (Master Enabled). The Timestamp Control Symbols shall carry the value of the current timestamp generator, with the addition of the Port n Timestamp Offset CSR
28	---	0x00	Reserved
29-31	Command	0b000	Contents of the “Cmd” field of a Timing control symbol to send to the link partner. Legal values are: 0b000 - Send Multicast Event Control Symbol 0b001 - Send Secondary Multicast Event Control Symbol 0b011 - Send Loop-Timing Request Control Symbol

7.9.16 Port *n* Timestamp Synchronization Status CSRs (Block Offsets 0x6C, 0xAC, ..., 0x42C)

These registers contain the status of pending commands sent using the Port n Timestamp Synchronization Command CSR.

The use and meaning of the bits and bit fields of these registers shall be as specified in Table 7-53. The bits and bit fields in these registers are read only.

Table 7-53. Bit Settings for Port n Timestamp Synchronization Status CSRs

Bit	Name	Reset Value	Description
0	Response_valid	0b0	If the value written to the Command field of the Port *n* Timestamp Synchronization Command CSR causes a loop-response, this bit indicates that the loop-response has been received and the status fields are valid. If the value written to the Command field of the Port *n* Timestamp Synchronization Command CSR does not cause a loop-response, then this bit indicates that the request has been transmitted. This bit automatically clears on read.
1-21	---	0x00	Reserved
22-31	Delay	0x000	Contents of the “Delay” field of the Link Response control symbol: This field shall be valid when a loop-timing request was transmitted and the response_valid field is 1. A value of 0x3FF indicates that the delay in the link partner exceeded 1022 nsec.

7.9.17 Port *n* Timestamp Offset CSRs (Block Offsets 0x70, 0xB0, ..., 0x430)

These registers contain the number of nanoseconds to add to the current Timestamp Generator value before sending a sequence of Timestamp control symbols to the link partner. The use and meaning of the bits and bit fields of these registers shall be as specified in Table 7-54. The bits and bit fields in these register are read/write.

Table 7-54. Bit Settings for Port n Timestamp Offset CSRs

Bit	Name	Reset Value	Description
0-15	Offset	0x0000	Count of the number of nanoseconds to add to the timestamp generator value when transmitting a sequence of Timestamp Control Symbols.
16-31	---	0x0000	Reserved

7.10 Miscellaneous Physical Layer Extension Block

The Miscellaneous Physical Layer Extension Block is optional. The Miscellaneous Physical Layer Extension Block contains different registers depending on the Miscellaneous Physical Layer Extension Block CAR field values as defined in Table 7-57.

The “SAL” column in Table 7-55 indicates which registers shall be implemented when the “SAL Support” bit is 1. The “SMECS” column indicates which registers shall be implemented when the “SMECS Support” bit is 1. The “PRBS” column indicates which registers shall be implemented when the “PRBS Support” bit is 1. In all cases, an “X” in a column means that the register shall be implemented.

The Miscellaneous Physical Layer Extension Block shall not be implemented if the “SAL Support”, “SMECS Support”, and “PRBS Support” bits are all 0.

Table 7-55. Miscellaneous Physical Layer Extension Block

	Block Byte Offset	Register Name	SAL	SMECS	PRBS
Header	0x00	Miscellaneous Physical Layer Extension Block Header	X	X	X
	0x04	Miscellaneous Physical Layer CAR	X	X	X
	0x08-3C	Reserved	-	-	-
Port 0	0x40	Port 0 Port Reinit Control CSR	X	-	X
	0x44	Port 0 SAL Control and Status CSR	X	-	-
	0x48	Port 0 SMECS Control CSR	-	X	-
	0x4C	Port 0 PRBS Control CSR	-	-	X
	0x50	Port 0 PRBS Lane Control CSR			X
	0x54	Port 0 PRBS Status 0 CSR	-	-	X
	0x58	Port 0 PRBS Status 1 CSR	-	-	X
	0x5C	Port 0 PRBS Locked Time CSR	-	-	X
	0x60	Port 0 PRBS Seed CSR	-	-	X
	0x64-7C	Reserved	-	-	-
Ports 1-14	0x80–3FC	Assigned to Port 1-14 CSRs			

Table 7-55. Miscellaneous Physical Layer Extension Block

	Block Byte Offset	Register Name	SAL	SMECS	PRBS
Port 15	0x440	Port 15 Port Reinit Control CSR	X	-	X
	0x444	Port 15 SAL Control and Status CSR	X	-	-
	0x448	Port 15 SMECS Control CSR	-	X	-
	0x44C	Port 15 PRBS Control CSR	-	-	X
	0x450	Port 15 PRBS Lane Control CSR	-	-	X
	0x454	Port 15 PRBS Status 0 CSR	-	-	X
	0x458	Port 15 PRBS Status 1 CSR	-	-	X
	0x45C	Port 15 PRBS Locked Time CSR	-	-	X
	0x460	Port 15 PRBS Seed CSR	-	-	X
	0x464-47C	Reserved	-	-	-

7.10.1 Miscellaneous Physical Layer Extension Block Header (Block Offset 0x0)

The Miscellaneous Physical Layer Extension Block Header register contains the EF_PTR to the next EF_BLK and the EF_ID that identifies this as the Miscellaneous Physical Layer Block Header. The use and meaning of the bit fields of this register shall be as specified in Table 7-56. The register is read-only.

Table 7-56. Bit Settings for Miscellaneous Physical Layer Extension Block Header

Bit	Name	Reset Value	Description
0-15	EF_PTR		Hard wired pointer to the next block in the data structure, if one exists
16-31	EF_ID	0x0010	Hard wired Extended Features ID

7.10.2 Miscellaneous Physical Layer CAR (Block Offset 0x04)

This register indicates which Timestamp Synchronization capabilities the device supports. The use and meaning of the bit fields of this register shall be as specified in Table 7-39. The bit fields in this register are read only.

Table 7-57. Bit Settings for Miscellaneous Physical Layer CAR

Bit	Name	Reset Value	Description
0	SAL Support	See Footnote 1	Indicates whether the device supports Structurally Asymmetric Links 0b0 - Device does not support Structurally Asymmetric Links 0b1 - Device supports Structurally Asymmetric Links
1	SMECS Support	See Footnote 1	Indicates whether the device supports Secondary Multicast Event Control Symbols (SMECS) 0b0 - Device does not support SMECS 0b1 - Device supports SMECS
2	PRBS Support	See Footnote 1	Indicates whether the device supports standard Pseudo Random Binary Sequence (PRBS) testing 0b0 - Device does not support standard PRBS testing 0b1 - Device supports standard PRBS testing
3-31	---	0	Reserved

1 The reset value of this field is implementation specific.

7.10.3 Port *n* Reinit Control CSR (Block Offset 0x40, 0x80, 0xC0,..., 0x440)

This register shall be implemented whenever at least one of the Miscellaneous Physical Layer CAR “SAL Support” and “PRBS Support” bits are set. If the SAL Support and Diagnostic Support bits are clear this register shall be reserved. The use and meaning of the bit fields of this register shall be as specified in Table 7-58. The bit fields in this register are read/write.

Table 7-58. Bit Settings for Port *n* Reinit Control CSR

Bit	Name	Reset Value	Description
0-12	-	0	Reserved
13-15	Silence Count	0	When non-zero, decremented each time the port initialization state machine enters the SILENT state. Structurally Asymmetric Link operation and/or PRBS operation may be enabled when this field is non-zero.
16-30	-	0	Reserved
31	Pulse Force-Reinit	0	When written with 1, causes the port initialization state machine to enter the SILENT state. Always reads as 0.

7.10.4 Port *n* SAL Control and Status CSR (Block Offset 0x44, 0x84, 0xC4,..., 0x444)

This register shall be implemented whenever the Miscellaneous Physical Layer CAR “SAL Support” bit is set. If the SAL Support bit is clear this register shall be reserved. The use and meaning of the bit fields of this register shall be as specified in Table 7-59. Except where noted below, the bit fields in this register are read/write.

Table 7-59. Bit Settings for Port *n* SAL Control and Status CSR

Bit	Name	Reset Value	Description
0	SAL Enabled	0	Status of Structurally Asymmetric Link operation: 0 - SAL is not active 1 - SAL is active SAL Enabled shall be set when the port transitions to the SILENT state and Silence Count is greater than 0. SAL Enabled shall be cleared when the Silence Count value is 0. This bit is read-only.
1-11	-	0	Reserved
12-15	SAL RX Width	0	When SAL Enabled is set, this field controls the receive operating width of the port. This field is encoded as follows: 0b0000 - No override 0b0001 - 1x, lane 0 0b0010 - 1x, lane 1 0b0011 - 1x, lane 2 0b0100 - 1x, lane 3 0b0101 - 2x, lanes 0 & 1. Lanes 2 and 3 are not used. 0b0110 - 2x, lanes 2 & 3 0b0111 - 4x, lanes 0-3 0b1000 - 8x, lanes 0-7 0b1001 - 16x 0b1010-0b1011 - Implementation specific 0b1100-0b1111 - Reserved
16-27	-	0	Reserved
28-31	SAL TX Width	0	When SAL Enabled is set, this field controls the transmit operating width of the port. This field is encoded as follows: 0b0000 - No override 0b0001 - 1x, lane 0. Disable lanes 1, 2, and 3. 0b0010 - 1x, lane 1. Disable lanes 0, 2 and 3. 0b0011 - 1x, lane 2. Disable lanes 0, 1, and 3. 0b0100 - 1x, lane 3. Disable lanes 0, 1, and 2. Transmit Lane 0 compliant data on lane 3. 0b0101 - 2x, lanes 0 & 1. Disable lanes 2 and 3. 0b0110 - 2x, lanes 2 & 3. Transmit lane 0 and 1 2x compliant data streams on lanes 2 and 3. Disable transmission on lanes 0 and 1. 0b0111 - 4x, lanes 0-3 0b1000 - 8x, lanes 0-7 0b1001 - 16x. 0b1010-0b1011 - Implementation specific 0b1100-0b1111 - Reserved

7.10.5 Port *n* SMECS Control CSR (Block Offset 0x48, 0x88, 0xC8,..., 0x448)

This register shall be implemented whenever the Miscellaneous Physical Layer CAR “SMECS Support” bit is set. If the SMECS Support bit is clear this register shall be reserved. The use and meaning of the bit fields of this register shall be as specified in Table 7-60. The bit fields in this register are read/write.

Table 7-60. Bit Settings for Port *n* SMECS Control CSR

Bit	Name	Reset Value	Description
0	Secondary Multicast-Event Participant	0	Retransmit incoming Secondary Multicast-event control symbols out this port (multiple port devices only)
1-31	-	0	Reserved

7.10.6 Port *n* PRBS Control CSR (Block Offset 0x4C, 0x8C, 0xCC,..., 0x44C)

This register shall be implemented whenever the Miscellaneous Physical Layer CAR “PRBS Support” bit is set. If the PRBS Support bit is clear this register shall be reserved. The use and meaning of the bit fields of this register shall be as specified in Table 7-61. The bit fields in this register are read/write.

Table 7-61. Bit Settings for Port *n* PRBS Control CSR

Bit	Name	Reset Value	Description
0	PRBS Active	0	Indicates whether a PRBS test is in progress on this port. 0b0 - PRBS test is not active 0b1 - PRBS test is active This bit shall be read only.
1	PRBS Completed	0	Indicates whether a PRBS test has been performed on this port. 0b0 - PRBS test has not been performed on this port 0b1 - PRBS test has been performed on this port This bit shall be cleared when '1' is written to this field.
2-6	PRBS Pattern Selection	0	When PRBS Active is set, this field controls the PRBS pattern that is transmitted and checked by this port. 0b00000 - Diagnostics are disabled 0b00001 - Transmit and check X^7+X^6+1. This pattern shall be supported by devices operating at Baud Rate Class 1 speeds. 0b00010 - Transmit and check X^9+X^5+1. This pattern shall be supported by devices operating at Baud Rate Class 2 and/or Baud Rate Class 3 lane speeds. 0b00011 - Transmit and check $X^{15}+X^{14}+1$. This pattern shall be supported by devices operating at Baud Rate Class 2 and/or Baud Rate Class 3 lane speeds. 0b00100 - Transmit and check $X^{23}+X^{18}+1$. This pattern shall be supported by devices operating at Baud Rate Class 2 and/or Baud Rate Class 3 lane speeds. 0b00101 - Transmit and check $X^{31}+X^{28}+1$. This pattern shall be supported by devices operating at Baud Rate Class 2 and/or Baud Rate Class 3 lane speeds. 0b00110 - Reserved 0b00111 - Reserved 0b01000-0b01111 - Implementation Specific 0b10000-0b11111 - Reserved Attempting to set the PRBS Pattern Selection value to an unsupported value shall result in a programmed value of 0b00000.

Table 7-61. Bit Settings for Port *n* PRBS Control CSR

Bit	Name	Reset Value	Description
7-15	PRBS Lock Interval Threshold	0	The period of time for which a PRBS sequence must be received error free before declaring PRBS lock. 0 - Lock immediately 1 - One Silence Timer period 2 - Two Silence Timer periods ... 0x1FF - 511 Silence Timer periods Note: A Silence Timer period is defined as the period of time required to cause “silence_tmr_done” to be asserted.
16-31	PRBS Test Interval	0	The period of time that a PRBS sequence must be transmitted before declaring the diagnostic complete. The granularity of this field is a PRBS Test Interval Tick. If an implementation supports the Port n Link Timers Control 2 CSR “Discovery Completion Timer” field, a Test Interval Tick shall be the same as the Discovery Completion Timer period. If an implementation does not support the Discovery Completion Timer field, as Test Interval Tick shall be one second, +/- 33%. 0x0000 - 65,536 PRBS Test Interval Ticks 0x0001 - One PRBS Test Interval Tick 0x0002 - Two PRBS Test Interval Ticks 0x0003 - Three PRBS Test Interval Ticks … 0xFFFF - 65,535 PRBS Test Interval Ticks

7.10.7 Port *n* PRBS Lane Control CSR (Block Offset 0x50, 0x90, 0xD0,..., 0x450)

This register shall be implemented whenever the Miscellaneous Physical Layer CAR “PRBS Support” bit is set. If the PRBS Support bit is clear this register shall be reserved. The use and meaning of the bit fields of this register shall be as specified in Table 7-62. The bit fields in this register are read/write.

Table 7-62. Bit Settings for Port *n* PRBS Lane Control CSR

Bit	Name	Reset Value	Description
0-15	PRBS Transmit Lane Control	0	Bit vector of lanes. When a bit is set, the PRBS pattern shall be transmitted on the corresponding lane. 0x0001 - Transmit PRBS on lane 0 0x0002 - Transmit PRBS on lane 1 0x0004 - Transmit PRBS on lane 2 … 0x8000 - Transmit PRBS on lane 15 Bits corresponding to lanes greater than the maximum transmit port width shall be reserved. It shall be possible to set all supported bits in any combination.
16-31	PRBS Receive Lane Control	0	Bit vector of lanes. When a bit is set, the PRBS pattern shall be checked on the corresponding lane. 0x0001 - Check PRBS on lane 0 0x0002 - Check PRBS on lane 1 0x0004 - Check PRBS on lane 2 … 0x8000 - Check PRBS on lane 15 Bits corresponding to lanes greater than the maximum receive port width shall be reserved. It shall be possible to set one bit at a time within this field. It may be possible to set more than one bit simultaneously within this field.

7.10.8 Port *n* PRBS Status 0 CSR (Block Offset 0x54, 0x94, 0xD4,..., 0x454)

This register shall be implemented whenever the Miscellaneous Physical Layer CAR “PRBS Support” bit is set. If the PRBS Support bit is clear this register shall be reserved. The use and meaning of the bit fields of this register shall be as specified in Table 7-63. The bit fields in this register are read only.

Table 7-63. Bit Settings for Port *n* PRBS Status 0 CSR

Bit	Name	Reset Value	Description
0	Lane 7 PRBS Lock Status	0	Indicates whether the PRBS checker for Lane 7 was able to achieve PRBS lock during the last PRBS test. 0 - PRBS Lock has not been achieved. 1 - PRBS Lock has been achieved. This field shall be cleared at the start of any PRBS test for the port. This field shall be reserved if the maximum port with is 4x or less.
1-3	Lane 7 PRBS Error Count	0	Saturating count of PRBS errors detected on Lane 7 during the last PRBS test. 0b000 - No errors were detected 0b001 - One error was detected 0b010 - Two errors were detected … 0b111 - At least seven errors were detected This field shall be cleared at the start of any PRBS test for the port. This field shall be reserved if the maximum port width is 4x or less.
4	Lane 6 PRBS Lock Status	0	Indicates whether the PRBS checker for Lane 6 was able to achieve PRBS lock during the last PRBS test. 0 - PRBS Lock has not been achieved. 1 - PRBS Lock has been achieved. This field shall be cleared at the start of any PRBS test for the port. This field shall be reserved if the maximum port width is 4x or less.
5-7	Lane 6 PRBS Error Count	0	Saturating count of PRBS errors detected on Lane 6 during the last PRBS test. 0b000 - No errors were detected 0b001 - One error was detected 0b010 - Two errors were detected … 0b111 - At least seven errors were detected This field shall be cleared at the start of any PRBS test for the port. This field shall be reserved if the maximum port width is 4x or less.
8	Lane 5 PRBS Lock Status	0	Indicates whether the PRBS checker for Lane 5 was able to achieve PRBS lock during the last PRBS test. 0 - PRBS Lock has not been achieved. 1 - PRBS Lock has been achieved. This field shall be cleared at the start of any PRBS test for the port. This field shall be reserved if the maximum port width is 4x or less.

Table 7-63. Bit Settings for Port *n* PRBS Status 0 CSR

Bit	Name	Reset Value	Description
9-11	Lane 5 PRBS Error Count	0	Saturating count of PRBS errors detected on Lane 5 during the last PRBS test. 0b000 - No errors were detected 0b001 - One error was detected 0b010 - Two errors were detected … 0b111 - At least seven errors were detected This field shall be cleared at the start of any PRBS test for the port. This field shall be reserved if the maximum port width is 4x or less.
12	Lane 4 PRBS Lock Status	0	Indicates whether the PRBS checker for Lane 4 was able to achieve PRBS lock during the last PRBS test. 0 - PRBS Lock has not been achieved. 1 - PRBS Lock has been achieved. This field shall be cleared at the start of any PRBS test for the port. This field shall be reserved if the maximum port width is 4x or less.
13-15	Lane 4 PRBS Error Count	0	Saturating count of PRBS errors detected on Lane 4 during the last PRBS test. 0b000 - No errors were detected 0b001 - One error was detected 0b010 - Two errors were detected … 0b111 - At least seven errors were detected This field shall be cleared at the start of any PRBS test for the port. This field shall be reserved if the maximum port width is 4x or less.
16	Lane 3 PRBS Lock Status	0	Indicates whether the PRBS checker for Lane 3 was able to achieve PRBS lock during the last PRBS test. 0 - PRBS Lock has not been achieved. 1 - PRBS Lock has been achieved. This field shall be cleared at the start of any PRBS test for the port. This field shall be reserved if the maximum port width is 2x or less.
17-19	Lane 3 PRBS Error Count	0	Saturating count of PRBS errors detected on Lane 3 during the last PRBS test. 0b000 - No errors were detected 0b001 - One error was detected 0b010 - Two errors were detected … 0b111 - At least seven errors were detected This field shall be cleared at the start of any PRBS test for the port. This field shall be reserved if the maximum port width is 2x or less.
20	Lane 2 PRBS Lock Status	0	Indicates whether the PRBS checker for Lane 2 was able to achieve PRBS lock during the last PRBS test. 0 - PRBS Lock has not been achieved 1 - PRBS Lock has been achieved This field shall be cleared at the start of any PRBS test for the port. This field shall be reserved if the maximum port width is 2x or less.

Table 7-63. Bit Settings for Port *n* PRBS Status 0 CSR

Bit	Name	Reset Value	Description
21-23	Lane 2 PRBS Error Count	0	Saturating count of PRBS errors detected on Lane 2 during the last PRBS test. 0b000 - No errors were detected 0b001 - One error was detected 0b010 - Two errors were detected … 0b111 - At least seven errors were detected This field shall be cleared at the start of any PRBS test for the port. This field shall be reserved if the maximum port width is 2x or less.
24	Lane 1 PRBS Lock Status	0	Indicates whether the PRBS checker for Lane 1 was able to achieve PRBS lock during the last PRBS test. 0 - PRBS Lock has not been achieved 1 - PRBS Lock has been achieved This field shall be cleared at the start of any PRBS test for the port. This field shall be reserved if the maximum port width is 1x.
25-27	Lane 1 PRBS Error Count	0	Saturating count of PRBS errors detected on Lane 1 during the last PRBS test. 0b000 - No errors were detected 0b001 - One error was detected 0b010 - Two errors were detected … 0b111 - At least seven errors were detected This field shall be cleared at the start of any PRBS test for the port. This field shall be reserved if the maximum port width is 1x.
28	Lane 0 PRBS Lock Status	0	Indicates whether the PRBS checker for Lane 0 was able to achieve PRBS lock during the last PRBS test. 0 - PRBS Lock has not been achieved 1 - PRBS Lock has been achieved This field shall be cleared at the start of any PRBS test for the port.
29-31	Lane 0 PRBS Error Count	0	Saturating count of PRBS errors detected on Lane 0 during the last PRBS test. 0b000 - No errors were detected 0b001 - One error was detected 0b010 - Two errors were detected … 0b111 - At least seven errors were detected This field shall be cleared at the start of any PRBS test for the port.

7.10.9 Port *n* PRBS Status 1 CSR (Block Offset 0x58, 0x98, 0xD8,..., 0x458)

This register shall be implemented whenever the Miscellaneous Physical Layer CAR “PRBS Support” bit is set and the maximum port width is 16x. If the PRBS Support bit is clear or the maximum port width is not 16x this register shall be reserved. The use and meaning of the bit fields of this register shall be as specified in Table 7-64. The bit fields in this register are read only.

Table 7-64. Bit Settings for Port *n* PRBS Status 1 CSR

Bit	Name	Reset Value	Description
0	Lane 15 PRBS Lock Status	0	Indicates whether the PRBS checker for Lane 15 was able to achieve PRBS lock during the last PRBS test. 0 - PRBS Lock has not been achieved 1 - PRBS Lock has been achieved This field shall be cleared at the start of any PRBS test for the port.
1-3	Lane 15 PRBS Error Count	0	Saturating count of PRBS errors detected on Lane 15 during the last PRBS test. 0b000 - No errors were detected 0b001 - One error was detected 0b010 - Two errors were detected … 0b111 - At least seven errors were detected This field shall be cleared at the start of any PRBS test for the port.
4	Lane 14 PRBS Lock Status	0	Indicates whether the PRBS checker for Lane 14 was able to achieve PRBS lock during the last PRBS test. 0 - PRBS Lock has not been achieved 1 - PRBS Lock has been achieved This field shall be cleared at the start of any PRBS test for the port.
5-7	Lane 14 PRBS Error Count	0	Saturating count of PRBS errors detected on Lane 14 during the last PRBS test. 0b000 - No errors were detected 0b001 - One error was detected 0b010 - Two errors were detected … 0b111 - At least seven errors were detected This field shall be cleared at the start of any PRBS test for the port.
8	Lane 13 PRBS Lock Status	0	Indicates whether the PRBS checker for Lane 13 was able to achieve PRBS lock during the last PRBS test. 0 - PRBS Lock has not been achieved 1 - PRBS Lock has been achieved This field shall be cleared at the start of any PRBS test for the port.
9-11	Lane 13 PRBS Error Count	0	Saturating count of PRBS errors detected on Lane 13 during the last PRBS test. 0b000 - No errors were detected 0b001 - One error was detected 0b010 - Two errors were detected … 0b111 - At least seven errors were detected This field shall be cleared at the start of any PRBS test for the port.

Table 7-64. Bit Settings for Port *n* PRBS Status 1 CSR

Bit	Name	Reset Value	Description
12	Lane 12 PRBS Lock Status	0	Indicates whether the PRBS checker for Lane 12 was able to achieve PRBS lock during the last PRBS test. 0 - PRBS Lock has not been achieved 1 - PRBS Lock has been achieved This field shall be cleared at the start of any PRBS test for the port.
13-15	Lane 12 PRBS Error Count	0	Saturating count of PRBS errors detected on Lane 12 during the last PRBS test. 0b000 - No errors were detected 0b001 - One error was detected 0b010 - Two errors were detected … 0b111 - At least seven errors were detected This field shall be cleared at the start of any PRBS test for the port.
16	Lane 11 PRBS Lock Status	0	Indicates whether the PRBS checker for Lane 11 was able to achieve PRBS lock during the last PRBS test. 0 - PRBS Lock has not been achieved 1 - PRBS Lock has been achieved This field shall be cleared at the start of any PRBS test for the port.
17-19	Lane 11 PRBS Error Count	0	Saturating count of PRBS errors detected on Lane 11 during the last PRBS test. 0b000 - No errors were detected 0b001 - One error was detected 0b010 - Two errors were detected … 0b111 - At least seven errors were detected This field shall be cleared at the start of any PRBS test for the port.
20	Lane 10 PRBS Lock Status	0	Indicates whether the PRBS checker for Lane 10 was able to achieve PRBS lock during the last PRBS test. 0 - PRBS Lock has not been achieved 1 - PRBS Lock has been achieved This field shall be cleared at the start of any PRBS test for the port.
21-23	Lane 10 PRBS Error Count	0	Saturating count of PRBS errors detected on Lane 10 during the last PRBS test. 0b000 - No errors were detected 0b001 - One error was detected 0b010 - Two errors were detected … 0b111 - At least seven errors were detected This field shall be cleared at the start of any PRBS test for the port.
24	Lane 9 PRBS Lock Status	0	Indicates whether the PRBS checker for Lane 9 was able to achieve PRBS lock during the last PRBS test. 0 - PRBS Lock has not been achieved 1 - PRBS Lock has been achieved This field shall be cleared at the start of any PRBS test for the port.
25-27	Lane 9 PRBS Error Count	0	Saturating count of PRBS errors detected on Lane 9 during the last PRBS test. 0b000 - No errors were detected 0b001 - One error was detected 0b010 - Two errors were detected … 0b111 - At least seven errors were detected This field shall be cleared at the start of any PRBS test for the port.

Table 7-64. Bit Settings for Port *n* PRBS Status 1 CSR

Bit	Name	Reset Value	Description
28	Lane 8 PRBS Lock Status	0	Indicates whether the PRBS checker for Lane 8 was able to achieve PRBS lock during the last PRBS test. 0 - PRBS Lock has not been achieved 1 - PRBS Lock has been achieved This field shall be cleared at the start of any PRBS test for the port.
29-31	Lane 8 PRBS Error Count	0	Saturating count of PRBS errors detected on Lane 8 during the last PRBS test. 0b000 - No errors were detected 0b001 - One error was detected 0b010 - Two errors were detected … 0b111 - At least seven errors were detected This field shall be cleared at the start of any PRBS test for the port.

7.10.10 Port *n* PRBS Locked Time CSR (Block Offset 0x5C, 0x9C, 0xDC,..., 0x45C)

This register shall be implemented whenever the Miscellaneous Physical Layer CAR “PRBS Support” bit is set. If the PRBS Support bit is clear this register shall be reserved. The use and meaning of the bit fields of this register shall be as specified in Table 7-65. The bit fields in this register are read only.

Table 7-65. Bit Settings for Port *n* PRBS Locked Time CSR

Bit	Name	Reset Value	Description
0-15	All PRBS Locked Time	0	This field contains the count of the full and partial PRBS Test Interval Ticks that occurred after all lanes selected in the “PRBS Receive Lane Control” field have asserted PRBS Lock. 0x0000 - Not all lanes declared PRBS Lock during the PRBS Test Interval 0x0001 - Between 0 and 1 full PRBS Test Interval Ticks elapsed after all lanes asserted PRBS Lock. 0x0002 - Between 1 and 2 PRBS Test Interval Ticks elapsed after all lanes asserted PRBS Lock … 0xFFFF - Between 65534 and 65535 PRBS Test Interval Ticks elapsed after all lanes asserted PRBS Lock. This field shall be cleared at the start of any PRBS test for the port.
16-31	-	0	Reserved

7.10.11 Port *n* PRBS Seed CSR (Block Offset 0x60, 0xA0, 0xE0,..., 0x460)

This register shall be implemented whenever the Miscellaneous Physical Layer CAR “PRBS Support” bit is set. If the PRBS Support bit is clear this register shall be reserved. The use and meaning of the bit fields of this register shall be as specified in Table 7-66.

This register defines the starting seed value used to generate the PRBS sequence.

The ability to write the bit fields in this register is optional.

Table 7-66. Bit Settings for Port *n* PRBS Seed CSR

Bit	Name	Reset Value	Description
0-31	Seed	0xFFFFFFFF	Starting seed value used for PRBS generation. Seed sizes of less than 32 bits shall be taken from the least significant bits of this register.

Chapter 8 Signal Descriptions

8.1 Introduction

This chapter contains the signal pin descriptions for a RapidIO LP-Serial port. The interface is defined either as a 1x, 2x, 4x, 8x, or 16x lane, full duplex, point-to-point interface using differential signaling. A lane implementation consists of Nx4 wires with two for the egress and two for the ingress direction. The electrical details are described in Chapter 9, "Common Electrical Specifications for less than 6.5 Gbaud LP-Serial Links" and Chapter 12, "Electrical Specification for 10.3125 and 12.5 Gbaud LP-Serial Links".

8.2 Signal Definitions

Table 8-1 provides a summary of the RapidIO LP-Serial signal pins as well as a short description of their functionality.

Table 8-1. LP-Serial Signal Description

Signal Name	I/O	Signal Meaning	Timing Comments
TD[0-(N-1)][1]	O	Transmit Data - The transmit data is a unidirectional point to point bus designed to transmit the packet information. The TD bus of one device is connected to the RD bus of the receiving device. TD[0] is used in 1x mode.	Clocking is embedded in data using 8b/10b or 64b/67b encoding.
$\overline{\text{TD}}$[0-(N-1)][1]	O	Transmit Data complement—These signals are the differential pairs of the TD signals.	
RD[0-(N-1)][1]	I	Receive Data - The receive data is a unidirectional point to point bus designed to receive the packet information. The RD bus of one device is connected to the TD bus of the receiving device. RD[0] is used in 1x mode.	Clocking is embedded in data using 8b/10b or 64b/67b encoding.
$\overline{\text{RD}}$[0-(N-1)][1]	I	Receive Data complement—These signals are the differential pairs of the RD signals.	
NOTES: 1. N has legal values of 1, 2, 4, 8, and 16			

8.3 Serial RapidIO Interface Diagrams

Figure 8-1 shows the signal interface diagram connecting two 1x devices together with the RapidIO LP-Serial interconnect.

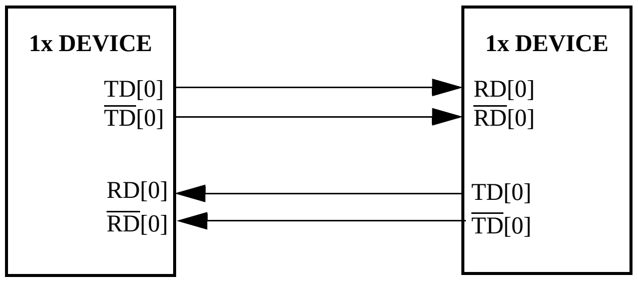


Figure 8-1. RapidIO 1x Device to 1x Device Interface Diagram

Figure 8-2 shows the signal interface diagram connecting two Nx devices together with the RapidIO LP-Serial interconnect.

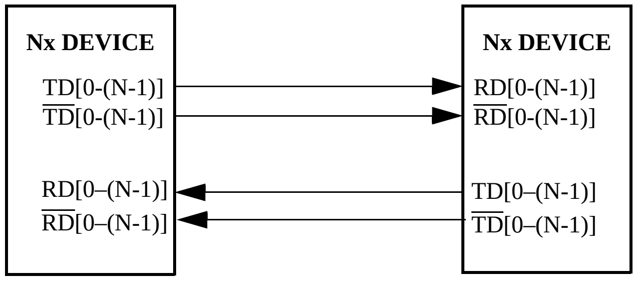


Figure 8-2. RapidIO Nx Device to Nx Device Interface Diagram

Figure 8-3 shows the connections between a Nx LP-Serial device and a 1x LP-Serial device.

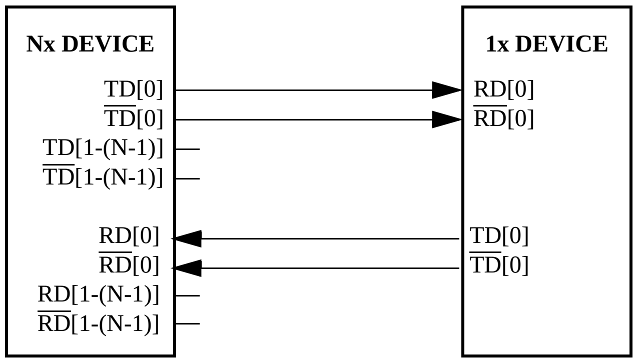


Figure 8-3. RapidIO Nx Device to 1x Device Interface Diagram

Chapter 9 Common Electrical Specifications for less than 6.5 Gbaud LP-Serial Links

9.1 Introduction

The chapter defines the common electrical specifications for the LP-Serial Physical Layer. Chapter 10, "1.25 Gbaud, 2.5 Gbaud, and 3.125 Gbaud LP-Serial Links" defines Level I links compatible with the 1.3 version of the Physical Layer Specification, that supports baud rates of 1.25 Gbaud, 2.5 Gbaud, and 3.125 Gbaud. Chapter 11, "5 Gbaud and 6.25 Gbaud LP-Serial Links" defines Level II links that support baud rates of 5 Gbaud and 6.25 Gbaud.

A Level I link shall:

- allow 1.25 Gbaud, 2.5 Gbaud, or 3.125 Gbaud baud rates
- supports AC coupling
- supports hot plug
- supports short run (SR) and long run (LR) links achieved with two transmitters
- support single receiver specification that will accept signals from both the short run and long run transmitter specifications
- achieve Bit Error Ratio of lower than 10^{-12} per lane

A Level II link shall:

- allow 5 Gbaud or 6.25 Gbaud baud rates
- supports AC coupling and optional DC coupling
- supports hot plug
- supports short run (SR), medium run (MR), and long run (LR) links achieved with two transmitters and two receivers
- achieves Bit Error Ratio of lower than 10^{-15} per lane but test requirements will be verified to 10^{-12} per lane.

Together, these specifications allow for solutions ranging from simple chip-to-chip interconnect to board-to-board interconnect driving two connectors across a backplane. The faster and wider electrical interfaces specified here are required to provide higher density and/or lower cost interfaces.

The short run defines a transmitter and a receiver that should be used mainly for chip-to-chip connections on either the same printed circuit board or across a single connector. This covers the case where connections are made to a mezzanine (daughter) card. The smaller swings of the short run specification reduces the overall power used by the transceivers.

The long run defines a transmitter and receiver that use larger voltage swings and channel equalization that allows a user to drive signals across two connectors and backplanes.

The two transmitter specifications allows for a medium run specification that also uses larger voltage swings that are capable of driving signals across a backplane but simplifies the receiver requirements to minimize power and complexity. This option has been included to allow the system integrator to deploy links that take advantage of either channel materials and/or construction techniques that reduce channel loss to achieve lower power systems.

It is also a goal of this specification to enable the inter-operability of Level I and Level II links to allow newer devices to be used with existing legacy devices.

All unit intervals are specified with a tolerance of ±100 ppm. The worst case frequency difference between any transmit and receive clock is 200 ppm.

The electrical specifications are based on loss, jitter, and channel cross-talk budgets and defines the characteristics required to communicate between a transmitter and a receiver using nominally 100Ω differential copper signal traces on a printed circuit board. Rather than specifying materials, channel components, or configurations, this specification focuses on effective channel characteristics. Hence a short length of poorer material should be equivalent to a longer length of premium material. A 'length' is effectively defined in terms of its attenuation rather than physical distance.

The RapidIO specification defines applicable data characteristics (e.g. DC balance, transition density, maximum run length), channel models and compliance points/parameters supporting the physical run and conditions.

Finally it is assumed that the link designer has taken care to minimize reflections and crosstalk so that the link can be sufficiently equalized with the transmitter and receiver chosen.

9.2 References

1. IEEE Standard 802.3ae-2002. “Standard for Information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Special Requirements. Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specification. Amendment: Media Access Control (MAC) Parameters, Physical Layers, and Management Parameters for 10 Gb/s Operation”, IEEE Std. 802.3ae-2002, August 30, 2002.

2. Optical Internetworking Forum “Common Electrical I/O (CEI) - Electrical and Jitter Interoperability Agreements for 6G+ bps and 11G+ bps I/O”, IA # OIF-CEI-02.0, January 28, 2005.
3. ITU-T Recommendation O.150 May 1996 and corrigendum May 2002. General requirements for instrumentation for performance measurements on digital transmission equipment.
4. Low Voltage Differential Swing (LVDS), ANSI/TIA/EIA-644-A-2001
5. Optical Internetworking Forum, OIF 2002.507.01 - High Speed Backplane (HSB) Interface Electrical Specification for 5-6.375Gbps Baud Rates over Currently Existing Communications Backplanes.

9.3 Abbreviations

Table 9-1. Abbreviations

Abbreviation	Meaning
BER	Bit Error Ratio
BERT	Bit Error Ratio Test or Tester
BUJ	Bounded Uncorrelated Jitter
CBGJ	Correlated Bounded Gaussian Jitter
CBHPJ	Correlated Bounded High Probability Jitter
CEI	Common Electrical I/O
CDF	Cumulative Distribution Function
CDR	Clock Data Recovery
CID	Consecutive Identical Digits
CML	Current Mode Logic
Cn	Cursor number
DCD	Duty Cycle Distortion
dB	Decibel
DDJ	Data Dependent Jitter
DFE	Decision Feedback Equalizer
DJ	Deterministic Jitter
DUT	Device Under Test
EMI	Electro-Magnetic Interference
erf	error function
erfinv	inverse error function
ESD	Electro-Static Discharge
FEXT	Far End Cross Talk
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
FR-4	Fire Retardant 4 Glass Reinforce Epoxy Laminate PCB material

Table 9-1. Abbreviations

Abbreviation	Meaning
Gbps	Giga bits per second
GJ	Gaussian Jitter
Gbaud	Giga symbols per second
HF	High Frequency
HPF	High Pass Filter
HPJ	High Probability Jitter
IA	Implementation Agreement
ISI	Inter-Symbol Interference
LMS	Least Mean Square
LPF	Low Pass Filter
LVDS [4]	Low Voltage Differential Signal
LR	Long Run
mA	milli-Amp
MR	Medium Run
mV	milli-Volt
NEXT	Near End Cross Talk
NRZ	Non Return to Zero
PCB	Printed Circuit Board
PDF	Probability Distribution Function
PECL	Positive Emitter Coupled Logic
PJ	Periodic Jitter
pp	Peak to Peak
ppd	Peak to Peak Differential (as in 300mVppd)
PLL	Phase Locked Loop
ps	pico second
PRBS	Pseudo Random Bit Stream
Q	Inverse error function
RJ	Random Jitter
RV	Random Variable
RX	Receiver
R_Zvtt	Resistance of termination to Vtt
S11 and S22	reflection coefficient
S21	transmission coefficient
SCC11 and SCC22	Common mode reflection coefficients
SCD11 and SCD22	Differential to common mode conversion coefficient
SDD11 and SDD22	Differential reflection coefficients
SDC11 and SDC22	Common mode to differential conversion coefficient

Table 9-1. Abbreviations

Abbreviation	Meaning
SJ	Sinusoidal Jitter
SR	Short Run
sym/s	symbols/second
TJ	Total Jitter
TDM	Time Division Multiplexed data
TFI	TDM Fabric to Framer Interface
TX	Transmitter
UBHPJ	Uncorrelated Bounded High Probability Jitter
UI	Unit Interval = 1/(baud rate)
UUGJ	Uncorrelated Unbounded Gaussian Jitter
Vtt	Termination Voltage
XAUI	10 Gigabit Attachment Unit Interface

9.4 Definitions

Table 9-2. General Definitions

Parameter	Description
Bit Error Ratio	A parameter that reflects the quality of the serial transmission and detection scheme. The Bit Error Ratio is calculated by counting the number of erroneous bits output by a receiver and dividing by the total number of transmitted bits over a specified transmission period.
Baud rate	Is a measure of the number of times per second a signal in a communications channel changes state. The state is usually voltage level, frequency, or phase angle. It is named after Émile Baudot, the inventor of the Baudot code for telegraphy.
Channel	In this specification Channel shall mean electrical differential channel. The channel is combination of electrical interconnects that together form the signal path from reference points T to R - see Figure 9-11. The channel will typically consist of PCB traces, via holes, component attachment pads and connectors. A characteristic of a signal channel is the complex characteristic impedance Z.
Common Mode Voltage	Average of the Vhigh and Vlow voltage levels - see Figure 9-1.
Confidence level	The use of this definition shall be understood as being with reference to a Gaussian distribution
Differential Termination Resistance mismatch	The difference in the DC termination resistance with respect to ground of any two signals forming a differential pair. Usually due to large process spread the absolute termination resistance is specified relatively loose, e.g. 20% where the relative difference of resistors of the same device will be much less, e.g 5%. This parameter is used to specify the relative difference tighter than the overall resistance for the purpose of minimizing differential signal mode conversion
Gaussian	A statistical distribution (also termed “normal”) characterized by populations that are not bound in value and have well defined “tails”. The term “random” in this document always refers to a Gaussian distribution.
Golden PLL	Refers to a defined clock extraction unit which phase tracks the inherent clock present in a data signal. The phase tracking bandwidth is usually defined in terms of a corner frequency and if not defined with a corner frequency of baud/1667, a roll off of 20 dB/dec and <0.1 dB peaking
Golden Channel	Refers to an electrical channel which is usually identified using a channel compliancy methodology and is used in the testing of transmitters and receivers

Table 9-2. General Definitions

Parameter	Description
Intersymbol Interference	Data dependent deterministic jitter caused by the time differences required for the signal to arrive at the receiver threshold when starting from different places in bit sequences (symbols). For example when using media that attenuates the peak amplitude of the bit sequence consisting of alternating 0, 1, 0, 1... more than peak amplitude of the bit sequence consisting of 0, 0, 0, 0, 1, 1, 1, 1... the time required to reach the receiver threshold with the 0, 1, 0, 1... is less than required from the 0, 0, 0, 0, 1, 1, 1, 1... The run length of 4 produces a higher amplitude which takes more time to overcome when changing bit values and therefore produces a time difference compared to the run length of 1 bit sequence. When different run lengths are mixed in the same transmission the different bit sequences (symbols) therefore interfere with each other. Intersymbol Interference is expected whenever any bit sequence has frequency components that are propagated at different rates by the transmission media.
Lane	A single RapidIO Channel
Link	A functional connection between the Tx and Rx ports of 2 components, that can be multiple or parallel RapidIO Lanes defined as 1:N. The definition a Link does not imply duplex operation.
Non-transparent applications	Defines an application where the high frequency transmit jitter of a device is defined independently to the high frequency jitter present at any data input of the same device
Skew	The constant portion of the difference in the arrival time between the data of any two in-band signals.
Stressed Signal (or) Stressed Eye	In order to test the tolerance of a receiver a stressed signal or eye is defined which when applied to the receiver must be received with the defined Bit Error Rate. The stressed signal or eye is defined in terms of its horizontal closure or jitter and amplitude normally in conjunction with an eye-mask.
Transparent applications	Defines an application where the high frequency transmit jitter of a device is dependent on the high frequency jitter present at one or more of the data inputs of the same device
Symbol	Unit of information conveyed by a single state transition in the medium
Symbol spaced	Describes a time difference equal to the nominal period of the data signal
Unit Interval	One nominal bit period for a given signaling speed. It is equivalent to the shortest nominal time between signal transitions. UI is the reciprocal of Symbol.

Table 9-3. Jitter and Wander Definitions

Parameter	Description
Correlated Bounded Gaussian Jitter	Jitter distribution where the value of the jitter shows a correlation to the signal level being transmitted. The distribution is quantified, using a Gaussian approximation, as the gradient of the bathtub linearization at the Bit Error Rate of interest. R_RJ = R_GJ
Correlated Bounded High Probability Jitter	Jitter distribution where the value of the jitter shows a strong correlation to the signal level being transmitted. This jitter may considered as being equalizable due to its correlation to the signal level. Was called Data Dependent Jitter in earlier specification revisions.
Correlated Wander	Components of wander that are common across all applicable in band signals.
Duty Cycle Distortion	The absolute value of the difference in the average width of a'1' symbol or a'0' symbol and the ideal periodic time in a clock-like repeating 0,1,0,1 sequence. Duty Cycle Distortion is part of the CBHPJ distribution and is measured at the time-averaged signal level.
Gaussian Jitter	An overall term that defines a jitter distribution that at the BER of interest e.g. 1e-15 still shows a Gaussian distribution. Unless otherwise specified Gaussian Jitter is the RMS sum of CBGJ and UUGJ. Was called Random Jitter in earlier specification revisions.

Table 9-3. Jitter and Wander Definitions

Parameter	Description
High Probability Jitter	Jitter distribution that at the BER of interest is approximated by a dual dirac. Unless otherwise specified High Probability Jitter is the sum of UBHPJ, CBHPJ, PJ, SJ, DCD. The distribution is quantified, using a dual dirac approximation, as the offset of the bathtub linearization at the Bit Error Rate of interest. Was called Deterministic Jitter in earlier specification revisions.
Jitter	Jitter is deviation from the ideal timing of an event at the mean amplitude of the signal population. Low frequency deviations are tracked by the clock recovery circuit, and do not directly affect the timing allocations within a bit interval. Jitter that is not tracked by the clock recovery circuit directly affects the timing allocations in a bit interval. Jitter is phase variations in a signal (clock or data) after filtering the phase with a single pole high pass filter with the -3 dB point at the jitter corner frequency.
Jitter Generation	Jitter generation is the process whereby jitter appears at the output port in the absence of applied input jitter at the input port.
Jitter RMS	The root mean square value or standard deviation of jitter. See clause 2 for more information.
Jitter Transfer	The ratio of the jitter output and jitter input for a component, device, or system often expressed in dB. A negative dB jitter transfer indicates the element removed jitter. A positive dB jitter transfer indicates the element added jitter. A zero dB jitter transfer indicates the element had no effect on jitter. The ratio should be applied separately to deterministic components and Gaussian (random) jitter components.
Peak-to-Peak Jitter	For any type of jitter, Peak to Peak Jitter is the full range of the jitter distribution that contributes within the specified BER.
Periodic Jitter	A sub form of HPJ that defines a jitter which has a single fundamental harmonic plus possible multiple even and odd harmonics.
Relative Wander	Components of wander that are uncorrelated between any two in band signals (See Figure 9-6)
Sigma	Refers to the standard deviation of a random variable modelled as a Gaussian Distribution. When used in reference to jitter, it refers to the standard deviation of the Gaussian Jitter component(s). When used in reference to confidence levels of a result refers to the probability that the result is correct given a Gaussian Mode, e.g. a measured result with 3 sigma confidence level would imply that 99.9% of the measurements are correct.
Sinusoidal Jitter	A sub form of HPJ that defines a jitter which has a single frequency harmonic.
Total Jitter	Sum of all jitter components.
Total Wander	The sum of the correlated and uncorrelated wander. (See Figure 9-7)
Unbounded Gaussian Jitter	Jitter distribution that shows a true Gaussian distribution where the observed peak to peak value has an expected value that grows as a function of the measurement time. This form of jitter is assumed to arise from phase noise random processes typically found in VCO structures or clock sources. It is usually quantified as either the Root Mean Square (RMS) or Sigma of the Gaussian distribution, or as the expected peak value for a given measurement population. (Formally defined as T_RJ)
Uncorrelated Bounded High Probability Jitter.	Jitter distribution where the value of the jitter show no correlation to any signal level being transmitted. Formally defined as T_DJ.
Uncorrelated Wander	Components of wander that are not correlated across all applicable in band signals.
Wander	The peak to peak variation in the phase of a signal (clock or data) after filtering the phase with a single pole low pass filter with the -3db point at the wander corner frequency. Wander does not include skew.

9.4.1 Definition of Amplitude and Swing

LP-Serial links use differential signaling. This section defines the terms used in the description and specification of these differential signals. Figure 9-1 shows how these signals are defined and sets out the relationship between absolute and differential voltage amplitude. The figure shows waveforms for either the transmitter output (TD and $\overline{\text{TD}}$) or a receiver input (RD and $\overline{\text{RD}}$).

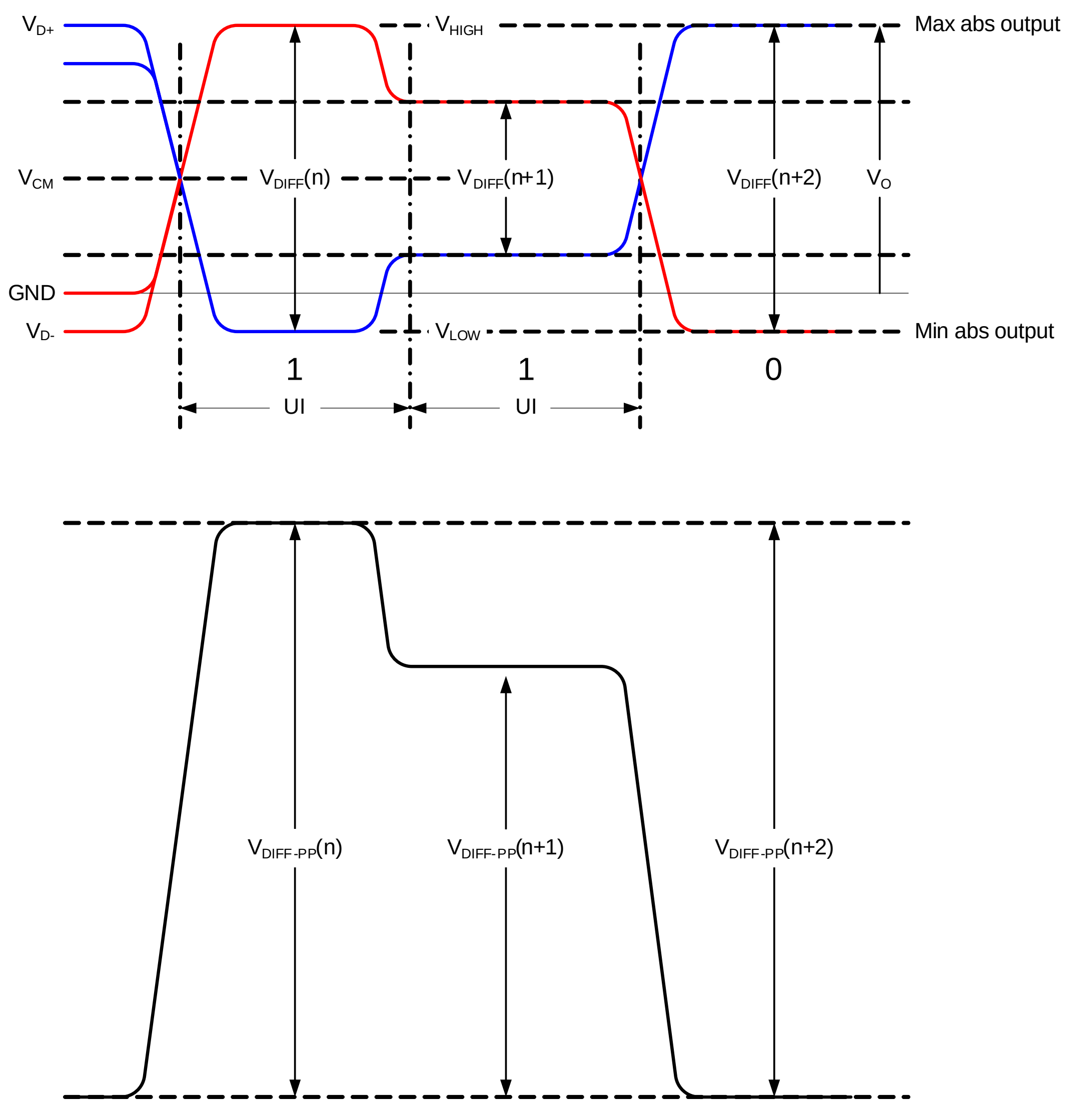


Figure 9-1. Definition of Transmitter Amplitude and Swing

Each signal swings between the voltages V_{HIGH} and V_{LOW} where

$$V_{HIGH} > V_{LOW}$$

The differential voltage, V_{DIFF} , is defined as

$$V_{DIFF} = V_{D+} - V_{D-}$$

where V_{D+} is the voltage on the positive conductor and V_{D-} is the voltage on the negative conductor of a differential transmission line. V_{DIFF} represents either the differential output signal of the transmitter, V_{OD}, or the differential input signal of the receiver, V_{ID} where

$$\mathbf{V_{OD} = V_{TD} - V_{\overline{TD}}}$$

and

$$\mathbf{V_{ID} = V_{RD} - V_{\overline{RD}}}$$

The common mode voltage, V_{CM}, is defined as the average or mean voltage present on the same differential pair. Therefore

$$\mathbf{V_{CM} = |V_{D+} + V_{D-}|/2}$$

The maximum value, or the peak-to-peak differential voltage, is calculated on a per unit interval and is defined as

$$\mathbf{V_{DIFFp\text{-}p} = 2 \text{ x } max|V_{D+} - V_{D-}|}$$

because the differential signal ranges from $V_{D+} - V_{D-}$ to $-(V_{D+} - V_{D-})$

To illustrate these definitions using real values, consider the case of a CML (Current Mode Logic) transmitter and each of its outputs, TD and $\overline{\text{TD}}$, has a swing that goes between V_{HIGH} = 2.5 V and V_{LOW} = 2.0 V, inclusive. Using these values the common mode voltage is calculated to be 2.25 V and the single-ended peak voltage swing of the signals TD and $\overline{\text{TD}}$ is 500 mVpp. The differential output signal ranges between 500 mV and -500 mV, inclusive. therefore the peak-to-peak differential voltage is 1000 mVppd.

9.4.2 Transmitter (Near-End) Template

For each baud rate at which the LP-Serial transmitter is specified to operate, the output eye pattern for transition symbols shall fall entirely within the unshaded portion of the Transmitter (near-end) Output Compliance Mask defined in Figure 9-2. Specific parameter values are called out in the sections that follow.

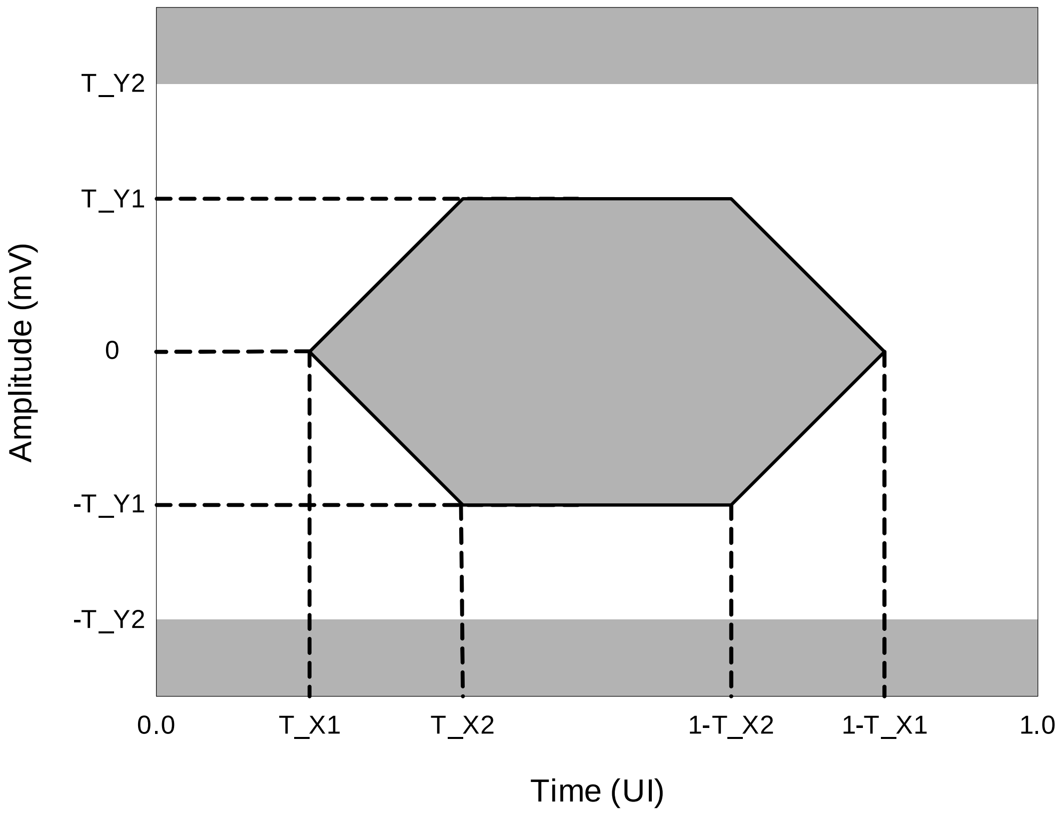


Figure 9-2. Transition Symbol Transmit Eye Mask

The output eye pattern of a LP-Serial transmitter that implements de-emphasis (to equalize the link and reduce intersymbol interference) need only comply with the Transition Transmitter Output Compliance Mask when there is a symbol transition from 1 to 0 or 1 to 0 or when pre-emphasis is disabled or minimized

For 5 Gbaud and 6.25 Gbaud links the Transmitters eye mask will also be evaluated during the steady-state where there are no symbol transitions, e.g a 1 followed by a 1 or a 0 followed by a 0, and the signal has been de-emphasized. This additional transmitter eye mask constraint is shown in Figure 9-3.

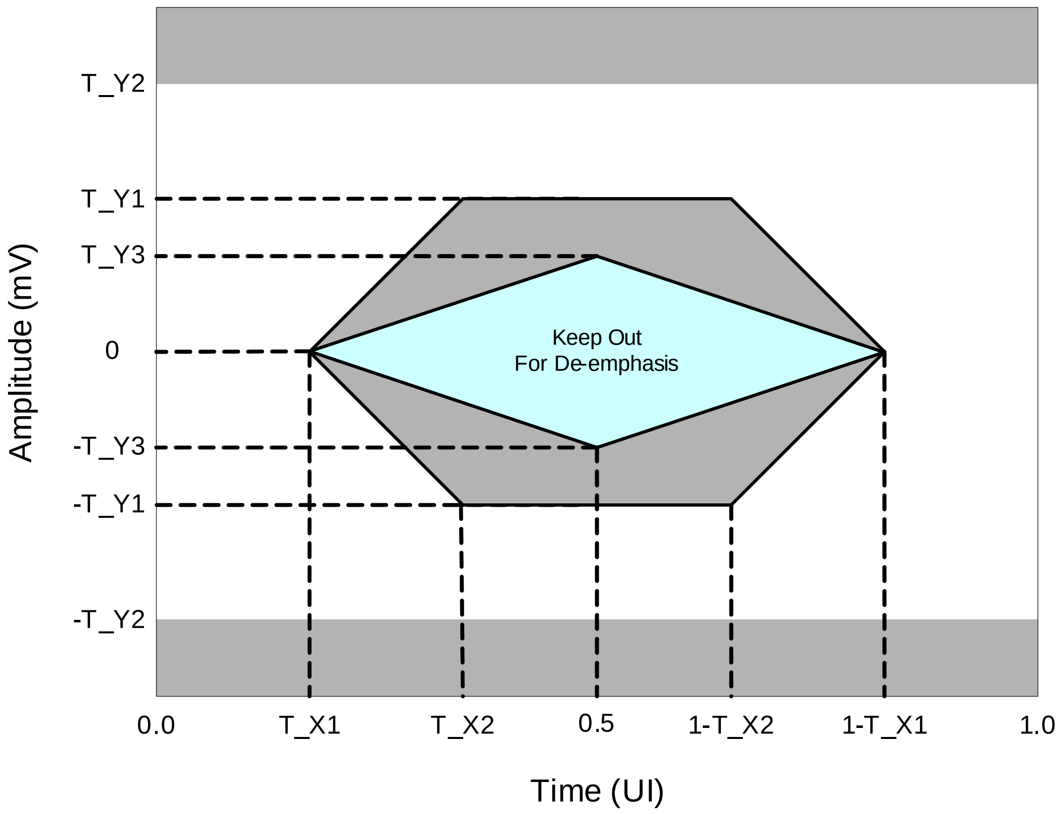


Figure 9-3. Transition and Steady State Symbol Eye Mask

During the steady-state the eye mask prevents the transmitter from de-emphasizing the low frequency content of the data too much and limiting the available signal-to-noise at the receiver.

The de-emphasis introduces a jitter artifact that is not accounted for in this eye mask. This additional jitter is the result of the finite rise/fall time of the transmitter and the non-uniform voltage swing between the transitions. This additional deterministic jitter must be accounted for as part of the high probability jitter.

Table 9-4 defines the standard parameters that will be specified for every transmitter.

Table 9-4. Transmitter Output Jitter Specification

Characteristic	Symbol	Conditions	Min	Typ	Max	Units
Total Jitter	T_TJ					UIpp
Eye Mask	T_X1					UI
Eye Mask	T_X2					UI
Eye Mask	T_Y1					mV
NOTES: Uncorrelated Unbounded Gaussian Jitter must be defined with respect to specified BER of 10^{-12}, Q=7.03 for 1.25 Gbaud, 2.5 Gbaud, and 3.125 Gbaud links Uncorrelated Unbounded Gaussian Jitter must be defined with respect to specified BER of 10^{-15}, Q=7.94 for 5 Gbaud and 6.25 Gbaud links						

Table 9-4. Transmitter Output Jitter Specification

Characteristic	Symbol	Conditions	Min	Typ	Max	Units
Eye Mask	T_Y2					mV
Eye Mask (5 Gbaud and 6.25 Gbaud only)	T_Y3					mV
NOTES: Uncorrelated Unbounded Gaussian Jitter must be defined with respect to specified BER of 10^{-12}, Q=7.03 for 1.25 Gbaud, 2.5 Gbaud, and 3.125 Gbaud links Uncorrelated Unbounded Gaussian Jitter must be defined with respect to specified BER of 10^{-15}, Q=7.94 for 5 Gbaud and 6.25 Gbaud links						

Note: In previous versions of the RapidIO LP-Serial specification different symbols names were used to define the time and voltage points on eye masks. Table 9-5 can be used as a cross reference for the transmitter eye mask symbol names.

Table 9-5. Transmitter Eye Mask Cross Reference

Current Version	1.3 Version
T_Y1	V_{DIFF} min
T_Y2	V_{DIFF} max
T_Y3	N/A
T_X1	A
T_X2	B

9.4.3 Receiver (Far-End) Template

The receiver (far-end) template has two definitions based on Level I and Level II links.

9.4.3.1 Level I Receiver Template

Figure 9-4 illustrates the definition in a Level I receiver eye template.

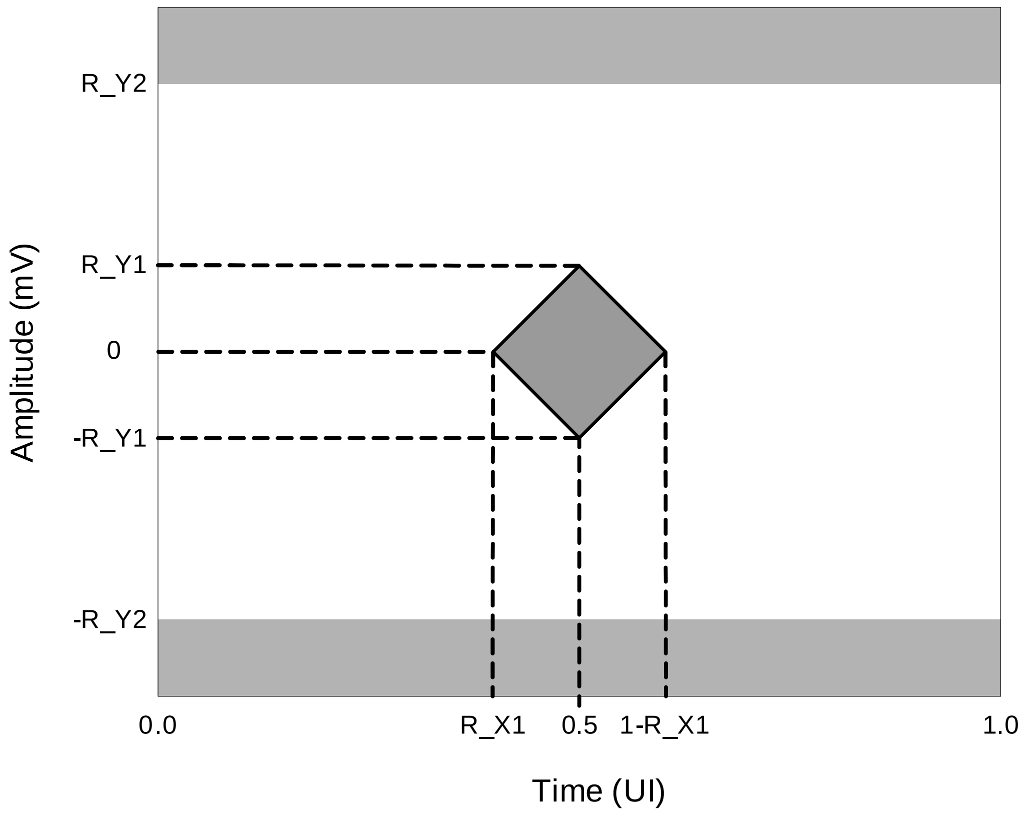


Figure 9-4. Level I Receiver Input Mask

Table 9-8 defines the standard parameters that will be specified for Level I receivers which have an open eye at the far-end. The termination conditions used to measure the received eye are defined Section 9.5.13.

Table 9-6. Level I Receiver Jitter Specification

Characteristic	Symbol	Conditions	Min	Typ	Max	Units
Total Jitter	R_TJ					UIpp
Eye Mask	R_X1					UI
Eye Mask	R_X2					UI
Eye Mask	R_Y1					mV
NOTES: Uncorrelated Unbounded Gaussian Jitter must be defined with respect to specified BER of 10^{-12}, Q=7.03 for 1.25 Gbaud, 2.5 Gbaud, and 3.125 Gbaud links						

Also in the previous versions of the RapidIO LP-Serial specification different symbols names were used to define the time and voltage points on eye masks. Table 9-8 can be used as a cross reference for the receiver eye mask.

Table 9-7. Receiver Eye Mask Cross Reference

Current Version	1.3 Version
R_Y1	V_{DIFF} min
R_Y2	V_{DIFF} max
R_X1	A
R_X2	B

9.4.3.2 Level II Receiver Template

For a Level II link the receiver mask it is defined as is defined in Figure 9-5. Specific parameter values for both masks are called out in the sections that follow.

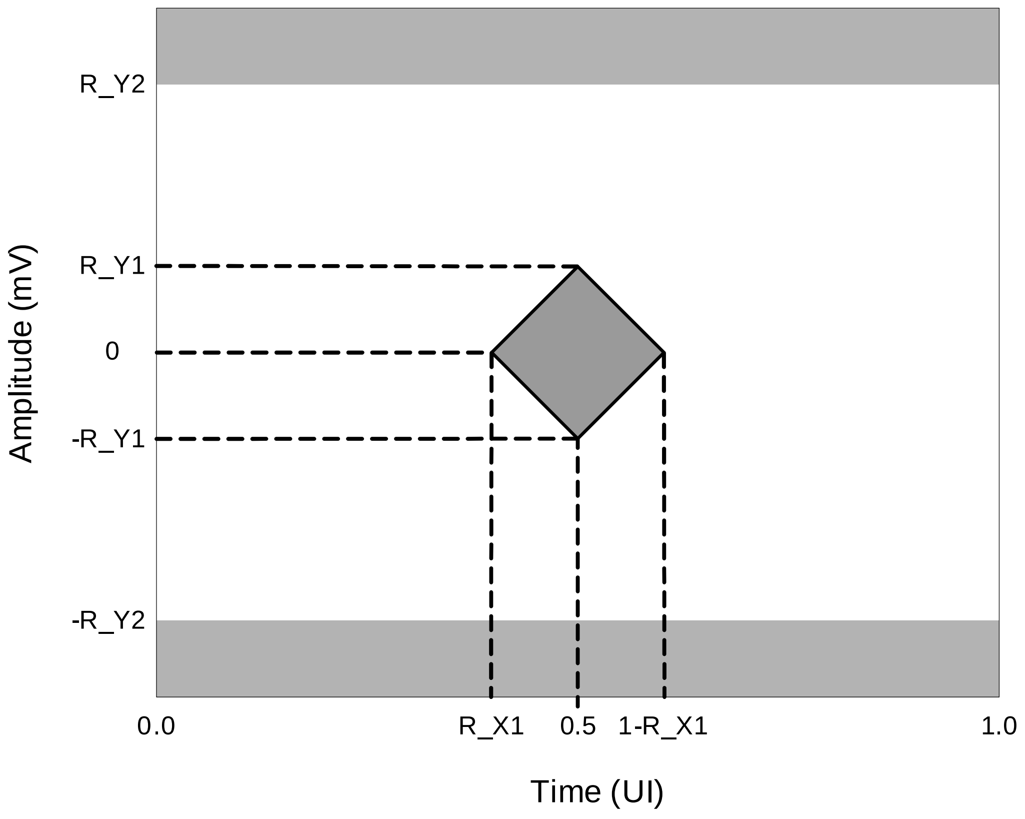


Figure 9-5. Receiver Input Mask

Table 9-8 defines the standard parameters that will be specified for receivers that have an open eye at the far-end. The termination conditions used to measure the received eye are defined Section 9.5.13.

Table 9-8. Level II Receiver Jitter Specification

Characteristic	Symbol	Conditions	Min	Typ	Max	Units
Total Jitter	R_TJ					UIpp
Eye Mask	R_X1					UI
Eye Mask	R_Y1					mV
NOTES: Uncorrelated Unbounded Gaussian Jitter must be defined with respect to specified BER of 10^{-15}, Q=7.94 for 5 Gbaud and 6.25 Gbaud links						

9.4.4 Definition of Skew and Relative Wander

See Figure 9-6 for an illustration of skew and relative wander. The definitions appear in Table 9-3.

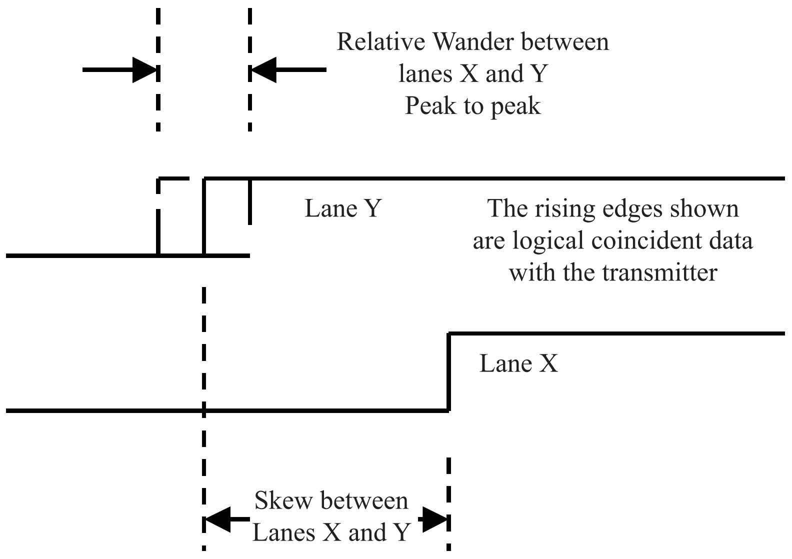


Figure 9-6. Skew and Relative Wander Between in Band Signals

See Figure 9-7 for an illustration of total wander in a signal. The definition appears in Table 9-3.

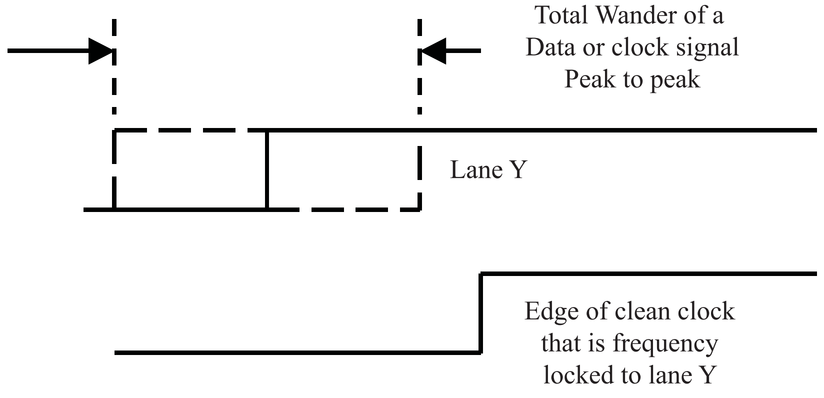


Figure 9-7. Total Wander of a Signal

9.4.5 Total Wander Mask

Total wander specifications should be considered as accumulated low frequency jitter. As modern CDRs are digitally based they show a corner tracking frequency plus slew limitation which has been guaranteed, therefore for jitter tolerance testing the total wander needs to be spectrally defined to ensure correct operation.

To this end, for jitter tolerance testing, the wander is considered a sinusoidal jitter source as shown in Figure 9-8 below.

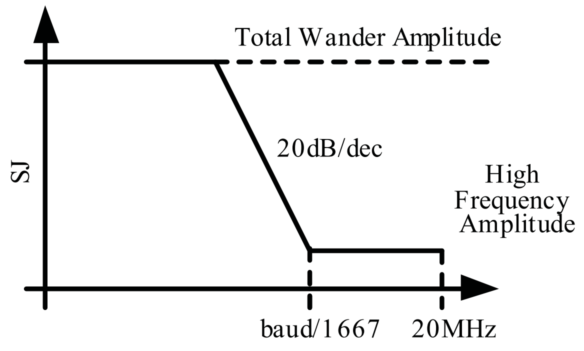


Figure 9-8. Total Wander Mask

At higher frequency this jitter source is used to ensure margin in the high frequency jitter tolerance of the receiver. At lower frequencies the higher SJ should then be tracked by the CDR.

9.4.6 Relative Wander Mask

Specifically for interfaces defining relative wander, Figure 9-9 is also defined in terms of a sinusoidal jitter sources as shown below.

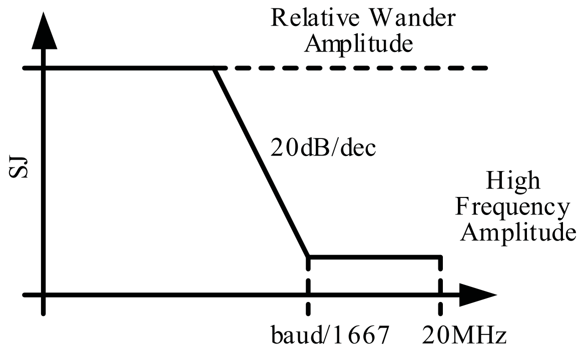


Figure 9-9. Relative Wander Mask

9.4.7 Random Jitter Mask

To ensure that the random jitter modulation of stressed signals is above the CDR bandwidth and therefore untracked, the filter mask shown in Figure 9-10 shall be applied where necessary.

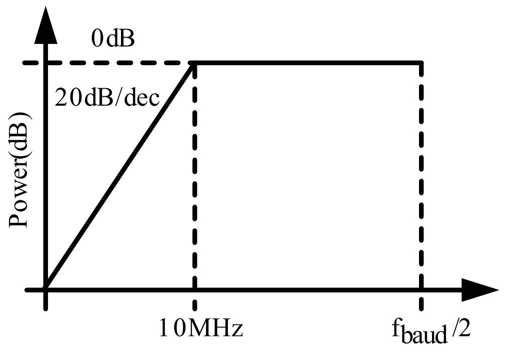


Figure 9-10. Random Jitter Spectrum

9.4.8 Defined Test Patterns

The data test patterns are unique to the two levels of link and will be defined in the sections specific to these.

9.4.9 Reference Model

The LP-Serial electrical reference model is defined in Figure 9-11. Note that the RX and TX blocks include all off-chip components associated with the respective function. Thus the reference points T and R are defined to be the component edge of the transmitter and receiver respectively.

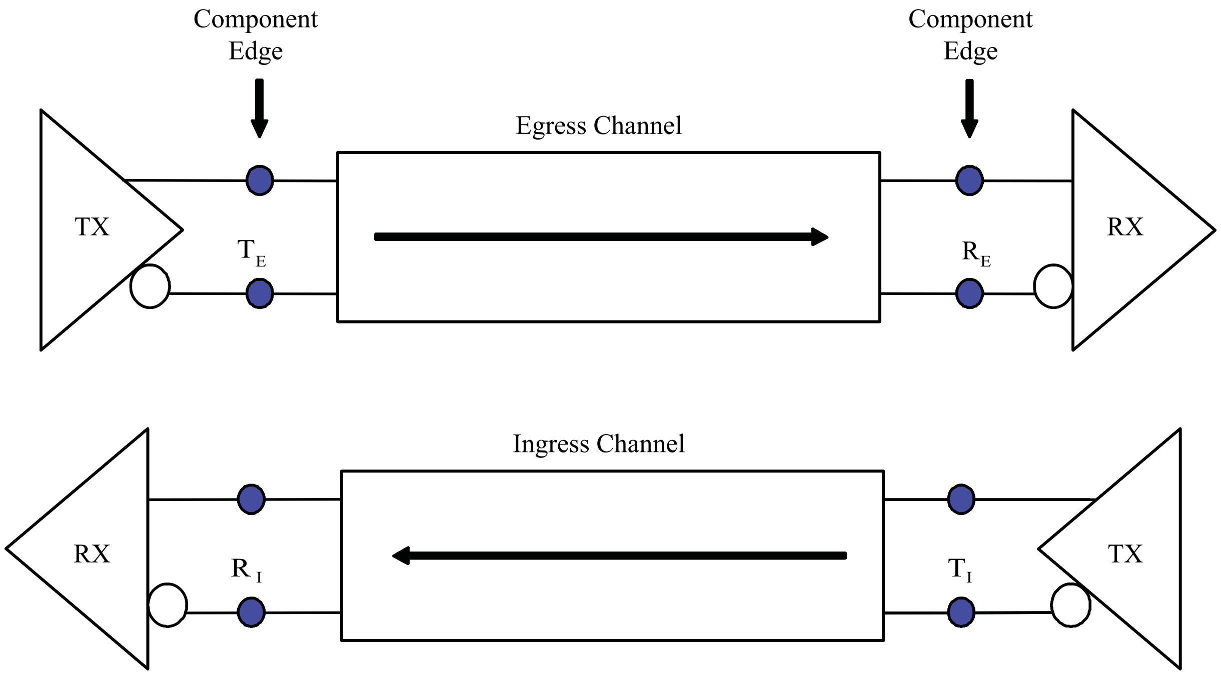


Figure 9-11. Reference Model

Note: Through out this specification the terms ‘near’ and ‘far’ are used to describe aspects of the channel. **Near-end** will always be used to refer to the end of the channel attached to the transmitter, e.g. T_E or T_I, independent of if it is the egress or ingress channel. **Far-end** will be used to refer to the end of the channel attached to the receiver, e.g. R_I or R_E.

9.5 Common Electrical Specification

9.5.1 Introduction

This section specifies electrical parameters and attributes common to all links. In the event of a difference between an individual link and these general requirements, the respective individual link shall prevail.

The LP-Serial 1.25 Gbaud, 2.5 Gbaud, and 3.125 Gbaud Electrical specifications are guided by the XAUI electrical interface specified in Clause 47 of IEEE 802.3ae-2002.[1]

The LP-Serial 5 Gbaud and 6.25 Gbaud Electrical specifications are based upon the Optical Internetworking Forum's Common Electrical Interface [2], referred to henceforth as CEI.

CEI includes the following sections that are the basis for the LP-Serial RapidIO 5 Gbaud and 6.25 Gbaud interfaces:

- CEI-6G-SR clause 6 specification for data lane(s) that support bit rates from 4.976 to 6.375 Gbaud over Printed Circuit Boards with physical runs from 0 to 20 cm and up to 1 connector. CEI-6G-SR forms the basis for the LP-Serial 5 Gbaud and 6.25 Gbaud Short Run Interface electrical specifications. RapidIO has enhanced this electrical specification to include a continuous-time equalizer with one zero and one pole.
- CEI-6G-LR Clause 7 specification for data lane(s) that support bit rates from 4.976 to 6.375 Gbaud over Printed Circuit Boards with physical runs from 0 to 100 cm and up to 2 connectors. CEI-6G-LR forms the basis for the LP-Serial 5 Gbaud and 6.25 Gbaud Long Run Interface electrical specifications.
- RapidIO has added a specification for data lane(s) that supports bit rates from 5 to 6.25 Gbaud over Printed Circuit Boards and physical runs from 0 to 60 cm and up 2 connectors. The CEI-6G-LR transmitter and a continuous-time receiver with one zero and one pole form the basis for the LP-Serial 5 Gbaud and 6.25 Gbaud Medium Run Interface electrical specifications.

  Note: The OIF CEI documentation uses the term “reach” to describe the length of the channel. Here “run” is used to maintain consistency with the RapidIO 1.3 interconnect specification.

  While the OIF CEI documentation defines support for 4.976 to 6.375 Gbaud RapidIO only supports 5.0 Gbaud and 6.25 Gbaud data rates

9.5.2 Data Patterns

There is a requirement that the link data follow 8b/10b encoding rules and when specified raw data scrambling requirements as defined in Chapter 4, "8b/10b PCS and PMA Layers", to ensure proper operation. The predicted BER performance and jitter requirements are only valid when this assumption is satisfied. If all of these conditions are not met, then the link may not work to the full distance, or meet the BER, or in fact work at all.

9.5.3 Signal Levels

The signal is a low swing differential interface. This implies that the receiver has a wide common mode range (within the maximum absolute input voltages). All devices must support load type 0 defined in Table 9-9. Level II SR devices can optionally support any or all of the other 3 load types while Level II MR and LR devices can optionally support load type 1.

Table 9-9. Definition of Load Types

Characteristics	Load Type 0	Load Type 1	Load Type 2	Load Type 3	Units
R_Zvtt	>1k	<30	<30	<30	Ω
Nominal Vtt	undefined	1.2	1.0	0.8	V

This type of differential interface allows for inter-operability between components operating from different supply voltages and different I/O types (CML, LVDS-like, PECL, etc.). Low swing differential signaling provides noise immunity and improved electromagnetic interference (EMI). Differential signal swings are defined in following sections and depend on several factors such as transmitter pre-equalization, receiver equalization, and transmission line losses.

9.5.4 Bit Error Ratio

9.5.4.1 Level I Bit Error Ratio

The LP-Serial 1.25 Gbaud, 2.5 Gbaud, and 3.125 Gbaud interface lanes will operate with a Bit Error Ratio (BER) of 10^{-12}.

It should be noted that most modern system are capable of achieving the improved BER required in Level II links.

9.5.4.2 Level II Bit Error Ratio

The LP-Serial 5 Gbaud and 6.25 Gbaud interface lanes will operate with a Bit Error Ratio (BER) of 10^{-15} (with a test requirement to verify 10^{-12}). See Clause 2 of CEI for more information on the jitter model and how to measure BER.

9.5.5 Ground Differences

The maximum ground difference between the transmitter and the receiver shall be ±50 mV for SR links and ±100mV for MR and LR links. This will affect the absolute maximum voltages at compliance point 'R'. If transmitter and receiver are on the same PCB with no intervening connectors, then the ground difference is approximately 0 mV.

9.5.6 Cross Talk

Cross talk arises from coupling within the connectors, on the PCB, the package and the die. Cross talk can be categorized as either Near-End or Far-End cross talk (NEXT and FEXT). In either of these categories, the amount of cross talk is dependent upon signal amplitudes, signal spectrum, and trace/cable length. There can be many aggressor channels onto one victim channel, however typically only a few are dominant.

Further consideration of cross talk can be found in Annex A, “Transmission Line Theory and Channel Information (Informative)".

9.5.7 Transmitter Test Load

All transmitter characteristics should be implemented and measured to a differential impedance of 100 Ω ±1% at DC with a return loss of better than 20 dB from baud rate divided by 1667 to 1.5 times the baud rate, unless otherwise noted.

9.5.8 Transmitter Lane-to-Lane Skew

While the protocol layer will control some of the lane to lane skew, the electrical level for the lane-to-lane skew caused by the transmitter circuitry and associated routing is allowed up to be:

- less than 1000 ps for port widths less than or equal to 4 lanes
- less than 2 UI + 1000 ps for port width with greater than 4 lanes

Hence, the total output (i.e. measured) lane-to-lane skew is to be specified in the protocol standards with the above skew taken into account. The transmitter lane-to-lane skew is only for the SerDes TX and does not include any effects of the channel.

9.5.9 Receiver Input Lane-to-Lane Skew

The maximum amount of lane-to-lane skew at the input pins of the receiver is determined by the ability of the receiver to resolve the difference between two successive ||A|| columns. Since the minimum number of non-||A|| columns between ||A|| columns is 16, the maximum lane skew that can be unambiguously corrected is the time it takes to transmit 7 code groups per lane. Therefore, the maximum lane-to-lane skew at the input pins of a receiver is calculated as:

(7 code groups) x (10 bits/code-group) x (1 UI/bit) x (ns/UI)

It is important to note that the total lane-to-lane skew specification includes the skew caused by the transmitter’s PCS and PMA (SerDes), the channel, the receivers’ PMA (SerDes) and PCS and any logic that is needed to create the aligned column of ||A|| at the receiving device.

9.5.10 Transmitter Short Circuit Current

The max DC current into or out of the transmitter pins when either shorted to each other or to ground shall be ±100 mA when the device is fully powered up. From a hot swap point of view, the ±100 mA limit is only valid after 10 μs.

9.5.11 Differential Resistance and Return Loss, Transmitter and Receiver

The DC differential resistance shall be between 80 and 120 Ω, inclusive.

The differential return loss shall be better than A0 from f0 to f1 and better than

A0 + Slope*log10(f/f1)

where f is the frequency from f1 to f2 (see Figure 9-12). Differential return loss is measured at compliance points T and R. If AC coupling is used, then all components (internal or external) are to be included in this requirement. The reference impedance for the differential return loss measurements is 100 Ω.

Common mode return loss measurement shall be better than -6 dB between a minimum frequency of 100 MHz and a maximum frequency of 0.75 times the baud rate. The reference impedance for the common mode return loss is 25 Ω.

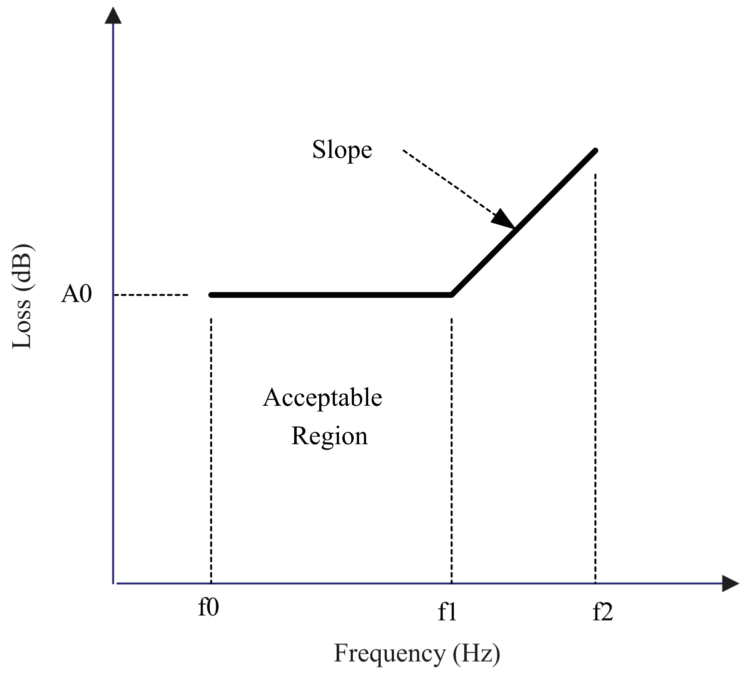


Figure 9-12. Transmitter and Input Differential Return Loss

9.5.12 Baud Rate Tolerance

The baud rates are defined to be 1.25 Gbaud, 2.5 Gbaud, 3.125 Gbaud, 5 Gbaud and 6.25 Gbaud. Each interface is required to operate asynchronously with a tolerance of ±100 ppm from the nominal baud rate.

Note: The minimum and maximum baud rates can be calculated as:

Baudrate∗(1 ± 100E-6)

9.5.13 Termination and DC Blocking

Each link requires a nominal 100 Ω differential source termination at the transmitter and a nominal 100 Ω differential load termination at the receiver. The terminations shall provide both differential and common mode termination to effectively absorb differential or common mode noise and reflections. Receivers and transmitters shall support AC coupling and may also optionally support DC coupling. AC Coupled receivers require a differential termination >1 kΩ at DC (by blocking capacitors in or near receivers as shown in Figure 9-13 or by circuit means within the receiver). DC Coupled devices shall meet additional electrical parameters T_Vcm, R_Vrcm, R_Vtt, R_Zvtt. All termination components are included within the RX and TX blocks as shown in the reference model as defined in Section 9.4.9.

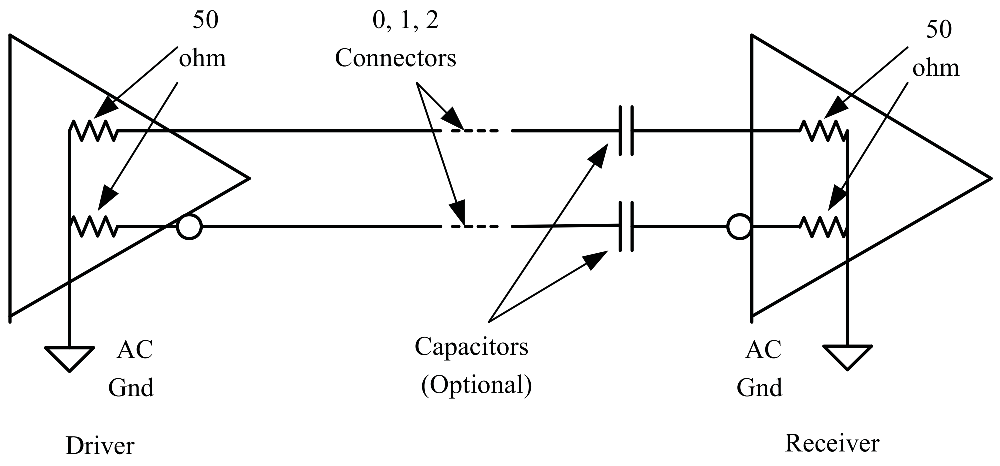


Figure 9-13. Termination Example

9.6 Pulse Response Channel Modelling

This section shall describe the theoretical background for channel modelling.

9.6.1 Generating a Pulse Response

Knowing the spectral transfer function for a channel allows the pulse response of the channel can be calculated using tools such as MATLAB®

The Pulse Response of the channel is the received pulse for an ideal square wave and is calculated by either

- convolving the pulse with the impulse response of the channel or
- multiplying the Fourier spectrum of the ideal transmitted square wave with the channel response and taking the inverse Fourier transform, where

f_{max} is difference between the maximum positive and minimum negative frequency

P is the number of equally space points in the frequency array

$tx(t)$ is the transmit signal pulse

$tx(\omega)$ is the transmit signal pulse in the frequency domain

$Tr(\omega)$ is the transfer function of the channel

$rx(t)$ is the resulting pulse response of the channel

$$t_{step} = \frac{1}{f_{max}}$$

$$t = t_{step} \cdot n$$

$$n = [1,P]$$

$$tx(t) = H(0) \cdot H(t_{period} - t)$$

$$rx(\omega) = tx(\omega) \cdot Tr(\omega)$$

$$rx(t) = ifft(rx(\omega))$$

9.6.2 Basic Pulse Response Definitions

A receive pulse response as calculated is graphically represented in Figure 9-14.

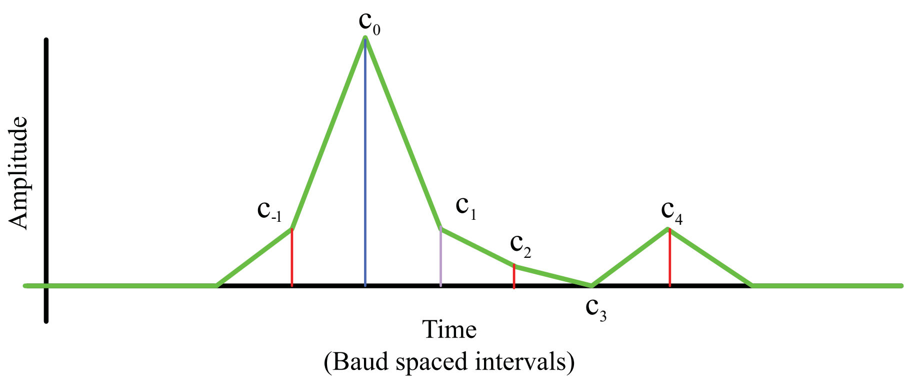


Figure 9-14. Graphical Representation of Receiver Pulse

Cursors are defined as being the amplitude of the received pulse at symbol spaces from the maximum signal energy at c_0, and extend to infinity in both negative and positive time. The exact position of c_0 is arbitrary and is defined specifically by the various methodologies.

A precursor is defined as a cursor that occurs before the occurrence of the main signal c_0, i.e. c_n where n<0, usually convergences to zero within a small number of bits

A post cursor is defined as a cursor that occurs after the occurrence of the main signal c_0, i.e. c_n where n>0, and usually convergences to zero within twice the propagation time of the channel.

Given a deterministic data stream travelling across the channel, the superposition of the channel pulses give rise to Inter-Symbol Interference (ISI). This ISI has a maximum occurring for a worst case pattern, which for a channel response where all cursors are positive would be a single *1* or *0* in the middle of a long run of *0*s or *1*s respectively. This maximum is referred to Total Distortion.

$$\Theta = \sum_{(n = -\infty), (n \neq 0)}^{n = \infty} |c_n|$$

Due to ISI an enclosure in the time domain also occurs which can be determined by either running exhaustive simulations or simulations with determined worst case patterns. For the case where the ISI is so large that the eye is closed, Inherent Channel Jitter has no meaning.

9.6.3 Transmitter Pulse Definition

A transmitter is defined by its ability to generate a transmit pulse. A single *1* transmit symbol has different amplitudes at symbol space intervals, t_n, where post taps have n>0, and pre-taps have n<0.

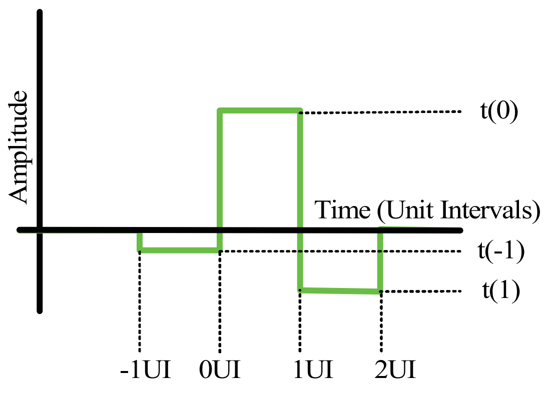


Figure 9-15. Transmit Pulse

When a pulse train is transmitted the exact transmitted amplitude is therefore the superposition of the pulses from the previous and to be transmitted pulses, such as in a FIR filter.

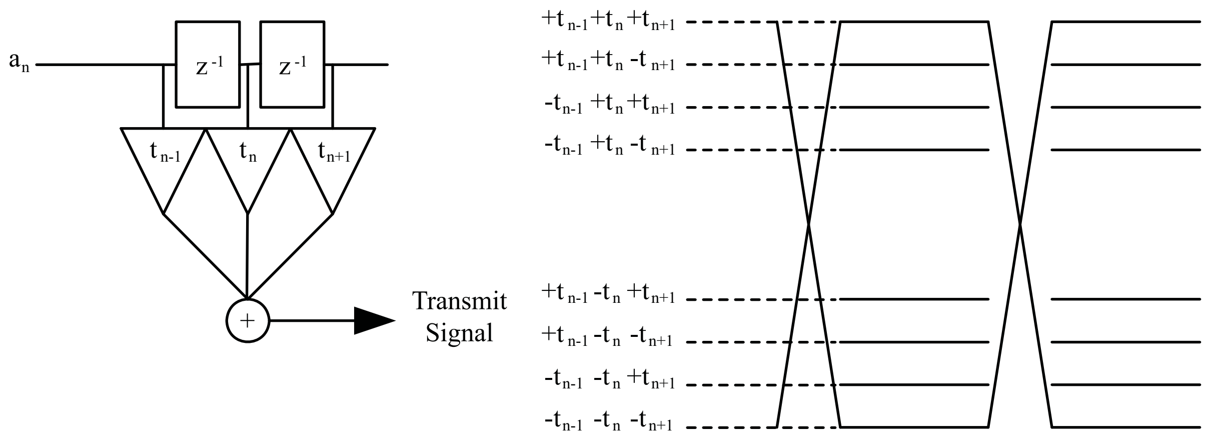


Figure 9-16. Transmitter FIR Filter Function

This superposition can be understood by referring to the amplitudes depicted for various bit sequences in Figure 9-16.

The transmit emphasis can be defined to have certain limits of maximum transmit amplitude or ratios of emphasis as defined below.

$$P_{post} = \frac{t_1}{t_0}$$

$$E = 20\log\frac{1 + P_{post}}{1 - P_{post}}$$

$$\sum |t_n| < V_{tx}\big|_{min}$$

where

P_{post} is the first coefficient of the transmit FIR

E is the emphasis of the transmit emphasis

$V_{tx}\big|_{min}$ is the maximum transmit amplitude

9.6.4 Receiver Pulse Response

Given an emphasized transmitter the pulse response of the receiver should be recalculated using the emphasized transmit pulse as opposed to a simple NRZ pulse.

The receiver pulse cursors are defined in Figure 9-17.

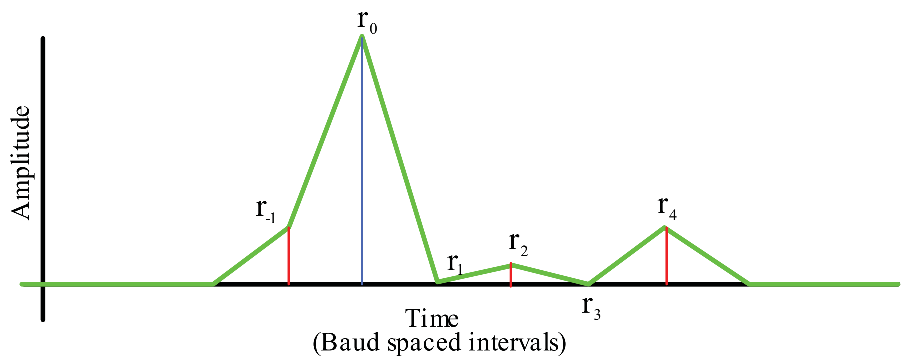


Figure 9-17. Receiver Pulse Definition

9.6.5 Crosstalk Pulse Response

The crosstalk pulse response is analogous to the receiver pulse response as defined in Section 9.6.4 but using the crosstalk channel, i.e. NEXT or FEXT network analysis measurement. The transmit signal as seen in the system should be used for the calculation of the resulting crosstalk pulse response, e.g. an emphasized transmitter from above, or XAUI transmit NRZ pulse.

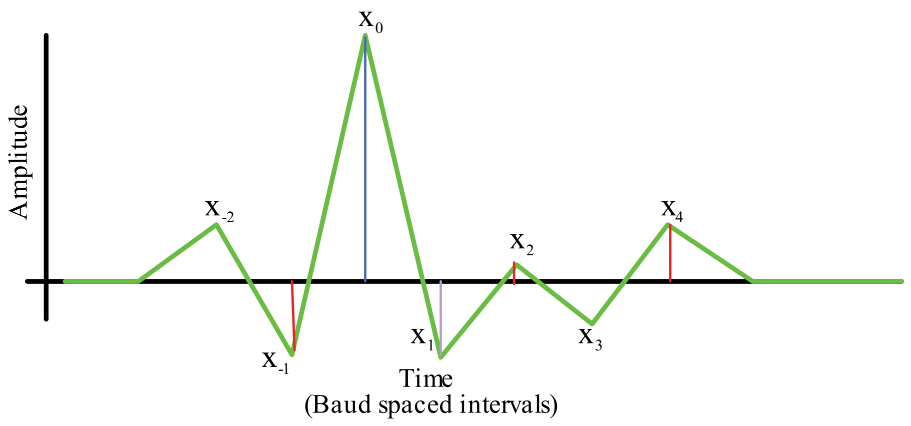


Figure 9-18. Crosstalk Pulse Definition

The Crosstalk pulse response is then defined as above in Figure 9-18 as being a set of cursors x_n usually oscillatory in form. The position of x_0 is defined as being at the maximum amplitude of the pulse response.

9.6.6 Decision Feedback Equalizer

The following filter function can be used to verify the capability of the channel to be used in such an application.

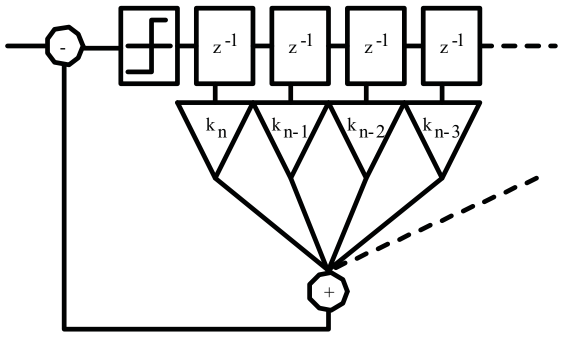


Figure 9-19. Decision Feedback Equalizer

The value of the coefficients are calculated directly from the channel pulse response or the receiver pulse using an emphasized transmitter.

$$k_n = c_n\big|_{n = [1,m]}$$ for unemphasized transmitters, or

$$k_n = r_n\big|_{n = [1,m]}$$ for emphasized transmitters

This equalizer is capable of equalizing a finite number of post cursors, whose individual values may be limited.

9.6.7 Time Continuous Transverse Filter

A.k.a. Feed forward Filter, Finite Input Response or Comb Structure, the Transverse Filter, Figure 9-20 consists of a finite number of coefficients, k. The sum of the continuous value of symbol spaced delayed samples multiplied by these coefficients then gives the resulting signal.

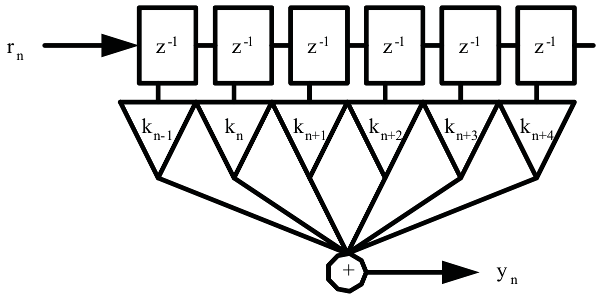


Figure 9-20. Feed Forward Filter

9.6.7.1 Time Continuous Zero-Pole Equalizer Adaption

The pole-zero algorithm takes the SDD21 magnitude response for the through channel and inverts it to produce a desired CTE filter response curve. From a set of initial conditions for p_n poles and z_n zeros, the squared differences are minimized between the CTE response and the inverse channel response curve. The minimization is done using a simplex method, specifically the Nelder-Mead

Multidimensional Unconstrained Non-Linear Minimization Method. The Nelder-Mead method provides a local minimization of the square of the difference between the two curves by descending along the gradient of the difference function. Once the optimization result is obtained, it is compared to a specified threshold. If the threshold exceeds the target tolerance, an incrementally offset seed point is generated from a 6-dimensional grid of seed points, and the process is iterated until the correct curve is obtained within the target tolerance.

9.6.8 Time Continuous Zero/Pole

The Zero/Pole Filter is defined, in the frequency domain by

$$H(f) = \frac{p}{z} \cdot \frac{(z + j2\pi f)}{(p + j2\pi f)}$$

and consists of a single zero, z, and single pole, p.

9.6.9 Degrees of Freedom

9.6.9.1 Receiver Sample Point

A receiver shall be allowed to either position the centre sampling point fully independently to the signal transitions or exactly in between the mean crossover of the receiver signal.

9.6.9.2 Transmit Emphasis

Transmit emphasis and receiver filter coefficients must be optimized with the defined resolution to give the best achievable results. Unless otherwise stated it shall be assumed that the coefficients are defined using floating point variables.

9.7 Jitter Modelling

This section describes the theoretical background of the methodology used for jitter budgeting and jitter measurement. To avoid fundamental issues with the additional of jitter using the dual dirac model through a band limited channel, a fundamental methodology call “stateye” is defined in Section 9.7.5, which uses only convolution of the jitter distribution for the calculation of the jitter at the receiver.

9.7.1 High Frequency Jitter vs. Wander

Jitter is defined as the deviation of the signal transition from an origin, usually its mean. This deviation has an amplitude and an associated spectrum. High frequency jitter is defined by a 1st order high pass phase filter with a corner frequency equal to the ideal CDR bandwidth. The low frequency Jitter or Wander is defined by a 1st order low pass phase filter with a corner frequency equal to the bandwidth.

9.7.2 Total Wander vs. Relative Wander

Generation of Total and Relative Wander can be achieved using a “Common” and “AntiPhase” Sinusoidal Source, where the total and relative wander are then related as defined below.

$$A_{total} = A_{common} + A_{antiphase}$$
$$A_{relative} = 2A_{antiphase}$$

By adding sinusoidal frequencies of slightly differing frequencies the maximum total and relative wander is achieved at various phase relationship like shown in Figure 9-21.

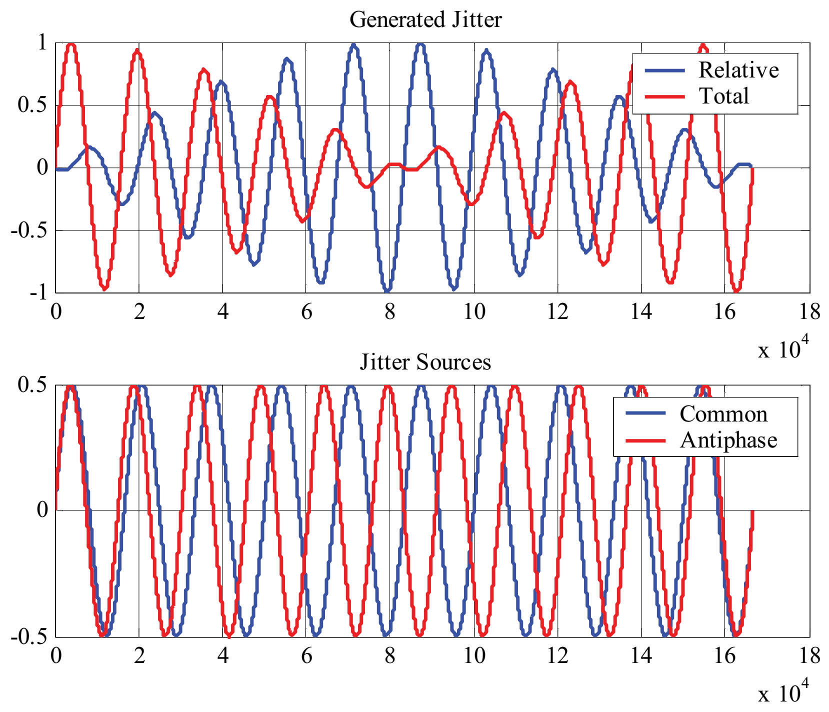


Figure 9-21. Generation of Total and Relative Wander

9.7.3 Correlated vs. Uncorrelated Jitter

If a correlation exists between the amplitude of the jitter and the current, past, and future signal level of a data channel, this type of jitter is deemed correlated. Typically this is encountered when band limitation and inter-symbol interference occurs. Due to amplitude to phase conversion of the ISI, a jitter is observed which has a direct correlation to the data pattern being transmitter.

9.7.4 Jitter Distributions

High frequency jitter is traditionally measured and described using probability density functions which describe the probability of the data signal crossing a decision threshold, as shown in Figure 9-22.

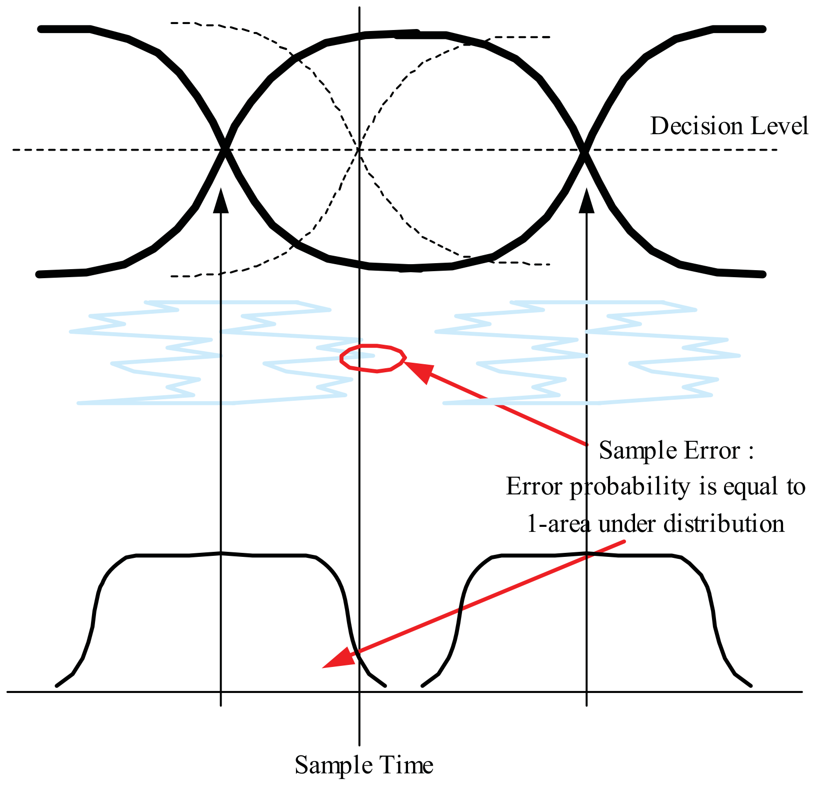


Figure 9-22. Jitter Probability Density Functions

The low probability part of the jitter distribution can be described by two components, mathematically described in the following sections.

9.7.4.1 Unbounded and Bounded Gaussian Distribution

We define a Unbounded Gaussian distribution function in terms of sigma as below.

$$GJ(\tau, \sigma) = \frac{1}{\sqrt{2\pi}} \cdot \frac{1}{\sigma} \cdot e^{-\frac{\tau^2}{2\sigma^2}}$$

For every offset τ, there exists a finite and non-zero probability.

9.7.4.2 Bounded Gaussian Distribution

We define a Bounded Gaussian Distribution function[1] in terms of sigma and a maximum value as below.

$$GJ(\tau,\sigma) = \begin{cases} \dfrac{1}{\sqrt{2\pi}} \cdot \dfrac{1}{\sigma} \cdot e^{\frac{\tau^2}{2\sigma^2}} & \text{if } \tau \le \tau_{max} \\ 0 & \tau > \tau_{max} \end{cases}$$

For random processes consisting of a finite number of random variables there exists a finite non-zero probability only if $\tau \le \tau_{max}$. For example, a band limited channel is bounded but shows a Gaussian Distribution below its maximum. See Section 9.7.4.8, "Example of Bounded Gaussian" for an explanation concerning extrapolation.

9.7.4.3 High Probability Jitter

We define a dual dirac distribution function for a High Probability jitter (W) as below.

$$HPJ(\tau, W) = \frac{\delta(\tau - \frac{W}{2})}{2} + \frac{\delta(\tau + \frac{W}{2})}{2}$$

9.7.4.4 Total Jitter

We define the convolution of the High Probability and Gaussian jitter as being the total jitter and define it as below.

$$TJ(\tau, W, \sigma) = \frac{1}{2\sqrt{2\pi}} \cdot \frac{1}{\sigma} \cdot \left[e^{-\frac{\delta\left(\tau - \frac{W}{2}\right)^2}{2\sigma^2}} + e^{-\frac{\delta\left(\tau + \frac{W}{2}\right)^2}{2\sigma^2}} \right]$$

9.7.4.5 Probability Distribution Function vs. Cumulative Distribution Function

An example of the convolution of GJ (magenta), HPJ (green) to give TJ (red) can be seen Figure 9-23.

[1]Due to the bounded function the function does not comply to the requirements that the integral of the pdf from minus infinity to infinity is one. This small inaccuracy is recognized and acceptance in this context.

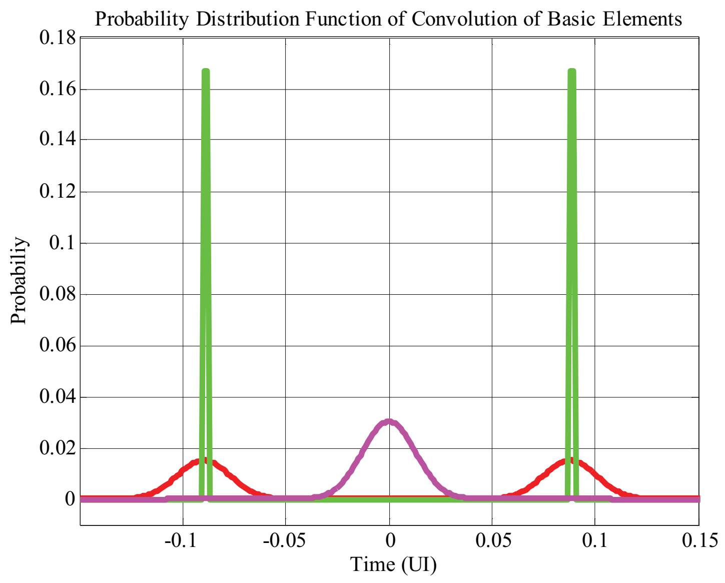


Figure 9-23. Example of Total Jitter PDF

When integrating the probability distribution functions, same colors, we obtain the cumulative distribution function or half the bathtub, shown in Figure 9-24.

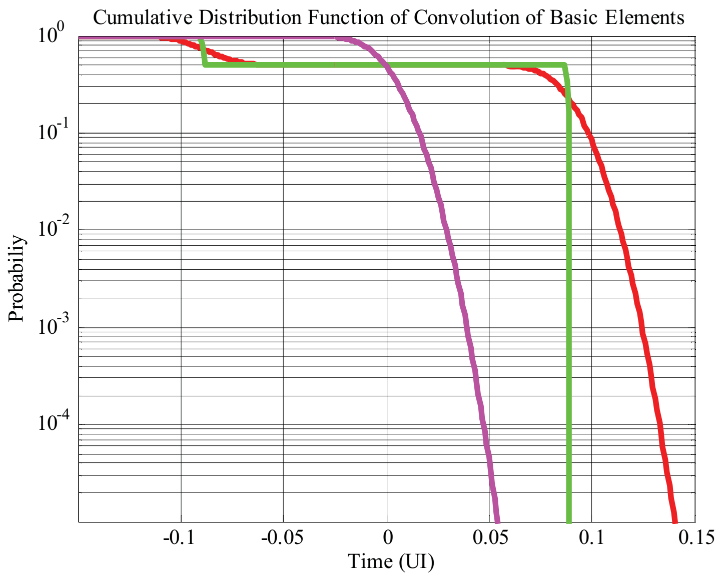


Figure 9-24. Example of Total Jitter CDF

9.7.4.6 BathTub Curves

Given a measured bathtub curve consisting of measured BER for various sampling offsets, the defined Gaussian and High Probability Distributions can be used to describe the important features of the distribution.

Initially the BER axis should be converted to Q as defined below, e.g. a BER of 10^{-12} is a Q=7.04, and a BER of 10^{-15} a Q=7.94[1].

$$Q = \sqrt{2} \cdot erf^{-1}(2 \cdot (1 - BER) - 1)$$

where $\mathrm{erf}^{-1}(x)$ is the inverse function of the error function $\mathrm{erf}(x)$.

$$erf(z) = \frac{2}{\sqrt{\pi}} \cdot \int_0^z e^{-t^2} dt$$

> **Note:** this conversion from BER to Q is only valid given a large time offset from the optimal sampling point. The use of the nomenclature BER in this reference should therefore be carefully used. Any accurate prediction of the BER towards the center of the eye should be done using Marcum's Q function, and is outside the scope of this document.

By linearizing the bathtub, as shown in Figure 9-25, we can describe the function of the left and right hand linear parts of the bathtub in terms of an offset (HPJ) and gradient (1/GJ).

$$Q_{left}(\tau_{offset}) = (\tau_{offset} - HPJ_{left}) \cdot \frac{1}{GJ_{left}}$$

$$Q_{right}(\tau_{offset}) = (HPJ_{left} - \tau_{offset}) \cdot \frac{1}{GJ_{right}}$$

The conversion to a linearized bathtub from a measurement should be calculated using a polynomial fit algorithm for parts of the measurement made at low BERs or high Q.

[1]It is assumed that when measuring the jitter bathtub that the left and right parts of the bathtub are independent to each other, e.g. the tail of the right hand part of the bathtub and negligible effect on the left hand side of the bathtub.

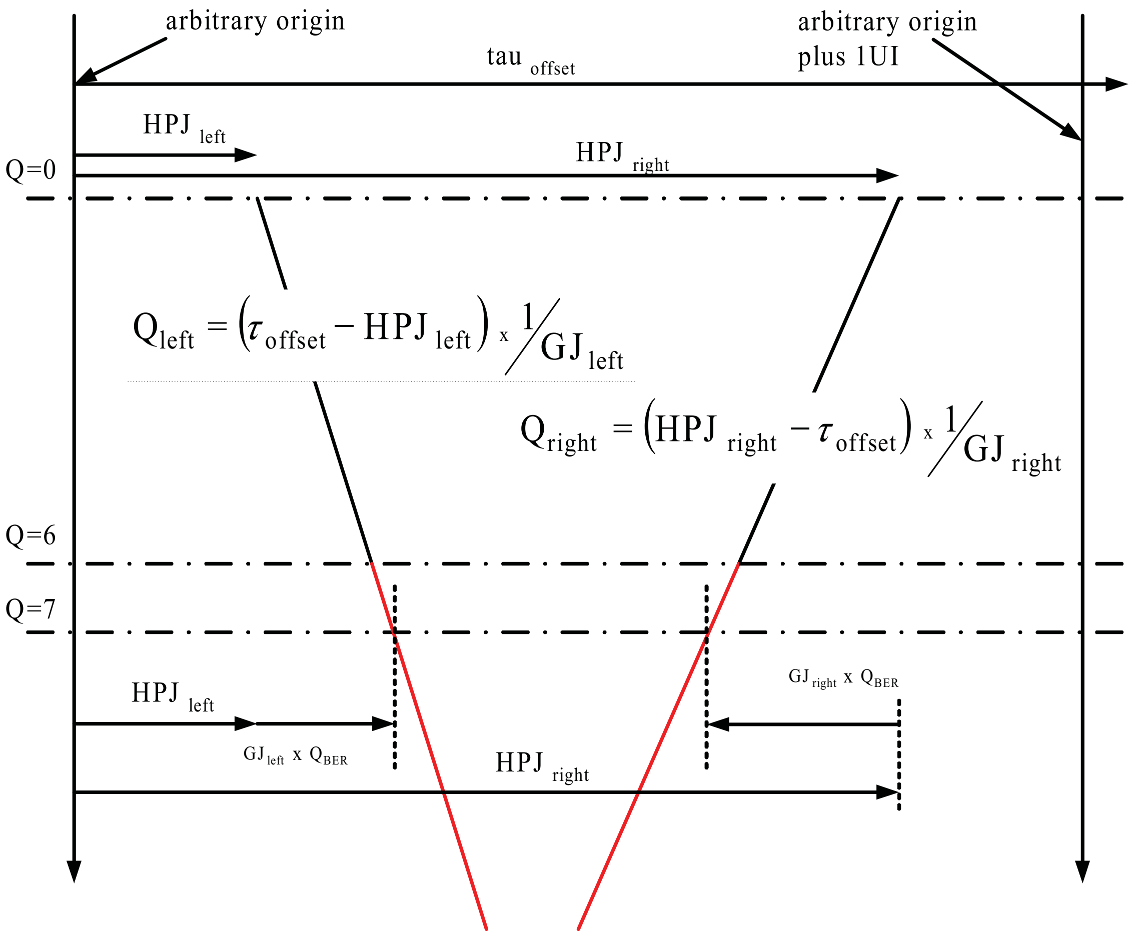

arbitrary origin
tau offset
arbitrary origin plus 1UI
HPJ left
HPJ right
Q=0
$Q_{left} = (\tau_{offset} - HPJ_{left}) \times 1/GJ_{left}$
$Q_{right} = (HPJ_{right} - \tau_{offset}) \times 1/GJ_{right}$
Q=6
Q=7
HPJ left
GJ left x Q BER
HPJ right
GJ right x Q BER

Figure 9-25. Bathtub Definition

9.7.4.7 Specification of GJ and HPJ

In this specification the left and right hand terms are combined to give a single definition as below where Q_{BER} is the Q for the BER of interest, e.g Q=7.49 for a $BER = 10^{-15}$.

$$HPJ_{total} = 1 - (HPJ_{right} - HPJ_{left})$$

$$GJ_{total} = GJ_{left} \cdot Q_{BER} + GJ_{right} \cdot Q_{BER} = 2Q_{BER} \cdot GJ_{rms}$$

$$GJ_{rms} = \frac{GJ_{left} + GJ_{right}}{2}$$

$$J_{total} = GJ_{total} + HPJ_{total}$$

9.7.4.8 Example of Bounded Gaussian

Assuming that the Cumulative Distribution Function of the jitter could be measured to the probabilities shown, Figure 9-26 shows an example of when a jitter should be classified as Correlated High Probability or Correlated Bounded Gaussian.

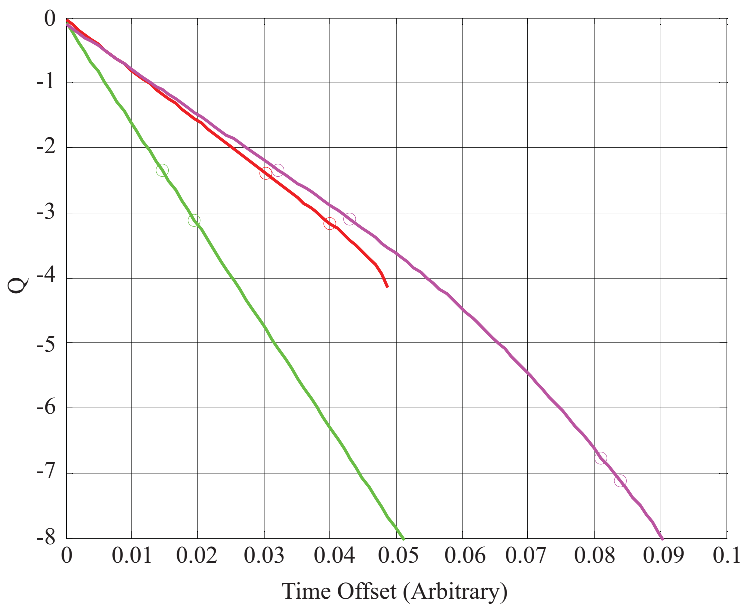


Figure 9-26. Example of Bounded Gaussian

The convolution of a true Unbounded Gaussian Jitter (green) with a Bounded Gaussian Jitter (Red) can be seen (Magenta). It can be clearly seen and measured that at a Q of -3 the Bounded Jitter is still Gaussian and the resulting convolution can be calculated using RMS addition. Below a Q of -5 the Bounding effect can be seen, and if we linearize the Bathtub we measure a non-zero High Probability Jitter and Gaussian component.

9.7.5 Statistical Eye Methodology

The following section describes the fundamental underlying the StatEye methodology. For a golden implementation please refer to the scripts on the OIF website, which are published separately.

9.7.5.1 Derivation of Cursors and Calculation of PDF

The Statistical Eye Methodology uses a channel pulse response and crosstalk pulse response in conjunction with a defined sampling jitter to generate an equivalent eye which represents the eye opening as seen by the receiver for a given probability of occurrence. This is shown in Figure 9-27.

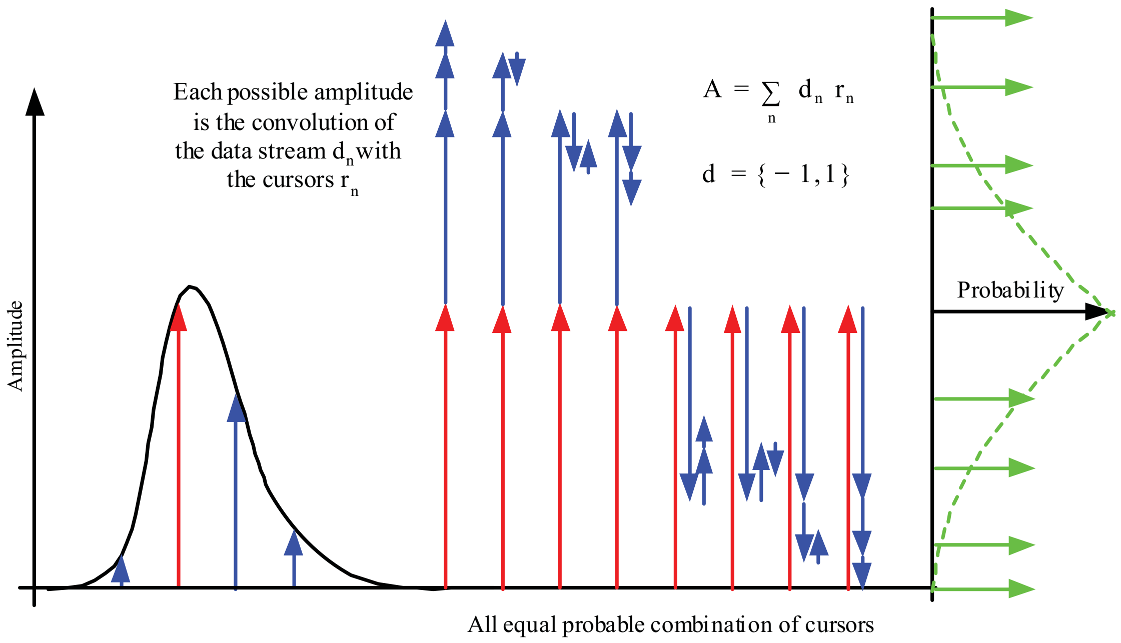

Each possible amplitude is the convolution of the data stream d_n with the cursors r_n
$A = \sum_n d_n r_n$
$d = \{-1, 1\}$
Amplitude
Probability
All equal probable combination of cursors

Figure 9-27. Statistics of Pulse Response Cursor

Given a pulse response (black left) we locate c_0 at an arbitrary point (red arrow) and measure the symbol space cursors (blue arrows),

Given a DFE the post cursors should be adjusted by negating the measured post cursors by the appropriate static coefficient of the DFE, up to the maximum number of cursors specified.

According to the exact data pattern these cursors superimpose to Inter-symbol Interference. Each possible combination of these cursors is calculated and from these combinations a histogram is generated to form the probability density function (PDF) (green).

By varying the reference sampling point for c0 as shown in Figure 9-28, the previous function is repeated and family of conditional PDFs build up.

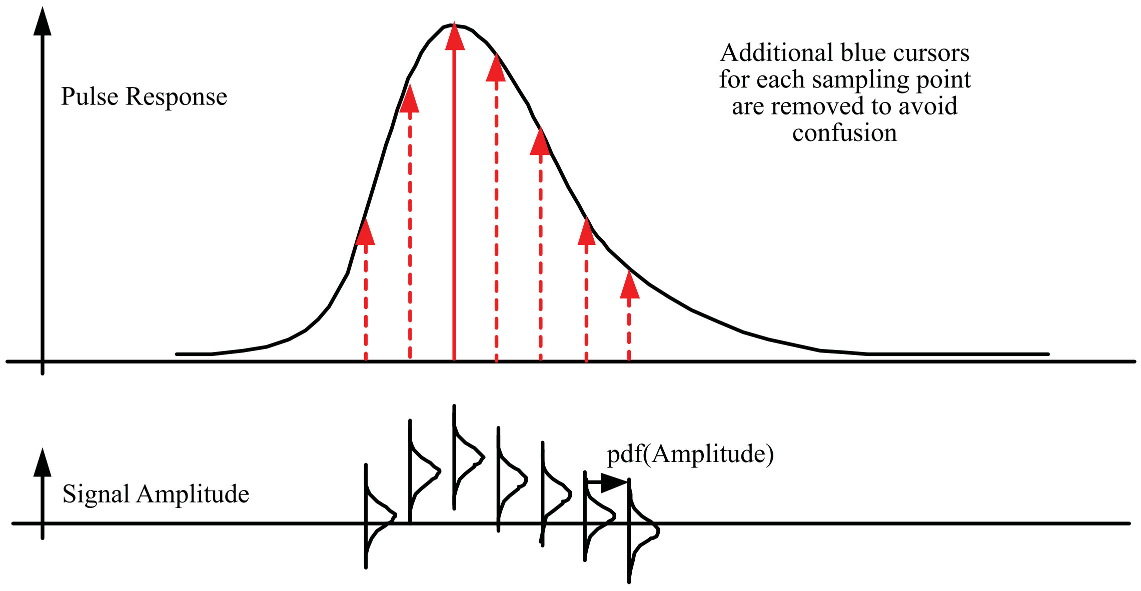

Pulse Response
Additional blue cursors for each sampling point are removed to avoid confusion
Signal Amplitude
pdf(Amplitude)

Figure 9-28. Variation of the c0 Sampling Time

This can be represented mathematically below.

Given,

$r_n(\tau)$ are the cursors of the pulse response at sampling τ

e_b is the ideal static equalization coefficients of the b tap DFE

$c(\tau)$ is the set of equalization cursors at sampling τ

$\delta(\tau) = \lim_{\varepsilon \to 0} \varepsilon|x|^{\varepsilon - 1}$ is the dirac or delta function

$d_{n, b}$ are all the possible combinations of the data stream and is either 1 or 0

$p(ISI, \tau)$ is the probability density function of the ISI for a given sample time

A similar family of PDFs are generated for the crosstalk pulse response and any other aggressors in the system using the cursor set below, noting that the entire pulse response is used.

$$c(\tau) = \left[r_{-\frac{m}{2}}(\tau) \; \dots \; r_{-1}(\tau) \; r_0(\tau) \; r_1(\tau) \; \dots \; r_{\frac{m}{2}}(\tau) \right]$$

9.7.5.2 Inclusion of Sampling Jitter

In a real system the sampling point c0 is defined by the CDR and is jittered, for the sake of standardization, by the transmitter. This jitter has a probability density function which is centered at the receiver CDR sampling point and defined the probability of each of the previous conditional PDFs occurring[1], as shown in Figure 9-29.

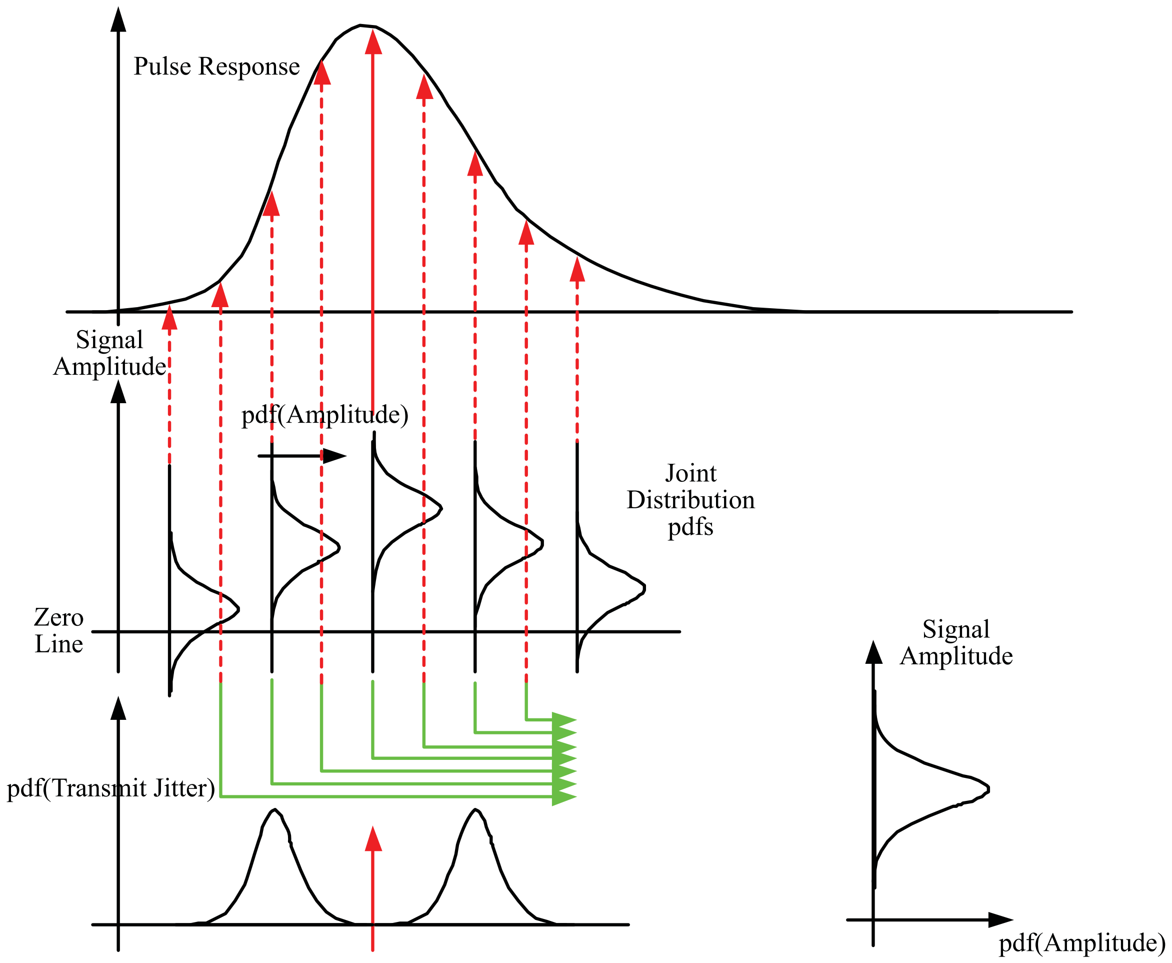


Figure 9-29. Varying the Receiver Sampling Point

By multiplying each the conditional PDFs by its associated sampling jitter probability and summing their results together, the joint probability density function at the given receiver CDR sample point can be calculated.

Given,

$p_{jitter}(\tau, w, \sigma)$ is the dual dirac probability density function of the sampling jitter in the system, as defined in Section 9.7.4.4, "Total Jitter"

$p_{crosstalk}(ISI, \tau)$ is the probability density function of the crosstalk

[1]Currently DCD effects are not taken into account

$p_{forward}(ISI, \tau)$ is the probability density function of the ISI of the forward channel

$a \otimes b$ is the convolution operative

$$p_{average}(ISI, \tau) = $$

$$\int_{-\infty}^{\infty} \{[p_{crosstalk}(ISI, \tau + \upsilon + w) \otimes p_{forward}(ISI, \tau + \upsilon)] \cdot p_{jitter}(\upsilon, w, \sigma)\} d\upsilon$$

9.7.5.3 Generation of Statistical Eye

By varying the receiver CDR sampling point a new joint probability density function, Figure 9-29 can be generated.

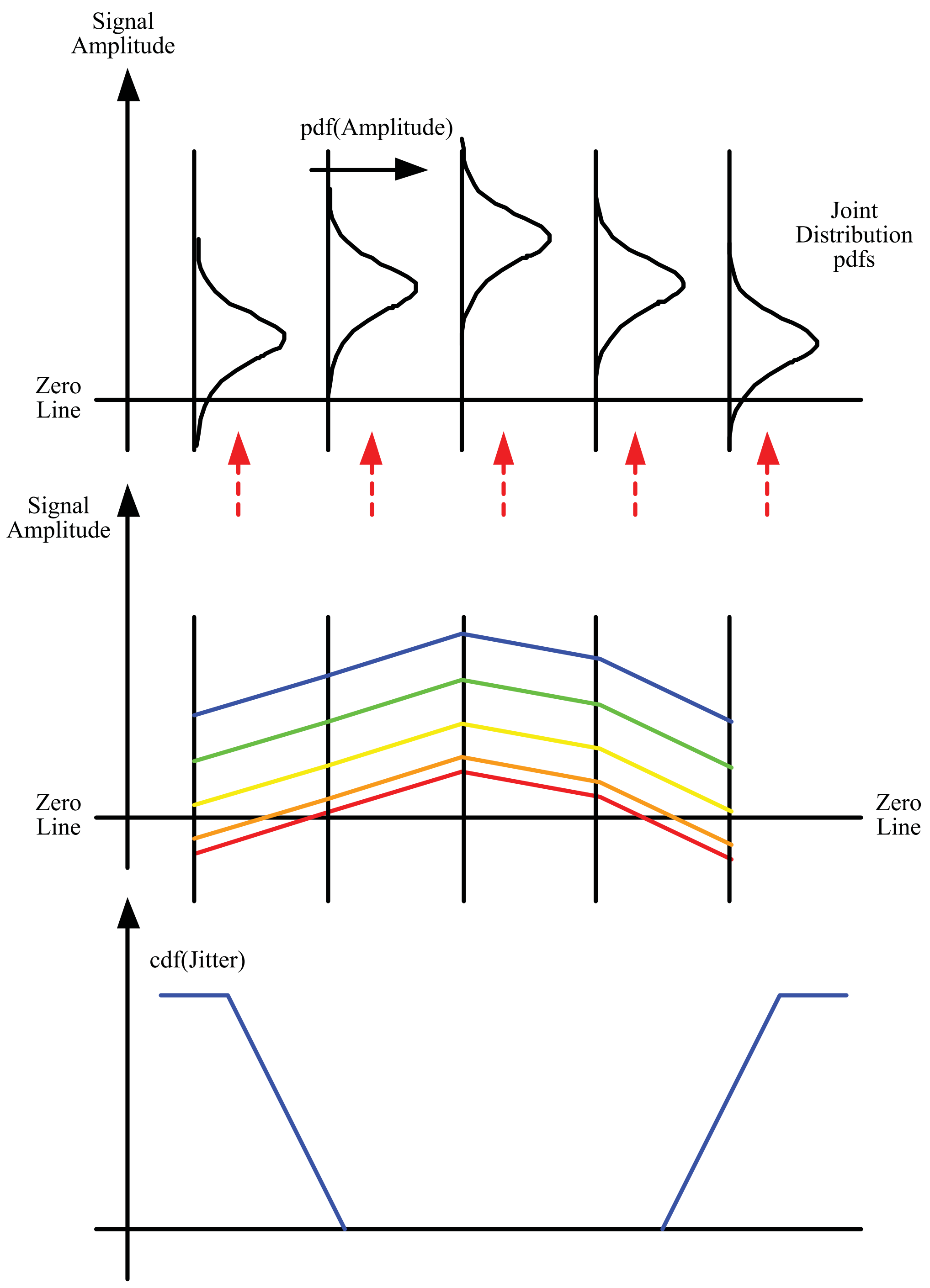


Figure 9-30. Generation of the Data Eye and Bathtub

By integrating the Joint Probability Density Function to give the Cumulative Distribution function, and creating a contour plot an equivalent of the receiver eye can be generated which shows the exact probability of obtaining a given amplitude, shown in Figure 9-30, this equivalent eye is termed the statistical eye, shown in Figure 9-31.

By only plotting the probability against time by cutting the statistical eye along the decision threshold axis can a bathtub of the jitter can be generated.

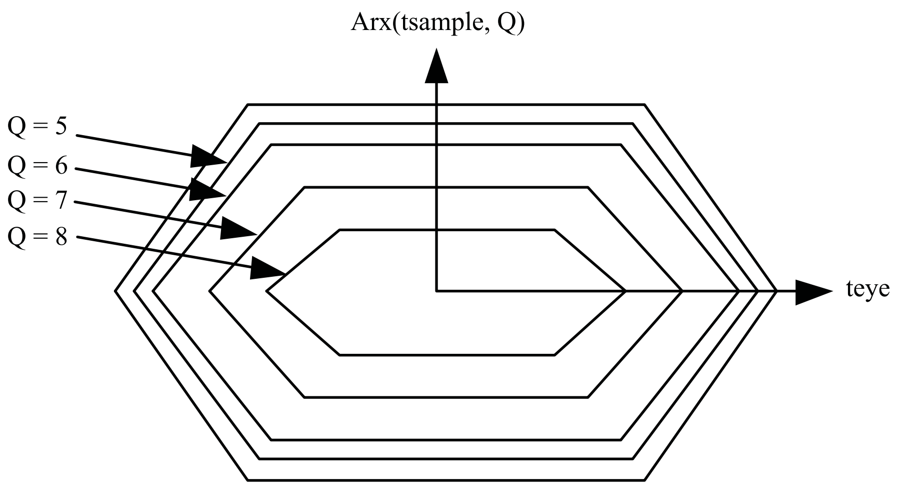


Figure 9-31. Statistical Eye

Chapter 10 1.25 Gbaud, 2.5 Gbaud, and 3.125 Gbaud LP-Serial Links

This chapter details the requirements for Level I RapidIO LP-Serial short and long run electrical interfaces of nominal baud rates of 1.25 Gbaud, 2.5 Gbaud, and 3.125 Gbaud using NRZ coding (hence 1 bit per symbol at the electrical level). A compliant device must meet all of the requirements listed below. The electrical interface is based on a high speed, low voltage logic with a nominal differential impedance of 100 Ω. Connections are point-to-point balanced differential pair and signalling is unidirectional.

The level of links defined in this section are identical to those defined in revision 1.3 of the 1x/4x LP-Serial electrical specification. The terminology has been updated to be consistent with the new level links defined in Section 9.1, "Introduction".

10.1 Level I Application Goals

The following are application requirements common to short run and long run at 1.25 Gbaud, 2.5 Gbaud, and 3.125 Gbaud:

- The electrical specifications shall support lane widths options of 1x to Nx where N=2, 4, 8, and 16.
- AC coupling at the receiver shall be specified to ensure inter-operability between transmitters and receivers of different vendors and technologies.
- A compliant device may implement any subset of baud rates contained in this section.
- A compliant device may implement either a short run transmitter, a long run transmitter, or both, at each of the baud rates that it supports.
- The clock frequency tolerance requirement for transmit and receive are ±100 ppm. The worst case frequency differences between any transmit and receive clock is 200 ppm.
- The Bit Error Ratio (BER) shall be better than 10^{-12} per lane.
- The transmitter pins shall be capable of surviving short circuit either to each other, to supply voltages, and to ground.
- The short run interface shall be capable of spanning at least 20 cm of PCB material with up to a single connector.

• The long run interface shall be capable of spanning at least 50 cm of PCB material with up to two connectors.

10.2 Equalization

At the high baud rates used by Level I LP-Serial links, the signals transmitted over a link are degraded by losses and characteristic impedance discontinuities in the interconnect media. The losses increase with increasing baud rate and interconnect media length and cause signal attenuation and inter-symbol interference that degrade the opening of the eye pattern at both the receiver input and the data decoder decision point. Depending on the baud rate and interconnect length, the degradation can be greater than that allowed by the specification.

The signal degradation can be partially negated by the use of equalization in the transmitter and/or receiver. Equalization in the transmitter can improve the eye pattern at both the receiver input and the data decoder decision point. Equalization in the receiver can only improve the eye pattern at the data decoder decision point. Equalization is likely to be required only for longer Level I interconnects and higher Level I baud rates.

The types of equalizers and, if the equalizers are adaptive, the adaptive equalizer training algorithms that may be used in Level I transmitter or receiver are subject to the following restrictions.

Equalizers that can convert a single bit error into a multiple bit burst error, such as decision feedback equalizers (DFEs), shall not be used when IDLE1 has been selected for use on the link.

The training algorithm for any adaptive equalization used by a Level I transmitter and/or receiver shall consistently train the equalizer and retain the equalizer’s training when IDLE1 is the training signal and shall consistently retain the equalizer’s training when IDLE1 has been selected for use on the link and the signal on the link is a continuous sequence of maximum length packets whose payload is either all ONES or all ZEROS.

10.3 Explanatory Note on Level I Transmitter and Receiver Specifications

AC electrical specifications are given for the transmitter and receiver. Long run and short run interfaces at three baud rates are described.

The parameters for the AC electrical specifications are guided by the XAUI electrical interface specified in Clause 47 of IEEE 802.3ae-2002.[1]

XAUI has similar application goals as serial RapidIO Level I devices as described in Section 9.5, "Common Electrical Specification". The goal of this standard is that electrical designs for Level I electrical designs can reuse XAUI, suitably modified for applications at the baud intervals and runs described herein.

10.4 Level I Electrical Specification

10.4.1 Level I Short Run Transmitter Characteristics

The key transmitter electrical specifications at compliance point T are summarized in Table 10-1 and Table 10-2 while the following sections fully detail all of the requirements.

Table 10-1. Level I SR Transmitter AC Timing Specifications

Characteristics	Symbol	Conditions	Min	Typ	Max	Units
Baud Rate	T_Baud	Section 10.4.1.2	1.25		3.125	Gbaud
Absolute Output Voltage	V_O	Section 10.4.1.3	-0.40		2.30	Volts
Output Differential Voltage (into floating load Rload = 100 Ω	T_Vdiff	Section 10.4.1.3	500		1000	mVppd
Differential Resistance	T_Rd	Section 10.4.1.5	80	100	120	W
Recommended output rise and fall times (20% to 80%)	T_tr, T_tf	Section 10.4.1.4	60			ps
Differential Output Return Loss (T_baud/10 ≤ f < T_Baud/2)	T_SDD22	Section 10.4.1.6				dB
Differential Output Return Loss (T_baud/2 ≤ f ≤ T_baud)						dB
Common Mode Return Loss (625 MHz ≤ f ≤ T_baud)	T_SCC22	Section 10.4.1.6			Note 3	dB
Transmitter Common Mode Noise[1]	T_Ncm				Note 4	mVppd
Output Common Mode Voltage	T_Vcm	Load Type 0[2]	0		2.1	V
Multiple output skew, N<=4	S_{MO}	Section 10.4.1.7			1000	ps
Multiple output skew, N>4	S_{MO}	Section 10.4.1.7			2UI +1000	ps
Unit Interval	UI		320		800	ps

NOTES:
1. For all Load Types: R_Rdin = 100 Ω ± 20 Ω. For Vcm definition, see Figure 9-1
2. Load Type 0 with min. T_Vdiff, AC-Coupling or floating load.
3. It is suggested that T_SCCC22 be -6 dB to be compatible with Level II transmitter requirements
4. It is suggested that T_Ncm be limited to 5% of T_Vdiff to be compatible with Level II transmitter requirements

Table 10-2. Level I SR Transmitter Output Jitter Specifications

Characteristic	Symbol	Condition	Min	Typ	Max	Units
Uncorrelated High Probability Jitter	T_UHPJ	Section 10.4.1.9			0.17	UIpp
Duty Cycle Distortion	T_DCD	Section 10.4.1.9			0.05	UIpp
Total Jitter	T_TJ	Section 10.4.1.9			0.35	UIpp
Eye Mask	T_X1	Section 10.4.1.9			0.17	UI
Eye Mask	T_X2	Section 10.4.1.9			0.39	UI
Eye Mask	T_Y1	Section 10.4.1.9	250			mV
Eye Mask	T_Y2	Section 10.4.1.9			500	mV

10.4.1.1 Level I SR Transmitter Test Load

All transmitter characteristics should be implemented and measured to a differential impedance of 100 Ω ± 5% at DC with a return loss of better than 20 dB from the baud rate divided by 1667 to 1.5 times the baud rate, unless otherwise noted.

10.4.1.2 Level I SR Transmitter Baud Rate

The baud rates are 1.25 Gbaud, 2.5 Gbaud, and 3.125 Gbaud with a tolerance of ±100 ppm.

10.4.1.3 Level I SR Transmitter Amplitude and Swing

Transmitter differential amplitude shall be between 500 to 1000 mVppd, inclusive, either with or without transmit emphasis. Absolute driver output voltage shall be between -0.4 V and 2.4 V, inclusive, with respect to the local ground. See Figure 9-1 for an illustration of absolute driver output voltage and definition of differential peak-to-peak amplitude.

10.4.1.4 Level I SR Transmitter Rise and Fall Times

The recommended minimum differential rise and fall time is 60 ps as measured between the 20% and 80% of the maximum measured levels; the maximum differential rise and fall times are defined by the Tx eye diagram (Figure 9-2 and Table 10-4). Shorter rise and falls may result in excessive high frequency components and increase EMI and cross talk.

10.4.1.5 Level I SR Transmitter Differential Pair Skew

It is recommended that the timing skew at the output of a LP-Serial transmitter between the two signals that comprise a differential pair not exceed 25 ps at 1.25 Gbaud, 20 ps at 2.5 Gbaud, and 15 ps at 3.125 Gbaud.

10.4.1.6 Level I SR Transmitter Output Resistance and Return Loss

Refer to Section 9.5.11, "Differential Resistance and Return Loss, Transmitter and Receiver" for the reference model for return loss. See Table 10-3 for Level I short and long run transmitter parameters. Definitions for these parameters are in Figure 9-12.

Table 10-3. Level I SR Transmitter Return Loss Parameters

Parameter	Value	Units
A0	-10	dB
f0	T_Baud/10	Hz
f1	625	MHz
f2	T_Baud	Hz
Slope	10.0	dB/dec

10.4.1.7 Level I SR Transmitter Lane-to-Lane Skew

The electrical level of lane-to-lane skew caused by the transmitter circuitry and associated routing must be less than 1000 ps for links of 4 lanes or less. Links with greater than 4 lanes must have lane-to-lane skew of less than 2 UI + 1000 ps. The transmitter lane-to-lane skew is only for the serdes Tx and does not include any effects of the channel.

10.4.1.8 Level I SR Transmitter Short Circuit Current

It is recommended that the max DC current into or out of the transmitter pins when either shorted to each other or to ground be ±100 mA when the device is fully powered up. From a hot swap point of view, the ±100 mA limit is only valid after 10 µs.

10.4.1.9 Level I SR Transmitter Template and Jitter

For each baud rate at which a transmitter is specified to operate, the output eye pattern of the transmitter shall fall entirely within the unshaded portion of the Transmitter Output Compliance Mask shown in Figure 9-2 with the parameters specified in Table 10-4. The output eye pattern of a LP-Serial transmitter that implements pre-emphasis (to equalize the link and reduce inter-symbol interference) need only comply with the Transmitter Output Compliance Mask when pre-emphasis is disabled or minimized.

Table 10-4. Level I SR Near-End (Tx) Template Intervals

Characteristics	Symbol	Near-End Value	Units
Eye Mask	T_X1	0.17	UI
Eye Mask	T_X2	0.39	UI
Eye Mask	T_Y1	250	mV
Eye Mask	T_Y2	500	mV
Eye Mask	T_Y3	N/A	mV
Uncorrelated Bounded High Probability Jitter	T_UBHPJ	0.17	UIpp
Duty Cycle Distortion	T_DCD	0.05	UIpp
Total Jitter	T_TJ	0.35	UIpp

10.4.2 Level I Long Run Transmitter Characteristics

The key transmitter electrical specifications at compliance point T are summarized in Table 10-5 and Table 10-6 while the following sub-clauses fully detail all of the requirements.

Table 10-5. Level I LR Transmitter AC Timing Specifications

Characteristics	Symbol	Conditions	Min	Typ	Max	Units
Baud Rate	T_Baud	Section 10.4.2.2	1.25		3.125	Gbaud
Absolute Output Voltage	V_O	Section 10.4.2.3	-0.40		2.30	Volts
Output Differential Voltage (into floating load Rload=100 Ω)	T_Vdiff	Section 10.4.2.3	800		1600	mVppd
Differential Resistance	T_Rd	Section 10.4.1.5	80	100	120	W
Recommended output rise and fall times (20% to 80%)	T_tr, T_tf		60			
Differential Output Return Loss (T_baud/10 ≤ f < T_Baud/2)	T_SDD22	Section 10.4.1.6				dB
Differential Output Return Loss (T_baud/2 ≤ f ≤ T_baud)						dB
Common Mode Return Loss (625 MHz ≤ f ≤ T_baud)	T_SCC22	Section 10.4.1.6			Note 3	dB
Transmitter Common Mode Noise[1]	T_Ncm				Note 4	mVppd
Output Common Mode Voltage	T_Vcm	Load Type 0[2]	0		2.1	V
Multiple output skew, N<=4	S_{MO}				1000	ps
Multiple output skew, N>4	S_{MO}				2UI+ 1000	ps

Table 10-5. Level I LR Transmitter AC Timing Specifications

Unit Interval	UI		320		800	ps

NOTES:
1. For all Load Types: R_Rdin = 100 Ω ± 20 Ω. For Vcm definition, see Figure 9-1.
2. Load Type 0 with min. T_Vdiff, AC-Coupling or floating load.
3. It is suggested that T_SCCC22 be -6 dB to be compatible with Level II transmitter requirements
4. It is suggested that T_Ncm be limited to 5% of T_Vdiff to be compatible with Level II transmitter requirements

Table 10-6. Level I LR Transmitter Output Jitter Specifications

Characteristic	Symbol	Condition	Min	Typ	Max	Units
Uncorrelated High Probability Jitter	T_UHPJ	Section 10.4.1.9			0.17	UIpp
Duty Cycle Distortion	T_DCD	Section 10.4.1.9			0.05	UIpp
Total Jitter	T_TJ	Section 10.4.1.9			0.35	UIpp
Eye Mask	T_X1	Section 10.4.1.9			0.17	UI
Eye Mask	T_X2	Section 10.4.1.9			0.39	UI
Eye Mask	T_Y1	Section 10.4.1.9	400			mV
Eye Mask	T_Y2	Section 10.4.1.9			800	mV

10.4.2.1 Level I LR Transmitter Test Load

All transmitter characteristics should be implemented and measured to a differential impedance of 100 Ω ± 5% at DC with a return loss of better than 20 dB from the baud rate divided by 1667 to 1.5 times the baud rate, unless otherwise noted.

10.4.2.2 Level I LR Transmitter Baud Rate

The baud rates are 1.25 Gbaud, 2.5 Gbaud, and 3.125 Gbaud with a tolerance of ±100 ppm.

10.4.2.3 Level I LR Transmitter Amplitude and Swing

Transmitter differential amplitude shall be between 400 to 1600 mVppd, inclusive, either with or without transmit emphasis. Absolute driver output voltage shall be between -0.4 V and 2.4 V, inclusive, with respect to the local ground. See Figure 9-1 for an illustration of absolute driver output voltage and definition of differential peak-to-peak amplitude.

10.4.2.4 Level I LR Transmitter Rise and Fall Times

The recommended minimum differential rise and fall time is 60 ps as measured between the 20% and 80% of the maximum measured levels; the maximum differential rise and fall times are defined by the Tx eye diagram (Figure 9-2 and

Table 10-8). Shorter rise and falls may result in excessive high frequency components and increase EMI and cross talk.

10.4.2.5 Level I LR Transmitter Differential Pair Skew

It is recommended that the timing skew at the output of a LP-Serial transmitter between the two signals that comprise a differential pair not exceed 25 ps at 1.25 Gbaud, 20 ps at 2.5 Gbaud and 15 ps at 3.125 Gbaud.

10.4.2.6 Level I LR Transmitter Output Resistance and Return Loss

Refer to Section 9.5.11 for the reference model for return loss. See Table 10-3 for Level I short and long run transmitter parameters. Definitions for these parameters are in Figure 9-12.

Table 10-7. Level I LR Transmitter Return Loss Parameters

Parameter	Value	Units
A0	-8	dB
f0	T_Baud/10	Hz
f1	T_Baud/2	MHz
f2	T_Baud	Hz
Slope	16.6	dB/dec

10.4.2.7 Level I LR Transmitter Lane-to-Lane Skew

The electrical level of lane-to-lane skew caused by the transmitter circuitry and associated routing must be less than 1000 ps for links of 4 lanes or less. Links with greater than 4 lanes must have lane-to-lane skew of less than 2 UI + 1000 ps. The transmitter lane-to-lane skew is only for the serdes Tx and does not include any effects of the channel.

10.4.2.8 Level I LR Transmitter Short Circuit Current

It is recommended that the max DC current into or out of the transmitter pins when either shorted to each other or to ground be ±100 mA when the device is fully powered up. From a hot swap point of view, the ±100 mA limit is only valid after 10 µs.

10.4.2.9 Level I LR Transmitter Template and Jitter

For each baud rate at which a LP-Serial transmitter is specified to operate, the output eye pattern of the transmitter shall fall entirely within the unshaded portion of the Transmitter Output Compliance Mask shown in Figure 9-2 with the parameters specified in Table 10-4. The output eye pattern of a LP-Serial transmitter that implements pre-emphasis (to equalize the link and reduce inter-symbol interference) need only comply with the Transmitter Output Compliance Mask when pre-emphasis is disabled or minimized.

Table 10-8. Level I LR Near-End (Tx) Template Intervals

Characteristics	Symbol	Near-End Value	Units
Eye Mask	T_X1	0.17	UI
Eye Mask	T_X2	0.39	UI
Eye Mask	T_Y1	400	mV
Eye Mask	T_Y2	800	mV
Eye Mask	T_Y3	N/A	mV
Uncorrelated Bounded High Probability Jitter	T_UBHPJ	0.17	UIpp
Duty Cycle Distortion	T_DCD	0.05	UIpp
Total Jitter	T_TJ	0.35	UIpp

10.4.3 Level I Receiver Specifications

Level I LP-Serial receiver electrical and timing specifications are stated in the text and tables of this section.

Table 10-9. Level I Receiver Electrical Input Specifications

Characteristic	Symbol	Conditions	Min	Typ	Max	Units
Rx Baud Rate (1.25 Gbaud)	R_Baud			1.250		Gbaud
Rx Baud Rate (2.5 Gbaud)				2.500		Gbaud
Rx Baud Rate (3.125 Gbaud)				3.125		Gbaud
Absolute Input Voltage	R_Vin	Section 10.4.3.4				
Input Differential voltage	R_Vdiff	Section 10.4.3.3	200		1600	mVppd
Differential Resistance	R_Rdin	Section 10.4.3.7	80	100	120	W
Differential Input Return Loss (100 MHz ≤ f ≤ R_Baud/2)	R_SDD11	Section 10.4.3.7				dB
Differential Input Return Loss (R_Baud/2 ≤ f ≤ R_Baud)						
Common mode Input Return Loss (100 MHz to 0.8 *R_Baud)	R_SCC11	Section 10.4.3.7				dB
Termination Voltage[1,2]	R_Vtt	R_Vtt floating[4]	Not Specified			V
Input Common Mode Voltage[1,2]	R_Vrcm	R_Vtt floating[3,4,]	-0.05		1.85	V
Wander divider (in Figure 9-8 & Figure 9-8)	n			10		

NOTES:
1. Input common mode voltage for AC-coupled or floating load input with min. T_Vdiff,
2. Receiver is required to implement at least one of specified nominal R_Vtt values, and typically implements only one of these values. Receiver is only required to meet R_Vrcm parameter values that correspond to R_Vtt values supported.
3. Input common mode voltage for AC-coupled or floating load input with min. T_Vdiff.
4. For floating load, input resistance must be ≥ 1kΩ.

Table 10-10. Level I Receiver Input Jitter Tolerance Specifications

Characteristic	Symbol	Conditions	Min	Typ	Max	Units
Bit Error Ratio	BER				10^{-12}	
Bounded High Probability Jitter	R_BHPJ	Section 10.4.3.8			0.37	UIpp
Sinusoidal Jitter, maximum	R_SJ-max	Section 10.4.3.8			8.5	UIpp
Sinusoidal Jitter, High Frequency	R_SJ-hf	Section 10.4.3.8			0.1	UIpp
Total Jitter (Does not include Sinusoidal Jitter)	R_TJ	Section 10.4.3.8			0.55	UIpp
Total Jitter Tolerance[1]	R_JT				0.65	UIpp
Eye Mask	R_X1	Section 10.4.3.8			0.275	UI
Eye Mask	R_Y1	Section 10.4.3.8			100	mV
Eye Mask	R_Y2	Section 10.4.3.8			800	mV

NOTES:
1. Total jitter is composed of three components, deterministic jitter, random jitter and single frequency sinusoidal jitter. The sinusoidal jitter may have any amplitude and frequency in the unshaded region of Figure 10-1. The sinusoidal jitter component is included to ensure margin for low frequency jitter, wander, noise, crosstalk and other variable system effects.

10.4.3.1 Level I Receiver Input Baud Rate

All devices shall work at either 1.25 Gbaud, 2.5 Gbaud, or 3.125 Gbaud or any combination of these baud rates with the baud rate tolerance as per Section 10.4.1.2.

10.4.3.2 Level I Receiver Reference Input Signals

Reference input signals to the receiver have the characteristics determined by compliant transmitter. The reference input signal must satisfy the transmitter near-end template and jitter given in Figure 9-2 and Table 10-1 (Table 10-5), Table 10-2 (Table 10-6), and Table 10-3 (Table 10-7) for short run (long run) as well as the far-end eye template and jitter given in Figure 9-5 and Table 10-13, with the differential load impedance of 100 Ω ± 1% at DC with a return loss of better than 20 dB from baud rate divided by 1667 to 1.5 times the baud rate. Note that the input signal might not meet either of these templates when the actual receiver replaces this load.

10.4.3.3 Level I Receiver Input Signal Amplitude

The receiver shall accept differential input signal amplitudes produced by compliant transmitters connected without attenuation to the receiver. This may be larger than the 1600 mVppd maximum of the transmitter due to output/input impedances and reflections.

The minimum input amplitude is defined by the far-end transmitter template, the actual receiver input impedance and the loss of the actual PCB. Note that the far-end transmitter template is defined using a well controlled load impedance, however the real receiver is not, which can leave the receiver input signal smaller than the minimum 200 mVppd.

10.4.3.4 Level I Receiver Absolute Input Voltage

The voltage levels at the input of an AC coupled receiver (if AC coupling is done within the receiver) or at the Tx side of the external AC coupling cap (if AC coupling is done externally) shall be between -0.40 V to 2.30 V, inclusive, with respect to local ground.

10.4.3.5 Level I Receiver Input Common Mode Impedance

AC coupling is to be considered part of the receiver for the purposes of this specification unless explicitly stated otherwise. It should be noted that various methods for AC coupling are allowed (for example, internal to the chip or done externally). See Section 9.5.13 for more information.

10.4.3.6 Level I Receiver Input Lane-to-Lane Skew

Refer to Section 9.5.9.

10.4.3.7 Level I Receiver Input Resistance and Return Loss

Refer to Section 9.5.11 for the reference model for return loss. See Table 10-11 for Level I receiver parameters. Definitions for these parameters are in Figure 9-12.

Table 10-11. Level I Input Return Loss Parameters

Parameter	Value	Units
A0	-8	dB
f0	100	MHz
f1	$\text{RBaud} \times \frac{1}{2}$	Hz
f2	R_Baud	Hz
Slope	16.6	dB/dec

Receiver input impedance shall result in a differential return loss better that -8 dB and a common mode return loss better than -6 dB from 100 MHz to (0.5)*(R_Baud Frequency). This includes contributions from on-chip circuitry, the chip package and any off-chip components related to the receiver. AC coupling components are included in this requirement. The reference impedance for return loss measurements is 100 Ω resistive for differential return loss and 25 Ω resistive for common mode.

10.4.3.8 Level I Receiver Input Jitter Tolerance

The DUT shall be measured to have a BER better than specified for stressed signal with a confidence level of three sigma. Therefore the receiver shall tolerate at least the far-end eye template and jitter requirements as given in Figure 9-5 and Table 10-10 with an additional SJ with any frequency and amplitude defined by the mask of Figure 10-1 where the minimum and maximum total wander amplitude are 0.1 UIpp and 8.5 UIpp respectively. This additional SJ component is intended to ensure margin for wander, hence is over and above any high frequency jitter from Table 10-13.

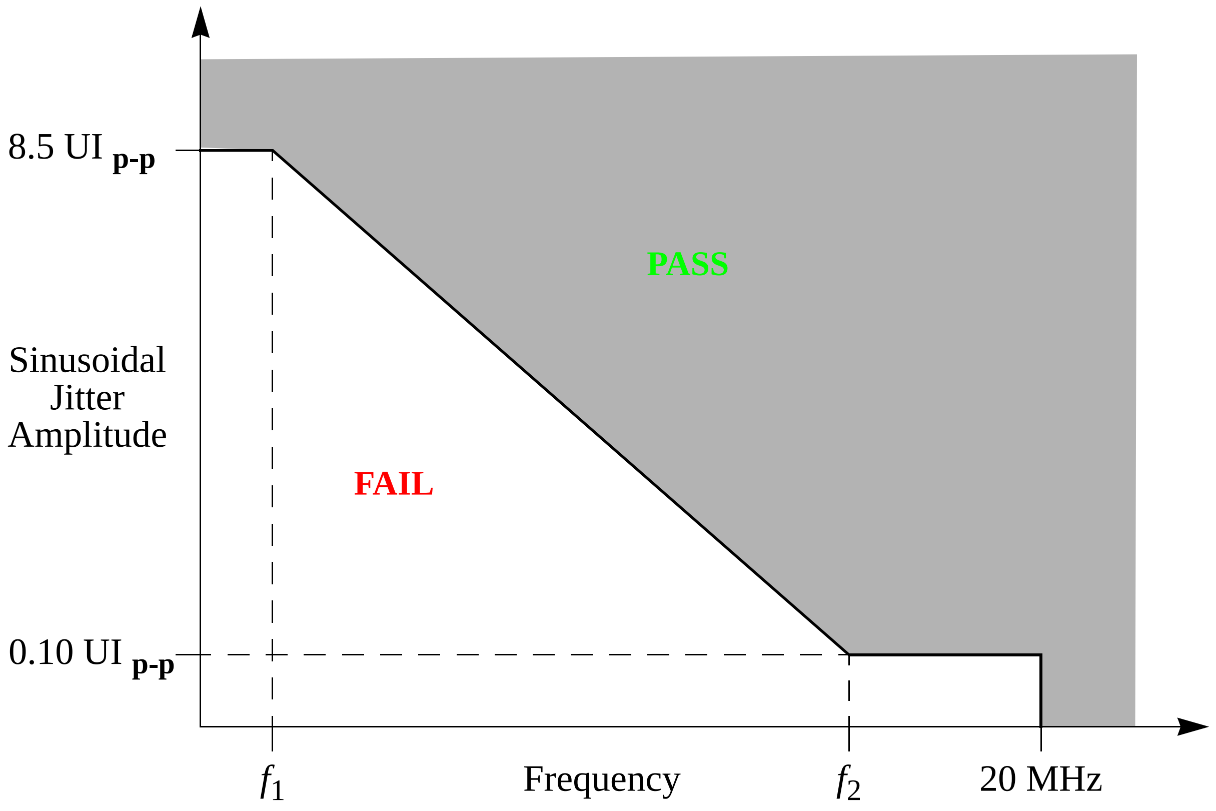


Figure 10-1. Single Frequency Sinusoidal Jitter Limits

Table 10-12 defines the low and high knee frequency for Level I links where the baud rates are defined as in Section 10.4.3.1.

Table 10-12. Level I Single Frequency Sinusoidal Jitter Limits Knee Frequencies

Receiver Data Baud Rate (Gbaud)	f_1 (kHz)	f_2 (kHz)
1.25	8.82	750
2.5	17.6	1500
3.125	22.1	1875

For each baud rate at which a LP-Serial receiver is specified to operate, the receiver shall meet the corresponding Bit Error Ratio specification in Table 10-10 when the eye pattern of the receiver test signal (exclusive of sinusoidal jitter) falls entirely within the unshaded portion of the Receiver Input Compliance Mask shown in Figure 9-5 with the parameters specified in Table 10-13. The eye pattern of the receiver test signal is measured at the input pins of the receiving device with the device replaced with a 100 Ω ± 5% differential resistive load.

Table 10-13. Level I Far-End (Rx) Template Intervals

Characteristics	Symbol	Far-End Value	Units
Eye Mask	R_X1	0.275	UI
Eye Mask	R_Y1	100	mV
Eye Mask	R_Y2	800	mV
High Probability Jitter	R_HPJ	0.37	UIpp
Total Jitter (Does not include Sinusoidal Jitter)	R_TJ	0.55	UIpp

10.5 Level I Measurement and Test Requirements

Since the LP-Serial electrical specification is guided by the XAUI electrical interface specified in Clause 47 of IEEE 802.3ae-2002, the measurement and test requirements defined here are similarly guided by Clause 47. In addition, the CJPAT test pattern defined in Annex 48A of IEEE802.3ae-2002 is specified as the test pattern for use in transmitter eye pattern and jitter measurements. Annex 48B of IEEE802.3ae-2002 is recommended as a reference for additional information on jitter test methods.

10.5.1 Level I Transmitter Measurements

10.5.1.1 Level I Eye Template Measurements

For the purpose of transmitter eye template measurements, the effects of a single-pole high pass filter with a 3 dB point at (Baud Frequency)/1667 is applied to the jitter. The data pattern for template measurements is the Continuous Jitter Test Pattern (CJPAT) defined in Annex 48A of IEEE802.3ae. All lanes of the LP-Serial link shall be active in both the transmit and receive directions, and opposite ends of the links shall use asynchronous clocks. N lane implementations shall use CJPAT as defined in Annex 48A. Single lane implementations shall use the CJPAT sequence specified in Annex 48A for transmission on lane 0. The amount of data represented in the eye shall be adequate to ensure that the bit error ratio is less than 10^{-12}. The eye pattern shall be measured with AC coupling and the compliance template centered at 0 Volts differential. The left and right edges of the template shall be aligned with the mean zero crossing points of the measured data eye. The load for this test shall be 100 Ω resistive ± 5% differential to 2.5 GHz.

10.5.1.2 Level I Jitter Test Measurements

For the purpose of transmitter jitter measurement, the effects of a single-pole high pass filter with a 3 dB point at (Baud Frequency)/1667 is applied to the jitter. The data pattern for jitter measurements is the Continuous Jitter Test Pattern (CJPAT) pattern defined in Annex 48A of IEEE802.3ae. All lanes of the LP-Serial link shall be active in both the transmit and receive directions, and opposite ends of the links shall use asynchronous clocks. N lane implementations shall use CJPAT as defined in Annex 48A. Single lane implementations shall use the CJPAT sequence specified in Annex 48A for transmission on lane 0. Jitter shall be measured with AC coupling and at 0 Volts differential. Jitter measurement for the transmitter (or for calibration of a jitter tolerance setup) shall be performed with a test procedure resulting in a BER curve such as that described in Annex 48B of IEEE802.3ae.

10.5.1.3 Level I Transmit Jitter Load

Transmit jitter is measured at the driver output when terminated into a load of 100 Ω resistive ± 5% differential to 2.5 GHz.

10.5.2 Level I Receiver Jitter Tolerance

Jitter tolerance is measured at the receiver using a jitter tolerance test signal. This signal is obtained by first producing the sum of deterministic and random jitter defined in Section 10.4.3 and then adjusting the signal amplitude until the data eye contacts the 4 points of the minimum eye opening of the receive template shown in Table 9-4 and Table 10-13. Note that for this to occur, the test signal must have vertical waveform symmetry about the average value and have horizontal symmetry (including jitter) about the mean zero crossing. Eye template measurement requirements are as defined above. Random jitter is calibrated using a high pass filter with a low frequency corner at 20 MHz and a 20 dB/decade rolloff below this. The required sinusoidal jitter specified in Section 10.4.3 is then added to the signal and the test load is replaced by the receiver being tested.

Chapter 11 5 Gbaud and 6.25 Gbaud LP-Serial Links

This chapter details the requirements for Level II RapidIO LP-Serial short, medium, and long run electrical interfaces of nominal baud rates of 5 Gbaud and 6.25 Gbaud using NRZ coding (hence 1 bit per symbol at the electrical level). A compliant device must meet all of the requirements listed below. The electrical interface is based on a high speed low voltage logic with a nominal differential impedance of 100 Ω. Connections are point-to-point balanced differential pair and signaling is unidirectional.

11.1 Level II Application Goals

11.1.1 Common to Level II Short run, Medium run and Long run

The following are application requirements common to short run, medium run and long run Level II links at 5 Gbaud and 6.25 Gbaud:

- The electrical specifications shall support lane widths options of 1x, 2x, 4x, 8x and 16x.
- Both AC coupled and DC coupled links options shall be specified. A compliant device must implement AC coupling and may implement DC coupling as an option.
- A compliant device may implement any subset of baud rates contained in this chapter.
- A compliant device may implement either a short run transmitter, a long run transmitter, or both, at each of the baud rates that it supports.
- A compliant device may implement either a short run receiver or a long run receiver at each of the baud rates that it supports.
- The clock frequency tolerance requirement for transmit and receive are ±100 ppm. The worst case frequency differences between any transmit and receive clock is 200 ppm.
- The Bit Error Ratio (BER) shall be better than 10^{-15} per lane but the test requirements will be to verify 10^{-12} per lane.

- Transmitters and receivers used on short, medium and long run links shall inter-operate for path lengths up to 20 cm.
- Transmitters and receivers used on medium and long run links shall inter-operate for path lengths up to 60 cm.
- The transmitter pins shall be capable of surviving short circuit either to each other, to supply voltages, and to ground.

11.1.2 Application Goals for Level II Short Run

- The short run interface shall be capable of spanning at least 20 cm of PCB material with up to a single connector.

11.1.3 Application Goals for Level II Medium Run

- The medium run interface shall be capable of spanning at least 60 cm of PCB material with up to two connectors.
- An AC coupled receiver used for a medium run shall be inter-operable with an AC coupled short run transmitter
- An AC coupled transmitter used for a medium run shall be inter-operable with an AC coupled short run receiver, provided that the signal swing values are lowered. This implies that the signal swing is configurable.
- The medium run PHY may use techniques such as increased signal swing and linear equalization to accommodate medium run backplane applications, where the receiver eye may be closed.

11.1.4 Application Goals for Long Run

- The long run interface shall be capable of spanning at least 100 cm of PCB material with up to two connectors.
- An AC coupled long run receiver shall be inter-operable with an AC coupled short or medium run transmitter
- An AC coupled long run transmitter shall be inter-operable with an AC coupled short run receiver provided that the signal swing values are lowered. This implies that the signal swing is configurable.
- The long run PHY may use techniques such as increased signal swing, linear equalization, and Decision Feedback Equalizer, designed to accommodate longer run backplane applications, where the receiver eye may be closed.
- A long run transmitter and receiver is intended to accommodate ‘legacy’ long run RapidIO 1.3 backplanes of at least 60 cm with up to two connectors that can operate at data rates up to 6.25 Gbaud.

11.1.5 Explanatory Note on Transmitter and Receiver Specifications

AC electrical specifications are given for transmitters and receivers. Long run, medium run and short run interfaces at two baud rates are described.

The parameters for the AC electrical specifications are guided by the OIF CEI Electrical and Jitter Inter-operability agreement for CEI-6G-SR and CEI-6G-LR[Reference 2].

OIF CEI-6G-SR and CEI-6G-LR have similar application goals to serial RapidIO, as described in Section 11.1, "Level II Application Goals". The goal of this standard is that electrical designs for serial RapidIO can reuse electrical designs for OIF CEI-6G, suitably modified for applications at the baud intervals and runs described herein.

11.2 Equalization

At the high baud rates used by Level II LP-Serial links, the signals transmitted over a link are degraded by losses and characteristic impedance discontinuities in the interconnect media. The losses increase with increasing baud rate and interconnect media length and cause signal attenuation and inter-symbol interference that degrade the opening of the eye pattern at both the receiver input and the data decoder decision point. Depending on the baud rate and interconnect length, the degradation can be greater than that allowed by the specification.

The signal degradation can be partially negated by the use of equalization in the transmitter and/or receiver. Equalization in the transmitter can improve the eye pattern at both the receiver input and the data decoder decision point. Equalization in the receiver can only improve the eye pattern at the data decoder decision point. Some degree of equalization is required by most Level II interconnects.

The types of equalizers and, if the equalizers are adaptive, the adaptive equalizer training algorithms that may be used in a Level II 5.0 Gbaud transmitter or receiver are subject to the following restrictions.

Equalizers that can convert a single bit error into a multiple bit burst error, such as decision feedback equalizers (DFEs), shall not be used when IDLE1 has been selected for use on the link.

The training algorithm for any adaptive equalization used by a Level II transmitter and/or receiver shall consistently train the equalizer and retain the equalizer’s training when IDLE1 is the training signal and shall consistently retain the equalizer’s training when IDLE1 has been selected for use on the link and the signal on the link is a continuous sequence of maximum length packets whose payload is either all ONES or all ZEROS.

The above restrictions on the types of equalizers and adaptive equalizer training algorithms do not apply to Level II transmitters and receivers operating at Baud Rate Class 2.

11.3 Link Compliance Methodology

11.3.1 Overview

A serial link is comprised of a transmitter, a receiver, and a channel which connects them. Typically, two of these are normatively specified, and the third is informatively specified. In this specification, the transmitter and channel are normatively specified, while the receiver is informatively specified.

This specification follows the OIF inter-operability or compliance methodology and is based on using transmitter and receiver reference models, measured channel S-parameters, eye masks, and calculated “statistical eyes”. These “statistical eyes” are determined by the reference models and measured channel S-parameters using publicly available StatEye MATLAB® scripts and form the basis for identifying compliant transmitters and channels. Compliant receivers are identified through a BER test.

Reference models are used extensively because at 5 Gbaud and 6.25 Gbaud data rates the incoming eye at the receiver may be closed. This prevents specifying receiver compliance through receiver eye masks as is typically done at lower data rates.

11.3.2 Reference Models

The OIF serial link reference model is shown in Figure 11-1. The reference models are simple models of the transmitter and receiver equalization with the effects of amplitude, return loss, and bandwidth included. These models do not include any other aspects of transmitter or receiver performance.

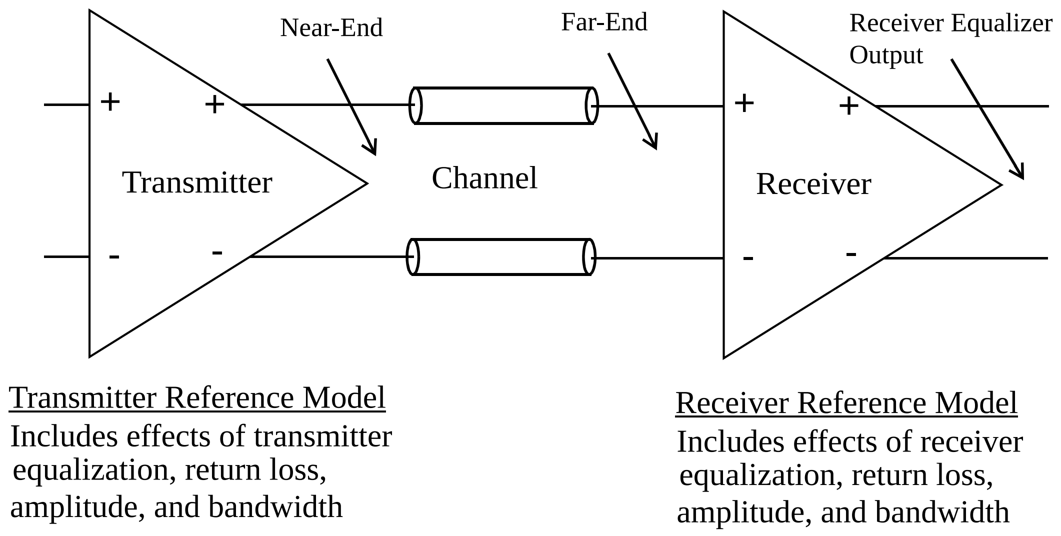

Near-End
Far-End
Receiver Equalizer Output
+
+
-
-
Transmitter
Channel
+
+
-
-
Receiver
Transmitter Reference Model
Includes effects of transmitter equalization, return loss, amplitude, and bandwidth
Receiver Reference Model
Includes effects of receiver equalization, return loss, amplitude, and bandwidth

Figure 11-1. OIF Reference Model

There are three target channel run goals in this specification which require various amounts of equalization. These different goals can be met using two transmitter and two receiver reference models. The run goals are short (20 cm), medium (60 cm), and long (100 cm). The reference models for each of the run goals are based on combining short and long run transmitter and receiver models as shown in Table 11-1.

Table 11-1. Reference Models

Run	Tx Reference Model	Rx Reference Model
Short	Short	Short
Medium	Long	Short
Long	Long	Long
NOTES: Transmitter Reference Models Short: 1 tap with ≤ 3 dB post cursor emphasis Long: 1 tap with ≤ 6 dB of either pre or post cursor emphasis Receiver Reference Model Short: Single pole, Single zero with ≤ 4 dB max gain Long: 5 tap DFE		

11.3.3 Channel Compliance

A compliant channel is determined using the appropriate transmitter and receiver reference model, measured S-parameters for the channel under consideration, and the StatEye script. A compliant channel is one that produces a receiver equalizer output “statistical eye” which meets a BER ≤ 10^{-15} using StatEye.

11.3.4 Transmitter Compliance

The experimental setup for transmitter compliance is shown in Figure 11-2. The shown setup consists of the transmitter under test connected to a compliant channel terminated with a 100 Ω differential load. OIF requires the compliant channel used in verifying transmitter compliance use at least half of the available transmitter emphasis to produce an open eye at the far-end of the channel.

Using the shown setup, the following three conditions shall be met for compliant transmitters:

1. After optimally adjusting the transmitter amplitude and emphasis to produce the most open far-end eye (given the transmitter emphasis constraint), the measured far-end eye must be equal or better than the calculated far-end eye as produced by StatEye.
2. The high frequency transmit jitter measured at the near-end must meet specification.
3. The measured near-end transmit eye mask must meet the specified near-end eye mask.

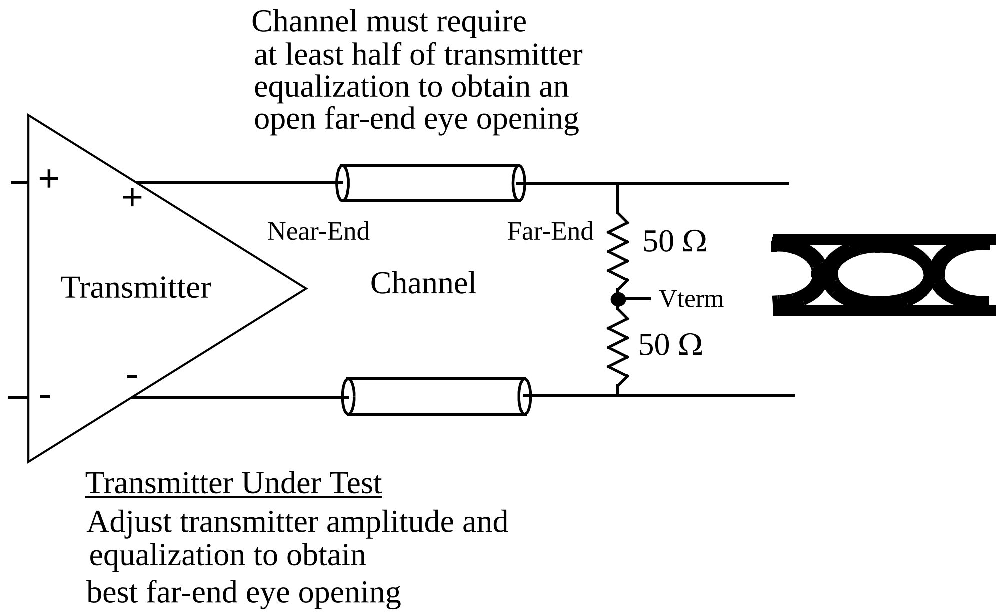

Channel must require at least half of transmitter equalization to obtain an open far-end eye opening
+
+
Transmitter
-
-
Near-End
Far-End
Channel
50 Ω
Vterm
50 Ω
Transmitter Under Test
Adjust transmitter amplitude and equalization to obtain best far-end eye opening

Figure 11-2. Transmitter Compliance Setup

11.3.5 Receiver Compliance

The experimental setup for receiver compliance is shown in Figure 11-3. The shown setup consists of a compliant channel connected to the receiver under test. To verify the receiver under test, the receiver must meet a BER < 10^{-12} with a stressed input eye mask. OIF does not place any requirements on the channel used in this measurement other than it must be compliant.

The input stressed eye used in this measurement includes sinusoidal, high probability, and Gaussian jitter as defined in the appropriate sections of this specification, along with any necessary additive crosstalk. Additive crosstalk is used to insure that the receiver under test is adequately stressed if a low loss channel is used in the measurement.

The additive input crosstalk signal is determined using the channel S-parameters, receiver reference model, and the StatEye script. It must be of amplitude such that the resulting receiver equalizer output eye, given the channel, jitter, and crosstalk, is as close as feasible in amplitude when compared to the defined minimum amplitude used for channel compliance.

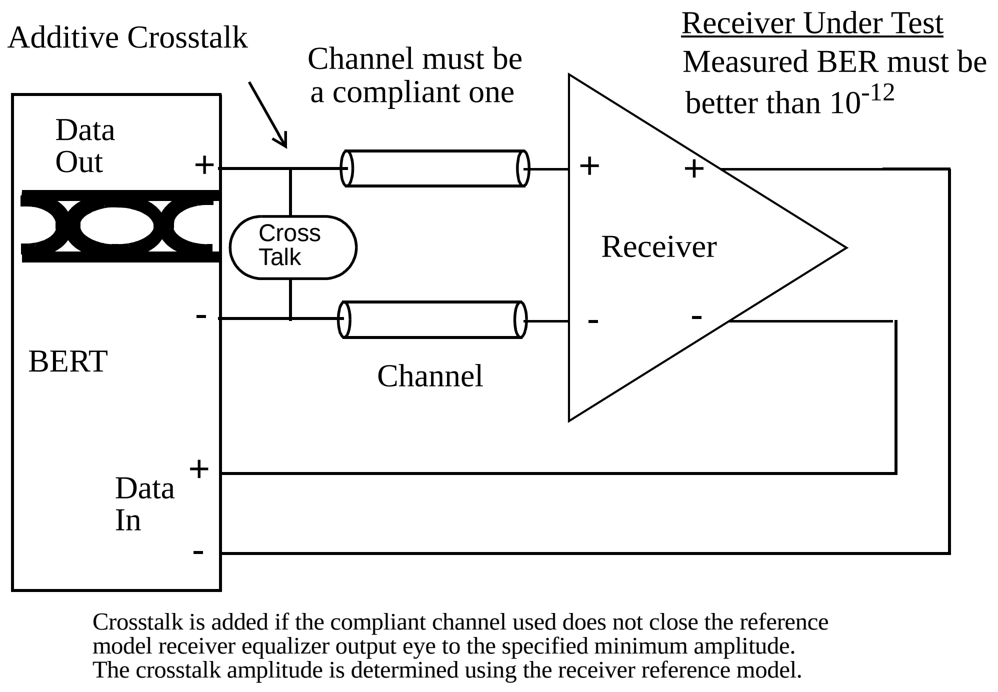

Additive Crosstalk
Channel must be a compliant one
Receiver Under Test
Measured BER must be better than 10^{-12}
Data Out
+
-
Cross Talk
Receiver
+
+
-
-
BERT
Channel
Data In
+
-
Crosstalk is added if the compliant channel used does not close the reference model receiver equalizer output eye to the specified minimum amplitude. The crosstalk amplitude is determined using the receiver reference model.

Figure 11-3. Receiver Compliance Setup

11.4 Level II Short Run Interface - General Requirements

11.4.1 Jitter and Inter-operability Methodology

This section describes the requirements for inter-operability testing of the electrical interfaces used to implement a Short Run link. The LP-Serial 5 Gbaud and 6.25 Gbaud short run interfaces use Method C, described in CEI sub-clause 2.2. This sub-clause defines the inter-operability methodology specifically for interfaces where transmit emphasis may be used and the receiver eye requires Linear Continuous Time equalization (from channel inter-operability point of view) to be open to within the BER of interest.

11.4.1.1 Level II SR Defined Test Patterns[1]

A free running PRBS31 polynomial [ITU-T 0.150] shall be used for the testing of jitter tolerance and output jitter compliance.

11.4.1.2 Level II SR Channel Compliance

The following steps shall be made to identify which channels are to be considered compliant:

1. The forward channel and significant crosstalk channels shall be measured using a network analyzer for the specified baud rate (see Section 11.7.4.5, "Network Analysis Measurement" for a suggested method). Differential S-parameters will be used to represent the characteristics of this channel.
2. The reference transmitter shall be a single post tap transmitter, with ≤ 3 dB of emphasis and infinite precision accuracy.
3. A Tx edge rate filter: a single pole 20 dB/dec low pass at 75% of baud rate, this is to emulate a Tx -3 dB bandwidth at $^3/_4$ baud rate.
4. A transmit amplitude of 400 mVppd shall be used.
5. Additional Uncorrelated Bounded High Probability Jitter of 0.15 UIpp (emulating part of the Tx jitter).
6. Additional Uncorrelated Unbounded Gaussian Jitter of 0.15 UIpp (emulating part of the Tx jitter)
7. The baud rate shall be 5 Gbaud or 6.25 Gbaud.
8. The reference transmitter shall use the worst case transmitter return loss at the baud frequency. In order to construct the worse case transmitter return loss, the reference transmitter should be considered to be a parallel R and C, where R is the defined maximum allowed DC resistance of the interface and C is increased until the defined maximum Return Loss at the baud frequency is reached. The transmitter return loss is specified in Section 11.4.2.1.6, "Level II SR Transmitter Output Resistance and Return Loss".
9. An ideal receiver filter of the form in CEI Section 9.6.7, "Time Continuous Transverse Filter". The reference receiver uses a continuous-time equalizer with 1 zero and 1 pole in the region of baudrate/100 to baudrate. Additional parasitic zeros and poles must be considered part of the receiver vendor’s device and be dealt with as they are for the reference receiver. Pole and Zero values have infinite precision accuracy. Maximum required gain/attenuation shall be less than or equal to 4 dB.
10. The reference receiver shall use a sampling point defined at the midpoint between the average zero crossings of the differential signal.

[1]All descriptions of PRBS31 imply the standard polynomial as described in [Reference 3]

11. The reference receiver shall use the worst case receiver return loss at the baud frequency. In order to construct the worse case receiver return loss, the reference receiver should be considered to be a parallel R and C, where R is the defined maximum allowed DC resistance of the interface and C is increased until the defined maximum Return Loss at the baud frequency is reached. The receiver return loss is specified in Section 11.4.2.2.7, "Level II SR Receiver Input Resistance and Return Loss".
12. The opening of the eye shall be calculated using Statistical Eye Analysis methods, as per Section 9.7.5, "Statistical Eye Methodology", and confirmed to be within the requirements as specified in Table 11-9 at the required BER, 10^{-15}.

11.4.1.3 Level II SR Transmitter Inter-operability

The following step shall be made to identify which transmitters are to be considered compliant:

1. It shall be verified that the measured eye is equal or better than the calculated eye for the given measurement probability Q (see Annex B.3, “Eye Mask Adjustment for Sampling Oscilloscopes") for a suggested method of calculating Q given a measurement population), given:
   – A “compliance” channel as per Section 11.4.1.2, "Level II SR Channel Compliance" that required at least half the maximum transmit emphasis.
   – Using this channel the transmitter shall be then be optimally adjusted and the resulting eye measured (see Section 11.7.4.6, "Eye Mask Measurement Setup" for a suggested method).
   – Using this channel the statistical eye shall then be calculated, as per CEI Section 9.7.5, "Statistical Eye Methodology", using the maximum defined transmit jitter and the actual transmitter's amplitude and emphasis.
2. The high frequency transmit jitter shall be within that specified (see Section 11.7.1, "High Frequency Transmit Jitter Measurement"for suggested methods)
3. The specified transmit eye mask shall not be not violated (see Section 11.7.4.6, "Eye Mask Measurement Setup" for a suggested method) after adjusting the horizontal time positions for the measured time with a confidence level of 3 sigma (see Annex B.3, “Eye Mask Adjustment for Sampling Oscilloscopes" for a suggested method).

11.4.1.4 Level II SR Receiver Inter-operability

The following step shall be made to identify which receivers are to be considered compliant:

1. The DUT shall be measured to have a BER[1] better than 10^{-12} for a stressed signal (see Section 11.7.4.2, "Jitter Tolerance with no Relative Wander Lab Setup" for a suggested method) with a confidence level of three sigma (see Annex B.2, “Confidence Level of Errors Measurement" for a suggested method), given:
   – The defined sinusoidal jitter mask for total and relative wander as per Section 11.4.2.2.8, "Level II SR Receiver Input Jitter Tolerance" with a high frequency total/relative wander and a maximum total/relative wander as defined in the CEI IA.
   – The specified amount of High Probability Jitter and Gaussian jitter per Section 11.4.2.2.8, "Level II SR Receiver Input Jitter Tolerance".
   – An additive crosstalk signal of amplitude such that the resulting statistical eye, given the channel, jitter and crosstalk, is as close as feasible in amplitude when compared to the defined minimum amplitude for channel compliance.

11.4.2 Level II SR Electrical Characteristics

The electrical interface is based on high speed, low voltage logic with nominal differential impedance of 100 Ω. Connections are point-to-point balanced differential pair and signalling is unidirectional.

11.4.2.1 Level II SR Transmitter Characteristics

The key transmitter characteristics are summarized in Table 11-2 and Table 11-3 while the following sections fully detail all the requirements.

Table 11-2. Level II SR Transmitter Output Electrical Specifications

Characteristic	Symbol	Condition	Min	Typ	Max	Units
Baud Rate (5 Gbaud)	T_Baud	Section 11.4.2.1.2	5.00 -0.01%	5.00	5.00 +0.01%	Gbaud
Baud Rate (6.25 Gbaud)			6.25 -0.01%	6.25	6.25 +0.01%	Gbaud
Absolute Output Voltage	V_O	Section 11.4.2.1.3	-0.40		2.30	Volts
Output Differential voltage (into floating load Rload=100 Ω)	T_Vdiff	Section 11.4.2.1.3	400		750	mVppd
Differential Resistance	T_Rd	Section 11.4.2.1.6	80	100	120	W

NOTES:
1. Load Type 0 with min T_Vdiff, AC-Coupling or floating load
2. For Load Types 1 through 3: R_Zvtt ≤ 30 Ω; Vtt is defined for each load type as follows: Load Type 1 R_Vtt = 1.2 V +5%/-8%; Load Type 2 R_Vtt = 1.0 V +5%/-8%; Load Type 3 R_Vtt = 0.8 V +5%/-8%.
3. DC Coupling compliance is optional (Type 1 through 3). Only Transmitters that support DC coupling are required to meet this parameter. It is acceptable for a transmitter to restrict the range of T_Vdiff in order to comply with the specified T_Vcm range. For a transmitter which supports multiple T_Vdiff levels, it is acceptable for a transmitter to claim DC Coupling Compliance if it meets the T_Vcm ranges for at least one of its T_Vdiff setting as long as those setting(s) that are compliant are indicated.

[1]if the defined measurement BER is different to system required BER, adjustments to applied stressed eye TJ are necessary

Table 11-2. Level II SR Transmitter Output Electrical Specifications

Characteristic	Symbol	Condition	Min	Typ	Max	Units
Recommended output rise and fall times (20% to 80%)	T_tr, T_tf	Section 11.4.2.1.4	30			ps
Skew between signals comprising a differential pair	T_SKEW$_{diff}$	Section 11.4.2.1.5			15	ps
Differential Output Return Loss (100 MHz to 0.5*T_Baud)	T_SDD22	Section 11.4.2.1.6			-8	dB
Differential Output Return Loss (0.5*T_Baud to T_Baud)						
Common Mode Return Loss (100 MHz to 0.75 *T_Baud)	T_SCC22	Section 11.4.2.1.6			-6	dB
Transmitter Common Mode Noise	T_Ncm				5% of T_Vdiff	mVppd
Output Common Mode Voltage Editor notes: This row is deleted and replaced with the following row.	T_Vcm	Load Type 0[1,2,3,4] Section 9.5.3	0.0		1.8	V
		Load Type 1[1,3,4,6] Section 9.5.3	735		1135	mV
		Load Type 2[1,3,4] Section 9.5.3	550		1060	mV
		Load Type 3[1,3,4,5] Section 9.5.3	490		850	mV
Output Common Mode Voltage	T_Vcm	Load Type 0[1] Section 9.5.3	100		1700	mV
		Load Type 1[2,3] Section 9.5.3	630		1100	mV

NOTES:
1. Load Type 0 with min T_Vdiff, AC-Coupling or floating load
2. For Load Types 1 through 3: R_Zvtt ≤ 30 Ω; Vtt is defined for each load type as follows: Load Type 1 R_Vtt = 1.2 V +5%/-8%; Load Type 2 R_Vtt = 1.0 V +5%/-8%; Load Type 3 R_Vtt = 0.8 V +5%/-8%.
3. DC Coupling compliance is optional (Type 1 through 3). Only Transmitters that support DC coupling are required to meet this parameter. It is acceptable for a transmitter to restrict the range of T_Vdiff in order to comply with the specified T_Vcm range. For a transmitter which supports multiple T_Vdiff levels, it is acceptable for a transmitter to claim DC Coupling Compliance if it meets the T_Vcm ranges for at least one of its T_Vdiff setting as long as those setting(s) that are compliant are indicated.

Table 11-3. Level II SR Transmitter Output Jitter Specifications

Characteristic	Symbol	Conditions	Min	Typ	Max	Units
Uncorrelated High Probability Jitter	T_UHPJ	Section 11.4.2.1.8			0.15	UIpp
Duty Cycle Distortion	T_DCD	Section 11.4.2.1.8			0.05	UIpp
Total Jitter	T_TJ	Section 11.4.2.1.8			0.30	UIpp
Eye Mask	T_X1	Section 11.4.2.1.8			0.15	UI
Eye Mask	T_X2	Section 11.4.2.1.8			0.40	UI
Eye Mask	T_Y1	Section 11.4.2.1.8	200			mV
Eye Mask	T_Y2	Section 11.4.2.1.8			375	mV

11.4.2.1.1 Level II SR Transmitter Test Load

All transmitter characteristics should be implemented and measured to a differential impedance of 100 Ω ± 1% at DC with a return loss of better than 20 dB from baud rate divided by 1667 to 1.5 times the baud rate, unless otherwise noted.

11.4.2.1.2 Level II SR Transmitter Baud Rate

The baud rates are 5 Gbaud and 6.25 Gbaud with a tolerance of ±100 ppm.

11.4.2.1.3 Level II SR Transmitter Amplitude and Swing

Transmitter differential output amplitude shall be between 400 and 750 mVppd, inclusive, either with or without any transmit emphasis. Absolute transmitter output voltage shall be between -0.1 V and 1.9 V, inclusive, with respect to local ground. See Figure 9-1 for an illustration of absolute transmitter output voltage limits and definition of differential peak-to-peak amplitude.

11.4.2.1.4 Level II SR Transmitter Rise and Fall Times

The recommended minimum differential rise and fall times are 30 ps as measured between the 20% and 80% of the maximum measured levels; the maximum differential rise and fall times are defined by the Tx eye diagram Figure 9-2 and Table 11-5. Shorter rise and fall times may result in excessive high frequency components and increase EMI and cross talk.

11.4.2.1.5 Level II SR Transmitter Differential Pair Skew

The timing skew at the output of a Level II SR transmitter between the two signals that comprise a differential pair shall not exceed 15 ps at 5.0 Gbaud and 6.25 Gbaud.

11.4.2.1.6 Level II SR Transmitter Output Resistance and Return Loss

Refer to Section 9.5.11, "Differential Resistance and Return Loss, Transmitter and Receiver" for the reference model for return loss. See Table 11-4 for 5 Gbaud and 6.25 Gbaud short run transmitter parameters. Definitions for these parameters are in Figure 9-12.

Table 11-4. Level II SR Transmitter Return Loss Parameters

Parameter	Value	Units
A0	-8	dB
f0	100	MHz
f1	T_Baud/2	Hz
f2	T_Baud	Hz
Slope	16.6	dB/dec

11.4.2.1.7 Level II SR Transmitter Lane-to-Lane Skew

The electrical level of lane-to-lane skew caused by the transmitter circuitry and associated routing must be less than 1000 ps for links of 4 lanes or less. Links with greater than 4 lanes must have lane-to-lane skew of less than 2 UI + 1000 ps. The transmitter lane-to-lane skew is only for the serdes Tx and does not include any effects of the channel.

11.4.2.1.8 Level II SR Transmitter Template and Jitter

As per Section 11.4.1.3, "Level II SR Transmitter Inter-operability" the transmitter shall satisfy both the near-end and far-end eye template and jitter requirements as given in Figure 9-2, Table 11-5, Figure 9-5, and Table 11-9 either with or without any transmit emphasis.

The maximum near-end duty cycle distortion (T_DCD) shall be less than 0.05 UIpp.

It should be noted that it is assumed the Uncorrelated High Probability Jitter component of the transmitter jitter is not Inter-symbol Interference (ISI). This is only assumed from a receiver point of view and does not in any way put any restrictions on the real transmitter HPJ.

Table 11-5. Level II SR Near-End (Tx) Template Intervals

Characteristics	Symbol	Near-End Value	Units
Eye Mask	T_X1	0.15	UI
Eye Mask	T_X2	0.40	UI
Eye Mask	T_Y1	200	mV
Eye Mask	T_Y2	375	mV
Eye Mask	T_Y3	125	mV
Uncorrelated Bounded High Probability Jitter	T_UBHPJ	0.15	UIpp
Duty Cycle Distortion	T_DCD	0.05	UIpp
Total Jitter	T_TJ	0.30	UIpp

11.4.2.2 Level II SR Receiver Characteristics

The key receiver characteristics are summarized in Table 11-6 and Table 11-7 while the following sections fully detail all the requirements.

Table 11-6. Level II SR Receiver Electrical Input Specifications

Characteristic	Symbol	Conditions	Min	Typ	Max	Units
Rx Baud Rate (5 Gbaud)	R_Baud	Section 11.4.2.2.1	5.00 -0.01%	5.00	5.00 +0.01%	Gbaud
Rx Baud Rate (6.25 Gbaud)			6.25 -0.01%	6.25	6.25 +0.01%	Gbaud
Absolute Input Voltage	R_Vin	Section 11.4.2.2.4				
NOTES: 1. DC Coupling compliance is optional. For Vcm definition, see Figure 9-1. 2. Receiver is required to implement at least one of specified nominal R_Vtt values, and typically implements only one of these values. Receiver is only required to meet R_Vrcm parameter values that correspond to R_Vtt values supported. 3. Input common mode voltage for AC-coupled or floating load input with min. T_Vdiff. 4. For floating load, input resistance must be ≥ 1 kΩ.						

Table 11-6. Level II SR Receiver Electrical Input Specifications

Characteristic	Symbol	Conditions	Min	Typ	Max	Units
Input Differential voltage	R_Vdiff	Section 11.4.2.2.3	125		1200	mVppd
Differential Resistance	R_Rdin	Section 11.4.2.2.7	80	100	120	Ω
Bias Voltage Source Impedance (load types 1 to 3)[1]	R_Zvtt				30	Ω
Differential Input Return Loss (100 MHz to 0.5*R_Baud)	R_SDD11	Section 11.4.2.2.7			-8	dB
Differential Input Return Loss (0.5*R_Baud to R_Baud))						
Common mode Input Return Loss (100 MHz to 0.5*R_Baud)	R_SCC11	Section 11.4.2.2.7			-6	dB
Termination Voltage[1,2]	R_Vtt	R_Vtt floating[4]	Not Specified			V
		R_Vtt = 1.2V Nominal	1.2 - 8%		1.2 + 5%	V
		R_Vtt = 1.0V Nominal	1.0 - 8%		1.0 + 5%	V
		R_Vtt = 0.8V Nominal	0.8 - 8%		0.8 + 5%	V
Input Common Mode Voltage[1,2] Editor notes: This row is deleted and replaced with the following row.	R_Vrcm	R_Vtt floating[3,4]	-0.05		1.85	V
		R_Vtt = 1.2V Nominal	720		R_Vtt - 10	mV
		R_Vtt = 1.0V Nominal	535		R_Vtt + 125	mV
		R_Vtt = 0.8V Nominal	475		R_Vtt + 105	mV
Input Common Mode Voltage	R_Vfcm	Load Type 0[2]	0		1800	mV
		Load Type 1[1,3]	595		R_Vtt - 60	mV
Wander divider (in Figure 9-8 & Figure 9-9)	n			10		

NOTES:
1. DC Coupling compliance is optional. For Vcm definition, see Figure 9-1.
2. Receiver is required to implement at least one of specified nominal R_Vtt values, and typically implements only one of these values. Receiver is only required to meet R_Vrcm parameter values that correspond to R_Vtt values supported.
3. Input common mode voltage for AC-coupled or floating load input with min. T_Vdiff.
4. For floating load, input resistance must be ≥ 1 kΩ.

Table 11-7. Level II SR Receiver Input Jitter Tolerance Specifications

Characteristics	Symbol	Conditions	Min	Typ	Max	Units
Bounded High Probability Jitter	R_BHPJ	Section 11.4.2.2.8			0.45	UIpp
Sinusoidal Jitter, maximum	R_SJ-max	Section 11.4.2.2.8			5	UIpp
Sinusoidal Jitter, High Frequency	R_SJ-hf	Section 11.4.2.2.8			0.05	UIpp
Total Jitter (Does not include Sinusoidal Jitter)	R_TJ	Section 11.4.2.2.8			0.60	UIpp
Eye Mask	R_X1	Section 11.4.2.2.8			0.30	UI

Table 11-7. Level II SR Receiver Input Jitter Tolerance Specifications

Characteristics	Symbol	Conditions	Min	Typ	Max	Units
Eye Mask	R_Y1	Section 11.4.2.2.8			62.5	mV
Eye Mask	R_Y2	Section 11.4.2.2.8			375	mV

11.4.2.2.1 Level II SR Receiver Input Baud Rate

All devices shall work at 5 Gbaud, 6.25 Gbaud or both baud rates with the baud rate tolerance as per Section 9.5.12, "Baud Rate Tolerance".

11.4.2.2.2 Level II SR Receiver Reference Input Signals

Reference input signals to the receiver have the characteristics determined by compliant transmitter. The reference input signal must satisfy the transmitter near-end template and jitter given in Figure 9-2 and Table 11-5, as well as the far-end eye template and jitter given in Figure 9-5 and Table 11-9, with the differential load impedance of 100 Ω ± 1% at DC with a return loss of better than 20 dB from baud rate divided by 1667 to 1.5 times the baud rate. Note that the input signal might not meet either of these templates when the actual receiver replaces this load.

11.4.2.2.3 Level II SR Receiver Input Signal Amplitude

The receiver shall accept differential input signal amplitudes produced by compliant transmitters connected without attenuation to the receiver. This may be larger than the 1200 mVppd maximum of the transmitter due to output/input impedances and reflections.

The minimum input amplitude is defined by the far-end transmitter template, the actual receiver input impedance, and the loss of the actual PCB. Note that the far-end transmitter template is defined using a well controlled load impedance, however, the real receiver is not, which can leave the receiver input signal smaller than the minimum 125 mVppd.

11.4.2.2.4 Level II SR Receiver Absolute Input Voltage

The absolute voltage levels with respect to the receiver ground at the input of the receiver are dependent on the transmitter implementation, the inter-ground difference, whether the receiver is AC or DC coupled, and (in the case of DC coupling load types 1 to 3) the nominal R_Vtt supported by the receiver. The voltage levels at the input of a DC coupled receiver shall be consistent with the R_Vrcm and R_Vdiff values defined in Table 11-6.

The voltage levels at the input of an AC coupled receiver (if AC coupling is done within the receiver) or at the Tx side of the external AC coupling cap (if AC coupling is done externally) shall be between -0.15 V and 1.95 V, inclusive, with respect to local ground.

11.4.2.2.5 Level II SR Receiver Input Common Mode Impedance

The input common mode impedance (R_Zvtt) at the input of the receiver is dependent on whether the receiver is AC or DC coupled. The value of R_Zvtt as measured at the input of an AC coupled receiver is undefined. The value of R_Zvtt as measured at the input of a DC coupled receiver is defined as per Table 11-6.

If AC coupling is used it is to be considered part of the receiver for the purposes of this specification unless explicitly stated otherwise. It should be noted that various methods for AC coupling are allowed (for example, internal to the chip or done externally). See Section 9.5.13, "Termination and DC Blocking" for more information.

11.4.2.2.6 Level II SR Receiver Input Lane-to-Lane Skew

Lane-to-lane skew at the input to the receiver shall not exceed 70 UI peak. See Section 9.5.9, "Receiver Input Lane-to-Lane Skew".

11.4.2.2.7 Level II SR Receiver Input Resistance and Return Loss

Refer to Section 9.5.11, "Differential Resistance and Return Loss, Transmitter and Receiver" for the reference model for return loss. See Table 11-8 for 5 Gbaud and 6.25 Gbaud short run receiver parameters. Definitions for these parameters are in Figure 9-12.

Table 11-8. Level II SR Input Return Loss Parameters

Parameter	Value	Units
A0	-8	dB
f0	100	MHz
f1	R_Baud/2	Hz
f2	R_Baud	Hz
Slope	16.6	dB/dec

11.4.2.2.8 Level II SR Receiver Input Jitter Tolerance

As per Section 11.4.1.4, "Level II SR Receiver Inter-operability", the receiver shall tolerate at least the far-end eye template and jitter requirements as given in Figure 9-5 and Table 11-9 with an additional SJ with any frequency and amplitude defined by the mask of Figure 9-9 where the minimum and maximum total wander amplitude are 0.05 UIpp and 5 UIpp respectively. This additional SJ component is intended to ensure margin for wander, hence is over and above any high frequency jitter from Table 11-9.

Table 11-9. Level II SR Far-End (Rx) Template Intervals

Characteristics	Symbol	Far-End Value	Units
Eye Mask	R_X1	0.30	UI
Eye Mask	R_Y1	62.5	mV
Eye Mask	R_Y2	375	mV
Uncorrelated Bounded High Probability Jitter	R_UBHPJ	0.15	UIpp
Correlated Bounded High Probability Jitter	R_CBHPJ	0.30	UIpp
Total Jitter (Does not include Sinusoidal Jitter)	R_TJ	0.60	UIpp

11.4.2.3 Level II SR Link and Jitter Budgets

The primary intended application is as a point-to-point interface of up to approximately 20 cm and up to one connector between integrated circuits using controlled impedance traces on low-cost printed circuit boards (PCBs). Informative loss and jitter budgets are presented in Table 11-10 to demonstrate the feasibility of legacy FR4 epoxy PCBs. The jitter budget is given in Table 11-11. The performance of an actual transceiver interconnect is highly dependent on the implementation.

Table 11-10. Level II SR Informative Loss, Skew and Jitter Budget

Description	Loss (dB)	Differential Skew (ps)	Bounded High Probability (UIpp)	TJ (UIpp)
Driver	0	15	0.15	0.30
Interconnect (with Connector)	6.6	25	0.15	0.15
Other	3.5		0.15	0.15
Total	10.1	40	0.45	0.60

Table 11-11. Level II SR High Frequency Jitter Budget

CEI-6G-SR	Uncorrelated Jitter		Correlated Jitter		Total Jitter				Amplitude	
	Unbounded Gaussian	High Probability	Bounded Gaussian	Bounded High Probability	Gaussian	Sinusoidal	Bounded High Probability	Total		
Abbreviation	UUGJ	UHPJ	CBGJ	CBHPJ	GJ	SJ	HPJ	TJ	k	
Units	UIpp	UIpp	UIpp	UIpp	UIpp	UIpp	UIpp	UIpp		mVppd
Transmitter	0.150	0.150		-0.200[1]	0.150		-0.050	0.100		400.0
Channel				0.500						
Receiver Input	0.150	0.150	0.000	0.300	0.150		0.450	0.600	0.25	125
Clock + Sampler	0.150	0.100		0.100						-50.0
Budget	0.212	0.250	0.000	0.400	0.212	0.050	0.650	0.912	0.13	75.0

NOTES:
1. Due to transmitter emphasis, it reduces the ISI as seen at the receiver. Thus this number is negative.

11.4.3 Level II SR StatEye.org Template

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% example template for setting up a standard, i.e. equalizer
% jitter and return loss

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

param.version = [param.version '_v1.0'];

% these are internal variables and should not be changed

param.scanResolution = 0.01;
param.binsize                        = 0.0005;
param.points                         = 2^13;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% set the transmitter and baud rate. The tx filter has two
% parameters defined for the corner frequency of the poles

param.bps                            = 6.25e9;
param.bitResolution                  = 1/(4*param.bps);
param.txFilter                       = 'singlepole';
param.txFilterParam                  = [0.75];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% set the return loss up. The return loss can be turned off
% using the appropriate option

param.returnLoss                     = 'on';
param.cpad                           = 1.0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% set the transmitter emphasis up. Some example setting are
% included which can be uncommented

% single tap emphasis
param.txpre                          = [];
param.signal                         = 1.0;
param.txpost                         = [-0.1];
param.vstart                         = [-0.3 -0.3];
param.vend                           = [+0.0 +0.0];
param.vstep                          = [0.1 0.05 0.025];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% set the de-emphasis of 4-point transmit pulse
% the de-emphasis run if param.txpre = [] and param.txpost = []

param.txdeemphasis = [1 1 1 1];             % de-emphasis is off

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% set the data coding changing the transmit pulse spectrum
% the coding run if param.txpre = [] and param.txpost = []

param.datacoding = 1;        % the coding is off

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% set PAM amplitude and rate

param.PAM = 2;              % PAM is switched off

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% the rxsample point does not need to be changed as it is
% automatically adjusted by the optimization scripts.
% The number of DFE taps should be set, however, the initial
% conditions are irrelevant.

param.rxsample                      = -0.1;

% no DFE
param.dfe                           = [];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% sampling jitter in HPJpp and GJrms is defined here

param.txdj                          = 0.15;
param.txrj                          = 0.15/(2*7.94);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% the following options are not yet implemented and should
% not be changed

param.user                          = [0.0];
param.useuser                       = 'no';
param.usesymbol                     = '';
param.xtAmp                         = 1.0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

param.TransmitAmplitude = 0.400; % mVppdif
param.MinEye            = 0.125; % mVppdif

param.Q                 = 2*7.94;
param.maxDJ             = 0.30;
param.maxTJ             = 0.60;
```

11.5 Level II Long Run Interface General Requirements

11.5.1 Long Run Jitter and Inter-operability Methodology

The LP-Serial 5 Gbaud and 6.25 Gbaud short run interfaces use Method D, described in CEI clause 2.4. This section defines the inter-operability methodology specifically for interfaces where transmit emphasis may be used and the receiver eye requires DFE equalization (from channel inter-operability point of view) to be open to within the BER of interest.

11.5.1.1 Level II LR Channel Compliance

The following steps shall be made to identify which channels are to be considered compliant:

1. The forward channel and significant crosstalk channels shall be measured using a network analyzer for the specified baud rate (see Section 11.7.4.5, "Network Analysis Measurement" for a suggested method).
2. A single pre or post tap transmitter with ≤ 6 dB of emphasis, with infinite precision accuracy.
3. A Tx edge rate filter: a two-pole 40 dB/dec low pass at 75% of baud rate, this is to emulate both Rx and Tx -3 dB bandwidths at $^3/_4$ baud rate.
4. A transmit amplitude of 800 mVppd.
5. Additional Uncorrelated Bounded High Probability Jitter of 0.15 UIpp (emulating part of the Tx jitter).
6. Additional Uncorrelated Unbounded Gaussian Jitter of 0.15 UIpp (emulating part of the Tx jitter).
7. The reference transmitter shall use the worst case transmitter return loss at the baud frequency. In order to construct the worse case transmitter return loss, the reference transmitter should be considered to be a parallel R and C, where R is the defined maximum allowed DC resistance of the interface and C is increased until the defined maximum Return Loss at the baud frequency is reached. The transmitter return loss is specified in Section 11.5.2.1.6, "Level II LR Transmitter Output Resistance and Return Loss".
8. An ideal receiver filter of the form in Section 9.6.6, "Decision Feedback Equalizer". The reference receiver uses a 5 tap DFE, with infinite precision accuracy and having the following restriction on the coefficient values:

   Let W[N] be sum of DFE tap coefficient weights from taps N through M where

N = 1 is previous decision (i.e. first tap)
M = oldest decision (i.e. last tap)
R_Y2 = T_Y2 = 400 mV
Y = min(R_X1, (R_Y2 - R_Y1) / R_Y2) = 0.30
Z = $^{2}/_{3}$ = 0.66667

Then $W[N] \le Y * Z^{(N - 1)}$

For the channel compliance model the number of DFE taps (M) = 5. This gives the following maximum coefficient weights for the taps:

W[1] ≤ 0.2625 (sum of taps 1 to 5)
W[2] ≤ 0.1750 (sum of taps 2 to 5)
W[3] ≤ 0.1167 (sum of taps 3 to 5)
W[4] ≤ 0.0778 (sum of taps 4 and 5)
W[5] ≤ 0.0519 (tap 5)

Notes:
- These coefficient weights are absolute assuming a T_Vdiff of 1 Vppd
- For a real receiver the restrictions on tap coefficients would apply for the actual number of DFE taps implemented (M)

9. The reference receiver shall use the worst case receiver return loss at the baud frequency. In order to construct the worse case receiver return loss, the reference receiver should be considered to be a parallel R and C, where R is the defined maximum allowed DC resistance of the interface and C is increased until the defined maximum Return Loss at the baud frequency is reached. The receiver return loss is specified in Section 11.5.2.2.7, "Level II LR Receiver Input Resistance and Return Loss".

Table 11-12. Level II LR Receiver Equalization Output Eye Mask

Parameter	Symbol	Max	Units
Eye mask	R_X1	0.30	UI
Eye mask	R_Y1	50	mV
Bounded High Probability Jitter	R_BHPJ	0.325	UI

10. Any parameters that have degrees of freedom (e.g. filter coefficients or sampling point) shall be optimized against the amplitude, at the zero phase offset, as generated by the Statistical Eye Output, e.g. by sweeping all degrees of freedom and selecting the parameters giving the maximum amplitude. A receiver return loss, as defined by the reference receiver, shall be used.

11. The opening of the eye shall be calculated using Statistical Eye Analysis methods, as per Section 9.7.5, "Statistical Eye Methodology", and confirmed to be within the requirements of the equalized eye mask as specified in Table 11-12 at the required BER, 10^{-15}.

11.5.1.2 Level II LR Transmitter Inter-operability

The following step shall be made to identify which transmitters are to be considered compliant:

1. It shall be verified that the measured eye is equal or better than the calculated eye for the given measurement probability Q (see Annex B.3, “Eye Mask Adjustment for Sampling Oscilloscopes" for a suggested method of calculating Q given a measurement population), given:
   - A “compliance” channel as per Section 11.5.1.1, "Level II LR Channel Compliance" that required at least half the maximum transmit emphasis with no receiver filtering to give an open eye.
   - Using this channel the transmitter shall be then optimally adjusted and the resulting near-end eye measured (see Section 11.7.4.6, "Eye Mask Measurement Setup" for a suggested method).
   - Using this channel the statistical eye shall then be calculated, as per Section 9.7.5, "Statistical Eye Methodology", using the maximum defined transmit jitter and the actual transmitter's amplitude and emphasis.

If the transmit jitter or transmit eye mask is additionally defined then the following steps shall also be made to identify which transmitters are to be considered compliant:

1. The high frequency transmit jitter shall be within that specified (see Section 11.7.1, "High Frequency Transmit Jitter Measurement" for suggested methods).

The specified transmit eye mask shall not be violated (see Section 11.7.4.6, "Eye Mask Measurement Setup" for a suggested method) after adjusting the horizontal time positions for the measured time with a confidence level of 3 sigma (see Annex B.3, “Eye Mask Adjustment for Sampling Oscilloscopes" for a suggested method).

11.5.1.3 Level II LR Receiver Inter-operability

The following step shall be made to identify which receivers are to be considered compliant:

1. The DUT shall be measured to have a BER[1] better than specified for a stressed signal (see Section 11.7.4.3, "Jitter Tolerance with Defined ISI and no Relative Wander" for a suggested method) with a confidence level of three sigma (see Annex B.2, “Confidence Level of Errors Measurement" for a suggested method), given:
   - The defined sinusoidal jitter mask for relative wander as per Section 9.4.6, "Relative Wander Mask" with a high frequency relative wander and a maximum relative wander as defined in Section 11.5.2.2.8, "Level II LR Receiver Jitter Tolerance".
   - The specified amount of High Probability Jitter and Gaussian jitter as defined in Section 11.5.2.2.8, "Level II LR Receiver Jitter Tolerance".
   - A compliance channel or filter as identified by Section 11.5.1.1, "Level II LR Channel Compliance".
   - An additive crosstalk signal of amplitude such that the resulting statistical eye, given the channel, jitter, and crosstalk, is as close as feasible in amplitude when compared to the defined minimum amplitude for channel compliance.

11.5.2 Level II LR Interface Electrical Characteristics

The electrical interface is based on high speed, low voltage logic with nominal differential impedance of 100 Ω. Connections are point-to-point balanced differential pair and signalling is unidirectional.

11.5.2.1 Level II LR Transmitter Characteristics

The key transmitter characteristics are summarized in Table 11-13 and Table 11-14 while the following sections fully detail all the requirements.

[1]if the defined measurement BER is different to system required BER, adjustments to applied stressed eye TJ are necessary

Table 11-13. Level II LR Transmitter Output Electrical Specifications

Characteristics	Symbols	Conditions	Min	Typ	Max	Units
Tx Baud Rate (5 Gbaud)	T_Baud	Section 11.5.2.1.2	5.00 -0.01%	5.00	5.00 +0.01%	Gbaud
Tx Baud Rate (6.25 Gbaud)			6.25 -0.01%	6.25	6.25 +0.01%	Gbaud
Absolute Output Voltage	V_O	Section 11.5.2.1.3	-0.40		2.30	Volts
Output Differential voltage (into floating load Rload = 100 Ω)	T_Vdiff	Section 11.5.2.1.3[1]	800		1200	mVppd
Differential Resistance	T_Rd	Section 11.5.2.1.6	80	100	120	Ω
Recommended output rise and fall times (20% to 80%)	T_tr, T_tf	Section 11.5.2.1.4	30			ps
Skew between signals comprising a differential pair	T_SKEW_{diff}	Section 11.5.2.1.5			15	ps
Differential Output Return Loss (100 MHz to 0.5*T_Baud)	T_SDD22	Section 11.5.2.1.6			-8	dB
Differential Output Return Loss (0.5*T_Baud to T_Baud)						
Common Mode Return Loss (100 MHz to 0.75 *T_Baud)	T_S11	Section 11.5.2.1.6			-6	dB
Transmitter Common Mode Noise	T_Ncm				5% of T_Vdiff	mVppd
Output Common Mode Voltage	T_Vcm	Load Type 0[2] Section 9.5.3	100		1700	mV
		Load Type 1[3,4] Section 9.5.3	630		1100	mV

NOTES:
1. The Transmitter must be capable of producing a minimum T_Vdiff greater than or equal to 800 mVppd. In applications where the channel is better than the worst case allowed, a Transmitter device may be provisioned to produce T_Vdiff less than this minimum value, but greater than or equal to 400 mVppd, and is still compliant with this specification.
2. Load Type 0 with min T_Vdiff, AC-Coupling or floating load.
3. For Load Type 1: R_Zvtt ≤ 30 Ω; T_Vtt & R_Vtt = 1.2V +5%/-8%.
4. DC Coupling compliance is optional (Load Type 1). Only Transmitters that support DC coupling are required to meet this parameter.

Table 11-14. Level II LR Transmitter Output Jitter Specifications

Characteristics	Symbol	Conditions	Min	Typ	Max	Units
Uncorrelated High Probability Jitter	T_UHPJ	Section 11.5.2.2.8			0.15	UIpp
Duty Cycle Distortion	T_DCD	Section 11.5.2.2.8			0.05	UIpp
Total Jitter	T_TJ	Section 11.5.2.2.8			0.30	UIpp
Eye Mask	T_X1	Section 11.5.2.2.8			0.15	UI
Eye Mask	T_X2	Section 11.5.2.2.8			0.40	UI
Eye Mask	T_Y1	Section 11.5.2.2.8	200			mV
Eye Mask	T_Y2	Section 11.5.2.2.8			600	mV

11.5.2.1.1 Level II LR Transmitter Test Load

All transmitter characteristics should be implemented and measured to a differential impedance of 100 Ω ± 1% at DC with a return loss of better than 20 dB from baud rate divided by 1667 to 1.5 times the baud rate, unless otherwise noted.

11.5.2.1.2 Level II LR Transmitter Baud Rate

The baud rates are 5 Gbaud and 6.25 Gbaud with a tolerance of ±100 ppm.

11.5.2.1.3 Level II LR Transmitter Amplitude and Swing

Transmitter differential output amplitude shall be able to drive between 800 and 1200 mVppd, inclusive, either with or without any transmit emphasis. DC referenced logic levels are not defined since the receiver must have high common mode impedance at DC. However, absolute transmitter output voltage shall be between -0.1 V and 1.9 V, inclusive, with respect to local ground. See Figure 9-1 for an illustration of absolute transmitter output voltage limits and definition of differential peak-to-peak amplitude.

11.5.2.1.4 Level II LR Transmitter Rise and Fall Times

The recommended minimum differential rise and fall time is 30 ps as measured between the 20% and 80% of the maximum measured levels; the maximum differential rise and fall times are defined by the Tx eye diagram (Figure 9-2 and Table 11-16). Shorter rise and falls may result in excessive high frequency components and increase EMI and cross talk.

11.5.2.1.5 Level II LR Transmitter Differential Pair Skew

The timing skew at the output of a Level II LR transmitter between the two signals that comprise a differential pair shall not exceed 15 ps at 5.0 Gbaud and 6.25 Gbaud.

11.5.2.1.6 Level II LR Transmitter Output Resistance and Return Loss

Refer to Section 9.5.11, "Differential Resistance and Return Loss, Transmitter and Receiver" for the reference model for return loss. See Table 11-15 for 5 Gbaud and 6.25 Gbaud long run transmitter parameters. Definitions for these parameters are in Figure 9-12.

Table 11-15. Level II LR Transmitter Return Loss Parameters

Parameter	Value	Units
A0	-8	dB
f0	100	MHz
f1	T_Baud/2	Hz
f2	R_Baud	Hz
Slope	16.6	dB/dec

11.5.2.1.7 Level II LR Transmitter Lane-to-Lane Skew

The electrical level of lane-to-lane skew caused by the transmitter circuitry and associated routing must be less than 1000 ps for links of 4 lanes or less. Links with greater than 4 lanes must have lane-to-lane skew of less than 2 UI + 1000 ps. The transmitter lane-to-lane skew is only for the serdes Tx and does not include any effects of the channel.

11.5.2.1.8 Level II LR Transmitter Short Circuit Current

The max DC current into or out of the transmitter pins when either shorted to each other or to ground shall be ±100 mA when the device is fully powered up. From a hot swap point of view, the ±100 mA limit is only valid after 10 µs.

11.5.2.1.9 Level II LR Transmitter Template and Jitter

The transmitter shall satisfy both the near-end eye template and jitter requirements as given in Figure 9-2 and Table 11-16 either with or without any transmit emphasis.

The maximum near-end duty cycle distortion (T_DCD) shall be less than 0.05 UIpp.

It should be noted that it is assumed the Uncorrelated High Probability Jitter component of the transmitter jitter is not Inter-symbol Interference (ISI). This is only assumed from a receiver point of view so that a receiver can't equalize it and does not in any way put any restrictions on the real transmitter HPJ.

Table 11-16. Level II LR Near-End Template Intervals

Characteristics	Symbol	Near-End Value	Units	Comments
Eye Mask	T_X1	0.15	UI	
Eye Mask	T_X2	0.40	UI	
Eye Mask	T_Y1	200	mV	For connection to short run Rx
		400		For connection to long run Rx
Eye Mask	T_Y2	375	mV	For connection to short run Rx
		600		For connection to long run Rx
Uncorrelated Bounded High Probability Jitter	T_UBHPJ	0.15	UIpp	
Duty Cycle Distortion	T_DCD	0.05	UIpp	
Total Jitter	T_TJ	0.30	UIpp	

11.5.2.2 Level II LR Receiver Characteristics

The key receiver characteristics are summarized in Table 11-17 while the following sections fully detail all the requirements.

Table 11-17. Level II LR Receiver Electrical Input Specifications

Characteristic	Symbol	Condition	Min	Typ	Max	Units
Rx Baud Rate (5 Gbaud)	R_Baud	Section 11.5.2.1.2	5.00 -0.01%	5.00	5.00 +0.01%	Gbaud
Rx Baud Rate (6.25 Gbaud)			6.25 -0.01%	6.25	6.25 +0.01%	Gbaud
Absolute Input Voltage	R_Vin	Section 11.5.2.2.4				
Input Differential voltage	R_Vdiff	Section 11.5.2.2.3			1200	mVppd
Differential Resistance	R_Rdin	Section 11.5.2.2.7	80	100	120	Ω
Bias Voltage Source Impedance (load type 1)[1]	R_Zvtt				30	Ω
Differential Input Return Loss (100 MHz to 0.5*R_Baud)	R_SDD11	Section 11.5.2.2.7			-8	dB
Differential Input Return Loss (0.5*R_Baud to R_Baud))						
Common mode Input Return Loss (100 MHz to 0.5*R_Baud)	R_SCC11	Section 11.5.2.2.7			-6	dB
Input Common Mode Voltage	R_Vfcm	Load Type 0[2]	0		1800	mV
		Load Type 1[1,3]	595		R_Vtt - 60	mV
Wander divider (in Figure 9-8 & Figure 9-9)	n			10		

NOTES:
1. DC Coupling compliance is optional (Load Type 1). Only receivers that support DC coupling are required to meet this parameter.
2. Load Type 0 with min T_Vdiff, AC-Coupling or floating load. For floating load, input resistance must be ≥ 1 kΩ
3. For Load Type 1: T_Vtt & R_Vtt = 1.2 V +5%/-8%.

11.5.2.2.1 Level II LR Receiver Input Baud Rate

All devices shall work at 5 Gbaud, 6.25 Gbaud or both baud rates with the baud rate tolerance as per Section 9.5.12.

11.5.2.2.2 Level II LR Receiver Reference Input Signals

Reference input signals to the receiver have the characteristics determined by the compliant transmitter. The reference input signal must satisfy the transmitter near-end template and jitter given in Figure 9-2 and Table 11-16, as well as the far-end eye jitter given in Table 11-20, with the differential load impedance of 100 Ω ± 1% at DC with a return loss of better than 20 dB from baud rate divided by 1667 to 1.5 times the baud rate. Note that the input signal might not meet either of these requirements when the actual receiver replaces this load.

11.5.2.2.3 Level II LR Receiver Input Signal Amplitude

The receiver shall accept differential input signal amplitudes produced by compliant transmitters connected without attenuation to the receiver. This may be larger than the 1200 mVppd maximum of the transmitter due to output/input impedances and reflections.

The minimum input amplitude is defined by the far-end transmitter template, the actual receiver input impedance, and the loss of the actual PCB. Note that the far-end transmitter template is defined using a well controlled load impedance, however the real receiver is not, which can leave the receiver input signal smaller than expected.

11.5.2.2.4 Level II LR Receiver Absolute Input Voltage

The absolute voltage levels with respect to the receiver ground at the input of the receiver are dependent on the transmitter implementation and the inter-ground difference.

The voltage levels at the input of an AC coupled receiver (if the effective AC coupling is done within the receiver) or at the Tx side of the external AC coupling cap (if AC coupling is done externally) shall be between -0.15 V and 1.95 V, inclusive, with respect to local ground.

11.5.2.2.5 Level II LR Receiver Input Common Mode Impedance

The input common mode impedance (R_Zvtt) at the input of the receiver is dependent on whether the receiver is AC or DC coupled. The value of R_Zvtt as measured at the input of an AC coupled receiver is undefined. The value of R_Zvtt as measured at the input of a DC coupled receiver is defined as per Table 11-17.

If AC coupling is used it is to be considered part of the receiver for the purposes of this specification unless explicitly stated otherwise. It should be noted that various methods for AC coupling are allowed (for example, internal to the chip or done externally). See Section 9.5.13, "Termination and DC Blocking" for more information.

11.5.2.2.6 Level II LR Receiver Input Lane-to-Lane Skew

Lane-to-lane skew at the input to the receiver shall not exceed 70 UI peak. See Section 9.5.9, "Receiver Input Lane-to-Lane Skew".

11.5.2.2.7 Level II LR Receiver Input Resistance and Return Loss

Refer to Section 9.5.11, "Differential Resistance and Return Loss, Transmitter and Receiver" for the reference model for return loss. See Table 11-18 for 5 Gbaud and 6.25 Gbaud short run receiver parameters. Definitions for these parameters are in Figure 9-12.

Table 11-18. Level II LR Input Return Loss Parameters

Parameter	Value	Units
A0	-8	dB
f0	100	MHz
f1	R_Baud/2	Hz
f2	R_Baud	Hz
Slope	16.6	dB/dec

11.5.2.2.8 Level II LR Receiver Jitter Tolerance

As per Section 11.5.1.3, "Level II LR Receiver Inter-operability", the receiver shall tolerate at least the far-end jitter requirements as given in Table 11-12 in combination with any compliant channel, as per Section 11.5.1.1, "Level II LR Channel Compliance", with an additional SJ with any frequency and amplitude defined by the mask of Figure 9-9 where the minimum and maximum total wander amplitude are 0.05 UIpp and 5 UIpp respectively. This additional SJ component is intended to ensure margin for wander, hence is over and above any high frequency jitter from Table 11-12.

11.5.3 Level II LR Link and Jitter Budgets

The primarily intended application is as a point-to-point interface of up to approximately 100 cm and up to two connector between integrated circuits using controlled impedance traces on low-cost printed circuit boards (PCBs). Informative loss and jitter budgets are presented in Table 11-19 to demonstrate the feasibility of legacy FR4 epoxy PCBs. The jitter budget is given in Table 11-20. The performance of an actual transceiver interconnect is highly dependent on the implementation.

Table 11-19. Level II LR Informative Loss, Skew and Jitter Budget

Description	Loss (dB)	Differential Skew (ps)	Bounded High Probability (UIpp)	TJ (UIpp)
Transmitter	0	15	0.15	0.30
Interconnect (with Connector)	15.9	25	0.35	0.513
Other	4.5		0.10	0.262
Total	20.4	40	0.60	0.875

Table 11-20. Level II LR High Frequency Jitter Budget

CEI-6G-LR	Uncorrelated Jitter		Correlated Jitter		Total Jitter				Amplitude	
	Unbounded Gaussian	High Probability	Bounded Gaussian	Bounded High Probability	Gaussian	Sinusoidal	Bounded High Probability	Total		
Abbreviation	UUGJ	UHPJ	CBGJ	CBHPJ	GJ	SJ	HPJ	TJ	k	
Units	UIpp	UIpp	UIpp	UIpp	UIpp	UIpp	UIpp	UIpp		mVppd
Transmitter	0.150	0.150			0.150		0.150	0.300		800.0
Channel			0.230	0.525						
Receiver Input	0.150	0.150	0.230	0.525	0.275		0.675	0.950	0.00	0.0[2]
Equalizer				-0.350[1]						
Post Equalization	0.150	0.150	0.230	0.175	0.275		0.325	0.60	0.20	100.0
DFE Penalties				0.100					-0.08	-45.0
Clock + Sampler	0.150	0.100		0.100						-45.0
Budget	0.212	0.250	0.230	0.375	0.313	0.050	0.625	0.988	0.06	10.0

NOTES:
1. Due to receiver equalization, it reduces the ISI as seen inside the receiver. Thus this number is negative.
2. It is assumed that the eye is closed at the receiver, hence receiver equalization is required as indicated below.

11.5.4 Level II LR StatEye.org Template

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% example template for setting up a standard, i.e. equalizer
% jitter and return loss

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

param.version = [param.version '_v1.0'];

% these are internal variables and should not be changed

param.scanResolution                 = 0.01;
param.binsize                        = 0.0005;
param.points                         = 2^13;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% set the transmitter and baud rate. The tx filter has two
% parameters defined for the corner frequency of the poles

param.bps                            = 6.25e9;
param.bitResolution                  = 1/(4*param.bps);
param.txFilter                       = 'twopole';
param.txFilterParam                  = [0.75 0.75];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% set the return loss up. The return loss can be turned off
% using the appropriate option

param.returnLoss                     = 'on';
param.cpad                           = 1.00;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% set the transmitter emphasis up. Some example setting are
% included which can be uncommented

% single tap emphasis
param.txpre                          = [-0.1];
param.signal                         = 1.0;
param.txpost                         = [];
param.vstart                         = [-0.3 -0.3];
param.vend                           = [+0.0 +0.0];
param.vstep                          = [0.1 0.05 0.025];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% set the de-emphasis of 4-point transmit pulse
% the de-emphasis run if param.txpre = [] and param.txpost = []

param.txdeemphasis = [1 1 1 1];             % de-emphasis is off

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% set the data coding changing the transmit pulse spectrum
% the coding run if param.txpre = [] and param.txpost = []

param.datacoding = 1;        % the coding is off

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% set PAM amplitude and rate

param.PAM = 2;              % PAM is switched off

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% the rxsample point does not need to be changed as it is
% automatically adjusted by the optimization scripts.
% The number of DFE taps should be set, however, the initial
% conditions are irrelevant.

param.rxsample                     = -0.1;

param.dfe                          = [0.3 0.1 0.1 0.1 0.1];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% sampling jitter in HPJpp and GJrms is defined here

param.txdj                         = 0.15;
param.txrj                         = 0.15/(2*7.94);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% the following options are not yet implemented and should
% not be changed

param.user                         = [0.0];
param.useuser                      = ’no’;
param.usesymbol                    = ’’;
param.xtAmp                        = 1.0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

param.TransmitAmplitude = 0.800; % mVppdif
param.MinEye            = 0.100; % mVppdif

param.Q                 = 2*7.94;
param.maxDJ             = 0.325;
param.maxTJ            = 0.60;
```

11.6 Level II Medium Run Interface General Requirements

11.6.1 Medium Run Jitter and Inter-operability Methodology

The LP-Serial 5 Gbaud and 6.25 Gbaud short run interfaces use Method C, described in CEI clause 2.4. This section defines the inter-operability methodology specifically for interfaces where transmit emphasis may be used and the receiver eye requires linear equalization (from channel inter-operability point of view) to be open to within the BER of interest.

11.6.1.1 Level II Medium Run Channel Compliance

The following steps shall be made to identify which channels are to be considered compliant:

1. The forward channel and significant crosstalk channels shall be measured using a network analyzer for the specified baud rate (see CEI Section 11.7.4.5, "Network Analysis Measurement" for a suggested method).
2. A single pre or post tap transmitter with <= 6 dB of emphasis, with infinite precision accuracy.
3. A Tx edge rate filter: simple 40 dB/dec low pass at 75% of baud rate, this is to emulate both Rx and Tx -3 dB bandwidths at $^3/_4$ baud rate.
4. A transmit amplitude of 800 mVppd.
5. Additional Uncorrelated Bounded High Probability Jitter of 0.15 UIpp (emulating part of the Tx jitter).
6. Additional Uncorrelated Unbounded Gaussian Jitter of 0.15 UIpp (emulating part of the Tx jitter).
7. The reference transmitter shall use the worst case transmitter return loss at the baud frequency. In order to construct the worse case transmitter return loss, the reference transmitter should be considered to be a parallel R and C, where R is the defined maximum allowed DC resistance of the interface and C is increased until the defined maximum Return Loss at the baud frequency is reached. The transmitter return loss is specified in Section 11.6.2.1.6, "Level II MR Transmitter Output Resistance and Return Loss".
8. An ideal receiver filter of the form in Section 9.6.8, "Time Continuous Zero/Pole". The reference receiver uses a continuous-time equalizer with 1 zero and 1 pole in the region of baudrate/100 to baudrate. Additional parasitic zeros and poles must be considered part of the receiver vendor’s device and

be dealt with as they are for the reference receiver. Pole and Zero values have infinite precision accuracy. Maximum required gain/attenuation shall be less than or equal to 4 dB.

9. The reference receiver shall use the worst case receiver return loss at the baud frequency. In order to construct the worse case receiver return loss, the reference receiver should be considered to be a parallel R and C, where R is the defined maximum allowed DC resistance of the interface and C is increased until the defined maximum Return Loss at the baud frequency is reached. The receiver return loss is specified in Section 11.6.2.2.7, "Level II MR Receiver Input Resistance and Return Loss".

Table 11-21. Level II LR Receiver Equalization Output Eye Mask

Parameter	Symbol	Max	Units
Eye mask	R_X1	0.30	UI
Eye mask	R_Y1	50	mV
Bounded High Probability Jitter	R_BHPJ	0.325	UI

10. Any parameters that have degrees of freedom (e.g. filter coefficients or sampling point) shall be optimized against the amplitude, at the zero phase offset, as generated by the Statistical Eye Output, e.g. by sweeping all degrees of freedom and selecting the parameters giving the maximum amplitude. A receiver return loss, as defined by the reference receiver, shall be used.
11. The opening of the eye shall be calculated using Statistical Eye Analysis methods, as per Section 9.7.5, "Statistical Eye Methodology", and confirmed to be within the requirements of the equalized eye mask as specified in Table 11-12 at the required BER, 10^{-12}.

11.6.1.2 Level II MR Transmitter Inter-operability

The following step shall be made to identify which transmitters are to be considered compliant:

1. It shall be verified that the measured eye is equal or better than the calculated eye for the given measurement probability Q (see Annex B.3, “Eye Mask Adjustment for Sampling Oscilloscopes" for a suggested method of calculating Q given a measurement population), given:
   – A “compliance” channel as per Section 11.6.1.1, "Level II Medium Run Channel Compliance" that required at least half the maximum transmit emphasis with no receiver filtering to give an open eye.
   – Using this channel the transmitter shall be then optimally adjusted and the resulting near-end eye measured (see Section 11.7.4.6, "Eye Mask Measurement Setup" for a suggested method).
   – Using this channel the statistical eye shall then be calculated, as per Section 9.7.5, "Statistical Eye Methodology", using the maximum defined transmit jitter and the actual transmitter's amplitude and emphasis.

If the transmit jitter or transmit eye mask is additionally defined then the following steps shall also be made to identify which transmitters are to be considered compliant:

1. The high frequency transmit jitter shall be within that specified (see Section 11.7.1, "High Frequency Transmit Jitter Measurement" for suggested methods).

The specified transmit eye mask shall not be violated (see Section 11.7.4.6, "Eye Mask Measurement Setup" for a suggested method) after adjusting the horizontal time positions for the measured time with a confidence level of 3 sigma (see Annex B.3, “Eye Mask Adjustment for Sampling Oscilloscopes" for a suggested method).

11.6.1.3 Medium Receiver Inter-operability

The following step shall be made to identify which receivers are to be considered compliant:

1. The DUT shall be measured to have a BER[1] better than specified for a stressed signal (see Section 11.7.4.3, "Jitter Tolerance with Defined ISI and no Relative Wander" for a suggested method) with a confidence level of three sigma (see Annex B.2, “Confidence Level of Errors Measurement" for a suggested method), given:
   – The defined sinusoidal jitter mask for relative wander as per Section 9.4.5, "Total Wander Mask" with a high frequency relative wander and a maximum relative wander as defined in Section 11.5.2.2.8, "Level II LR Receiver Jitter Tolerance".
   – The specified amount of High Probability Jitter and Gaussian jitter as defined in Section 11.5.2.2.8, "Level II LR Receiver Jitter Tolerance".
   – A compliance channel or filter as identified by Section 11.5.1.1, "Level II LR Channel Compliance".
   – An additive crosstalk signal of amplitude such that the resulting statistical eye, given the channel, jitter, and crosstalk, is as close as feasible in amplitude when compared to the defined minimum amplitude for channel compliance.

11.6.2 Level II MR Interface Electrical Characteristics

The electrical interface is based on high speed low voltage logic with nominal differential impedance of 100 Ω. Connections are point-to-point balanced differential pair and signalling is unidirectional.

11.6.2.1 Level II MR Transmitter Characteristics

The key transmitter characteristics are summarized in Table 11-22 and Table 11-23 while the following sections fully detail all the requirements.

[1]if the defined measurement BER is different to system required BER, adjustments to applied stressed eye TJ are necessary

Table 11-22. Level II MR Transmitter Output Electrical Specifications

Characteristics	Symbols	Conditions	Min	Typ	Max	Units
Tx Baud Rate (5 Gbaud)	T_Baud	Section 11.6.2.1.2	5.00 -0.01%	5.00	5.00 +0.01%	Gbaud
Tx Baud Rate (6.25 Gbaud)			6.25 -0.01%	6.25	6.25 +0.01%	Gbaud
Absolute Output Voltage	V_O	Section 11.6.2.1.3	-0.40		2.30	Volts
Output Differential voltage (into floating load Rload = 100 Ω)	T_Vdiff	Section 11.6.2.1.3[1]	800		1200	mVppd
Differential Resistance	T_Rd	Section 11.6.2.1.6	80	100	120	Ω
Recommended output rise and fall times (20% to 80%)	T_tr, T_tf	Section 11.6.2.1.4	30			ps
Skew between signals comprising a differential pair	T_SKEW_{diff}	Section 11.6.2.1.5			15	ps
Differential Output Return Loss (100 MHz to 0.5*T_Baud)	T_SDD22	Section 11.6.2.1.6			-8	dB
Differential Output Return Loss (0.5*T_Baud to T_Baud)						
Common Mode Return Loss (100 MHz to 0.75 *T_Baud)	T_S11	Section 11.6.2.1.6			-6	dB
Transmitter Common Mode Noise	T_Ncm				5% of T_Vdiff	mVppd
Output Common Mode Voltage	T_Vcm	Load Type 0[2] Section 9.5.3	100		1700	mV
		Load Type 1[3,4] Section 9.5.3	630		1100	mV

NOTES:
1. The Transmitter must be capable of producing a minimum T_Vdiff greater than or equal to 800 mVppd. In applications where the channel is better than the worst case allowed, a Transmitter device may be provisioned to produce T_Vdiff less than this minimum value, but greater than or equal to 400 mVppd, and is still compliant with this specification.
2. Load Type 0 with min T_Vdiff, AC-Coupling or floating load.
3. For Load Type 1: R_Zvtt ≤ 30 Ω; T_Vtt & R_Vtt = 1.2 V +5%/-8%
4. DC Coupling compliance is optional (Load Type 1). Only Transmitters that support DC coupling are required to meet this parameter.

Table 11-23. Level II MR Transmitter Output Jitter Specifications

Characteristics	Symbol	Conditions	Min	Typ	Max	Units
Uncorrelated High Probability Jitter	T_UHPJ	Section 11.6.2.2.8			0.15	UIpp
Duty Cycle Distortion	T_DCD	Section 11.6.2.2.8			0.05	UIpp
Total Jitter	T_TJ	Section 11.6.2.2.8			0.30	UIpp
Eye Mask	T_X1	Section 11.6.2.2.8			0.15	UI
Eye Mask	T_X2	Section 11.6.2.2.8			0.40	UI
Eye Mask	T_Y1	Section 11.6.2.2.8	200			mV
Eye Mask	T_Y2	Section 11.6.2.2.8			600	mV

11.6.2.1.1 Level II MR Transmitter Test Load

All transmitter characteristics should be implemented and measured to a differential impedance of 100 Ω ± 1% at DC with a return loss of better than 20 dB from baud rate divided by 1667 to 1.5 times the baud rate, unless otherwise noted.

11.6.2.1.2 Level II MR Transmitter Baud Rate

The baud rates are 5 Gbaud and 6.25 Gbaud with a tolerance of ±100 ppm.

11.6.2.1.3 Level II MR Transmitter Amplitude and Swing

Transmitter differential output amplitude shall be able to drive between 800 and 1200 mVppd, inclusive, either with or without any transmit emphasis. DC referenced logic levels are not defined since the receiver must have high common mode impedance at DC. However, absolute transmitter output voltage shall be between -0.1 V and 1.9 V, inclusive, with respect to local ground. See Figure 9-1 for an illustration of absolute transmitter output voltage limits and definition of differential peak-to-peak amplitude.

11.6.2.1.4 Level II MR Transmitter Rise and Fall Times

The recommended minimum differential rise and fall time is 30 ps as measured between the 20% and 80% of the maximum measured levels; the maximum differential rise and fall times are defined by the Tx eye diagram (Figure 9-2 and Table 11-16). Shorter rise and falls may result in excessive high frequency components and increase EMI and cross talk.

11.6.2.1.5 Level II MR Transmitter Differential Pair Skew

The timing skew at the output of a Level II MR transmitter between the two signals that comprise a differential pair shall not exceed 15 ps at 5.0 Gbaud and 6.25 Gbaud.

11.6.2.1.6 Level II MR Transmitter Output Resistance and Return Loss

Refer to Section 9.5.11, "Differential Resistance and Return Loss, Transmitter and Receiver" for the reference model for return loss. See Table 11-15 for 5 Gbaud and 6.25 Gbaud long run transmitter parameters. Definitions for these parameters are in Figure 9-12.

Table 11-24. Level II MR Transmitter Return Loss Parameters

Parameter	Value	Units
A0	-8	dB
f0	100	MHz
f1	T_Baud/2	Hz
f2	R_Baud	Hz
Slope	16.6	dB/dec

11.6.2.1.7 Level II MR Transmitter Lane-to-Lane Skew

The electrical level of lane-to-lane skew caused by the transmitter circuitry and associated routing must be less than 1000 ps for links of 4 lanes or less. Links with greater than 4 lanes must have lane-to-lane skew of less than 2 UI + 1000 ps. The transmitter lane-to-lane skew is only for the serdes Tx and does not include any effects of the channel.

11.6.2.1.8 Level II MR Transmitter Short Circuit Current

The max DC current into or out of the transmitter pins when either shorted to each other or to ground shall be ±100 mA when the device is fully powered up. From a hot swap point of view, the ±100 mA limit is only valid after 10 μs.

11.6.2.1.9 Level II MR Transmitter Template and Jitter

The transmitter shall satisfy both the near-end eye template and jitter requirements as given in Figure 9-2, Figure 9-3, and Table 11-16 either with or without any transmit emphasis.

The maximum near-end duty cycle distortion (T_DCD) shall be less than 0.05 UIpp.

It should be noted that it is assumed the Uncorrelated High Probability Jitter component of the transmitter jitter is not Inter-symbol Interference (ISI). This is only assumed from a receiver point of view so that a receiver can’t equalize it and does not in any way put any restrictions on the real transmitter HPJ.

Table 11-25. Level II MR Near-End Template Intervals

Characteristics	Symbol	Near-End Value	Units	Comments
Eye Mask	T_X1	0.15	UI	
Eye Mask	T_X2	0.40	UI	
Eye Mask	T_Y1	200	mV	For connection to short run Rx
		400		For connection to long run Rx
Eye Mask	T_Y2	375	mV	For connection to short run Rx
		600		For connection to long run Rx
Uncorrelated Bounded High Probability Jitter	T_UBHPJ	0.15	UIpp	
Duty Cycle Distortion	T_DCD	0.05	UIpp	
Total Jitter	T_TJ	0.30	UIpp	

11.6.2.2 Level II MR Receiver Characteristics

The key receiver characteristics are summarized in Table 11-26 while the following sections fully detail all the requirements.

Table 11-26. Level II MR Receiver Electrical Input Specifications

Characteristic	Symbol	Condition	Min	Typ	Max	Units
Rx Baud Rate (5 Gbaud)	R_Baud	Section 11.6.2.2.1	5.00 -0.01%	5.00	5.00 +0.01%	Gbaud
Rx Baud Rate (6.25 Gbaud)			6.25 -0.01%	6.25	6.25 +0.01%	Gbaud
Absolute Input Voltage	R_Vin	Section 11.6.2.2.4				
Input Differential voltage	R_Vdiff	Section 11.6.2.2.3			1200	mVppd
Differential Resistance	R_Rdin	Section 11.5.2.2.7	80	100	120	Ω
Bias Voltage Source Impedance (load type 1)[1]	R_Zvtt				30	Ω
Differential Input Return Loss (100 MHz to 0.5*R_Baud)	R_SDD11	Section 11.6.2.2.7			-8	dB
Differential Input Return Loss (0.5*R_Baud to R_Baud))						
Common mode Input Return Loss (100 MHz to 0.5*R_Baud)	R_SCC11	Section 11.6.2.2.7			-6	dB
Input Common Mode Voltage	R_Vfcm	Load Type 0[2]	0		1800	mV
		Load Type 1[1,3]	595		R_Vtt - 60	mV
Wander divider (in Figure 9-8 & Figure 9-9)	n			10		

NOTES:
1. DC Coupling compliance is optional (Load Type 1). Only receivers that support DC coupling are required to meet this parameter.
2. Load Type 0 with min T_Vdiff, AC-Coupling or floating load. For floating load, input resistance must be ≥ 1 kΩ
3. For Load Type 1: T_Vtt & R_Vtt = 1.2 V +5%/-8%.

11.6.2.2.1 Level II MR Receiver Input Baud Rate

All devices shall work at 5 Gbaud, 6.25 Gbaud or both baud rates with the baud rate tolerance as per Section 9.5.12, "Baud Rate Tolerance".

11.6.2.2.2 Level II MR Receiver Reference Input Signals

Reference input signals to the receiver have the characteristics determined by the compliant transmitter. The reference input signal must satisfy the transmitter near-end template and jitter given in Figure 9-2, Figure 9-3, and Table 11-16, as well as the far-end eye jitter given in Table 11-20, with the differential load impedance of 100 Ω ± 1% at DC with a return loss of better than 20 dB from baud rate divided by 1667 to 1.5 times the baud rate. Note that the input signal might not meet either of these requirements when the actual receiver replaces this load.

11.6.2.2.3 Level II MR Receiver Input Signal Amplitude

The receiver shall accept differential input signal amplitudes produced by compliant transmitters connected without attenuation to the receiver. This may be larger than the 1200 mVppd maximum of the transmitter due to output/input impedances and reflections.

The minimum input amplitude is defined by the far-end transmitter template, the actual receiver input impedance, and the loss of the actual PCB. Note that the far-end transmitter template is defined using a well controlled load impedance, however the real receiver is not, which can leave the receiver input signal smaller than expected.

11.6.2.2.4 Level II MR Receiver Absolute Input Voltage

The absolute voltage levels with respect to the receiver ground at the input of the receiver are dependent on the transmitter implementation and the inter-ground difference.

The voltage levels at the input of an AC coupled receiver (if the effective AC coupling is done within the receiver) or at the Tx side of the external AC coupling cap (if AC coupling is done externally) shall be between -0.15 V and 1.95 V, inclusive, with respect to local ground.

11.6.2.2.5 Level II MR Receiver Input Common Mode Impedance

The input common mode impedance (R_Zvtt) at the input of the receiver is dependent on whether the receiver is AC or DC coupled. The value of R_Zvtt as measured at the input of an AC coupled receiver is undefined. The value of R_Zvtt as measured at the input of a DC coupled receiver is defined as per Table 11-17.

If AC coupling is used it is to be considered part of the receiver for the purposes of this specification unless explicitly stated otherwise. It should be noted that various methods for AC coupling are allowed (for example, internal to the chip or done externally). See Section 9.5.13, "Termination and DC Blocking" for more information.

11.6.2.2.6 Level II MR Receiver Input Lane-to-Lane Skew

Lane-to-lane skew at the input to the receiver shall not exceed 70 UI peak. See Section 9.5.9, "Receiver Input Lane-to-Lane Skew".

11.6.2.2.7 Level II MR Receiver Input Resistance and Return Loss

Refer to Section 9.5.11, "Differential Resistance and Return Loss, Transmitter and Receiver" for the reference model for return loss. See Table 11-27 for 5 Gbaud and 6.25 Gbaud short run receiver parameters. Definitions for these parameters are in Figure 9-12.

Table 11-27. Level II MR Input Return Loss Parameters

Parameter	Value	Units
A0	-8	dB
f0	100	MHz
f1	R_Baud/2	Hz
f2	R_Baud	Hz
Slope	16.6	dB/dec

11.6.2.2.8 Level II MR Receiver Jitter Tolerance

As per Section 11.5.1.3, "Level II LR Receiver Inter-operability", the receiver shall tolerate at least the far-end jitter requirements as given in Table 11-12 in combination with any compliant channel, as per Section 11.5.1.1, "Level II LR Channel Compliance", with an additional SJ with any frequency and amplitude defined by the mask of Figure 9-8 where the minimum and maximum total wander amplitude are 0.05 UIpp and 5 UIpp respectively. This additional SJ component is intended to ensure margin for wander, hence is over and above any high frequency jitter from Table 11-12.

11.6.3 Level II MR Link and Jitter Budgets

The primarily intended application is as a point-to-point interface of up to approximately 60 cm and up to two connector between integrated circuits using controlled impedance traces on low-cost printed circuit boards (PCBs). Informative loss and jitter budgets are presented in Table 11-19 to demonstrate the feasibility of legacy FR-4 epoxy PCBs. The jitter budget is given in Table 11-20. The performance of an actual transceiver interconnect is highly dependent on the implementation.

Table 11-28. Level II MR Informative Loss, Skew and Jitter Budget

Description	Loss (dB)	Differential Skew (ps)	Bounded High Probability (UIpp)	TJ (UIpp)
Transmitter	0	15	0.15	0.30
Interconnect (with Connector)	15.9	25	0.35	0.513
Other	4.5		0.10	0.262
Total	20.4	40	0.60	0.875

Table 11-29. Level II MR High Frequency Jitter Budget

CEI-6G-LR	Uncorrelated Jitter		Correlated Jitter		Total Jitter				Amplitude	
	Unbounded Gaussian	High Probability	Bounded Gaussian	Bounded High Probability	Gaussian	Sinusoidal	Bounded High Probability	Total		
Abbreviation	UUGJ	UHPJ	CBGJ	CBHPJ	GJ	SJ	HPJ	TJ	k	
Units	UIpp	UIpp	UIpp	UIpp	UIpp	UIpp	UIpp	UIpp		mVppd
Transmitter	0.150	0.150			0.150		0.150	0.300		800.0
Channel			0.230	0.525						
Receiver Input	0.150	0.150	0.230	0.525	0.275		0.675	0.950	0.00	0.0[2]
Equalizer				-0.350[1]						
Post Equalization	0.150	0.150	0.230	0.175	0.275		0.325	0.60	0.20	100.0
DFE Penalties				0.100					-0.08	-45.0
Clock + Sampler	0.150	0.100		0.100						-45.0
Budget	0.212	0.250	0.230	0.375	0.313	0.050	0.625	0.988	0.06	10.0

NOTES:
1. Due to receiver equalization, it reduces the ISI as seen inside the receiver. Thus this number is negative.
2. It is assumed that the eye is closed at the receiver, hence receiver equalization is required as indicated below.

11.6.4 Level II MR StatEye.org Template

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% example template for setting up a standard, i.e. equalizer
% jitter and return loss

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

param.version = [param.version '_v1.0'];

% these are internal variables and should not be changed

param.scanResolution                 = 0.01;
param.binsize                        = 0.0005;
param.points                         = 2^13;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% set the transmitter and baud rate. The tx filter has two
% parameters defined for the corner frequency of the poles

param.bps                            = 6.25e9;
param.bitResolution                  = 1/(4*param.bps);
param.txFilter                       = 'twopole';
param.txFilterParam                  = [0.75 0.75];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% set the return loss up. The return loss can be turned off
% using the appropriate option

param.returnLoss                     = 'on';
param.cpad                           = 1.0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% set the transmitter emphasis up. Some example setting are
% included which can be uncommented

% single tap emphasis
param.txpre                          = [];
param.signal                         = 1.0;
param.txpost                         = [-0.1];
param.vstart                         = [-0.3 -0.3];
param.vend                           = [+0.0 +0.0];
param.vstep                          = [0.1 0.05 0.025];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% set the de-emphasis of 4-point transmit pulse
% the de-emphasis run if param.txpre = [] and param.txpost = []

param.txdeemphasis = [1 1 1 1];             % de-emphasis is off

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% set the data coding changing the transmit pulse spectrum
% the coding run if param.txpre = [] and param.txpost = []

param.datacoding = 1;        % the coding is off

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% set PAM amplitude and rate

param.PAM = 2;              % PAM is switched off

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% the rxsample point does not need to be changed as it is
% automatically adjusted by the optimization scripts.
% The number of DFE taps should be set, however, the initial
% conditions are irrelevant.

param.rxsample                        = -0.1;

% no DFE
param.dfe                             = [];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% The CTE shall be controlled.

param.cte = 1; % CTE setting “0” = off; “1” = on;
param.ctethresh = 3; % max gain;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% sampling jitter in HPJpp and GJrms is defined here

param.txdj                            = 0.15;
param.txrj                            = 0.15/(2*7.94);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% the following options are not yet implemented and should
% not be changed

param.user                            = [0.0];
param.useuser                         = ’no’;
param.usesymbol                       = ’’;
param.xtAmp                           = 1.0;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

param.TransmitAmplitude = 0.800; % mVppdif
param.MinEye            = 0.100; % mVppdif

param.Q                 = 2*7.94;
param.maxDJ             = 0.325;
param.maxTJ             = 0.60;
```

11.7 Level II Measurement and Test Requirements

All methodology described in this section is only relevant for verification of low level CDR functionality, and does not cover any required tests for protocol compliance, e.g. deskew. The methodology is based on the assumption that either an integrated BERT is present in the DUT or a loop or functionality for the attachment of external equipment.

11.7.1 High Frequency Transmit Jitter Measurement

The following section describes various methods for measuring high frequency jitter, which depending upon the baud rate can be applied for various levels of accuracy.

11.7.1.1 BERT Implementation

Referring to Figure 11-4, this section describes test methodology based on bathtub extraction, which relies on equipment being available for the given baud rate.

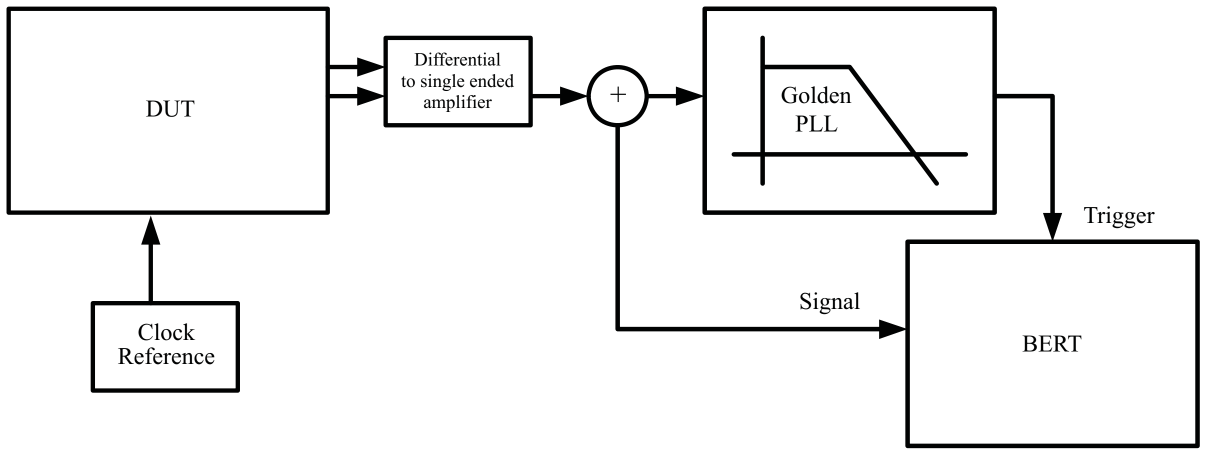


Figure 11-4. BERT with Golden PLL

- This same methodology can be used by equalized transmitters by initially turning the equalization off, or by performing the measurement at the end of a golden channel.

- The transmitter under test shall transmit the specified data pattern, while all other signals are active.
  - The other channels can transmit the same pattern if they have at least a 16 bit offset with the channel under test.
  - All links within a device under test to be active in both transmit and receive directions, and receive links are to use asynchronous clocks with respect to transmit links to maximum allowed ppm offset as specified in the protocol specifications.
- The data should be differentially analyzed using an external differential amp or differential input BERT and golden PLL.
  - Use of single ended signals will give an inaccurate measurement and should not be used.
  - The use of a balun will most likely degrade the signal integrity and is only recommended for 3 Gbaud signaling when the balun is linear with a return loss of better than -15 dB until three times the baud rate.
- Inherent bandwidth of clock reference inputs of the BERT should be verified, e.g. in the case of parBERTs. Additional bandwidth limitation of the BERT will lead to inaccurate results.
- The use of a golden PLL is required to eliminate inherent clock content (Wander) in transmitted data signals for long measurement periods.
  - The golden PLL should have at maximum a bandwidth of baud rate over 1667, with a maximum of 20 dB/dec rolloff, until at least baud rate over 16.67, with no peaking around the corner frequency.
- The output jitter for the DUT is not defined as the contributed jitter from the DUT but as the total output jitter including the contributions from the reference clock. To this end, the reference clock of the DUT should be verified to have a performance similar to the real application.
- A confidence level of three sigma should be guaranteed in the measurement of BER for the Bathtub as per Annex B.2, “Confidence Level of Errors Measurement".[1]
- The High Probability and Gaussian Jitter components should be extracted from the bathtub measurement using the methodology defined in Section 9.7.4.6, "BathTub Curves".
- If not defined the maximum Gaussian jitter is equal to the maximum total jitter minus the actual High Probability jitter.

[1]It is assumed due to the magnitude of jitter present at the transmitter that the left and right hand parts of the bathtub are independent to each other

11.7.1.2 Spectrum Analyzer and Oscilloscope Methodology

11.7.1.2.1 Band Limited[1] Unbounded Gaussian Noise

Referring to Figure 11-5, blandishment or high frequency Gaussian noise can be measured at the transmitter of the DUT accurately using a high frequency 101010 pattern and measuring the spectral power[2]. In Figure 11-5 the clock reference is such that its power noise represents the typical power noise of the reference in the system.

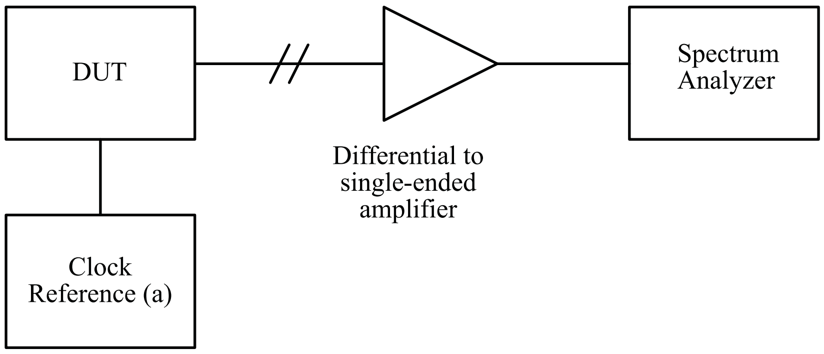


Figure 11-5. Spectral Measurement Setup

The spectral power is calculating by integrating over the frequency band of interest and converting into time jitter.

$$\tau_{rms} = \frac{1}{2\pi}\sqrt{2 \cdot \int_{f_1/100}^{100f_2} \left|\frac{1/f_1 \cdot j \cdot f}{(1 + j \cdot f/f_1)(1 + j \cdot f/f_2)}\right| \cdot 10^{\frac{P(f)}{10}}}$$

where

τ_{rms} is the time jitter

$P(f)$ is the measured spectral power for 1 Hz Bandwidth

It should be noted that the measured Gaussian noise for a driver can usually be considered equivalent to that derived from a full bathtub jitter distribution.

11.7.1.2.2 Band Limited 60 Second Total Jitter Measurements

In certain CEI-11G-SR applications total jitter measurements of 60 seconds are required. The Gaussian Jitter, as measured above, should be multiplied by a Q of 6.96[3]. If spurs are present in the spectrum then these must be converted to time jitter

[1]Normal CEI application will integrate from the defined ideal CDR bandwidth to infinity, while some CEI-11G-SR application will integrate over a specific band

[2]The spectral power should be measured using averaging

[3]Traditional measurements are performed for 60 seconds using a demodulator and performing a real time peak to peak measurement of the jitter. Given this, the number of bits transmitter across the link in 60 seconds is calculated and the associated three sigma confidence level, peak to peak multiplication factor, Q, for the random jitter.

separately using an inverse of the Bessel function as per Figure 11-6, which describes the power spectrum for a given phase modulated signal

where

$F(P_n)$ is the inverse spectral SSB power to time modulation (below)

$$\tau_{pkpk} = 2Q\tau_{rms} + \sum_{n} F(P_n)$$

P_n is the relative SSB power of a spur.

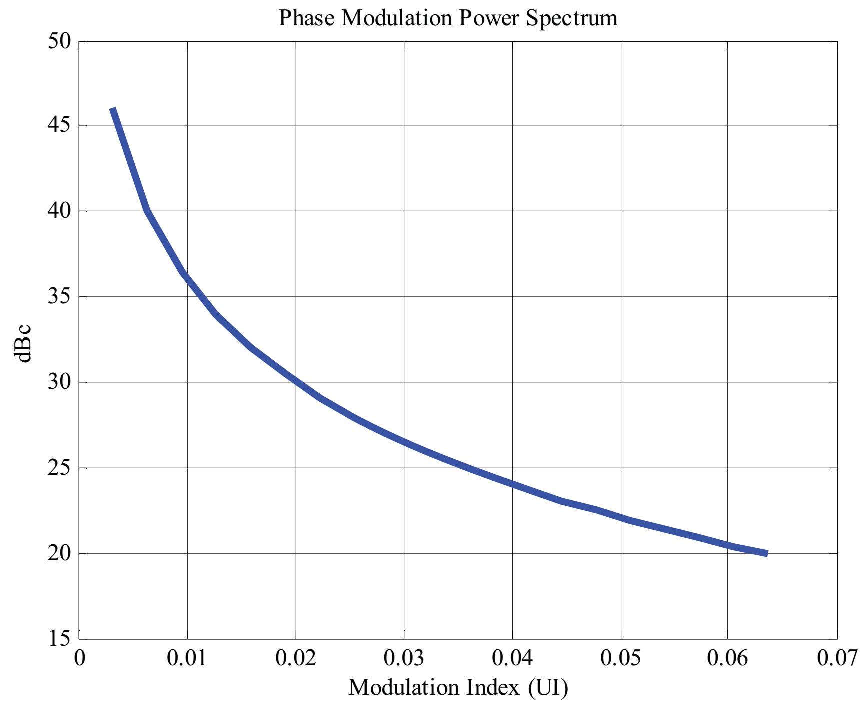


Figure 11-6. Single Side Band Relative Power Spectrum for Phase Modulated Signal

11.7.1.2.3 Uncorrelated High Probability Jitter

After measuring the Gaussian Jitter, as above, an oscilloscope measurement, as per Section 11.7.4.6, "Eye Mask Measurement Setup", of the peak to peak jitter should be performed using a 101010 pattern.

The Uncorrelated High Probability Jitter is then calculated by removing the accumulated Unbounded Gaussian jitter

$$\tau_{UBHJ} = \tau_{pkpk} - 2Q\tau_{rms}$$

using a Q calculated for a 3 sigma confidence level[1] as per Annex B.3, “Eye Mask Adjustment for Sampling Oscilloscopes".

11.7.1.2.4 Total High Probability Jitter

After measuring the Unbounded Gaussian Jitter, as above, an oscilloscope measurement, as per Section 11.7.4.6, "Eye Mask Measurement Setup", of the peak to peak jitter should be performed using the standard pattern e.g. PRBS31.

The Total High Probability Jitter is then calculated by removing the accumulated Gaussian jitter.

$$\tau_{HPJ} = \tau_{pkpk} - 2Q\tau_{rms}$$

using a Q calculated for a 3 sigma confidence level[2] as per Annex B.3, “Eye Mask Adjustment for Sampling Oscilloscopes".

11.7.2 Total Transmit Wander Measurement

This section describes the total transmit wander of a simple non-equalized transmitter as depicted in Figure 11-7 below.

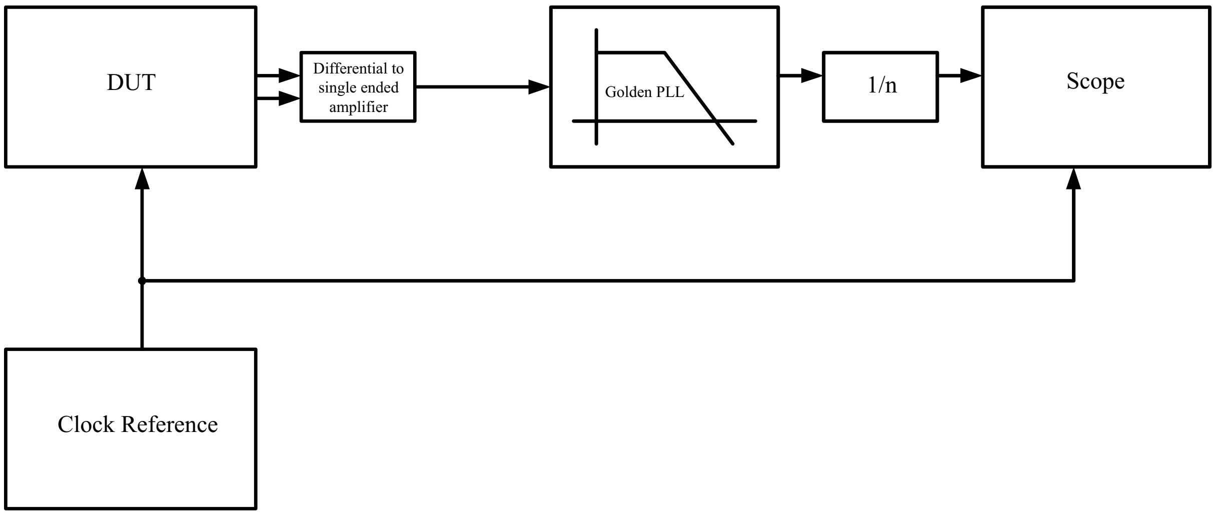


Figure 11-7. Transmit Wander Lab Setup

- The transmitter under test shall transmit the specified data pattern while all other signals are active.
  - The other channels can transmit the same pattern if they have at least a 16 bit offset with the channel under test.

[1]It is recommended that enough samples on the oscilloscope should be made such that Q>4
[2]It is recommended that enough samples on the oscilloscope should be made such that Q>4

  - All lanes to be active in both transmit and receive directions, and opposite ends of the link, i.e. transmit to receiver, are to use asynchronous clocks to maximum allowed ppm offset as specified in the protocol specifications.
- The transmitter can be tested single ended as high frequency jitter components are filtered by the golden PLL.
- Temperature and supply voltage should be cycled with a rate slower than baud rate over 166700 Hz during test to exercise any delay components in the DUT.
- The inherent clock wander in signal shall be extracted using golden PLL and divided by the 1/n block, such as to limit the measured wander to 1 UI at the divided frequency, and thus allowing it to be measured on an oscilloscope.
  - The golden PLL should have at a minimum bandwidth of baud rate over 1667, with a maximum of 20 dB/dec rolloff, until at least baud rate over 16.67, and is suggested to have no peaking around the corner frequency.
- The peak to peak total wander of the extracted clock should be measured using a scope triggered by the reference clock. The measured peak to peak wander should be verified to be bounded by repeating the measurement for ever increasing periods of time until the measurement is constant.

11.7.3 Relative Transmit Wander Measurement

This section describes specifically for SxI-5 interfaces, where limitations are defined in terms of relative wander between data lanes and clocks, whose relative wander can be measured as depicted below.

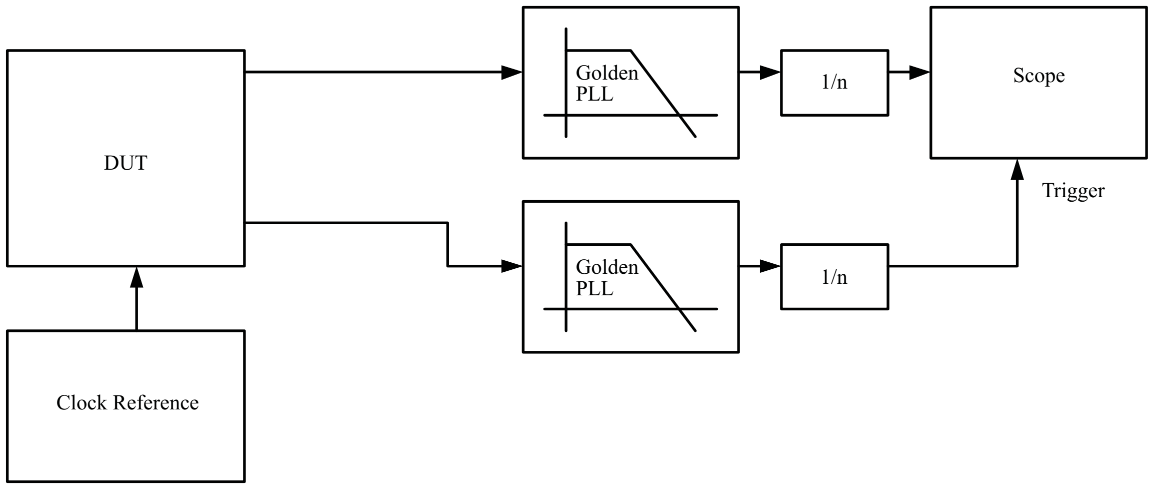


Figure 11-8. Relative Wander Lab Setup

- The transmitter under test shall transmit the specified data pattern while all other signals are active.
  - The other channels can transmit the same pattern if they have at least a 16 bit

offset with the channel under test.

- All lanes to be active in both transmit and receive directions, and opposite ends of the link, i.e. transmit to receiver, are to use asynchronous clocks to maximum allowed ppm. offset as specified in the protocol specifications.
- The transmitters can be tested single ended as high frequency jitter components are filtered by the golden PLL.
- Temperature and supply voltage should be cycled with a rate slower than baud rate over 166700 Hz during test to exercise any delay components in the DUT.
- The inherent clock wander in each signal shall be extracted using golden PLL and divided by the 1/n block, such as to limit the measured wander to 1 UI at the divided frequency, and thus allowing it to be measured on an oscilloscope.
- The golden PLL should have at a minimum bandwidth of baud rate over 1667, with a maximum of 20 dB/dec rolloff, until at least baud rate over 16.67, and is suggested to have no peaking around the corner frequency.

• The peak to peak relative wander between the extracted clocks should be measured using a scope triggered by one of the extracted clocks. The measured peak to peak wander should be verified to be bounded by repeating the measurement for ever increasing periods of time until the measurement is constant.

11.7.4 Jitter Tolerance

11.7.4.1 Jitter Tolerance with Relative Wander Lab Setup

The following section describes the required jitter tolerance methodology for devices where Relative Wander is applicable, e.g. SxI.5, and where no receive equalization is implemented.

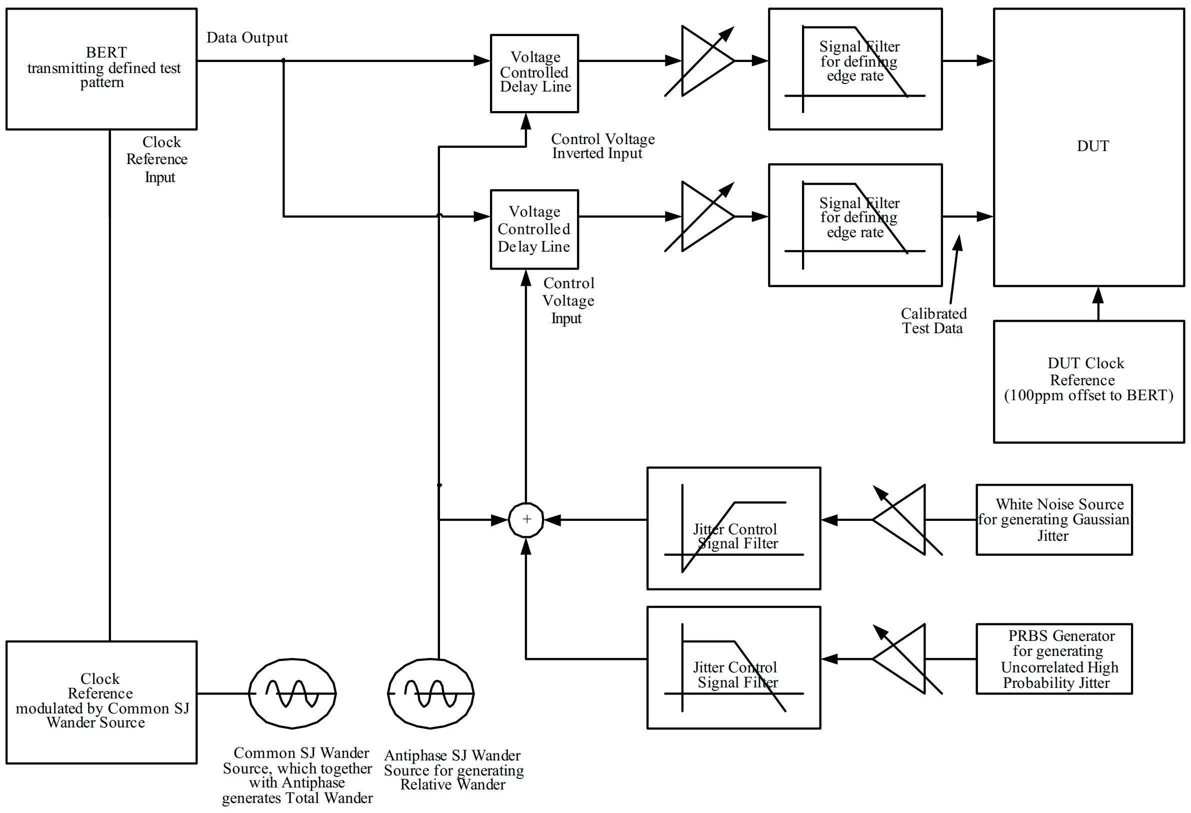

BERT
transmitting defined test pattern
Data Output
Voltage Controlled Delay Line
Signal Filter for defining edge rate
DUT
Clock Reference Input
Control Voltage Inverted Input
Voltage Controlled Delay Line
Signal Filter for defining edge rate
Control Voltage Input
Calibrated Test Data
DUT Clock Reference (100ppm offset to BERT)
+
Jitter Control Signal Filter
White Noise Source for generating Gaussian Jitter
Jitter Control Signal Filter
PRBS Generator for generating Uncorrelated High Probability Jitter
Clock Reference modulated by Common SJ Wander Source
Common SJ Wander Source, which together with Antiphase generates Total Wander
Antiphase SJ Wander Source for generating Relative Wander

Figure 11-9. Jitter Tolerance with Relative Wander Lab Setup

11.7.4.1.1 General

The transmitter under test shall transmit the specified data pattern while all other signals are active.

– The other channels can transmit the same pattern if they have at least a 16 bit offset with the channel under test.

– All lanes to be active in both transmit and receive directions, and opposite ends of the link, i.e. transmit to receiver, are to use asynchronous clocks to maximum allowed ppm offset as specified in the protocol specifications.

• The DUT shall be tested using an internal BERT or loop to have the defined BER performance.

• The confidence level of the BER measurement should be at least three sigma as per Annex B.2, “Confidence Level of Errors Measurement".

11.7.4.1.2 Synchronization

• All lanes are to be active in both transmit and receive directions.

• All reference clocks should have the maximum offset frequency, with respect to each other, as defined in the CEI IA.

11.7.4.1.3 Jitter

- The applied calibrated test signal shall have applied a calibrated amount of HF, GJ, and HPJ.
- The jitter control signal for generating High Probability Jitter should be filtered using at least a first order low pass filter with a corner frequency between 1/20 - 1/10 of the baud rate of the PRBS generator to ensure that high frequency components are removed. The distribution of the jitter after the filter must be reasonably even, symmetrical, and large spikes should be avoided. The order of the PRBS polynomial may be between 7 and 11, inclusive, to allow flexibility in meeting this objective. The rate of the PRBS generator should be between 1/10 - 1/3 of the data rate of the DUT, and their rates must be not harmonically related. The upper -3 dB frequency of the filtered HPJ should be at least 1/100 of the data rate of the DUT to represent transmitter jitter that is above the tracking frequencies of the DUT's CDR. Calibration of HPJ must be done with a golden PLL in place. Once these objectives are achieved, there is no need to vary these settings; any combination of settings that meets all the objectives is satisfactory.
- The jitter control signal for generating Unbounded Gaussian Jitter shall be filtered as per Figure 9-10 using the “Jitter Control Signal Filter”. However, the upper frequency of the Gaussian Jitter spectrum will be, acceptably, limited by the bandwidth of the voltage controlled delay line. The crest factor of the white noise generator should be better than 18 dB.
- The calibrated test signal shall have a calibrated amount of Total Wander and Relative Wander as compared to the used clock by using the Common SJ Wander and Antiphase SJ Sources with 1% frequency offsets (note the use of the inverted input to the uppermost delay line) as per Section 9.7.2, "Total Wander vs. Relative Wander".
- The amplitude of the Total Wander and Relative Wander is defined by the sinusoidal masks defined in Section 9.4.5, "Total Wander Mask" and Section 9.4.6, "Relative Wander Mask" with the specified amplitudes from the CEI IA.
- Wander should be applied
  – from a frequency equivalent to 1 UI of Total Jitter up to 20 MHz modulation frequency.
  – at a maximum of 2 MHz frequency steps above the corner frequency.
  – at a maximum of 200 kHz frequency steps below the corner frequency.

11.7.4.1.4 Amplitude

- The calibrated data signals should be filtered using single pole low pass filter with a corner frequency of 0.7 times the baud rate to define the edge rate.
- The amplitude of the signal should be adjusted such that it just passes the defined receiver data eye sensitivity.

- For testing of DC coupled receivers either a pattern generator capable of generating differential signals and setting the common mode should be used, or a combined AC coupled signal together with a biased-T. Using this setup the common mode should be varied between the defined maximum and minimum.

11.7.4.2 Jitter Tolerance with no Relative Wander Lab Setup

The following section describes the required jitter tolerance methodology for devices where Relative Wander is not applicable and no receive equalization is implemented.

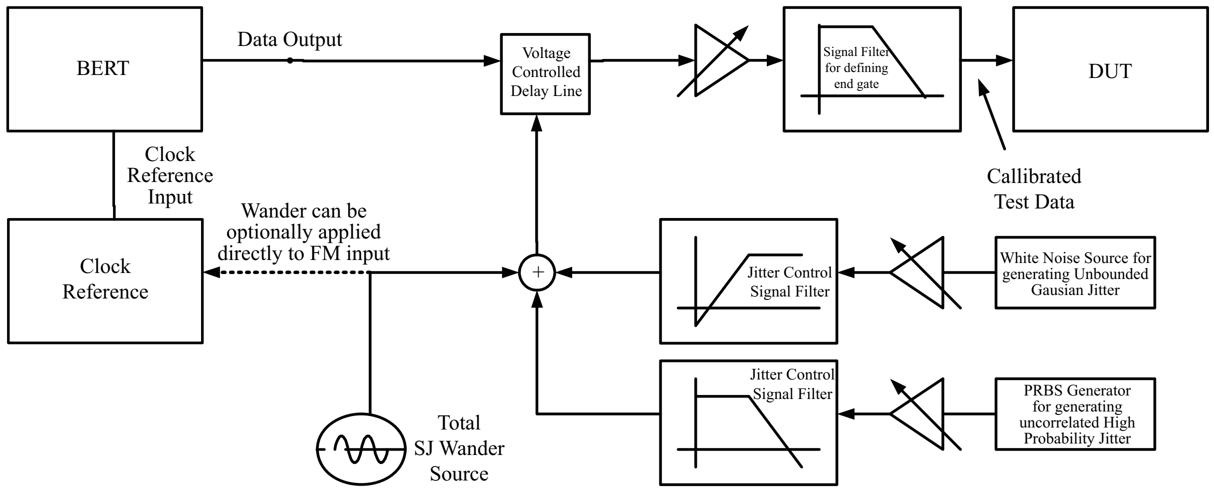

BERT
Data Output
Voltage Controlled Delay Line
Signal Filter for defining end gate
DUT
Clock Reference Input
Callibrated Test Data
Wander can be optionally applied directly to FM input
Clock Reference
+
Jitter Control Signal Filter
White Noise Source for generating Unbounded Gausian Jitter
Jitter Control Signal Filter
PRBS Generator for generating uncorrelated High Probability Jitter
Total SJ Wander Source

Figure 11-10. Jitter Tolerance with no Relative Wander

Referring to Figure 11-10, the DUT shall be tested as per the description in Section 11.7.4.1, "Jitter Tolerance with Relative Wander Lab Setup", omitting any requirements relating to relative wander and where only Total Wander is applied via the SJ Source shown.

11.7.4.3 Jitter Tolerance with Defined ISI and no Relative Wander

The following section describes the required jitter tolerance methodology for devices where Relative Wander is not applicable, e.g. SxI.5, and where receive equalization is implemented and the performance of the equalization must be verified.

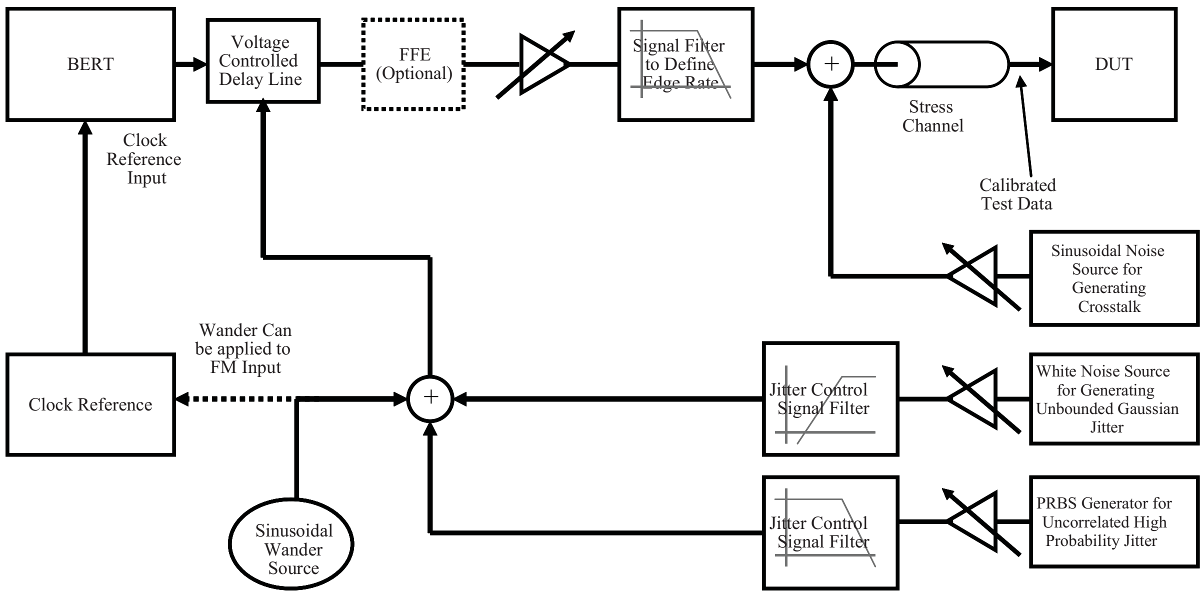

BERT
Voltage Controlled Delay Line
FFE (Optional)
Signal Filter to Define Edge Rate
+
Stress Channel
DUT
Calibrated Test Data
Clock Reference Input
Sinusoidal Noise Source for Generating Crosstalk
Wander Can be applied to FM Input
Clock Reference
+
Jitter Control Signal Filter
White Noise Source for Generating Unbounded Gaussian Jitter
Jitter Control Signal Filter
PRBS Generator for Uncorrelated High Probability Jitter
Sinusoidal Wander Source

Figure 11-11. Jitter Tolerance with Defined ISI

Referring to Figure 11-11, the DUT shall be tested as per the description in Section 11.7.4.1, "Jitter Tolerance with Relative Wander Lab Setup", omitting any requirements relating to relative wander, and additionally:

- The transmit jitter and amplitude shall be initially calibrated as per Section 11.7.1, "High Frequency Transmit Jitter Measurement" at the output of the delay line.
- A compliance channel shall be added.
- The defined amount of uncorrelated additive noise shall be applied via a sinusoidal source differentially to the signal. The frequency used shall be between 100 MHz and the lesser of 1/4 the data rate and 2 GHz. There is no need to sweep the frequency.

11.7.4.4 Jitter Transfer

This section describes how jitter transfer relevant interfaces can be tested for compliance:

- The BERT shall generate a data pattern as defined by the CEI IA.
- The jitter present before the delay line should be minimized so as to maximize any transfer bandwidth function of the DUT.
- A sinusoidal jitter should be applied following the same defined SJ mask as used for jitter tolerance and with the same resolution as described in Section 11.7.4, "Jitter Tolerance".

The peak to peak jitter for a 60 second period measured on the scope should be compared before and after the application of the sinusoidal jitter. The ratio of the difference to the jitter applied is then defined as the jitter transfer function.

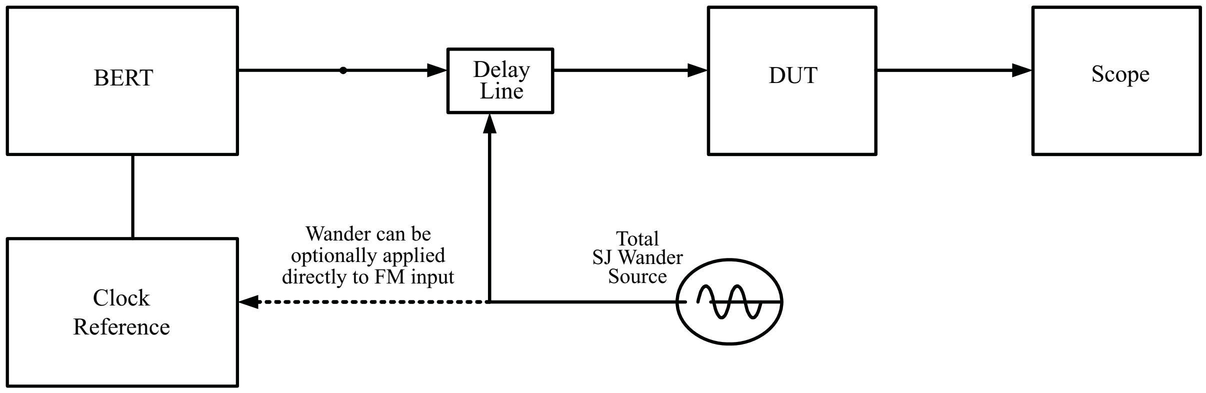


Figure 11-12. Jitter Transfer Lab Setup

11.7.4.5 Network Analysis Measurement

To enable accurate analysis of a channel the following methodology should be followed for the measurement and calculation of the effective channel transfer function.

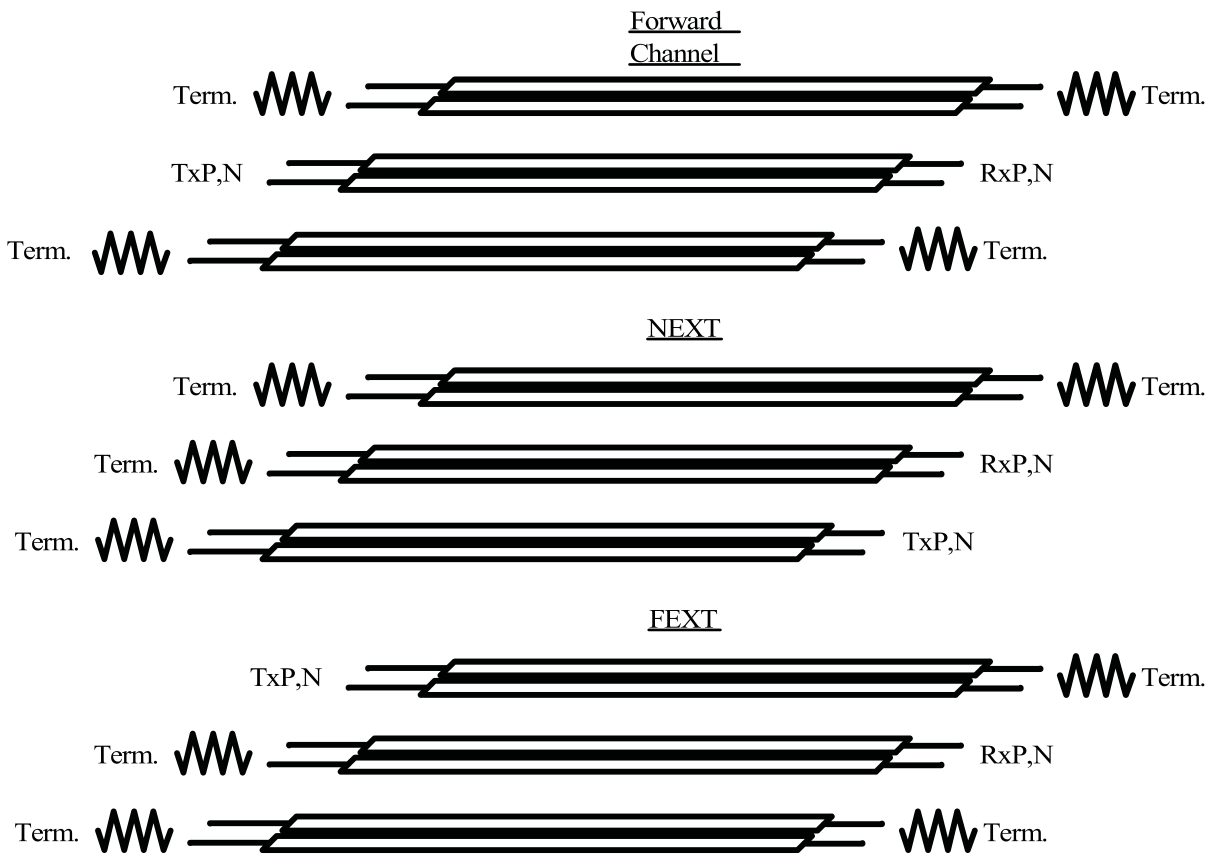


Figure 11-13. S-parameter Port Definitions

- Figure 11-13 shows an overview of the termination and port definitions typically used when measuring the forward channel and NEXT/FEXT crosstalk aggressors.
- The intermediate frequency (IF) bandwidth should be set to a maximum of 300 Hz with 100 Hz preferred. The launch power shall be specified to the highest available leveled output power not to exceed 0 dBm.
- Either direct differential measurements of the channel S21 and S11 should be performed or multiple single ended measurements from which the differential modes can be calculated.[1]
- Linear frequency steps of the measurements shall be no larger than 12.5 MHz.
- A frequency range from no higher than 100 MHz to no lower than three times the fundamental frequency should be measured.
- Extrapolation towards DC should be performed linearly on magnitude part with the phase being extrapolated to zero at DC, i.e. only a real part is present at DC.
- The channel response of the channel should be calculated by cascading the complete 4 port S-parameter matrix with a worst case transmitter and receiver. The transmitter/receiver should be described as a parallel R and C, where R is the defined maximum allowed DC resistance of the interface and C is increased until the defined maximum Return Loss at the defined frequency is reached.
- Any defined effective transmit or receiver filters should also be cascaded with the channel response.
- The time resolution should be increased by resampling the impulse response in the time domain.
- If required, interpolation of the frequency domain should be performed on the magnitude and unwrapped phase components of the channel response

$$Tr(\omega) = \begin{bmatrix} 1 & 1 \\ 1 & Tx_{22}(\omega) \end{bmatrix} \otimes \begin{bmatrix} S_{11}(\omega) & S_{21}(\omega) \\ S_{12}(\omega) & S_{22}(\omega) \end{bmatrix} \otimes \begin{bmatrix} Rx_{11}(\omega) & 1 \\ 1 & 1 \end{bmatrix}$$

where

$S_{m,n}$ is the measured 4 port differential data of the channel

Tx_{22} is the transmitter return loss

Rx_{11} is the receiver return loss

$Tr(\omega)$ is the receiver return loss.

Converting the original frequency range to time domain, we obtain

[1]Special care must be taken when performing multiple single ended measurements if the system is tightly coupled

$$i(t_m) = ifft(Tr(\omega))$$

where

$$\omega = [-\frac{3}{4}f_{baud}, \frac{3}{4}f_{baud}]$$

11.7.4.6 Eye Mask Measurement Setup

The measurement of an eye mask is defined by the various CEI IAs in terms of a polygon for the probability of the required Bit Error Rate. This polygon may have to be altered given that the sample population of the scope is limited and must be adjusted as per Annex B.3, “Eye Mask Adjustment for Sampling Oscilloscopes". For the measurement of the signal the laboratory setup shown in Figure 11-14 should be used, including the recommendations list in Section 11.7.1, "High Frequency Transmit Jitter Measurement".

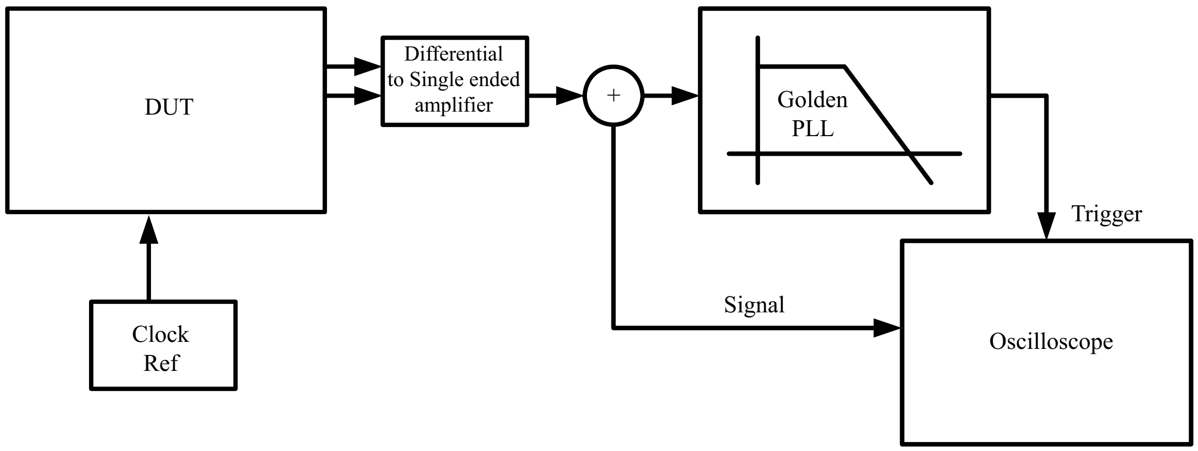


Figure 11-14. Mask Measurement with Golden PLL

Chapter 12 Electrical Specification for 10.3125 and 12.5 Gbaud LP-Serial Links

This chapter details the requirements for Level III RapidIO LP-Serial short and long run electrical interfaces of nominal baud rates of 10.3125 and 12.5 Gbaud using NRZ coding (hence 1 bit per symbol at the electrical level). A compliant device must meet all of the requirements listed below. The electrical interface is based on a high speed low voltage logic with a nominal differential impedance of 100 Ω. Connections are point-to-point balanced differential pair and signaling is unidirectional.

12.1 References

1. IEEE Standard 802.3-2008. “IEEE Standard for Information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements - Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications”, IEEE Std. 802.3-2008, December 26, 2008.
2. IEEE Standard 802.3ba-2010. “IEEE Standard for Information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements - Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications. Amendment 4: Media Access Control Parameters, Phhysical Layers, and Management Parameters for 40 Gb/s and 100 Gb/s Operation”, IEEE Std. 802.3ba-2010, June 22, 2010.

12.2 Level III Application Goals

12.2.1 Common to Level III Short run and Long run

The following are application requirements common to short run and long run Level III links at 10.3125 and 12.5 Gbaud:

- The electrical specifications shall support lane width options of 1x, 2x, 4x, 8x and 16x.
- A compliant device must implement AC coupling.

- A compliant device may implement any subset of baud rates contained in this chapter.
- A compliant device may implement either a short run transmitter, a long run transmitter, or both, at each of the baud rates that it supports.
- A compliant device may implement either a short run receiver or a long run receiver at each of the baud rates that it supports.
- The clock frequency tolerance requirement for transmit and receive are ±100 ppm. The worst case frequency differences between any transmit and receive clock is 200 ppm.
- The Bit Error Ratio (BER) shall be better than 10^{-15} per lane but the test requirements will be to verify 10^{-12} per lane.
- Transmitters and receivers used on short and long run links shall inter-operate for path lengths up to 20 cm.
- Transmitters and receivers used on long run links shall inter-operate for path lengths up to 100 cm.
- The transmitter pins shall be capable of surviving short circuit either to each other, to supply voltages, and to ground.

12.2.2 Application Goals for Level III Short Run

- The short run interface shall be capable of spanning at least 20 cm of PCB material with up to a single connector.

12.2.3 Application Goals for Long Run

- The long run interface shall be capable of spanning at least 100 cm of PCB material with up to two connectors at 10.3125 Gbaud.
- An AC coupled long run receiver shall be inter-operable with an AC coupled short run transmitter
- An AC coupled long run transmitter shall be inter-operable with an AC coupled short run receiver provided that the signal swing values are lowered. This implies that the signal swing is configurable.
- The long run PHY may use techniques such as increased signal swing, linear equalization, and Decision Feedback Equalizer, designed to accommodate longer run backplane applications, where the receiver eye may be closed.
- A long run transmitter and receiver is intended to accommodate ‘legacy’ long run RapidIO 1.3 backplanes of at least 60 cm with up to two connectors that can operate at data rates up to 10.3125 Gbaud.

12.3 Equalization

At the high baud rates used by Level III LP-Serial links, the signals transmitted over a link are degraded by losses and characteristic impedance discontinuities in the interconnect media. The losses increase with increasing baud rate and interconnect media length and cause signal attenuation and inter-symbol interference that degrade the opening of the eye pattern at both the receiver input and the data decoder decision point. Depending on the baud rate and interconnect length, the degradation can be greater than that allowed by the specification.

The signal degradation can be partially negated by the use of equalization in the transmitter and/or receiver. Equalization in the transmitter can improve the eye pattern at both the receiver input and the data decoder decision point. Equalization in the receiver can only improve the eye pattern at the data decoder decision point. Some degree of equalization is required by most Level III interconnects.

12.3.1 Receiver

Adaptive equalization in the receiver and the algorithms for training that equalization are entirely within the receiver. The configurations, characteristics and adjustment algorithms for equalization in the receiver are implementation specific and outside the scope of this specification.

12.3.2 Transmitter

Adaptive equalization in the transmitter shall be controlled by the connected receiver.

12.4 Level III Electrical Specification

Two sets of electrical specifications are specified for 10.3125 and 12.5 Gbaud, a short reach set and a long reach set. The transmitters and receivers of an LP-Serial port operating at a nominal baud rate of 10.3125 or 12.5 Gbaud shall comply with at least one of these sets of specifications.

12.4.1 Level III Short Run

The electrical specifications for the short-reach 10.3125 and 12.5 Gbaud PHY shall be the same as those specified in Annex 83A.3 of the IEEE Standard 802.3ba-2010[2] for XLAUI/CAUI. The specifications for the short-reach channel shall be the same as those specified in Annex 83A.4 of the IEEE Standard 802.3ba-2010[2] for a single XLAUI/CAUI lane.

12.4.2 Level III Long Run

The electrical specifications for the long-reach 10.3125 and 12.5 Gbaud PHY shall be the same as those specified in Clauses 72.6.1 and 72.7.1 through 72.9.5 of the IEEE Standard 802.3-2008[1] (Part 5). The specifications for the long-reach channel shall be the same as those specified for the 10GBASE-KR channel in Annex 69A of the IEEE Standard 802.3-2008[1] (Part 5).

12.4.3 Level III Transmitter Lane-to-Lane Skew

The electrical level of lane-to-lane skew caused by the transmitter circuitry and associated routing must be less than 2*67 UI + 1000 ps. The transmitter lane-to-lane skew is only for the SerDes Tx and does not include any effects of the channel.

12.4.4 Receiver Input Lane-to-Lane Skew

The maximum amount of lane-to-lane skew at the input pins of the receiver is determined by the ability of the receiver to resolve the difference between two successive Status/Control columns. Since the minimum number of non-Status/Control columns between Status/Control columns is 16, the maximum lane skew that can be unambiguously corrected is the time it takes to transmit 7 codewords per lane. Therefore, the maximum lane-to-lane skew at the input pins of a receiver is calculated as follows:

(7 codewords) x (67 bits/codeword) x (1 UI/bit) x (ns/UI)

It is important to note that the total lane-to-lane skew specification includes the skew caused by the transmitter’s PCS and PMA (SerDes), the channel, the receiver’s PCS and PMA (SerDes), and any logic that is needed to create the aligned column of Status/Control at the receiving device.

12.4.5 Electrical IDLE

When a Level III transmitter is disabled due to the deassertion of the drvr_oe[k] signal, the transmitter shall output a constant output level with no transitions. It is also recommended that the transmitter outputs should meet the requirements for ‘Differential peak-to-peak output voltage (max.) with TX disabled’ as specified in Table 72-6 of IEEE 802.3-2008 [1] (Part 5).

Annex A Transmission Line Theory and Channel Information (Informative)

A.1 Transmission Lines Theory

The performance of a high frequency transmission line is strongly affected by impedance matching, high frequency attenuation and noise immunity.

It is possible to design a high frequency transmission line using only a single conductor. Nevertheless, most high frequency signals use differential transmission lines (i.e. a pair of coupled conductors carrying signals of opposite polarity). Although differential signaling appears wasteful of both pins and signal traces it results in much better noise immunity. Differential signals produce less conducted noise because the opposite power and ground current flows cancel each other both in the line driver and in the transmission line. Differential signals produce less radiated noise because over a modest distance the opposite fields induced by the opposite currents cancel each other. Differential signals are less susceptible to noise because most sources of noise (common mode noise) tend to affect both signal lines identically, producing a variation in common mode voltage but not in differential voltage.

A.2 Impedance Matching

The AC impedance of a single conductor is determined by the trace geometry, distance to the nearest AC ground plane(s) and the dielectric constant of the material between the trace and the ground plane(s). If the distance between the signal trace and the nearest ground plane is significantly less than the distance to other signal traces the signal trace will behave as a single-ended transmission line. Its AC impedance does not vary with signal polarity although it may vary with frequency due to the properties of the dielectric material. This impedance is often called single ended impedance, Zse.

The AC impedance, Z of a differential transmission line is affected by the configuration of the pair of conductors and the relationship between their signal polarities, in addition to the trace geometry, distance to the nearest AC ground plane(s) and the dielectric constant of the material between the trace and the ground plane(s). If the paired conductors are close enough to interact (coupled), then the impedance for signals of opposite polarity (odd mode impedance, Zodd) will be lower than the impedance for signals of the same polarity (even mode impedance, Zeven).

If there is minimal coupling between the paired conductors then *Zodd* = *Zeven* = *Zse*. Coupled transmission lines always produce *Zodd* < *Zse* < *Zeven*. The following equations relate effective differential impedance, *Zdiff*, to common mode impedance, *Zcm*, and single ended impedance, *Zse*, to even and odd mode impedances:

$$Zdiff = 2Zodd \qquad Zcm = \frac{Zeven}{2} \qquad Zse = \frac{Zeven + Zodd}{2}$$

Most differential data signals are designed with *zdiff* = 100Ω and 25Ω < *Zcm* < 50Ω.

There is a trade-off in the choice of *Zcm*. *Zcm* = 25Ω (no coupling) may reduce conducted noise for transmission lines with inadequate AC or DC grounding. *Zcm* = 50Ω (close coupling) may reduce radiated noise (crosstalk) which is more critical in backplanes. However close coupling requires careful ground construction to control common mode noise.

The reader may wonder why common mode impedance is meaningful in a differential transmission system. In a perfectly constructed system only odd mode (opposite polarity) signals propagate. However imperfections in the transmission system cause differential to common mode conversion. Once converted into common mode the energy may convert back to differential mode by the same imperfections. Thus, these imperfections convert some of the signal energy from opposite polarities to the same polarity and back.

The two main sources of mode conversion are impedance mismatches which cause part of the energy to be reflected, and differential skew which causes variations in forward signal propagation delay between the individual paths of the differential pair. Impedance mismatches typically occur at boundaries between transmission line segments, including wire bonds, solder joints, connectors, vias, and trace-to-via transitions. Often ignored sources of impedance mismatches at these boundaries are discontinuities within the AC ground itself as well as asymmetric coupling between the individual traces and the AC ground. Differential skew can occur at these same boundaries and also due to mismatched trace lengths in device packages and in PCBs.

A.3 Impedance Definition Details

Differential transmission lines consist of two conductors and a ground plane. The voltage-current relationships at one end of this line can be formulated in terms of a two-port as in Figure A-1.

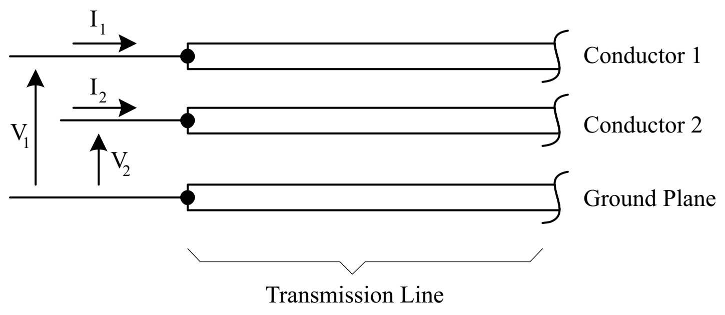


Figure A-1. Transmission Line as 2-port

The voltage current relationships are:

$$V_1 = Z_{11}I_1 + Z_{12}I_2 \qquad V_2 = Z_{21}I_1 + Z_{22}I_2$$

If the line is infinitely long or perfectly terminated, then these four impedance values are the characteristic impedance of the line. The characteristic impedance is a 2 x 2 matrix:

$$\hat{Z}_c = \begin{bmatrix} Z_{11} & Z_{12} \\ Z_{21} & Z_{22} \end{bmatrix}$$

Generally, all four of the matrix entries are complex. But, at frequencies of interest, the inductance and capacitance per unit length dominate so that all four quantities are approximately real positive numbers. For engineering purposes it is common to speak of the impedances as though they are resistances with no imaginary part, keeping in mind that the imaginary part exists. Since the line is passive and symmetric, we have $Z_{11} = Z_{22}$ and $Z_{12} = Z_{21}$ so that the line is described by just two impedance values. If the line is to be perfectly terminated, then we must create a network that is equivalent to $\hat{Z}_c$. That is, we need a 3-terminal (2 nodes + ground) network that presents the same values of Z_{11} and Z_{12} as the line. A T or pi network could be used. The pi network is shown in Figure A-2, along with the impedance values in terms of Z_{11} and Z_{12}.

$$Za = Z_{11} + Z_{12} \qquad Zb = \frac{Z_{11}^2 - Z_{12}^2}{Z_{12}}$$

$$Zodd = \frac{ZaZb}{2Za + Zb} = Z_{11} - Z_{12} \qquad Zeven = Za = Z_{11} + Z_{12}$$

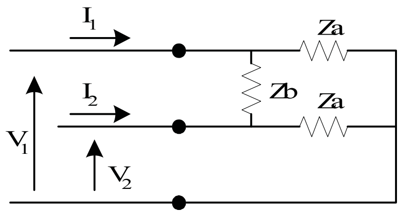


Figure A-2. Network Terminations

The odd and even mode impedances, *Zodd* and *Zeven*, are other impedance definitions that are more descriptive, referring to the polarity of the signal propagating the differential pair. In the case of opposite signal polarity in the two lines of the signal pair the odd mode impedance is used. In the case of same signal polarity the even mode is used. *Zodd* and *Zeven* are measured as shown in Figure A-3.

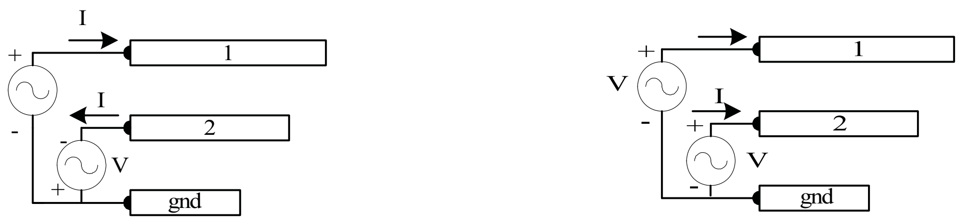


Figure A-3. Measurement of Zodd, Zeven

Zodd	*Zeven*
$V = V_1 = -V_2$	$V = V_1 = \mathrm{V}_2$
$I = I_1 = -I_2$	$\mathrm{I} = I_1 = I_2$
$Zodd = \frac{V}{I}$	$Zeven = \frac{V}{I}$

Odd mode impedance is the impedance measured when the two halves of the line are driven by equal voltage or current sources of opposite polarity. Even mode impedance is the impedance measured when the two halves of the line are driven by equal voltage or current sources of the same polarity.

From the above equations we see that *Zeven* is always greater than *Zodd* by $2Z_{12}$, where Z_{12} is a measure of the amount of coupling between the lines. This means that *Zeven* is larger than *Zodd* for coupled transmission lines.

A.4 Density considerations

The preceding section showed that, for two idealized forms of termination, *Zodd* is correctly terminated but *Zeven* is not. The first illustrated case, using a 50Ω resistor (or its equivalent) from either terminal to ground (or to AC ground), has become relatively standard. Because it has *ZoddT* = *ZevenT* = 50Ω, it provides correct differential termination and is often close to providing correct common-mode termination.

By increasing the conductor spacing in the transmission line we can decrease Zeven (decrease z_{12}) and bring it closer to 50Ω. But dense backplanes require a large number of transmission lines per unit cross-sectional area of the printed circuit board. This means that the two printed circuit traces comprising the differential transmission line are forced close together, which increases z_{12}. The backplane design is therefore, a compromise between the desire for high density of transmission lines and a desire for correct common-mode termination.

Transmission lines act as low-pass filters due to skin effect and dielectric absorption. As the density of transmission lines increases, both the series resistance per unit length and the parallel conductance per unit length increase. This, in turn, results in greater attenuation at a given frequency. Thus, high speed backplane design is not just a compromise between density and common-mode matching. There is also a compromise between density and attenuation.

A.5 Common-Mode Impedance and Return Loss

It is demonstrated above that increasing the density of transmission lines in a backplane results in higher common-mode impedance, which is known as interference, and for high amplitudes the receiver is likely to be disrupted.

Common-mode interference arises from several sources. Among them are:

1. Imperfections in driver circuits
2. A difference in length between the two conductors of the transmission line
3. Imperfections in impedance matching across board boundaries, connectors, and vias causing mode conversion, from differential to common mode
4. EMI

The interference resulting from the driver probably has a spectrum that is the same as or similar to that of the signal. EMI arising from coupling into the printed circuit traces should be small, assuming that coupled stripline is used. However, connector pins may be exposed. EMI may have frequency components that are well below signal frequencies, which means that it won’t necessarily be attenuated to the extent that signals are. But, at the same time, the lower frequencies are probably poorly coupled into the backplane circuit.

Earlier, two ideal forms of termination were presented based on either one or two resistors. These ideal terminating devices are helpful in examining the relationship between the parameters of the transmission line versus those of the device. Real devices, however, are not simple resistances. They contain parasitic components and a non-ideal path from package pins to die. There may also be a need to AC-couple the terminations.

The most that can be done in this situation is to make the package and the die appear as close to ideal as possible over as much of the signal spectrum as possible. The extent of the deviation from ideal is specified and measured as a function of frequency. The preferred measures are S_{11} (single-ended return loss) or S_{DD11} (differential return loss) as functions of frequency. (Sometimes S_{22} or S_{DD22} are used to indicate an output.) Ideally these return losses are 0 (no reflection) over the frequency range of interest. In dB this is $-\infty$.

Note: Sometimes a return loss is specified as a positive number, it being understood that this still refers to the log of a reflection coefficient in the range of 0 to 1.

A.6 Crosstalk Considerations

This implementation assumes that the dominant cross talk can come from aggressors other than the transmitter associated with the receiver. Hence NEXT cancellation is not useful.

Crosstalk between channels should be minimized by good design practices. This includes the pin-out arrangement to the driving/receiving ICs, connectors and backplane tracking.

Optimum arrangement for minimizing crosstalk between channels at IC pins is illustrated in Figure A-4 below. Crosstalk between channels can be reduced by grouping TX and RX pins and avoiding close proximity between individual TX and Rx pins. This practice will minimize coupling of noise from TX drivers into RX inputs.

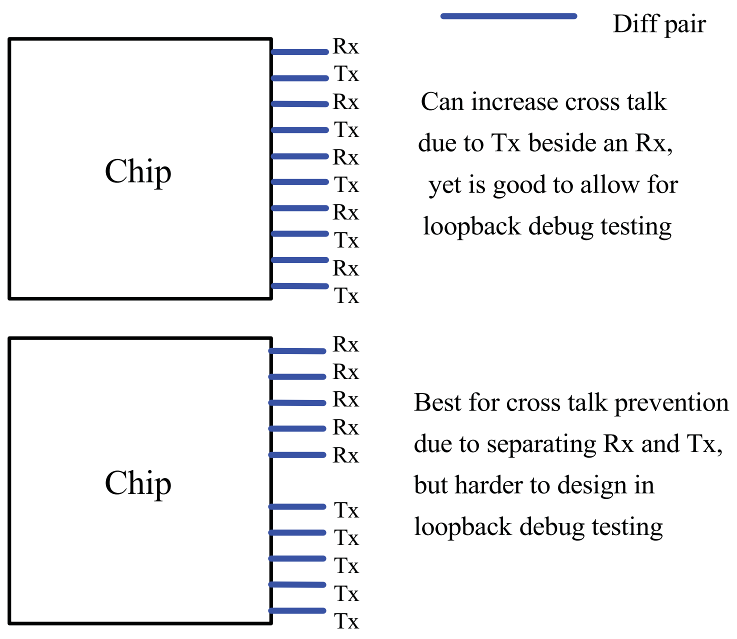


Figure A-4. Minimization of Crosstalk at IC Pins

Crosstalk at connector pins can be minimized by careful optimization of connections as shown in Figure A-5 below.

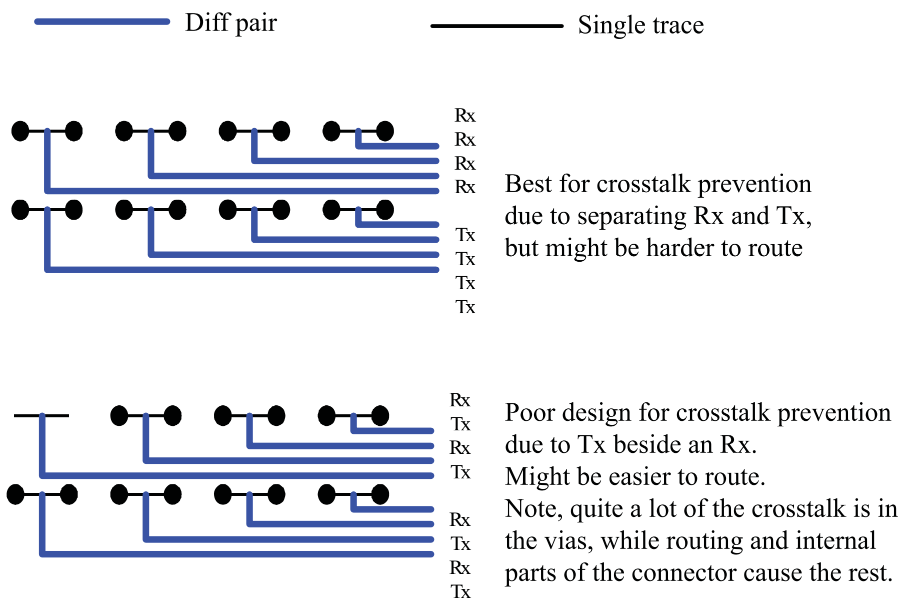

Diff pair
Single trace
Rx
Rx
Rx
Rx
Tx
Tx
Tx
Tx
Best for crosstalk prevention due to separating Rx and Tx, but might be harder to route
Rx
Tx
Rx
Tx
Rx
Tx
Rx
Tx
Poor design for crosstalk prevention due to Tx beside an Rx.
Might be easier to route.
Note, quite a lot of the crosstalk is in the vias, while routing and internal parts of the connector cause the rest.

Figure A-5. Minimization of Crosstalk At Connector Pins

Crosstalk between channels over a backplane can be minimized by careful arrangement of tracking, avoiding coupling of noise into RX inputs and increasing spacing “d” between channels as far as possible as shown in Figure A-6 below.

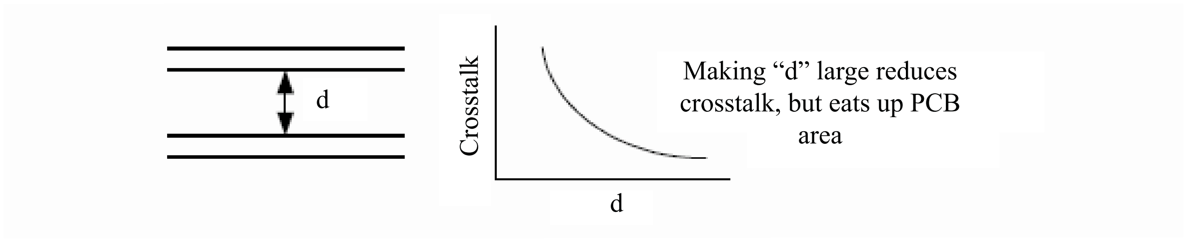


Figure A-6. Minimization of Crosstalk Over Backplane

A.7 Equation Based Channel Loss by Curve Fit

This section describes a technique with specific limitations. It does not include any phase data for the SDD_{21}, and includes no return loss information about SDD_{11} or SDD_{22}, information that is critical for the evaluation of a specific topology's performance. The preceding proposed statistical-eye characterization includes these effects by including the full 4-port s-parameter measurements. The following method is included for information only and is believed to be of relevance to the overall understanding of the channel transfer loss.

One way to specify the channel loss is to have an average or worst case “curve” fit to several real channels. This method includes effects of real vias and connectors. This method typically uses the equation below:

$$Att = -20 * \log(e) * \left(a_1 * \sqrt{f} + a_2 * f + a_3 * f^2\right)$$

Where ***f*** is frequency in Hz, ***a*₁**, ***a*₂**, and ***a*₃** are the curve fit coefficients and ***Att*** is in dB.

Table A-1 gives some examples of these coefficients and Figure A-7 plots them along with the PCB model and a real 75cm backplane with 5cm paddle cards on both ends. These examples are representative of Level II LR applications but do not represent specifications that a RapidIO link is to comply with.

Table A-1. Curve fit Coefficients

Channel	a_1	a_2	a_3
sRIO[1] (50cm)	6.5e-6	2.0e-10	3.3e-20
75cm[2] “Worse”	6.5e-6	3.9e-10	6.5e-20
75cm[3] “Typical”	6.0e-6	3.9e-10	3.5e-20

[1]Chapter 8 reference 1
[2]Chapter 8 reference 5
[3]Chapter 8 reference 5

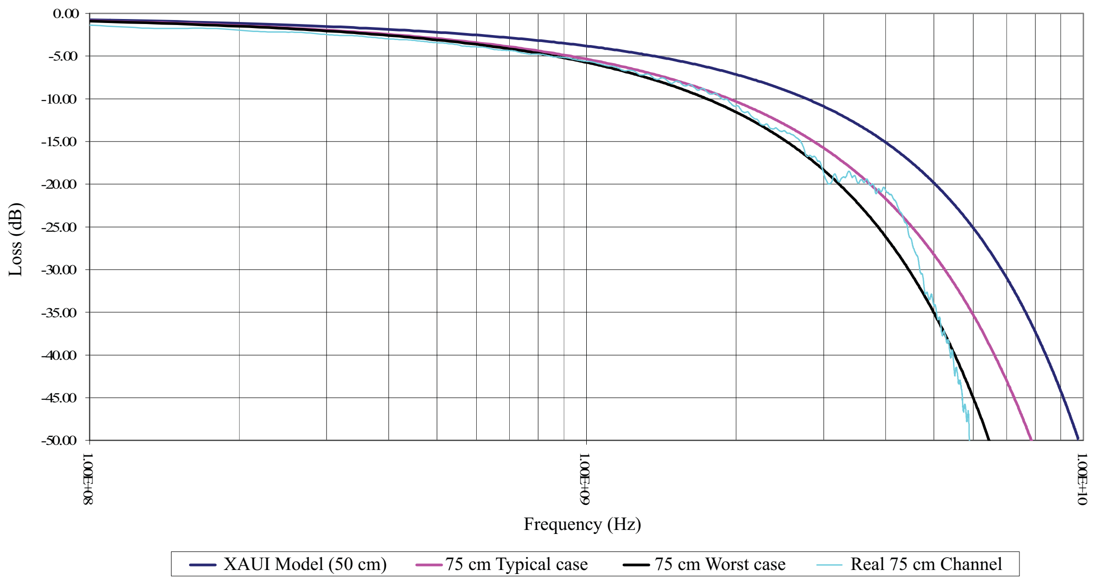


Figure A-7. Equations Based Channel Loss Curves

Annex B BER Adjustment Methodology (Informative)

B.1 Extrapolation of Correlated Bounded Gaussian Jitter to low BERs

For this specification, which has a BER requirement of 1×10^{-15} or lower, measurements to that level are very time consuming (or rely on averaging multi-links), hence it is more practical to only take measurements to Qs around 7 (BER around 1×10^{-12}).

B.1.1 Bathtub Measurements

CBGJ can appear as either GJ or CBHPJ depending upon the Q at which it is linearized.

If HPJ and GJ are measured using a bathtub there is no knowledge as to if the GJ is UUGJ or CBGJ. For system budgeting it is recommended that the bathtub GJ should be assumed to be all UUGJ.

If combined spectral oscilloscope methods are used then UUGJ, UBHPJ, and CBHPJ can be estimated. It is not possible to estimate the CBGJ as it has already become bounded and appears as CBHPJ. For system budgeting it is recommended that this peak value is valid for the extrapolated Q of interest.

B.2 Confidence Level of Errors Measurement

Assuming that a link with a given BER can be modelled as a Bernoulli random process, the following statistics can be assumed.

Given,

p is the probability of error

$q = (1 - p)$ is the probability of not having an error

n is the number of bits received and measured

then

$m = np$ is the expected number of errors received

$\sigma = \sqrt{npq}$ is the sigma of the variation of the number of errors received

As an example process, for a 3 sigma confidential level

$$p = 10^{-12}$$

$$n = 100 \cdot 10^{12}$$

$$m = 100$$

$$\sigma = 10$$

$$m\Big|_{max}^{min} = [m + Q\sigma]\Big|_{Q = -3}^{Q = 3}$$

$$m\Big|_{max}^{min} = \begin{matrix} 70 \\ 130 \end{matrix}$$

To assess the accuracy of such a measurement an equivalent process with a higher BER can be calculated that would show the same limit of error for the same confidence level and measured number of bits.

$$\begin{aligned} m\big|_{max} &= E[m] - Q\sigma \\ m\big|_{max} &= np - Q\sqrt{npq} \\ m\big|_{max} &= np - Q\sqrt{np(1-p)} \end{aligned}$$

Solving the quadratic equation for p

$$p = 1.69 \times 10^{-12}$$

B.3 Eye Mask Adjustment for Sampling Oscilloscopes

The data mask is defined for the bit error rate of the link. Given that this bit error rate is very small, typical oscilloscope measurement will not sample enough points to be able to verify compliance to these mask.

B.3.1 Theory

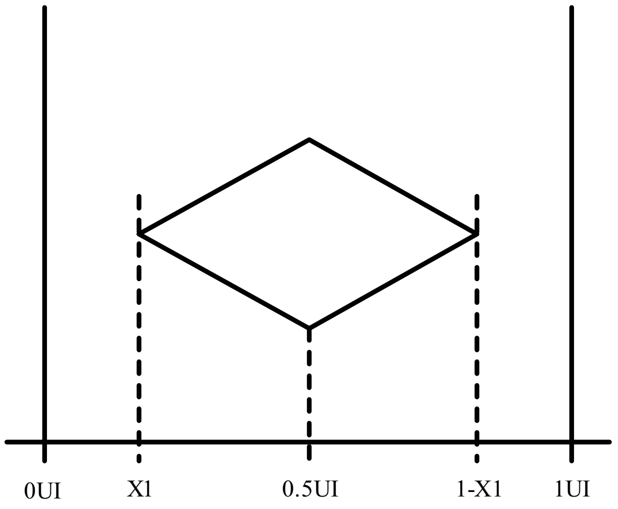


Figure B-1. Example Data Mask

Given an example eye mask, Figure B-1, the extremes of the mask, X1, are defined as a linear addition of a Gaussian and high probability jitter component.

$$X1 = \frac{HPJ}{2} + Q \cdot GJ_{rms}$$

where

HPJ is the high probability jitter

GJ_{rms} is the Gaussian distributed jitter

Q is the GJ multiplication factor

Given a low sample population and the requirements for mask verification to achieve a hit or no-hit result, X1 must be adjusted according to the sample population and the confidence level that a particular peak to peak is achieved.

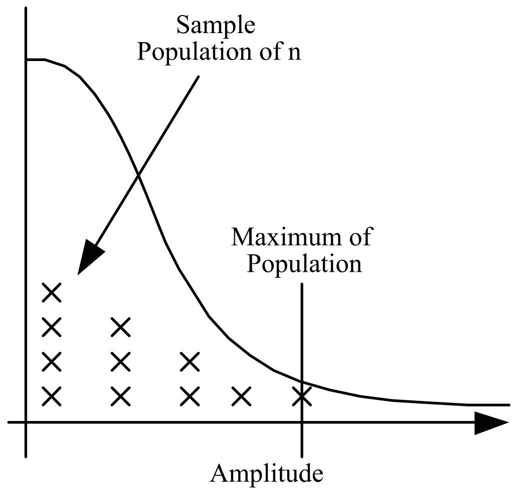


Figure B-2. Example Data Mask

Given a random process the probability of measuring a particular maximum amplitude on an oscilloscope requires one sample to lie on the maximum and all other samples to lie below this value. Referring this all to a half Gaussian distribution and a population of n, there are n different ways this can occur,

$$P(x_m) = nQ(x_m)\left(\int_0^{x_m} Q(x)dx\right)^{n-1}$$

where

x_m is the random variable of the maximum amplitude measured

x is the random variable of the underlying random jitter process

$Q(x)$ is the Q function of the Normal probability density function

n is the sample population

$P(x_m)$ is a probability density function

The equation above is solved and the probability of attaining a given maximum (normalized to the sigma) for various populations plotted, Figure B-3.

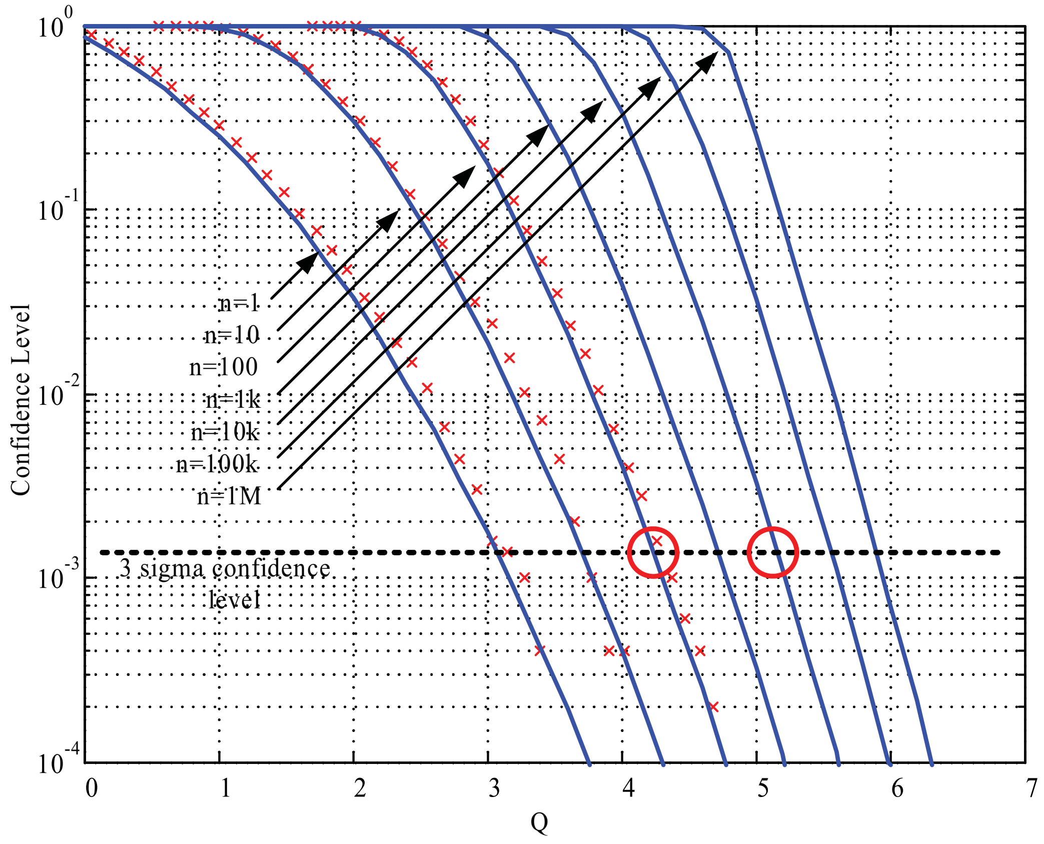


Figure B-3. Cumulative Distribution Function of Maximum Amplitude

B.3.2 Usage

Given a known sampling population, n, calculated from the measurement time, average transition density and sampling/collection frequency of the oscilloscope the three sigma confidence level (i.e. 1.3×10^{-3}) of the measured Gaussian jitter peak value can be read from Figure B-3. This value should be multiplied by 2 to give the full peak to peak value of the random jitter.

The three sigma confidence level should be understood as ensuring that 99.96% of all good devices do not violate the eye mask. To limit the number of bad devices that also pass the eye mask it is strongly recommended that the sample population be chosen as to give a Q larger than 5.

For example, referring to the red circled intersections Figure B-3, if we calculate that the sample population for an oscilloscope was 100 i.e. n=100, then for a 3 sigma confidence this equals a Q of 4.2. As the recommended Q value is 5 we should increase the sample population to 10k to give a Q of 5.2.

Annex C Interface Management (Informative)

C.1 Introduction

This appendix contains state machine descriptions that illustrate a number of behaviors that are described in the *RapidIO Part 6: LP-Serial Physical Layer Specification*. They are included as examples and are believed to be correct, however, actual implementations should not use the examples directly.

C.2 Packet Retry Mechanism

This section contains the example packet retry mechanism state machine referred to in Section 6.8, “Packet Transmission Protocol“.

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

C.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 C-1.

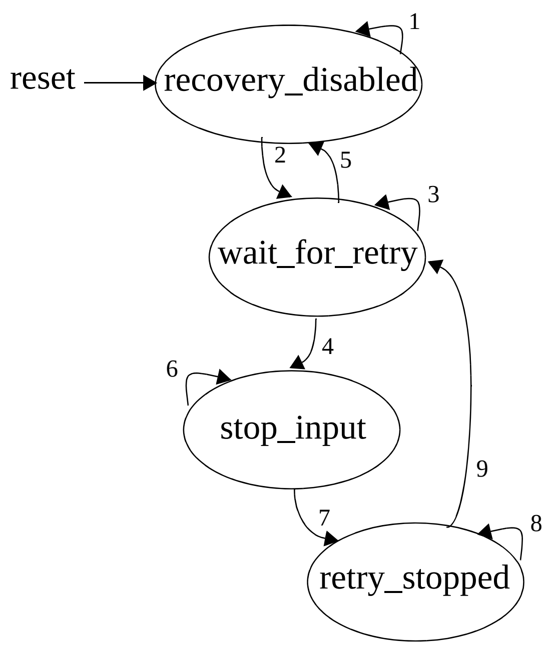


Figure C-1. Input Port Retry Recovery State Machine

Table C-1 describes the state transition arcs for Figure C-1. The states referenced in the comments in quotes are the RapidIO LP-Serial defined status states, not states in this state machine.

Table C-1. 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 initialization sequence has been completed, and may be controlled through other mechanisms as well, such as a software enable bit.
2	recovery_disabled	wait_for_retry	Input port is enabled.	
3	wait_for_retry	wait_for_retry	Remain in this state until a packet retry situation has been detected.	
4	wait_for_retry	stop_input	A packet retry situation has been detected.	Usually this is due to an internal resource problem such as not having packet buffers available for low priority packets.
5	wait_for_retry	recovery_disabled	Input port is disabled.	
6	stop_input	stop_input	Remain in this state until described input port stop activity is completed.	Send a packet-retry control symbol with the expected ackID, discard the packet, and don’t change the expected ackID. This will force the attached device to initiate recovery starting at the expected ackID. Clear the “Port Normal” state and set the “Input Retry-stopped” state.
7	stop_input	retry_stopped	Input port stop activity is complete.	

Table C-1. Input Port Retry Recovery State Machine Transition Table (Continued)

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

C.2.2 Output port retry recovery state machine

On receipt of an error-free packet-retry control symbol, the attached output port follows the behavior shown in Figure C-2. The states referenced in the comments in quotes are the RapidIO LP-Serial defined status states, not states in this state machine.

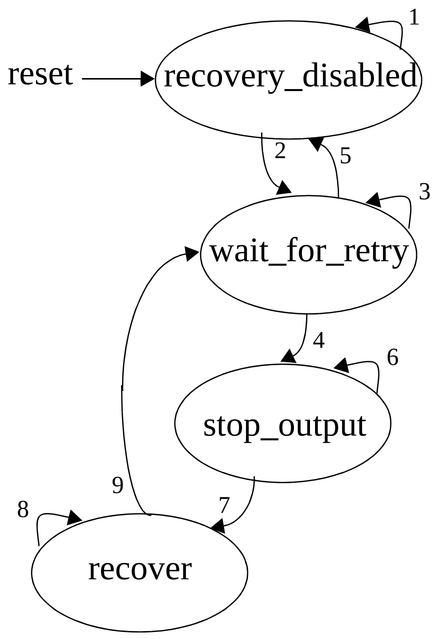


Figure C-2. Output Port Retry Recovery State Machine

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

Table C-2. 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 initialization sequence has been completed, and may be controlled through other mechanisms as well, such as a software enable bit.
2	recovery_disabled	wait_for_retry	Output port is enabled.	
3	wait_for_retry	wait_for_retry	Remain in this state until a packet-retry control symbol is received.	The packet-retry control symbol shall be error free.
4	wait_for_retry	stop_output	A packet-retry control symbol has been received.	Start the output port stop procedure.
5	wait_for_retry	recovery_disabled	Output port is disabled.	
6	stop_output	stop_output	Remain in this state until the output port stop procedure is completed.	Clear the "Port Normal" state, set the "Output Retry-stopped" state, and stop transmitting new packets.
7	stop_output	recover	Output port stop procedure is complete.	
8	recover	recover	Remain in this state until the internal recovery procedure is completed.	The packet sent with the ackID value returned in the packet-retry control symbol and all subsequent packets shall be retransmitted. Output port state machines and the outstanding ackID scoreboard shall be updated with this information, then clear the "Output Retry-stopped" state and set the "Port Normal" state to restart the output port. Receipt of a packet-not-accepted control symbol or other output port error during this procedure shall cause a clean transition between the retry recovery state machine and the error recovery state machine. Send restart-from-retry control symbol.
9	recover	wait_for_retry	Internal recovery procedure is complete.	Retransmission has started, so return to the wait_for_retry state to wait for the next packet-retry control symbol.

C.3 Error Recovery

This section contains the error recovery state machine referred to in Section 6.13.2, “Link Behavior Under Error."

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

C.3.1 Input port error recovery state machine

There are a variety of recoverable error types described in detail in Section 6.13.2, “Link Behavior Under Error“. The first group of errors are associated with the input port, and consists mostly of corrupt packet and control symbols. An example of a corrupt packet is a packet with an incorrect CRC. An example of a corrupt control symbol is a control symbol with error on the 5-bit CRC control symbol. The recovery state machine for the input port of a RapidIO link is shown in Figure C-3.

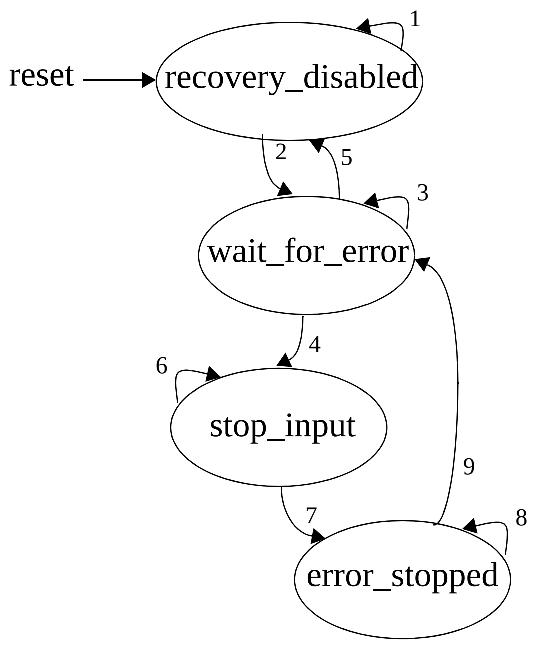


Figure C-3. Input Port Error Recovery State Machine

Table C-3 describes the state transition arcs for Figure C-3. The states referenced in the comments in quotes are the RapidIO LP-Serial defined status states, not states in this state machine.

Table C-3. 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 initialization sequence has been completed, and may be controlled through other mechanisms as well, such as a software enable bit.
2	recovery_disabled	wait_for_error	Error recovery is enabled.	
3	wait_for_error	wait_for_error	Remain in this state until a recoverable error is detected.	Detected errors and the level of coverage is implementation dependent.
4	wait_for_error	stop_input	A recoverable error has been detected.	An output port associated error will not cause this transition, only an input port associated error.
5	wait_for_error	recovery_disabled	Error recovery is disabled.	This can occur when a port operating with IDLE3 deasserts receive_enable, or whenever any port deasserts port_initialized.
6	stop_input	stop_input	Remain in this state until described input port stop activity is completed.	Send a packet-not-accepted control symbol and, if the error was on a packet, discard the packet and don't change the expected ackID value. This will force the attached device to initiate recovery. Clear the "Port Normal" state and set the "Input Error-stopped" state.
7	stop_input	error_stopped	Input port stop activity is complete.	
8	error_stopped	error_stopped	Remain in this state until a link request (restart-from-error) control symbol is received.	The "Input Error-stopped" state causes the input port to silently discard all subsequent incoming packets and ignore all subsequent input port errors.
9	error_stopped	wait_for_error	Received a link request (restart-from-error) control symbol.	Clear the "Input Error-stopped" state and set the "Port Normal" state, which will put the input port back in normal operation.

C.3.2 Output port error recovery state machine

The second recoverable group of errors described in Section 6.13.2, "Link Behavior Under Error" is associated with the output port, and is comprised of control symbols that are error-free and indicate that the attached input port has detected a transmission error or some other unusual situation has occurred. An example of this situation is indicated by the receipt of a packet-not-accepted control symbol. The state machine for the output port is shown in Figure C-4.

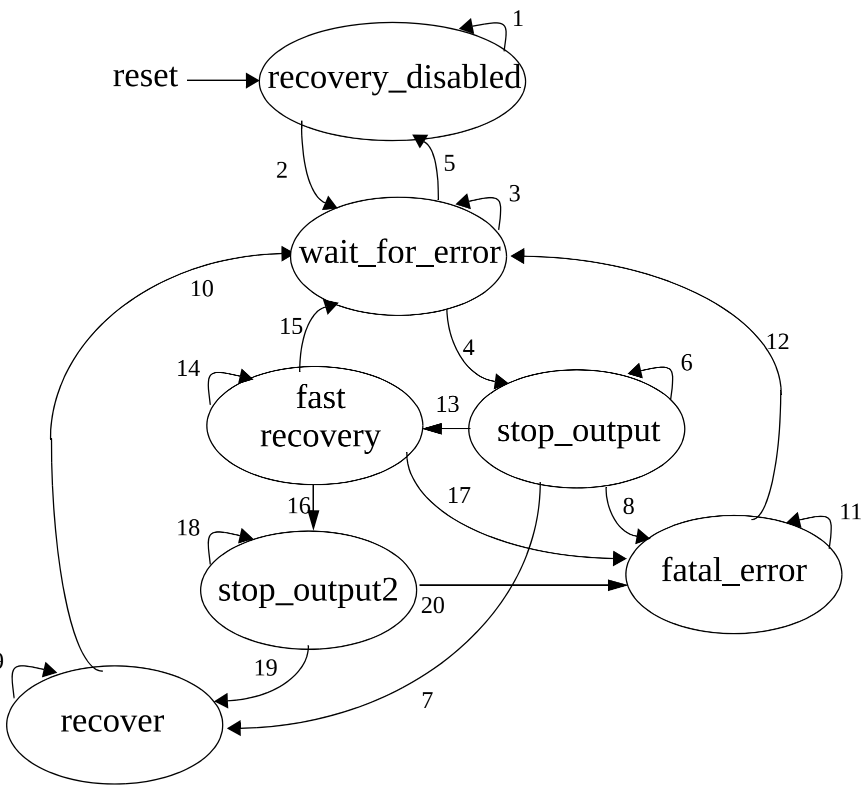


Figure C-4. Output Port Error Recovery State Machine

Table C-4 describes the state transition arcs for Figure C-4. The states referenced in the comments in quotes are the RapidIO LP-Serial defined status states, not states in this state machine.

Table C-4. 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 initialization sequence has been completed, and may be controlled through other mechanisms as well, such as a software enable bit.
2	recovery_disabled	wait_for_error	Error recovery is enabled.	
3	wait_for_error	wait_for_error	Remain in this state until a recoverable error is detected.	Detected errors and the level of coverage is implementation dependent.
4	wait_for_error	stop_output	A recoverable error has been detected.	An input port associated error will not cause this transition, only an output port associated error.
5	wait_for_error	recovery_disabled	Error recovery is disabled.	

Table C-4. 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 Normal” state, set the “Output Error-stopped” state, stop transmitting new packets, and send a link-request/port-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/port-status in order to be re-enabled to receive packets. An implementation may wish to timeout several times before regarding a timeout as fatal using a threshold counter or some other mechanism.
7	stop_output	recover	The link-response is received and returned an outstanding ackID value	An outstanding ackID is a value sent out on a packet that has not been acknowledged yet. In the case where no ackID is outstanding the returned ackID value shall match the next expected/next assigned ackID value, indicating that the devices are synchronized. Recovery is possible, so follow recovery procedure.
8	stop_output	fatal_error	The link-response is received and returned an ackID value that is not outstanding, or timed out waiting for the link-response.	Recovery is not possible, so start error shutdown procedure.
9	recover	recover	Remain in this state until the internal recovery procedure is completed.	The packet sent with the ackID value returned in the link-response and all subsequent packets shall be retransmitted. All packets transmitted with ackID values preceding the returned value were received by the attached device, so they are treated as if packet-accepted control symbols have been received for them. Output port state machines and the outstanding ackID scoreboard shall be updated with this information, then clear the “Output Error-stopped” state and set the ‘Port Normal” state to restart the output port.
10	recover	wait_for_error	The internal recovery procedure is complete.	retransmission (if any was necessary) has started, so return to the wait_for_error state to wait for the next error.
11	fatal_error	fatal_error	Remain in this state until error shutdown procedure is completed.	Clear the “Output Error-stopped” state, set the “Port Error” state, and signal a system error.

Table C-4. Output Port Error Recovery State Machine Transition Table (Continued)

Arc	Current State	Next state	Cause	Comments
12	fatal_error	wait_for_error	Error shutdown procedure is complete.	Return to the wait_for_error state.
13	stop_output	fast_recovery	Port has stopped transmitting new packets, link-request/port-status has been transmitted, cause of error was receipt of a packet-not-accepted with valid ackID status field, and Error Recovery with PNA Ackid Enabled is set.	This transition cannot be taken after transition 16 has been taken and before returning to wait_for_error.
14	fast_recovery	fast_recovery	Remain in this state until error recovery has completed.	The packet sent with the ackID value received in the Packet Not Accepted and all subsequent packets shall be retransmitted. All packets transmitted with ackID values preceding the returned value were received by the attached device, so they are treated as if packet-accepted control symbols have been received for them. Output port state machines and the outstanding ackID scoreboard shall be updated with this information, then clear the “Output Error-stopped” state and set the ‘Port Normal” state to restart the output port.
15	fast_recovery	wait_for_error	Link-response is received	retransmission (if any was necessary) has started, so return to the wait_for_error state to wait for the next error.
16	fast_recovery	stop_output2	Packet Accept, Packet Retry, or Paclet Not Accepted is recevied, or a recoverable error has been detected.	An input port associated error will not cause this transition, only an output port associated error.
17	fast_recovery	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.
18	stop_output2	stop_output2	Remain in this state until an exit condition occurs.	Clear the “Port Normal” state, set the “Output Error-stopped” state, stop transmitting new packets, and send a link-request/port-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/port-status in order to be re-enabled to receive packets. An implementation may wish to timeout several times before regarding a timeout as fatal using a threshold counter or some other mechanism.

Table C-4. Output Port Error Recovery State Machine Transition Table (Continued)

Arc	Current State	Next state	Cause	Comments
19	stop_output2	recover	The link-response is received and returned an outstanding ackID value	An outstanding ackID is a value sent out on a packet that has not been acknowledged yet. In the case where no ackID is outstanding the returned ackID value shall match the next expected/next assigned ackID value, indicating that the devices are synchronized. Recovery is possible, so follow recovery procedure.
20	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.

C.3.3 Changes in Error Recovery Behavior for CT

The basic states, as previously described, apply to the overall port. Each VC must carry some independent state:

Packets in a transmitter's VC queue are: pending transmission, sent-pending acknowledgement, or acknowledged (and subsequently removed from the queue). RT and CT VCs keep this same information, but behave slightly differently on error recovery. In RT queues, packets sent-pending acknowledgment, are returned to the pending transmission state. Packets pending acknowledgment in CT queues are moved to the acknowledged state. In this way, the sent packets in the CT queue are not resent.

(Note that it may not be necessary to keep the actual packet in a CT VC, only track the needed acknowledges to keep the credit balance for transmitter flow control accurate.)

Annex D Critical Resource Performance Limits (Informative)

D.1 IDLE1 and IDLE2

The RapidIO LP-Serial Physical Layer is intended for use over links whose length ranges from centimeters to tens of meters. The shortest length links will almost certainly use copper printed circuit board traces. The longer lengths will usually require the use of fiber optics (optical fiber and electro-optical converters) to overcome the high frequency losses of long copper printed circuit board traces or cable. Repeaters and/or retimers may also be inserted into longer length links. The longer lengths will also have significant propagation delay which can degrade the usable bandwidth of a link.

The serial protocol is a handshake protocol. Each packet transmitted by a port is assigned an ID (the ackID) and a copy of the packet is retained by the port in a holding buffer until the packet is accepted by the port’s link partner. The number of packets that a port can transmit without acknowledgment is limited to the lesser of the number of distinct ackIDs and the number of buffers available to hold unacknowledged packets. Which ever is the limiting resource, ackIDs or holding buffers, will be called the “critical resource”. The number of ackIDs is limited by the number of bits provided to define them: 5 bits for Control Symbol 24, 6 bits for Control Symbol 48, and 12 bits for Control Symbol 64. The number of holding buffers may be more constrained by the size of memory provided for this purpose.

The concern is the time between the assignment of a critical resource to a packet and the release of that resource as a consequence of the packet being accepted by the link partner. Call this time the resource_release_delay. When the resource_release_delay is less than the time it takes to transmit a number of packets equal to the number of distinct critical resource elements, there is no degradation of link performance. When the resource_release_delay is greater than the time it takes to transmit a number of packets equal to the number of distinct critical resource elements, the transmitter may have to stall from time to time waiting for a free critical resource. This will degraded the usable link bandwidth. The onset of degradation will depend on the average length of transmitted packets and the physical length of the link as reflected in the resource_release_delay.

The following example provides some idea of the impact on link performance of the interaction between link length and a critical resource for the case of operation with

IDLE1 or IDLE2. For purposes of this example, the following assumptions are made.

1. The link is a 4 lane (4x) link with 8b/10b encoding for transmission and both directions of the link have the same number of active lanes.
2. The link uses optical fiber and electro-optical transceivers to allow link lengths of tens of meters. The fiber is assumed to be single mode with a refraction index of 1.46. This may vary with the specific fiber, and those with higher indexes will be more constraining to cable length.
3. The width of the data path within the port is 4 bytes, equivalent to 1 byte per lane. The widths of the FIFOs that feed the lane transmitters and that are fed by the receivers have a width in bytes equal to the number of serial lanes. For the higher data rates, it may be advisable to increase this width to support a lower clock speed.
4. The data path and logic within the port run at a clock rate equal to the aggregate unidirectional data rate of the link divided by 32, based on the defined data path width. This is referred to as the logic clock. One cycle of this clock is referred to a one logic clock cycle. (If the aggregate unidirectional baud rate of the link was used to compute the logic clock, the baud rate would be divided by 40. With 8b/10b encoding, the baud rate is 1.25 times the data rate.) To support the higher link baud rates without increasing the internal clock rate, it may be advisable to divide by 64 vs. 32 and use wider internal data buses. Doing so will increase latency, but allow for longer cable lengths as the cable delay remains the same.
5. The minimum length packet header is used, reflecting Dev8 source/destination IDs and 34-bit addressing. Write request packets have a length of 12 bytes including CRC for packets with a payload of <80 bytes plus a payload containing an integer multiple of 8 bytes. Read request packets also have a length of 12 bytes. Read response packets have a length of 8 bytes reflecting Dev8 and independent of addressing and including CRC for a payload length of <80 bytes, plus a payload containing an integer multiple of 8 bytes.
6. The beginning and end of each packet is delimited by a control symbol. A single control symbol may delimit both the end of one packet and the beginning of the next packet. The length of the delimited control symbol changes between IDLE1 and IDLE2 from 4 to 8 bytes.
7. Packet acknowledgments are carried in packet delimiter control symbols when ever possible to achieve the efficiency provided by the dual stype control symbol. This implies that a packet acknowledgment must wait for an end-of-packet control symbol if packet transmission is in progress when the packet acknowledgment becomes available.

8. The logic cycle represents circuitry in the 8b/10b encoding and decoding stages of the transmit/receive pipelines where one 8b/10b code word is decoded per logic clock cycle.
9. Logic delays as shown are not based on any known physical implementation and should be adjusted by the user as needed to reflect a given solution. It is recommended to work with the suppliers of RapidIO products for assistance. Composite delay data may be provided as opposed to individual values for the delay elements.
10. Optical fiber delay is a function of the speed of light “c” and the refractive index of the chosen cable medium.

The logic and propagation delay in the packet transmission direction is comprised of the following components.

Table D-1. Packet Transmission Delay Components

Delay Element	Time required	Comments
Generate non-CRC portion of delimited start-of-packet control symbol (critical resource is available)	1 logic clock cycle	
Generate the control symbol CRC with link width = 4	1 logic clock cycle	
8b/10b encode the first N bytes of the delimited control symbol	1 logic clock cycle	
Serialize the first N code-groups of the encoded delimited control symbol	1 logic clock cycle	
Output register and output driver delay	2 ns	
PCB copper and electro-optical transmitter delay	2 ns	
Optical fiber delay	fiber_length (meters)/0.685c	Assumes refraction index = 1.46
Electro-optical receiver and pcb copper delay	2 ns	
Receiver delay	2 ns	
Deserialize the first N code-groups of the delimited control symbol	1 logic clock cycle	
8b/10b decode the delimited control symbol	(control symbol length/link width) logic clock cycles	1 cycle for IDLE1, 2 cycles for IDLE2
8b/10b decode average length packet	(average packet length/link width) logic clock cycles	
8b/10b decode packet terminating delimited control symbol	(control symbol length/link width) logic clock cycles	1 cycle for IDLE1, 2 cycles for IDLE2
Check control symbol CRC	1 logic clock cycle	
Make packet acceptance decision	1 logic clock cycle	

The logic and propagation delay in the packet acknowledgment direction is comprised of the following.

Table D-2. Packet Acknowledgment Delay Components

Delay element	Time required	Comments
Average wait for the end of the current packet	(1/2 of the packet transmit time)	Function of packet length
Generate non-CRC portion of the delimited acknowledgement control symbol	1 logic clock cycle	
Generate the control symbol CRC (This cycle is needed for CRC generation if the active link width N in lanes is equal to or greater than the length in bytes of the delimited control symbol. Otherwise, the CRC can be generated while the first N bytes of the delimited control symbol are being 8b/10b encoded)	1 logic clock cycle	
8b/10b encode the first N bytes of the delimited control symbol	1 logic clock cycle	
Serialize the first N code-groups of the encoded delimited control symbol	1 logic clock cycle	
Output register and output driver delay	2 ns	
PCB copper and electro-optical transmitter delay	2 ns	
Optical fiber delay	fiber_length (meters)/0.685c	Assumes refraction index = 1.46
Electro-optical receiver and pcb copper delay	2 ns	
Receiver delay	2 ns	
Deserialize the first N codewords of the delimited control symbol	1 logic clock cycle	
8b/10b decode the delimited control symbol	(control symbol length/link width) logic clock cycles	1 cycle for IDLE1, 2 cycles for IDLE2
Check control symbol CRC	1 logic clock cycle	
Make decision to free critical resource	1 logic clock cycle	

The packet times in the above tables depend on packet length which in turn depends on packet type and payload size. Since packet traffic will typically involve a mixture of packet types and payload sizes, the traffic in each direction will be assumed to contain an equal number of read, write and response packets and average payloads of 8, 32, 64 and 128 bytes.

The number of logic clock cycles required to transmit or receive a packet is given in the following table as a function of packet type and payload size.

Table D-3. Packet Delays

Packet Type	Packet Header bytes	CRC and padding bytes	Data Payload bytes	Transmit/Receive Time logic clock cycles (IDLE1)	Transmit/Receive Time logic clock cycles (IDLE2)
Read	12	2	0	5	6
Response	8	2	8	6	7
		2	32	10	11
		2	64	18	19
	10	6	128	37	38

Table D-3. Packet Delays

Packet Type	Packet Header bytes	CRC and padding bytes	Data Payload bytes	Transmit/Receive Time logic clock cycles (IDLE1)	Transmit/Receive Time logic clock cycles (IDLE2)
Write	12	2	8	7	8
		2	32	10	11
		2	64	18	19
	14	6	128	38	39

Using the above table and the assumed equal number of read, write and response packets where the payload size in write and response packets is identical, the average number of logic clock cycles to transmit or received a packet is 5, 9, 14.3 and 25.7 respectively for packet payloads of 8, 32, 64 and 128 bytes. The average wait for the completion of a packet being transmitted is assumed to be 1/2 the transmit time.

The following table gives the maximum length of the optical fiber before the packet transmission rate becomes limited by the critical resource for a 4x link operating at unidirectional data rates of 4.0, 8.0, 10.0, 16.0 and 20.0 Gb/s where the highest rate makes use of IDLE2 control words and the four lower rates use IDLE1 control words. An assumption is made that the number of ackIDs and buffers is the same. Note that regardless of rate, the internal data path is assumed to be 32 bits wide. An alternative might be to increase internal data width to 64 bits at the higher baud rates.

Table D-4. Maximum Transmission Distances

Number of Critical Resources Available (ackIDs or buffers)	Data Payload (bytes)	Maximum Fiber Length Before Critical Resource Limited (meters)				
		4.0 Gb/s link (IDLE1)	8.0 Gb/s link (IDLE1)	10.0 Gb/s link (IDLE1)	16.0 Gb/s link (IDLE1)	10.0 Gb/s link (IDLE2)
4	8	-	-	-	-	-
	32	3.7	1.0	0.5	-	-
	64	14.7	6.5	4.9	2.4	2.0
	128	37.9	18.1	14.8	8.3	6.7
8	8	11.9	5.1	3.8	1.8	2.1
	32	33.3	15.8	12.3	7.1	6.4
	64	61.7	30.1	23.7	14.2	21.1
	128	122.2	60.3	47.9	29.3	24.2
16	8	44.8	21.6	16.9	10.0	10.0
	32	92.4	45.4	36.0	21.9	19.6
	64	155.9	77.1	61.4	37.8	32.3
	128	290.9	144.6	115.4	71.5	59.3

Table D-4. Maximum Transmission Distances

Number of Critical Resources Available (ackIDs or buffers)	Data Payload (bytes)	Maximum Fiber Length Before Critical Resource Limited (meters)				
		4.0 Gb/s link (IDLE1)	8.0 Gb/s link (IDLE1)	10.0 Gb/s link (IDLE1)	16.0 Gb/s link (IDLE1)	10.0 Gb/s link (IDLE2)
24	8	77.6	38.0	30.1	18.2	17.9
	32	151.5	75.0	59.6	36.7	32.7
	64	250.1	124.2	99.1	61.3	52.4
	128	459.5	229.0	182.8	113.7	94.3
31 (max for IDLE1 - limited to 2^N-1)	8	106.4	52.4	41.6	25.4	24.8
	32	203.3	100.8	80.3	49.6	44.2
	64	332.5	165.4	132.0	81.9	70.0
	128	607.1	302.7	241.9	150.6	125.0
63 (max for IDLE2 - limited to 2^N-1)	8					56.3
	32					96.8
	64					150.6
	128					265.1

When the information above is combined together, a single spreadsheet computation can be developed to solve for fiber length. A spreadsheet with this capability is located in the members only section of the RapidIO.org web site.

D.2 IDLE3

IDLE3 links are based on a different coding mechanism than IDLE1 and IDLE2, using 64b/67b encoding. Separate tables are required to support IDLE3, which also includes additional features.

The following example provides some idea of the impact on link performance of the interaction between link length and a critical resource for the case of a IDLE3 link. For purposes of this example, the following assumptions are made.

1. The link is a 4 lane (4x) link with 64b/67b encoding for transmission and the both directions of the link have the same number of active lanes. As a result, the example does not consider the effects of an asymmetric link.
2. The link uses optical fiber and electro-optical transceivers to allow link lengths of tens of meters. The fiber is assumed to be single mode with a refraction index of 1.46.
3. The width of the data path within the port is 16 bytes, equivalent to 4 bytes per lane. This was used to allow for a lower speed clock within the protocol logic.

4. The data path and logic within the port run at a clock rate equal to the aggregate unidirectional data rate of the link divided by 128, based on the defined data path width. This is referred to as the logic clock. One cycle of this clock is referred to a one logic clock cycle.
5. The minimum length packet header is used, reflecting Dev8 source/destination IDs and 34-bit addressing. Write request packets have a length of 12 bytes including CRC for packets with a payload of <80 bytes plus a payload containing an integer multiple of 8 bytes. Read request packets also have a length of 12 bytes. Read response packets have a length of 8 bytes reflecting Dev8 and independent of addressing and including CRC for a payload length of <80 bytes, plus a payload containing an integer multiple of 8 bytes.
6. The beginning and end of each packet is delimited by a control symbol. A single control symbol may delimit both the end of one packet and the beginning of the next packet. The length of the delimited control symbol is 8 bytes for Baud Rate Class 3.
7. Packet acknowledgments are carried in packet delimiter control symbols whenever possible to achieve the efficiency provided by the dual stype control symbol. This implies that a packet acknowledgment must wait for an end-of-packet control symbol if packet transmission is in progress when the packet acknowledgment becomes available.
8. The multiple packet acknowledgement capability required for Baud Rate Class 3 is not included in the example or the calculations. It is assumed that the signal allowing this is de-asserted.
9. Logic delays as shown are not based on any known physical implementation and should be adjusted by the user as needed to reflect a given solution. It is recommended to work with the suppliers of RapidIO products for assistance. Composite delay data may be provided as opposed to individual values for the delay elements.
10. Optical fiber delay is a function of the speed of light “c” and the refractive index of the chosen cable medium.

The logic and propagation delay for IDLE3 in the packet transmission direction is comprised of the following components. The logic cycle represents circuitry in the 64b/67b encoding and decoding stages of the transmit/receive pipelines where one 64b/67b codeword is decoded per logic clock cycle.

Table D-5. IDLE3 Packet Transmission Delay Components

Delay Element	Time required	Comments
Generate non-CRC portion of delimited start-of-packet control symbol (critical resource is available)	1 logic clock cycle	
Generate the control symbol CRC (This cycle is needed for CRC generation if the active link width N in lanes is equal to or greater than the length in bytes of the delimited control symbol. Otherwise, the CRC can be generated while the first N bytes of the delimited control symbol are being 64b/67b encoded.)	1 logic clock cycle	
Generate the link CRC-32, which is unique to IDLE3 (This cycle is needed for CRC generation if the active link width N in lanes is equal to or greater than the length in bytes of the delimited control symbol. Otherwise, the CRC can be generated while the first N bytes of the delimited control symbol are being 64b/67b encoded.)	1 logic clock cycle	
64b/67b encode the first N bytes of the delimited control symbol	1 logic clock cycle	
Serialize the first N codewords of the encoded delimited control symbol	1 logic clock cycle	
Output register and output driver delay	2 ns	
PCB copper and electro-optical transmitter delay	2 ns	
Optical fiber delay	fiber_length (meters)/0.685c	Assumes refraction index = 1.46
Electro-optical receiver and pcb copper delay	2 ns	
Receiver delay (need 1 logic clock cycle to translate between the SerDes data width and the 64b/67b encoded width)	1 logic clock cycle	
Deserialize the first N codewords of the delimited control symbol	1 logic clock cycle	
64b/67b decode the delimited control symbol	(control symbol length/link width) logic clock cycles	
64b/67bB decode average length packet	(average packet length/link width) logic clock cycles	
64b/67b decode packet terminating delimited control symbol	(control symbol length/link width) logic clock cycles	
Check control symbol CRC	1 logic clock cycle	
Make packet acceptance decision	1 logic clock cycle	

The logic and propagation delay for IDLE3 in the packet acknowledgment direction is comprised of the following components.

Table D-6. IDLE3 Packet Acknowledgment Delay Components

Delay element	Time required	Comments
Wait for multiple other packets, per the multiple packet acknowledge capability	Average packet length * number of packets waited for	This is a new standard capability, defaulted to zero
Average wait for the end of the current packet	(1/2 of the packet transmit time)	Function of packet length
Generate non-CRC portion of the delimited acknowledgement control symbol	1 logic clock cycle	

Table D-6. IDLE3 Packet Acknowledgment Delay Components

Delay element	Time required	Comments
Generate the control symbol CRC (This cycle is needed for CRC generation if the active link width N in lanes is equal to or greater than the length in bytes of the delimited control symbol. Otherwise, the CRC can be generated while the first N bytes of the delimited control symbol are being 64b/67b encoded)	1 logic clock cycle	
64b/67b encode the first N bytes of the delimited control symbol	1 logic clock cycle	
Serialize the first N codewords of the encoded delimited control symbol	1 logic clock cycle	
Output register and output driver delay	2 ns	
PCB copper and electro-optical transmitter delay	2 ns	
Optical fiber delay	fiber_length (meters)/0.685c	Assumes refraction index = 1.46
Electro-optical receiver and pcb copper delay	2 ns	
Receiver delay (need 1 logic clock cycle to translate between the SerDes data width and the 64b/67b encoded width)	1 logic clock cycle	
Deserialize the first N codewords of the delimited control symbol	1 logic clock cycle	
64b/67b decode the delimited control symbol	(control symbol length/link width) logic clock cycles	
Check control symbol CRC	1 logic clock cycle	
Make decision to free critical resource	1 logic clock cycle	

The packet times in the above tables depend on packet length which in turn depends on packet type and payload size. Since packet traffic will typically involve a mixture of packet types and payload sizes, the traffic in each direction will be assumed to contain an equal number of read, write and response packets and average payloads of 8, 32, 64, and 128 bytes. The number of logic clock cycles required to transmit or receive a packet is given in the following table as a function of packet type and payload size.

Table D-7. IDLE3 Packet Delays

Packet Type	Packet Header bytes	CRC and padding bytes	Data Payload bytes	Transmit/Receive Time logic clock cycles
Read	12	4	0	6
Response	8	4	8	7
		4	32	11
		4	64	19
	10	4	128	38
Write	12	4	8	8
		4	32	11
		4	64	19
	14	4	128	39

Using the above table and the assumed equal number of read, write and response packets where the payload size in write and response packets is identical, the average number of logic clock cycles to transmit or received a Class 3 packet is 7.3, 11.3, 16.7, and 28 respectively for packet payloads of 8, 32, 64, and 128 bytes. The average wait for the completion of a packet being transmitted is assumed to be 1/2 the transmit time.

The following table gives the maximum length of the optical fiber before the packet transmission rate becomes limited by the critical resource for a 4x link operating at a unidirectional data rate of 40.0 Gb/s. An assumption is made that the number of ackIDs and buffers is the same. Note that for Baud Rate Class 3, the internal data path is assumed to be 128 bits wide. Because of the advanced technologies expected for implementation of Baud Rate Class 3 interfaces and the very large number of supported ackIDs, a minimum of 16 ackIDs is shown in the table.

Table D-8. IDLE3 Maximum Transmission Distances

Number of Critical Resources Available (ackIDs or buffers)	Data Payload (bytes)	Maximum Fiber Length Before Critical Resource Limited (meters)
		40.0 Gb/s link (IDLE3)
16	8	27.2
	32	46.6
	64	72.4
	128	127.2
31 (max for IDLE1 - limited to 2^N-1)	8	63.9
	32	103.3
	64	155.7
	128	267.3
48	8	105.5
	32	167.5
	64	250.2
	128	426.0
63 (max for IDLE2 - limited to 2^N-1)	8	142.2
	32	224.2
	64	333.6
	128	566.1
80	8	183.8
	32	288.5
	64	428.1
	128	724.8

When the information above is combined together, a single spreadsheet computation can be developed to solve for fiber length. A spreadsheet with this capability is located in the members only section of the RapidIO.org web site. Different work sheets are required for the IDLE3 calculation than for IDLE1 and IDLE2.

Annex E Manufacturability and Testability (Informative)

It is not possible in many cases for assembly vendors to verify the integrity of soldered connections between components and the printed circuit boards to which they are attached. Alternative methods to direct probing are needed to insure high yields for printed circuit assemblies which include LP-Serial RapidIO devices.

It is recommended that component vendors support IEEE Std. 1149.6 (commonly known as “AC-JTAG”) on all connections to LP-Serial RapidIO links. (Note: IEEE Std. 1149.6 is needed, in addition to IEEE Std. 1149.1, due the fact that RapidIO LP-Serial lanes are AC-coupled.) This provides boundary scan capability on all TD, TDN, RD, and RDN pins on a component which supports one or more LP-Serial RapidIO ports.

The IEEE Std. 1149.6 is available from the IEEE.

Annex F Multiple Port Configuration Example (Informative)

F.1 Introduction

This appendix contains flow-chart descriptions that illustrates the Port-Width negotiation process described in Chapter 4, “8b/10b PCS and PMA Layers. They are included as examples and are believed to be correct, however, actual implementations and system design should not use the examples directly.

F.2 System with Different Port Width Capabilities

In a high-performance system, a high-bandwidth switch processing element is often used to aggregate traffic; while the connecting agents can be ones of lower bandwidth. Under this circumstance, the switch processing element has to identify the discrepancy of port-widths between link partners and set up accordingly. Figure shows a typical system with a switch processing element connected between the System Host and two connecting Agents. The system is set up as follows:

- System Host is connected to Switch A
- Switch A has a 8x port which is capable of multiple port configuration.
- Agent B has a 4x port connected to Switch A lanes 4-7
- Agent C has a 2x port connected to Switch A lanes 0-3.

The following example is used to illustrate the negotiation that will take place between Switch A and Agents B and C. It is assumed that the System Host and the Switch A have already established error-free communication.

1. By default, the 8x-port of Switch A looks for an 8x connection but fails to come up with its link partner; thus, falling to 1x mode on lane 0 or 2.
2. Agent B fails to establish 4x link with Switch A. It tries to fall back to 1x mode on its lane 0 or 2 but still fails. Its 4x port has failed.
3. Agent C fails to establish a 2x link with Switch A. When it tries to fall back to 1x mode, it succeeds in lane 0 and re-establish communication with Switch A on its lane 0.
4. System Host reads through the established 1x link between Switch A and Agent C. From the Vendor Port-Width CAR of Agent C, System Host discovers that Agent C can support 2x mode.

5. System Host checks Switch A for its support of 2x mode on its lower quad-link.
6. System Host writes to the Port Width Override CSR to force both Switch A lower quad-link.
7. System Host puts Agent C back to 2x mode.
8. A 2x-link is established between Switch A and Agent C.
9. System Host discovers from Vendor Port-Width CAR in Agent B (not through Switch A because the link was not established yet) that Agent B supports 4x mode. It also discovers that Switch A supports multiple port configuration (from its Vendor Specific registers) and its extra port is available (Vendor Port-Width CAR).[1]
10. System Host configures the new port on the upper-quad link of Switch A.
11. Agent B now recognizes a 4x link partner.
12. A 4x link is now established between Switch A and Agent B.

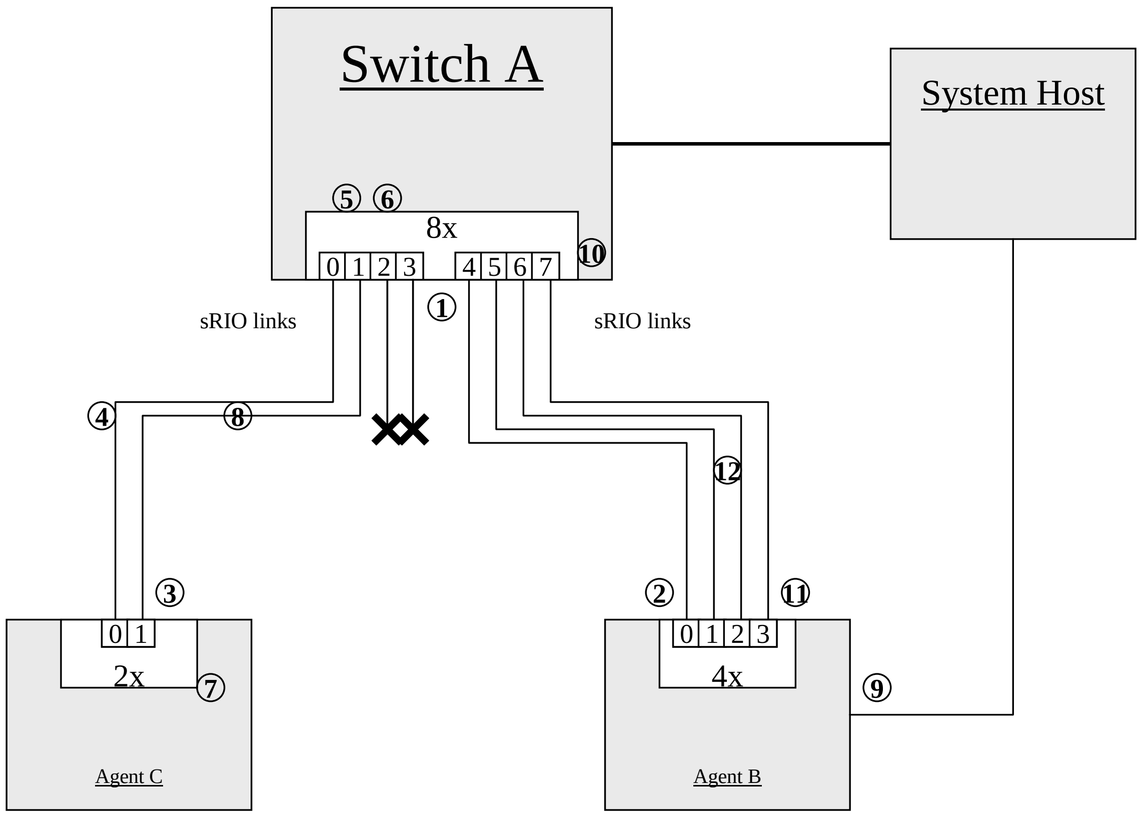


Figure F-1. Example system with asymmetric port-width capabilities

[1]Steps 9 to 12 are optional. Switch A is not required to support multiple-port configuration to be compliant.

Annex G MECS Time Synchronization (Informative)

G.1 Introduction

This annex provides examples and discussion that illustrate a possible approach to configuration and operation of (S)MECS Time Synchronization. The examples are considered to be correct; however, the requirements for individual system designs may differ from the assumptions for these examples.

G.2 Detection of Missing MECS

It is possible for MECS or SMECS to be corrupted during transmission. RapidIO standard error recovery procedures do not include retransmission of MECS or SMECS. If an MECS or SMECS is lost, the loss of the “tick” associated with the (S)MECS could cause timestamp generators in multiple nodes to become unsynchronized with the rest of the system.

It should be possible to bound the jitter in the propagation of MECS and SMECS through the fabric. Typically, the jitter should be quite small in comparison to the time of a particular “tick”. One approach to detecting the loss of an (S)MECS is to assume an (S)MECS should have been received if the current timestamp generator value has surpassed the MECS Next Timestamp Value by a small binary fraction of the tick interval.

Reaction to the loss of an MECS should be limited to updating the MECS Next Timestamp Value. Implementation specific events related to detection of a lost (S)MECS may be generated and reported.

G.3 MECS and SMECS Redundant Operation

When redundant sources for time synchronization exist in the system, ideally it would be possible to fail over from one source to another while allowing the timestamp generator to smoothly continue increasing. For single switch systems, MECS and SMECS redundant operation is a simple problem, as MECS and SMECS are propagated to all endpoints with the same latency. It is therefore possible to closely synchronize the generation of MECS and SMECS. The fail over operation is limited to selecting whether MECS or SMECS will drive the timestamp generator in each slave.

In more complex fabrics, the MECS and SMECS sources may be connected to different switches. As a result, the propagation delay from the MECS source to a particular node may differ significantly from the propagation delay from the SMECS source, so it is not possible to closely synchronize the arrival of the MECS and SMECS at all nodes. The fail over operation is more complex for these fabrics.

For complex fabrics, two timestamp generators, one driven by MECS and the other by SMECS, could be maintained. The most intuitive implementation would be to incorporate two copies of the Timestamp Register Extension Block, both of which support operation with MECS or SMECS.

During system initialization, the MECS master would set time on all nodes, including the SMECS master. The SMECS master could then set up the SMECS time on all nodes. Since the SMECS original time is the MECS time, the SMECS time on all nodes should remain closely synchronized to the MECS time. Note that the MECS master must also support an SMECS timestamp generator, and would act as an SMECS slave. The fail over operation becomes a matter of selecting which timestamp generator to use as the source of system time. It is also possible to “repair” the failed timestamp generator by locally resynchronizing the failed timestamp generator to the operable timestamp generator, and then changing the failed timestamp generator to use the operable (S)MECS source.

G.4 Detection of (S)MECS Source Failure

If two consecutive MECS or SMECS control symbols are not received, the (S)MECS slave can be confident that it is no longer connected to that source and/or the source has failed.

The slave can then initiate a fail over, if necessary, as described in Section G.3, “MECS and SMECS Redundant Operation”.

Glossary of Terms and Abbreviations

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

A

AC Coupling. A method of connecting two devices together that does not pass DC.

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

ANSI. American National Standards Institute.

B

Big-endian. A byte-ordering method in memory where the address n of a word corresponds to the most significant byte. In an addressed memory word, the bytes are ordered (left to right) 0, 1, 2, 3, with 0 being the most significant byte.

Bridge. A processing element that connects one computer bus to another, allowing a processing element on one bus to access an processing element on the other.

Byte. An 8-bit unit of information. Each bit of a byte has the value 0 or 1.

C

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

Code-group. A 10-bit entity produced by the 8b/10b encoding process and the input to the 8b/10b decoding process.

Codeword. A 67-bit entity produced by the 64b/67b encoding process and the input to the 64b/67b decoding process.

Command and status registers (CSRs). A set of registers that allow a processing element to control and determine the status of another processing element’s internal hardware.

Continuous Transmission (CT). A mode of packet transmission that allows some packet loss to minimize latency by not retransmitting packets.

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 a processing element connected to the RapidIO interconnect.

Direct Memory Access (DMA). A process element that can independently read and write system memory.

Distributed memory. System memory that is distributed throughout the system, as opposed to being centrally located.

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

E

EMI. Electromagnetic Interference.

End point. A processing element which is the source or destination of transactions through a RapidIO fabric.

End point device. A processing element which contains end point functionality.

End point free device. A processing element which does not contain end point functionality.

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

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

F

Fabric. A series of interconnected switch devices, typically used in reference to a switch fabric.

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

FIFO. First in, first out.

Full-duplex. Data can be transmitted in both directions between connected processing elements at the same time.

G

Globally shared memory (GSM). Cache coherent system memory that can be shared between multiple processors in a system.

H

Half-word. A two byte or 16-bit quantity, aligned on two byte boundaries.

Header. Typically the first few bytes of a packet, containing control information.

I

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

I/O. Input-output.

IP. Intellectual Property

ITU. International Telecommunication Union.

L

Little-endian. A byte-ordering method in memory where the address n of a word corresponds to the least significant byte. In an addressed memory word, the bytes are ordered (left to right) 3, 2, 1, 0, with 3 being the most significant byte.

Local memory. Memory associated with the processing element in question.

LP. Link Protocol

LSB. Least significant byte.

LVDS. Low voltage differential signaling.

M

Message passing. An application programming model that allows processing elements to communicate through special hardware instead of through memory as with the globally shared memory programming model.

MSB. Most significant byte.

N

Non-coherent. A transaction that does not participate in any system globally shared memory cache coherence mechanism.

O

Operation. A set of transactions between end point devices in a RapidIO system (requests and associated responses) such as a read or a write.

P

Packet. A set of information transmitted between devices in a RapidIO system.

Payload. The user data embedded in the RapidIO packet.

PCB. Printed circuit board.

PCS. Physical Coding Sublayer.

PMA. Physical Media Attachment.

Port-write. An address-less write operation.

Priority. The relative importance of a transaction or packet; in most systems a higher priority transaction or packet will be serviced or transmitted before one of lower priority.

Processing Element (**PE**). A generic participant on the RapidIO interconnect that sends or receives RapidIO transactions, also called a device.

Processor. The logic circuitry that responds to and processes the basic instructions that drive a computer.

R

Receiver. The RapidIO interface input port on a processing element.

Reliable Transmission (RT). A mode of operation that guarantees packet delivery by retransmitting packets when an error occurs.

S

Sender. The RapidIO interface output port on a processing element.

Semaphore. A technique for coordinating activities in which multiple processing elements compete for the same resource.

Serializer. A device which converts parallel data (such as 8-bit data) to a single bit-wide datastream.

Source. The origin of a packet on the RapidIO interconnect, also referred to as an initiator.

SRAM. Static random access memory.

Switch. A multiple port processing element that directs a packet received on one of its input ports to one of its output ports.

T

Target. The termination point of a packet on the RapidIO interconnect, also referred to as a destination.

Transaction. A specific request or response packet transmitted between end point devices in a RapidIO system.

Transaction request flow. A sequence of transactions between two processing elements that have a required completion order at the destination processing element. There are no ordering requirements between transaction request flows.

W

Word. A four byte or 32 bit quantity, aligned on four byte boundaries.

Write port. Hardware within a processing element that is the target of a port-write operation.

RapidIO™ Interconnect Specification Part 7: System and Device Inter-operability Specification

3.2, 1/2016

RapidIO.org

Revision History

Revision	Description	Date
1.1	First public release.	04/06/2001
1.2	Technical changes: incorporate Rev. 1.1 errata rev. 1.1.1, errata 3	06/26/2002
1.3	Technical changes: incorporate Rev 1.2 errata 1 as applicable, the following errata showings: 004-05-00002.002 Converted to ISO-friendly templates	02/23/2005
2.0	Technical changes: errata showing 06-02-00001.005	06/14/2007
2.1	No technical changes.	07/09/2009
2.2	No technical changes.	05/05/2011
3.0	Changed RTA contact information. No technical changes	11/9/2013
3.1	Technical changes: Addition of Space device profiles.	09/18/2014
3.2	No technical changes.	01/28/2016

Table of Contents

Chapter 1 Overview

1.1 Introduction 11
1.2 Overview 11

Chapter 2 System Exploration and Initialization

2.1 Introduction 13
2.2 Boot code access 13
2.3 Exploration and initialization 15
2.3.1 Exploration and initialization rules 15
2.3.2 Exploration and initialization algorithm 16
2.3.3 Exploration and initialization example 16

Chapter 3 RapidIO Device Class Requirements

3.1 Introduction 21
3.2 Class Partitioning 22
3.2.1 Generic: All devices 22
3.2.1.1 General requirements 22
3.2.1.2 Operation support as target 23
3.2.1.3 Operation support as source 23
3.2.2 Class 1: Simple target device 24
3.2.2.1 General requirements 24
3.2.2.2 Operation support as target 24
3.2.2.3 Operation support as source 24
3.2.3 Class 2: Simple mastering device 24
3.2.3.1 General requirements 24
3.2.3.2 Operation support as target 24
3.2.3.3 Operation support as source 24
3.2.4 Class 3: Complex mastering device 25
3.2.4.1 General requirements 25
3.2.4.2 Operation support as target 25
3.2.4.3 Operation support as source 26
3.3 Space Device Definition 26
3.3.1 Basic Space Device Requirements 26
3.3.2 Enhanced Space Device Requirements 27
3.3.3 Space-10xN Device Requirements 28
3.3.4 Space Switch Device Requirements 28
3.3.5 Space Endpoint Device Requirements 30
3.3.6 Space Endpoint-E Device Requirements 31

Table of Contents

Chapter 4 PCI Considerations

4.1 Introduction.. 33
4.2 Address Map Considerations 34
4.3 Transaction Flow 35
4.3.1 PCI 2.2 Transaction Flow 35
4.3.2 PCI-X Transaction Flow... 38
4.4 RapidIO to PCI Transaction Mapping 39
4.5 Operation Ordering and Transaction Delivery 41
4.5.1 Operation Ordering 41
4.5.2 Transaction Delivery Ordering 42
4.5.3 PCI-X Relaxed Ordering Considerations 42
4.6 Interactions with Globally Shared Memory... 43
4.6.1 I/O Read Operation Details.. 46
4.6.1.1 Internal Request State Machine 46
4.6.1.2 Response State Machine 46
4.6.2 Data Cache Flush Operation Details.. 47
4.6.2.1 Internal Request State Machine 47
4.6.2.2 Response State Machine 47
4.7 Byte Lane and Byte Enable Usage 47
4.8 Error Management 47

Chapter 5 Globally Shared Memory Devices

5.1 Introduction.. 49
5.2 Processing Element Behavior 49
5.2.1 Processor-Memory Processing Element 50
5.2.1.1 I/O Read Operations 50
5.2.1.1.1 Response State Machine 50
5.2.1.1.2 External Request State Machine.. 51
5.2.2 Memory-only Processing Element.. 52
5.2.2.1 Read Operations... 52
5.2.2.1.1 Response State Machine 52
5.2.2.1.2 External Request State Machine.. 52
5.2.2.2 Instruction Read Operations 53
5.2.2.2.1 Response State Machine 53
5.2.2.2.2 External Request State Machine.. 54
5.2.2.3 Read for Ownership Operations 54
5.2.2.3.1 Response State Machine 54
5.2.2.3.2 External Request State Machine.. 55
5.2.2.4 Data Cache and Instruction Cache Invalidate Operations 56
5.2.2.4.1 Response State Machine 56
5.2.2.4.2 External Request State Machine.. 56
5.2.2.5 Castout Operations... 57
5.2.2.5.1 External Request State Machine.. 57
5.2.2.6 Data Cache Flush Operations 57

Table of Contents

5.2.2.6.1 Response State Machine 57
5.2.2.6.2 External Request State Machine 58
5.2.2.7 I/O Read Operations 59
5.2.2.7.1 Response State Machine 59
5.2.2.7.2 External Request State Machine 59
5.2.3 Processor-only Processing Element 60
5.2.3.1 Read Operations 60
5.2.3.1.1 Internal Request State Machine 60
5.2.3.1.2 Response State Machine 60
5.2.3.1.3 External Request State Machine 61
5.2.3.2 Instruction Read Operations 61
5.2.3.2.1 Internal Request State Machine 61
5.2.3.2.2 Response State Machine 61
5.2.3.2.3 External Request State Machine 62
5.2.3.3 Read for Ownership Operations 63
5.2.3.3.1 Internal Request State Machine 63
5.2.3.3.2 Response State Machine 63
5.2.3.3.3 External Request State Machine 63
5.2.3.4 Data Cache and Instruction Cache Invalidate Operations 64
5.2.3.4.1 Internal Request State Machine 64
5.2.3.4.2 Response State Machine 64
5.2.3.4.3 External Request State Machine 65
5.2.3.5 Castout Operations 65
5.2.3.5.1 Internal Request State Machine 65
5.2.3.5.2 Response State Machine 65
5.2.3.6 TLB Invalidate Entry, TLB Invalidate Entry Synchronize Operations 66
5.2.3.6.1 Internal Request State Machine 66
5.2.3.6.2 Response State Machine 66
5.2.3.6.3 External Request State Machine 66
5.2.3.7 Data Cache Flush Operations 66
5.2.3.7.1 Internal Request State Machine 66
5.2.3.7.2 Response State Machine 67
5.2.3.7.3 External Request State Machine 67
5.2.3.8 I/O Read Operations 67
5.2.3.8.1 External Request State Machine 68
5.2.4 I/O Processing Element 69
5.2.4.1 I/O Read Operations 69
5.2.4.1.1 Internal Request State Machine 69
5.2.4.1.2 Response State Machine 69
5.2.4.2 Data Cache Flush Operations 70
5.2.4.2.1 Internal Request State Machine 70
5.2.4.2.2 Response State Machine 70
5.2.5 Switch Processing Element 70
5.3 Transaction to Priority Mappings 70

Table of Contents

Blank page

List of Figures

2-1 Example system with boot ROM....................14
2-2 Automatically finding the boot ROM....................14
2-3 Example system....................16
2-4 Finding the adjacent device....................17
2-5 Finding the device on switch port 0....................18
2-6 Finding the device on switch port 1....................18
2-7 Finding the device on switch port 3....................19
2-8 Final initialized system state....................19
4-1 Example System with PCI and RapidIO....................33
4-2 Host segment PCI Memory Map Example....................34
4-3 AMT and Memory Mapping....................35
4-4 PCI Mastered Posted Write Transaction Flow Diagram....................36
4-5 PCI Mastered non-posted (delayed) Transaction Flow Diagram....................37
4-6 RapidIO Mastered Transaction....................38
4-7 PCI-X Mastered Split Response Transaction....................39
4-8 Traditional Non-coherent I/O Access Example....................43
4-9 Traditional Globally Coherent I/O Access Example....................44
4-10 RapidIO Locally Coherent I/O Access Example....................45

List of Figures

Blank Page

List of Tables

4-1 PCI 2.2 to RapidIO Transaction Mapping 39
4-2 PCI-X to RapidIO Transaction Mapping 40
4-3 Packet priority assignments for PCI ordering 42
4-4 Packet priority assignments for PCI-X ordering 43
5-1 Transaction to Priority Mapping 71

List of Tables

Blank Page

Chapter 1 Overview

1.1 Introduction

This chapter provides an overview of the *RapidIO Part 7: System and Device Inter-operability Specification* document. This document assumes that the reader is familiar with the RapidIO specifications, conventions, and terminology.

1.2 Overview

The RapidIO Architectural specifications set a framework to allow a wide variety of implementations. This document provides a standard set of device and system design solutions to provide for inter-operability.

Each chapter addresses a different design topic. This revision of the system and device inter-operability specification document covers the following issues:

Chapter 2, “System Exploration and Initialization”

Chapter 3, “RapidIO Device Class Requirements”

Chapter 4, “PCI Considerations”

Chapter 5, “Globally Shared Memory Devices”

Blank page

Chapter 2 System Exploration and Initialization

2.1 Introduction

There are several basic ways of exploring and initializing a RapidIO system. The simplest method is to somehow define the power-up state of the system components such that all devices have adequate knowledge of the rest of the system to communicate as needed. This is frequently accomplished by shifting initialization information into all of the devices in the machine at boot time from serial ROMs or similar devices. This method is most applicable for relatively static systems and systems where boot-up time is important. A second method, having processors explore and configure the system at boot time, requires more time but is much more flexible in order to support relatively fast changing plug-and-play or hot-swap systems. This document describes a simple form of this second method. A much more detailed multiple host exploration and configuration algorithm utilizing the same system reset requirements is specified in the *RapidIO Interconnect Specification Annex 1: Software/System Bring Up Specification*.

2.2 Boot code access

In most RapidIO applications system initialization requires software for exploring and initializing devices. This is typically done by a processor or set of processors in the system. The boot code for the processor(s) may reside in a ROM local to the processor(s) or on a remote RapidIO agent device. A method of accessing the boot code through an uninitialized system is required if the boot code is located on a remote RapidIO agent device.

After resetting, a processor typically vectors to a fixed address and issues a code fetch. The agent hardware between the processor and the RapidIO fabric is required to take this read request and map it automatically to a NREAD transaction. The transaction is also mapped to a dedicated device ID at the proper address offset to find the boot code. All devices between the processor and the agent device where the boot ROM resides shall default to a state that will route the NREAD transaction to the boot ROM device and route the response back to the processor. The device ID for the agent device where the boot ROM resides is device ID=0xFE (0x00FE for 16-bit device IDs). The processor default device IDs are assigned sequentially starting at 0x00 (0x0000 for 16-bit device IDs).

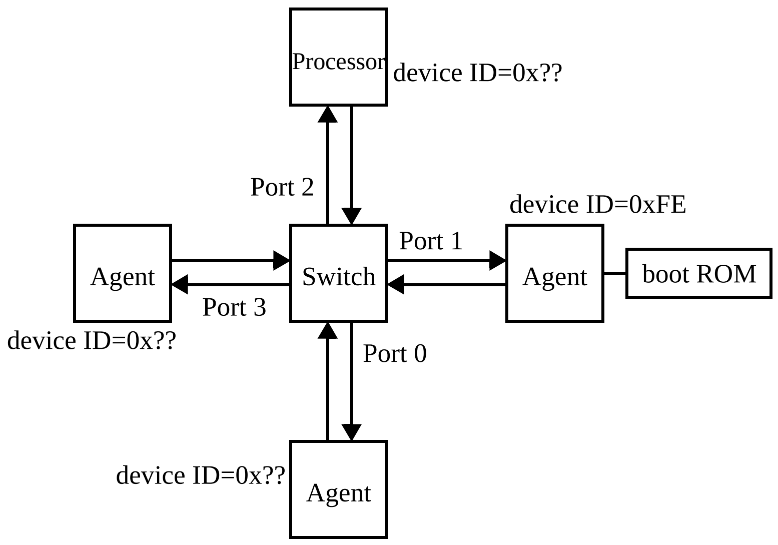


Figure 2-1. Example system with boot ROM

Figure 2-1 shows an example system with the boot ROM residing on an Agent device. The default routing state for the switch device between the processor and the agent shall allow all requests to device ID=0xFE to get to the agent device and all response packets to get from the agent device back to the processor. This means that the switch may also have to know the device ID that the processor will be using while fetching boot code (processor device IDs are assigned starting at 0x00 as described above). For the example in Figure 2-2, the system processor defaults to device ID=0x00, and the switch's default state routes device ID=0x00 to port 2.

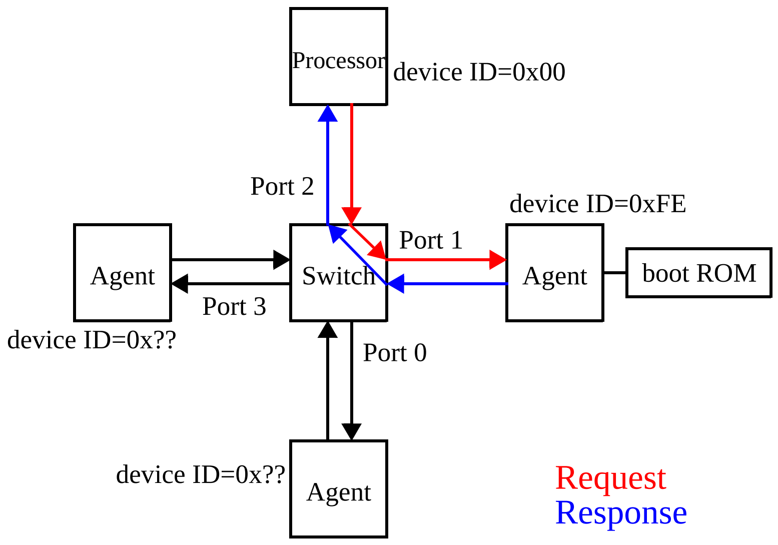


Figure 2-2. Automatically finding the boot ROM

Once the processor is able to begin running boot code, it can begin executing the exploration and initialization of the rest of the system.

2.3 Exploration and initialization

This example algorithm addresses the simple case of a system with a single processor that is responsible for exploring and initializing a system, termed a Host. The exploration and initialization process starts with a number of rules that the component and system designers shall follow.

2.3.1 Exploration and initialization rules

1. A Host shall be able to "reach" all agent devices that it is to be responsible for. This may require mechanisms to generate third party transactions to reach devices that are not transparently visible.
2. Maintenance responses generated by agent and switch devices shall be sent to the port that the maintenance request was received on. For example, consider a device that implements a 5 port switch. The system Host issues a maintenance read request to the switch device, which is received on input port 3. The switch, upon generating the maintenance response to the maintenance read request, must route it to output port 3 even though the switch may have been configured by default to route the response to a port other than port 3 (when the switch is configured it should also route the response to port 3).
3. All devices have CSRs to assist with exploration and initialization procedures. The registers used in this example contain the following information:
   - Base device ID register - This is the default device ID for the device, and it resides in a standard register in the CSR space at offset 0x60. At power-up, the base device ID defaults to logic 0xFF for all agent devices (0xFFFF for 16-bit route fields), with the exception of the boot code device and the Host device. The boot code device (if present) will have it's device ID default to 0xFE and the Host device will have it's device ID default to 0x00 as described in Section 2.2. A device may have multiple device IDs, but only this architecturally defined device ID is used in the exploration and initialization procedure.
   - Master Enable bit - the Master Enable bit is reset at power-up for agent devices and set for Host devices. The Master Enable bit is located in the *RapidIO Part 4: 8/16 LP-LVDS Physical Layer Specification* or the *RapidIO Part 6: 1x/4x LP-Serial Physical Layer Specification* Port General Control CSR at block offset 0x3C. If the Master Enable bit is clear the agent device is not allowed to issue requests and is only able to respond to received requests. This bit is used by the system Host to control when agents are allowed to issue transactions into the system. Switches are by default enabled and do not have a Master Enable bit.
   - Discovered bit - the Discovered bit is reset at power-up for agent devices and set for the Host device, and is located in the 8/16 LP-LVDS or 1x/4x LP-Serial physical layer Port General Control

CSR at block offset 0x3C. The system Host device sets this bit when the device has been discovered through the exploration mechanism. The Discovered bit is useful for detecting routing loops, and for hot plug or swap environments.

2.3.2 Exploration and initialization algorithm

If the above rules are followed, all agent devices are now accessible either as an end point that responds to any maintenance transaction or, for switches, via the hop_count mechanism.

The basic algorithm is to explore the system through each end point in sequence by first locating the adjacent device by sending a maintenance read to device ID=0xFF and hop count= 0x00, which is guaranteed to cause the adjacent device to respond. That device is then configured to reach the next device by assigning it a unique base device ID other than 0xFF, setting up route tables to reach the next device, etc.

When all devices in the system have been identified and have unique base device IDs assigned (no devices have a base device ID value=0xFF), the Host can then complete the final device ID assignment and configuration required for the application and enable agent devices to issue requests.

2.3.3 Exploration and initialization example

Figure 2-3 shows the previous example of a small single Host system.

Following the rules defined above, the base device ID value for all devices except the Host and boot ROM device after reset is applied is 0xFF, the Host has it’s Master Enable and Discovered bits set, and the agent devices have their Master Enable and Discovered bits cleared.

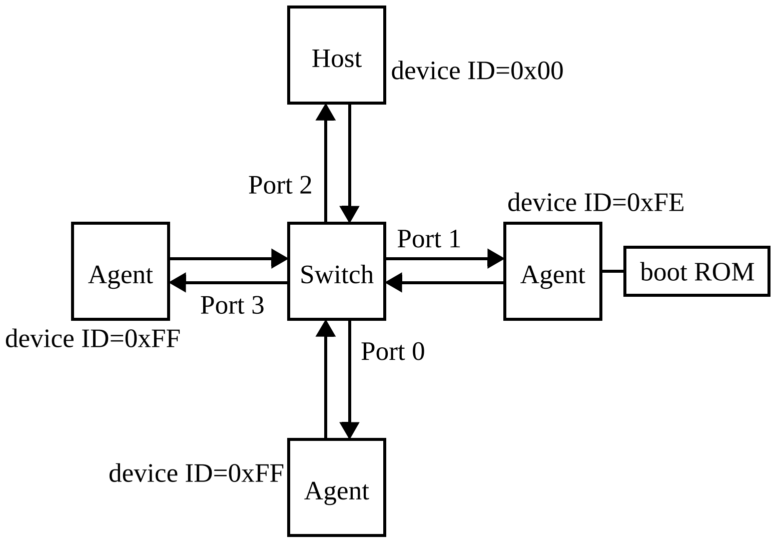


Figure 2-3. Example system

Assigning the Host's base device ID=0x00 is the first step in the process. The next step is to find the adjacent device, so the Host sends a maintenance read of offset 0x00_0000 to device ID=0xFF and hop_count=0x00. The switch consumes the request because the hop_count field is equal to zero and responds by sending the contents of it's Device Identity and Information CARs back to the port the request came from. From the returned information, the software on the Host can identify this as a switch. The Host then reads the switch port information CAR at offset 0x00_0014 to find out which port it is connected to. The response indicates a 4 port switch (which the Host may have already known from the device information register), connected to port 2.

The Host then examines the default routing tables for the switch to find the port route for the boot device ID=0xFE so it can preserve the path to the boot code (which it may still be running), and discovers that the boot device is located through port 1 of the switch. It also sets the switch's Discovered bit.

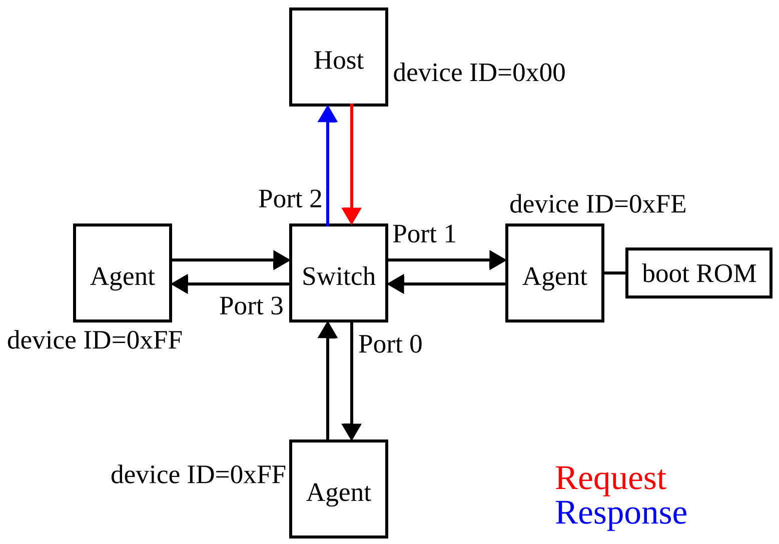


Figure 2-4. Finding the adjacent device

The next step is for the Host to configure the switch to route device ID=0xFF to port 0 and device ID=0x00 to port 2 (which it already was because of the boot device in the system) via maintenance write requests to hop_count=0x00. The Host then issues another maintenance read request, this time to device ID=0xFF and hop_count=0x01. The switch discovers that it is not the final destination of the maintenance request packet, so it decrements the hop_count and routes the packet to port 0 and on to the attached agent device. The agent device responds, and the switch routes the response packet to device ID=0x00 back through port 2 to the Host. Again, software identifies the device, sets its Discovered bit, configures it as required, and assigns the base device ID=0x01.

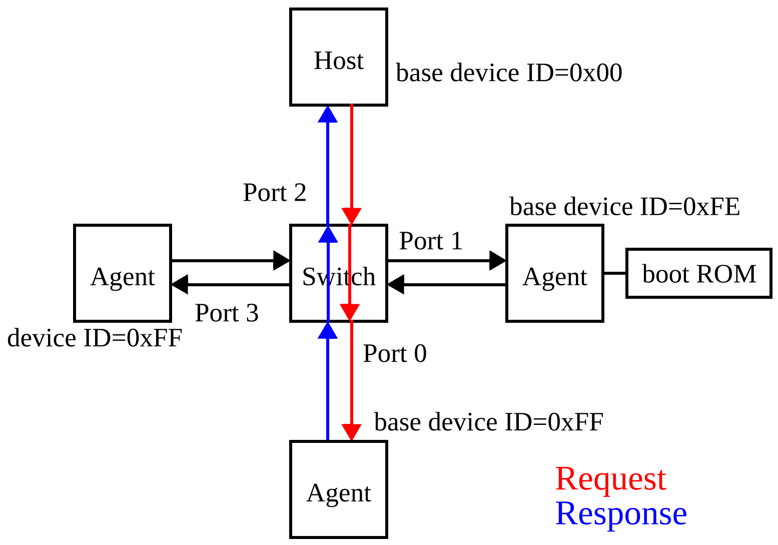


Figure 2-5. Finding the device on switch port 0

The Host then modifies the routing tables to now route device ID=0x01 to port 0. Since the boot device is located through port 1, instead of modifying the routing tables to route device ID=0xFF to port 1, the Host issues a maintenance read of device ID=0xFE (the boot device) and hop_count=0x01. The response identifies the agent on port 1, sets the agent's Discovered bit, and configures it as necessary, leaving the base device ID=0xFE so the Host can continue to execute the boot code.

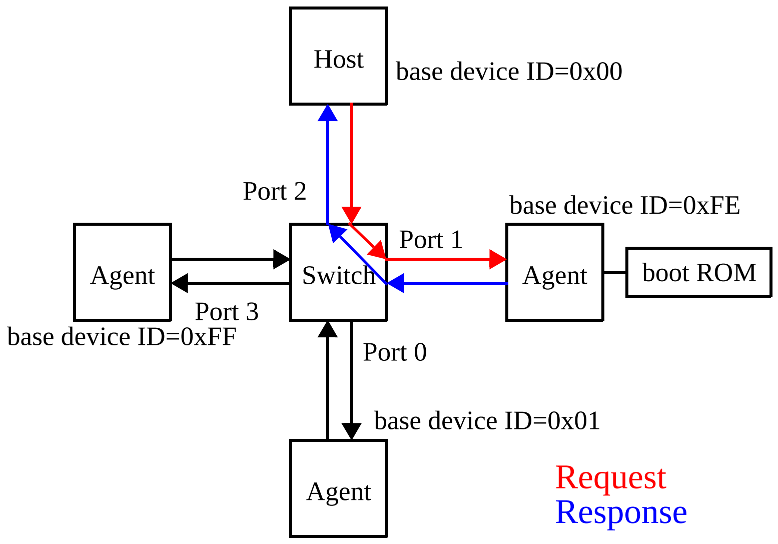


Figure 2-6. Finding the device on switch port 1

For the next iteration, the Host sets the switch device routing table entry for device ID=0xFF to route to port 3 (the Host already knows it is directly connected to port 2), and issues the maintenance read transaction as before.

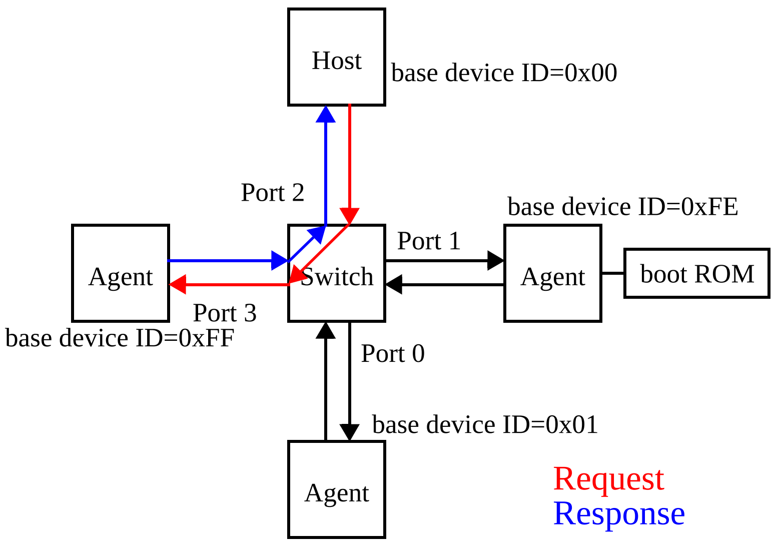


Figure 2-7. Finding the device on switch port 3

When the end point only agent responds with the requested CAR information the Host now knows that exploration is completed (there are no other paths to follow through the fabric), and can finalize configuring the system as shown in Figure 2-8. The agent devices can then have their Master Enable bits set so they can begin to issue transactions into the initialized system. The boot device ID can be changed, if desired, when the Host completes executing code from the boot ROM.

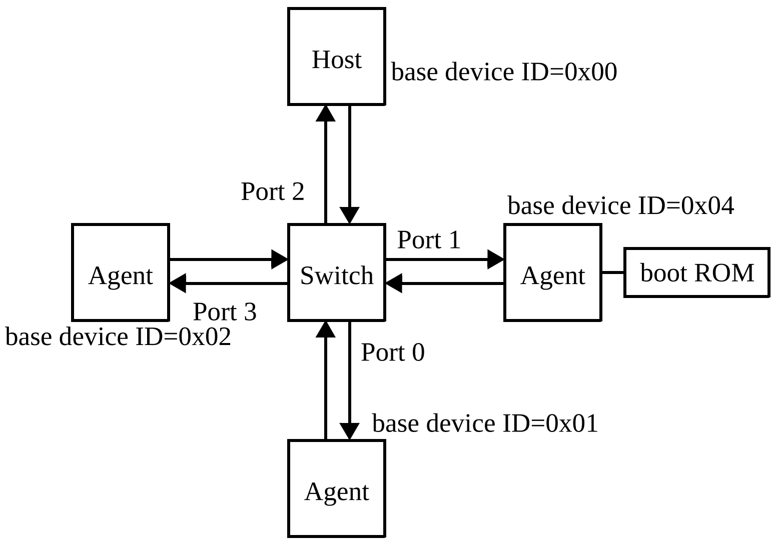


Figure 2-8. Final initialized system state

Variants to this procedure may be desirable. For example, a system may wish to enable some devices before exploration has been completed.

More complex systems with multiple Hosts, failed Host recovery, and hot swap requirements can be addressed with more complex algorithms utilizing the Host base device ID Lock Register and the Component Tag Register in standard registers in the CSR space at offsets 0x68 and 0x6C.

Chapter 3 RapidIO Device Class Requirements

3.1 Introduction

The RapidIO Architecture specifications allow for a variety of implementations. In order to form standard points of support for RapidIO, this chapter describes the requirements for RapidIO devices adhering to the *RapidIO Part 4: 8/16 LP-LVDS Physical Layer Specification* or the *RapidIO Part 6: 1x/4x LP-Serial Physical Layer Specification* and corresponding to different measures of functionality. Three device “classes” are defined, each with a minimum defined measure of support. The first class defines the functionality of the least capable device, with subsequent classes expanding the measure of support, in order to establish levels of inter-operability.

The RapidIO Architecture specifications allow for a variety of implementations. This chapter defines these implementations using two orthogonal capability dimensions.

The first dimension of measuring device capabilities is the logical layer capabilities for the devices. Three device "classes" are defined, each with a minimum defined measure of logical layer support. The first class defines the functionality of the least capable device, with subsequent classes expanding the measure of support in order to establish levels of inter-operability.

The second dimension of device capabilities is the ability of the device to support space based applications, known as "Space" class devices. Capabilities for "Space" devices are optional functions defined in other parts of the RapidIO specification. "Space" devices make these optional functions mandatory, usually for reasons of fault tolerance and robustness. "Space" device capability requirements are defined using the following categories:

- Basic Space device requirements apply to all space device profiles
- Enhanced Space device requirements are additional physical layer features that extend the Basic Space device requirements.
- Switch Space device requirements are specific to switch devices.
- Endpoint device requirements apply to all space endpoint devices.
- Endpoint-E device requirements are additional logical layer features that extend the endpoint device requirements.

Due to the orthogonal nature of the device dimensions, it is possible to combine these categories when describing a device. For example, it is possible to describe a device as a "Class 3 Enhanced Space Endpoint-E".

Note that some requirements may be in conflict when combining orthogonal categories. Space requirements shall be used to determine the capabilities of the device in the event of such conflicts.

3.2 Class Partitioning

Each class includes the functionality defined in all previous class devices and defines the minimum additional functionality for that class. A device is not required to comply *exactly* with a class, but may optionally supply additional features as a value-add for that device. All functions that are not required in any class list are also optional value-adds for a device.

First is a set of requirements that are applicable to all RapidIO compliant devices, including switch devices without end point functionality.

3.2.1 Generic: All devices

3.2.1.1 General requirements

- One or more 8/16 LP-LVDS and/or 1x/4x LP-Serial ports
  - (refer to *RapidIO Part 4: 8/16 LP-LVDS Physical Layer Specification* and/or *RapidIO Part 6: 1x/4x LP-Serial Physical Layer Specification*)
- Support for small (8-bit) transport device ID fields
  - (refer to *RapidIO Part 3: Common Transport Specification*, Section 2.4)
- Ability to accept requests with all sourceID and destinationID values on exit from reset
  - (refer to *RapidIO Part 3: Common Transport Specification*, Section 2.3)
- Support for recovery from a single corrupt packet or control symbol
  - (refer to *RapidIO Part 4: 8/16 LP-LVDS Physical Layer Specification*, Section 1.3.5) and/or *RapidIO Part 6: 1x/4x LP-Serial Physical Layer Specification*, Section 5.10.2)
- Support for packet retry protocol
  - (refer to *RapidIO Part 4: 8/16 LP-LVDS Physical Layer Specification*, Section 1.2.4) and/or *RapidIO Part 6: 1x/4x LP-Serial Physical Layer Specification*, Section 5.6)
- Support for throttle based flow control on 8/16 LP-LVDS physical layer ports
  - (refer to *RapidIO Part 4: 8/16 LP-LVDS Physical Layer Specification*, Section 2.3)
- Support for transaction ordering for flowID B
  - (end point programmability for all flow levels is recommended)
  - (refer to *RapidIO Part 4: 8/16 LP-LVDS Physical Layer Specification*,

Section 1.2.2) and/or *RapidIO Part 6: 1x/4x LP-Serial Physical Layer Specification*, Section 5.3.3)

- Switch devices maintain error coverage internally
  - (refer to *RapidIO Part 4: 8/16 LP-LVDS Physical Layer Specification*, Section 1.3.6) and/or *RapidIO Part 6: 1x/4x LP-Serial Physical Layer Specification*, Section 5.5)
- Support for maximum size packets for switch devices:
  - Switch devices that support Dev8 and/or Dev16 device IDs shall support a maximum packet size of 276 bytes. See *RapidIO Part 6: 1x/4x LP-Serial Physical Layer Specification*, Section 2.5
  - Switch devices that support Dev32 device IDs shall support a maximum packet size of 280 bytes. See *RapidIO Part 6: 1x/4x LP-Serial Physical Layer Specification*, Section 2.5
- Support for maximum size (256 byte) data payloads for end point devices
  - (refer to *RapidIO Part 1: Input/Output Logical Specification*, Section 3.1.2)
- Device must contain the following registers:
  - Device Identity CAR
  - Device Information CAR
  - Assembly Identity CAR
  - Assembly Information CAR
  - Processing Element Features CAR
  - Source Operations CAR
  - Destination Operations CAR
  - (refer to *RapidIO Part 1: Input/Output Logical Specification*, Section 4.4)

3.2.1.2 Operation support as target

- Maintenance read
  - (switch targeted by hop_count transport field)
  - (refer to *RapidIO Part 1: Input/Output Logical Specification*, Section 2.3.1, Section 3.1.10)
- Maintenance write
  - (switch targeted by hop_count transport field)
  - (refer to *RapidIO Part 1: Input/Output Logical Specification*, Section 2.3.1, Section 3.1.10)

3.2.1.3 Operation support as source

- <none>

3.2.2 Class 1: Simple target device

3.2.2.1 General requirements

- all Generic requirements
- Support for 34-bit address packet formats
  - (refer to *RapidIO Part 1: Input/Output Logical Specification,* Section 4.4.5)

3.2.2.2 Operation support as target

- all Generic requirements
- Write
  - (refer to *RapidIO Part 1: Input/Output Logical Specification*, Section 2.2.2, Section 3.1.7)
- Streaming-write
  - (refer to *RapidIO Part 1: Input/Output Logical Specification*, Section 2.2.2, Section 3.1.8)
- Write-with-response
  - (refer to *RapidIO Part 1: Input/Output Logical Specification*, Section 2.2.3, Section 3.1.7)
- Read
  - (refer to *RapidIO Part 1: Input/Output Logical Specification*, Section 2.2.1, Section 3.1.5)

3.2.2.3 Operation support as source

- all Generic requirements

3.2.3 Class 2: Simple mastering device

3.2.3.1 General requirements

- all Class 1 requirements

3.2.3.2 Operation support as target

- all Class 1 requirements

3.2.3.3 Operation support as source

- all Class 1 requirements
- Maintenance read
  - (refer to *RapidIO Part 1: Input/Output Logical Specification*, Section 2.3.1, Section 3.1.10)
- Maintenance write
  - (refer to *RapidIO Part 1: Input/Output Logical Specification*, Section 2.3.1, Section 3.1.10)

- Write
  - (refer to *RapidIO Part 1: Input/Output Logical Specification*, Section 2.2.2, Section 3.1.7)
- Streaming-write
  - (refer to *RapidIO Part 1: Input/Output Logical Specification*, Section 2.2.2, Section 3.1.8)
- Write-with-response
  - (refer to *RapidIO Part 1: Input/Output Logical Specification*, Section 2.2.3, Section 3.1.7)
- Read
  - (refer to *RapidIO Part 1: Input/Output Logical Specification*, Section 2.2.1, Section 3.1.5)

3.2.4 Class 3: Complex mastering device

3.2.4.1 General requirements

- all Class 2 requirements

3.2.4.2 Operation support as target

- all Class 2 requirements
- Atomic set
  - (refer to *RapidIO Part 1: Input/Output Logical Specification*, Section 2.2.4, Section 3.1.7)
- Maintenance port-write
  - (refer to *RapidIO Part 1: Input/Output Logical Specification*, Section 2.3.1, Section 3.1.10)
- Data message mailbox 0, letter 0, single segment, 8 byte payload
  - (refer to *RapidIO Part 2: Message Passing Logical Specification*, Section 2.2.2, Section 3.1.5)

3.2.4.3 Operation support as source

- all Class 2 requirements
- Atomic set
  - — (refer to *RapidIO Part 1: Input/Output Logical Specification*, Section 2.2.4, Section 3.1.7)
- Data message mailbox 0, letter 0, single segment, 8 byte payload
  - — (refer to *RapidIO Part 2: Message Passing Logical Specification*, Section 2.2.2, Section 3.1.5)\

3.3 Space Device Definition

3.3.1 Basic Space Device Requirements

- Basic Space devices shall support Dev8 and Dev16 system sizes.
  - — Refer to *RapidIO Part 3: Common Transport Specification*
- Basic Space devices shall support Baud Rate Class 1 operation.
  - — Refer to *RapidIO Part 6: 1x/4x LP-Serial Physical Layer Specification*
  - — Lane speeds of 1.25, 2.5, and 3.125 Gbaud are allowed.
- Basic Space devices shall support VC0 packet priorities 0, 1, 2, and 3
  - — Refer to *RapidIO Part 6: 1x/4x LP-Serial Physical Layer Specification*, Chapter 2 Packets
  - — Refer to *RapidIO Part 6: 1x/4x LP-Serial Physical Layer Specification*, section 6.6.3 Packet Priority and Transaction Request Flows
  - — Refer to *RapidIO Part 6: 1x/4x LP-Serial Physical Layer Specification*, Section 7.4.1 Processing Element Features CAR
- Basic Space devices shall not support baud rate discovery.
  - — Refer to *RapidIO Part 6: 1x/4x LP-Serial Physical Layer Specification*, Section 4.12.3 Baud Rate Discovery
  - — Refer to *RapidIO Part 6: 1x/4x LP-Serial Physical Layer Specification*, Section 7.6.9 Port n Control 2 CSRs
- Basic Space devices shall support Physical Layer Error Management functionality.
  - — Refer to *RapidIO Part 8: Error Management/Hot Swap Extensions Specification*, Section 1.2 Physical Layer Extensions
  - — Note that Hot Swap functionality is not required.
- Basic Space devices shall support the Port-Write error notification functionality.
  - — Refer to *RapidIO Part 8: Error Management/Hot Swap Extensions Specification*, 1.4 System Software Notification of Error

- Basic Space devices shall support packet multicast for Dev8 and Dev16 system sizes.
  - — Refer to *RapidIO Part 11: Multicast Extensions Specification*

3.3.2 Enhanced Space Device Requirements

- Enhanced Space devices shall be compliant to the RapidIO Revision 3.1 specification stack, or later versions.
- Enhanced Space devices shall support Baud Rate Class 1 and 2 operation
  - — Refer to *RapidIO Part 6: LP-Serial Physical Layer Specification*
  - — Lane speeds of 1.25, 2.5, 3.125, 5 and 6.25 Gbaud are allowed
- Enhanced Space devices shall support the Critical Request Flow bit.
  - — Refer to *RapidIO Part 6: LP-Serial Physical Layer Specification*, Chapter 2 Packets
  - — Refer to *RapidIO Part 6: LP-Serial Physical Layer Specification*, Section 6.6.3 Packet Priority and Transaction Request Flows
  - — Refer to *RapidIO Part 6: LP-Serial Physical Layer Specification*, Section 7.4.1 Processing Element Features CAR
- Enhanced Space devices shall support the Miscellaneous Physical Layer Register Block.
  - — Refer to *RapidIO Part 6: LP-Serial Physical Layer Specification*, Section 7.10 Miscellaneous Physical Layer Extension Block
- Enhanced Space devices shall support Structurally Asymmetric Links.
  - — Refer to *RapidIO Part 6: LP-Serial Physical Layer Specification*, Section 4.13 Structurally Asymmetric Links
  - — Refer to *RapidIO Part 6: LP-Serial Physical Layer Specification*, Section 5.18 Structurally Asymmetric Links
- When operating at 6.25 Gbaud lane rates, Enhanced Space devices shall implement register control of the transmit emphasis coefficient set.
- Enhanced Space devices shall support the LP-Serial Lane Extended Features Block.
  - — Refer to *RapidIO Part 6: LP-Serial Physical Layer Specification*, Section 7.7 LP-Serial Lane Extended Features Block
- Enhanced Space devices shall support Physical Layer Error Management and Hot Swap functionality.
  - — Refer to *RapidIO Part 8: Error Management/Hot Swap Extensions Specification*, Section 1.2 Physical Layer Extensions
- Enhanced Space devices shall support PRBS diagnostics functionality.
  - — Refer to *RapidIO Part 6: LP-Serial Physical Layer Specification*, Section 4.14 Pseudo Random Binary Sequence Testing

  — Refer to *RapidIO Part 6: LP-Serial Physical Layer Specification*, Section 5.20 Pseudo Random Binary Sequence Testing

- Enhanced Space Devices may support baud rate discovery.
  — Refer to *RapidIO Part 6: LP-Serial Physical Layer Specification*, Section 4.12.3 Baud Rate Discovery
  — Refer to *RapidIO Part 6: LP-Serial Physical Layer Specification*, Section 7.6.9 Port n Control 2 CSRs
  — Baud rate discovery functionality shall be compatible with devices that do not support baud rate discovery
  — It shall be possible to disable baud rate discovery functionality
- Enhanced Space Devices may support Multiple Event Capture functionality.
  — Refer to *RapidIO Part 8: Error Management/Hot Swap Extensions Specification*, Section 1.2.7 Physical Layer Multiple Event Capture
  — Refer to *RapidIO Part 8: Error Management/Hot Swap Extensions Specification*, Section 1.3.4 Logical/Transport Layer Multiple Event Capture

3.3.3 Space-10xN Device Requirements

- Space-10xN devices shall support Baud Rate Class 3 operation
  — Refer to RapidIO Part 6: LP-Serial Physical Layer Specification
  — Support for 10.3125 Gbaud lane speed is required for Space-10xN devices
- Space-10xN devices may support Baud Rate Class 1 and 2 operation
  — Refer to RapidIO Part 6: LP-Serial Physical Layer Specification
  — Lane speeds of 1.25, 2.5, 3.125, 5, and 6.25 Gbaud are allowed
- When operating at 10.3125 or 6.25 Gbaud lane rates, Space-10xN devices shall implement register control of the transmit emphasis coefficient set.

3.3.4 Space Switch Device Requirements

- Space Switches shall support packet routing for Dev8 and Dev16 system sizes.
  — Refer to *RapidIO Part 3: Common Transport Specification*
  — Space devices shall support at least 256 Device IDs (all of Dev8)
- Space Switches shall support standard registers to configure packet routing functionality for Dev8 and Dev16 system sizes.
  — Refer to *RapidIO Part 3: Common Transport Specification*, Section 3.4 Capability Registers (CARs)
  — Refer to *RapidIO Part 3: Common Transport Specification*, Section 3.5 Command and Status Registers (CSRs)
- Space Switches shall support distribution of Multicast-Event Control Symbols with predictable, low latency.

    - — Refer to *RapidIO Part 6: LP-Serial Physical Layer Specification,* 3.5.6.1 Multicast-Event Control Symbol and 6.5.3.4.1 Multicast-Event Control Symbols

- Space Switches shall support the LP-Serial Extended Feature Block for Generic Endpoint Free Devices, Software-assisted Error Recovery Option for all switch ports.
    - — Refer to *RapidIO Part 6: LP-Serial Physical Layer Specification,* section 7.5.4 Generic Endpoint Free Devices, Software-assisted Error Recovery Option
    - — Refer to *RapidIO Part 6: LP-Serial Physical Layer Specification,* Section 7.5.5 Register Map - I
    - — Refer to *RapidIO Part 6: LP-Serial Physical Layer Specification,* Section 7.5.6 Register Map - II
    - — "Register Map - I" or "Register Map - II" may be used
    - — The Port n Initialization Status CSRs shall be supported by devices that implement the "Register Map - II" layout.
        - – The encoding of the Lane Alignment, 1x/2x Mode Detection, and Port Initialization State Machine fields shall be documented.
        - – The value of the Lane Alignment, 1x/2x Mode Detection, and Port Initialization State Machine fields shall be consistent with the state of the lane alignment, 1x/2x Mode Detection, and Port Initialization State Machine implementations.
    - — Space Switches shall not support the Enable Inactive Lanes bit found in the *RapidIO Part 6: LP-Serial Physical Layer Specification,* Section 7.6.9 Port n Control 2 CSRs.
    - — Space Switches shall not support the Data Scrambling Disable bit found in the *RapidIO Part 6: LP-Serial Physical Layer Specification,* Section 7.6.9 Port n Control 2 CSRs.
    - — Space Switches shall not support the Port n Error and Status CSRs "Error Checking Disable" bit.
- Space Switches shall support packet "Time to Live" functionality.
    - — Refer to *RapidIO Part 8: Error Management/Hot Swap Extensions Specification,* Section 1.2.5 Packet Timeout Mechanism in a Switch Device
- Space Switches shall support Logical and Transport Layer error detection functionality for Maintenance packets.
    - — Refer to *RapidIO Part 8: Error Management/Hot Swap Extensions Specification,* Section 1.3 Logical and Transport Layer Extensions
- Space Switches shall support standard registers used to manage multicast functionality.
    - — Refer to *RapidIO Part 11: Multicast Extensions Specification,* Section 3.2

Capability Registers (CARs)

- — Refer to *RapidIO Part 11: Multicast Extensions Specification,* Section 3.3 Command and Status Registers (CSRs)
- — The Multicast Support bit shall be set in the Processing Element Features CAR.

• Space Switches shall support system implementation of MECS Time Synchronization Protocol.

- — Space Switches shall support low latency, low jitter distribution of Multicast Event Control Symbols
- — Refer to *RapidIO Part 6: LP-Serial Physical Layer Specification,* Section 6.5.3.6 MECS Time Synchronization Protocol

3.3.5 Space Endpoint Device Requirements

• Space Endpoints shall be capable of accepting packets whose destination ID does not match that found in the standard Base Device ID CSR, in order to support packet multicast.

- — Refer to *RapidIO Part 3: Common Transport Specification,* Section 2.3 System Packet Routing
- — Refer to *RapidIO Part 3: Common Transport Specification*, Section 3.5 Command and Status Registers (CSRs)
- — Refer to *RapidIO Part 11: Multicast Extensions Specification*

• Space Endpoints shall support the LP-Serial Extended Feature Block for Generic Endpoint Devices, Software-assisted Error Recovery Option for all ports.

- — Register Map - I or Register Map - II may be used
- — Refer to *RapidIO Part 6: LP-Serial Physical Layer Specification*, Section 7.5.2 Generic Endpoint Devices, Software-assisted Error Recovery Option
- — Refer to *RapidIO Part 6: LP-Serial Physical Layer Specification*, Section 7.5.5 Register Map - I
- — Refer to *RapidIO Part 6: LP-Serial Physical Layer Specification*, Section 7.5.6 Register Map - II
- — Space Endpoints shall not support the Enable Inactive Lanes bit found in *RapidIO Part 6: LP-Serial Physical Layer Specification,* Section 7.6.9 Port n Control 2 CSRs.
- — Space Endpoints shall not support the Data Scrambling Disable bit found in *RapidIO Part 6: LP-Serial Physical Layer Specification*, Section 7.6.9 Port n Control 2 CSRs.
- — Space Endpoints shall not support the Port n Error and Status CSRs "Error Checking Disable" bit.

- Space Endpoints shall support error detection functionality for the logical layer functionality indicated in their Source Operations CAR and Destination Operations CAR registers.
  - Refer to *RapidIO Part 1: Input/Output Logical Specification*, Chapter 3 Operation Descriptions
  - Refer to *RapidIO Part 2: Message Passing Logical Specification*, Chapter 3 Operation Descriptions
  - Refer to *RapidIO Part 5: Globally Shared Memory Logical Specification*, Chapter 3 Operation Descriptions
  - Refer to *RapidIO Part 8: Error Management/Hot Swap Extensions Specification*, Section 1.3 Logical and Transport Layer Extensions
  - Refer to *RapidIO Part 10: Data Streaming Logical Specification*, Chapter 3 Operation Descriptions
  - Refer to *RapidIO Part 10: Data Streaming Logical Specification*, Section 5.4 Additions to Existing Registers

3.3.6 Space Endpoint-E Device Requirements

- Space Endpoint-E devices shall implement support for the LCS Disable Present and LCS Disable bits.
  - Refer to *RapidIO Part 1: Input/Output Logical Specification*, Section 5.5.1 Processing Element Logical Layer Control CSR
- Space Endpoint-E devices shall support the MECS Time Synchronization Protocol.
  - Refer to *RapidIO Part 6: LP-Serial Physical Layer Specification*, Section 6.5.3.6 MECS Time Synchronization Protocol
- Space Endpoint-E devices shall support the Timestamp Generation register extension block required for MECS Time Synchronization Protocol.
  - At least one of the "MECS Slave Supported" and "MECS Master Supported" bits shall be set in the Timestamp CAR
  - Refer to *RapidIO Part 6: LP-Serial Physical Layer Specification*, Section 7.9 Timestamp Generation Extension Block

Blank page

Chapter 4 PCI Considerations

4.1 Introduction

RapidIO contains a rich enough set of operations and capabilities to allow transport of legacy interconnects such as PCI[1]. While RapidIO and PCI share similar functionality, the two interconnects have different protocols thus requiring a translation function to move transactions between them. A RapidIO to PCI bridge processing element is required to make the necessary translation between the two interconnects. This chapter describes architectural considerations for an implementation of a RapidIO to PCI bridge processing element. This chapter is not intended as an implementation instruction manual, rather, it is to provide direction to the bridge processing element architect and aid in the development of interoperable devices. For this chapter it is assumed that the reader has a thorough understanding of the PCI 2.2 and/or the PCI-X 1.0 specifications.

Figure 4-1 shows a typical system with devices connected using various RapidIO and PCI bus segments. A host bridge is connected to various peripherals via a PCI bus. A RapidIO bridge is used to translate PCI formatted transactions to the equivalent RapidIO operations to allow access to the rest of the system, including additional subordinate PCI bus segments.

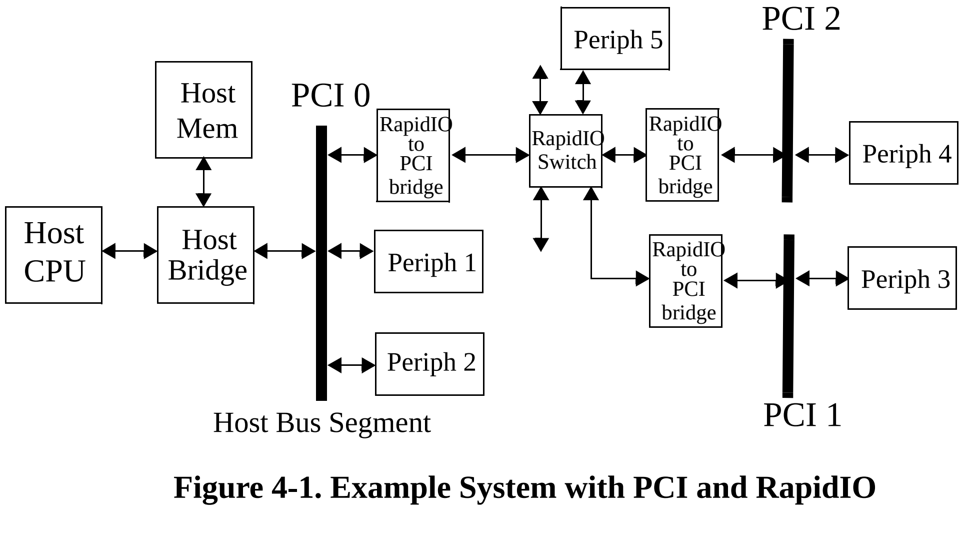


Figure 4-1. Example System with PCI and RapidIO

[1]For additional information on the Peripheral Component Interconnect PCI refer to the PCI 2.2 and the PCI-X 1.0 specifications.

Where RapidIO is introduced into a legacy system, it is desirable to limit changes to software. For transactions which must travel between RapidIO and PCI it is necessary to map address spaces defined on the PCI bus to those of RapidIO, translate PCI transaction types to RapidIO operations, and maintain the producer/consumer requirements of the PCI bus. This chapter will address each of these considerations for both PCI version 2.2 and PCI-X.

4.2 Address Map Considerations

PCI defines three physical address spaces, specifically, the memory, I/O memory, and configuration spaces. RapidIO, on the other hand, only addresses memory and configuration space. This section discusses memory space. Configuration space is discussed in Section 4.4. Figure 4-2 shows a simple example of the PCI memory and I/O address spaces for a host bus segment. In order for devices on the PCI bus to communicate with those connected through RapidIO, it is necessary to provide a memory mapping function. The example PCI host memory map uses a 32-bit physical address space resulting in 4 Gbytes of total address space. Host memory is shown at the bottom of the address map and peripheral devices at the top. Consider that the RapidIO to PCI bridge processing element contains a specified window(s) of address space mapped to it using the PCI base address register(s)[1]. The example shown in Figure 4-2 illustrates the RapidIO bridge address window located in an arbitrary software defined location. Likewise, if it was desired to communicate with PCI legacy I/O devices over RapidIO an I/O window would be assigned to the RapidIO to PCI bridge as shown.

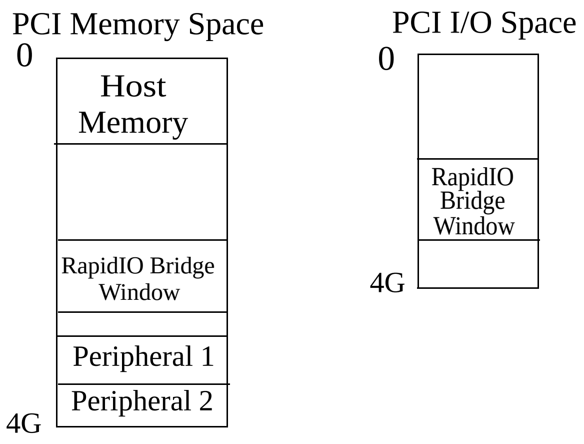


Figure 4-2. Host segment PCI Memory Map Example

Any transactions issued to the bus segment with an address that matches the RapidIO bridge window will be captured by the RapidIO to PCI bridge for forwarding. Once the transaction has been accepted by the RapidIO to PCI bridge processing element it must be translated to the proper RapidIO context as shown in

[1]Refer to the PCI 2.2 Specification Chapter 6 for a discussion on PCI address maps and configuration registers

Figure 4-3. For the purposes of this discussion this function is called the Address Mapping and Translation function (AMT). The AMT function is responsible for translating PCI addresses to RapidIO addresses as well as the translation and assignment of the respective PCI and RapidIO transaction types. The address space defined by the RapidIO bridge window may represent more than one subordinate RapidIO target device. A device on PCI bus segment 0 shown in Figure 4-1 may require access to a peripheral on PCI bus 1, bus 2, or RapidIO Peripheral 5. Because RapidIO uses source addressing (device IDs), the AMT is responsible for translating the PCI address to both a target device ID and associated offset address. In addition to address translation, RapidIO attributes, transaction types, and other necessary delivery information are established.

Similarly, transactions traveling from a RapidIO bus to a PCI bus must also pass through the AMT function. The address and transaction type are translated back into PCI format, and the AMT selects the appropriate address for the transaction. Memory mapping is relied upon for all transactions bridged between PCI and RapidIO.

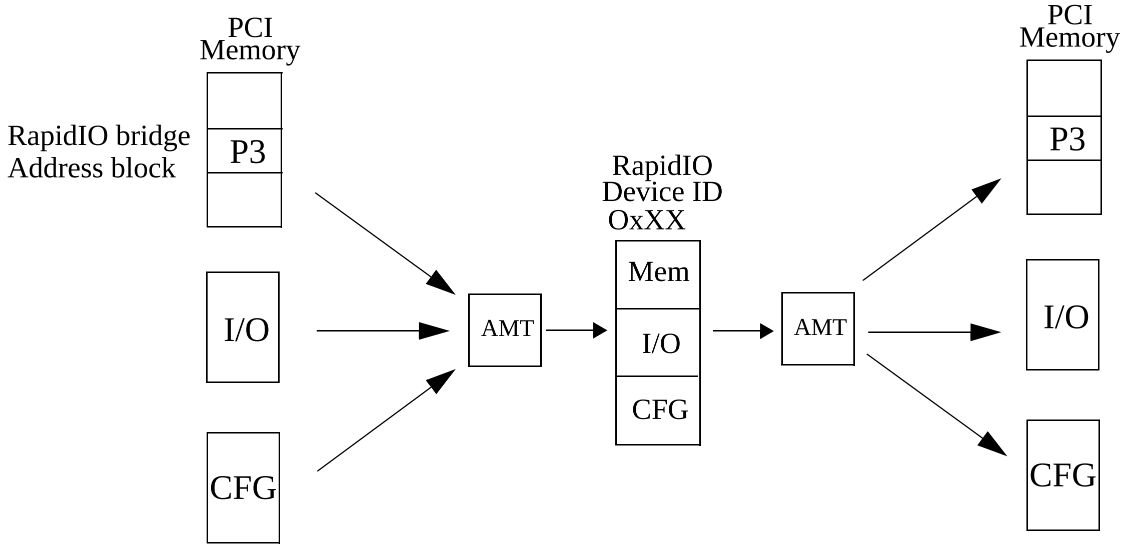


Figure 4-3. AMT and Memory Mapping

4.3 Transaction Flow

In considering the mapping of the PCI bus to RapidIO it is important to understand the transaction flow of PCI transactions through RapidIO.

4.3.1 PCI 2.2 Transaction Flow

The PCI 2.2 specification defines two classes of transaction types, posted and non-posted. Figure 4-4 shows the route taken by a PCI-RapidIO posted write transaction. Once the request is sent from the PCI Master on the bus, it is claimed by the bridge processing element which uses the AMT to translate it into a RapidIO request. Only when the transaction is in RapidIO format can it be posted to the RapidIO target. In some cases it may be desirable to guarantee end to end delivery of the posted write transaction. For this case the RapidIO NWRITE_R transaction is used which results in a response as shown in the figure.

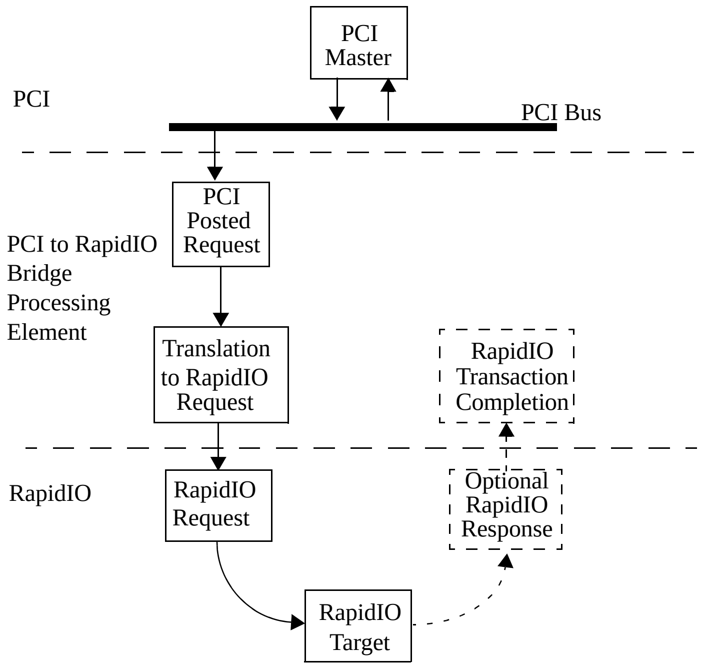


Figure 4-4. PCI Mastered Posted Write Transaction Flow Diagram

A non-posted PCI transaction is shown in Figure 4-5. The transaction is mastered by the PCI agent on the PCI bus and accepted by the RapidIO to PCI bridge. The transaction is retried on the PCI bus if the bridge is unable to complete it within the required timeout period. In this case the transaction is completed as a delayed transaction. The transaction is translated to the appropriate RapidIO operation and issued on the RapidIO port. At some time later a RapidIO response is received and the results are translated back to PCI format. When the PCI master subsequently retries the transaction, the delayed results are returned and the operation is completed.

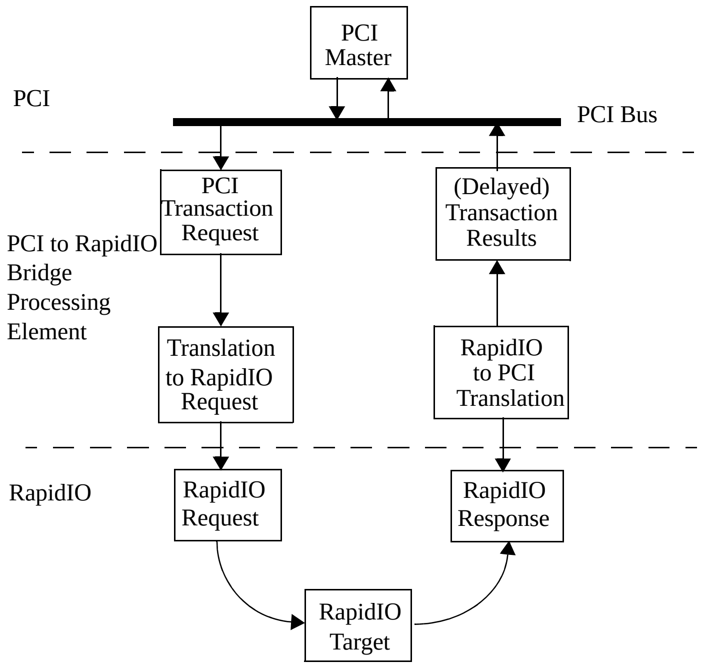

PCI Master
PCI
PCI Bus
PCI Transaction Request
(Delayed) Transaction Results
PCI to RapidIO Bridge Processing Element
Translation to RapidIO Request
RapidIO to PCI Translation
RapidIO
RapidIO Request
RapidIO Response
RapidIO Target

Figure 4-5. PCI Mastered non-posted (delayed) Transaction Flow Diagram

Because PCI allows unbounded transaction data tenures, it may be necessary for the RapidIO to PCI bridge to break the single PCI transaction into multiple RapidIO operations. In addition, RapidIO does not have byte enables and therefore does not support sparse byte transactions. For this case the transaction must be broken into multiple operations as well. “Section 4.7, Byte Lane and Byte Enable Usage” on page 47 describes this situation in more detail.

A RapidIO mastered operation is shown in Figure 4-6. For this case the RapidIO request transaction is received at the RapidIO to PCI bridge. The bridge translates the request into the appropriate PCI command which is then issued to the PCI bus. The PCI target may complete the transaction as a posted, non-posted, or delayed non-posted transaction depending on the command type. Once the command is successfully completed on the PCI bus the results are translated back into the RapidIO format and a response transaction is issued back to the RapidIO Master.

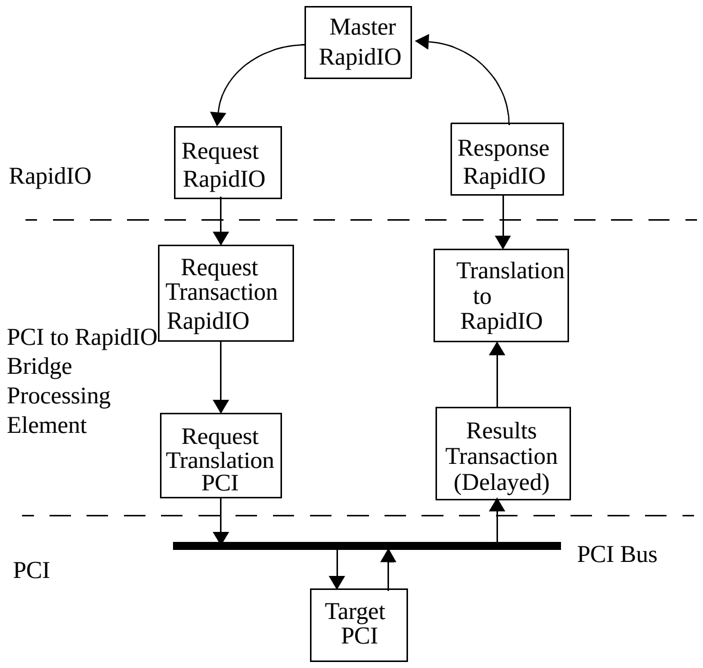


Figure 4-6. RapidIO Mastered Transaction

4.3.2 PCI-X Transaction Flow

The flow of transactions described in the previous section applies to the PCI-X bus as well. PCI-X supports split transactions instead of delayed transactions. The example shown in Figure 4-7 illustrates a transaction completed with a PCI-X split completion. The PCI-X master issues a transaction. The RapidIO to PCI-X bridge determines that it must complete the transaction as a split transaction, and responds with a split response. The transaction is translated to RapidIO and a request is issued on the RapidIO port. The RapidIO target returns a response transaction which is translated to a PCI-X Split Completion transaction completing the operation. PCI-X allows up to a 4 Kilobyte request. Larger PCI-X requests must be broken into multiple RapidIO operations. The RapidIO to PCI-X bridge may return the results back to the PCI-X Master using multiple Split Completion transactions in a pipelined fashion. Since PCI-X only allows devices to disconnect on 128 byte boundaries it is advantageous to break the large PCI-X request into either 128 or 256 byte RapidIO operations.

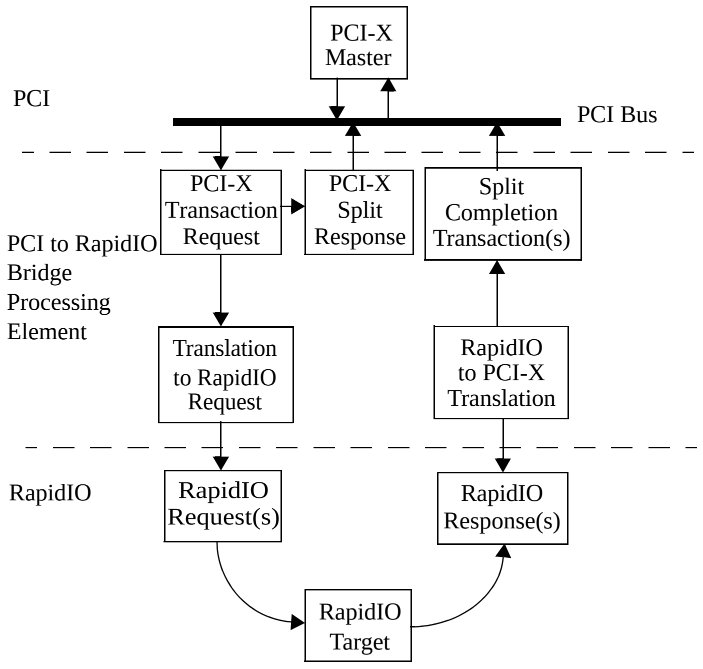

PCI-X Master
PCI
PCI Bus
PCI-X Transaction Request
PCI-X Split Response
Split Completion Transaction(s)
PCI to RapidIO Bridge Processing Element
Translation to RapidIO Request
RapidIO to PCI-X Translation
RapidIO
RapidIO Request(s)
RapidIO Response(s)
RapidIO Target

Figure 4-7. PCI-X Mastered Split Response Transaction

4.4 RapidIO to PCI Transaction Mapping

The RapidIO I/O and GSM specifications include the necessary transactions types to map all PCI transactions. Table 4-1 lists the map of transactions between PCI and RapidIO. A mapping mechanism such as the AMT function described in Section 4.2 is necessary to assign the proper transaction type based on the address space for which the transaction is targeted.

Table 4-1. PCI 2.2 to RapidIO Transaction Mapping

PCI Command	RapidIO Transaction	Comment
Interrupt-acknowledge	NREAD	
Special-cycle	NWRITE	
I/O-read	NREAD	
I/O-write	NWRITE_R	
Memory-read, Memory-Read-Line, Memory-Read-Multiple	NREAD or IO_READ_HOME	The PCI memory read transactions can be represented by the NREAD operation. If the operation is targeted to hardware maintained globally coherent memory address space then the I/O Read operation must be used (see “Section 4.6, Interactions with Globally Shared Memory” on page 43.)
Memory-write, Memory-write-and-invalidate	NWRITE, NWRITE_R, or FLUSH	The PCI Memory Write and Memory-Write-and-Invalidate can be represented by the NWRITE operation. If reliable delivery of an individual write transaction is desired then the NWRITE_R is used. If the operation is targeted to hardware maintained globally coherent memory address space then the Data Cache Flush operation must be used (refer to “Section 4.6, Interactions with Globally Shared Memory” on page 43.)

Table 4-1. PCI 2.2 to RapidIO Transaction Mapping

PCI Command	RapidIO Transaction	Comment
Configuration-read	NREAD	
Configuration-write	NWRITE_R	

PCI 2.2 memory transactions do not specify a size. It is possible for a PCI master to read a continuous stream of data from a target or to write a continuous stream of data to a target. Because RapidIO is defined to have a maximum data payload of 256 bytes, PCI transactions that are longer than 256 bytes must be broken into multiple RapidIO operations.

Table 4-2 shows the transaction mapping between PCI-X and RapidIO.

Table 4-2. PCI-X to RapidIO Transaction Mapping

PCI-X Command	RapidIO Transaction	Comment
Interrupt-acknowledge	NREAD	
Special-cycle	NWRITE	
I/O-read	NREAD	
I/O-write	NWRITE_R	
Memory-read DWORD	NREAD or IO_READ_HOME	The PCI-X memory read DWORD transactions can be represented by the NREAD operation. If the operation is targeted to hardware maintained coherent memory address space then the I/O Read operation must be used (refer to “Section 4.6, Interactions with Globally Shared Memory” on page 43.) This is indicated in PCI-X using the No Snoop (NS) bit described in Section 2.5 of the PCI-X 1.0 specification.
Memory-write	NWRITE, NWRITE_R, or FLUSH	The PCI-X Memory Write and Memory-Write-and-Invalidate can be represented by the NWRITE operation. If reliable delivery of an individual write transaction is desired then the NWRITE_R is used. If the operation is targeted to hardware maintained coherent memory address space then the Data Cache Flush operation must be used (refer to “Section 4.6, Interactions with Globally Shared Memory” on page 43.) This is indicated in PCI-X using the No Snoop (NS) bit described in Section 2.5 of the PCI-X 1.0 specification.
Configuration-read	NREAD	
Configuration-write	NWRITE_R	
Split Completion	--	The Split Completion transaction is the result of a request on the PCI-X bus that was terminated by the target with a Split Response. In the case of the RapidIO to PCI-X bridge this would be the artifact of a transaction that either the bridge mastered and received a split response or was the target and issued a split response. This command is equivalent to a RapidIO response transaction and does not traverse the bridge.

Table 4-2. PCI-X to RapidIO Transaction Mapping

PCI-X Command	RapidIO Transaction	Comment
Memory-read-block	NREAD or IO_READ_HOME	The PCI-X memory read transactions can be represented by the NREAD operation. If the operation is targeted to hardware maintained globally coherent memory address space then the I/O Read operation must be used (refer to “Section 4.6, Interactions with Globally Shared Memory” on page 43.) This is indicated in PCI-X using the No Snoop (NS) bit described in Section 2.5 of the PCI-X 1.0 specification.
Memory-write-block	NWRITE, NWRITE_R, or FLUSH	The PCI-X Memory Write and Memory-Write-and-Invalidate can be represented by the NWRITE operation. If reliable delivery of an individual write transaction is desired then the NWRITE_R is used. If the operation is targeted to hardware maintained globally coherent memory address space then the Data Cache Flush operation must be used (refer to “Section 4.6, Interactions with Globally Shared Memory” on page 43.) This is indicated in PCI-X using the No Snoop (NS) bit described in Section 2.5 of the PCI-X 1.0 specification.

The PCI-X addendum to the PCI specification adds the ability to do split operations. This results in an operation being broken into a Split Request and one or more Split Completions. As a target of a PCI-X Split Request, the RapidIO to PCI bridge may reply with a Split Response and complete the request using multiple RapidIO operations. The results of these operations are issued on the PCI-X bus as Split Completions. If the RapidIO to PCI-X bridge is the initiator of a Split Request, the target may also indicate that it intends to run the operation as a split transaction with a Split Response. In this case the target would send the results to the RapidIO to PCI-X bridge using Split Completions.

4.5 Operation Ordering and Transaction Delivery

This section discusses what the RapidIO to PCI bridge must do to address the requirements of the ordering rules of the PCI specifications.

4.5.1 Operation Ordering

Section 1.2.1 of the *RapidIO Part 1: Input/Output Logical Specification* describes a set of ordering rules. The rules guarantee ordered delivery of write data and that results of read operations will contain any data that was previously written to the same location.

For bridge devices, the PCI 2.2 specification has the additional requirement that the results of a read command push ahead posted writes in both directions.

In order for the RapidIO to PCI bridge to be consistent with the PCI 2.2 ordering rules it is necessary to follow the transaction ordering rules listed in section 1.2.1 of the I/O logical specification. In addition, the RapidIO to PCI bridge is required to adhere to the following RapidIO rule:

Read responses must push ahead all write requests and write responses.

4.5.2 Transaction Delivery Ordering

The RapidIO 8/16 LP-LVDS and 1x/4x LP-Serial physical layer specifications describe the mechanisms by which transaction ordering and delivery occur through the system. When considering the requirements for the RapidIO to PCI bridge it is first necessary to follow the transaction delivery ordering rules in section 1.2.4.1 of the 8/16 LP-LVDS specification and/or Section 5.8 of the 1x/4x LP-Serial specification. Further, it is necessary to add additional constraints to maintain programming model compatibility with PCI.

As described in Section 4.5.1 above, PCI has an additional transaction ordering requirement over RapidIO. In order to guarantee inter-operability, transaction ordering, and deadlock free operation, it is recommended that devices be restricted to utilizing transaction request flow level 0. In addition, it is recommended that response transactions follow a more strict priority assignment. Table 4-3 illustrates the priority assignment requirements for transactions in the PCI to RapidIO environment.

Table 4-3. Packet priority assignments for PCI ordering

RapidIO packet type	priority	comment
read request	0	This will push write requests and responses ahead
write request	1	Forces writes to complete in order, but allows write requests to bypass read requests
read response	1	Will force completion of preceding write requests and allows bypass of read requests
write response	2	Will prevent NWRITE_R request based deadlocks

The PCI transaction ordering model requires that a RapidIO device not issue a read request into the system unless it has sufficient resources available to receive and process a higher priority write or response packet in order to prevent deadlock. PCI 2.2 states that read responses cannot pass write transactions. The RapidIO specification provides PCI ordering by issuing priority 0 to read requests, and priority 1 to read responses and PCI writes. Since read responses and writes are issued at the same priority, the read responses will not pass writes.

4.5.3 PCI-X Relaxed Ordering Considerations

The PCI-X specification defines an additional ordering feature called relaxed ordering. If the PCI-X relaxed ordering attribute is set for a read transaction, the results for the read transaction are allowed to pass posted write transactions. PCI-X read transactions with this bit set allow the PCI-X to RapidIO bridge to ignore the rule described in Section 4.5.1. Table 4-4 shows the results of this additional function.

Table 4-4. Packet priority assignments for PCI-X ordering

RapidIO packet type	priority	comment
read request	0	This will push write requests and responses ahead
write request	1	Forces writes to complete in order, but allows write requests to bypass of read requests
read response	1	When PCI-X Relaxed Ordering attribute is set to 0. Will force completion of preceding write requests and allows bypass of read requests
read response	2, 3	When PCI-X Relaxed Ordering attribute is set to 1. The endpoint may promote the read response to higher priority to allow it to move ahead of posted writes.
write response	2	

4.6 Interactions with Globally Shared Memory

Traditional systems have two notions of system or subsystem cache coherence. The first, non-coherent, means that memory accesses have no effect on the caches in the system. The memory controller reads and writes memory directly, and any cached address becomes incoherent in the system. This behavior requires that all cache coherence with I/O be managed using software mechanisms, as illustrated in Figure 4-8.

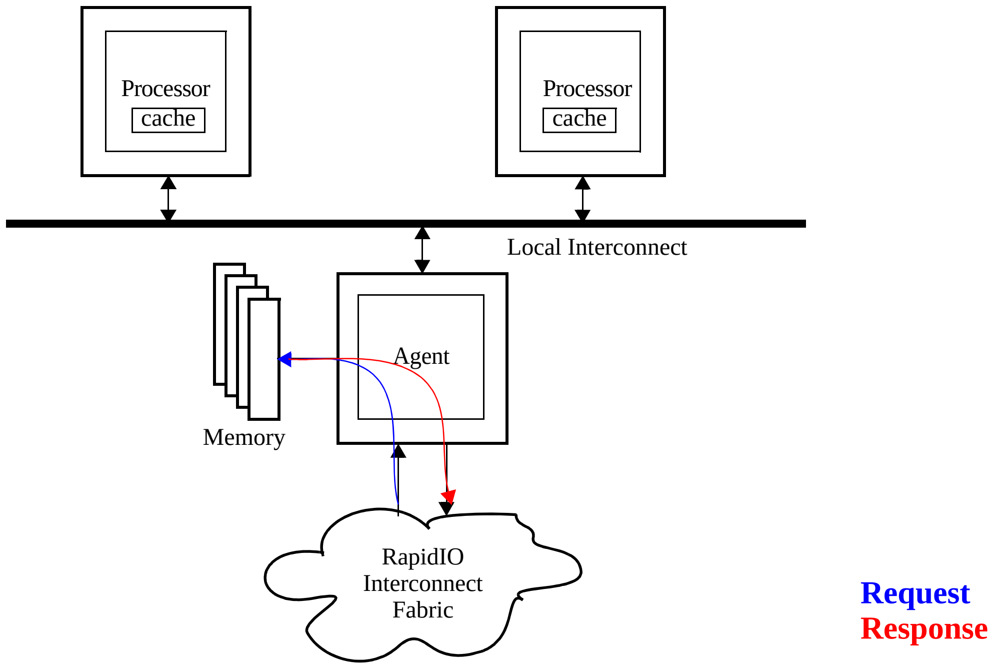


Figure 4-8. Traditional Non-coherent I/O Access Example

The second notion of system cache coherence is that of global coherence. An I/O access to memory causes a snoop cycle to be issued on the processor bus, keeping all of the system caches coherent with the memory, as illustrated in Figure 4-9.

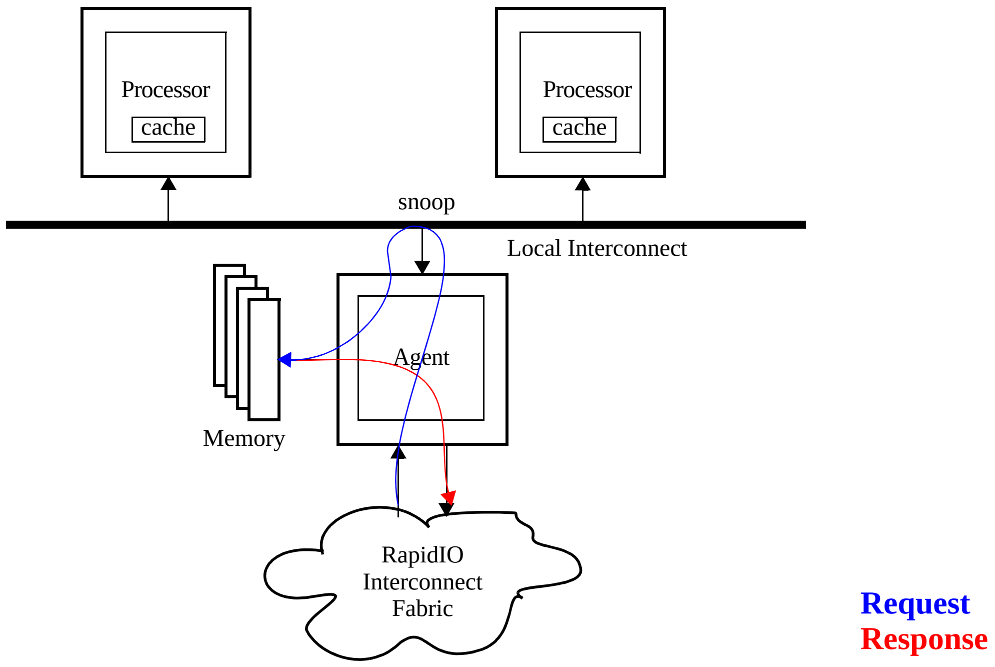


Figure 4-9. Traditional Globally Coherent I/O Access Example

With RapidIO globally shared systems, there is no common bus that can be used in order to issue the snoop, so global coherence requires special hardware support beyond simply snooping the bus. This leads to a third notion of cache coherence, termed local coherence. For local coherence, a snoop on a processor bus local to the targeted memory controller can be used to keep those caches coherent with that part of memory, but not caches associated with other memory controllers, as illustrated in Figure 4-10. Therefore, what once was regarded in a system as a “coherent access” is no longer globally coherent, but only locally coherent. Typically, deciding to snoop or not snoop the local processor caches is either determined by design or system architecture policy (always snoop or never snoop), or by an attribute associated with the physical address being accessed. In PCI-X, this attribute is the No Snoop (NS) bit described in Section 2.5 of the PCI-X 1.0 specification.

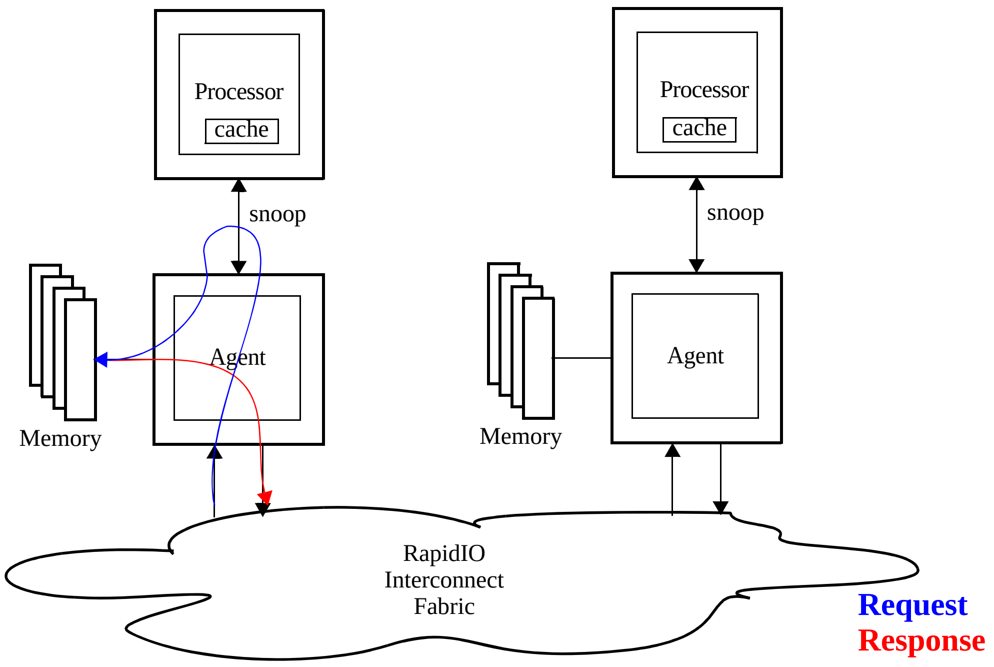


Figure 4-10. RapidIO Locally Coherent I/O Access Example

In order to preserve the concept of global cache coherence for a system, the *RapidIO Part 5: Globally Shared Memory Logical Specification* defines several operations that allow a RapidIO to PCI bridge processing element to access data in the globally shared space without having to implement all of the cache coherence protocol. These operations are the I/O Read and Data Cache Flush operations (globally shared memory specification, sections 3.2.9 and 3.2.10). For PCI-X bridging, these operations can also be used as a way to encode the NO SNOOP attribute for locally as well as globally coherent transactions. The targeted memory controller can be designed to understand the required behavior of such a transaction. These encodings also are useful for tunneling PCI-X transactions between PCI-X bridge devices.

The data payload for an I/O Read operation is defined as the size of the coherence granule for the targeted globally shared memory domain. However, the Data Cache Flush operation allows coherence granule, sub-coherence granule, and sub-double-word writes to be performed.

The IO_READ_HOME transaction is used to indicate to the GSM memory controller that the memory access is globally coherent, so the memory controller finds the latest copy of the requested data within the coherence domain (the requesting RapidIO to PCI bridge processing element is, by definition, not in the coherence domain) without changing the state of the participant caches. Therefore, the I/O Read operation allows the RapidIO to PCI bridge to cleanly extract data from a coherent portion of the system with minimal disruption and without having to be a full participant in the coherence domain.

The Data Cache Flush operation has several uses in a coherent part of a system. One such use is to allow a RapidIO to PCI bridge processing element to write to globally shared portions of the system memory. Analogous to the IO_READ_HOME transaction, the FLUSH transaction is used to indicate to the GSM memory controller that the access is globally coherent. The memory controller forces all of the caches in the coherence domain to invalidate the coherence granule if they have a shared copy (or return the data to memory if one had ownership of the data), and then writes memory with the data supplied with the FLUSH request. This behavior allows the I/O device to cleanly write data to the globally shared address space without having to be a full participant in the coherence domain.

Since the RapidIO to PCI bridge processing element is not part of the coherence domain, it is never the target of a coherent operation.

4.6.1 I/O Read Operation Details

Most of the complexity of the I/O Read operation resides in the memory controller. For the RapidIO to PCI Bridge processing element the I/O Read operation requires some additional attention over the non-coherent read operation. The necessary portions of the I/O Read state machine description in Section 6.10 of the globally shared memory specification are extracted below. Refer to Chapter 6 of the GSM specification for state machine definitions and conventions. The GSM specification takes precedence in the case of any discrepancies between the corresponding portions of the GSM specification and this description.

4.6.1.1 Internal Request State Machine

This state machine handles requests to the remote globally shared memory space.

```
remote_request(IO_READ_HOME, mem_id, my_id);
```

4.6.1.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect.

```
switch(remote_response)
case DONE:
            return_data();
            free_entry();
case DONE_INTERVENTION:                     // must be from third party
            set_received_done_message();
            if (received_data_only_message)
                      free_entry();
            else
                      // wait for a DATA_ONLY
            endif;
case DATA_ONLY:                             // this is due to an intervention, a
                                            // DONE_INTERVENTION should come
                                            // separately
            set_received_data_only_message();
            if (received_done_message)
                      return_data();
                      free_entry();
            else
                      return_data();        // OK for weak ordering
            endif;
```

```
case RETRY:
        remote_request(IO_READ_HOME, received_srcid, my_id);
default
        error();
```

4.6.2 Data Cache Flush Operation Details

As with the I/O Read operation, the complexity for the Data Cache Flush operation resides in the memory controller. The necessary portions of the Data Cache Flush state machine description from Section 6.10 of the GSM logical specification are extracted below. Refer to Chapters 2 and 3 of the GSM specification to determine the size of data payloads for the FLUSH transaction. The GSM specification takes precedence in the case of any discrepancies between the corresponding portions of the GSM specification and this description.

4.6.2.1 Internal Request State Machine

This state machine handles requests to the remote globally shared memory space.

```
remote_request(FLUSH, mem_id, my_id, data);
```

4.6.2.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect.

```
switch (received_response)
case DONE:
        local_response(OK);
        free_entry();
case RETRY:
        remote_request(FLUSH, received_srcid, my_id, data);
default:
        error();
```

4.7 Byte Lane and Byte Enable Usage

PCI makes use of byte enables and allows combining and merging of transactions. This may have the result of write transactions with sparse valid bytes. In order to save on transaction overhead, RapidIO does not include byte enables. RapidIO does, however, support a set of byte encodings defined in Chapter 3 of the *RapidIO Part 1: Input/Output Logical Specification*. PCI to RapidIO operations may be issued with sparse bytes. Should a PCI write transaction with byte enables that do not match a RapidIO byte encoding be issued to a RapidIO to PCI bridge, that operation must be broken into multiple valid RapidIO operations.

4.8 Error Management

Errors that are detected on a PCI bus are signaled using side band signals. The treatment of these signals is left to the system designer and is outside of the PCI specifications. Likewise, this document does not recommend any practices for the delivery of error interrupts in the system.

Blank page

Chapter 5 Globally Shared Memory Devices

5.1 Introduction

Different processing elements have different requirements when participating in a RapidIO GSM environment. The GSM protocols and address collision tables are written from the point of view of a fully integrated processing element comprised of a local processor, a memory controller, and an I/O controller. Obviously, the complexity and implementation requirements for this assumed device are much greater than required for a typical design. This chapter assumes that the reader is familiar with the *RapidIO Part 5: Globally Shared Memory Logical Specification*.

Additionally, this chapter contains the 8/16 LP-LVDS and 1x/4x LP-Serial physical layer transaction to priority mappings to guarantee that a system maintains cache coherence and is deadlock free.

5.2 Processing Element Behavior

In Chapter 2 of the globally shared memory specification are a number of examples of possible processing elements:

- A processor-memory processing element
- A memory-only processing element
- A processor-only processing element
- An I/O processing element
- A switch processing element

Of all of these, only the switch processing element does not have to implement anything additional to exist in a GSM system or sub-system. All of the remaining processing element types are of interest, and all are likely to exist in some form in the marketplace. This chapter is intended to define the portions of the protocol necessary to implement each of these devices. Other processing elements are allowed by the globally shared memory specification, for example, a memory-I/O processing element. The portions of the protocol necessary to implement these devices are not addressed in this chapter.

The behaviors described in this chapter have been extracted directly from revision 1.1 of the globally shared memory specification, and may be out of date with respect to the latest revision of that document. The GSM specification takes precedence in the case that there are discrepancies between it and this chapter.

5.2.1 Processor-Memory Processing Element

This processing element is very nearly the same as the assumed processing element used for the state machine description in Chapter 6, and requires nearly all of the described functionality. The following operation behavior is not changed from the Chapter 6 descriptions:

- Read
- Instruction read
- Read for ownership
- Data cache and instruction cache invalidate
- Castout
- TLB invalidate entry and TLB invalidate entry synchronize
- Data cache flush

This leaves the I/O Read operation. Since the processor-memory processing element does not contain an I/O device, this processing element will not generate the I/O read operation, but is required to respond to it. This removes the internal request state machine and portions of the response state machine, requiring the behavior described in Section 2.1.1 below. The only exception to this is the special case where there exists multiple coherence domains. It is possible that a processor in one coherence domain may wish to read data in another coherence domain and thus would require support of the I/O Read operation.

5.2.1.1 I/O Read Operations

This operation is used for I/O reads of globally shared memory space.

5.2.1.1.1 Response State Machine

This machine handles responses to requests made to the RapidIO interconnect made on behalf of a third party.

```
switch(remote_response)
case INTERVENTION:
            update_memory();
            remote_response(DONE_INTERVENTION, original_srcid, my_id);
            free_entry();
case NOT_OWNER,                                  // data comes from memory, mimic
                                                 // intervention
case RETRY:
            switch(directory_state)
            case LOCAL_MODIFIED,
            case LOCAL_SHARED:
                      remote_response(DATA_ONLY, original_srcid, my_id,
                                data);
                      remote_response(DONE_INTERVENTION, original_srcid,
                                my_id);
                      free_entry();
            case REMOTE_MODIFIED:                // spin or wait for castout
```

```
                remote_request(IO_READ_OWNER, received_srcid, my_id,
                        my_id);
        default:
                error();
default:
        error();
```

5.2.1.1.2 External Request State Machine

This machine handles requests from the system to the local memory or the local processor. This may require making further external requests.

```
if (address_collision)                          // use collision tables in
                                                // Chapter 7, "Address Collision Resolution
Tables"
elseif (IO_READ_HOME)                           // remote request to our local memory
        assign_entry();
        switch (directory_state)
        case LOCAL_MODIFIED:
                local_request(READ_LATEST);
                remote_response(DONE, received_srcid, my_id, data);
                                                // after push completes
                free_entry();
        case LOCAL_SHARED:
                remote_response(DONE, received_srcid, my_id, data);
                free_entry();
        case REMOTE_MODIFIED:
                remote_request(IO_READ_OWNER, mask_id, my_id, received_srcid);
        case SHARED:
                remote_response(DONE, received_srcid, my_id, data);
                free_entry();
        default:
                error();
else                                            // IO_READ_OWNER request to our caches
        assign_entry();
        local_request(READ_LATEST);             // spin until a valid response from
                                                // the caches
        switch (local_response)
        case MODIFIED:                          // processor indicated a push;
                                                // wait for it
                if (received_srcid == received_secid)
                                                // original requestor is also home
                                                // module
                        remote_response(INTERVENTION, received_srcid, my_id,
                                data);
                else
                        remote_response(DATA_ONLY, received_secid, my_id,
                                data);
                        remote_response(INTERVENTION, received_srcid, my_id);
                endif;
        case INVALID:                           // must have cast it out during
                                                // an address collision
                remote_response(NOT_OWNER, received_srcid, my_id);
        default:
                error();
        free_entry();
endif;
```

5.2.2 Memory-only Processing Element

This processing element is simpler than the assumed processing element used in Chapter 6, removing all of the internal request state machines and portions of all of the external request and response state machines. A memory-only processing element does not receive TLB invalidate entry or TLB invalidate synchronize operations. The required behavior for each operation is described below.

5.2.2.1 Read Operations

This operation is a coherent data cache read.

5.2.2.1.1 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of a third party.

```
switch(remote_response)
case INTERVENTION:
            update_memory();
            update_state(SHARED, original_srcid);
            remote_response(DONE_INTERVENTION, original_srcid, my_id);
            free_entry();
case NOT_OWNER,                                         // data comes from memory,
                                                        // mimic intervention
case RETRY:
            switch(directory_state)
            case LOCAL_SHARED:
                      update_state(SHARED, original_srcid);
                      remote_response(DATA_ONLY, original_srcid,
                                my_id, data);
                      remote_response(DONE_INTERVENTION, original_srcid,
                                my_id);
                      free_entry();
            case LOCAL_MODIFIED:
                      update_state(SHARED, original_srcid);
                      remote_response(DATA_ONLY, original_srcid,
                                my_id, data);
                      remote_response(DONE_INTERVENTION, original_srcid,
                                my_id);
                      free_entry();
            case REMOTE_MODIFIED:                       // spin or wait for castout
                      remote_request(READ_OWNER, received_srcid,
                                my_id, my_id);
            default:
                      error();
default:
            error();
```

5.2.2.1.2 External Request State Machine

This state machine handles read requests from the system to the local memory. This may require making further external requests.

```
if (address_collision)                                  // use collision tables in
                                                        // Chapter 7, “Address Collision Resolution
Tables”
else                                                    // READ_HOME
            assign_entry();
            switch (directory_state)
            case LOCAL_MODIFIED:
                      local_request(READ);
```

```
                update_state(SHARED, received_srcid);
                                                    // after possible push completes
                remote_response(DONE, received_srcid, my_id, data);
                free_entry();
        case LOCAL_SHARED,
        case SHARED:
                update_state(SHARED, received_srcid);
                remote_response(DONE, received_srcid, my_id, data);
                free_entry();
        case REMOTE_MODIFIED:
                if (mask_id ~= received_srcid)
                                                    // intervention case
                        remote_request(READ_OWNER, mask_id,
                                my_id, received_srcid);
                else
                        error();                    // he already owned it;
                                                    // cache paradox (or I-fetch after d-
                                                    // store if not fixed elsewhere)
                endif;
        default:
                error();
endif;
```

5.2.2.2 Instruction Read Operations

This operation is a partially coherent instruction cache read.

5.2.2.2.1 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of a third party.

```
switch(remote_response)
case INTERVENTION:
        update_memory();
        update_state(SHARED, original_srcid);
        remote_response(DONE, original_srcid, my_id);
        free_entry();
case NOT_OWNER,                                     // data comes from memory,
                                                    // mimic intervention
case RETRY:
        switch(directory_state)
        case LOCAL_SHARED:
                update_state(SHARED, original_srcid);
                remote_response(DONE, original_srcid, my_id);
                free_entry();
        case LOCAL_MODIFIED:
                update_state(SHARED, original_srcid);
                remote_response(DONE, original_srcid, my_id);
                free_entry();
        case REMOTE_MODIFIED:                       // spin or wait for castout
                remote_request(READ_OWNER, received_srcid,
                        my_id, my_id);
        default:
                error();
default:
        error();
```

5.2.2.2.2 External Request State Machine

This state machine handles instruction read requests from the system to the local memory. This may require making further external requests.

```
if (address_collision)                          // use collision tables in
                                                // Chapter 7, “Address Collision Resolution
Tables”
else                                            // IREAD_HOME
        assign_entry();
        switch (directory_state)
        case LOCAL_MODIFIED:
                local_request(READ);
                update_state(SHARED, received_srcid);
                                                // after possible push completes
                remote_response(DONE, received_srcid, my_id, data);
                free_entry();
        case LOCAL_SHARED,
        case SHARED:
                update_state(SHARED, received_srcid);
                remote_response(DONE, received_srcid, my_id, data);
                free_entry();
        case REMOTE_MODIFIED:
                if (mask_id ~= received_srcid)
                                                // intervention case
                        remote_request(READ_OWNER, mask_id,
                                my_id, received_srcid);
                else                            // he already owned it in his
                                                //data cache; cache paradox case
                        remote_request(READ_OWNER, mask_id, my_id, my_id);
                endif;
        default:
                error();
endif;
```

5.2.2.3 Read for Ownership Operations

This is the coherent cache store miss operation.

5.2.2.3.1 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of a third party.

```
switch(received_response)
case DONE:                                      // invalidates for shared
                                                // directory states
        if ((mask ~= (my_id OR received_id)) == 0)
                                                // this is the last DONE
                update_state(REMOTE_MODIFIED, original_srcid);
                remote_response(DONE, original_srcid, my_id, data);
                free_entry();
        else
                mask <= (mask ~= received_srcid);
                                                // flip the responder’s shared bit
        endif;                                  // and wait for next DONE
case INTERVENTION:
                                                // remote_modified case
        update_memory();                        // for possible coherence error
                                                // recovery
        update_state(REMOTE_MODIFIED, original_id);
        remote_response(DONE_INTERVENTION, original_id, my_id);
        free_entry();
case NOT_OWNER:                                 // data comes from memory, mimic
```

```
                                        // intervention
        switch(directory_state)
        case LOCAL_SHARED:
        case LOCAL_MODIFIED:
                update_state(REMOTE_MODIFIED, original_srcid);
                remote_response(DATA_ONLY, original_srcid, my_id,
                        data);
                remote_response(DONE, original_srcid, my_id);
                free_entry();
        case REMOTE_MODIFIED:
                remote_request(READ_TO_OWN_OWNER, received_srcid,
                        my_id, original_srcid);
        default:
                error();
case RETRY:
        switch (directory_state)
        case LOCAL_MODIFIED,
        case LOCAL_SHARED:
                update_state(REMOTE_MODIFIED, original_srcid);
                remote_response(DATA_ONLY, original_srcid, my_id,
                        data);
                remote_response(DONE, original_srcid, my_id);
                free_entry();
        case REMOTE_MODIFIED:           // mask_id must match received_srcid
                                        // or error condition
                remote_request(READ_TO_OWN_OWNER, received_srcid,
                        my_id, my_id);
        case SHARED:
                remote_request(DKILL_SHARER, received_srcid, my_id,
                        my_id);
        default:
                error();
default:
        error();
```

5.2.2.3.2 External Request State Machine

This state machine handles requests from the interconnect to the local memory. This may require making further external requests.

```
if (address_collision)                  // use collision tables
                                        // in Chapter 7, "Address Collision Resolution
Tables"
else                                    // READ_TO_OWN_HOME
        assign_entry();
        switch (directory_state)
        case LOCAL_MODIFIED,
        case LOCAL_SHARED:
                local_request(READ_TO_OWN);
                remote_response(DONE, received_srcid, my_id, data);
                                        // after possible push
                update_state(REMOTE_MODIFIED, received_srcid);
                free_entry();
        case REMOTE_MODIFIED:
                if (mask_id ~= received_srcid)
                                        //intervention case
                        remote_request(READ_TO_OWN_OWNER, mask_id, my_id,
                                received_srcid);
                else
                        error();        // he already owned it!
                endif;
        case SHARED:
                local_request(READ_TO_OWN);
                if (mask == received_srcid)
                                        //requestor is only remote sharer
                        update_state(REMOTE_MODIFIED, received_srcid);
```

```
                                remote_response(DONE, received_srcid, my_id, data);
                                                    // from memory
                                free_entry();
                      else                          //there are other remote sharers
                                remote_request(DKILL_SHARER, (mask ~= received_srcid),
                                          my_id, my_id);
                      endif;
            default:
                      error();
endif;
```

5.2.2.4 Data Cache and Instruction Cache Invalidate Operations

This operation is used with coherent cache store-hit-on-shared, cache operations.

5.2.2.4.1 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of a third party.

```
switch(received_response)
case DONE:                                          // invalidates for shared
                                                    // directory states
            if ((mask ~= (my_id OR received_id)) == 0)
                                                    // this is the last DONE
                      update_state(REMOTE_MODIFIED, original_srcid);
                      remote_response(DONE, original_srcid, my_id);
                      free_entry();
            else
                      mask <= (mask ~= received_srcid);
                                                    // flip the responder's shared bit
            endif;                                  // and wait for next DONE
case RETRY:
            remote_request({DKILL_SHARER, IKILL_SHARER}, received_srcid,
                      my_id);                       // retry
default:
            error();
```

5.2.2.4.2 External Request State Machine

This state machine handles requests from the system to the local memory. This may require making further external requests.

```
if (address_collision)                              // use collision tables in
                                                    // Chapter 7, "Address Collision Resolution
Tables"
else                                                // DKILL_HOME or IKILL_HOME
            assign_entry();
            if (DKILL_HOME)
                      switch (directory_state)
                      case LOCAL_MODIFIED,          // cache paradoxes; DKILL is
                                                    // write-hit-on-shared
                      case LOCAL_SHARED,
                      case REMOTE_MODIFIED:
                                error();
                      case SHARED:                  // this is the right case, send
                                                    // invalidates to the sharing list
                                local_request(DKILL);
                                if (mask == received_srcid
                                                    // requestor is only remote sharer
                                          update_state(REMOTE_MODIFIED, received_srcid);
                                          remote_response(DONE, received_srcid, my_id);
                                          free_entry();
                                else                // there are other remote sharers
                                          remote_request(DKILL_SHARER,
                                                    (mask ~= received_srcid), my_id, NULL);
                                endif;
                        default:
                                error();
                else                                // IKILL goes to everyone except the
                                                    // requestor
                        remote_request(IKILL_SHARER,
                                (mask <= (participant_list ~=
                                (received_srcid AND my_id), my_id);
endif;
```

5.2.2.5 Castout Operations

This operation is used to return ownership of a coherence granule to home memory, leaving it invalid in the cache.

5.2.2.5.1 External Request State Machine

This machine handles requests from the system to the local memory. This may require making further external requests.

```
assign_entry();
update_memory();
state_update(LOCAL_SHARED, my_id);                  // may be LOCAL_MODIFIED if the
                                                    // default is owned locally
remote_response(DONE, received_srcid, my_id);
free_entry();
```

5.2.2.6 Data Cache Flush Operations

This operation returns ownership of a coherence granule to home memory and performs a coherent write.

5.2.2.6.1 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of a third party.

```
switch(received_response)
case DONE:                                          // invalidates for shared directory
                                                    // states
                if ((mask ~= (my_id OR received_id)) == 0)
                                                    // this is the last DONE
                        remote_response(DONE, original_srcid, my_id, my_id);
                        if (received_data)
                                                    // with original request or response
                                update_memory();
                        endif;
                        update_state(LOCAL_SHARED); // or LOCAL_MODIFIED
                        free_entry();
                else
                        mask <= (mask ~= received_srcid);
                                                    // flip responder's shared bit
                endif;                              // and wait for next DONE
case NOT_OWNER:
                switch(directory_state)
                case LOCAL_SHARED,
                case LOCAL_MODIFIED:
                        remote_response(DONE, original_srcid, my_id);
                        if (received_data)
                                                    // with original request
                                update_memory();
                        endif;
                        free_entry();
                case REMOTE_MODIFIED:
                        remote_request(READ_TO_OWN_OWNER, received_srcid,
                                my_id, original_srcid);
                default:
                        error();
case RETRY:
                switch(directory_state)
                case LOCAL_SHARED,
                case LOCAL_MODIFIED:
                        remote_response(DONE, original_srcid, my_id);
                        if (received_data)
                                                        // with original request
                                update_memory();
                        endif;
                        free_entry();
                case REMOTE_MODIFIED:
                        remote_request(READ_TO_OWN_OWNER, received_srcid,
                                my_id, original_srcid);
                case SHARED:
                        remote_request(DKILL_SHARER, received_srcid, my_id);
                default:
                        error();
default:
                error();
```

5.2.2.6.2 External Request State Machine

This state machine handles requests from the system to the local memory. This may require making further external requests.

```
if (address_collision)                                  // use collision tables in
                                                        // Chapter 7, "Address Collision Resolution
Tables"
else                                                    // FLUSH
                assign_entry();
                switch (directory_state)
                case LOCAL_MODIFIED,
                case LOCAL_SHARED:
                        local_request(READ_TO_OWN);
                        remote_response(DONE, received_srcid, my_id);
                                                        // after snoop completes
                        if (received_data)              // from request or local response
                                update_memory();
                        endif;
                        update_state(LOCAL_SHARED, my_id);
                                                        // or LOCAL_MODIFIED
                        free_entry();
                case REMOTE_MODIFIED:
                        if (mask_id ~= received_srcid)  // owned elsewhere
                                remote_request(READ_TO_OWN_OWNER, mask_id, my_id,
                                        received_srcid);
                        else                            // requestor owned it; shouldn't
                                                        // generate a flush
                                error();
                        endif;
                case SHARED:
                        local_request(READ_TO_OWN);
                        if (mask == received_srcid)     // requestor is only remote sharer
                                remote_response(DONE, received_srcid, my_id);
                                                        // after snoop completes
                                if (received_data)      // from request or response
                                        update_memory();
                                endif;
                                update_state(LOCAL_SHARED, my_id); // or LOCAL_MODIFIED
                                free_entry();
                else                                    //there are other remote sharers
                         remote_request(DKILL_SHARER, (mask ~= received_srcid), my_id,
                                  my_id);
                endif;
        default:
                error();
endif;
```

5.2.2.7 I/O Read Operations

This operation is used for I/O reads of globally shared memory space.

5.2.2.7.1 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of a third party.

```
switch(remote_response)
case INTERVENTION:
        update_memory();
        remote_response(DONE_INTERVENTION, original_srcid, my_id);
        free_entry();
case NOT_OWNER,                                         // data comes from memory, mimic
                                                        // intervention
case RETRY:
        switch(directory_state)
        case LOCAL_MODIFIED,
        case LOCAL_SHARED:
                remote_response(DATA_ONLY, original_srcid, my_id,
                         data);
                remote_response(DONE_INTERVENTION, original_srcid,
                         my_id);
                free_entry();
        case REMOTE_MODIFIED:                           // spin or wait for castout
                remote_request(IO_READ_OWNER, received_srcid, my_id,
                         my_id);
        default:
                error();
default:
        error();
```

5.2.2.7.2 External Request State Machine

This machine handles requests from the system to the local memory. This may require making further external requests.

```
if (address_collision)                                  // use collision tables in
                                                        // Chapter 7, "Address Collision Resolution
Tables"
else                                                    // IO_READ_HOME
        assign_entry();
        switch (directory_state)
        case LOCAL_MODIFIED:
                local_request(READ_LATEST);
                remote_response(DONE, received_srcid, my_id, data);
                                                        // after push completes
                free_entry();
        case LOCAL_SHARED:
                remote_response(DONE, received_srcid, my_id, data);
                free_entry();
        case REMOTE_MODIFIED:
                remote_request(IO_READ_OWNER, mask_id, my_id, received_srcid);
        case SHARED:
                remote_response(DONE, received_srcid, my_id, data);
                free_entry();
```

```
            default:
                    error();
endif;
```

5.2.3 Processor-only Processing Element

A processor-only processing element is much simpler than the assumed combined processing described in Chapter 6. Much of the internal request, response, and external request state machines are removed.

5.2.3.1 Read Operations

This operation is a coherent data cache read.

5.2.3.1.1 Internal Request State Machine

This state machine handles requests to remote memory from the local processor.

```
if (address_collision)                              // this is due to an external request
                                                    // in progress or a cache
        local_response(RETRY);                      // index hazard from a previous request
else                                                // remote - we've got to go
                                                    // to another module
        assign_entry();
        local_response(RETRY);                      // can't guarantee data before a
                                                    // snoop yet
        remote_request(READ_HOME, mem_id, my_id);
endif;
```

5.2.3.1.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of the local processor.

```
switch(remote_response)
case DONE:
        local_response(SHARED);                     // when processor re-requests
        return_data();
        free_entry();
case DONE_INTERVENTION:                             // must be from third party
        set_received_done_message();
        if (received_data_only_message)
                free_entry();
        else
                                                    // wait for a DATA_ONLY
        endif;
case DATA_ONLY:                                     // this is due to an intervention, a
                                                    // DONE_INTERVENTIONshould come
                                                    // separately
        local_response(SHARED);
        set_received_data_only_message();
        if (received_done_message)
                return_data();
                free_entry();
        else
                return_data();                      // OK for weak ordering
        endif;
case RETRY:
        remote_request(READ_HOME, received_srcid, my_id);
default
        error();
```

5.2.3.1.3 External Request State Machine

This state machine handles read requests from the system to the local processor. This may require making further external requests.

```
if (address_collision)                              // use collision tables in
                                                    // Chapter 7, "Address Collision Resolution
Tables"
else                                                // READ_OWNER
            assign_entry();
            local_request(READ);                    // spin until a valid response
                                                    // from caches
            switch (local_response)
            case MODIFIED:                          // processor indicated a push;
                                                    // wait for it
                      cache_state(SHARED or INVALID);
                                                    // surrender ownership
                      if (received_srcid == received_secid)
                                                    // original requestor is also home
                                remote_response(INTERVENTION, received_srcid,
                                          my_id, data);
                      else
                                remote_response(DATA_ONLY, received_secid,
                                          my_id, data);
                                remote_response(INTERVENTION, received_srcid,
                                          my_id, data);
                      endif;
            case INVALID:                           // must have cast it out
                      remote_response(NOT_OWNER, received_srcid, my_id);
            default:
                      error();
            free_entry();
endif;
```

5.2.3.2 Instruction Read Operations

This operation is a partially coherent instruction cache read.

5.2.3.2.1 Internal Request State Machine

This state machine handles requests to remote memory from the local processor.

```
if (address_collision)                              // this is due to an external
                                                    // request in progress or a cache
            local_response(RETRY);                  // index hazard from a previous request
else                                                // remote - we've got to go
                                                    // to another module
            assign_entry();
            local_response(RETRY);
                                                    // can't guarantee data before a
                                                    // snoop yet
            remote_request(IREAD_HOME, mem_id, my_id);
endif;
```

5.2.3.2.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of the local processor.

```
switch(remote_response)
case DONE:
            local_response(SHARED);                 // when processor re-requests
            return_data();
            free_entry();
```

```
case DONE_INTERVENTION:                              // must be from third party
            set_received_done_message();
            if (received_data_only_message)
                      free_entry();
            else
                                                     // wait for a DATA_ONLY
            endif;
case DATA_ONLY:                                      // this is due to an intervention; a
                                                     // DONE_INTERVENTION should come
                                                     // separately
            local_response(SHARED);
            set_received_data_only_message();
            if (received_done_message)
                      return_data();
                      free_entry();
            else
                      return_data();                 // OK for weak ordering
            endif;
case RETRY:
            remote_request(IREAD_HOME, received_srcid, my_id);
default
            error();
```

5.2.3.2.3 External Request State Machine

This state machine handles instruction read requests from the system to the local processor.

```
if (address_collision)                               // use collision tables in
                                                     // Chapter 7, "Address Collision Resolution
Tables"
else                                                 // READ_OWNER request to our caches
            assign_entry();
            local_request(READ);                     // spin until a valid response
                                                     // from caches
            switch (local_response)
            case MODIFIED:                           // processor indicated a push;
                                                     // wait for it
                      cache_state(SHARED or INVALID);
                                                     // surrender ownership
                      if (received_srcid == received_secid)
                                                     // original requestor is also home
                                remote_response(INTERVENTION, received_srcid,
                                          my_id, data);
                      else
                                remote_response(DATA_ONLY, received_secid,
                                          my_id, data);
                                remote_response(INTERVENTION, received_srcid,
                                          my_id, data);
                      endif;
            case INVALID:                            // must have cast it out
                      remote_response(NOT_OWNER, received_srcid, my_id);
            default:
                      error();
            free_entry();
endif;
```

5.2.3.3 Read for Ownership Operations

This is the coherent cache store miss operation.

5.2.3.3.1 Internal Request State Machine

This state machine handles requests to remote memory from the local processor.

```
if (address_collision)                                  // this is due to an external request
                                                        // in progress or a cache index
            local_response(RETRY);                      // hazard from a previous request
else                                                    // remote - we’ve got to go to another
                                                        // module
            assign_entry();
            local_response(RETRY);
            remote_request(READ_TO_OWN_HOME, mem_id, my_id);
endif;
```

5.2.3.3.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of the local processor.

```
switch (received_response)
case DONE:
            local_response(EXCLUSIVE);
            return_data();
            free_entry();
case DONE_INTERVENTION:
            set_received_done_message();
            if (received_data_message)
                        free_entry();
            else
                                                        // wait for DATA_ONLY
            endif;
case DATA_ONLY:
            set_received_data_message();
            local_response(EXCLUSIVE);
            if (received_done_message)
                        return_data();
                        free_entry();
            else
                        return_data();                  // OK for weak ordering
            endif;                                      // and wait for a DONE
case RETRY:                                             // lost at remote memory so retry
            remote_request(READ_TO_OWN_HOME, mem_id, my_id);
default:
            error();
```

5.2.3.3.3 External Request State Machine

This state machine handles requests from the interconnect to the local processor.

```
if (address_collision)                                  // use collision tables
                                                        // in Chapter 7, “Address Collision Resolution
Tables”
elseif(READ_TO_OWN_OWNER                                // request to our caches
            assign_entry();
            local_request(READ_TO_OWN);                 // spin until a valid response from
                                                        // the caches
            switch (local_response)
            case MODIFIED:                              // processor indicated a push
                        cache_state(INVALID);
                                                        // surrender ownership
                        if (received_srcid == received_secid)
```

```
                                                //the original request is from the home
                                remote_response(INTERVENTION, received_srcid, my_id,
                                        data);
                        else                    // the original request is from a
                                                // third party
                                remote_response(DATA_ONLY, received_secid, my_id,
                                        data);
                                remote_response(INTERVENTION, received_srcid, my_id,
                                        data);
                        endif;
                        free_entry();
                case INVALID:                   // castout address collision
                        remote_response(NOT_OWNER, received_srcid, my_id);
                default:
                        error();
else                                            // DKILL_SHARER request to our caches
                assign_entry();
                local_request(READ_TO_OWN);
                                                // spin until a valid response from the
                                                // caches
                switch (local_response)
                case SHARED,
                case INVALID:                   // invalidating for shared cases
                        cache_state(INVALID);   // surrender copy
                        remote_response(DONE, received_srcid, my_id);
                        free_entry();
                default:
                        error();

endif;
```

5.2.3.4 Data Cache and Instruction Cache Invalidate Operations

This operation is used with coherent cache store-hit-on-shared, cache operations.

5.2.3.4.1 Internal Request State Machine

This state machine handles requests to remote memory from the local processor.

```
if (address_collision)                          // this is due to an external request in
                                                // progress or a cache index
                local_response(RETRY);          // hazard from a previous request
else                                            // remote - we’ve got to go to another
                                                // module
                assign_entry();
                local_response(RETRY);
                remote_request({DKILL_HOME, IKILL_HOME}, mem_id, my_id);
endif;
```

5.2.3.4.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of the local processor.

```
switch (received_response)
case DONE:
                local_response(EXCLUSIVE);
                free_entry();
case RETRY:
                remote_request({DKILL_HOME, IKILL_HOME}, received_srcid,
                        my_id);         // retry the transaction
default:
                error();
```

5.2.3.4.3 External Request State Machine

This state machine handles requests from the system to the local processor.

```
if (address_collision)                          // use collision tables in
                                                // Chapter 7, “Address Collision Resolution Tables”
else                                            // DKILL_SHARER or IKILL_SHARER request to our
caches
        assign_entry();
        local_request({READ_TO_OWN, IKILL});
                                                        // spin until a valid response from the
                                                        // caches
        switch (local_response)
        case SHARED,
        case INVALID:                                   // invalidating for shared cases
                cache_state(INVALID);                   // surrender copy
                remote_response(DONE, received_srcid, my_id);
                free_entry();
        default:
                error();
endif;
```

5.2.3.5 Castout Operations

This operation is used to return ownership of a coherence granule to home memory, leaving it invalid in the cache. A processor-only processing element is never the target of a castout operation.

5.2.3.5.1 Internal Request State Machine

A castout may require local activity to flush all caches in the hierarchy and break possible reservations.

```
assign_entry();
local_response(OK);
remote_request(CASTOUT, mem_id, my_id, data);
```

5.2.3.5.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of the local processor.

```
switch (received_response)
case DONE:
        free_entry();
default:
        error();
```

5.2.3.6 TLB Invalidate Entry, TLB Invalidate Entry Synchronize Operations

These operations are used for software coherence management of the TLBs.

5.2.3.6.1 Internal Request State Machine

The TLBIE and TLBSYNC transactions are always sent to all domain participants except the sender and are always to the processor, not home memory.

```
assign_entry();
remote_request({TLBIE, TLBSYNC}, participant_id, my_id);
endif;
```

5.2.3.6.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of the local processor. The responses are always from a coherence participant, not a home memory.

```
switch (received_response)
case DONE:
        if ((mask ~= (my_id OR received_id)) == 0)
                                                    // this is the last DONE
                free_entry();
        else
                mask <= (mask ~= received_srcid);
                                                    // flip the responder’s participant
                                                    // bit and wait for next DONE
        endif;
case RETRY:
        remote_request({TLBIE, TLBSYNC}, received_srcid, my_id, my_id);
default
        error();
```

5.2.3.6.3 External Request State Machine

This state machine handles requests from the system to the local memory or the local processor. The requests are always to the local caching hierarchy.

```
assign_entry();
local_request({TLBIE, TLBSYNC});                    // spin until a valid response
                                                    // from the caches
remote_response(DONE, received_srcid, my_id);
free_entry();
```

5.2.3.7 Data Cache Flush Operations

This operation returns ownership of a coherence granule to home memory and performs a coherent write.

5.2.3.7.1 Internal Request State Machine

This state machine handles requests to remote memory from the local processor.

```
if (address_collision)                              // this is due to an external
                                                    // request in progress or a cache index
        local_response(RETRY);                      // hazard from a previous request
else                                                // remote - we’ve got to go to
                                                    // another module
        assign_entry();
        remote_request(FLUSH, mem_id, my_id, data);
```

```
                                                        // data is optional
endif;
```

5.2.3.7.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect on behalf of the local processor.

```
switch (received_response)
case DONE:
        local_response(OK);
        free_entry();
case RETRY:
        remote_request(FLUSH, received_srcid, my_id, data);
                                                        // data is optional
default:
        error();
```

5.2.3.7.3 External Request State Machine

This state machine handles requests from the system to the local processor.

```
if (address_collision)                                  // use collision tables in
                                                        // Chapter 7, "Address Collision Resolution
Tables"
elseif (READ_TO_OWN_OWNER)                              // remote request to our caches
        assign_entry();
        local_request(READ_TO_OWN);                     // spin until a valid response
                                                        // from the caches
        switch (local_response)
        case MODIFIED:                                  // processor indicated a push,
                                                        // wait for it
                cache_state(INVALID);                   // surrender ownership
                remote_response(DONE, received_srcid, my_id, data);
        case INVALID:
                                                        // must have cast it out during an
                                                        // address collision
                remote_response(NOT_OWNER, received_srcid, my_id);
        default:
                error();
        free_entry();
else                                                    // DKILL_SHARER remote request
                                                        // to our caches
        assign_entry();
        local_request(DKILL);                           // spin until a valid response from
                                                        // the caches
        switch (local_response)
        case MODIFIED:                                  // cache paradox
                remote_response(ERROR, received_srcid, my_id);
        case INVALID:
                remote_response(DONE, received_srcid, my_id);
        default:
                error();
        free_entry();
endif;
```

5.2.3.8 I/O Read Operations

This operation is used for I/O reads of globally shared memory space. A processor-only processing element never initiates an I/O read operation.

5.2.3.8.1 External Request State Machine

This machine handles requests from the system to the local memory or the local processor. This may require making further external requests.

```
if (address_collision)                                  // use collision tables in
                                                        // Chapter 7, “Address Collision Resolution
Tables”
else                                                    // IO_READ_OWNER request to our caches
        assign_entry();
        local_request(READ_LATEST);                     // spin until a valid response from
                                                        // the caches
        switch (local_response)
        case MODIFIED:                                  // processor indicated a push;
                                                        // wait for it
                if (received_srcid == received_secid)
                                                        // original requestor is also home
                                                        // module
                        remote_response(INTERVENTION, received_srcid, my_id,
                                data);
                else
                        remote_response(DATA_ONLY, received_secid, my_id,
                                data);
                        remote_response(INTERVENTION, received_srcid, my_id);
                endif;
        case INVALID:                                   // must have cast it out during
                                                        // an address collision
                remote_response(NOT_OWNER, received_srcid, my_id);
        default:
                error();
        free_entry();
endif;
```

5.2.4 I/O Processing Element

The simplest GSM processing element is an I/O device. A RapidIO I/O processing element does not actually participate in the globally shared memory environment (it is defined as not in the coherence domain), but is able to read and write data into the GSM address space through special I/O operations that provide for this behavior. These operations are the I/O Read and Data Cache Flush operations. Other than the ability to read and write into the GSM address space, an I/O device has no other operational requirements. Since the I/O processing element is not part of the coherence domain, it is never the target of a coherence transaction and thus does not have to implement any of the related behavior, including the address collision tables.

Requirements for a specific I/O processing element, a RapidIO to PCI/PCI-X bridge, is discussed in Chapter 4, “PCI Considerations,” on page 4-33.

5.2.4.1 I/O Read Operations

This operation is used for I/O reads of globally shared memory space.

5.2.4.1.1 Internal Request State Machine

This state machine handles requests to remote memory.

```
remote_request(IO_READ_HOME, mem_id, my_id);
```

5.2.4.1.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect.

```
switch(remote_response)
case DONE:
            return_data();
            free_entry();
case DONE_INTERVENTION:                 // must be from third party
            set_received_done_message();
            if (received_data_only_message)
                      free_entry();
            else
                      // wait for a DATA_ONLY
            endif;
case DATA_ONLY:                         // this is due to an intervention, a
                                        // DONE_INTERVENTION should come
                                        // separately
            set_received_data_only_message();
            if (received_done_message)
                      return_data();
                      free_entry();
            else
                      return_data();    // OK for weak ordering
            endif;
case RETRY:
            remote_request(IO_READ_HOME, received_srcid, my_id);
default
            error();
```

5.2.4.2 Data Cache Flush Operations

This operation returns ownership of a coherence granule to home memory and performs a coherent write.

5.2.4.2.1 Internal Request State Machine

This state machine handles requests to remote memory.

```
remote_request(FLUSH, mem_id, my_id, data);
```

5.2.4.2.2 Response State Machine

This state machine handles responses to requests made to the RapidIO interconnect.

```
switch (received_response)
case DONE:
                local_response(OK);
                free_entry();
case RETRY:
                remote_request(FLUSH, received_srcid, my_id, data);
default:
                error();
```

5.2.5 Switch Processing Element

A switch processing element is required to be able to route all defined packets. Since it is not necessary for a switch to analyze a packet in order to determine how it should be treated outside of examining the priority and the destination device ID, a switch processing element does not have any additional requirements to be used in a globally shared memory environment.

5.3 Transaction to Priority Mappings

The Globally Shared Memory model does not have the concept of an end point to end point request transaction flow like the I/O programming model. Instead, all transaction ordering is managed by the load-store units of the processors participating in the globally shared memory protocol. The GSM logical specification behaviors assume an unordered and resource unconstrained communication fabric. The ordered fabric of the 8/16 LP-LVDS and the 1x/4x LP-Serial physical layers requires the proper transaction to priority mappings to mimic the effect of an unordered fabric to suit the GSM model. These mappings leverage the physical layer ordering and deadlock avoidance rules that are required by the I/O Logical layer. In addition, it is assumed that the latency-critical GSM operations are of necessity higher priority than non-coherent I/O traffic, therefore I/O operations are recommended to be assigned to the lowest system priority flow.

Table 5-1 shows the GSM transaction to priority mappings.

Table 5-1. Transaction to Priority Mapping

Request transaction	Request Packet Priority	Response Packet Priority
READ_TO_OWN_HOME	1	2 or 3
READ_HOME	1	2 or 3
IO_READ_HOME	1	2 or 3
IREAD_HOME	1	2 or 3
DKILL_HOME	1	2 or 3
IKILL_HOME	1	2 or 3
FLUSH (without data)	1	2 or 3
FLUSH (with data)	1	2 or 3
TLBIE	1	2 or 3
TLBSYNC	1	2 or 3
READ_OWNER	2	3
READ_TO_OWN_OWNER	2	3
IO_READ_OWNER	2	3
DKILL_SHARER	2	3
IKILL_SHARER	2	3
CASTOUT	2	3

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.

B

Bridge. A processing element that connects one computer bus to another, allowing a processing element on one bus to access an processing element on the other.

C

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.

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

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.

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.

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.

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

E

End point. A processing element which is the source or destination of transactions through a RapidIO fabric.

End point device. A processing element which contains end point functionality.

F

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

G

Globally shared memory (GSM). Cache coherent system memory that can be shared between multiple processors in a system.

H

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

I

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

I/O. Input-output.

L

Local memory. Memory associated with the processing element in question.

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 GSM. Message senders do not write to a memory address in the target.

N

Non-coherent. A transaction that does not participate in any system globally shared memory cache coherence mechanism.

O

Operation. A set of transactions between end point devices in a RapidIO system (requests and associated responses) such as a read or a write.

P

Packet. A set of information transmitted between devices in a RapidIO system.

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

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

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

ROM. Read-only memory.

S

Sender. The RapidIO interface output port on a processing element.

Source. The origin of a packet on the RapidIO interconnect, also referred to as an initiator.

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.

Blank page

RapidIO™ Interconnect Specification Part 8: Error Management/Hot Swap Extensions Specification

3.2, 1/2016

RapidIO.org

Revision History

Revision	Description	Date
1.2	First public release	09/13/2002
1.3	Technical changes: the following errata showings: 04-02-00002.001 and the following new features showings: 04-09-00022.002 Converted to ISO-friendly templates	02/23/2005
2.0	Technical changes: errata showing 06-02-00001.005	06/14/2007
2.1	Technical changes: errata showing 07-07-00000.004	07/09/2009
2.2	Technical changes: errata showing 10-08-00000.003	05/05/2011
3.0	Changed RTA contact information. Technical Changes: Standardized support for hot extraction and hot insertion, including register refinement/ additions. Register additions/changes to support Dev32. Register additions/changes to allow reuse of registers to latch information for packet types previously unsupported by this part of the RapidIO standard, and to enable latching control symbol information encoded using 64b/67b.	10/11/2013
3.1	Minor editorial changes. Technical changes: Added support for multiple error capture FIFO for physical layer and logical/transport layer errors. Register additions/changes for multiple error capture FIFO support. Register field additions for MECS and SMECS time synchronization error reporting.	9/18/2014
3.2	Inclusion of errata 9: Port n Link Uninit Discard Timer CSR Change	01/28/2016

Questions regarding RapidIO.org, specifications, or membership should be forwarded to:

RapidIO.org
8650 Spicewood Springs #145-515
Austin, TX 78759
512-827-7680 Tel.

Table of Contents

Chapter 1 Error Management Extensions

1.1 Introduction 9
1.2 Physical Layer Extensions 9
1.2.1 Port n Error Detect, Enable, and Capture CSRs 10
1.2.2 Error Reporting Thresholds 10
1.2.3 Error Rate Control and Status 11
1.2.4 Port Behavior When Error Rate Failed Threshold is Reached 12
1.2.5 Packet Timeout Mechanism in a Switch Device 13
1.2.6 Hot Swap Extensions 13
1.2.7 Physical Layer Multiple Event Capture 14
1.3 Logical and Transport Layer Extensions 15
1.3.1 Logical/Transport Error Detect, Enable, and Capture CSRs 16
1.3.2 Message Passing Error Detection 16
1.3.3 Other Logical Layer Errors 16
1.3.4 Logical/Transport Layer Multiple Event Capture 17
1.4 System Software Notification of Error 18
1.5 Mechanisms for Software Debug 18
1.6 IDLE3 Port_Status Extension 19

Chapter 2 Error Management Registers

2.1 Introduction 21
2.2 Additions to Existing Registers 21
2.2.1 Port n Control CSRs 21
2.2.2 Port n Error and Status CSRs 21
2.3 New Error Management Registers 22
2.4 Register Map 23
2.5 Command and Status Registers (CSRs) 26
2.5.1 Error Management Extensions Block Header 26
2.5.2 Error Management/Hot Swap Extension Block CAR 26
2.5.3 Logical/Transport Layer Error Detect CSR 27
2.5.4 Logical/Transport Layer Error Enable CSR 28
2.5.5 Logical/Transport Layer High Address Capture CSR 30
2.5.6 Logical/Transport Layer Address Capture CSR 31
2.5.7 Logical/Transport Layer Device ID Capture CSR 31
2.5.8 Logical/Transport Layer Control Capture CSR 32
2.5.9 Logical/Transport Layer Dev32 Destination ID Capture CSR 32
2.5.10 Logical/Transport Layer Dev32 Source ID Capture CSR 33
2.5.11 Port-Write Target deviceID CSR 33
2.5.12 Packet Time-to-live CSR 34
2.5.13 Port-write Dev32 Target deviceID CSR 34

Table of Contents

2.5.14 Port-Write Transmission Control CSR 34
2.5.15 Port n Error Detect CSR 35
2.5.16 Port n Error Rate Enable CSR 37
2.5.17 Port n Attributes Capture CSR 40
2.5.18 Port n Capture 0 CSR 41
2.5.19 Port n Capture 1 CSR 41
2.5.20 Port n Capture 2 CSR 42
2.5.21 Port n Capture 3 CSR 42
2.5.22 Port n Capture 4 CSR 43
2.5.23 Port n Error Rate CSR 44
2.5.24 Port n Error Rate Threshold CSR 45
2.5.25 Port n Link Uninit Discard Timer CSR 46
2.5.26 Port n FIFO Error Detect CSR 46

Annex A Error Management Discussion (Informative)

A.1 Introduction 49
A.2 Limitations of Error Management Discussion 49
A.3 Hot-insertion/extraction Discussion 50
A.4 Port-write Discussion 51
A.5 Physical Layer Fatal Error Recovery Discussion 52

List of Tables

1-1 Port Behavior when Error Rate Failed Threshold has been hit 12
1-2 Port-Write Packet Data Payload for Error Reporting 18
1-3 Port_status Field Definitions 20
2-1 Bit Settings for Port n Control CSRs 21
2-2 Bit Settings for Port n Error and Status CSRs 22
2-3 Extended Feature Space Reserved Access Behavior 22
2-4 Error Management/Hot Swap Extensions Register Requirements 23
2-5 Bit Settings for Error Management Extensions Block Header 26
2-6 Bit Settings for Error Management/Hot Swap Extension Block CAR 26
2-7 Bit Settings for Logical/Transport Layer Error Detect CSR 27
2-8 Bit Settings for Logical/Transport Layer Error Enable CSR 29
2-9 Bit Settings for Logical/Transport Layer High Address Capture CSR 30
2-10 Bit Settings for Logical/Transport Layer Address Capture CSR 31
2-11 Bit Settings for Logical/Transport Layer Device ID Capture CSR 31
2-12 Bit Settings for Logical/Transport Layer Control Capture CSR 32
2-13 Bit Settings for Logical/Transport Layer Dev32 Destination ID Capture CSR 32
2-14 Bit Settings for Logical/Transport Layer Dev32 Source ID Capture CSR....................................33
2-15 Bit Settings for Port-Write Target deviceID CSR 33
2-16 Bit Settings for Packet Time-to-live CSR....................................34
2-17 Bit Settings for Port-Write Dev32 Target deviceID CSR....................................34
2-18 Bit Settings for Port-Write Transmission Control CSR....................................35
2-19 Bit Settings for Port n Error Detect CSRs 35
2-20 Bit Settings for Port n Error Rate Enable CSRs 37
2-21 Bit Settings for Port n Attributes Capture CSRs....................................40
2-22 Bit Settings for Port n Capture 0 CSRs....................................41
2-23 Bit Settings for Port n Capture 1 CSRs....................................41
2-24 Bit Settings for Port n Capture 2 CSRs....................................42
2-25 Bit Settings for Port n Capture 3 CSRs....................................42
2-26 Bit Settings for Port n Capture 4 CSRs....................................43
2-27 Bit Settings for Port n Error Rate CSRs 44
2-28 Bit Settings for Port n Error Rate Threshold CSRs 45
2-29 Bit Settings for Port n Link Uninit Discard Timer CSRs 46
2-30 Bit Settings for Port n FIFO Error Detect CSRs....................................46

List of Tables

Blank page

Chapter 1 Error Management Extensions

1.1 Introduction

The error management extensions describe added requirements in all physical and logical layers. These extensions add definitions to bits that were previously reserved in the Port *n* Control CSRs and add new registers that are contained within the Error Management Extended Features Block. This chapter describes the behavior of a device when an error is detected and how the new registers and bits are managed by software and hardware. Implementation of this specification is optional.

1.2 Physical Layer Extensions

The following registers and register bit extensions allow software to monitor and control the reporting of physical layer errors:

- (Extensions to the) Port n Control CSRs defined in Section 2.2
- (Extensions to the) Port n Error and Status CSRs defined in Section 2.2
- Port-Write Target deviceID CSR defined in Section 2.5.11
- Port n Error Detect CSR defined in Section 2.5.15
- Port n Error Rate Enable CSR defined in Section 2.5.16
- Port n Attributes Capture CSR defined in Section 2.5.17
- Port n Capture 0 CSR through Port n Capture 4 CSR defined in Section 2.5.18 through Section 2.5.22
- Port n Error Rate CSR defined in Section 2.5.23
- Port n Error Rate Threshold CSR defined in Section 2.5.24

The Hot Swap Extensions consist of the following registers and register bit extensions, which allow software to be notified of the addition and removal of processing elements:

- Port-Write Target deviceID CSR defined in Section 2.5.11
- (Extensions to the) Port n Error Detect CSR defined in Section 2.5.15
- (Extensions to the) Port n Error Rate Enable CSR defined in Section 2.5.16
- Port n Link Uninit Discard Timer CSR defined in Section 2.5.25

1.2.1 Port *n* Error Detect, Enable, and Capture CSRs

Each detected occurrence of a physical layer error shall be logged by hardware in the Port *n* Error Detect CSRs by setting the appropriate error indication bit. Each detected error occurrence should set no more than one error indication bit, the bit that most specifically identifies the detected error. The Port *n* Error Detect CSRs does not lock when a detected error bit is set allowing each subsequent detected error to also be logged in the register. By reading the register, software may see the types of physical layer errors that have occurred since the register was last cleared.

Physical layer errors are enabled for error capture and error counting when the bit corresponding to the error has been set in the Port *n* Error Rate Enable CSRs by software. Error information is captured in the Port *n* Attributes Capture CSRs and the Port *n* Capture 0-4 CSRs. The Capture Valid Info bit in the Port *n* Attributes Capture CSRs indicates whether the error information in the capture CSRs is valid.

When the Capture Valid Info status bit is not set in the Port *n* Attributes Capture CSRs, information about the next enabled physical layer error shall be saved to the Port *n* Capture 0-4 CSRs. The Info Type and Error Type fields of the Port *n* Attributes Capture CSRs shall be updated and the register’s Capture Valid Info status bit shall be set by hardware to lock the error capture registers. Typically, the first 16 or 20 bytes of a packet, the 4 bytes of a delimited Control Symbol 24, or the 8 bytes of a delimited Control Symbol 48 or Control Symbol 64 that have a detected error are saved in the Port *n* Capture 0-4 CSRs. Packets smaller than 16 bytes are captured in their entirety. The Port *n* Capture 0-4 CSRs and the Port *n* Attributes Capture CSRs are not overwritten by hardware with error capture information for subsequent errors until software writes a zero to the Capture Valid Info bit.

The characters used for data transfer by the 8b/10b encoded LP-Serial physical layer protocol are 9 bit entities, but the specified formats for the Port *n* Capture 0-4 CSRs allocate only 8 bits per character (4 characters per 32-bit CSR) for recording the characters of a corrupted control symbol or the first 16 characters of a corrupted packet. Therefore it is not possible to unambiguously capture all possible LP-Serial control symbol and packet corruptions using the specified Port *n* Capture 0-4 CSRs format. Examples of this are when a code-group encoding a data character is changed by a transmission error into a code-group encoding a special character or a code-group with no 8b/10b decoding (an invalid character).

1.2.2 Error Reporting Thresholds

Physical layer errors are normally hidden from system software since they may be recovered with no loss of data and without software intervention. Two thresholds are defined in the Port *n* Error Rate Threshold CSRs which can be set to force a report to system software when the physical layer error rate reaches a level that is deemed by the system to be either degraded or unacceptable. The two thresholds are respectively the Degraded Threshold and the Failed Threshold. These thresholds are used as follows.

When the error rate counter is incremented, the Error Rate Degraded Threshold Trigger provides a threshold value that, when equal to or exceeded by the value in the Error Rate Counter in the Port *n* Error Rate register, shall cause the error reporting logic to set the Output Degraded-encountered bit in the Port *n* Error and Status CSRs, and notify the system software as described in Section 1.4.

The Error Rate Failed Threshold Trigger, if enabled, shall be larger than the degraded threshold trigger. It provides a threshold value that, when equal to or exceeded by the value in the Error Rate Counter, shall trigger the error reporting logic to set the Output Failed-encountered bit in the Port *n* Error and Status CSRs, and notify system software as described in Section 1.4.

No action shall be taken if the Error Rate Counter continues to exceed either threshold value after initial notification when additional errors are detected. No action shall be taken when the Error Rate Counter drops below either threshold.

1.2.3 Error Rate Control and Status

The fields in the Port *n* Error Rate CSRs are used to monitor the error rate of the port *n* physical layer.

The Error Rate Counter field contains the 8-bit Error Rate Counter. The Error Rate Counter shall increment when a physical layer error is detected whose associated enable bit is set in the Port *n* Error Rate Enable CSRs. The Error Rate Counter shall decrement at the rate specified by the Error Rate Bias field of the Port *n* Error Rate CSRs. The Error Rate Counter shall not underflow (shall not decrement when equal to 0x00) and shall not overflow (shall not increment when equal to 0xFF). The incrementing and decrementing of the Error Rate Counter are in no way affected by the values in the Degraded and Failed threshold fields. Software may reset the Error Rate Counter at any time.

The rate at which events are counted by the Error Rate Counter depends on the error rates and the bits set in the Port *n* Error Rate Enable CSRs. If bit 11 “Received packet-not-accepted control symbol enable” of the Port *n* Error Rate Enable CSRs is not set, only errors detected by Port *n* and whose counting is enabled are counted. If bit 11 is set, then errors detected by the connected port are also counted as the reception of a packet-not-accepted control symbol, while not an error in itself, is an indication that the connected port has detected a physical layer error. If in addition to bit 11 being set, one or more of virtual channels 1-8 are enabled and are operating in reliable transmission (RT) mode, packet retries requested by the connected port will also be counted because packet-not-accepted control symbols are used in this case to signal the rejection of an RT packet by the connected port due to a lack of buffer space.

The Error Rate Bias field determines the rate at which the Error Rate Counter is decremented and defines the acceptable error rate of the physical layer for error reporting purposes. In the absence of additional counted physical layer errors, this mechanism allows the system to recover from both Failed and Degraded levels of operation without a software reset of the Error Rate Counter. If the error rate of the physical layer errors being counted is less than the decrement rate specified in the Error Rate Bias field, the value of the Error Rate counter will rarely be greater than 0x01 or 0x02.

The Error Rate Recovery field defines how far above the Error Rate Failed Threshold Trigger in the Port *n* Error Rate Threshold Register the Error Rate Counter is allowed to count. In the absence of additional counted errors, this allows software to control the length of time required for the value of the Error Rate Counter to drop below both the Failed and Degraded Thresholds.

The Peak Error Rate field shall contain the largest value encountered by the Error Rate Counter since the field was last reset. This field is loaded whenever the current value of the Peak Error Rate field is exceeded by the value of the Error Rate Counter.

1.2.4 Port Behavior When Error Rate Failed Threshold is Reached

The behavior of a port when the Error Rate Counter in the Port *n* Error Rate CSRs reaches the Error Rate Failed Threshold and the threshold is enabled depends upon the values of the Stop on Port Failed-encountered Enable and the Drop Packet Enable bits in the Port *n* Control CSRs. The Table 1-1 below defines the required behavior.

Table 1-1. Port Behavior when Error Rate Failed Threshold has been hit

Stop on Port Failed Encountered Enable	Drop Packet Enable	Port Behavior	Comments
0	0	The port shall continue to attempt to transmit packets to the connected device if the Output Failed-encountered bit is set and/or if the Error Rate Failed threshold has been met or exceeded.	All devices
0	1	The port shall discard packets that receive a Packet-not-accepted control symbol when the Error Rate Failed Threshold has been met or exceeded. Upon discarding a packet, the port shall set the Output Packet-dropped bit in the Port *n* Error and Status CSRs. If the output port “heals”, the Error Rate Counter falls below the Error Rate Failed Threshold, the output port shall continue to attempt to forward all packets.	Switch Device Only

Table 1-1. Port Behavior when Error Rate Failed Threshold has been hit

Stop on Port Failed Encountered Enable	Drop Packet Enable	Port Behavior	Comments
1	0	The port shall stop attempting to send packets to the connected device when the Output Failed-encountered bit is set. The output port will congest.	All devices.
1	1	The port shall discard all output packets without attempting to send when the port's Output Failed-encountered bit is set. Upon discarding a packet, the port shall set Output Packet-dropped bit in the Port *n* Error and Status CSRs.	All devices.

1.2.5 Packet Timeout Mechanism in a Switch Device

In some systems, it is either desirable or necessary to bound the length of time a packet can remain in a switch. To enable this functionality, a switch shall monitor the length of time each packet accepted by one of its ports has been in the switch. The acceptance of a packet by a port is signaled by the port issuing a packet-accepted control symbol for the packet. The timing begins when the port accepts the packet.

If a packet remains in a switch longer than the Time-to-Live time specified by the Time-to-Live field of the Packet Time-to-live CSR as defined in Section 2.5.12, the packet shall be discarded rather than forwarded, the Output Packet-Dropped bit shall be set in the Port *n* Error and Status CSRs, and the system shall be notified as described in Section 1.4.

1.2.6 Hot Swap Extensions

When a Field Replaceable Unit (FRU) is inserted into a running system, it may be necessary to immediately inform system software. Similarly, when an FRU is removed from a running system, it may be necessary to immediately inform system software. The Link Uninit to OK Transition event can be used to inform system software of the insertion or removal of an FRU.

In the event that an FRU is removed from a system unexpectedly, the number of physical layer errors detected is uncertain. It is not possible to set the Physical Layer Error Management extensions thresholds, as described in sections 1.2.2 and 1.2.3, to differentiate between an expected bit error rate and FRU removal. The Hot Swap Extensions uses a timeout period for link reinitialization, the Port *n* Link Uninit Discard Timer CSRs, to detect when a link has been unavailable for a period of time deemed excessive by the system. When the Port *n* Link Uninit Discard Timer period expires, packets are discarded to avoid system congestion. Depending on the system design, the congestion could prevent system software from handling the unexpected FRU removal, which could lead to system failure.

A port-write may be sent to inform system software of a Hot Swap Extensions event. The Hot Swap Extensions events are incorporated into the Port *n* Error Detect CSRs, as the contents of this CSR are sent in a port-write. The Hot Swap Extensions events are also included in the Port *n* Error Rate Enable CSRs, as this is the standard register that controls notification and information capture for physical layer events.

However, unlike the Error Management Extensions physical layer events, the removal or insertion of an FRU is not a correctable error. For this reason, unlike Error Management Extensions events, Hot Swap Extension events shall not contribute to the error reporting thresholds described in section 1.2.2/1.2.3, shall not cause any error information to be latched, and shall not cause the Port *n* Capture 0-4 CSRs to lock.

1.2.7 Physical Layer Multiple Event Capture

Some fault tolerant applications require capture of multiple events to understand the sequence of events that lead to a failure. It is possible to implement the capture of multiple errors as a First-In-First-Out (FIFO) queue of events, where each entry in the FIFO represents the six registers starting with the Port n Attributes Capture CSR at the time of the entry. The Port n FIFO Error Detect CSR, also part of each FIFO entry, captures information similar to the Port n Error Detect CSR. The Port n Error Detect CSR behavior is constant, whether or not a FIFO is implemented.

The FIFO shall provide a queue of at least two entries, with the oldest entry at the front of the queue and new entries added at the end of the queue.

A FIFO entry becomes occupied when that entry is added to the end of the FIFO queue. The six registers starting with the Port n Attributes Capture CSR, and the Port n FIFO Error Detect CSR, shall occupy the oldest unoccupied entry in the FIFO.

The oldest occupied FIFO entry shall become unoccupied when software writes a zero to the Capture Valid Info bit. The FIFO shall be considered full if all entries are occupied. The FIFO shall be considered empty if all entries are unoccupied.

The FIFO error capture function supplies the value that occupies a FIFO entry. The FIFO error capture function operates on enabled and disabled events, which are defined in 1.2.1, “Port n Error Detect, Enable, and Capture CSRs”. The FIFO error capture function shall operate as follows:

- The FIFO error capture function value for the Port n FIFO Error Detect CSR shall consist of all events detected since the last time the FIFO error capture function value occupied an entry in the FIFO.
- The FIFO error capture function value for the Port n Attributes Capture CSR shall be updated for every detected enabled event, and shall consist of attributes information for the detected enabled event.
- The FIFO error capture function value for the five registers starting with the Port n Capture 0 CSR shall be updated for every detected enabled event.

If the FIFO is not full and an enabled event has been detected, the current FIFO error capture function value shall occupy the next entry in the FIFO. If multiple enabled events occur simultaneously, at least one event shall occupy an entry in the FIFO.

The Capture Valid Info bit in the Port n Attributes Capture CSR shall be 1 when at least one FIFO entry is occupied. The Capture Valid Info bit in the Port n Attributes Capture CSR shall be 0 when all FIFO entries are unoccupied. The value of other fields in the Port n Attributes Capture CSR, and the value of the Port n FIFO Error Detect CSR and the five registers starting with the Port n Capture 0 CSR, is undefined when all FIFO entries are unoccupied.

1.3 Logical and Transport Layer Extensions

While the RapidIO physical layer may be working properly, an end point processing element may encounter logical or transport layer errors, or other errors unrelated to its RapidIO ports, while trying to complete a transaction. The “ERROR” status response transaction is the mechanism for the target device to indicate to the source that there is a problem completing the request. Experiencing a timeout waiting for a response is also a symptom of an end point or switch fabric with a problem. These types of errors are logged and reporting enabled with a set of registers that are separate from those used for the Physical Layer errors:

- Logical/Transport Layer Error Detect CSR defined in Section 2.5.3
- Logical/Transport Layer Error Enable CSR defined in Section 2.5.4
- Logical/Transport Layer Capture CSRs defined in Section 2.5.5 to Section 2.5.8

1.3.1 Logical/Transport Error Detect, Enable, and Capture CSRs

When a logical or transport layer error is detected, the appropriate error bit shall be set by the hardware in the Logical/Transport Layer Error Detect CSR. If the corresponding bit is also set in the Logical/Transport Layer Error Enable CSR, the detect register shall lock, the appropriate information is saved in the Logical/Transport Layer Capture registers, all resources held by the transaction are freed, and system software is notified of the error as described in Section 1.4. If multiple enabled errors occur during the same clock cycle, multiple bits will be set in the detect register and the contents of the Logical/Transport Layer Capture registers are implementation dependent. Once locked, subsequent errors will not set another error detect bit. The contents of the Logical/Transport Capture CSRs are valid if the bitwise AND of the Logical/Transport Layer Error Detect CSR and the Logical/Transport Layer Error Detect Enable CSR is not equal to zero (0x00000000).

Software shall write the Logical/Transport Detect register with all logic 0s to clear the error detect bits or a corresponding enable bit to unlock the register. Any other recovery actions associated with these types of errors are system dependent and outside the scope of this specification.

1.3.2 Message Passing Error Detection

Message passing is a special case of logical layer error recovery requiring error detection at both the source and destination ends of the message. The source of the message has the request-to-response timeout (defined in the Port Response Timeout Control CSR in the RapidIO Physical Layer specifications) to detect lost request or response packets in the switch fabric. However, in order to not hang the recipient mailbox in the case of a lost request packet for a multiple packet message, the recipient mailbox shall have an analogous response-to-request timeout. This timeout is for sending a response packet to receiving the next request packet of a given message operation, and has the same value as the request-to-response timeout that is already specified. The Logical/Transport Layer Control Capture CSR contains the ‘msg info’ field to capture the critical information of the last received (or sent) message segment before timeout.

1.3.3 Other Logical Layer Errors

The RapidIO specification contains many logical layer packet types beyond the Logical I/O and Messaging types, which can be used in systems with greatly varying complexity. The capabilities necessary to find the root cause of defects for these packet types and systems also vary with customer requirements and the implementation technology. While the need to find defects in these systems is constant, it is not necessary to define standard methods of debugging them, as this does not affect interoperability.

RapidIO devices should consider their users needs for defect determination, and capture information appropriate to the scale and complexity of the system. Debug needs should be considered both as the source of a request and the target. Debug capabilities for customer field use may also be needed.

The basis of defect determination is information about detected errors. The Logical/Transport layer capture registers may be used to capture such information.

1.3.4 Logical/Transport Layer Multiple Event Capture

Some fault tolerant applications require capture of multiple events to understand the sequence of events that lead to a failure. It is possible to implement the capture of multiple errors as a First-In-First-Out (FIFO) queue of events, where each entry in the FIFO consists of the Logical/Transport Layer Error Detect CSR, and all Logical/Transport Layer Capture CSRs, at the time of the entry.

The FIFO shall provide a queue of at least two entries, with the oldest entry at the front of the queue and new entries added at the end of the queue.

A FIFO entry becomes occupied when that entry is added to the end of the FIFO queue. The Logical/Transport Layer Error Detect CSR, and all Logical/Transport Layer Capture CSRs shall access the oldest occupied entry in the FIFO.

The oldest occupied FIFO entry shall become unoccupied when software writes 0x00000000 to the Logical/Transport Layer Error Detect CSR. The FIFO shall be considered full if all entries are occupied. The FIFO shall be considered empty if all entries are unoccupied.

The FIFO error capture function supplies the value that occupies a FIFO entry. The FIFO error capture function operates on enabled and disabled events, as defined in 1.3.1, “Logical/Transport Error Detect, Enable, and Capture CSRs”. The FIFO error capture function shall operate as follows:

- The FIFO error capture function value for the Logical/Transport Layer Error Detect CSR shall consist of all events detected since the last time the FIFO error capture function value occupied an entry in the FIFO.
- The FIFO error capture function value for all Logical/Transport Layer Capture CSRs shall be updated for every detected enabled event.

If the FIFO is not full and an enabled event has been detected, the current FIFO error capture function value shall occupy the next entry in the FIFO. If multiple enabled events occur simultaneously, at least one event shall occupy an entry in the FIFO.

The Logical/Transport Layer Error Detect CSR shall not be 0 when the FIFO is not empty. The Logical/Transport Layer Error Detect CSR shall be 0 when the FIFO is empty. The value for all Logical/Transport Layer Capture CSRs is undefined when the FIFO is empty.

1.4 System Software Notification of Error

System software is notified of logical, transport, and physical layer errors in two ways. An interrupt is issued to the local system by a device, the method of which is not defined in this specification, or a Maintenance port-write operation is issued by a device. Maintenance port-write operations are sent to a predetermined system host (defined in the Port-write Target deviceID CSR in Section 2.5.11). The sending device sets the Port-write Pending status bit in the Port *n* Error and Status CSRs. A 16 byte data payload of the Maintenance Port-write packet contains the contents of several CSRs, the port on the device that encountered the error condition (for port-based errors), and some optional implementation specific additional information as shown in Table 1-2. Software indicates that it has seen the port-write operation by clearing the Port-write Pending status bit.

The Component Tag CSR is defined in the *RapidIO Part 3: Common Transport Specification*, and is used to uniquely identify the reporting device within the system. The Port ID field, the Logical/Transport Layer Detect CSR defined in Section 2.5.3, and the Port *n* Error Detect CSRs defined in Section 2.5.15, are used to describe the encountered error condition.

Table 1-2. Port-Write Packet Data Payload for Error Reporting

Data Payload Byte Offset	Word	
0x0	Component Tag CSR	
0x4	Port *n* Error Detect CSRs	
0x8	Implementation specific	Port ID (byte)
0xC	Logical/Transport Layer Error Detect CSR	

1.5 Mechanisms for Software Debug

In most systems, it is difficult to verify the error handling software. The Error management extensions make some registers writable for easier debug.

The Logical/Transport Layer Error Detect CSR and the six error capture registers starting with Logical/Transport Layer High Address Capture CSR are writable by software to allow software debug of the system error recovery mechanisms. For software debug, software must write the desired information into the six error capture registers starting with Logical/Transport Layer High Address Capture CSR. If the Logical/Transport Layer Error Capture FIFO Implemented bit is set, the FIFO error capture function value is updated by the register writes. The next step is to write the Logical/Transport Layer Error Detect CSR to set an enabled error bit. If the Logical/Transport Layer Error Capture FIFO Implemented bit is cleared, this will lock the registers. If the Logical/Transport Layer Error Capture FIFO Implemented bit is set and the FIFO is not full, this will cause the FIFO error capture function value to occupy a FIFO entry. When an error detect bit is set, the hardware will inform the system software of the error using its standard error reporting mechanism. After the error has been reported, the system software may read and clear registers as necessary to complete its error handling protocol testing.

The Port n Error Detect CSR and the five registers starting with the Port n Capture 0 CSR are also writable by software to allow software debug of the system error recovery and thresholding mechanism.

For debug when the Physical Layer Error Capture FIFO Implemented bit is cleared, software must write the Port n Attributes Capture CSR to set the Capture Valid Info bit and then the packet/control symbol information in the five registers starting with the Port n Capture 0 CSR.

For debug when the Physical Layer Error Capture FIFO Implemented bit is set, software must write the five registers starting with the Port n Capture 0 CSR, and then write to the Port n Error Detect CSR to update the FIFO error capture function values for these registers. When the Port n Attributes Capture CSR is written with a value that sets the Capture Valid Info bit, and the FIFO is not full, the FIFO error capture function value shall occupy a FIFO entry.

Each write of a non-zero value to the Port n Error Detect CSR shall cause the Error Rate Counter to increment if the corresponding error bit is enabled in the Port n Error Rate Enable CSR. When a threshold is reached, the hardware will inform the system software of the error using its standard error reporting mechanism. After the error has been reported, the system software may read and clear registers as necessary to complete its error handling protocol testing.

1.6 IDLE3 Port_Status Extension

The Error Management and Hot Swap functions can affect the operation of a port’s input and output directions. Some of these conditions prevent acceptance or transmission of any packets by a processing element, causing errors to be detected by the link partner. In response to these error conditions, the link partner will initiate the standard error recovery protocol as defined in Part 6. Depending on the state of both link partners, software intervention may be necessary to resume packet exchange.

To enable software interrogation of link partner Error Management/Hot Swap status without using packets, the link-response port_status field is extended for IDLE3 devices as defined in Table 1-3. IDLE1 and IDLE2 devices shall not use port_status values other than those defined in Part 6.

Table 1-3. Port_status Field Definitions

Port_status bit number	**Description**
0-2	Reserved
3	Input Port Enabled The conditions defined for this bit to be set are extended as follows: - The Port *n* Control CSRs “Port Lockout” bit is cleared.
4-6	Reserved
7	Output Port Enabled The conditions defined for this bit to be set are extended as follows: - The Port *n* Control CSRs “Port Lockout” bit is cleared.
8	Reserved
9	Output Port Failed. This bit shall be asserted when at least one of the following conditions is true, otherwise de-asserted: - The Port *n* Error and Status CSRs “Output Failed-Encountered” bit is set - The Port *n* Error Detect CSRs “Link Uninit Packet Discard Active” bit is set - An implementation specific condition exists which forces continuous output port packet discard
10-11	Reserved

Chapter 2 Error Management Registers

2.1 Introduction

This section describes the Error Management Extended Features block, and adds a number of new bits to the existing standard physical layer registers. ‘End-point only’ and ‘switch only’ register bits shall be considered reserved when the registers are implemented on devices for which these bits are not required.

2.2 Additions to Existing Registers

2.2.1 Port *n* Control CSRs

The following bits are added to the RapidIO Part 6: LP-Serial Physical Layer Specification Port *n* Control CSRs.

Table 2-1. Bit Settings for Port *n* Control CSRs

Bit	Name	Reset Value	Description
28	Stop on Port Failed-encountered Enable	0b0	This bit is used with the Drop Packet Enable bit to force certain behavior when the Error Rate Failed Threshold has been met or exceeded. See Section 1.2.4 of the Part 8: Error Management Extensions for detailed requirements.
29	Drop Packet Enable	0b0	This bit is used with the Stop on Port Failed-encountered Enable bit to force certain behavior when the Error Rate Failed Threshold has been met or exceeded. See Section 1.2.4 of the Part 8: Error Management Extensions for detailed requirements.
30	Port Lockout	0b0	When this bit is cleared, the packets that may be received and issued are controlled by the state of the Output Port Enable and Input Port Enable bits in the Port *n* Control CSR. When this bit is set, this port is stopped and is not enabled to issue or receive any packets; the input port can still follow the training procedure and can still send and respond to link-requests; all received packets return packet-not-accepted control symbols to force an error condition to be signaled by the sending device

2.2.2 Port *n* Error and Status CSRs

The following bits are added to the RapidIO Part 6: LP-Serial Physical Layer Specification Port *n* Error and Status CSRs.

Table 2-2. Bit Settings for Port *n* Error and Status CSRs

Bit	Name	Reset Value	Description
5	Output Packet-dropped	0b0	Output port has discarded a packet. Once set remains set until written with a logic 1 to clear.
6	Output Failed-encountered	0b0	Output port has encountered a failed condition, meaning that the port's failed error threshold has been reached in the Port *n* Error Rate Threshold register. Once set remains set until written with a logic 1 to clear. Receipt of a reset-port request shall clear this bit to 0. State machines associated with this bit shall be reset to their power-up state.
7	Output Degraded-encountered	0b0	Output port has encountered a degraded condition, meaning that the port's degraded error threshold has been reached in the Port *n* Error Rate Threshold register. Once set remains set until written with a logic 1 to clear.

2.3 New Error Management Registers

This section describes the Extended Features block (EF_ID=0x0007 or EF_ID=0x0017) that allows an external processing element to manage the error status and reporting for a processing element. This chapter only describes registers or register bits defined by this extended features block. All registers are 32-bits and aligned to a 32-bit boundary.

Table 2-3 describes the required behavior for accesses to reserved register bits and reserved registers for the RapidIO Extended Features register space,

Table 2-3. Extended Feature Space Reserved Access Behavior

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

[1]Do not depend on reserved bits being a particular value; use appropriate masks to extract defined bits from the read value.

[2]All register writes shall be in the form: read the register to obtain the values of all reserved bits, merge in the desired values for defined bits to be modified, and write the register, thus preserving the value of all reserved bits.

2.4 Register Map

The register map for the error management registers shall be as specified by Table 2-4. 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] to [EF_PTR+0x13C]. Register map offset [EF_PTR+0x140] can be used for another Extended Features block.

The registers that appear in the Error Management/Hot Swap Extensions Register Block vary based on the functionality indicated in the Error Management/Hot Swap Extensions Block CAR. Table 2-4 describes what registers shall be implemented based on the value of the Error Management/Hot Swap Extensions Block CAR. The register offsets and names are listed, along with three columns that indicate which registers must be implemented. An "X" in the column means that the register shall be implemented.

Table 2-4. Error Management/Hot Swap Extensions Register Requirements

	Block Byte Offset	**Register Name**	**Error Mgmt Only**	**Hot Swap & Error Mgmt**	**Hot Swap Only**
General	0x0	Error Management/Hot Swap Extensions Block Header	X	X	X
	0x4	Error Management/Hot Swap Extensions Block CAR	X	X	X
	0x8	Logical/Transport Layer Error Detect CSR	X	X	-
	0xC	Logical/Transport Layer Error Enable CSR	X	X	-
	0x10	Logical/Transport Layer High Address Capture CSR	X	X	-
	0x14	Logical/Transport Layer Address Capture CSR	X	X	-
	0x18	Logical/Transport Layer Device ID Capture CSR	X	X	-
	0x1C	Logical/Transport Layer Control Capture CSR	X	X	-
	0x20	Logical/Transport Layer Dev32 Destination ID Capture CSR	X	X	-
	0x24	Logical/Transport Layer Dev32 Source ID Capture CSR	X	X	-
	0x28	Port-write Target deviceID CSR	X	X	X
	0x2C	Packet Time-to-live CSR	X	X	X
	0x30	Port-write Dev32 Target deviceID CSR	X	X	X
	0x34	Port-Write Transmission Control CSR	X	X	X
	0x38-3C	Reserved			

Table 2-4. Error Management/Hot Swap Extensions Register Requirements

	Block Byte Offset	Register Name	Error Mgmt Only	Hot Swap & Error Mgmt	Hot Swap Only
Port 0	0x40	Port 0 Error Detect CSR	X	X	X
	0x44	Port 0 Error Rate Enable CSR	X	X	X
	0x48	Port 0 Attributes Capture CSR	X	X	-
	0x4C	Port 0 Capture 0 CSR	X	X	-
	0x50	Port 0 Capture 1 CSR	X	X	-
	0x54	Port 0 Capture 2 CSR	X	X	-
	0x58	Port 0 Capture 3 CSR	X	X	-
	0x5C	Port 0 Capture 4 CSR	X	X	-
	0x60-64	Reserved			
	0x68	Port 0 Error Rate CSR	X	X	-
	0x6C	Port 0 Error Rate Threshold CSR	X	X	-
	0x70	Port 0 Link Uninit Discard Timer CSR	-	X	X
	0x74-78	Reserved			
	0x7C	Port 0 Error Detect FIFO CSR	X	X	
Port 1	0x80	Port 1 Error Detect CSR	X	X	X
	0x84	Port 1 Error Rate Enable CSR	X	X	X
	0x88	Port 1 Attributes Capture CSR	X	X	-
	0x8C	Port 1 Capture 0 CSR	X	X	-
	0x90	Port 1 Capture 1 CSR	X	X	-
	0x94	Port 1 Capture 2 CSR	X	X	-
	0x98	Port 1 Capture 3 CSR	X	X	-
	0x9C	Port 1 Capture 4 CSR	X	X	-
	0xA0-A4	Reserved			
	0xA8	Port 1 Error Rate CSR	X	X	-
	0xAC	Port 1 Error Rate Threshold CSR	X	X	-
	0xB0	Port 1 Link Uninit Discard Timer CSR	-	X	X
	0xB4-B8	Reserved			
	0xBC	Port 1 Error Detect FIFO CSR	X	X	
Ports 2-14	0xC0–3FC	Assigned to Port 2-14 CSRs Register implementation requirements are the same as for port 0			

Table 2-4. Error Management/Hot Swap Extensions Register Requirements

	Block Byte Offset	Register Name	Error Mgmt Only	Hot Swap & Error Mgmt	Hot Swap Only
Port 15	0x400	Port 15 Error Detect CSR	X	X	X
	0x404	Port 15 Error Rate Enable CSR	X	X	X
	0x408	Port 15 Attributes Capture CSR	X	X	-
	0x40C	Port 15 Capture 0 CSR	X	X	-
	0x410	Port 15 Capture 1 CSR	X	X	-
	0x414	Port 15 Capture 2 CSR	X	X	-
	0x418	Port 15 Capture 3 CSR	X	X	-
	0x41C	Port 15 Capture 4 CSR	X	X	-
	0x420-424	Reserved			
	0x428	Port 15 Error Rate CSR	X	X	-
	0x42C	Port 15 Error Rate Threshold CSR	X	X	-
	0x430	Port 15 Link Uninit Discard Timer CSR	-	X	X
	0x434-438	Reserved			
	0x43C	Port 15 Error Detect FIFO CSR	X	X	

2.5 Command and Status Registers (CSRs)

Refer to Table 2-3 for the required behavior for access to reserved registers and register bits.

2.5.1 Error Management Extensions Block Header (Block Offset 0x0)

This register contains the EF_PTR to the next EF_BLK and the EF_ID that identifies this as the error management extensions block header. The use and meaning of the bits shall be as specified in Table 2-5. The register is read-only.

Table 2-5. Bit Settings for Error Management Extensions Block Header

Bit	Name	Reset Value	Description
0-15	EF_PTR		Hard-wired pointer to the next block in the data structure, if one exists
16-31	EF_ID	0x0007 or 0x0017	Devices which implement the standard Error Management Registers, with or without Hot Swap Extensions, shall use an EF_ID value of 0x07. Devices which only implement the Hot Swap Extensions, shall use an EF_ID value of 0x17.

2.5.2 Error Management/Hot Swap Extension Block CAR (Block Offset 0x4)

This register indicates the supported Error Management Extension and Hot Swap Extension functionality. The use and meaning of the bits shall be as specified in Table 2-6. The register is read-only.

Table 2-6. Bit Settings for Error Management/Hot Swap Extension Block CAR

Bit	Name	Reset Value	Description
0	Error Management Extensions Not Implemented	Implementation Specific	Indicates whether or not Error Management Extensions functionality (and registers) is supported. 0b0 - all registers and bit fields specific to Error Management Extensions shall be supported. 0b1 - all registers and/or bit fields specific to Error Management Extensions may not be supported.
1	Hot Swap Extensions Implemented	Implementation Specific	Indicates whether or not Hot Swap functionality and registers are supported. 0b0 - all registers and bit fields specific to Hot Swap Extensions support may not be supported. 0b1 - all registers and bit fields specific to Hot Swap Extensions support shall be supported.

Table 2-6. Bit Settings for Error Management/Hot Swap Extension Block CAR

Bit	Name	Reset Value	Description
2	Physical Layer Error Capture FIFO Implemented	Implementation Specific	Indicates whether or not a FIFO for capture of multiple physical layer events is implemented. 0b0 - all registers, bit fields and behavior specific to FIFO capture of multiple physical layer events may not be supported 0b1 - all registers, bit fields and behavior specific to FIFO capture of multiple physical layer events shall be supported by all ports on the device
3	Logical/Transport Layer Error Capture FIFO Implemented	Implementation Specific	Indicates whether or not a FIFO for capture of multiple logical/transport layer events is implemented. 0b0 - all registers, bit fields and behavior specific to FIFO capture of multiple logical/transport layer events may not be supported 0b1 - all registers, bit fields and behavior specific to FIFO capture of multiple logical/transport layer events shall be supported
4-31	—		Reserved

2.5.3 Logical/Transport Layer Error Detect CSR (Block Offset 0x08)

This register indicates the error that was detected by the Logical or Transport logic layer. Multiple bits may get set in the register if simultaneous errors are detected during the same clock cycle that the errors are logged. The use and meaning of the bits shall be as specified in Table 2-7. Unless otherwise specified, the bits in this register are read/write.

Table 2-7. Bit Settings for Logical/Transport Layer Error Detect CSR

Bit	Name	Reset Value	Description
0	IO error response	0b0	Received a response of ‘ERROR’ for an IO Logical Layer Request. (end point device only)
1	Message error response	0b0	Received a response of ‘ERROR’ for an MSG Logical Layer Request. (end point device only)
2	GSM error response	0b0	Received a response of ‘ERROR’ for a GSM Logical Layer Request. (end point device only)
3	Message Format Error	0b0	Received MESSAGE packet data payload with an invalid size or segment (MSG logical) (end point device only)
4	Illegal transaction decode	0b0	Received a supported request/response packet with undefined field values (IO/MSG/GSM logical) (switch or endpoint device)
5	Illegal transaction target error	0b0	Received a packet that contained a destination ID that is not defined for this processing element. End points with multiple ports and a built-in switch function may not report this as an error (Transport) (optional) (switch or end point device)

Table 2-7. Bit Settings for Logical/Transport Layer Error Detect CSR

Bit	Name	Reset Value	Description
6	Message Request Timeout	0b0	A required message request has not been received within the specified timeout interval (MSG logical) (end point device only)
7	Packet Response Timeout	0b0	A required response has not been received within the specified time out interval (IO/MSG/GSM logical) (end point device only)
8	Unsolicited Response	0b0	An unsolicited/unexpected Response packet was received (IO/MSG/GSM logical; only Maintenance response for switches) (switch or endpoint device)
9	Unsupported Transaction	0b0	A transaction is received that is not supported in the Destination Operations CAR (IO/MSG/GSM logical; only Maintenance port-write for switches) (switch or endpoint device)
10-21	—		Reserved
22	Lost Tick Error Status	0b0	Indicates the current status of the MECS Tick Interval CSR Lost Tick Error Status bit. The Lost Tick Error Status bit must be cleared by writing to the MECS Tick Interval CSR. If the Lost Tick Error Status bit is not defined, this bit is reserved. (end point device only)
23	Lost TSG Sync Error Status	0b0	Indicates the current status of the MECS Tick Interval CSR Lost TSG Sync Error Status bit. The Lost TSG Sync Error Status bit must be cleared by writing to the MECS Tick Interval CSR. If the Lost TSG Sync Error Status bit is not defined, this bit is reserved. (end point device only)
24-31	Implementation Specific error	0x00	An implementation specific error has occurred. (switch or end point device)

2.5.4 Logical/Transport Layer Error Enable CSR (Block Offset 0x0C)

This register contains the bits that control if an error condition locks the Logical/Transport Layer Error Detect and Capture registers and is reported to the system host. Without exception, bit “b” of this register controls the capture and reporting of the error whose occurrence is indicated by bit “b” of the Logical/Transport Layer Error Detect CSR. The use and meaning of the bits shall be as specified in Table 2-8. Unless otherwise specified, the bits in this register are read/write.

Table 2-8. Bit Settings for Logical/Transport Layer Error Enable CSR

Bit	Name	Reset Value	Description
0	IO error response enable	0b0	Enable reporting of an IO 'ERROR' response. Save and lock original request transaction information in all Logical/Transport Layer Capture CSRs. (end point device only)
1	Message error response enable	0b0	Enable reporting of a Message 'ERROR' response. Save and lock original request transaction information in all Logical/Transport Layer Capture CSRs. (end point device only)
2	GSM error response enable	0b0	Enable reporting of a GSM 'ERROR' response. Save and lock original request transaction capture information in all Logical/Transport Layer Capture CSRs. (end point device only)
3	Message Format Error enable	0b0	Enable reporting of a MESSAGE packet data payload with an invalid size or segment (MSG logical). Save and lock transaction capture information in Logical/Transport Layer Device ID and Control Capture CSRs. (end point device only)
4	Illegal transaction decode enable	0b0	Enable reporting of a supported request/response packet with undefined field values (IO/MSG/GSM logical). Save and lock transaction capture information in Logical/Transport Layer Device ID and Control Capture CSRs. (switch or end-point device)
5	Illegal transaction target error enable	0b0	Enable reporting of a packet that contains a destination ID that is not defined for this processing element. Save and lock transaction capture information in Logical/Transport Layer Device ID and Control Capture CSRs. (optional) (switch or end point device)
6	Message Request timeout error enable	0b0	Enable reporting of a Message Request timeout error. Save and lock transaction capture information in Logical/Transport Layer Device ID and Control Capture CSRs for the last Message request segment packet received. (end point device only)
7	Packet Response Timeout error enable	0b0	Enable reporting of a packet response timeout error. Save and lock original request address in Logical/Transport Layer Address Capture CSRs. Save and lock original request Destination ID in Logical/Transport Layer Device ID Capture CSR. (end point device only)
8	Unsolicited Response error enable	0b0	Enable reporting of receiving an unsolicited/unexpected Response packet (IO/MSG/GSM logical; only Maintenance responses for switches). Save and lock transaction capture information in Logical/Transport Layer Device ID and Control Capture CSRs. (switch or end-point device)
9	Unsupported Transaction error enable	0b0	Enable reporting of an unsupported transaction error. Save and lock transaction capture information in Logical/Transport Layer Device ID and Control Capture CSRs. (switch or end-point device)
10-21	—		Reserved
22	Lost Tick Error Enable	0b0	Enable reporting of the current status of the MECS Tick Interval CSR Lost Tick Error Status bit. The Logical/Transport Layer Device ID and Control Capture CSRs shall not lock when this error is detected. If the Lost Tick Error Status bit is not defined, this bit is reserved. (end point device only)

Table 2-8. Bit Settings for Logical/Transport Layer Error Enable CSR

Bit	Name	Reset Value	Description
23	Lost TSG Sync Error Enable	0b0	Enable reporting of the MECS Tick Interval CSR Lost TSG Sync Error Status bit. The Logical/Transport Layer Device ID and Control Capture CSRs shall not lock when this error is detected. If the Lost TSG Sync Error Status bit is not defined, this bit is reserved. (end point device only)
24-31	Implementation Specific error enable	0x00	Enable reporting of an implementation specific error has occurred. Save and lock capture information in appropriate Logical/Transport Layer Capture CSRs.

2.5.5 Logical/Transport Layer High Address Capture CSR (Block Offset 0x10)

This register contains error information. It is locked when a Logical/Transport error is detected and the corresponding enable bit is set. This register is required for end point devices that support 66 or 50 bit addresses, and for all switch devices which support operation with IDLE3. The use and meaning of the bits shall be as specified in Table 2-9. Unless otherwise specified, the bits in this register are read/write.

Table 2-9. Bit Settings for Logical/Transport Layer High Address Capture CSR

Bit	Name	Reset Value	Description
0-31	address[0-31]	All 0s	For switch devices, this field may capture implementation specific information for detected logical/transport layer errors. When an endpoint detects an error related to a Logical I/O packet, this field may capture implementation specific information when the address size is 34 bits or less. When an endpoint detects an error related to a Logical I/O packet and the address size is 50 bits, bits 0 to 15 of this field may capture implementation specific information and bits 16 to 31 of this field shall contain address bits 2-17. When an endpoint detects an error related to a Logical I/O packet and the address size is 66 bits, this field shall capture address bits 2-33. For all other logical/transport layer errors, this field may contain implementation specific information.

2.5.6 Logical/Transport Layer Address Capture CSR (Block Offset 0x14)

This register contains error information. It is locked when a Logical/Transport error is detected and the corresponding enable bit is set. The use and meaning of the bits shall be as specified in Table 2-10. Unless otherwise specified, the bits in this register are read/write.

Table 2-10. Bit Settings for Logical/Transport Layer Address Capture CSR

Bit	Name	Reset Value	Description
0-28	address[32-60]	All 0s	When a logical/transport error is detected for a maintenance packet which has an offset, the offset is found in the least significant 21 bits of this field. Otherwise, when a Logical/transport layer error is detected by a switch device, this field may contain implementation specific information. When an endpoint detects an error with a Logical I/O transaction, this field contains the least significant 29 bits of the address field. When an endpoint detects and error for any other transaction type, this field may contain implementation specific information.
29	Implementation Specific	0b0	This field may contain implementation specific information.
30-31	xamsbs	0b00	When an endpoint detects an error with a logical I/O transaction, this field shall contain the extended address bits of the address associated with the error (for requests, for responses if available). For all other errors detected by an endpoint, and in switch devices, this field may contain implementation specific information.

2.5.7 Logical/Transport Layer Device ID Capture CSR (Block Offset 0x18)

This register contains error information. It is locked when an error is detected and the corresponding enable bit is set. The use and meaning of the bits shall be as specified in Table 2-11. Unless otherwise specified, the bits in this register are read/write.

Table 2-11. Bit Settings for Logical/Transport Layer Device ID Capture CSR

Bit	Name	Reset Value	Description
0-7	MSB destinationID	0x00	Most significant byte of the destinationID associated with the error (Dev16 systems only)
8-15	destinationID	0x00	The destinationID associated with the error (Dev8 and Dev16 systems only)
16-23	MSB sourceID	0x00	Most significant byte of the sourceID associated with the error (Dev16 systems only)
24-31	sourceID	0x00	The sourceID associated with the error (Dev8 and Dev16 systems only)

2.5.8 Logical/Transport Layer Control Capture CSR (Block Offset 0x1C)

This register contains error information. It is locked when a Logical/Transport error is detected and the corresponding enable bit is set. The use and meaning of the bits shall be as specified in Table 2-12. Unless otherwise specified, the bits in this register are read/write.

Table 2-12. Bit Settings for Logical/Transport Layer Control Capture CSR

Bit	Name	Reset Value	Description
0-3	ftype	0x0	Format type associated with the error
4-7	ttype	0x0	Transaction type associated with the error. If the format type does not include a ttype field, this field may contain implementation specific information.
8-15	msg info	0x00	letter, mbox, and msgseg for the last Message request received for the mailbox that had an error (Message errors only). For non-Message errors, this field may contain implementation specific information.
16-31	Implementation specific	0x0000	Implementation specific information associated with the error

2.5.9 Logical/Transport Layer Dev32 Destination ID Capture CSR (Block Offset 0x20)

This register contains error information. It is locked when a Logical/Transport error is detected and the corresponding enable bit is set. This register shall be implemented for devices that have bit 19 (Dev32 Support) set in the Processing Element Features CAR. The use and meaning of the bits shall be as specified in Table 2-13. This register is read/write.

Table 2-13. Bit Settings for Logical/Transport Layer Dev32 Destination ID Capture CSR

Bit	Name	Reset Value	Description
0-31	Dev32 DestID	All 0’s	The Dev32 destination ID associated with the error.

2.5.10 Logical/Transport Layer Dev32 Source ID Capture CSR (Block Offset 0x24)

This register contains error information. It is locked when a Logical/Transport error is detected and the corresponding enable bit is set. This register shall be implemented for devices that have bit 19 (Dev32 Support) set in the Processing Element Features CAR. The use and meaning of the bits shall be as specified in Table 2-14. This register is read/write.

Table 2-14. Bit Settings for Logical/Transport Layer Dev32 Source ID Capture CSR

Bit	Name	Reset Value	Description
0-31	Dev32 SrcID	All 0’s	The Dev32 source ID associated with the error.

2.5.11 Port-Write Target deviceID CSR (Block Offset 0x28)

This register contains the target Dev8 or Dev16 deviceID to be used when a device generates a Maintenance port-write operation to report errors to a system host. The use and meaning of the bits shall be as specified in Table 2-15. Unless otherwise specified, the bits in this register are read/write.

Table 2-15. Bit Settings for Port-Write Target deviceID CSR

Bit	Name	Reset Value	Description
0-7	Dev16_deviceID_msb	0x00	This is the most significant byte of the port-write target deviceID (Dev16 deviceIDs only).
8-15	Dev8_deviceID	0x00	This is the port-write target Dev8 deviceID, or least significant byte of the Dev16 deviceID.
16	Dev8_or_16	0b0	Dev8 or Dev16 deviceID size to use for a port-write 0b0 - Port-writes originated by this device shall use Dev8 deviceIDs 0b1 - Port-writes originated by this device shall use Dev16 deviceIDs This bit field controls the deviceID size to use for a port-write when Dev32_PW is 0.
17	Dev32_PW	0b0	This bit field shall be implemented for devices that have bit 19 (Dev32 Support) set in the Processing Element Features CAR. This bit field controls the use of Dev32 deviceID size for a port-write 0b0 - Port-write deviceID size shall be controlled by the Dev8_or_16 field 0b1 - Port-writes originated by this device shall use the Dev32 deviceID defined in the 2.5.13, “Port-write Dev32 Target deviceID CSR
18-31	—		Reserved

2.5.12 Packet Time-to-live CSR (Block Offset 0x2C)

The Packet Time-to-live register specifies the length of time that a packet is allowed to exist within a switch device. The maximum value of the Time-to-live variable (0xFFFF) shall correspond to 100 msec. +/-34%. The resolution (minimum step size) of the Time-to-live variable shall be (maximum value of Time-to-live)/(2^{16}-1). The reset value is all logic 0s, which disables the Time-to-live function so that a packet never times out. This register is not required for devices without switch functionality. The use and meaning of the bits shall be as specified in Table 2-16. Unless otherwise specified, the bits in this register are read/write.

Table 2-16. Bit Settings for Packet Time-to-live CSR

Bit	Name	Reset Value	Description
0-15	Time-to-live value	0x0000	Maximum time that a packet is allowed to exist within a switch device
16-31	—		Reserved

2.5.13 Port-write Dev32 Target deviceID CSR (Block Offset 0x30)

This register contains the Dev32 target deviceID to be used when a device generates a Maintenance port-write operation to report errors to a system host and the Dev32_PW bit is set. This register shall be implemented for devices that have bit 19 (Dev32 Support) set in the Processing Element Features CAR. The use and meaning of the bits shall be as specified in Table 2-17. Unless otherwise specified, the bits in this register are read/write.

Table 2-17. Bit Settings for Port-Write Dev32 Target deviceID CSR

Bit	Name	Reset Value	Description
0-31	Dev32_deviceID	All 0’s	The port-write Dev32 target device ID.

2.5.14 Port-Write Transmission Control CSR (Block Offset 0x34)

The Port-Write transmission control CSR determines whether port-write notification is enabled or disabled for the device. The use and meaning of the bits shall be as specified in Table 2-18. Unless otherwise specified, the bits in this register are read/write.

Table 2-18. Bit Settings for Port-Write Transmission Control CSR

Bit	Name	Reset Value	Description
0-30	—		Reserved
31	Port-write Transmission Disable	0b0	0 - Enabled events for the device shall cause port-writes to be generated 1 - Enabled events for the device shall not cause new port-writes to be generated. Previously generated port-writes may be transmitted after this bit is set.

2.5.15 Port *n* Error Detect CSR (Block Offset 0x40, 80,..., 400)

The Port *n* Error Detect Register indicates the physical layer errors that have been detected by the Port *n* hardware since the register was last cleared. The register is cleared by software writing the register with the data 0x0000_0000.

The use and meaning of the bits shall be as specified in Table 2-19. Unless otherwise specified, the bits in this register are read/write.

The two right-most columns in Table 2-19 indicate which bit fields must be implemented for Error Management Extensions and Hot Swap Extensions. An “X” in these columns means that the bit shall be implemented for that extension.

Table 2-19. Bit Settings for Port *n* Error Detect CSRs

Bit	Name	Reset Value	Description	Error Mgmt	Hot Swap
0	Implementation specific error	0b0	An implementation specific error has been detected	X	-
1	Link OK to Uninit Transition	0b0	The link has transitioned from a link initialized to link uninitialized state.	-	X
2	Link Uninit Packet Discard Active	0b0	The Link Uninit Discard Timer CSR period has expired.	-	X
3	Link Uninit to OK Transition	0b0	The link has transitioned from a link uninitialized to link initialized state.	-	X
4-7	—		Reserved		
8	Deprecated	0b0	Deprecated	X	-
9	Received corrupt control symbol	0b0	Received a control symbol with a bad CRC value or Received an incorrect sequence of control symbol codewords (for example, CSE or CSEB without a preceding CSB, or a CSB or CSEB that is not followed by a CSE or CSEB (IDLE3)	X	-
10	Received acknowledge control symbol with unexpected ackID	0b0	Received a packet-accepted or packet-retry control symbol with an unexpected ackID. "Error Recovery with ackID in PNA Enabled" is set in the Port n Latency Optimization CSR and a packet-not-accepted control symbol is received with an unexpected ackID.	X	-

Table 2-19. Bit Settings for Port *n* Error Detect CSRs

Bit	Name	Reset Value	Description	Error Mgmt	Hot Swap
11	Received packet-not-accepted control symbol	0b0	Received packet-not-accepted control symbol	X	-
12	Received packet with unexpected ackID	0b0	Received packet with unexpected ackID value - out-of-sequence ackID	X	-
13	Received packet with bad CRC	0b0	Received packet with a bad CRC value	X	-
14	Received packet exceeds maximum allowed size	0b0	Received packet that exceeds the maximum allowed size	X	-
15	Received illegal or invalid character	0b0	Received an 8b/10b code-group that is invalid (a code-group that does not have a 8b/10b decode given the current running disparity) or illegal (a code-group that is valid but whose use is not allowed by the LP-Serial protocol). This bit may be set in conjunction with bit 29, Delineation error. The implementation of this bit is optional, but strongly recommended. (IDLE1 and IDLE2) or Bit Interleaved Parity (BIP) check failed on at least one lane (IDLE3)	X	-
16	Received data character in IDLE1 sequence	0b0	Received a data character in an IDLE1 sequence. This bit may be set in conjunction with bit, 29 Delineation error. The implementation of this bit is optional, but strongly recommended.	X	-
17	Loss of descrambler synchronization	0b0	Loss of receiver descrambler synchronization while receiving scrambled control symbol and packet data. This bit shall be implemented only if port n supports descrambling of packet and control symbol data.	X	-
18	Invalid Ordered Sequence	0b0	Received an invalid ordered sequence on at least one lane (IDLE3)	X	-
19-25	—		Reserved		
26	Non-outstanding ackID	0b0	When there are outstanding ackIDs, a link_response was received with an ackID that is not outstanding	X	-
27	Protocol error	0b0	An unexpected control symbol was received	X	-
28	Deprecated	0b0	Deprecated	X	-
29	Delineation error	0b0	Received an 8b/10b code-group that is invalid (a code-group that does not have a 8b/10b decode given the current running disparity), illegal (a code-group that is valid, but whose use is not allowed by the LP-Serial protocol) or that is in a position in the received code-group stream that is not allowed by the LP-Serial protocol (IDLE1 and IDLE2) or Received a 64b/67b codeword whose type and !type bit values are the same (IDLE3)	X	-
30	Unsolicited acknowledgement control symbol	0b0	An unsolicited packet acknowledgement control symbol was received	X	-

Table 2-19. Bit Settings for Port *n* Error Detect CSRs

Bit	Name	Reset Value	Description	Error Mgmt	Hot Swap
31	Link timeout	0b0	A packet acknowledgement or link-response control symbol was not received within the specified timeout interval	X	-

2.5.16 Port *n* Error Rate Enable CSR (Block Offset 0x44, 84,..., 404)

The two right-most columns of Table 2-20 indicate which bit fields must be implemented for Error Management Extensions and Hot Swap Extensions. An “X” in these columns means that the bit shall be implemented for that extension.

If Hot Swap Extension is supported in this register, the Hot Swap bits enable the event notification and packet discard for the detected event.

When the Error Management Extension is supported, the Error Management bits when set cause specific detected errors to increment the error rate counter in the Port *n* Error Rate Threshold Register and capture information about the error in, and then lock, the Port *n* Capture 0-4 CSRs. Without exception, bit “b” of this register controls the capture and counting of the detected error whose occurrence is indicated by bit “b” of the Port *n* Error Detect CSR.

The use and meaning of the bits shall be as specified in Table 2-20. Unless otherwise specified, the bits in this register are read/write.

Table 2-20. Bit Settings for Port *n* Error Rate Enable CSRs

Bit	Name	Reset Value	Description	Error Mgmt	Hot Swap
0	Implementation specific error enable	0b0	Enable error rate counting of implementation specific errors	X	-
1	Link OK to Uninit Transition Enable	0b0	Enable event notification for when the link has transitioned from a link initialized to link uninitialized state.	-	X
2	Link Uninit Packet Discard Active Enable	0b0	Enable event notification for Link Uninit Packet Discard Timer events.	-	X
3	Link Uninit to OK Transition Enable	0b0	Enable event notification for when the link has transitioned from a link uninitialized to link initialized state.	-	X
4-7	—		Reserved		
8	Deprecated	0b0	Deprecated	X	-
9	Received corrupt control symbol enable	0b0	Enable error rate counting of a corrupt control symbol	X	-

Table 2-20. Bit Settings for Port *n* Error Rate Enable CSRs

Bit	Name	Reset Value	Description	Error Mgmt	Hot Swap
10	Received acknowledge control symbol with unexpected ackID enable	0b0	Enable error rate counting of: - packet-accepted or packet-retry control symbol is received with an unexpected ackID - If "Error Recovery with ackID in PNA Enabled" is set in the Port n Latency Optimization CSR and a packet-not-accepted control symbol is received with an unexpected ackID	X	-
11	Received packet-not-accepted control symbol enable	0b0	Enable error rate counting of received packet-not-accepted control symbols.	X	-
12	Received packet with unexpected ackID enable	0b0	Enable error rate counting of packet with unexpected ackID value - out-of-sequence ackID	X	-
13	Received packet with bad CRC enable	0b0	Enable error rate counting of packet with a bad CRC value	X	-
14	Received packet exceeds maximum allowed size enable	0b0	Enable error rate counting of packet that exceeds the maximum allowed size	X	-
15	Received illegal or invalid character enable	0b0	Enable error rate counting of reception of illegal or invalid characters (IDLE1 or IDLE2), or failed BIP check (IDLE3). This bit shall be implemented only if bit 15 (Received illegal or invalid character) of the Port *n* Error Detect CSR is implemented.	X	-
16	Received data character in an IDLE1 sequence enable	0b0	Enable error rate counting of reception of a data character in an IDLE1 sequence. This bit shall be implemented only if bit 16 (Received data character in IDLE1 sequence) of the Port *n* Error Detect CSR is implemented.	X	-
17	Loss of descrambler synchronization enable	0b0	Enable error rate counting of loss of receiver descrambler synchronization when scrambled control symbol and packet data is being received. This bit shall be implemented only if bit 17 (Loss of descrambler synchronization) of the Port *n* Error Detect CSR is implemented.	X	-
18	Invalid ordered sequence enable	0b0	Enable error rate counting of invalid ordered sequence reception for all enabled lanes. This bit shall be implemented only if bit 18 (Invalid ordered sequence) of the Port *n* Error Detect CSR is implemented. (IDLE3)	X	-
19-25	—		Reserved		
26	Non-outstanding ackID enable	0b0	Enable error rate counting of link-responses received with an ackID that is not outstanding when there are outstanding ackIDs	X	-
27	Protocol error enable	0b0	Enable error rate counting of received unexpected control symbol symbols	X	-
28	Deprecated	0b0	Deprecated	X	-

Table 2-20. Bit Settings for Port *n* Error Rate Enable CSRs

Bit	Name	Reset Value	Description	Error Mgmt	Hot Swap
29	Delineation error enable	0b0	Enable error rate counting of:- Reception of an 8b/10b code-group that is invalid, illegal, or is in a position in the received code-group stream that is not allowed by the LP-Serial protocol (IDLE1 and IDLE2) - Reception of 64b/67b codeword whose type and !type bits are the same (IDLE3)	X	-
30	Unsolicited acknowledgement control symbol enable	0b0	Enable error rate counting of received unsolicited packet acknowledgement control symbols	X	-
31	Link timeout enable	0b0	Enable error rate counting of link timeout errors	X	-

2.5.17 Port *n* Attributes Capture CSR (Block Offset 0x48, 88,..., 408)

This register indicates the type of information captured in the Port *n* Capture 0-4 CSRs. If multiple errors are detected during the same clock cycle, one of the detected errors shall be selected for capture and the captured error shall be indicated in the Error type field. When multiple errors are detected during the same clock cycle, the error selected for capture is implementation specific.

The use and meaning of the bits shall be as specified in Table 2-21. Bits 0-30 of this register shall be valid and locked when the Capture valid info bit is set. Unless otherwise specified, the bits in this register are read/write.

Table 2-21. Bit Settings for Port *n* Attributes Capture CSRs

Bit	Name	Reset Value	Description
0-2	Info type	0b000	Type of information logged 0b000 - packet 0b001 - reserved 0b010 - Control Symbol 24 0b011 - Control Symbol 48 0b100 - implementation specific (capture register contents are implementation specific) 0b101 - Control Symbol 64 0b110 - deprecated 0b111 - reserved
3-7	Error type	0x00	The encoded value of the bit in the Port *n* Error Detect CSR that describes the error captured in the Port *n* Capture 0-4 CSRs.
8-27	Implementation Dependent	All 0s	Implementation specific error information. If port *n* is operating with IDLE1 or IDLE2, the following should be implemented: • If the Info_type is “packet”, the “control” bits of packet characters 0-15 should be captured in bits 8-23, respectively. • If the Info_type is “Control Symbol 24”, the “control” bits of delimited control symbol characters 0-3 should be captured in bits 8-11, respectively. • If the Info_type is “Control Symbol 48 symbol”, the “control” bits of delimited control symbol characters 0-7 should be captured in bits 8-15, respectively. If port *n* is operating with IDLE3, the following should be implemented: • The inverted, !type and type bits of the codewords that are captured in the Port *n* Capture 0-4 CSRs should be captured here.
28-30	—		Reserved
31	Capture valid info	0b0	This bit is set by hardware to indicate that the Port *n* Capture 0-4 CSRs and the other bits in this register contain valid information and are locked. This bit is cleared and the Port *n* Capture 0-4 CSRs and the other bits in this register are unlocked when software writes 0b0 to the bit.

2.5.18 Port *n* Capture 0 CSR (Block Offset 0x4C, 8C,..., 40C)

The use and meaning of the bits shall be as specified in Table 2-22. The contents of the register are valid and locked when the Capture Valid Info bit of the Port *n* Attributes Capture CSR is set (0b1). Unless otherwise specified, the bits in this register are read/write.

Table 2-22. Bit Settings for Port *n* Capture 0 CSRs

Bit	Name	Reset Value	Description
0-31	Capture 0	All 0s	If the info_type field of the Port n Attributes Capture CSR is “Control Symbol 24”, “Control Symbol 48” or “Control Symbol 64”, Delimited Control Symbol 24, Delimitted Control Symbol 48 bytes 0-3, or Control Symbol 64 bytes 0-3. If the info_type field of the Port *n* Attributes Capture CSR is “packet”, packet bytes 0-3. Otherwise, implementation specific.

2.5.19 Port *n* Capture 1 CSR (Block Offset 0x50, 90,..., 410)

The use and meaning of the bits shall be as specified in Table 2-23. The contents of the register are valid and locked when the Capture Valid Info bit of the Port *n* Attributes Capture CSR is set (0b1). Unless otherwise specified, the bits in this register are read/write.

Table 2-23. Bit Settings for Port *n* Capture 1 CSRs

Bit	Name	Reset Value	Description
0-31	Capture 1	All 0s	If the info_type field of the Port n Attributes Capture CSR is “Control Symbol 48” or “Control Symbol 64”, delimited control symbol Bytes 4-7. If the info_type field of the Port n Attributes Capture CSR is “packet”, packet Bytes 4-7. Otherwise, implementation specific.

2.5.20 Port *n* Capture 2 CSR (Block Offset 0x54, 94,..., 414)

The use and meaning of the bits shall be as specified in Table 2-24. The contents of the register are valid and locked when the Capture Valid Info bit of the Port *n* Attributes Capture CSR is set (0b1). Unless otherwise specified, the bits in this register are read/write.

Table 2-24. Bit Settings for Port *n* Capture 2 CSRs

Bit	Name	Reset Value	Description
0-31	Capture 2	All 0s	If the info_type field of the Port n Attributes Capture CSR is “packet”, packet Bytes 8-11. Otherwise, implementation specific.

2.5.21 Port *n* Capture 3 CSR (Block Offset 0x58, 98,..., 418)

The use and meaning of the bits shall be as specified in Table 2-25. The contents of the register are valid and locked when the Capture Valid Info bit of the Port *n* Attributes Capture CSR is set (0b1). Unless otherwise specified, the bits in this register are read/write.

Table 2-25. Bit Settings for Port *n* Capture 3 CSRs

Bit	Name	Reset Value	Description
0-31	Capture 3	All 0s	If the info_type field of the Port n Attributes Capture CSR is “packet”, packet Bytes 12-15. Otherwise, implementation specific.

2.5.22 Port *n* Capture 4 CSR (Block Offset 0x5C, 9C,..., 41C)

The use and meaning of the bits shall be as specified in Table 2-26. This register shall be implemented for devices that have bit 19 (Dev32 Support) set in the Processing Element Features CAR. The contents of the register are valid and locked when the Capture Valid Info bit of the Port *n* Attributes Capture CSR is set. Unless otherwise specified, the bits in this register are read/write.

Table 2-26. Bit Settings for Port *n* Capture 4 CSRs

Bit	Name	Reset Value	Description
0-31	Capture 4	All 0s	If the info_type field of the Port n Attributes Capture CSR is “packet”, packet Bytes 16-19. Otherwise, implementation specific.

2.5.23 Port *n* Error Rate CSR (Block Offset 0x68, A8,..., 428)

This register is used in conjunction with the Port *n* Error Rate Threshold register to monitor and control the reporting of Port *n* physical layer errors. The use and meaning of the bits shall be as specified in Table 2-27. Unless otherwise specified, the bits in this register are read/write.

Table 2-27. Bit Settings for Port *n* Error Rate CSRs

Bit	Name	Reset Value	Description
0-7	Error Rate Bias	0x80	This field specifies the rate at which the Error Rate Counter is decremented (the error rate bias value) 0x00 - do not decrement the error rate counter 0x01 - decrement every 1ms (+/-34%) 0x02 - decrement every 10ms (+/-34%) 0x04 - decrement every 100ms (+/-34%) 0x08 - decrement every 1s (+/-34%) 0x10 - decrement every 10s (+/-34%) 0x20 - decrement every 100s (+/-34%) 0x40 - decrement every 1000s (+/-34%) 0x80 - decrement every 10000s (+/-34%) other values are reserved
8-13	—		Reserved
14-15	Error Rate Recovery	0b00	The value of this field limits the incrementing of the Error Rate Counter above the failed threshold trigger. 0b00 - only count 2 errors above 0b01 - only count 4 errors above 0b10 - only count 16 error above 0b11 - do not limit incrementing the error rate count
16-23	Peak Error Rate	0x00	This field contains the peak value attained by the error rate counter since the field was last reset.
24-31	Error Rate Counter	0x00	This field contains a count of the number of physical layer errors that have been detected by the port, decremented by the Error Rate Bias mechanism, to create an indication of the physical layer error rate. Receipt of a reset-port request shall clear this field to 0x00. State machines associated with this field shall be reset to their power-up state.

2.5.24 Port *n* Error Rate Threshold CSR (Block Offset 0x6C, AC, ..., 42C)

This register controls the reporting of the link status to the system host. The use and meaning of the bits shall be as specified in Table 2-28. Unless otherwise specified, the bits in this register are read/write.

Table 2-28. Bit Settings for Port *n* Error Rate Threshold CSRs

Bit	Name	Reset Value	Description
0-7	Error Rate Failed Threshold Trigger	0xFF	This field contains the threshold value for reporting an error condition due to a possible broken link. 0x00 - Disable the Error Rate Failed Threshold Trigger 0x01 - Set the error reporting threshold to 1 0x02 - Set the error reporting threshold to 2 ... 0xFF - Set the error reporting threshold to 255
8-15	Error Rate Degraded Threshold Trigger	0xFF	This field contains the threshold value for reporting an error condition due to a degrading link. 0x00 - Disable the Error Rate Degraded Threshold Trigger 0x01 - Set the error reporting threshold to 1 0x02 - Set the error reporting threshold to 2 ... 0xFF - Set the error reporting threshold to 255
16-31	—		Reserved

2.5.25 Port *n* Link Uninit Discard Timer CSR (Block Offset 0x70, 0xB0, ..., 0x430)

The maximum value of the Link Uninit Timeout variable (0xFFFFFF) shall correspond to 6 to 12 seconds. The resolution of the Link Uninit Timeout variable shall be (maximum Link Uninit Timeout interval)/(2^{24}-1). The use and meaning of the bits shall be as specified in Table 2-29. Unless otherwise specified, the bits in this register shall be readable and writable.

Table 2-29. Bit Settings for Port n Link Uninit Discard Timer CSRs

Bit	Name	Reset Value	Description
0-23	Link Uninit Timeout	0x000000	On IDLE1 and IDLE2 links, this timer shall start counting when link_initialized is deasserted, and shall continue counting until link_initialized is asserted. On IDLE3 links this timer shall start when at least one of link_initialized, receive_enable, or transmit_enable is deasserted, and shall continue counting until link_initialized, receive_enable and transmit enable are all asserted. When this timer expires, all packets directed to this port from inside the device shall be discarded, and a “Link Uninit Packet Discard Active” event shall be detected. Packet discard shall occur until the “Link Uninit Packet Discard Active” status bit is cleared. The Link Uninit Discard Timer shall be disabled when Link Uninit Timeout is 0.
24-31	—		Reserved

2.5.26 Port *n* FIFO Error Detect CSR (Block Offset 0x7C, BC,..., 43C)

The Port *n* FIFO Error Detect Register shall be implemented when bit 2 of the Error Management/Hot Swap Extension Block CAR is set. The behavior of the register shall be as defined in Section 1.2.7 on page 14.

The use and meaning of the bits shall be as specified in Table 2-30. Unless otherwise specified, the bits in this register are read/write.

The two right-most columns in Table 2-30 indicate which bit fields must be implemented for Error Management Extensions and Hot Swap Extensions. An “X” in these columns means that the bit shall be implemented for that extension.

Table 2-30. Bit Settings for Port *n* FIFO Error Detect CSRs

Bit	Name	Reset Value	Description	Error Mgmt	Hot Swap
0	Implementation specific error	0b0	An implementation specific error has been detected	X	-
1	Link OK to Uninit Transition	0b0	The link has transitioned from a link initialized to link uninitialized state.	-	X

Table 2-30. Bit Settings for Port *n* FIFO Error Detect CSRs

Bit	Name	Reset Value	Description	Error Mgmt	Hot Swap
2	Link Uninit Packet Discard Active	0b0	The Link Uninit Discard Timer CSR period has expired.	-	X
3	Link Uninit to OK Transition	0b0	The link has transitioned from a link uninitialized to link initialized state.	-	X
4-7	—		Reserved		
8	Deprecated	0b0	Deprecated	X	-
9	Received corrupt control symbol	0b0	Received a control symbol with a bad CRC value or Received an incorrect sequence of control symbol codewords (for example, CSE or CSEB without a preceding CSB, or a CSB or CSEB that is not followed by a CSE or CSEB (IDLE3)	X	-
10	Received acknowledge control symbol with unexpected ackID	0b0	Received a packet-accepted or packet-retry control symbol with an unexpected ackID. "Error Recovery with ackID in PNA Enabled" is set in the Port n Latency Optimization CSR and a packet-not-accepted control symbol is received with an unexpected ackID.	X	-
11	Received packet-not-accepted control symbol	0b0	Received packet-not-accepted control symbol	X	-
12	Received packet with unexpected ackID	0b0	Received packet with unexpected ackID value - out-of-sequence ackID	X	-
13	Received packet with bad CRC	0b0	Received packet with a bad CRC value	X	-
14	Received packet exceeds maximum allowed size	0b0	Received packet that exceeds the maximum allowed size	X	-
15	Received illegal or invalid character	0b0	Received an 8b/10b code-group that is invalid (a code-group that does not have a 8b/10b decode given the current running disparity) or illegal (a code-group that is valid but whose use is not allowed by the LP-Serial protocol). This bit may be set in conjunction with bit 29, Delineation error. The implementation of this bit is optional, but strongly recommended. (IDLE1 and IDLE2) or Bit Interleaved Parity (BIP) check failed on at least one lane (IDLE3)	X	-
16	Received data character in IDLE1 sequence	0b0	Received a data character in an IDLE1 sequence. This bit may be set in conjunction with bit, 29 Delineation error. The implementation of this bit is optional, but strongly recommended.	X	-
17	Loss of descrambler synchronization	0b0	Loss of receiver descrambler synchronization while receiving scrambled control symbol and packet data. This bit shall be implemented only if port n supports descrambling of packet and control symbol data.	X	-
18	Invalid Ordered Sequence	0b0	Received an invalid ordered sequence on at least one lane (IDLE3)	X	-
19-25	—		Reserved		

Table 2-30. Bit Settings for Port *n* FIFO Error Detect CSRs

Bit	Name	Reset Value	Description	Error Mgmt	Hot Swap
26	Non-outstanding ackID	0b0	When there are outstanding ackIDs, a link_response was received with an ackID that is not outstanding	X	-
27	Protocol error	0b0	An unexpected control symbol was received	X	-
28	Deprecated	0b0	Deprecated	X	-
29	Delineation error	0b0	Received an 8b/10b code-group that is invalid (a code-group that does not have a 8b/10b decode given the current running disparity), illegal (a code-group that is valid, but whose use is not allowed by the LP-Serial protocol) or that is in a position in the received code-group stream that is not allowed by the LP-Serial protocol (IDLE1 and IDLE2) or Received a 64b/67b codeword whose type and !type bit values are the same (IDLE3)	X	-
30	Unsolicited acknowledgement control symbol	0b0	An unsolicited packet acknowledgement control symbol was received	X	-
31	Link timeout	0b0	A packet acknowledgement or link-response control symbol was not received within the specified timeout interval	X	-

Annex A Error Management Discussion (Informative)

A.1 Introduction

This section is intended to provide useful information/background on the application of the error management capabilities. This section is a guideline, not part of the specification.

A.2 Limitations of Error Management Discussion

The RapidIO hardware that implements the Error Management extensions is able to log physical layer errors and errors that occur at a higher level. Some error scenarios require no software intervention and recovery procedures are done totally by the hardware.

Some error scenarios detected require fault management software for recovery to be successful. For example, some types of logical layer errors on a Read or Write operation may be recoverable by killing the software process using the affected memory space and removing the memory space from the available system resource pool. It may also be possible for software to retry the operation, possibly through a different path in the switch fabric. Since such fault management software is typically tightly coupled to a particular system and/or implementation, it is considered outside of the scope of this specification.

Another area of fault recovery that requires fault management software to be implemented is correcting of system state after an error during an atomic operation. The swap style Atomic operations are possibly recoverable through software and require software convention to uniquely identify attempts to take locks. For example, if the request is lost and times out, software can examine the current lock value to determine if the request or the associated response was the transaction that was lost in the switch fabric. For all other Atomic operations (such as the Atomic set operation), it is impossible to correct the system state in the presence of a ‘lost packet’ type of error.

The use of RapidIO message packets relies on the use of higher layer protocols for error management. Since end points that communicate via messaging are typically running a variety of higher layer protocols, error reporting of both request and response timeouts is done locally by the message queue management controller.

Note that side effect errors can occur, for example, ERROR responses or RETRY responses during an active (partially completed) message, which may complicate the recovery procedure. The recovery strategies for messages lost in this manner are outside of the scope of this specification.

Globally Shared Memory systems that encounter a logical or transport layer error are typically not recoverable by any mechanism as this usually means that the processor caches are no longer coherent with the main memory system. Historically, recovery from such errors requires a complete reboot of the machine after the component that caused the error is repaired or replaced.

A.3 Hot-insertion/extraction Discussion

Hot-insertion can be regarded as an error condition in which a new part of the system is detected, therefore, hot-insertion of a Field Replaceable Unit (FRU) can be handled utilizing the above described mechanisms. This section describes two approaches for hot insertion. The first generally applies to high availability systems, or systems where FRUs need to brought into the system in a controlled manner. The second generally applies to systems where availability is less of a concern, for example, a trusted system or a system without a system host.

At system boot time, the system host identifies all of the unattached links in the machine through system discovery and puts them in a locked mode, whereby all incoming packets are to be rejected, leaving the drivers and receivers enabled. This is done by setting the Discovered bit in the Port General Control CSR and the Port Lockout bit in the Port *n* Control CSR. Note that whenever an FRU is removed, the port lockout bit should be used to ensure that whatever new FRU is inserted cannot access the system until the system host allows it. When a FRU is hot-inserted connecting to a switch device, the now connected link will automatically start the training sequence. When training is complete (the Port OK bit in the Port *n* Error and Status CSR is now set), the locked port generates a Maintenance port-write operation to notify the system host of the new connection, and sets the Port-write Pending bit.

On receipt of the port-write, the system host is responsible for bringing the inserted FRU into the system in a controlled manner. The system host can communicate with the inserted FRU using Maintenance operations after clearing all error conditions, if any, clearing the Port Lockout bit and clearing the Output and Input Port Enable bits in the Port *n* Control CSR. This procedure allows the system host to access the inserted FRU safely, without exposing itself to incorrect behavior by the inserted FRU.

In order to issue Maintenance operations to the inserted FRU, the system host must first make sure that the ackID values for both ends are consistent. Since the inserted FRU has just completed a power-up reset sequence, both it’s transmit and receive ackID values are the reset value of 0x00. The system host can set the switch device’s transmit and receive ackID values to also be 0x00 through the Port *n* Local ackID Status CSR if they are not already in that state, and can then issue packets normally.

The second method for hot insertion would allow the replaced FRU to bring itself into the system, which is necessary for a system in which the FRU is the system host itself. In this approach, the Port Lockout bit is not set and instead the Output and Input Port Enable bits are set for any unconnected port, allowing inserted FRUs free access to the system without reliance on a system host. Also, a port-write operation is not generated when the training sequence completes and the link is active, so a host is not notified of the event. However, this method leaves the system vulnerable to corruption from a misbehaving hot-inserted FRU.

As with the first case, the inserted FRU must make the ackID values for both link partners match in order to begin sending packets. In order to accomplish this, the inserted FRU may send a reset-port request to reinitialize the connected port. If the connected port does not support the reset-port request, the inserted FRU may recover the link by generating a link-request/link-status to the attached device to obtain it's expected receiver value using the Port *n* Link Maintenance Request and Response CSRs. It can then set its transmit ackID value to match. Next, the inserted FRU generates a Maintenance write operation to set the attached device's Port *n* Local ackID Status CSR to set the transmit ackID value to match the receive ackID value in the system host. Upon receipt of the maintenance write, the attached device sets it's transmit ackID value as instructed, and generates the maintenance response using the new value. Packet transmission can now proceed normally.

Hot extraction from a port's point of view behaves identically to a very rapidly failing link and therefore can utilize the above described error reporting mechanism. Hot extraction is ideally done in a controlled fashion by taking the FRU to be removed out of the system as a usable resource through the system management software so that extraction does not cause switch fabric congestion or result in a loss of data. Note that the Port n Link Uninit Discard Timer CSR can be used to prevent congestion in the case of an FRU removal which is not coordinated by the system host.

The required mechanical aspects of hot-insertion and hot-extraction are not addressed in this specification.

A.4 Port-write Discussion

The error management specification includes only one destination for port-write operations, while designers of reliable systems would assume that two is the minimum number. This section explains the rationale for only having one port-write destination.

It is assumed that in the event of an error on a link that both ends of the link will see the error. Thus, there are two parties who can be reporting on any error. In the case that the sole link between an end point and a switch fails completely, the switch is expected to see and report the error. When one of a set of redundant links between an end point and a switch device fails, it is expected that the switch and possibly the end point will report the failure.

When a link between two switches fails, it is assumed that there are multiple paths to the controlling entity available for the port-write to travel. The switches will be able to send at least one, and possibly two, reports to the system host. It is assumed that it is possible to set up a switch’s routing parameters such that the traffic to the system host will follow separate paths from each switch.

In some reliable systems, the system host is implemented as multiple redundant subsystems. It is assumed in RapidIO that only one subsystem is actually in control at any one time, and so should be the recipient of all port-writes. If the subsystem that should be in control is detected to be insane, it is the responsibility of the rest of the control subsystem to change the destination for port-writes to be the new subsystem that is in control.

A.5 Physical Layer Fatal Error Recovery Discussion

Recovery from a fatal error under software control at the physical layer may be possible under certain circumstances. An example of this would be if the transmitter and receiver have lost synchronization of their ackIDs. This could occur if one end of the link experienced a spurious reset. In this case a loss of packets may occur as there may be outstanding unacknowledged packets between the transmitter and the receiver.

Such an event would cause an error to be detected given the appropriate initial conditions at the transmitter, and, eventually a port-write to the system host to be generated if the system is properly configured:

- The reset state of the Input Port Enable bit in the Port *n* Control CSR set to disabled throughout the system.
- All defined errors in the Port *n* Error Detect CSRs are enabled and will increment the error rate counter throughout the system.

If a device experiences a reset event, numerous errors will be detected by the transmitter over time, and eventually an error threshold is reached as described in Section 1.2.2, “Error Reporting Thresholds”, and the system host is notified as described in Section 1.4, “System Software Notification of Error”. The most likely errors that will be detected are bits 12 (Received packet-not-accepted control symbol) and 26 (Non-outstanding ackID) in the Port *n* Error Detect CSRs, but others could be encountered depending upon the state of the link at the time of the reset event.

Re-synchronizing the ackIDs must be done from the transmitter side as it is not possible to communicate with the receiver with maintenance transactions in this situation. Re-synchronizing the ackIDs can be done by sending a reset-port request to those devices which support it. ackID re-synchronization can also be done by resetting some of the physical layer state by writing the Port *n* Local ackID Status CSR with the appropriate ackID values. It may be necessary to have the transmitter drop outstanding packets using that CSR as well, depending upon the situation. It may not be desirable, or it might not be possible, to resend the packets, depending upon the state of the overall system and the transmitter implementation.

Therefore, the following sequence of events occur:

1. The system is configured as described above and is operating.
2. The receiver of a transmitter/receiver pair experiences a reset.
3. The transmitter enters error recovery mode and attempts to re-train the link.
4. Eventually the receiver comes back and link re-training completes.
5. The transmitter starts the error recovery sequence and begins to encounter large numbers of errors due to a bad ackID for a link-response (which may immediately cause a port-write transaction to be sent to the system host to report the condition) or having all packets receive packet-not-accepted control symbols. As noted earlier, other errors may also be detected.
6. At some point, an error threshold is reached and the system host is sent a port-write maintenance transaction to report the condition, if one has not already been sent.
7. The system host cleans up the machine using reset-port requests and/or maintenance transactions, including resetting ackIDs in the transmitter and rediscovering and reconfiguring the lost portion of the machine. This may be a very complex and time-consuming task.

Note that it may be useful to implement resetting the ackIDs and restarting the link in hardware for lab debug or for applications where frequent resets are expected and software intervention is not required. This could leverage the standard port-reset request function.

Blank page

Glossary of Terms and Abbreviations

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

D

Degraded threshold. Bits 8-15 of the Port n Error Rate Threshold CSR. An application-specific level that indicates an unacceptable error rate resulting in degraded throughput, when equal to the error rate count.

F

Failed threshold. Bits 0-7 of the Port n Error Rate Threshold CSR. An application-specific level that indicates an error rate due to a broken link, when equal to the error rate count.

H

Hot-insertion. Hot-insertion is the insertion of a processing element into a powered-up system.

Hot-extraction. Hot-extraction is the removal of a processing element from a powered-up system.

L

Logical/Transport error. A logical/transport error is one that cannot be resolved using the defined transmission error recovery sequence, results in permanent loss of data or causes system corruption. Recovery may possible under software control.

N

Non-reporting processing element. A non-reporting processing element depends upon an attached device (usually a switch) to report its logged errors to the system host on its behalf.

O

Operation. A set of transactions between end point devices in a RapidIO system (requests and associated responses) such as a read or a write.

Ownership. A processing element has the only valid copy of a coherence granule and is responsible for returning it to home memory.

P

Physical error. A physical error occurs only in the physical layer.

Port healing. The process whereby software resets the error rate count, or allows it to decrement as required by the error rate bias field of the Port n Error Rate CSR.

R

Read operation. An operation used to obtain a globally shared copy of a coherence granule.

Reporting processing element. A reporting processing element is capable of reporting its logged errors to the system host.

S

Switch processing element. One of three processing elements, a switch processing element, or switch, is capable of logging and reporting errors to the host system.

T

Transmission error. A transmission error is one that can be resolved using the defined transmission error recovery sequence, results in no permanent loss of data and does not cause system corruption. Recovery may also be possible under software control using mechanisms outside of the scope of this specification.

RapidIO™ Interconnect Specification Part 9: Flow Control Logical Layer Extensions Specification

3.2, 1/2016

RapidIO.org

Revision History

Revision	Description	Date
1.0	First release	06/18/2003
1.3	No technical changes, revision changed for consistency with other specifications Converted to ISO-friendly templates	02/23/2005
2.0	Technical changes: new features showing 05-07-00001.003	06/14/2007
2.1	No technical changes	07/09/2012
2.2	Technical changes: errata showing 10-08-00000.003	05/05/2011
3.0	Changed RTA contact information. Technical changes: Addition of references to Dev32 in packet format descriptions.	10/11/2013
3.1	No technical changes.	09/18/2014
3.2	No technical changes.	01/28/2016

Table of Contents

Chapter 1 Flow Control Overview

1.1 Congestion Management 9
1.1.1 Introduction 9
1.1.2 Requirements 10
1.1.3 Problem Illustration 10
1.2 Flow Arbitration 12
1.2.1 Fixed / Static Resource Allocation 13
1.2.2 Dynamic Resource Allocation Protocol 13

Chapter 2 Logical Layer Flow Control Operation

2.1 Fabric Link Congestion 15
2.2 Flow Arbitration 15
2.2.1 Arbitration Protocol 16
2.2.2 Number Of Outstanding Requests 18
2.3 Flow Control Operation 19
2.4 Physical Layer Requirements 20
2.4.1 Fabric Topology 20
2.4.2 Flow Control Transaction Transmission 20
2.4.2.1 Orphaned XOFF Mechanism 21
2.4.2.2 Controlled Flow List 21
2.4.2.3 XOFF/XON Counters 22
2.4.3 Priority to Transaction Request Flow Mapping 22
2.4.4 Flow Control Transaction Ordering Rules 23
2.4.5 End Point Congestion Management Rules 23
2.4.6 Switch Congestion Management Rules 24
2.4.7 Endpoint Rules for the Arbitration Protocol 24
2.4.8 Abnormal De-allocation of Resources 25

Chapter 3 Packet Format Descriptions

3.1 Introduction 27
3.2 Logical Layer Packet Format 27
3.3 Flow Arbitration Message Fields (FAM) 29
3.4 Transport and Physical Layer Packet Format 29

Chapter 4 Logical Layer Flow Control Extensions Register Bits

4.1 Introduction 31
4.2 Capability Registers (CARs) 31
4.2.1 Processing Elements Features CAR 31

Table of Contents

4.2.2 Port n Control CSR 32

Annex A Flow Control Examples (Informative)

A.1 Congestion Detection and Remediation 33
A.2 Orphaned XOFF Mechanism Description 34
A.3 Discussion on Flow Arbitration 35

List of Figures

1-1 Interconnect Fabric Congestion Example..11
2-1 Single PDU Transfer Scenario..16
2-2 Multi-PDU Transfer Scenario...17
2-3 Multi-PDU scenario with receiver based de-allocation scheme.....................................18
2-4 Pipelined Requests...19
2-5 Flow Control Operation..20
3-1 Type 7 Packet Bit Stream Logical Layer Format...28
3-2 LP-Serial Flow Control Packet..30

List of Figures

Blank page

List of Tables

2-1 Prio field to flowID Mapping 23
3-1 Specific Field Definitions and Encodings for Type 7 Packets 28
3-2 Flow Arbitration Protocol Commands....................29
4-1 Bit Settings for Processing Elements Features CAR....................31
4-2 Bit Settings for Port n Control CSR....................32

List of Tables

Blank Page

Chapter 1 Flow Control Overview

RapidIO transacts operations in “flows”. A flow is defined as the nexus of a source, a destination, and a physical channel. The physical channel is a virtual channel and/or a priority within a virtual channel. Since a large number of simultaneous connections can exist within a fabric, resources within the fabric and at the endpoints can be a constraint. The protocols defined in this specification permit the management of resources on a flow basis.

The protocol consists of two functions, congestion management and flow arbitration. Congestion management may be implemented by endpoints or switches independent of the flow arbitration protocol. The flow arbitration protocol only applies to endpoints. Implementation of this specification is optional.

1.1 Congestion Management

1.1.1 Introduction

A switch fabric based system can encounter several types of congestion, differentiated by the duration of the event:

- Ultra short term
- Short term
- Medium term
- Long term

Congestion can be detected inside a switch, at the connections between the switch, and other switches and end points. Conceptually, the congestion is detected at an output port that is trying to transmit data to the connected device, but is receiving more information than it is able to transmit. This excess data can possibly “pile up” until the switch is out of storage capacity, and then the congestion spreads to other devices that are connected to the switch’s inputs, and so on. Therefore, contention for a particular connection in the fabric can affect the ability of the fabric to transmit data unrelated to the contested connection. This is highly undesirable behavior for many applications.

The length of time that the congestion lasts determines the magnitude of the effect the congestion has upon the system overall.

Ultra short term congestion events are characterized as lasting a very small length of time, perhaps up to 500 or so nanoseconds. In a RapidIO type system these events are adequately handled by a combination of buffering within the devices on either end of a link and the retry based link layer mechanism defined in the RapidIO Part 4: 8/16 LP-LVDS Physical Layer and RapidIO Part 6: 1x/4x LP-Serial Physical Layer Specifications. This combination adds “elasticity” to each link in the system. The impact of ultra short term events on the overall system is minor, if noticeable at all.

Short term congestion events last much longer than ultra short term events, lasting up into the dozens or hundreds of microseconds. These events can be highly disruptive to the performance of the fabric (and the system overall), in both aggregate bandwidth and end to end latency. Managing this type of congestion requires some means of detecting when an ultra short term event has turned into a short term event, and then using some mechanism to reduce the amount of data being injected by the end points into the congested portion of the fabric. If this can be done in time, the congestion stays localized until it clears, and does not adversely affect other parts of the fabric.

Medium term congestion is typically a frequent series of short term congestion events over a long period of time, such as seconds or minutes. This type of event is indicative of an unbalanced data load being sent into the fabric. Alleviating this type of congestion event requires some sort of software based load balancing mechanism to reconfigure the fabric.

Long term congestion is a situation in which a system does not have the raw capacity to handle the demands placed upon it. This situation is corrected by upgrading (or replacing) the system itself.

This specification addresses the problem of short term congestion.

1.1.2 Requirements

The flow control mechanism shall fulfill the following goals:

- Simple - excess complexity will not gain acceptance
- React quickly - otherwise the solution won’t work
- Robust - same level of protection and recovery as the rest of RapidIO
- Scalable - must be able to extend to multi-layer switch systems
- Compatibility with all physical layers

1.1.3 Problem Illustration

The *RapidIO Part 1: Input/Output Logical Specification* defines a transaction request flow as a series of packets that have a common source identifier and a common destination identifier at some given priority. On a link, packets of a single transaction request flow can be interleaved with packets from one or more other transaction request flows.

No assumptions are made on the underlying switch architecture for this discussion of the short term congestion problem. Also for the purposes of this discussion, an idealized output queued switch is assumed, which in literature is also used to compare the performance of a particular switch under study. Packet buffers are associated with the output of the switch. An example switch topology showing output buffers is illustrated in Figure 1-1 below. A point of congestion is therefore associated with an output buffer of such a switch.

The problem that is to be addressed by this specification is caused by multiple independent transaction request flows, each with burst and spatial locality characteristics that typically do not exceed the bandwidth capacity of links or end points. Due to the statistical combination of such transaction request flows, usually in the middle of multistage topologies, the demand for bandwidth through a particular link exceeds the link's capacity for some period of time, for example, Data Flows a, b, and c for an output port of Switch 3 as shown in Figure 1-1. As a result, the output buffer for this port will fill up, causing the link layer flow control to be activated on the links of the preceding switch stages. The output packet buffers for Switches 1 and 2 then also fill up. Packets for transaction request flows, such as data flow d, in these same output buffers not destined for the output port with the full buffer in Switch 3 are now also waiting, causing additional system performance loss. This phenomenon is known as higher order head of line blocking.

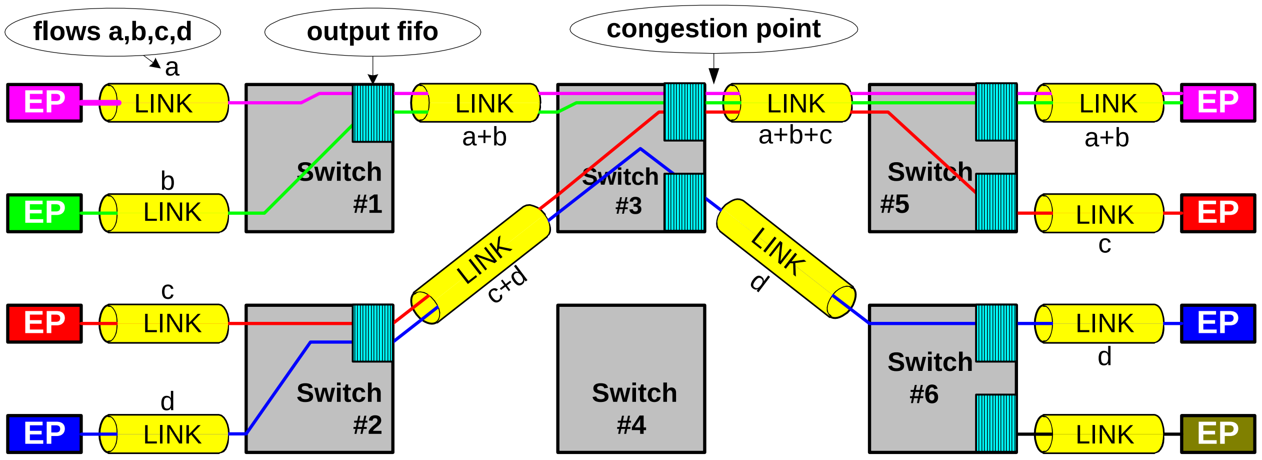


Figure 1-1. Interconnect Fabric Congestion Example

A second problem, less frequently a contributor to system performance loss, occurs when an end point cannot process the incoming bandwidth and employs link layer flow control to stop packets from coming in. This results in a similar sequence of events as described above.

The problem described in this section is very well known in the literature. The aggregate throughput of the fabric is reduced with increased load when congestion control is not applied (see reference [1]). Such non-linear behavior is known as 'performance-collapse'. It is the objective of this specification to provide a logical layer flow control mechanism to avoid this collapse. Research also shows that relatively simple "XON/XOFF" controls on transaction request flows can be adequate to control congestion in fabrics of significant size.

The reason for the described non-linear behavior is illustrated with a saturation tree. The point at which a single transaction request flow that causes link bandwidth to be exceeded and causes buffer overflow is referred to as the root of the saturation tree. This tree grows backward towards the sources of all transaction request flows going through these buffers, and all buffers that these transaction request flows pass through in preceding stages, causing even more transaction request flows to be affected.

An important design factor for interconnect fabrics is the latency between a congestion control action being initiated and the transaction request flow source acting in response. This latency determines, among other factors, the required buffer sizes for the switches. To keep such buffers small, the latency of a congestion control mechanism must be minimized. For example, 10 data flows contribute to a buffer overflow (forming what is known as a "hotspot"). If it takes 10 packet transmission times for the congestion notification to reach the sources and the last packets sent from the sources to reach the point of congestion after the sources react to the congestion notification, up to 100 packets could be added to the congested buffer. The number of packets added may be much smaller depending on the rate of oversubscription of the congested port.

Reference

[1] "Tree saturation control in the AC3 velocity cluster interconnect", W. Vogels et.al., Hot Interconnects 2000, Stanford.

1.2 Flow Arbitration

Protocols such as the RapidIO Data Streaming Logical Layer are designed to carry Protocol Data Units (PDUs) of lengths greater than 256 bytes by utilizing Segmentation and Reassembly (SAR). Coherency of the segmentation and reassembly process is enforced by RapidIO's ordering rules for a packet flow. Since a flow (the nexus of a source, destination, and physical channel) must deliver packets in order, an endpoint ensures coherency of a segmentation / reassembly process by only introducing one complete PDU into a specific flow at a time.

However, an endpoint can potentially connect to up to 64K other endpoints, with 4 or 8 or even 16 physical channels available between each endpoint. As such, an endpoint could have to potentially support millions of dedicated SAR contexts and reassembly buffers. For large PDUs having dedicated reassembly buffers per endpoint could be costly.

The Data Streaming Segmentation and Reassembly contexts are one example of flow based resources that may be a limited resource. Other logical layer functions, like DMA contexts, can also run into resource constraints.

Managing limited resources can be done in a variety of ways, the use of an arbitration protocol is not mandatory:

1.2.1 Fixed / Static Resource Allocation

Fixed allocation of resources occurs by system design. Systems with smaller topologies, or with endpoint resources sufficient for all anticipated flows, do not require any specific management. In larger systems, some portion of the resources can be fixed and assumed to be always available, reducing the number of resources that might have to be further managed.

Resources may also be statically allocated on an individual connection basis. These resources would only be allocated via the overall connection admission algorithm. This additional layer of protection prevents flows from being admitted to the fabric that do not have corresponding resources on the receiving end.

1.2.2 Dynamic Resource Allocation Protocol

The dynamic arbitration protocol is designed to arbitrate and allocate resources to flows for short durations of time. It allows a fewer number of resources to be dynamically shared among a larger number of flows. The system may still require the use of these resources to be intelligently managed in order to achieve desired system performance. The dynamic arbitration of resources will prevent data loss caused by overrunning the receiver.

The congestion management commands affect flows on a packet boundary basis. The arbitration protocol commands affect flows at PDU boundaries (a PDU can consist of one or more packets). Endpoints must have the same understanding of PDU boundaries.

Blank page

Chapter 2 Logical Layer Flow Control Operation

This chapter describes the logical layer flow control mechanisms.

2.1 Fabric Link Congestion

In compliant devices, logical layer flow control methods shall be employed within a fabric or destination end point for the purpose of short term congestion abatement at the point in time and location at which excessive congestion is detected. This remediation scheme shall be enacted via explicit flow control messages referred to as transmit off (XOFF) and transmit on (XON) congestion control packets (CCPs) which, like any other packet, require link-level packet acknowledgements. The XOFF CCPs are sent to shut off select flows at their source end points. Later, when the congestion event has passed, XON CCPs are sent to the same source end points to restore those flows.

The method used to detect congestion is implementation specific and is heavily dependent upon the internal packet buffering structure and capacity of the particular switch device. In the example output port buffered switch from “Section 1.1.3, Problem Illustration”, on page 10, congestion occurs when some output buffer watermark is exceeded, but this is not the only way of detecting congestion. Several possible implementation methods are described in Appendix A. These described methods are purely exemplary and are not intended to be an exhaustive list of possible methods.

2.2 Flow Arbitration

The flow arbitration protocol extends the Congestion Control Packet (CCP) protocol first introduced in Revision 1.3 of this specification. In addition to the XON/XOFF congestion management functionality the arbitration protocol adds the following commands:

- REQUEST
- XOFF to indicate un-availability of resources.
- XON to allow and grant use of resources
- RELEASE

2.2.1 Arbitration Protocol

The protocol is illustrated in the following diagrams, using Data Streaming allocation of SAR resources as examples. There are two request messages pertaining to single PDU and multi-PDU transfers. The single PDU case is illustrated in Figure 2-1. The transmitting endpoint sends a single PDU request. The receiving endpoint will respond with either a XON(ARB) or XOFF(ARB) message depending on whether it has buffer and context resources available.

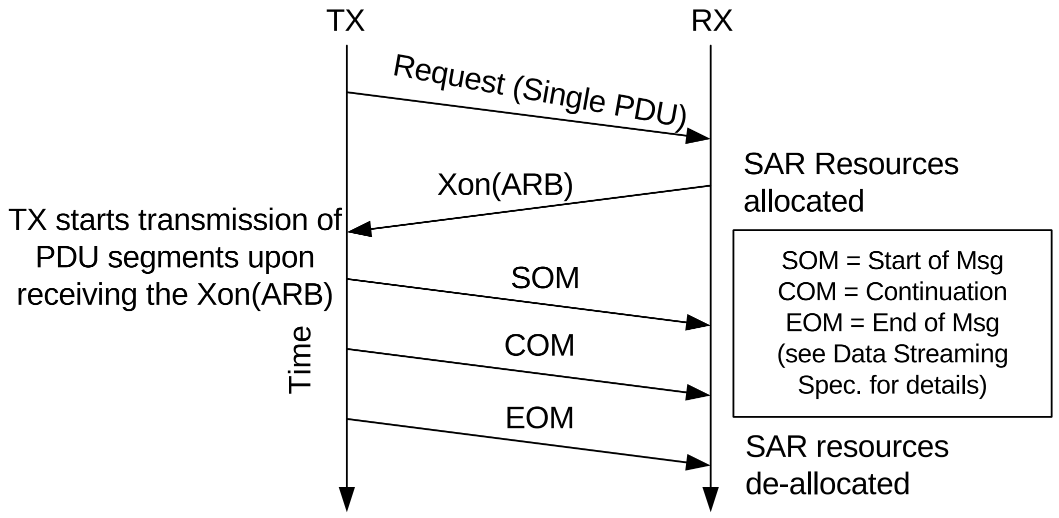

TX
RX
Request (Single PDU)
SAR Resources allocated
Xon(ARB)
TX starts transmission of PDU segments upon receiving the Xon(ARB)
SOM
COM
EOM
Time
SOM = Start of Msg
COM = Continuation
EOM = End of Msg
(see Data Streaming Spec. for details)
SAR resources de-allocated

Figure 2-1. Single PDU Transfer Scenario

In the single PDU transfer case, if the receiving endpoint responds with a XOFF(ARB), the transmitting endpoint can send a new REQUEST message to ask for resources. If the receiving endpoint responds with a XON(ARB), the transmitting endpoint can start transmitting the PDU segments once it receives the XON(ARB) message. The receiver will automatically de-allocate resources once it receives the last packet for the PDU.

When the transmitting endpoint sends a request pertaining to the transfer of multiple PDUs the receiving endpoint, similarly to the single PDU case, shall respond with either the XON(ARB) or the XOFF(ARB) protocol depending on the availability of buffering resources. If the receiving endpoint responds with a XON(ARB) message, as shown in Figure 2-2, the transmitting endpoint can start sending the PDU segments once it receives the XON(ARB) message. The transmitter can send multiple PDUs without having to renegotiate the resources. The receiver will hold the allocated resources until it receives a RELEASE message from the transmitting endpoint.

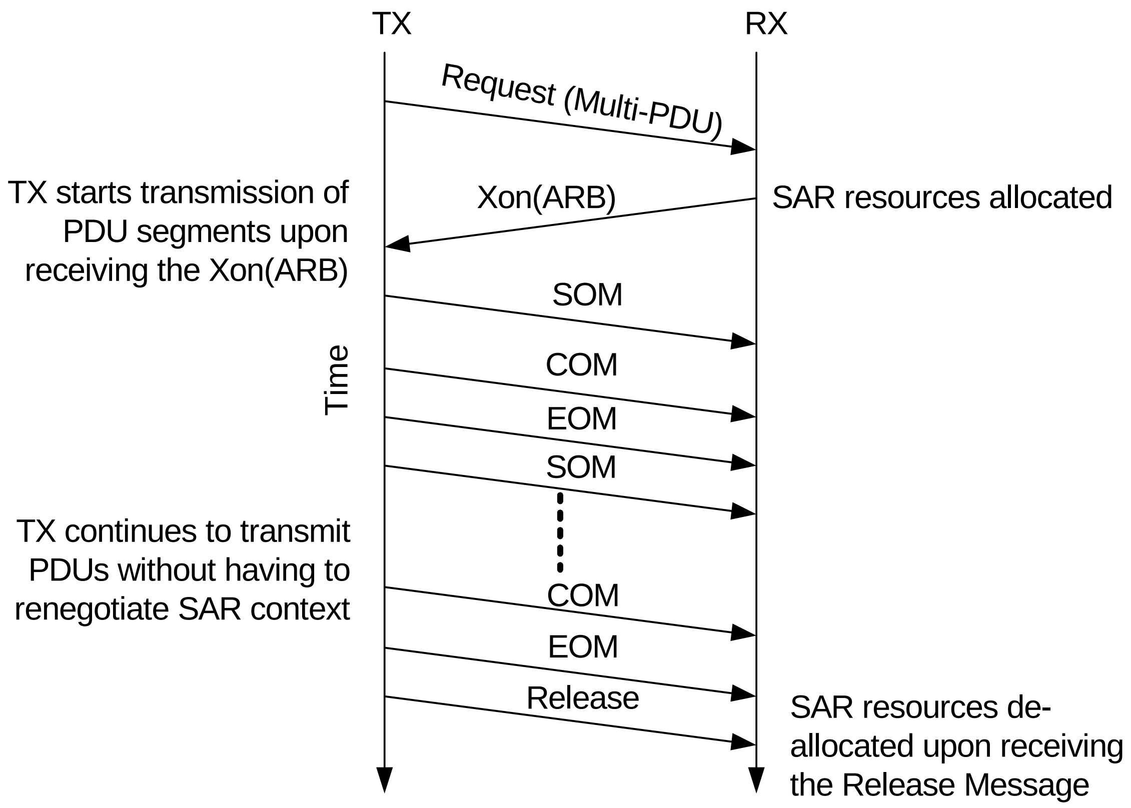

TX
RX
Request (Multi-PDU)
Xon(ARB)
SAR resources allocated
TX starts transmission of PDU segments upon receiving the Xon(ARB)
SOM
Time
COM
EOM
SOM
TX continues to transmit PDUs without having to renegotiate SAR context
COM
EOM
Release
SAR resources de-allocated upon receiving the Release Message

Figure 2-2. Multi-PDU Transfer Scenario

The receiver can also inform the transmitting endpoint of its desire to de-allocate the resources, by sending a XOFF(ARB) message. The transmitting endpoint, after sending the last packet at the current PDU boundary, will send a RELEASE message. The receiver shall de-allocate the resources only when it has received the RELEASE message. This scenario is illustrated in Figure 2-3.

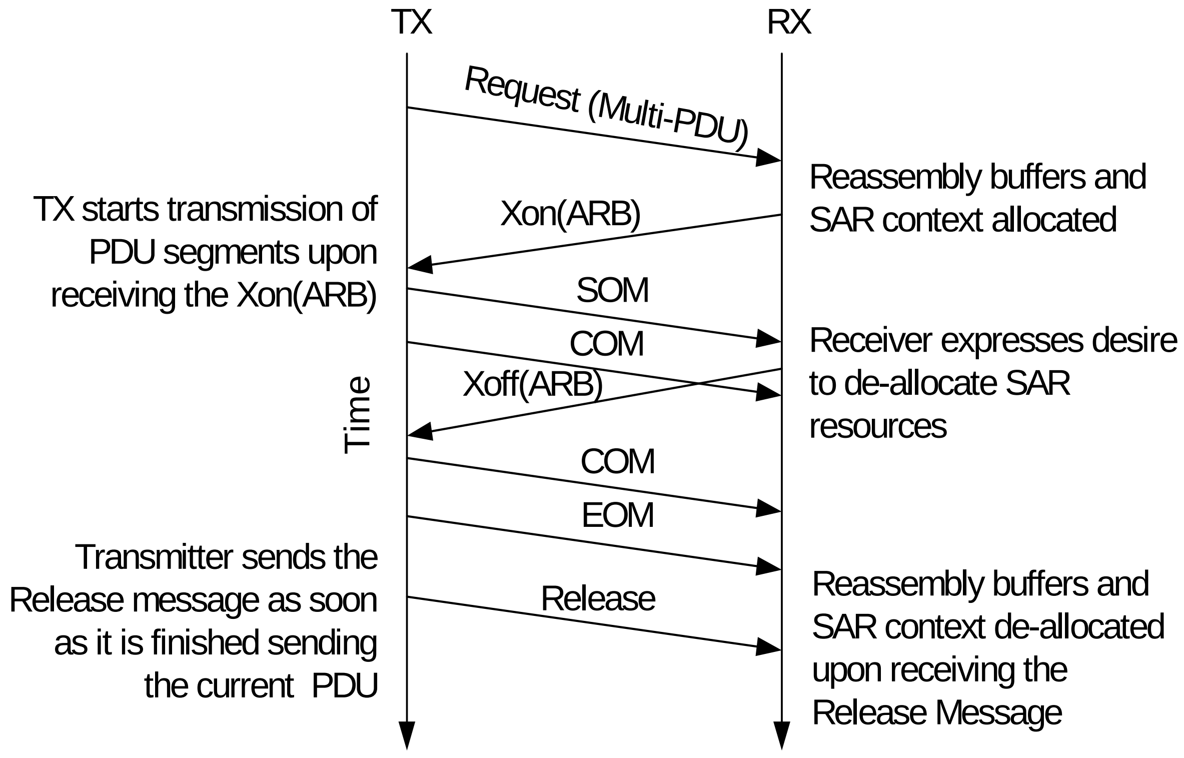

TX
RX
Request (Multi-PDU)
Reassembly buffers and SAR context allocated
TX starts transmission of PDU segments upon receiving the Xon(ARB)
Xon(ARB)
SOM
COM
Xoff(ARB)
Receiver expresses desire to de-allocate SAR resources
Time
COM
EOM
Transmitter sends the Release message as soon as it is finished sending the current PDU
Release
Reassembly buffers and SAR context de-allocated upon receiving the Release Message

Figure 2-3. Multi-PDU scenario with receiver based de-allocation scheme

2.2.2 Number Of Outstanding Requests

In the arbitration protocol the transmitting endpoint has to wait at least a round trip time after it has sent the request message before it can start transmitting. This delay may be undesirable in high performance systems. Therefore, in order to overlap the request phase with the data transmission phase, the transmitter is allowed to have a maximum of one outstanding request in the system, that is, it can pipeline requests to increase the efficiency of the system. The requests and the corresponding responses are identified by a 1 bit sequence number. This pipelining of requests is allowed for both single PDU and multi-PDU requests.

Consider the exchange shown in Figure 2-4 below. The transmitting endpoint issues a request. The request is processed by the receiving endpoint and a XON issued. Once the transmitter receives the XON message, it can start transmitting the data and it may also pipeline another request. The pipelined request shall not be honored until the current transaction has been completely received and resources are available for the next transaction. In Figure 2-4 Request_1 is sent after the transmitting endpoint has received the XON_0(ARB) message for the previous request (Request_0). If the receiving endpoint for some reason cannot queue/process the requests, it can send a XOFF (ARB) message immediately to indicate lack of resources.

The pipelining of requests is managed by the source. It only issues the next request when the current request has been acknowledged. The destination only acknowledges the next request when the current request has completed. So, the destination only has to queue up one outstanding request per flow. This pipelining also offers the destination an opportunity to use the pending requests to get a look at the incoming traffic and make better allocation decisions should there be limited resources.

A single level of pipelining of requests is adequate because this is on a per flow basis. flow may only have a single open context at a time, so the current context must complete before the flow can be used for another transaction.

NOTE: Context Definition

As a reminder, a “context” is a group of individual transactions that must remain ordered, and may not have intervening transactions from a different context in the same flow.

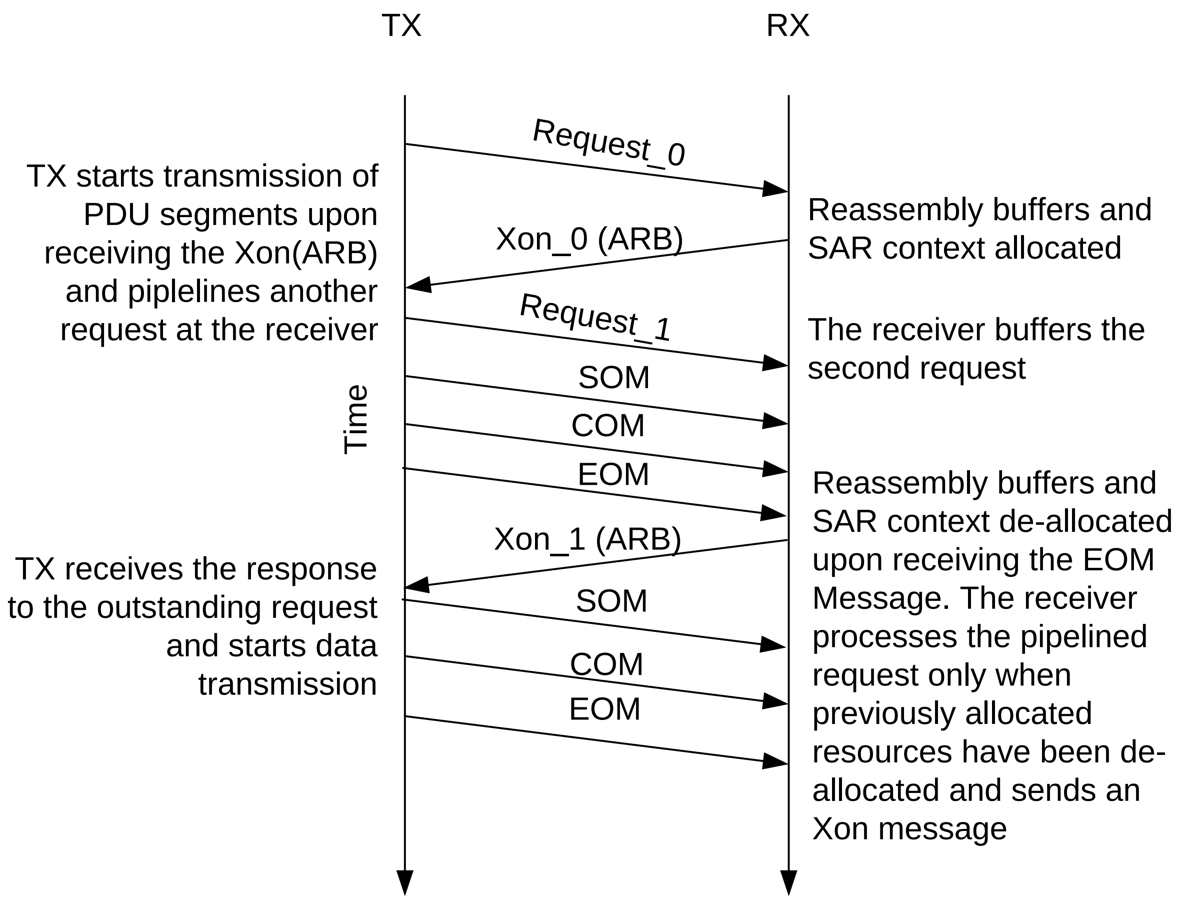

TX
RX
Request_0
TX starts transmission of PDU segments upon receiving the Xon(ARB) and piplelines another request at the receiver
Reassembly buffers and SAR context allocated
Xon_0 (ARB)
Request_1
The receiver buffers the second request
SOM
COM
EOM
Time
Reassembly buffers and SAR context de-allocated upon receiving the EOM Message. The receiver processes the pipelined request only when previously allocated resources have been de-allocated and sends an Xon message
Xon_1 (ARB)
TX receives the response to the outstanding request and starts data transmission
SOM
COM
EOM

Figure 2-4. Pipelined Requests

2.3 Flow Control Operation

The flow control operation consists of a single FLOW CONTROL transaction as shown in Figure 2-5. The FLOW CONTROL transaction is issued by a switch or end point to control the flow of data. This mechanism is backward compatible with RapidIO legacy devices in the same system.

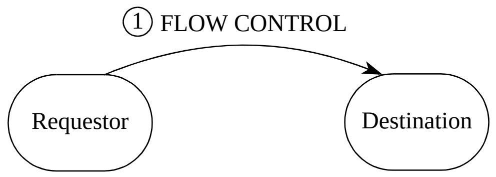


Figure 2-5. Flow Control Operation

While FLOW CONTROL packets do not contain response packets, the arbitration protocol does consist of multiple transactions between the source of a data flow and the destination. Some of the transactions flow from the destination to the source.

2.4 Physical Layer Requirements

This section describes requirements put upon the system physical layers in order to support efficient logical layer flow control.

2.4.1 Fabric Topology

The interconnect fabric for a system utilizing the logical layer flow control extensions must have a topology such that a flow control transaction can be sent back to any transaction request flow source. This path through the fabric may be back along the path taken by the transaction request flow to the congestion point or it may be back along a different path, depending upon the requirements of the particular system.

2.4.2 Flow Control Transaction Transmission

Flow control transactions are regarded as independent traffic flows. They are the most important traffic flow defined by the system. Congestion management transactions are always transmitted at the first opportunity at the expense of all other traffic flows if possible. For the 8/16 LP-LVDS and 1x/4x LP-Serial physical layer specifications, this requires marking flow control packets with a “prio” field value of 0b11, and a “CRF” bit value of 0b1, if supported. Flow arbitration has additional requirements for some transactions to be transmitted in the same flow as the data packets. All of these transactions use a normal packet format for purposes of error checking and format.

Because an implicit method of flow restoration was simulated and found to be impractical for RapidIO fabrics due to lack of system knowledge in the end point, an explicit restart mechanism using a XON transaction is used. In the CCP flow back to the source end point, XOFF and XON CCPs may be dropped on input ports of downstream elements in the event of insufficient buffer space.

2.4.2.1 Orphaned XOFF Mechanism

Due to the possibility of XON flow control packets being lost in the fabric, there shall be an orphaned XOFF mechanism for the purpose of restarting orphaned flows which were XOFF’d but never XON’d in end points. Details of this mechanism are implementation specific, however the end point shall have sufficient means to avoid abandonment of orphaned flows. A typical implementation of such a mechanism would be some sort of counter. A description of a possible implementation is given in Appendix A. The Orphaned XOFF Mechanism is intended to work with the rest of the XON/XOFF CCPs to handle the short term congestion problem as previously described, and so shall operate such that software intervention is not required or inadvertently invoked.

Counter mechanisms for arbitrated flows are also an implementation decision, but care should be taken before just enabling transmission on a flow. Unlike the congestion management protocol, arbitrated flows are expected to remain off until explicitly enabled. Timeouts at source end points should result in retrying requests, not just arbitrarily starting a flow.

2.4.2.2 Controlled Flow List

It is required that elements which send XOFFs keep a list of flows they have stopped, along with whatever flow-specific information is needed to select flows for restart, such as per-flow XON watermark level, or relative shut off order. This information shall be stored along with flow identification information in a “controlled flow list”, a memory structure associated with the controlling element. It shall be permissible in the time following the sending of a XOFF CCP for the flow control -initiating element to re-evaluate system resources and modify the flow restart ordering or expected XON watermark level within the controlled flow list to better reflect current system state. It shall not however be permissible to abandon the controlled flow by “forgetting” it, either due to lack of controlled flow list resources or other factors. In the event that limited controlled flow list resources cause the congested element to have insufficient room to issue another XOFF CCP which is deemed more important than a previously-XOFF’d controlled flow, then that previously-XOFF’d controlled flow may be prematurely XON’d and removed from the controlled flow list. The new, more important flow may be XOFF’d and take its place in the controlled flow list.

Details of the controlled flow list are implementation specific, though at the very least it shall contain entries for each currently XOFF’d flow, including flow identification information. It is likely that some state information will be required, such as expected time of flow restart, or per-flow restart watermark levels. The controlled flow list size is selected to provide coverage for short term congestion events only. Remediation for medium and greater -term congestion events is beyond the scope of logical layer flow control as these events likely indicate systemic under-provisioning in the fabric.

Arbitrated resources must also be associated with a flow list to keep track of the flow the resources are allocated to. Should the source fail to utilize the resource in an expected interval, the destination may take action to recover the resource. The arbitration protocol provides a method to attempt to deallocate the resource in concert with the source to avoid packet loss (see “Section 2.2.1, Arbitration Protocol”, on page 16). Should that fail, asynchronous de-allocation of the resource may be used, with the understanding that packet loss could result. The implementation of such a mechanism is not specified here. Care should be taken to account for fabric latencies and not cause excessive packet loss during higher latency intervals.

2.4.2.3 XOFF/XON Counters

XOFF/XON counters shall be instantiated for some number of output flows at the end point. Since the number of flows may be large or unpredictable, the number of counters and how flows are aggregated to a particular counter is implementation dependant. However, all flows must be associated with a counter. For simplicity, the following behavioral description assumes a single flow associated with a single counter. The counter is initialized to zero at start up or when a new DestinationID and given Priority is initialized. The counter increments by one for each associated XOFF CCP and decrements by one for each associated XON CCP, stopping at zero. Only when this counter is equal to zero is the flow enabled. In no event shall the counter wrap upon terminal count. If the orphaned XOFF mechanism activates, the counter is reset to zero and the flow is restarted.

2.4.3 Priority to Transaction Request Flow Mapping

When a switch or end point determines that it is desirable to generate a flow control transaction, it must determine the associated flowID for the (non-maintenance and non-flow control) packet that caused the flow control event to be signalled. Maintenance and flow control transaction request flows must never cause the generation of a flow control transaction. For the 8/16 LP-LVDS and the 1x/4x LP-Serial physical layer specifications, the flowID of a transaction request flow is mapped to the “prio” bits as summarized in Table 1-3 of the 8/16 LP-LVDS specification and Table 5-1 of the 1x/4x LP-Serial specification. Determining the original transaction request flow for the offending packet requires the switch to do a reverse mapping.

It is recognized that mapping a particular response to a particular transmission request may be inaccurate because the end point that generated the response is permitted in the physical layer to promote the response to a priority higher than would normally be assigned. Deadlock avoidance rules permit this promotion. For this reason the choice of which flow to XOFF is preferably made using request packets, not response packets, as responses release system resources, which also may help alleviate system congestion.

Additionally, the CRF bit should also be used in conjunction with flowID to decide whether or not a particular transaction request flow should be targeted with a XOFF flow control transaction. A switch may select for shut off a packet with CRF=0 over a packet with CRF=1 if there are two different flows of otherwise equal importance. Correspondingly, an end point may choose to ignore a flow control XOFF request for a transaction request flow that it regards as critical.

The reverse mappings from the transaction request flow prio field to the CCP flowID field for the 8/16 LP-LVDS and 1x/4x LP-Serial physical layers are summarized in Table 2-1.

Table 2-1. Prio field to flowID Mapping

Transaction Request flow prio Field	Transaction Type	System Priority	CCP flowID
0b00	request	Lowest	A
0b00	response	Illegal	
0b01	request	Next	B
0b01	response	Lowest	A
0b10	request	Highest	C or higher
0b10	response	Lowest or Next	A or B
0b11	request	Illegal	
0b11	response	Lowest or Next or Highest	A, B, C or higher

2.4.4 Flow Control Transaction Ordering Rules

The ordering rules for flow control transactions within a system are analogous to those for maintenance transactions.

1. Ordering rules apply only between the source (the original issuing switch device or destination end point) of flow control transactions and the destination of flow control transactions.
2. There are no ordering requirements between flow control transactions and maintenance or non-maintenance request transactions.
3. A switch processing element must pass through flow control transactions between an input and output port pair in the order they are received.
4. An end point processing element must process flow control transactions from the same source (the destination of the packet that caused the flow control event) in the order they are received.

2.4.5 End Point Congestion Management Rules

There are a number of rules related to flow control that are required of an end point that supports the logical layer flow control extensions.

1. A XOFF flow control transaction stops all transaction request flows of the specified priority and lower targeted to the specified destination and increments the XON/XOFF counter associated with the specified flowID.
2. A XON flow control transaction decrements the XON/XOFF counter associated with the specified flowID. If the resulting value is zero, the transaction request flows for that flowID and flowIDs of higher priority are restarted.
3. An end point must be able to identify an orphaned XOFF'd flow and restart it.
4. A destination end point issuing a XOFF Flow Control transaction must maintain the information necessary to restart the flow with a XON flow control transaction when congestion abates.
5. Upon detection of congestion within one of its ports, the destination end point shall send required CCP(s) as quickly as possible to reduce latency back to the source end point.

2.4.6 Switch Congestion Management Rules

There are a number of rules related to flow control that are required of a switch that supports the logical layer flow control extensions.

1. Upon detection of congestion within a port, the switch shall send a CCP (XOFF) for each congested flow to their respective end points.
2. If a switch runs out of packet buffer space, it is permitted to drop CCPs.
3. A switch issuing a XOFF Flow Control transaction must maintain the information necessary to restart the flow with a XON flow control transaction when congestion abates.

2.4.7 Endpoint Rules for the Arbitration Protocol

Transmitters shall not transmit on an arbitrated flow unless a resource is available for reception of the PDU. Assumption of an available resource can either be fixed, statically allocated, or dynamically allocated. If dynamically allocated, the protocol must obey the following rules:

1. The transmitter shall issue a REQUEST when it wishes the receiver to allocate a resource to a particular flow. Note that this does not imply a PDU is immediately ready for transmission. The algorithms for resource allocation are up to the implementation.
2. The receiver shall respond to all REQUEST messages with a XOFF(ARB) or XON(ARB) depending on the availability of resources and the arbitration policy at the receiver.
3. The transmitter may send a new REQUEST message if: 1) it did not receive a response to the previous REQUEST message and timed out, or 2) it received a XOFF(ARB) message from the receiving endpoint.

4. Sequence numbers shall remain coherent for each individual flow. The sequence number shall remain the same for REQUESTs reissued for a given flow, without having received a response. The sequence number shall advance for any new REQUESTs on a given flow.
5. If a single PDU request is granted, the receiver may deallocate the resources at any time after: 1) it receives the last segment for the PDU, or 2) it does not receive a packet and an idle counter for the session times out (see rule l). The transmitter shall assume the context is no longer available upon sending the last segment for the PDU.
6. If the resources were granted in response to a multi-PDU request the transmitter may transmit PDUs continuously on that flow until the resource is de-allocated.
7. The transmitter may relinquish a multi-PDU context by sending a RELEASE message after completion of the current PDU.
8. The receiver may send a XOFF(ARB) message during the multi-PDU transfer to indicate its desire to deallocate resources. The transmitter, upon receiving a XOFF(ARB) message during the multi-PDU transfer, shall complete transmission of the current PDU and send a RELEASE message to allow the receiver to de-allocate the resources. The receiver may not deallocate the resources until the RELEASE message is received.
9. The receiver may delay sending responses to the REQUEST commands to consider which REQUESTs to grant or reject. There is no ordering requirement for processing requests from different flows.
10. Only a single instance of resources shall be allocated to a flow at any point in time.
11. The transmitter may issue a single additional REQUEST in advance of completion of the current PDU. However, the transmitter shall not have more than one outstanding REQUEST at any point in time.
12. REQUEST, XON(ARB), and XOFF(ARB) messages may be sent in any flow (such as a high priority channel). RELEASE messages shall be sent in the same flow that the context is allocated for.

2.4.8 Abnormal De-allocation of Resources

Abnormal de-allocation of Resources will occur under the following circumstances:

1. If the resources were allocated in response to a single PDU request:

– The PDU may be aborted according to the exceptions defined in the logical layer rules for the segmentation process. An aborted PDU results in the de-allocation of the resources.

– If the receiver does not receive a packet from the transmitter before the idle counter for the session times out, the resources would be de-allocated. Note that the use of a timer is implementation specific. Incorrect use of a timer may result in packet loss.

2. If the resources were allocated in response to a Multi-PDU request:

– A PDU may be aborted according to the exceptions defined in the logical layer. The resources will still not be de-allocated until a RELEASE message is received.

– If the receiver does not receive a packet from the transmitter before the idle counter for the session times out, the receiver shall first attempt to use the XOFF(ARB) / RELEASE handshake to deallocate the context. If a subsequent timeout is encountered, the SAR resources are asynchronously de-allocated.

NOTE: Timeouts for Arbitration Protocol are Optional

Use of timers with the arbitration protocol is optional. Timer intervals are specific to system implementation and performance goals. Aggressive timer intervals may result in retrying operations that were simply slowed down due to system congestion. Aggressive recovery of resources may also result in packet loss. Conservative time intervals may result in poor performance if transactions are lost or corrupted. It is up to the implementer to determine the correct behavior for the specific system environment.

Chapter 3 Packet Format Descriptions

3.1 Introduction

This chapter contains the definition of the flow control packet format. The type 7 FLOW CONTROL packets are used by the switch or the end points to signal congestion buildup within the node (switch or endpoint) or exchange flow arbitration protocol messages.

3.2 Logical Layer Packet Format

The type 7 FLOW CONTROL packet formats (Flow Control Class) are used by a RapidIO switch or end point processing element to stop (XOFF) and start (XON) the flow of traffic to it from a targeted RapidIO end point processing element. A single transaction request flow is targeted with a CCP. Type 7 packets do not have a data payload and do not generate response packets. The origin of a flow control packet shall set the SOC (Source of Congestion) bit to (SOC=0) if it is a switch or (SOC=1) if it is an end point. The SOC bit is informational only but may be useful for system software in identifying a failing end point.

The Flow Arbitration Message (FAM) field is used to modify the XON or XOFF message for the purposes of flow arbitration.

Definitions and encodings of fields specific to type 7 packets are provided in Table 3-1.

Table 3-1. Specific Field Definitions and Encodings for Type 7 Packets

Type 7 Fields	Encoding	Definition
XON/XOFF	0b0	XOFF For devices not supporting flow arbitration: Stop issuing requests for the specified and lower priority transaction request flows For devices supporting flow arbitration (see Table 3-2)
	0b1	XON For devices not supporting flow arbitration: Start issuing requests for the specified and higher priority transaction request flows For devices supporting flow arbitration (see Table 3-2)
flowID	 0000000 (Flow0A) 0000001 (Flow 0B) 0000010 (Flow 0C) 0000011 (Flow 0D) 0000100 (Flow 0E) 0000101 (Flow 0F) 1000001 (Flow 1A) 1000010 (Flow 2A) 1000011 (Flow 3A) 1000100 (Flow 4A) 1000101 (Flow 5A) 1000110 (Flow 6A) 1000111 (Flow 7A) 1001000 (Flow 8A)	Highest priority affected transaction request flow for VC0 transaction request flow A transaction request flow B transaction request flow C transaction request flow D transaction request flow E transaction request flow F and higher For VC1-VC8 the following flow IDs will result in the VC1-VC8 flow control Flow 1A and higher for VC1 Flow 2A and higher for VC2 Flow 3A and higher for VC3 Flow 4A and higher for VC4 Flow 5A and higher for VC5 Flow 6A and higher for VC6 Flow 7A and higher for VC7 Flow 8A and higher for VC8 Remaining encodings are reserved for the 8/16 LP-LVDS and the 1x/4x LP-Serial physical layers.
destinationID	—	Indicates which end point the CCP is destined for (sourceID of the packet which caused the generation of the CCP).
tgtdestinationID	—	Combined with the flowID field, indicates which transaction request flows need to be acted upon (destinationID field of the packet which caused the generation of the CCP).
SOC	0b0	Source Of Congestion is a Switch
	0b1	Source Of Congestion is an End Point
FAM		See Section 3.3
rsrv	0b0000	Reserved

Figure 3-1 displays a CCP packet with all its fields. The field value 0b0111 in Figure 3-1 specifies that the packet format is of type 7. Dev8 (tt=0b00) and Dev16 (tt=0b01) Transport Formats are shown in the figure, additionally there is the Dev32 (tt=0b10) transport size.

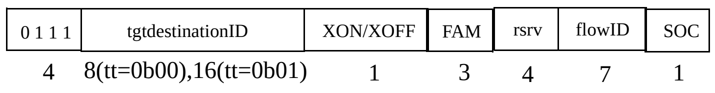


Figure 3-1. Type 7 Packet Bit Stream Logical Layer Format

3.3 Flow Arbitration Message Fields (FAM)

The flow arbitration protocol uses the 3 FAM bits along with the XON/XOFF bit to identify the messages. A device that does not support the SAR protocol ignores the FAM bits. It should be noted that a device which supports the flow arbitration protocol when communicating with an end point that does not support SAR protocol should default to the congestion management (XON/XOFF) functionality and not send other messages as they would be mis-interpreted. The CAR bits define whether the device supports the flow arbitration protocol or not. The bit "Y" is the sequence number bit previously described.

Table 3-2. Flow Arbitration Protocol Commands

XON/XOFF	FAM	Definition
0	0b000	XOFF: Transmit off (Congestion management) Stop issuing requests for the specified and lower priority transaction request flows
	0b010	XOFF(ARB): Flow Request Rejected. Message with sequence number 0 (LSB) in response to REQUEST with sequence number 0.
	0b011	XOFF(ARB):Flow Request Rejected. Message with sequence number 1(LSB) in response to REQUEST with sequence number 1.
	0b10Y	RELEASE: Release message informs the receiving endpoint to de-allocate the buffer space reserved by the receiving endpoint. The release message should be used in conjunction with the request.
1	0b000	XON: Transmit on. (Congestion management) Start issuing requests for the specified and higher priority transaction request flows
	0b01Y	XON(ARB): Flow Request Granted. Reassembly buffer space is now available and allocated.
	0b10Y	REQUEST: Request Flow Single PDU. Buffer space will be de-allocated once the end transaction is received for that PDU.
	0b11Y	REQUEST: Request Flow Multi-PDU. This request message informs the receiving endpoint to reserve the buffer space until it receives the release message. The buffer space will be de-allocated once the release command is received by the receiver.

3.4 Transport and Physical Layer Packet Format

Figure 3-2 shows a complete flow control packet, including all transport and 1x/4x LP-Serial physical layer fields except for delineation characters. The destinationID field of the CCP packet is the sourceID field from packets associated with the congestion event, and is the target of the flow control transaction. The tgtdestinationID field is the destinationID field from packets associated with the congestion event, and was the target of those packets. The tgtdestinationID field is used by the target of the flow control packet to identify the transaction request flow that needs to be acted upon. For all undefined flowID encodings, there is no action required and the tgtdestinationID is ignored. Field size differences for 8 bit address Dev8 Transport Format (tt=0b00) vs. 16 bit address Dev16 Transport Format (tt=0b01) are shown, additionally the Dev32 (tt=0b10) format can be used. Note: when tt=0b01 there will be a pad after the CRC.

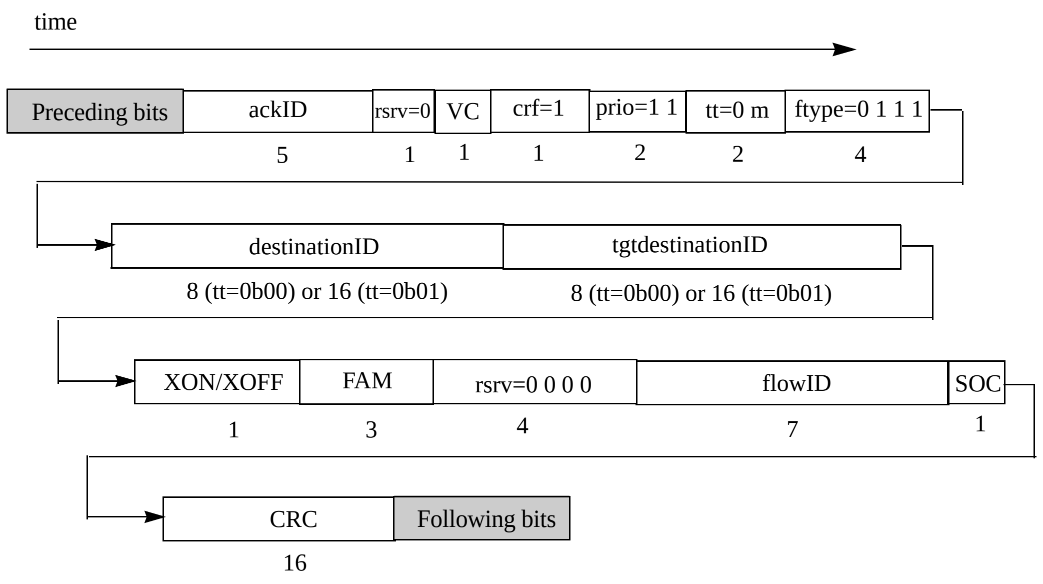


Figure 3-2. LP-Serial Flow Control Packet

Chapter 4 Logical Layer Flow Control Extensions Register Bits

4.1 Introduction

This section describes the Logical Layer Flow Control Extensions CAR and CSR bits that allow an external processing element to determine if a switch or end point device supports the flow control extensions defined in this specification, and to manage the transmission of flow control transactions for a switch processing element. This chapter only describes registers or register bits defined by this specification. Refer to the other RapidIO logical, transport, physical, and extension specifications of interest to determine a complete list of registers and bit definitions for a device. All registers are 32-bits and aligned to a 32-bit boundary.

4.2 Capability Registers (CARs)

4.2.1 Processing Elements Features CAR (Offset 0x10 Word 0)

The Processing Elements Features CAR contains 31 processing elements features bits defined in various RapidIO specifications, as well as the Flow Control Support bit, and Flow Arbitration Participant bit are defined here.

Table 4-1. Bit Settings for Processing Elements Features CAR

Bit	Name	Reset Value	Description
0-19	—		Reserved (defined elsewhere)
20	Flow Arbitration Support	*	Support for flow arbitration 0b0 - does not support flow arbitration 0b1 - supports flow arbitration
21-23	—		Reserved (defined elsewhere)
24	Flow Control Support	*	Support for flow control extensions 0b0 - Does not support flow control extensions 0b1 - Supports flow control extensions
26-31	—		Reserved (defined elsewhere)

* Implementation dependant

4.2.2 Port *n* Control CSR (Block Offsets 0x5C, 7C, ... , 23C)

The Port *n* Control CSR contains 30 bits specifying individual port controls defined in various RapidIO specifications, as well as the Flow Control Participant and Flow Arbitration Participant bits.

Table 4-2. Bit Settings for Port *n* Control CSR

Bit	Name	Reset Value	Description
0-12 (serial)	—		Reserved (defined elsewhere)
13 (serial)	Flow Control Participant	0b0	Enable flow control transactions 0b0 - Do not route or issue flow control transactions to this port 0b1 - Route or issue flow control transactions to this port
14 (serial	—		Reserved (defined elsewhere)
15 (serial)	Flow Arbitration Participant	0b0	Enable Flow Arbitration Transactions: 0b0 - do not route or issue flow arbitration transactions to this port 0b1 - route or issue flow arbitration transaction to this port
16-31 (serial	—		Reserved (defined elsewhere)

Annex A Flow Control Examples (Informative)

A.1 Congestion Detection and Remediation

The method used to detect congestion is implementation specific and is heavily dependent upon the internal packet buffering structure and capacity of the particular switch device. In the example output port buffered switch from “Section 1.1.3, Problem Illustration” on page 10, congestion occurs when some output buffer watermark is exceeded. As long as the watermark is exceeded the output port is said to be in a congested state. The watermark can have different levels when entering the congested state and leaving the congested state.

Fabric elements should monitor their internal packet buffer levels, comparing them on a packet by packet basis to pre-established, locally-defined watermark levels. These levels likely would be configurable depending upon the local element's position within the fabric relative to source endpoints and its particular architecture. On the high watermark side, a level should be selected which is low enough that the remaining buffer space is adequate to provide ample storage for packets in-flight, given a worse-case latency for XOFF CCPs to travel back to the source endpoint and shut off the flow in the endpoint. On the low watermark side (if a watermark is used for XON), a yet-lower level should be selected which meets the following criteria;

a) Provides sufficient hysteresis. When considered in context with the high watermark, it should not be so close as to provide a high flow of XON/XOFF CCP traffic back to the source endpoint.

b) Is set high enough that the switch output buffer does not run dry (underflow) in the typical live-flow case (one or more packets are present in the source endpoint output buffer waiting to be sent when the flow is restarted), given the latency of XON CCP travel back to the source endpoint and restoration of the shut-off flow in the endpoint.

The following two examples are provided to show possible methods for detecting and reacting to congestion:

1. Histogram analysis:
   - — The switch keeps track of packet quantities for the different transaction request flows for which packets are stored in its output buffer.
   - — The switch sorts the transaction request flows according to the number of packets.
   - — The switch selects the 1 to 5 transaction request flows with the most

packets stored in the buffers.

— The switch sends an XOFF flow control request to those transaction request flow sources when the watermark threshold is exceeded, as long as flow control transaction routing is enabled on that switch port. Handling of system critical flows intending to bypass the flow control operation is outside the scope of this document.

— The CCP-targeted sources stop transmitting packets for the indicated transaction request flow and all lower priority transaction request flows.

— The switch sends a flow control XON request to those transaction request flow sources when the watermark drops below the threshold.

— The CCP-targeted sources begin to transmit packets for the indicated transaction request flow and all higher priority transaction request flows.

2. Simple threshold:

— The switch sends an XOFF flow control to the source of every new transaction flow it receives as long as the watermark is exceeded, provided flow control transaction routing is enabled on that switch port. Handling of system critical flows intending to bypass the flow control operation is outside the scope of this document.

— The CCP-targeted sources stop transmitting packets for the indicated transaction request flow and all lower priority transaction request flows.

— The switch sends a flow control XON request to those transaction request flow sources when the watermark drops below the threshold.

— The CCP-targeted sources begin to transmit packets for the indicated transaction request flow and all higher priority transaction request flows.

Note that the first method is reasonably fair in that it targets the source of the data flows that are consuming most of the link bandwidth, and that the second method is unfair in that it indiscriminately targets any source unfortunate enough to have a packet be transmitted while the link is congested.

A.2 Orphaned XOFF Mechanism Description

This timer may take the form of a low precision counter in the end point which monitors the oldest XOFF’d flow at any given time. When a flow first becomes the oldest flow (reaches top of an XOFF’d flow FIFO list within the end point) the timer is reset to its programmed value and begins to count down with time. If it is allowed to elapse without a change to the oldest XOFF’d flow, that flow will be presumed to be orphaned due to lost XON CCP and be restarted as if an XON CCP had been received, with the orphaned flow entry removed from the top of the list and the counter reset to count down for the next oldest XOFF’d flow. The length of the count should be long enough to insure that significant degradation of the flow control function does not occur, on the order of several times the width of the fabric expressed in terms of packet transit time, yet not so large that it would fail to elapse between uncorrelated congestion events. The length of this count shall be programmable through an implementation-dependent register in the end point. The orphaned XOFF mechanism is intended solely as a last-resort mechanism for restarting orphaned flows. It will not be adequate for the purpose of implicit controlled flow reinstatement owing to inherent fairness issues as well as burstyness due to uncontrolled simultaneous multi-flow restart.

A.3 Discussion on Flow Arbitration

The objectives of the flow arbitration protocol are:

1) Conserve resources at the end point, and allow for limited resources to be utilized in a larger system context.
2) Conserve system resources, minimizing protocol overhead.
3) Provide robustness against failures and lost messages as well as provide for low implementation complexity.

Flow arbitration allows for managing resources that are critical to “flows”. A flow is a nexus of a source, a destination, and a physical channel. With the priorities and virtual channels that exist at the physical layer, even a medium system with a few nodes could have 100s or 1000s of flows. Most RapidIO transactions are self contained (as with an IO_WRITE) and thus have a limited “context”. But the data streaming logical type can have a context that spans multiple transactions, and thus needs a persistent resource. The segmentation/reassembly resource is one example of a resource that may be in limited supply in a large system. But, SAR resource management is not the only use of this protocol. Any resource provided by the logical interface to help offload the system may use contexts that span multiple transactions.

As described in the introduction, resources may be managed in three ways:

- The system designer can use end points with enough resources for the worst case combination of flows (fixed)
- The system designer can provide enough resources for the worst case number of flows based on expected traffic, allocating them on a connection by connection basis (static)
- The limited resources can be shared among a larger number of connections on a PDU by PDU basis (dynamic)

It is important to provide some management of resources because an overrun will cause packet loss, at least for the data streaming protocol.

Dynamic allocation is what the protocol defined in this specification provides, but it is not expected to be the sole method of resource allocation. If all the contexts were to use a dynamic protocol, goals #2 and #3, as stated above, might not be met. It is expected that some number of flows will be fixed or static, and only a portion of the lower quality of service flows will arbitrate for some portion of the resources.

System designers are responsible for selecting components that match their strategy for resource management.

Glossary of Terms and Abbreviations

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

C

Congestion. A condition found in output ports of switch and bridge elements characterized by excessive packet buildup in the buffer, when packet entry rate into the buffer exceeds packet exit rate for a long enough period of time.

CCP. (Congestion Control Packet). A packet sent from the point of congestion in the fabric back to the source endpoint of particular flows instructing the source to either turn on or off the flow.

Controlled Flow List. A memory structure associated with controlling elements which holds a list of currently controlled flows, used by the element to turn back on controlled flows.

CRF. Critical Request Flow. For packets or packets of a given priority, this bit further defines which packet or notice should be moved first from the input queue to the output queue (see *RapidIO Part 4: 8/16 LP-LVDS Physical Layer Specification*, Section 1.2.2 and *RapidIO Part 6: 1x/4x LP-Serial Physical Layer Specification*, Section 5.3.3).

F

flowID. Transaction request flow indicator (see *RapidIO Part 1: Input/Output Logical Specification*, Section 1.2.1).

L

Long Term Congestion. A severe congestion event in which a system does not have the raw capacity to handle the demands placed upon it in actual use.

M

Medium Term Congestion. A congestion event in which a frequent series of short term congestion events occur over a long period of time such as seconds or minutes, handled in RapidIO systems by reconfiguration of the fabric by system-level software.

O

Orphaned XOFF Mechanism. A mechanism in an end point which is used to restart the oldest controlled flow within the end point after a certain period of time has elapsed without the flow being XON'd.

P

Performance Collapse. Non-linear behavior found in non- congestion controlled fabrics, whereby reduced aggregate throughput is exhibited with increased load.

S

Saturation Tree. A pattern of congestion identified within the fabric which grows backward from the root buffer overflow towards the sources of all transaction request flows passing through this buffer.

Short Term Congestion. A congestion event lasting up into the dozens or hundreds of microseconds, handled in RapidIO by Logical Layer Flow Control.

T

Topology. The structure represented by the physical interconnections of a switch fabric.

Transaction Request Flow. A series of packets that have a common source identifier and a common destination identifier at some given priority.

U

Ultra Short Term Congestion. A congestion event lasting from dozens to hundreds of nanoseconds, handled in RapidIO by Link Level Flow Control.

Underflow. A condition within output buffers of switches in which the buffer runs dry.

W

Watermark. A predetermined buffer occupancy level indicating either congestion (high watermark) or abatement of congestion (low watermark).

X

XOFF (Transmit Off). A congestion control packet sent from the point of congestion back to the source of a particular flow, telling the source endpoint to shut off the flow.

XON (Transmit On). A congestion control packet sent from the point of congestion back to the source of a particular flow, telling the source endpoint to restart a controlled flow.

RapidIO™ Interconnect Specification Part 10: Data Streaming Logical Specification

3.2, 1/2016

RapidIO.org

Revision History

Revision	Description	Date
1.3	First release	06/09/2004
1.3.a	No technical changes Converted to ISO-friendly templates	02/23/2005
2.0	Technical changes: errata showing 06-04-00002.004; new features showing 06-01-00000.002	06/14/2007
2.1	Technical changes: errata showing 07-07-00000.004	07/09/2009
2.2	Technical changes: errata showing 10-08-00000.003, 10-08-00001.005, Consolidated Comments on 11-01-00000.000	05/05/2011
3.0	Changed RTA contact information. No technical changes.	10/11/2013
3.1	No technical changes.	09/18/2014
3.2	No technical changes.	01/28/2016

Questions regarding RapidIO.org, specifications, or membership should be forwarded to:

RapidIO.org
8650 Spicewood Springs #145-515
Austin, TX 78759
512-827-7680 Tel.

Table of Contents

Chapter 1 Overview

1.1 Introduction 9
1.2 Overview 9
1.3 Features of the Data Streaming Specification 10
1.3.1 Functional Features 10
1.3.2 Physical Features 10
1.3.3 Performance Features 11
1.4 Contents 11
1.5 Terminology 12
1.6 Conventions 13
1.7 Useful References 13

Chapter 2 Data Streaming Systems

2.1 Introduction 15
2.2 System Example 15
2.3 Traffic Streams 16
2.4 Operation Ordering 17
2.5 Class of Service and Virtual Queues 19
2.6 End-to-end Traffic Management 20
2.7 Deadlock Considerations 21

Chapter 3 Operation Descriptions

3.1 Introduction 23
3.2 Data Streaming Protocol 23
3.2.1 Data Streaming Operation 23
3.2.2 Virtual Streams 24
3.2.3 PDU Sequences Within Streams 25
3.2.4 Segments within a PDU 25
3.2.5 Rules for Segmentation and Reassembly 28
3.3 Class of Service and Traffic Streams 29
3.4 Traffic Management 30
3.4.1 Traffic Management Operand 31
3.4.2 On/Off Traffic Management 31
3.4.3 Rate Base Traffic Management 31
3.4.4 Credit Based Traffic Management 32
3.4.5 Rules for Traffic Management 35

Table of Contents

Chapter 4 Packet Format Descriptions

4.1 Introduction.......... 37
4.2 Type 9 Packet Format (Data-Streaming Class) 37
4.3 Type 9 Extended Packet Format.......... 40
4.3.1 TM Operand.......... 42
4.3.2 Basic Traffic Management.......... 42
4.3.3 Rate Based Traffic Management 43
4.3.4 Credit Based Traffic Management.......... 44

Chapter 5 Data Streaming Registers

5.1 Introduction.......... 45
5.2 Register Summary.......... 45
5.3 Reserved Register, Bit and Bit Field Value Behavior 46
5.4 Additions to Existing Registers 48
5.5 Capability Registers (CARs).......... 50
5.5.1 Source Operations CAR.......... 50
5.5.2 Destination Operations CAR 51
5.5.3 Data Streaming Information CAR 52
5.6 Command and Status Registers (CSRs).......... 53
5.6.1 Data Streaming Logical Layer Control CSR 53

Annex A   VSID Usage Examples

A.1 Introduction.......... 55
A.2 Background.......... 55
A.3 Packet Classification.......... 55
A.3.1 Sub-port Addressing at the Destination.......... 56
A.3.1.1 DSLAM application.......... 56
A.3.1.2 VOIP application 56
A.3.2 Virtual Output Queueing - Fabric On-ramp 56
A.4 System Requirements.......... 57
A.4.1 UTOPIA to RapidIO ATM bridge.......... 57
A.4.2 Network processor 57
A.4.3 CSIX to RapidIO interface 57
A.4.4 10Gb Metropolitan Area Network interface.......... 58

List of Figures

1-1 End to End Communication Circuit....................10
2-1 Example of a RapidIO-Based Networking System....................15
2-2 Mapping Virtual Streams at the System Ingress....................19
2-3 Mapping Virtual Streams at the System Egress....................20
2-4 Class Based and Stream Base Traffic Management....................21
3-1 Data Streaming Operation....................24
3-2 Virtual Streams....................24
3-3 PDU Segmentation and Reassembly Example 1....................27
3-4 PDU Segmentation and Reassembly Example 2....................28
3-5 Traffic Sorting Based on CoS ID....................30
3-6 Typical Credit Based Flow Control Example....................34
3-7 Credit Based Protocol Example....................35
4-1 Single Segment Type 9 Packet Bit Stream Format Example....................38
4-2 Start Segment Type 9 Packet Bit Stream Format Example....................39
4-3 Continuation Segment Type 9 Packet Bit Stream Format Example....................39
4-4 End Segment Type 9 Packet Bit Stream Format....................40
4-5 Traffic Management Bit Stream Format....................40

List of Figures

Blank page

List of Tables

4-1 Specific Field Definitions and Encodings for Type 9 Packets 37
4-2 Specific Field Definitions and Encodings for Type 9 Packets 38
4-3 Extended Header Fields 41
4-4 Basic Traffic Management Message Formats....................................42
4-5 Rate Based Traffic Management Message Formats 43
4-6 Credit Based Traffic Management Message Formats....................................44
5-1 Data Streaming Register Map....................................45
5-2 Configuration Space Reserved Access Behavior....................................46
5-3 Bit Settings for Logical/Transport Layer Error Detect CSR 48
5-4 Bit Settings for Logical/Transport Layer Error Enable CSR....................................48
5-5 Bit Settings for Source Operations CAR 50
5-6 Bit Settings for Destination Operations CAR....................................51
5-7 Bit Settings for Data Streaming Information CAR....................................52
5-8 Bit Settings for Data Streaming Logical Layer Control CSR....................................53

List of Tables

Blank page

Chapter 1 Overview

1.1 Introduction

This chapter provides an overview of the *RapidIO Part 10: Data Streaming Logical Specification*. The goal of the specification is to combine the need for efficiency, flexibility, and protocol independence in order to minimize the resources necessary to support a data plane interconnect fabric, and to maintain compatibility and fully inter-operate with the rest of the RapidIO specifications. Implementation of this specification is optional.

The rationale for this optimization is based upon the assumption that platforms are expected to produce many times more revenue than the initial cost of the platform. For example, a platform is expected to produce 10 times the revenue vs. its initial capital costs. If that same platform could cost 10% more but allow 10% more resources for producing revenue rather than doing fabric support, the result would be a significant net gain on the investment. Therefore, enabling more intelligence within the system fabric and relieving the system processing resources to produce revenue, even if that fabric is more expensive, is believed to be a good trade-off.

The features of the data streaming specification define virtual mechanisms in simple forms for building cost sensitive systems and also provides for complex high functioning fabrics for more demanding applications.

It is assumed that the reader has a thorough understanding of the other RapidIO specifications and of data plane equipment and applications in general.

1.2 Overview

Standard encapsulation schemes have been developed for the transmission of datagrams over most popular LANs. A number of different proposals currently exist for the encapsulation of one protocol over another protocol [RFC1226, RFC1234, RFC1701]. The data streaming logical specification defines a mechanism for transporting an arbitrary protocol over a standard RapidIO interface, and addresses interconnection between elements in an end-to-end data communications circuit. The protocol has been carefully designed to provide complete compatibility and inter-operability with existing RapidIO specifications.

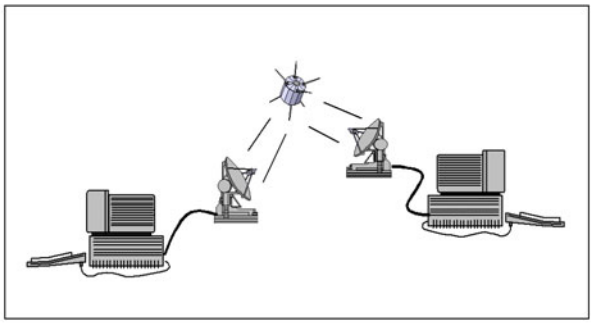


Figure 1-1. End to End Communication Circuit

The defined encapsulation methodology provides for the multiplexing of different network-layer protocols simultaneously over the same link and provides a common solution for easy connection of a wide variety of hosts, bridges and switches. It is envisioned that a RapidIO system will be capable of carrying a wide variety of data types, supporting a diverse set of protocol regimens concurrently.

The Data Streaming Protocol also includes a methodology for end-to-end traffic management. Loosely coupled systems with individual traffic managers admitting traffic to a fabric have to rely on statistical performance and back pressure to try and optimize use of the fabric resources. End-to-end traffic management allows traffic mangers at the ingress endpoints to work in concert with egress endpoints to coordinate traffic flows.

1.3 Features of the Data Streaming Specification

The following are features of the RapidIO data streaming specification designed to satisfy the needs of various applications and systems:

1.3.1 Functional Features

- Protocol encapsulation, independent of the protocol being encapsulated.
- Support for Protocol Data Units (PDUs) of up to 64k bytes through Segmentation and Reassembly (SAR).
- Support for hundreds of traffic classes.
- Support for thousands of data streams between end points.
- Support for concurrent interleaved PDUs between end points.
- Seamless inter-operability with other RapidIO specifications.

1.3.2 Physical Features

- Packet definition is independent of the choice of physical layer interconnection 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.
- No dependencies exist on the bandwidth or latency of the physical fabric.
- The protocol requires in-order packet transmission and reception; out-of-order packet delivery is not tolerated.
- Certain devices have bandwidth and latency requirements for proper operation. The data streaming logical layer specification does not preclude an implementation from imposing these constraints within the system.

1.3.3 Performance Features

- Packet headers are small to minimize the control overhead and be organized for fast, efficient assembly and disassembly.
- Multiple transactions are allowed concurrently in the system, otherwise a majority of the potential system throughput is wasted.
- Multiple end point to end point concurrent data streams are supported for high fabric utilization.
- Optional end-to-end traffic management for advanced fabric designs.

1.4 Contents

Following are the contents of the *RapidIO Part 10: Data Streaming Logical Specification:*

- Chapter 1, “Overview,” is an overview of the data streaming logical specification.
- Chapter 2, “Data Streaming Systems,” introduces system issues such as transaction ordering and deadlock prevention.
- Chapter 3, “Operation Descriptions,” describes the set of operations and transactions supported by the RapidIO data streaming protocol.
- Chapter 4, “Packet Format Descriptions,” contains the packet format definitions for the data streaming specification.
- Chapter 5, “Data Streaming Registers,” describes the visible register set that allows an external processing element to determine the data streaming capabilities, configuration, and status of a processing element using this logical specification. Only registers or register bits specific to the data streaming 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.
- Annex A, “VSID Usage Examples,” contains a number of examples of how the virtual stream identifier can be used in a system.

1.5 Terminology

The data streaming logical specification introduces some new terms:

Protocol Data Unit - (PDU) A self contained unit of data transfer comprised of data and protocol information that defines the treatment of that data.

Virtual Stream ID - (VSID) an identifier comprised of several fields in the protocol to identify individual data streams.

Virtual input Queue (ViQ), Virtual output Queue (VoQ) - an intermediate point in the system where one or more virtual streams may be concentrated.

Class of service - (cos) a term used to describe different treatment (quality of service) for different data streams. Support for class of service is provided by a class of service field in the data streaming protocol. The class of service field is used in the virtual stream ID and in identifying a virtual queue.

StreamID - a specific field in the data streaming protocol that is combined with the data streams's transaction request flow ID and the source ID or destination ID from the underlying packet transport fabric to form the virtual stream ID.

Segment - A portion of a PDU.

Segmentation - a process by which a PDU is transferred as a series of smaller segments.

Segmentation context - Information that allows a receiver to associate a particular packet with the correct PDU.

Ingress - Ingress is the device or node where traffic enters the system. The ingress node also becomes the source for traffic into the RapidIO fabric. The terms ingress and source may or may not be used interchangeably when considering a single end to end connection.

Egress - Egress is the device or node where traffic exits the system. The egress node also becomes the destination for traffic out of the RapidIO fabric. The terms egress and destination may or may not be used interchangeably when considering a single end to end connection.

Physical Channel ID - Identifies a physical channel using the virtual channel, critical request flow, and priority physical layer fields. The physical channel ID is used to determine how to treat a packet with respect to priority, bandwidth allocation, and transaction ordering.

RapidIO flow - A RapidIO flow is a nexus of the source ID, destination ID and physical channel.

Refer to the Glossary at the back of this document for additional definitions.

1.6 Conventions

||	Concatenation, used to indicate that two fields are physically associated as consecutive bits
ACTIVE_HIGH	Names of active high signals are shown in uppercase text with no overbar. Active-high signals are asserted when high and not asserted when low.
$\overline{\text{ACTIVE\_LOW}}$	Names of active low signals are shown in uppercase text with an overbar. Active low signals are asserted when low and not asserted when high.
italics	Book titles in text are set in italics.
REG[FIELD]	Abbreviations or acronyms for registers are shown in uppercase text. Specific bits, fields, or ranges appear in brackets.
TRANSACTION	Transaction types are expressed in all caps.
operation	Device operation types are expressed in plain text.
n	A decimal value.
[n-m]	Used to express a numerical range from n to m.
0bnn	A binary value, the number of bits is determined by the number of digits.
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.
x	This value is a don’t care.

1.7 Useful References

[RFC791] Postel, J., "Internet Protocol", STD 5, RFC791, September 1981

[RFC1226] Kantor, B. "Internet Protocol Encapsulation of AX.25 Frames", RFC1226, University of California, San Diego, May 1991.

[RFC1234] Provan, D. "Tunneling IPX Traffic through IP Networks", RFC 1234, Novell, Inc., June 1991.

[RFC1700] J. Reynolds and J. Postel, "Assigned Numbers", RFC1700, October 1994.

[RFC2460] S. Deering, R. Hinden, "Internet Protocol, Version 6 (IPv6)", RFC2460, December 1998.

[RFC1884] Hinden, R., and S. Deering, Editors, "IP Version 6 Addressing Architecture", RFC1884, Ipsilon Networks, Xerox PARC, December 1995.

[RFC2004] C. Perkins, "Minimal Encapsulation within IP", RFC2004, October 1996.

Chapter 2 Data Streaming Systems

2.1 Introduction

This overview introduces the role of the data streaming logical layer in an overall system. It provides some possible use examples. See Annex A, “VSID Usage Examples”, for more example details.

2.2 System Example

Figure 2-1 shows a block diagram of an example RapidIO-based networking system in which protocol encapsulation is required. A number of typical data path type devices are connected with a variety of proprietary and/or somewhat standard interfaces and the entire system is tied together with a RapidIO switching fabric of some topology.

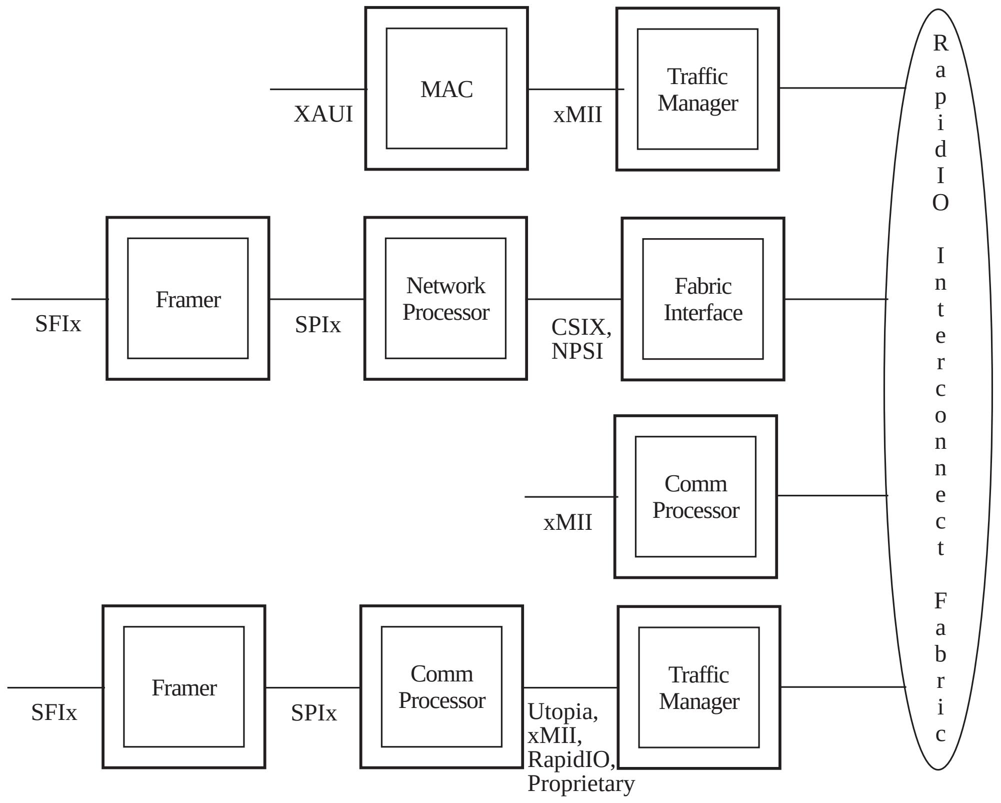


Figure 2-1. Example of a RapidIO-Based Networking System

Data “streams” represent logical connections between an ingress port and an egress port. A connection spans the transfer of multiple PDUs. The transfer of PDUs may be separated by discrete intervals of time, based on the arrival of data at the ingress. Transfer between an ingress process and an egress process is unidirectional. An I/O device may be bi-directional, containing both an ingress process and an egress process. These processes are usually completely independent consisting of separate streams in each direction.

A given ingress may service hundreds, thousands, even millions of streams at any given time depending on how specifically a PDU is classified. Traffic may be lumped into a single stream, or classified by user and application to form millions of data streams.

Data streaming transactions differ from most other RapidIO transactions in two ways: they must accommodate larger variably sized data transfers, and the transactions are not acknowledged with a response packet. The *data streaming logical layer* is intended to support data from a variety of hardware and processing devices. These devices have a variety of different interfaces, protocols, and degrees of sophistication. This specification is intended to enable these kinds of devices to exist on the RapidIO interconnect.

2.3 Traffic Streams

A stream identifier identifies independent streams of traffic between the end producer (for example, a web server) and end consumer (for example, a home personal computer) of the encapsulated data. Stream identifiers vary with protocol and may include multiple fields from the various networking layers included in the protocol. A unit of data that contains a discrete identifier is called a Protocol Data Unit, or PDU. A PDU may or may not have an ordering relationship with another PDU being transmitted between that same producer and consumer, depending upon the higher layer protocol being carried. A traffic stream is a series of PDUs that have an ordering relationship between each other. A PDU has no ordering relationship with a PDU from different producers and consumers pairs.

The data streaming logical layer uses a **virtual stream identifier** (VSID) to allow multiple end to end traffic streams of PDUs to be uniquely identified and managed concurrently within the RapidIO system. Creation of a VSID is done by performing a protocol specific classification process on a PDU. The complexity of the classification process is directly proportional to the sophistication of the system as required by the application. The VSID allows the traffic to be reassociated with an appropriate application at the egress without having to perform a second protocol-specific classification. A VSID is comprised of fields from the data streaming protocol: **source** and **destination ID** from the underlying packet transport fabric, **class of service**, and **streamID**.

2.4 Operation Ordering

A transaction request flow is defined as an ordered sequence of request transactions comprising a specific PDU from a given source ID to a given destination ID. Each packet in a transaction request flow has the same source identifier and the same destination identifier. All traffic streams are mapped onto transaction request flows. These flows may also be shared with other RapidIO logical layers transactions, and therefore the relationship between streams, traffic classes, virtual queues, and all RapidIO transaction request flows are implementation specific.

There may be multiple transaction request flows between a given source ID and destination ID pair. When multiple flows exist between a source ID and destination ID pair, the flows are distinguished by a flow indicator referred to as a “flowID”, introduced in the *RapidIO Part 1: Input/Output Logical Specification*. RapidIO allows multiple transaction request flows between any source ID and destination ID pair. Any number of transaction request flows may exist between the two IDs. The flowID represents the lowest level of traffic management in a RapidIO system as that is the construct mapped directly on to the switch fabric itself.

The transaction request flows between each source and destination ID pair may be allocated to different virtual channels in the underlying fabric and may also be prioritized within a channel. The flows are labeled and identified alphabetically as in the other logical layer specifications, and the channels labeled and identified numerically with channel then priority, starting with 0 as first channel or lowest priority, then 1 as second channel or next lowest priority, etc. For example, flowID 0A is channel 0 flow A, flowID 1C is channel 1 flow C, flowID 3E is channel 3 flow E, and so on. This flow information provides class of service information when mapped by the application to the switch fabric.

Allocation of transaction request flows to virtual channels and the relative priority within each channel is application dependent. A special case is a single virtual channel application which must follow the same prioritization of flows and labeling as the other logical layers (flowID A, flowID B, flowID C, etc.). The channel label (0) is dropped. This channel may include traffic from the other logical layers.

At the link level, when multiple transaction request flows within the same virtual channel exist between a given connected source and destination ID 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. There are no ordering rules for flows in different channels. A traffic stream being transmitted between a source and a destination ID pair must utilize the same flowID value so that the ordering of the traffic stream is maintained. As a class of service indicator, the physical channel ID is used by the underlying RapidIO fabric to determine how to treat a packet with respect to other packets with respect to priority and ordering. It is expected that in a mixed control and data plane application that both I/O logical and data streaming transaction request flows will exist in a RapidIO system simultaneously, possibly between the same ID pairs.

To support transaction request flows, all devices that support the RapidIO data streaming logical specification shall comply as applicable with the following Fabric Delivering Ordering and End point Completion Ordering rules. Note that these rules are very similar and complementary to the rules specified in *RapidIO Part 1: Input/Output Logical Specification*.

Fabric Delivery Ordering Rules

1. **Transactions within a transaction request flow (same source identifier, same destination identifier, same flowID, same PDU) 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. **Request transactions that have the same source (same source identifier) and the same destination (same destination identifier) within the same virtual channel but with 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) or different virtual channels are unordered with respect to each other.**

End point Completion Ordering Rules

1. **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.**

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.

2.5 Class of Service and Virtual Queues

Data streaming systems may support thousands, even millions of active data streams. These streams are eventually interleaved onto the single underlying packet transport fabric. The process for deciding which streams may share common resources is sometimes referred to as virtual queueing. To facilitate virtual queueing at the ingress and/or egress of the fabric, and to provide for more sophisticated management of traffic streams, the data streaming logical layer provides a class of service (cos) identifier. The cos field exists to provide a common semantic as to how the traffic stream is to be treated. The relationship between the ingress/egress cos and the end to end flowID assigned to the traffic stream is implementation specific.

At the ingress to the fabric, thousands of streams may be combined into fewer virtual output queues (VoQs) using just the **destination ID** and the **class of service** portions of the VSID as shown in Figure 2-2. The cos field defined by this specification is comprised of one byte. The number of bits utilized by a particular device depends upon the number of data buffering structures implemented, but are always from the most significant bit of the cos field to the least significant bit. For example, a device with two buffering structures (or “bins”) maps a packet to a bin using bit 0, a device with four bins maps a packet to a bin using bits 0 and 1, and so on.

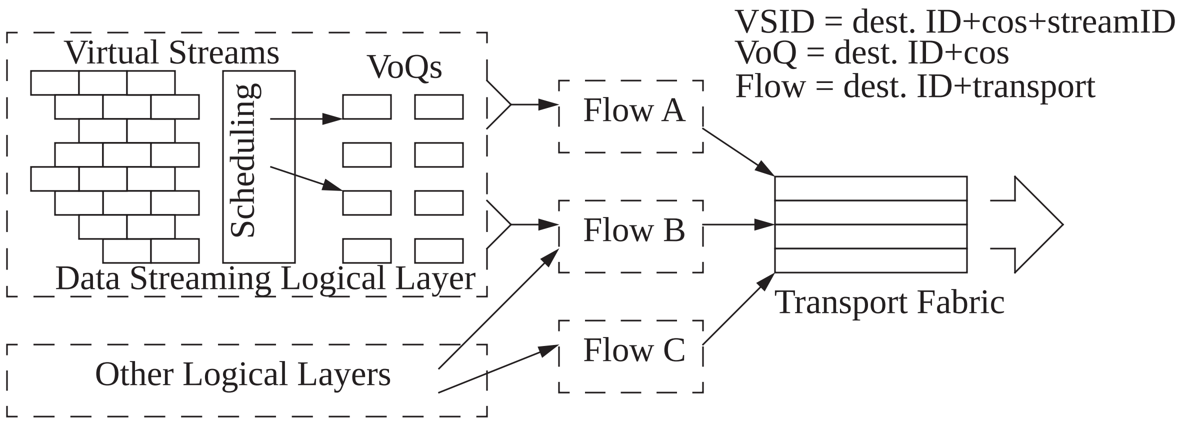


Figure 2-2. Mapping Virtual Streams at the System Ingress

As shown in Figure 2-2, as the virtual output queues are mapped on to the flowIDs and then on to the underlying packet transport fabric, they may be intermingled with other logical layer transactions. The use of the transport fabric must account for the needs of the total environment and is application and implementation specific. End points designed to support a wide variety of applications for data streaming should offer some flexibility in how virtual queues are mapped down on to the transport fabric in the implementation.

A reverse process (virtual input queueing) may or may not occur at the destination. If there is a critical resource needed to process traffic on egress from the fabric, the system designer may choose to fan the traffic back out into virtual queues. This allows the fabric egress processing to re-prioritize utilization of the critical resource.

This is illustrated in Figure 2-3.

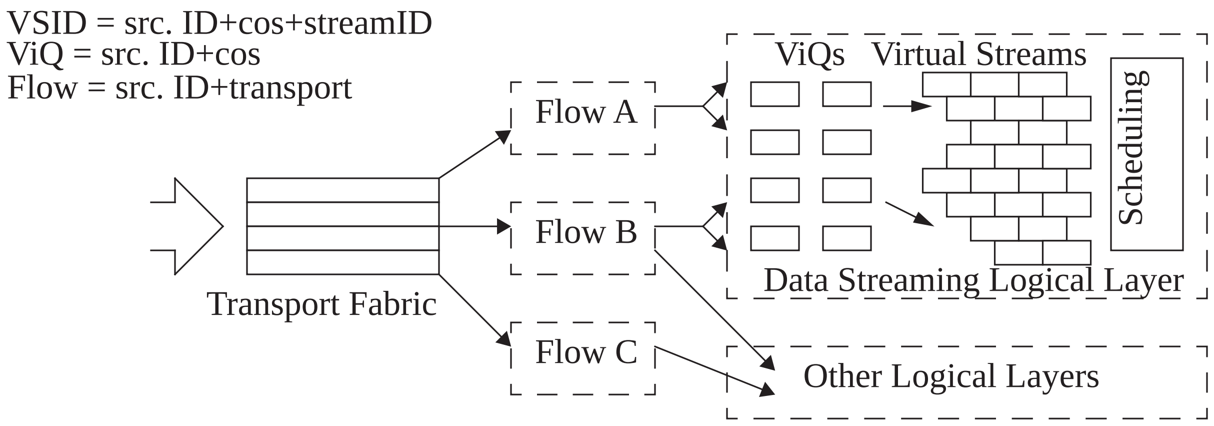


Figure 2-3. Mapping Virtual Streams at the System Egress

A switch device may choose to utilize the information carried in the cos field by acting as a “virtual” end point, removing the traffic streams from the underlying packet transport fabric, reassembling the individual PDUs, and fanning the streams back out into some larger number of queues. It then re-injects the traffic streams back into the underlying transport fabric re-ordering the traffic using the cos. This permits intervening devices to participate in the overall assurance of quality of service in the system.

2.6 End-to-end Traffic Management

Other RapidIO specifications provide for traffic management at the physical layer, and for flows at the logical layer. The traffic management for Data Streaming allows endpoints to coordinate traffic flows between class based queues and stream based queues. The protocol includes a hierarchy of methods and field specifiers to be adaptable to a wide variety of queueing and management designs. The use of Data Streaming’s traffic management is optional, and may be inter-operable only between endpoints with similar capabilities. Endpoints that support the same traffic management capabilities will be able to exchange traffic management packets related to those capabilities. The implementation specific nature of the algorithms which determine traffic management in different endpoints may lead to unexpected system behavior.

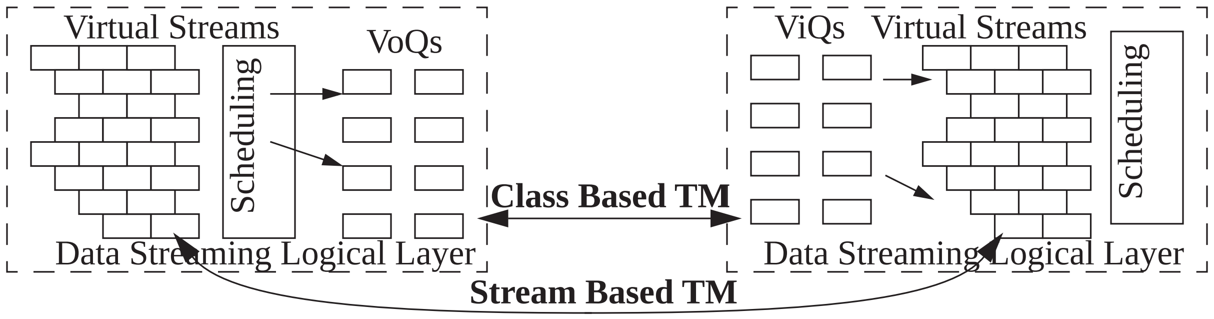


Figure 2-4. Class Based and Stream Base Traffic Management

2.7 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 data streaming logical specification does not have any dependency loops since the defined operations do not require responses. However, a real RapidIO system is required to support the I/O logical maintenance operation, and will very likely require the use of other logical operations for control functions. Support for these other logical operations may have significant deadlock considerations for processing element and system designs.

Blank page

Chapter 3 Operation Descriptions

3.1 Introduction

This chapter describes the RapidIO data streaming protocol. The field encodings and packet formats are described in Chapter 4, “Packet Format Descriptions.”

Data path data movement through a machine has requirements that are significantly different than those for control path and traditional DMA functions. Many times this data is encapsulated data, which also many times contains further encapsulated data. For example, the data moving through the system may be encapsulated Ethernet packets, which may in turn be encapsulating TCP/IP packets.

This style of data movement is typically not address-based as with DMA type I/O, and consequently follows a queue based message passing paradigm. Data path data movement also has much more complex requirements in the area of class (or quality) of service than control path communications, and generally requires managing a number of queues at the egress of the system. There is also a need to be able to identify and manage many thousands of data traffic streams that pass through a RapidIO based data path system. The data being passed through the RapidIO system may not be directly generated or consumed by the device connected to the RapidIO portion of the machine, but instead by a distant end user, such as a personal computer attached to a LAN. This necessitates the addition of a new protocol to the RapidIO logical layers, the data streaming protocol.

The RapidIO data streaming protocol uses request transactions through the interconnect fabric as with other RapidIO operation protocols. Since many data movement protocols guarantee data delivery in an upper layer protocol, the generation of responses indicating completion are not needed. Such upper layer protocols may also allow data to be discarded if necessary, for example, under error or fabric congestion conditions.

3.2 Data Streaming Protocol

This section describes the RapidIO data streaming protocol.

3.2.1 Data Streaming Operation

A data stream represents a logical connection between a source and a destination pair. A stream may consist of multiple transactions and requires the allocation of resources at both the source and the destination. This may be done in advance of any data transfer, or in response to receiving a new transaction. Since streams are virtual constructs between source and destination pairs, they may be reused for different data transfers at any time as long as the source and destination pair are both synchronized as to the stream usage.

A data streaming operation consists of individual data streaming transactions, as shown in Figure 3-1. A series of transactions is used to send PDUs between two end points. The data streaming protocol is completely independent of the PDU's native protocol.

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

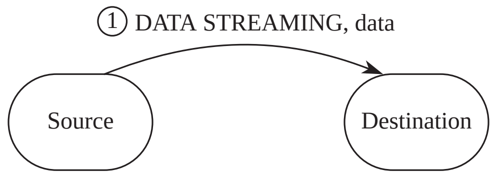


Figure 3-1. Data Streaming Operation

3.2.2 Virtual Streams

A stream is represented by a unique virtual stream identifier, or VSID. This identifier represents the handling of all PDUs within the stream for the duration of a PDU's transit of the RapidIO fabric. The identifier is created by performing some form of protocol specific classification of the PDU. The classification can be as complex or as simple as the application warrants. The VSID allows this protocol specific classification to take place one time at the ingress to the fabric. After that, the handling of the PDU is protocol independent.

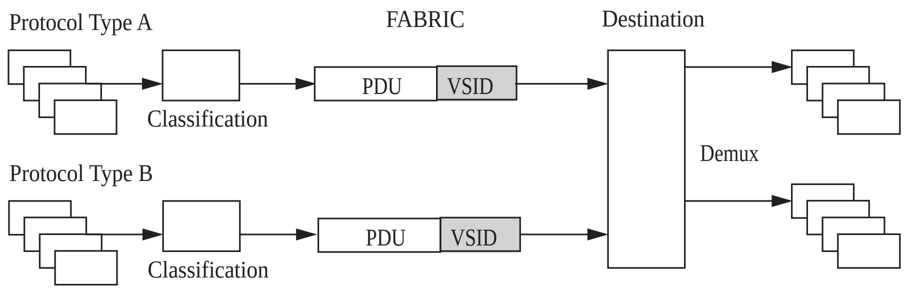


Figure 3-2. Virtual Streams

The VSID is used at the destination to "reclassify" the PDU. This sorts the data back into contexts that can now be protocol specific again. This virtual addressing model eliminates the need for the source and the destination to align the use of buffers and other resources. Therefore, the VSID can be used to carry a wide variety of information about a stream through the system, such as the protocol being encapsulated, de-multiplexing exit port IDs instructions, very fine grained buffer management, etc., as required for a specific application.

The VSID is a “key” comprised of multiple fields. These fields are the source/destination ID, cos, and streamID.
From the source’s viewpoint: **destination ID+cos+streamID** represents a unique stream.
From the destination’s viewpoint: **source ID+cos+streamID** represents a unique stream.

By using the complete key, each source and destination pair is free to allocate the use of these fields independently. Some examples of how the VSID may be applied in a system are described in Appendix A, “VSID Usage Examples,” on page 55.

NOTE:VSIDs

Destinations are permitted to define their use of Virtual Stream IDs to pre-associate certain kinds of traffic with certain end processes. Sources shall be able to label a stream with any VSID necessary to inter-operate with the largest number of possible destination implementations.

3.2.3 PDU Sequences Within Streams

As described earlier, a traffic stream may consist of a sequence of related PDUs that have ordering requirements between each other. A stream of PDUs is transmitted one PDU at a time to preserve the required ordering. PDUs that do not have an ordering relationship may be separated into different streams or may be interleaved in common streams. A stream is identified by the interconnect fabric by the combination of the destination ID and either the cos field or the flowID, depending upon the complexity of the fabric, as described in “Section 2.5, Class of Service and Virtual Queues” on page 19.

Only one PDU from any given stream will be transmitted at a time at the source, but fabric conditions may result in multiple PDUs in transit. The fabric must guarantee that delivery of PDUs (and segments of PDUs as described below) remain in order. A fabric may load balance traffic through multiple paths on a stream by stream basis.

3.2.4 Segments within a PDU

The basic mechanism of segmentation defines a general methodology to provide for larger PDUs than are accommodated by the standard 256 byte limit on a RapidIO data payload. The standard industry term for this function is “Segmentation and Reassembly”, or SAR. A PDU that is to be transmitted from the initial producer to the final consumer is broken up (segmented) into a series of blocks of data. The consumer “reassembles” that data back into the original PDU. The maximum size of a PDU that a particular destination can accept is specified in a CAR (see Chapter 5, “Data Streaming Registers”). The system must be configured in accordance with these limitations.

The block size used for the segmentation process is specified by the Maximum Transmission Unit, or MTU, parameter. The MTU is defined in Chapter 5, "Data Streaming Registers". The MTU is a system-wide parameter agreed to by all processing elements participating in the SAR process. By managing the MTU size for the system, the variability in latency for the system can be controlled.

A data streaming transaction is also referred to as a segment. The transmission of a PDU for any given stream may result in one or more transactions (segments). A typical sequence is made up of three types of transactions, a start segment, some number of continuation segments, and an end segment. Start segments and continuation segments are always filled to the MTU size. End segments are variable in size containing the remainder of the PDU. If a PDU is equal to or less than the MTU size, it is carried in a single segment. A single segment may also be variable in size, matching the PDU payload. Since flowIDs and the cos are assigned on a PDU basis, all segments of a PDU must also have that same flowID and cos assignments.

A start segment contains the necessary fields to identify the VSID and "open" a *segmentation context*. The segmentation context for a stream is defined as the combination of the source ID and the flowID, and is used by a receiver to reassociate the segments of a particular PDU. Using **source ID+physical channel ID** allows each source and destination pair to have one PDU for each physical channel ID that is explicitly supported by the system interleaved in the fabric at any one point in time. Note that for a destination device that can be a multicast target and/or supports multiple destination device IDs, the **destination ID** for a PDU must also be included as part of the segmentation context in order to prevent possible PDU corruption at the destination device. The VSID is used when opening a segmentation process at the destination to associate the PDU with its stream since the continuation and end segments do not carry that information. After the receipt of the end segment, the segmentation context is "closed" (the sending processing element has an analogous definition for open and closed). The stream and PDU associated with a segmentation context is not permitted to change during the time that the context is open.

Since there may be a large number of PDU sources and concurrent contexts per source, the amount of context state that a destination may have to handle can potentially get very large. The number of contexts that can be supported by a particular destination end point is specified in a CAR (see Chapter 5, "Data Streaming Registers"). These segmentation contexts must be allocated to sources by system software.

For efficiency, information as to which block of the PDU is contained in a specific packet is not included in the header. This requires that the transmitter issue the sequence starting with the first block of the PDU and proceeding sequentially through the PDU, and requires the underlying transport fabric to deliver the sequence to the data streaming logical layer in the issued order.

Figure 3-3 shows a 24 byte PDU that is to be segmented for transmission, with an eight byte MTU (note that an eight byte MTU is not permitted in this specification; it is used to simplify the illustration). Since the PDU is divisible by the size specified as the MTU, all data payloads are exactly that size and no padding is necessary. The sender takes byte 0 (the first byte of the PDU) through byte 7 as the data payload to transmit in the start segment. The second data payload consists of bytes 8 through 15, which is transmitted in a continuation segment. The last data payload consists of bytes 16 through 23, which is transmitted in the end segment. Since the data payloads are required to be delivered to the receiver's data management hardware in order of transmission, the receiver can correctly reassemble the original PDU when all three packets have arrived.

To guarantee the packet ordering, all packets making up an individual PDU and all PDUs in a stream must be in the same transaction request flow, as described in "Section 2.4, Operation Ordering" on page 17.

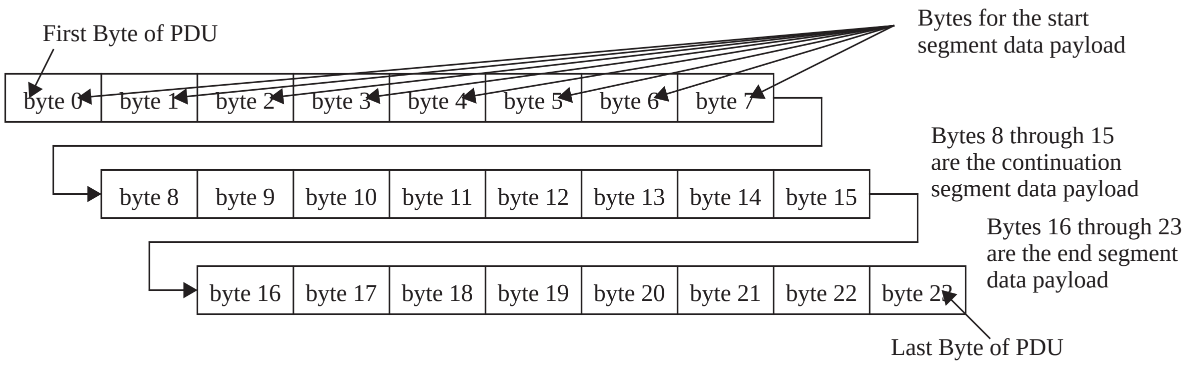


Figure 3-3. PDU Segmentation and Reassembly Example 1

Figure 3-4 shows an example of a similar situation, except that this time the PDU is 21 bytes. In this case, the end segment has a data payload that is less than the specified MTU, and also has a pad byte to round out the data payload to be a multiple of half-words. A bit in the end segment (the "P" bit) indicates the presence of the pad byte. An additional bit (the "O" bit) indicates that the data payload has an odd number of half-words and is therefore oddly aligned. The number of half-words in the data payload as well as the presence of a pad byte can be determined from a PDU length field contained in the end segment header.

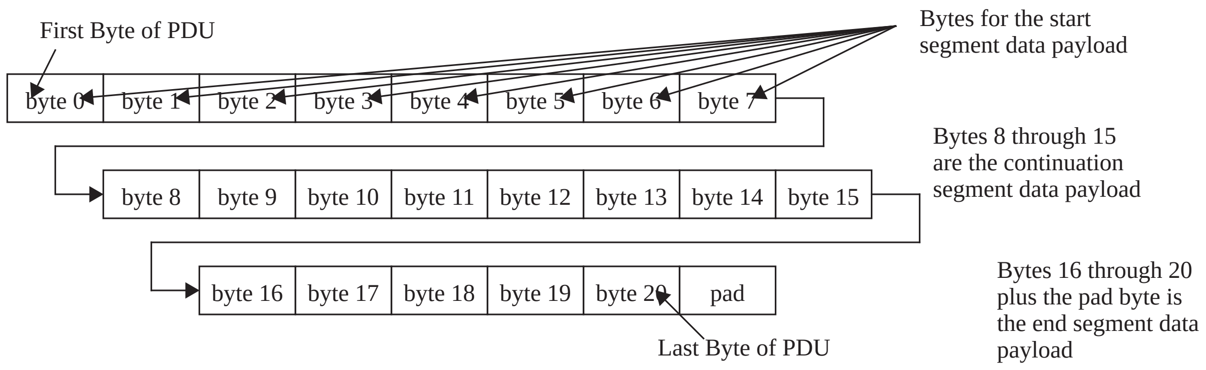


Figure 3-4. PDU Segmentation and Reassembly Example 2

3.2.5 Rules for Segmentation and Reassembly

Segmentation (source)

1. In order to limit implementation complexity due to possible PDU ordering issues, the following conditions must be met:
   – No more than one PDU from a given stream shall be segmented at a time
   – No more than one PDU from a given RapidIO flow shall be segmented at a time
   – PDUs associated with different RapidIO flows may be segmented concurrently
2. Segments are filled with bytes from the PDU in order as shown in Figure 3-3 and Figure 3-4.
3. The first segment is marked as start segment (see section 4).
4. The start segment is filled to the end of the PDU data or to the MTU size.
5. If the end of the PDU data is encountered, the start segment then re-marked as a single segment.
6. If the start segment reaches MTU size (and there is remaining PDU data), the start segment is encapsulated, and a continuation segment is opened.
7. Continuation segments are filled to MTU size from the PDU data, in order.
8. When the end of PDU data is encountered, the segment is marked as the end segment. The end segment data payload size may be less than or equal to the MTU size.
9. If the source wishes to abort a PDU transmission, it sends an end segment with no data payload and with the length field set to zero.

Reassembly (destination)

1. Upon receiving a segment with a start bit, the reassembly unit opens a "context" containing the virtual stream ID and associates it with the segmentation context consisting of the source ID, the destination ID, and the physical channel ID. (The destination ID is only required for devices supporting multicast and/or multiple destination IDs.)
2. The reassembly process transfers the entire payload into the reassembly buffer in order. The amount of data transferred is counted for comparison to the length field.
3. If the packet is a single segment, the amount of payload data must be equal to or less than the MTU size or the PDU is defective.
4. If the packet is a start segment and the payload data does not match the MTU size the PDU is defective.
5. Reassembly continues with continuation packets. All continuation packets must match the MTU size or the PDU is defective. All data transferred to the reassembly buffer is counted.
6. An end segment terminates the reassembly process. An end segment may be received immediately after a start segment. The data payload size must be less than or equal to the MTU size or the PDU is defective. The data from the end segment is transferred according to the data payload size and counted.
7. Once all the data has been reassembled, the length (provided by the end segment packet header) is checked against the received data count. A mismatch indicates a lost continuation segment and the PDU is defective.
8. Receiving a continuation or end segment on a closed context indicates a lost start segment and the PDU is defective.
9. Receiving a start or single segment on an open context indicates a lost end segment and the PDU is defective. The existing context is closed, and the new context is opened.

In all cases, a defective PDU results in discarding the entire PDU. The method used for reporting the discard event is beyond the scope of this specification. It may be desirable for a destination to have a timeout as part of the lost packet detection mechanism, but the definition and time interval are also outside of the scope of this specification.

3.3 Class of Service and Traffic Streams

A virtual stream ID is partitioned into three pieces as previously discussed: port (identified by the source/destination ID), class (the cos field), and the stream identifier (the streamID field). These fields form a specific hierarchy for transitioning packets from highly individualized streams to coarser groupings of traffic. At the fabric ingress, egress, and potentially at interim points (where competition for resources may occur) the traffic may be resegregated and queued by class. In the packet transport fabric, switching is done by destination ID and the

mapped flowID, as described in Section 2.4. The full class of service identifier (CoS ID) is a subset of the VSID. It consists of the source/destination ID (or ingress/egress port) plus the cos field.
Ingress queueing should be based on: **destination ID+cos**
Egress queueing should be based on: **source ID+cos**
as shown in Figure 3-5.

Including the source or destination ID in the CoS ID allows the class of service to be specific to the source and destination pairing.

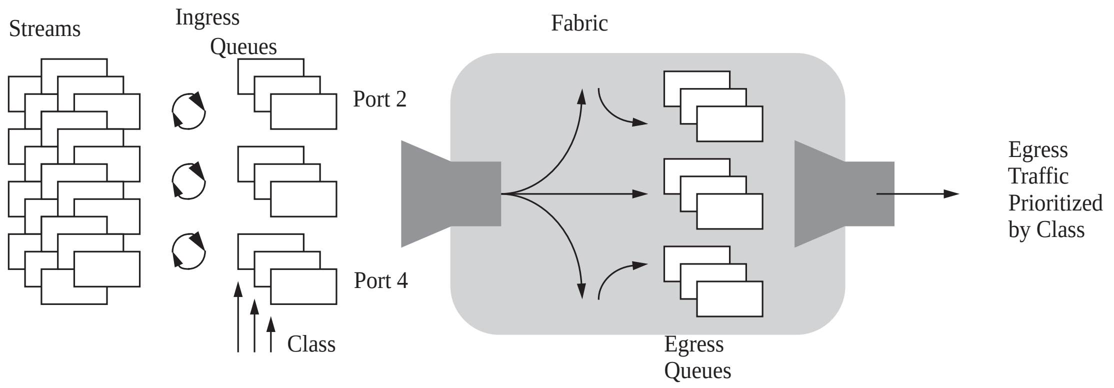


Figure 3-5. Traffic Sorting Based on CoS ID

The cos field shall be used beginning with the MSB (bit 0) using the necessary number of bits for the number of classes supported.

Bit 0 - 2 Classes of Service

Bits 0, 1 - 4 Classes of Service

Bits 0, 1, 2 - 8 Classes of Service supported

etc.

3.4 Traffic Management

Data Streaming Traffic Management (TM) supports end-to-end flow control through multiple mechanisms. The protocol includes On/Off, Rate based and Credit based schemes. There is also room in the protocol for user defined operations. Traffic Management uses the extended packet format for Type 9 (see Chapter 4 for packet definitions). The traffic management format makes up a message that contains:

<VSID><Wildcard+Mask> <Message XType><Parameter 1><Parameter 2>

The VSID (from the standard type 9 header) plus the wildcard and mask fields in the extended packet header forms the operand (the queue or queues) for the traffic management message. The message type and type specific parameters form the operation.

3.4.1 Traffic Management Operand

All Data Streaming traffic management protocols use a common mechanism for queue designation. The message overloads the data streaming VSID fields to define the stream to which to apply the message. In addition, the extended header format includes modifiers, in the form of a wild card and a mask to expand the scope of the message beyond a single stream. The message can apply to:

- A single stream, identified by <Dest><cos><streamID>
- A group of streams in a class identified by <Dest><cos>
- A group of classes, identified by <Dest><cos><mask>
- All classes for a given port, identified by <Dest>

Note that the operand is always based on a VSID that is the target (intended destination) of the data. The egress is not responsible for knowing the mapping of queues taking place at the ingress.

Example: Endpoint 21 sends <cos 3> XOFF to endpoint 6
Endpoint 6 would stop all streams to VSID: DestID 21, class 3.

3.4.2 On/Off Traffic Management

The messages supported are:

Egress to Ingress:
<Q> XOFF (where <Q> is any operand described in 3.4.1)
<Q> XON

Ingress to Egress:
<Q> Q_STATUS <Level> (see Section 4.3.2 for message and parameter formats)

Basic On/Off traffic management consists of egress to ingress messages that direct the operand <Q> be stopped (XOFF) or started (XON). The ingress may signal the need for service with the Q_STATUS message, with *level* indicating the relative fullness of the queue.

Ingress devices may admit traffic based on any ingress specific scheduling algorithm. This message does not modify the algorithm except to suspend/resume traffic flow.

When traffic management is enabled and set to *TM Type 0* (basic), the messages shall be supported by the egress and honored by the ingress according to the rules in Section 3.2.5.

3.4.3 Rate Base Traffic Management

The messages supported are:

From Egress to Ingress

<Q> XON (where <Q> is any operand described in Section 3.4.1)
<Q> XOFF
<Q> INCREASE <Amount>
<Q> DECREASE <Amount>

From Ingress to Egress
<Q> Q_STATUS <Level>   (see Section 4.3.3 for message and parameter formats)

XON and XOFF messages are as defined in Section 3.4.2.

Rate based flow control is a *relative* control protocol. <Amount> is a ratio relative to the current rate of traffic flow (see Chapter 4 for field definitions). The ingress is pre-configured with a specific traffic scheduling algorithm. The egress uses the INCREASE / DECREASE mechanism to modify the ingress' scheduling process, usually based on the egress' ability to move the traffic to its next destination.

DECREASE<0> is a special message meaning MAINTAIN current rate, and INCREASE<max> is a special message to DOUBLE the current rate.

The ingress may use the Q_STATUS message to indicate the status of its queues, with *level* indicating the relative fullness of the queue, allowing a closed loop decision process. For example, if the ingress sends successive messages indicating rising queue levels, the egress may choose to increase the rate at which the ingress has permission to admit traffic from that queue.

When the traffic management mode is set to *TM Type 1* (rate), the XON, XOFF messages are included to carry forward the basic operation (the type 0 messages no longer need to be used).

3.4.4 Credit Based Traffic Management

Credit based traffic management permits the egress to control traffic on a PDU by PDU basis. In this mode PDUs are only transmitted when an ingress is given an allocation of credits for a specific queue (identified by the operand <Q>). Traffic flow stops when the allocation of credits reaches zero.

The basic message formats are:

Egress to Ingress:

<Q> XON (where <Q> is any operand described in Section 3.4.1)
<Q> XOFF
<Q> ALLOCATE <AU> <Credits>

Ingress to Egress:
<Q> CREDIT STATUS <AU> <Credits>
<Q> Q_STATUS <Level>   (see Section 4.3.4 for message and parameter formats)

Allocate is used by an egress endpoint to tell an ingress endpoint that it has <some number of> credits available for use. The credits are assigned to an *allocation unit*

<AU>. The allocation unit allows blocks of resources to be grouped, permitting coarser management, and requiring less precision in the synchronization between an ingress and the egress.

When the credits in an allocation unit are consumed, the ingress begins to use credits in the next allocation unit. The egress assigns credits in allocation units on a rotating basis. Allocation units have local scope. The allocation unit value has local scope to the <Q> designation.

The protocol allows or pipelining up to 16 allocation units. Endpoints do not have to use 16 allocation units. The number used is a function of how far in advance credits need to be issued. The minimum implementation is 2 to ping-pong buffer groups and keep some credit available at the ingress. The egress may issue any number of allocation units (starting with 0) and rolling over at whatever limit it supports. It is up to the egress to be sure an AU is unused before reusing that number.

<Q> ALLOCATE <AU> <0> is used to retire an allocation. It may be used by an egress endpoint to free a block of buffering for a number of reasons. It can be used to pause a transmission by forcing the only credits to zero. It can be used to clean up memory allocation by forcing an ingress onto the next allocation unit.

Credit Status: is sent by the ingress to delimit the use of allocation units, and to indicate status to trigger new allocations. <Q> CREDIT STATUS <AU> <nn>, where nn is the number of initial credits for an AU, is sent ahead of the first packet to delimit the beginning of the use of that AU.

<Q> CREDIT STATUS <AU> <00> is always sent after the last packet resulting in the number of credits going to 0 for that AU. The egress can close out that chunk of buffering, even if its notion of the number of PDUs received does not agree with the ingress (a PDU has been lost somewhere). <Q> CREDIT STATUS <AU> <00> is also sent in response to an <Q> ALLOCATE <AU> <00> acknowledging that the ingress will send no more PDUs for that allocation, delimiting any PDUs in the pipeline.

<Q> CREDIT STATUS <AU> <xx>, where xx is some number of remaining credits, may be sent by an ingress to indicate a low level of credit allocation as a trigger to request more credits. The egress may also track the number of incoming PDUs from an ingress keeping a local credit balance. Either or both mechanisms can be used to sequence allocations. Note that the egress is not required to keep a specific credit balance, it can allocate and retire allocation units using the delimiting messages.

Queue Status: provides a means to indicate the overall status of a specific queue. The source can use this message to get attention for a queue that has become active, needing credits (transitioning from empty). It can be used to indicate a queue that has gone empty, allowing the destination to deallocate the remaining credits and retire the AU. It can also be used to indicate that the current rate of credits being issued is not keeping up with incoming traffic.

XOFF and **XON** are used to pause / resume transmission without changing the state of credit allocation.

In the example shown in Figure 3-6, the ingress initiates activity with queue status, indicating traffic available. The egress responds with an allocation of 16 credits.

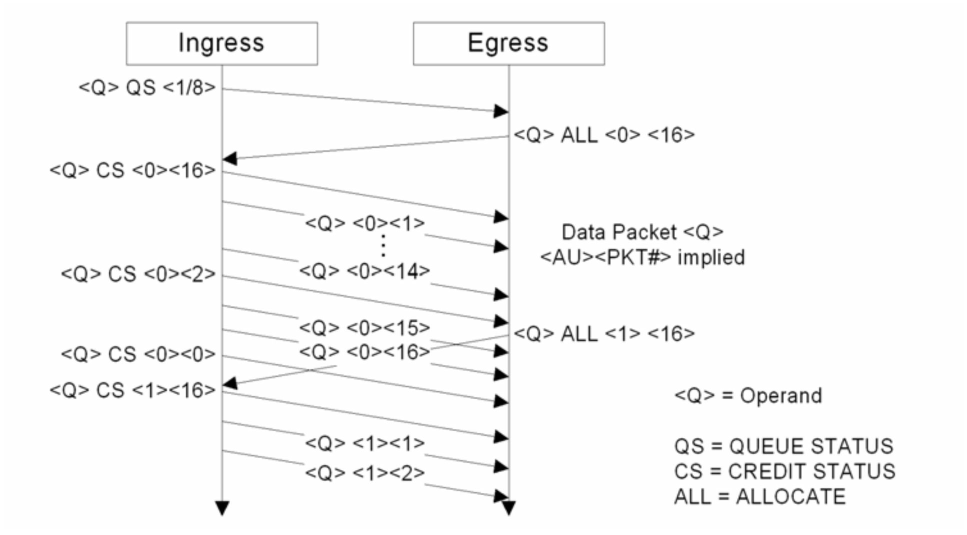


Figure 3-6. Typical Credit Based Flow Control Example

At a level of 2 credits remaining, the ingress sends a credit status of <2> indicating a low level of credits. It proceeds with sending the last two PDUs.

The egress responds to the request for more credits with an allocation of 16 additional PDUs. The ingress indicates the last PDU has been sent on the first allocation, and that new PDUs are being sent on the new allocation with the two sequential credit status messages. This is a “typical” scenario.

In the scenario shown in Figure 3-7, the egress moves the traffic from AU 0 to AU 1 by first allocating more credits with AU 1, then terminating the credits for AU 0.

The ingress responds by sending credit status of 0 for AU 0, and credit status to indicate the beginning of AU 1.

There are many ways to use this protocol. An ingress may use the credit status <0> to asynchronously relinquish credits for a stream if an activity timer indicates that there’s been no new traffic for a period of time (or the egress may do the same).

The size of the allocations can be varied using the Queue Status message, allocating larger blocks for fuller queues. An ingress endpoint might have prescribed thresholds for sending queue status message.

The ingress might also have adjustable thresholds for sending Credit Status messages to adjust for pipeline delays, or algorithmically move that threshold up or down based on gaps in allocation.

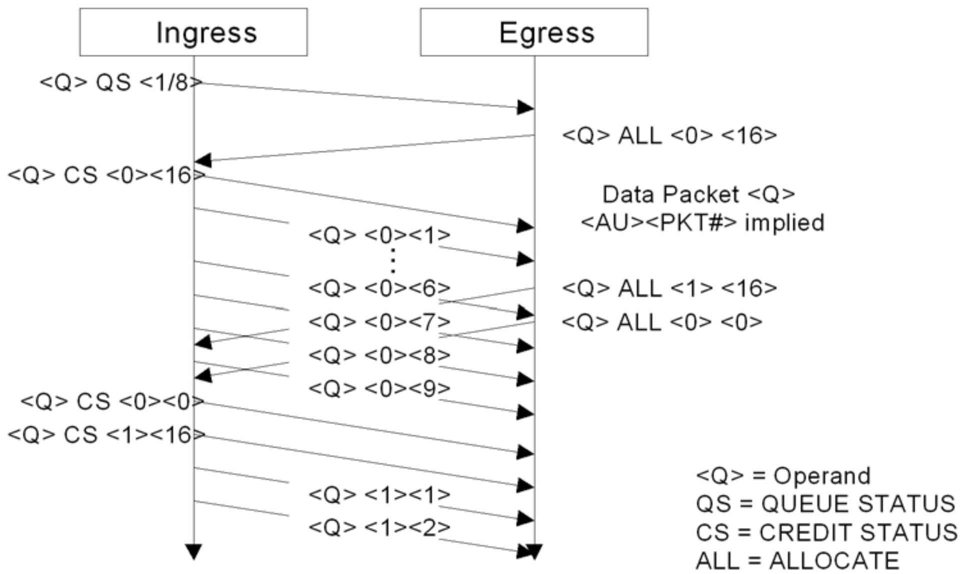


Figure 3-7. Credit Based Protocol Example

3.4.5 Rules for Traffic Management

Supported Functionality

1. Supporting traffic management is entirely optional.
2. Any endpoint advertising support for traffic management shall support the basic mode of operation.
3. An endpoint may or may not support rate or credit based modes of operation.
4. Any supported modes shall be supported fully, incorporating all defined message formats (use of user defined fields is optional).

Therefor the valid combinations for supporting traffic management are:

- Basic Only
- Basic + Rate
- Basic + Credit
- Basic + Rate + Credit

Protocol Rules

1. All TM transactions use the VSID in the type 9 header, so by definition, all TM transactions occur in the same flows as the data. When wild cards are used, the don't care fields shall be set to values that put the message in the same flow as the affected data.

2. Any rules used by the ingress to associate traffic with a specific VSID shall be used in reverse to associate the VSID in the message from the egress with the at least one queue. Beyond that, any internal hierarchies of queues and relationships to different messages are up to the implementation.
3. Endpoints may implement <Amount>, or <Level> with less precision than described. Receiving endpoints shall support the full range of values by rounding to the desired precision.
4. An ingress shall not overrule a TM directive from the egress. The ingress may discard traffic should the egress not adequately permit that traffic onto the fabric. Any discard algorithm is implementation specific.

Error Handling

5. There are no requirements for timers. An ingress may “insist” by sending repeated Q_STATUS messages.
6. Lost messages are recovered by sending duplicates. Endpoints shall recognize duplicates as such and not behave inappropriately.

   An XOFF to a <Q> already in the off state is ignored.

   An XON to a <Q> already in the on state is ignored.

   A lost Rate DECREASE message results in the egress sending a second DECREASE with a larger requested decrease amount.

   A lost Rate INCREASE message results in the ingress issuing more urgent Q_STATUS messages and an the egress issuing additional INCREASE messages.
7. The exception to rule 6 is the sequencing of allocation units with the CREDIT STATUS message:

   If the <Q> CREDIT STATUS <AU> <0> message is lost, the allocation unit will be assumed closed when the <Q> CREDIT STATUS <AU+1> <N> is received opening operation on the next allocation unit.

   If the <Q> CREDIT STATUS <AU+1> <N> message is lost, the next allocation unit is assumed to be opened when the next PDU for that stream is received.

   Also, as an exception to rule 6, ALLOCATE messages are never duplicated. If an allocation unit message is lost, the egress may recover it with a <Q> ALLOCATE <AU><0> message, insuring it is de-allocated before reusing it.

Chapter 4 Packet Format Descriptions

4.1 Introduction

This chapter contains the definition of the data streaming packet format.

4.2 Type 9 Packet Format (Data-Streaming Class)

The type 9 packet format is the DATA STREAMING transaction format. Type 9 packets always have a data payload, unless terminating the PDU. Unlike other RapidIO logical specifications, the data payload length is defined as a multiple of half-words rather than double-words. A pad bit allows a sender to transmit an odd number of bytes in a packet. An odd bit indicates that the data payload has an odd number of half-words. This bit makes it possible for the destination to determine the end of a data payload if packet padding is done by the underlying transport. An extended header bit allows future expansion of the functionality of the type 9 packet format.

Definitions and encodings of fields specific to type 9 packets are provided in Table 4-1.

Table 4-1. Specific Field Definitions and Encodings for Type 9 Packets

Field	Definition
cos	class of service - This field defines the class of service to be applied by the destination end point (and possibly intervening switch processing elements) to the specified traffic stream.
S	Start - If set, this packet is the first segment of a new PDU that is being transmitted. The new PDU is identified by the combination of the source of the packet and the flowID.
E	End - If set, this packet is the last segment of a PDU that is being transmitted. Both S and E set indicates that the PDU is fully contained in a single packet.
rsrv	Reserved - Assigned to logic 0s by the sender, ignored by the receiver
xh	Extended header - There is an extended header on this packet. The extended header is used for traffic management.
O	Odd - If set, the data payload has an odd number of half-words
P	Pad - If set, a pad byte was used to pad to a half-word boundary

Table 4-1. Specific Field Definitions and Encodings for Type 9 Packets (Continued)

Field	Definition
streamID	traffic stream identifier - This is an end to end (producer to consumer) traffic stream identifier.
length	PDU length - This is the length in bytes of the segmented PDU. 0x0000 - 64kbytes 0x0001 - 1 byte 0x0002 - 2 bytes 0x0003 - 3 bytes ... 0xFFFF - 64kbytes - 1

Table 4-1 details the O and P bit combinations.

Table 4-2. Specific Field Definitions and Encodings for Type 9 Packets

O bit	P bit	Definition
0b0	0b0	Even number of half-words and no pad byte
0b0	0b1	Even number of half-words and a pad byte
0b1	0b0	Odd number of half-words and no pad byte
0b1	0b1	Odd number of half-words and a pad byte

There are three type 9 packet headers, determined by the value of the Start and End bits, which determine if the header is a Start/Single header, a Continuation header, or an End header. The following set of figures shows examples of type 9 packets. Field sizes are specified in bits.

Figure 4-1 is an example of a Single Segment type 9 packet with all of its fields. The data payload size may or may not match the MTU size, so *n* and *m* are determined by the size of the PDU itself. In this example, the data payload is un-padded and there are an even number of half-words. The value 0b1001 in Figure 4-1 specifies that the packet format is of type 9. This is the only type 9 packet that has the xh field.

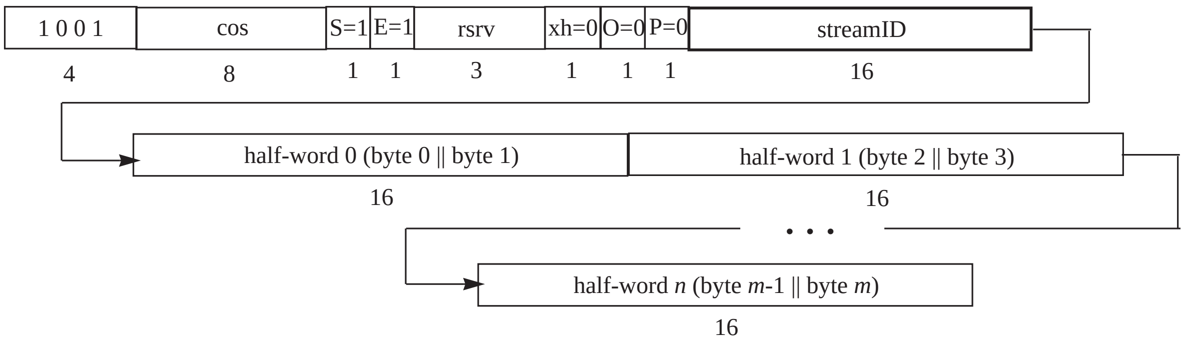

1 0 0 1
4
cos
8
S=1
1
E=1
1
rsrv
3
xh=0
1
O=0
1
P=0
1
streamID
16
half-word 0 (byte 0 || byte 1)
16
half-word 1 (byte 2 || byte 3)
16
. . .
half-word *n* (byte *m*-1 || byte *m*)
16

Figure 4-1. Single Segment Type 9 Packet Bit Stream Format Example

Figure 4-2 is an example of a Start Segment type 9 packet with all of its fields. The data payload that matches the MTU, so *n* and *m* are determined by the MTU size. The value 0b1001 in Figure 4-2 specifies that the packet format is of type 9.

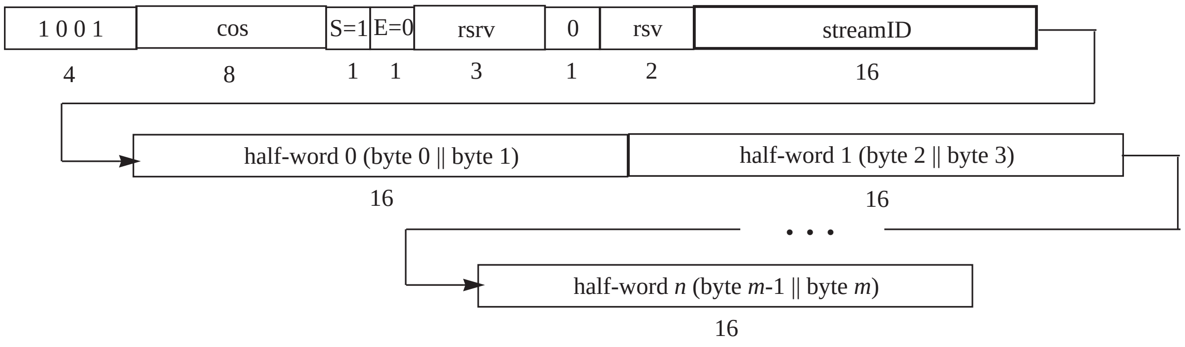

1 0 0 1
4
cos
8
S=1
1
E=0
1
rsrv
3
0
1
rsv
2
streamID
16
half-word 0 (byte 0 || byte 1)
16
half-word 1 (byte 2 || byte 3)
16
. . .
half-word n (byte m-1 || byte m)
16

Figure 4-2. Start Segment Type 9 Packet Bit Stream Format Example

Figure 4-3 is an example of a Continuation Segment type 9 packet with all of its fields. The size of the data payload must match the MTU size. The half-words (and correspondingly, bytes) are contiguous in the manner shown in the preceding examples. The value 0b1001 in Figure 4-3 specifies that the packet format is of type 9.

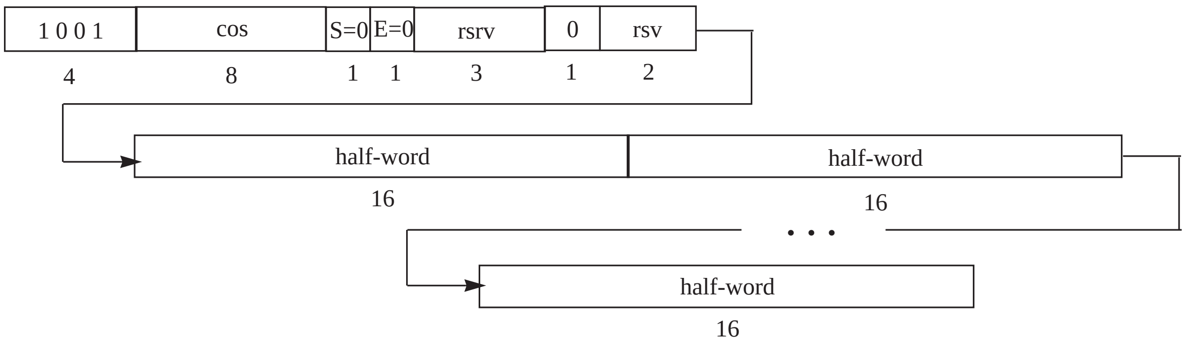


Figure 4-3. Continuation Segment Type 9 Packet Bit Stream Format Example

Figure 4-4 is an example of an End Segment type 9 packet with all of its fields. The size of the data payload is determined by the remainder of the size of the PDU (the length field) divided by the size of the MTU. For convenience at the destination, the O and P bits are used as they are for a single segment. In this example, the data payload size does not match the PDU size, has a pad byte, and is an odd number of half-words. The half-words (and correspondingly, bytes) are contiguous in the manner shown in the preceding examples. A length value of 0 and no data payload can be used to force the PDU to be discarded. The value 0b1001 in Figure 4-4 specifies that the packet format is of type 9.

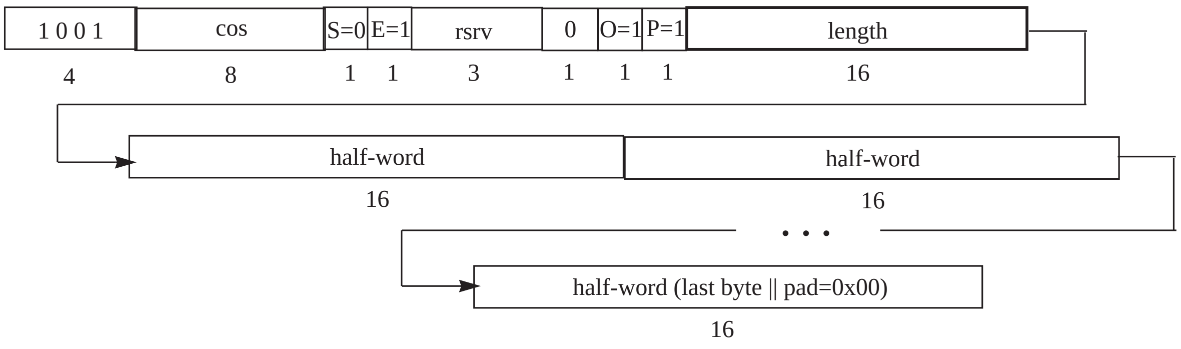


Figure 4-4. End Segment Type 9 Packet Bit Stream Format

4.3 Type 9 Extended Packet Format

The type 9 extended packet format for traffic management between two data streaming endpoints is shown below.

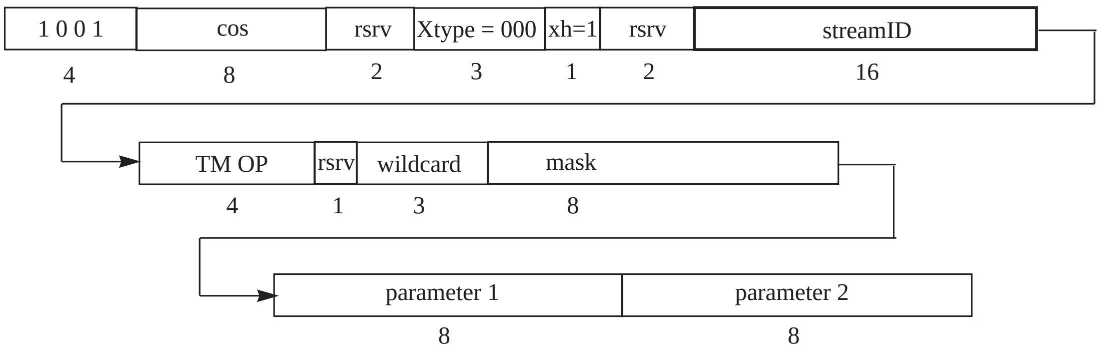


Figure 4-5. Traffic Management Bit Stream Format

The extended type 9 packet is identified first by the XH bit equal to 1. The Class of Service field and the StreamID are included from the data packet format. The segmentation bits, Start and End are not used, or are the Odd and Pad fields. Those are left 0s. The 3 bit reserved field, is defined for the extended packet format as defining the type of extended packet. Type 0 is a traffic management packet.

The TM packet adds a fixed 4 bytes to the packet as shown above and defined below:

Table 4-3. Extended Header Fields

Type 9 Fields	Encoding	Definition
cos	—	Class Of Service: This field defines the class of service assigned to the stream that is being managed with this message. See Section 4.3.1.
rsrv	—	Reserved
xtype	0b000	Traffic Management Packet
	0b001 - 0b111	Reserved
xh	0b1	Extended Header: There is an extended header on this packet. Currently the extended header is only used for stream management messages. It is always assigned to 0b1 for type 9 extended packets.
streamID	—	Traffic Stream Identifier: This is an end to end (producer to consumer) traffic stream identifier that is being managed with this message. See Section 4.3.1.
TM OP	0b0000	Basic Stream Management Message: message specifying base level XON/XOFF functionality. This message flows from one endpoint to another. See Section 4.3.2.
	0b0001	Rate Control Stream Management Message: Rate flow control messages to support the rate control protocol. See Section 4.3.3.
	0b0010	Credit Control Stream Management Message: Credit Control messages to support the credit based flow control protocol. Section 4.3.4
	0b0011	Application Defined Stream Management Message: A message from one end point to another end point. The use of this message is application defined.
	0b0100-1111	Reserved
wildcard	0bnxx	VSID dest wildcard: If this bit is set, the message applies to all destinations. If clear, the message applies to the specified destination. See Section 4.3.1.
	0bxnx	VSID class wildcard: If this bit is set the message applies to all classes. If clear, the message applies to the specified class or classes as specified by the mask bits. See Section 4.3.1.
	0bxxn	VSID stream wildcard: If this bit is set the message applies to all streams. If clear, the message applies to the specified stream. See Section 4.3.1.
mask	—	Class Mask: Used to mask portions of the class of service (COS) field to allow groups of classes to be included in the message. The mask is left justified to identify specific class bits as don't cares: Mask Class 0b00000000 0bnnnnnnnn 256 classes n = valid class bits 0b00000001 0bnnnnnnnx 128 classes x = don't cares 0b00000011 0bnnnnnnxx 64 classes 0b00000111 0bnnnnnxxx 32 classes 0b00001111 0bnnnnxxxx 16 classes 0b00011111 0bnnnxxxxx 8 classes 0b00111111 0bnnxxxxxx 4 classes 0b01111111 0bnxxxxxxx 2 classes 0b11111111 0bxxxxxxxx 1 class
Parameter1	—	Parameter1: Argument specific to the TM message operation. See below
Parameter2	—	Parameter2: Argument specific to the TM message operation. See below

4.3.1 TM Operand

The operand for the specific TM operation is defined by the following fields:

<Source or Destination ID> <CoS> <StreamID> + <wild cards> + <Mask>

The follow operands are valid:

Specific Stream:
    <DestID><CoS><StreamID> + <wc=000> + <m = 0x00>

All Streams in a specific class:
    <DestID><CoS>< xx > + <wc = 001> + <m = 0x00>
    where StreamID is a don’t care

All Streams in a group of classes:
    <DestID><CoS>< xx > + <wc = 001> + <m = 0xnn>
    where mask is one of the non-zero values identified in table 4-3

All Streams and Classes (all traffic) to a specific destination
    <DestID>< xx >< xx > + <wc = 011> + <m = xx>
    where CoS, StreamID and mask are don’t cares

All traffic from this source
    < xx >< xx >< xx > + <wc = 111> + <m = xx>

No other combinations of these fields is permitted.

4.3.2 Basic Traffic Management

Basic traffic management message formats are as follows:

Table 4-4. Basic Traffic Management Message Formats

TM OP	Parameter 1	Parameter 2	Definition
0b0000	0b0000 0000	0b0000 0000	XOFF: Transmit off
		0b1111 1111	XON: Transmit on.
		0b0000 0001-0b1111 1110	User/application defined
	0b0000 0011	0b0000 0000 - 0b1111 1111	Q_Status: Source queue is n/255 full (where n is parameter 2). 0 = empty, 0xFF = full

All other parameter 1 values are reserved.

The implementer may overload the parameter 2 field (Parameter1 = 0) for other messages. Only 0x00 shall be assured to be XOFF, and 0xFF shall assured to be XON.

4.3.3 Rate Based Traffic Management

Rate based traffic management messages are as follows:

Table 4-5. Rate Based Traffic Management Message Formats

TM OP	Parameter 1	Parameter 2	Definition
0b0001	0b0000 0000	0b0000 0000	XOFF: Transmit off
		0b1111 1111	XON: Transmit On. Start transmitting at the rate prior to receiving the XOFF
		0b0000 0001 - 0b1111 1110	User Defined: May be overloaded with any implementation specific message.
	0b0000 0Y01	0b0000 0000	Maintain_Rate: Maintain the current rate
		0b0000 0001	REDUCE: Reduce the current rate to = CurrentRate x (1-1/256)
		0b0000 0010	REDUCE: Reduce the current rate to = CurrentRate x (1-2/256)
		0b0000 0011- 0b1111 1111	REDUCE: Reduce the current rate to = CurrentRate x (1-n/256) (where n is parameter 2)
	0b0000 0Y10	0b0000 0001- 0b1111 1110	INCREASE: Increase the current rate to = CurrentRate x (1+ n/256) (additive increase where n is parameter 2)
		0b1111 1111	DOUBLE: Double the current rate.
	0b0000 0011	0b0000 0001- 0b1111 1111	Q_Status: Source queue is n/255 full (where n is parameter 2). 0x00 = Empty, 0xFF = Full

For ’Y’ = 0 (Param 1 = 0x01, 0x02) the messages apply to average rate.

For ’Y’ = 1 (Param 1 = 0x05, 0x06) the messages apply to peak rate.

All other parameter 1 values are reserved.

The implementer may overload the parameter 2 field (Parameter1=0) for other messages. Only 0x00 shall be assured to be XOFF, and 0xFF shall assured to be XON.

Notes on reducing precision:

REDUCE values should be rounded UP to the nearest precision.
Example: for n/16 precision - 0b0000 0011 would be rounded to 0b0001 xxxx

INCREASE values should be rounded DOWN to the nearest precision.
Example for n/32 precision 0b1100 1110 would be rounded to 0b1100 1xxx

Q_STATUS values should be rounded UP to the nearest precision.

Whatever the minimum precision the ingress uses for rate scheduling, a decrease must never reduce the rate completely to zero. The egress must use XOFF to stop the flow completely.

4.3.4 Credit Based Traffic Management

The message formats for credit based traffic management are as follows:

Table 4-6. Credit Based Traffic Management Message Formats

TM OP	Parameter 1	Parameter 2	Description
0b0010	0b00000000	0b00000000	XOFF: Transmit off
		0b11111111	XON: Transmit On. Start transmitting at the rate prior to receiving the XOFF
		0b0000 0001 - 0b1111 1110	User Defined: May be overloaded with any implementation specific message.
	0b0001 nnnn	0b0000 0000 - 0b1111 1111	Allocate: nnnn - Allocation Unit Parameter 2 - number of credits
	0b0010 nnnn	0b0000 0000 - 0b1111 1111	Credit Status: nnnn - allocation Unit Parameter 2 - number of Credits
	0b0011 0000	0b0000 0000 - 0b1111 1111	Queue Status: Source Queue is n/255 full (where n is parameter 2) 0 = Empty; 0xFF = Full

All other parameter 1 values are reserved.

The implementer may overload the parameter 2 field (Param1=0) for other messages. Only 0x00 shall be assured to be XOFF, and 0xFF shall assured to be XON.

Chapter 5 Data Streaming Registers

5.1 Introduction

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, physical, and extension specifications of interest to determine a complete list of registers and bit definitions. All registers are 32 bits and aligned to a 32 bit boundary.

5.2 Register Summary

Table 5-1 shows the register map for this RapidIO specification. These capability registers (CARs) and command and status registers (CSRs) can be accessed using *RapidIO Interconnect Specification Part 1: Input/Output Logical Specification* maintenance operations. Any register offsets not defined are considered reserved for this specification unless otherwise stated. Other registers required for a processing element are defined in other applicable RapidIO specifications and by the requirements of the specific device and are beyond the scope of this specification. Read and write accesses to reserved register offsets shall terminate normally and not cause an error condition in the target device. Writes to CAR (read-only) space shall terminate normally and not cause an error condition in the target device.

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

Table 5-1. Data Streaming Register Map

Configuration Space Byte Offset	Register Name
0x0-14	Reserved
0x18	Source Operations CAR
0x1C	Destination Operations CAR
0x20–38	Reserved
0x3C	Data Streaming Information CAR

Table 5-1. Data Streaming Register Map (Continued)

Configuration Space Byte Offset	Register Name
0x40–44	Reserved
0x48	Data Streaming Logical Layer Control CSR
0x4C–FC	Reserved
0x100–FFFC	Extended Features Space
0x10000–FFFFFC	Implementation-defined Space

5.3 Reserved Register, Bit and Bit Field Value Behavior

Table 5-2 describes the required behavior for accesses to reserved register bits and reserved registers for the RapidIO register space,

Table 5-2. Configuration Space Reserved Access Behavior

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

Table 5-2. 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.

When a writable bit field is set to a reserved value, device behavior is implementation specific.

5.4 Additions to Existing Registers

The following bits are added to the Logical/Transport Layer Error Detect CSR (see *RapidIO Interconnect Specification Part 8: Error Management Extensions Specification*).

Table 5-3. Bit Settings for Logical/Transport Layer Error Detect CSR

Bit	Name	Reset Value	Description
10	Missing data streaming context	0b0	A continuation or end data streaming segment was received for a closed or non-existent segmentation context (end point device only)
11	Open existing data streaming context	0b0	A start or single data streaming segment was received for an already open segmentation context (end point device only)
12	Long data streaming segment	0b0	Received a data streaming segment with a payload size greater than the MTU (end point device only)
13	Short data streaming segment	0b0	Received a start or continuation data streaming segment with a payload size less than the MTU (end point device only)
14	Data streaming PDU length error	0b0	The length of a reassembled PDU differs from the PDU length carried in the end data streaming segment packet header (end point device only)

The following bits are added to the Logical/Transport Layer Error Enable CSR (see *RapidIO Interconnect Specification Part 8: Error Management Extensions Specification*).

Table 5-4. Bit Settings for Logical/Transport Layer Error Enable CSR

Bit	Name	Reset Value	Description
10	Missing data streaming context error enable	0b0	Enable reporting of a continuation or end data streaming segment received for a closed or non-existent segmentation context. Save and lock capture information in the appropriate Logical/Transport Layer Capture CSRs. (end point device only)
11	Open existing data streaming context error enable	0b0	Enable reporting of a start or single data streaming segment received for an already open segmentation context. Save and lock capture information in the appropriate Logical/Transport Layer Capture CSRs. (end point device only)
12	Long data streaming segment error enable	0b0	Enable reporting of a any data streaming segment received with a payload size greater then the MTU. Save and lock capture information in the appropriate Logical/Transport Layer Capture CSRs. (end point device only)

Table 5-4. Bit Settings for Logical/Transport Layer Error Enable CSR

Bit	Name	Reset Value	Description
13	Short data streaming segment error enable	0b0	Enable reporting of a start or continuation data streaming segment received with a payload size less that the MTU. Save and lock capture information in the appropriate Logical/Transport Layer Capture CSRs. (end point device only)
14	Data streaming PDU length error enable	0b0	Enable reporting of a reassembled PDU length that differs from the PDU length carried in the end data streaming segment packet header. Save and lock capture information in the appropriate Logical/Transport Layer Capture CSRs. (end point device only)

5.5 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 5-2 for the required behavior for accesses to reserved registers and register bits.

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

5.5.1 Source Operations CAR (Configuration Space Offset 0x18)

This register defines the set of RapidIO data streaming logical operations that can be issued by this processing element; see Table 5-5. It is assumed that a processing element can generate I/O logical maintenance read and write requests if it is required to access CARs and CSRs in other processing elements. The Source Operations CAR is applicable for end point devices only. RapidIO switches shall be able to route any packet.

Table 5-5. Bit Settings for Source Operations CAR

Bit	Field Name	Description
0–11	—	Reserved
12	Data streaming traffic management	PE can support data streaming traffic management
13	Data streaming	PE can support a data streaming operation
14-15	Implementation defined	Defined by the device implementation
16-29	—	Reserved
30–31	Implementation defined	Defined by the device implementation

5.5.2 Destination Operations CAR (Configuration Space Offset 0x1C)

This register defines the set of RapidIO data streaming operations that can be supported by this processing element; see Table 5-6. 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 5-6. Bit Settings for Destination Operations CAR

Bit	Field Name	Description
0-11	—	Reserved
12	Data streaming traffic management	PE can support data streaming traffic management
13	Data streaming	PE can support a data streaming operation
14-15	Implementation defined	Defined by the device implementation
16-29	—	Reserved
30-31	Implementation defined	Defined by the device implementation

5.5.3 Data Streaming Information CAR (Configuration Space Offset 0x3C)

This register defines the data streaming capabilities of a processing element. It is required for destination end point devices.

Table 5-7. Bit Settings for Data Streaming Information CAR

Bit	Field Name	Description
0–15	MaxPDU	Maximum PDU - The maximum PDU size in bytes supported by the destination end point 0x0000 - 64kbytes 0x0001 - 1 byte 0x0002 - 2 bytes ... 0xFFFF - 64kbytes - 1
16–31	SegSupport	Segmentation Support - The number of segmentation contexts supported by the destination end point 0x0000 - 64k segmentation contexts 0x0001 - 1 segmentation context 0x0002 - 2 segmentation contexts ... 0xFFFF - 64k - 1 segmentation contexts

5.6 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 5-2 for the required behavior for accesses to reserved registers and register bits.

5.6.1 Data Streaming Logical Layer Control CSR (Configuration Space Offset 0x48)

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

Table 5-8. Bit Settings for Data Streaming Logical Layer Control CSR

Bit	Field Name	Description
0-3	TM Types Supported (read only)	Bit 0 = 1, Basic Type Supported Bit 1 = 1, Rate Type Supported Bit 2 = 1, Credit Type Supported Bit 3 = Reserved Valid Combinations: 0b1000, 0b1100, 0b1010, 0b1110. All others invalid
4 - 7	TM Mode	Traffic Management Mode of operation 0b0000 = TM Disabled 0b0001 = Basic 0b0010 = Rate 0b0011 = Credit 0b0100 = Credit + Rate 0b0101 - 0b0111 = Reserved 0b1000 - 0b1111 = allowed for user defined modes
8 - 23		Reserved
24-31	MTU	Maximum Transmission Unit - controls the data payload size for segments of an encapsulated PDU. Only single segment PDUs and end segments are permitted to have a data payload that is less than this value. The MTU can be specified in increments of 4 bytes. Support for the entire range is required. 0b0000_0000 - reserved ... 0b0000_0111 - reserved 0b0000_1000 - 32 byte block size 0b0000_1001 - 36 byte block size 0b0000_1010 - 40 byte block size ... 0b0100_0000 - 256 byte block size 0b0100_0001 - Reserved ... 0b1111_1111 - Reserved All other encodings reserved

Blank page

Annex A VSID Usage Examples

A.1 Introduction

The virtual stream identification (VSID) mechanism provides multiple features condensed in a single 32 bit key. These features include:

- A mechanism to manage traffic for ingress to the fabric
- A mechanism to manage traffic in transit within the fabric
- A protocol independent tag to reclassify packets on fabric egress
- A flexible "sub-port" addressing mechanism
- Independence in buffer management

A.2 Background

The VSID is a composite of the port, class, and streamID fields as described in Section 3.2.2. The port address used in the VSID is either the destination ID or the source ID depending on which side of the fabric the packet is on. At the ingress to the fabric (source) the destination IDs are unique. At the egress from the fabric, the source IDs are unique.

By including the source/destination IDs in the VSID, these keys are unique for each source and destination pairing. This allows the other fields (class and streamID) to be set up independently without consideration of how these fields are used with any other port pairings.

The usage of the VSID can vary depending on the sophistication of the fabric and the demands of the application, from very simplistic port or queue steering to conveying significant amounts of information (requiring intensive computation) as to the content of the PDU.

A.3 Packet Classification

All PDUs require some form of classification for ingress to the fabric. Fields in the PDU specific to the protocol are examined and routing information is produced. The VSID produced is a 32 bit tag as opposed to just a port address. At the destination, this 32 bit tag can be used to re-associate the PDU with a target buffer. This can be done by direct addressing, or using a single key table lookup.

This mechanism provides a finely grained and protocol independent way to sort traffic, and a virtual mechanism for buffer pool management. Without a virtual tag, the packet would have to undergo a re-classification based on the protocol specific portion of the PDU. In multi-service platforms, this could involve numerous and elaborate processes, duplicating what was already done at the source.

The following sections illustrate in degrees of increasing complexity, the versatility of the VSID scheme.

A.3.1 Sub-port Addressing at the Destination

The simplest use of the VSID is to de-multiplex the traffic into coarse sub-ports at the destination. These may be to separate traffic by protocol, or into multiple sub-ports of the same protocol.

A.3.1.1 DSLAM application

Assume that each line card contain 128 user ports. The system could expose each of these as independent destinations to the RapidIO fabric, requiring the use of an excessively large number of destination IDs in the system, and imposing the associated cost in overhead. Alternatively the ATM traffic can be encapsulated into 128 VSIDs, one for each port. The line card would then expose a single port to the RapidIO fabric. The VSID would be used as the address to fan out the traffic on various UTOPIA busses to the user ports. This also has an advantage for fault recovery. Should a line card fail, a single port entry in routing tables in the fabric needs to be updated rather than all 128 sub-ports.

A.3.1.2 VOIP application

The VSID can be used to separate the traffic into just 2 channels, one destined for a control processor to handle control messages and one channel that goes to a network processor to be distributed to DSPs. The VSID could contain the address of the target DSP, to further off-load the network processor on distribution. The VSID could also contain the user channel within the DSP de-multiplexing the traffic even further.

A.3.2 Virtual Output Queueing - Fabric On-ramp

Applications involving larger numbers of flows can use the class field to regulate the ingress to the fabric (known as virtual output queueing). For example, the RapidIO fabric interface could contain 256 queues for 64 destination ports with 4 traffic classes. Traffic for each destination of the same class is fairly weighted. The weighting between classes can be application unique.

The traffic is kept sorted by destination. If traffic was just dumped into 4 queues, and a destination port was to fail, the traffic could head of line block the traffic to the other ports, or it would have to be discarded while the port is being recovered or re-routed. By keeping the traffic sorted by destination at the fabric ingress, that destination can be re-routed with minimal traffic loss.

Virtual output queueing can be expanded to 2K or even 16K buffers depending on how large the fabric is, and how many different traffic classes are involved. This fabric ingress management can be a simple mechanism to add some quality of service to a system using the destination ID and the class portion of the VSID. Note that this can be done separately from the use of the streamID at the destination for de-multiplexing.

A.4 System Requirements

The use of the VSID is determined by all three elements in a system, the source, the fabric, and the destination. This section contains descriptions of some example source devices.

A.4.1 UTOPIA to RapidIO ATM bridge

The UTOPIA to RapidIO ATM bridge classifies traffic using the VPI field as the destination port, and the VCI as a sub-port address. It maps all (type 9) traffic to a single RapidIO flow, setting the class to 0 and the streamID to the VCI. The fabric switches on flows. The destination uses the streamID portion of the VSID as a hard-wired sub-port address.

A.4.2 Network processor

The network processor (NP) contains a OC-48 link aggregating traffic to and from multiple 1MB/s ports distributed on line cards. The NP classifies traffic for each user into two classes: high priority for voice (using RTP) and low priority for all others. It sets the class field to 0 or 1, the port to the proper line card, and the streamID to the desired sub-port.

A.4.3 CSIX to RapidIO interface

The CSIX packet contains the destination and class fields (the source is a preset parameter in the interface chip). The streamID is the first 16 bits of the CSIX payload. The RapidIO packet is easily constructed from this information. The fabric interface contains multiple virtual output queues, 2 per destination port. Since the CSIX to NP interface is also a segmented interface, PDUs are reassembled in the virtual queues until enough information is available to form the required MTU on the RapidIO fabric.

The fabric maps the class to a higher or lower priority flow. The destination uses the streamID to map the traffic to the correct user sub-port. Each sub-port contains two class queues to collect traffic as it is reassembled.

A.4.4 10Gb Metropolitan Area Network interface

A specialized classification processor creates the 32 bit VSID based on IP, TCP/UDP, and application information. The tag is prepended to a SPI4.2 packet. The interface to the fabric is a SPI4.2 to RapidIO bridge, which contains virtual output queues.

The destination is a processor that only supports memory and IO logical transactions. The RapidIO to processor interface bridge contains the segmentation and reassembly buffers and look up tables and associated engines that maps the VSID to a DMA buffer address (and vice-versa).

The system contains multiple of these processing cards to support address translation, encryption, or firewall processing. The source classifies traffic based on which of these applications applies. A connection is created by allocating a buffer address in the destination, and assigning a streamID. The source table is created with the search tree requirements for the protocol, and setting up the VSID result.

Destinations may use the VSID in a hard-wired method, or it may be a flexible mapping to virtual buffers. In either case, the source must be flexible to assign the VSID according the destination's needs. This is normally not an issue as the source needs to classify the packet to determine the destination anyway. The use of the VSID can be to separate the traffic by protocol, sub-port, service class, or into as many virtual queues as necessary. If the destination is managing a large number of buffers, the VSID allows the destination to use a single protocol independent key to re-map the traffic and completely abstract any buffer management.

Glossary of Terms and Abbreviations

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

B

Bridge. A processing element that connects one computer bus to another, allowing a processing element on one bus to access an processing element on the other.

C

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

Class of service (cos) a term used to describe different treatment (quality of service) for different data streams. Support for class of service is provided by a class of service field in the data streaming protocol. The class of service field is used in the virtual stream ID and in identifying a virtual queue.

Command and status registers (CSRs). A set of registers that allows a processing element to control and determine the status of another processing element's internal hardware.

D

Deadlock. A situation in which two processing elements that are sharing resources prevent each other from accessing the resources, resulting in a halt of system operation.

Destination. The termination point of a packet on the RapidIO interconnect, also referred to as a target.

Device. A generic participant on the RapidIO interconnect that sends or receives RapidIO transactions, also called a processing element.

Direct Memory Access (**DMA**). The process of accessing memory in a device by specifying the memory address directly.

Double-word. An eight byte or 64 bit quantity, aligned on eight byte boundaries.

E

Egress - Egress is the device or node where traffic exits the system. The egress node also becomes the destination for traffic out of the RapidIO fabric. The terms egress and destination may or may not be used interchangeably when considering a single end to end connection.

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.

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.

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

Ingress - Ingress is the device or node where traffic enters the system. The ingress node also becomes the source for traffic into the RapidIO fabric. The terms ingress and source may or may not be used interchangeably when considering a single end to end connection.

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

I/O. Input-output.

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.

PDU. Protocol Data Unit, the OSI term for a packet.

Priority. The relative importance of a transaction or packet; in most systems a higher priority transaction or packet will be serviced or transmitted before one of lower priority.

Processing Element (**PE**). A generic participant on the RapidIO interconnect that sends or receives RapidIO transactions, also called a device.

Processor. The logic circuitry that responds to and processes the basic instructions that drive a computer.

R

Receiver. The RapidIO interface input port on a processing element.

S

SAR. Segmentation and Reassembly, a mechanism for encapsulating a PDU within multiple packets.

Segmentation. A process by which a PDU is transferred as a series of smaller *segments*.

Segmentation Context. Information that allows a receiver to associate a particular packet with the correct PDU.

Sender. The RapidIO interface output port on a processing element.

Sequence. Sequentially ordered, uni-directional group of messages that constitute the basic unit of data delivered from one end point to another.

StreamID. A specific field in the data streaming protocol that is combined with the data stream's transaction request flow ID and the sourceID or destinationID from the underlying packet transport fabric to form the virtual stream ID.

Source. The origin of a packet on the RapidIO interconnect, also referred to as an initiator.

Switch. A multiple port processing element that directs a packet received on one of its input ports to one of its output ports.

T

Target. The termination point of a packet on the RapidIO interconnect, also referred to as a destination.

Transaction. A specific request or response packet transmitted between end point devices in a RapidIO system.

Transaction request flow. A sequence of transactions between two processing elements that have a required completion order at the destination processing element. There are no ordering requirements between transaction request flows.

V

Virtual Stream ID (VSID). An identifier comprised of several fields in the protocol to identify individual data streams.

Virtual input Queue (ViQ), Virtual output Queue (VoQ). An intermediate point in the system where one or more virtual streams may be concentrated.

W

Word. A four byte or 32 bit quantity, aligned on four byte boundaries.

RapidIO™ Interconnect Specification Part 11: Multicast Extensions Specification

3.2, 1/2016

RapidIO.org

Revision History

Revision	Description	Date
1.3	First release	06/09/2004
1.3.1	Technical changes: the following new features showings: 04-08-00015.003 Converted to ISO-friendly templates	02/23/2005
2.0	Technical changes: errata showing 06-02-00001.005	06/14/2007
2.1	No technical changes	07/09/2009
2.2	No technical changes	05/05/2011
3.0	Changed RTA contact information. Technical Changes: Addition of standard register block to support Dev32 multicast programming model. Note that if Dev32 is supported, some registers are deprecated.	10/11/2013
3.1	No Technical changes.	09/18/2014
3.2	No technical changes.	01/28/2016

RapidIO.org

Table of Contents

Chapter 1 Overview

1.1 Introduction 11
1.2 Overview 11
1.3 Requirements 12

Chapter 2 Multicast Extensions Behavior

2.1 Introduction 13
2.2 Packet Replication 13
2.3 Multicast Operation 13
2.4 Multicast Transaction Ordering Requirements 17

Chapter 3 Multicast Extensions Registers

3.1 Introduction 19
3.2 Capability Registers (CARs) 20
3.2.1 Processing Elements Features CAR 20
3.2.2 Switch Multicast Support CAR 20
3.2.3 Switch Multicast Information CAR 21
3.3 Command and Status Registers (CSRs) 22
3.3.1 Multicast Mask Port CSR 22
3.3.2 Multicast Associate Select CSR 23
3.3.3 Multicast Associate Operation CSR 24
3.4 Switch Routing Table Register Block 25
3.4.1 Register Map 25
3.4.2 Broadcast Routing Table Control CSR 26
3.4.3 Broadcast Multicast Info CSR 29
3.4.4 Port n Routing Table Control CSR 29
3.4.5 Port n Multicast Info CSR 30
3.4.6 Broadcast Multicast Mask x Set Register y CSR 30
3.4.7 Broadcast Multicast Mask x Clear Register y CSR 31
3.4.8 Port n Multicast Mask x Set Register y CSR 32
3.4.9 Port n Multicast Mask x Clear Register y CSR 32

Chapter 4 Configuration Examples

4.1 Introduction 35
4.2 Configuring Multicast Masks 35
4.2.1 Clearing Multicast Masks 35
4.2.2 Assigning Ports to Multicast Masks 36
4.2.3 Removing a Port from a Multicast Mask 36

Table of Contents

4.2.4 Querying a Multicast Mask.......... 36
4.3 Simple Association 37
4.3.1 Restrictions on Block Size.......... 37
4.3.2 Restrictions on Block Associate 38
4.3.3 Restrictions on Associations.......... 38
4.4 Configuring Associations.......... 38
4.4.1 Basic Association.......... 38
4.4.2 Using Per-Ingress Port Association 39
4.4.3 Using Block Association 40
4.4.4 Using Per-Ingress Port and Block Association.......... 41
4.4.5 Removing a Destination ID to Multicast Mask Association 42
4.4.6 Querying an Association.......... 42

Annex A End Point Considerations (Informative)

A.1 Introduction.......... 47
A.2 Multicast Destination ID.......... 47
A.3 End Point Multicast Channels.......... 47

Annex B Multicast Applications (Informative)

B.1 Introduction.......... 49
B.2 Example 1 - Static Multicast Masks 50
B.3 Example 2 - Linking Multicast Masks to Destination IDs 57

List of Figures

2-1 Multicast System Example 14
2-2 Multicast Association Example 15
2-3 Multicast Configuration Example....................16
4-1 Example System using Multicast....................49

List of Figures

Blank page

List of Tables

3-1 Multicast Register Map....19
3-2 Bit Settings for Processing Elements Features CAR....20
3-3 Bit Settings for Switch Multicast Support CAR....20
3-4 Bit Settings for Switch Multicast Information CAR....21
3-5 Bit Settings for Multicast Mask Port CSR....22
3-6 Bit Settings for Multicast Associate Select CSR....23
3-7 Bit Settings for Multicast Associate Operation CSR....24
3-8 Switch Routing Table Register Map....25
3-9 Bit Settings for Broadcast Routing Table Control CSR....26
3-10 Mask 0-7 Set/Clear Registers, Mask_size = 0....27
3-11 Mask 0-3 Set/Clear Registers, Mask_size = 1....27
3-12 Mask 0-1 Set/Clear Registers, Mask_size = 2....28
3-13 Mask 0 Set/Clear Registers, Mask_size = 3....28
3-14 Bit Settings for Broadcast Multicast Info CSR....29
3-15 Bit Settings for Port n Routing Table Control CSR....29
3-16 Bit Settings for Port n Multicast Info CSR....30
3-17 Bit Settings for Broadcast Multicast Mask x Set Register y CSR....31
3-18 Bit Settings for Broadcast Multicast Mask x Clear Register y CSR....31
3-19 Bit Settings for Port n Multicast Mask x Set Register y CSR....32
3-20 Bit Settings for Port n Multicast Mask x Clear Register y CSR....33
4-1 Multicast Masks for Switch A1....50
4-2 Multicast Masks for Switch B1....51
4-3 Multicast Masks for Switch B2....53

List of Tables

Blank page

Chapter 1 Overview

1.1 Introduction

This chapter provides an overview of the *RapidIO Part 11: Multicast Extensions Specification*. The goal of this specification is to add a simple mechanism to the existing RapidIO specifications that provides multicast functionality to a system. This specification assumes that the reader has a working understanding of the other RapidIO specifications. Implementation of this specification is optional.

1.2 Overview

The concept of duplicating a single message and sending it to multiple selected destinations is known as ‘multicast’, and is found to be useful in many computing systems. This can be accomplished by a variety of means. The most efficient and highest performance method is to have hardware support for the duplication of messages.

Within a RapidIO system, the ability to duplicate messages should scale with the number of end points in a system. Since the number of end points scales with the number of switches in the system, the multicast extensions are defined for switches only and end points are largely unaffected. Possible end point design considerations are described in Annex A.

The multicast specification is limited to request transactions that do not require responses, for example, *RapidIO Part 1: Input/Output Logical Specification* SWRITE transactions. This is because implementing support for collecting the response transactions within a switch device, which are typically not aware of RapidIO logical layer protocols, is problematic and complex.

The ability for a switch to send a single message to a variety of destinations can be implemented in a wide variety of ways, depending on system needs. There are two reasons, however, that motivate definition of a common interface and behavior for multicast in a system. Without a standard interface and behavioral definition, the wide variety of possible implementations would not allow a common multicast software driver to exist. The second reason is that without a standard definition for interface and behavior it is impossible to guarantee inter-operability of different components which support multicast.

In defining a common interface for a wide variety of implementations, it is necessary to define the standard interface with some level of abstraction in order to avoid limiting implementation flexibility. Therefore, several examples of the use of the interface have been included.

1.3 Requirements

The multicast mechanism shall fulfill the following goals:

- Simple - excess complexity will not gain acceptance
- Compact - Does not cost excessive silicon area in a switch
- Robust - same level of protection and recovery as the rest of RapidIO
- Scalable - must be able to extend to multi-layer switch systems
- Compatibility with all physical layers

Chapter 2 Multicast Extensions Behavior

2.1 Introduction

This chapter describes the multicast extensions rules of operation in a RapidIO system. A RapidIO switch which does not support multicast can co-exist in a RapidIO fabric with other switches that do support multicast. The only requirement is that the switch be capable of routing the destination IDs used for multicast transactions.

2.2 Packet Replication

A RapidIO multicast operation consists of the replication of a single packet so that it can be received by multiple end points. This replication is performed by the switch devices in the fabric rather than by the end point itself, so that the capability to replicate packets expands with the number of switches (and hence possible end points) in a system. Each switch may be individually programmed to control which egress ports of the switch the replicated packets are sent to, and thus indirectly which specific set of end point devices receive the replicated packet. The packets themselves are not modified by the replication process, merely transmitted out through the appropriate ports.

This specification only addresses multicasting request packets for transactions which do not require responses. This greatly simplifies multicast support for RapidIO switches, which will therefore have no need to aggregate responses from other types of RapidIO operations. Examples of transactions which can be multicast are I/O logical specification NWRITE and SWRITE transactions. Multicasting transactions which require responses have implementation defined behavior.

2.3 Multicast Operation

Multicast operations have two control value types **- multicast masks** and **multicast groups**. The set of target end points which all receive a particular multicast packet is known as a **multicast group**. Each multicast group is associated with a unique destination ID. The destination ID of a received packet allows a RapidIO switch device to determine that a packet is to be replicated for a multicast.

A **multicast mask** is a value that controls which egress ports one or more multicast groups are associated with. Conceptually, a multicast mask is a register with one enable bit for each possible switch egress port. There is one set of multicast masks for the entire switch. All multicast masks in a switch are assigned unique sequential ID numbers beginning with 0. Figure 2-1 shows an example of the use of multicast in a RapidIO system.

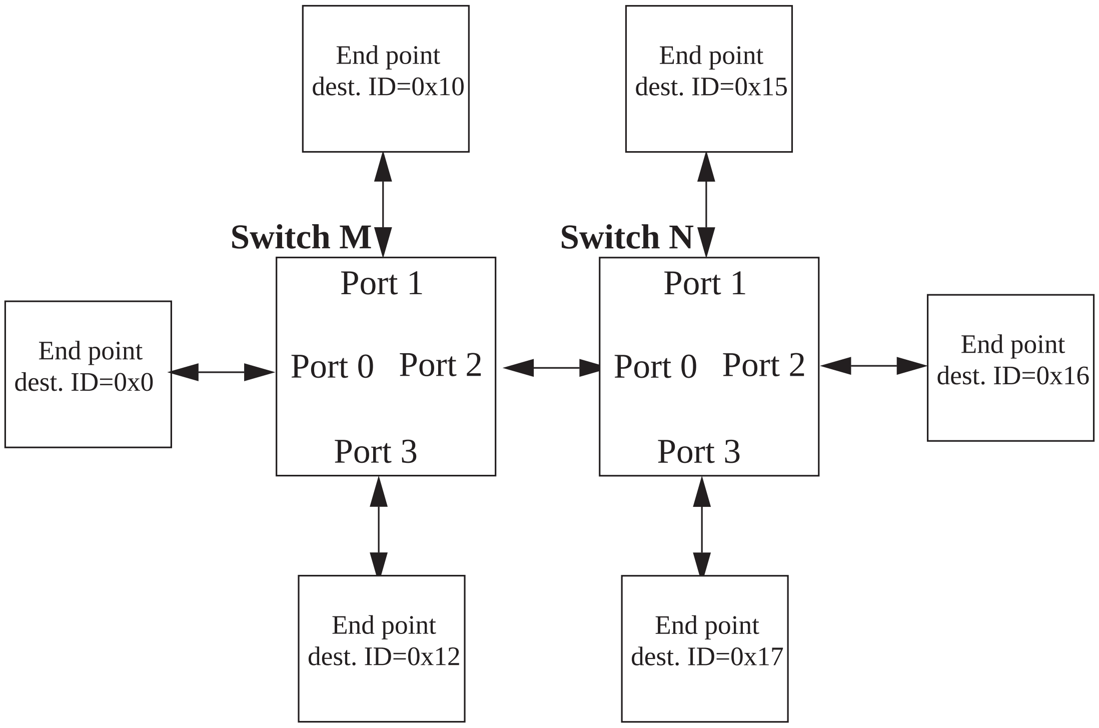


Figure 2-1. Multicast System Example

In this example, the end point assigned destination ID 0x0 uses destination ID 0x80 to perform multicast operations to the multicast group comprised of end points 0x10, 0x15, 0x16, and 0x17, arbitrarily called group A. Software configures the switch devices in the fabric to **associate** the destination IDs that represent multicast groups with multicast masks. For Figure 2-1 switch M associates destination ID 0x80 with egress ports 1 and 2, and switch N associates destination ID 0x80 with ports 1, 2, and 3. Figure 2-2 shows a possible relationship between the multicast group, the multicast masks for the switches, and the global system address map.

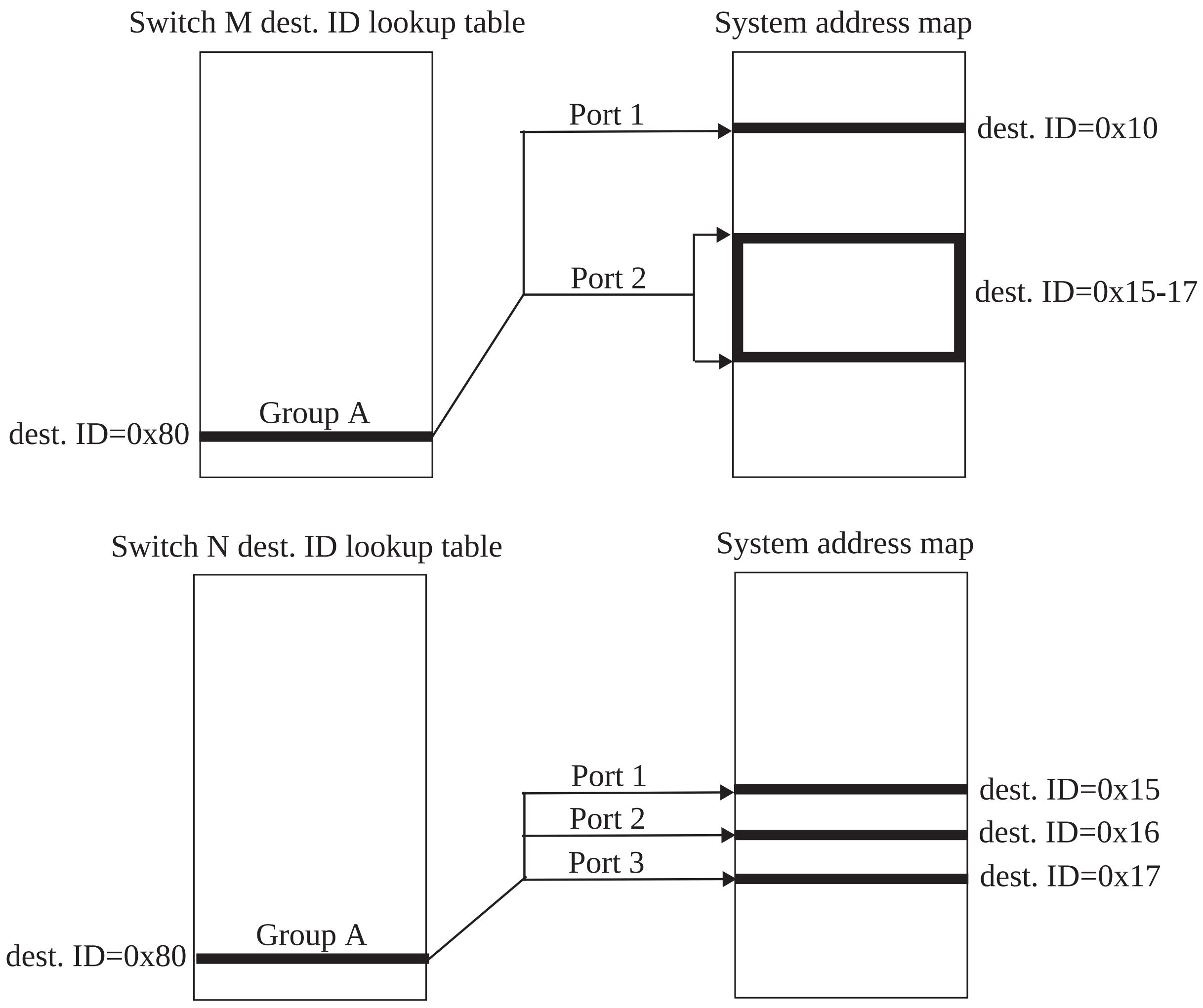


Figure 2-2. Multicast Association Example

Configuring a RapidIO switch to replicate packets for a multicast group is a two-step process. First, a list of egress ports is set in a multicast mask list. Second, one or more destination IDs which represent the multicast groups are associated with the multicast mask in the switch. During normal system operation, any time a switch receives a packet with a destination ID which has been associated with a multicast mask it will send that packet to all egress ports enabled by that multicast mask.

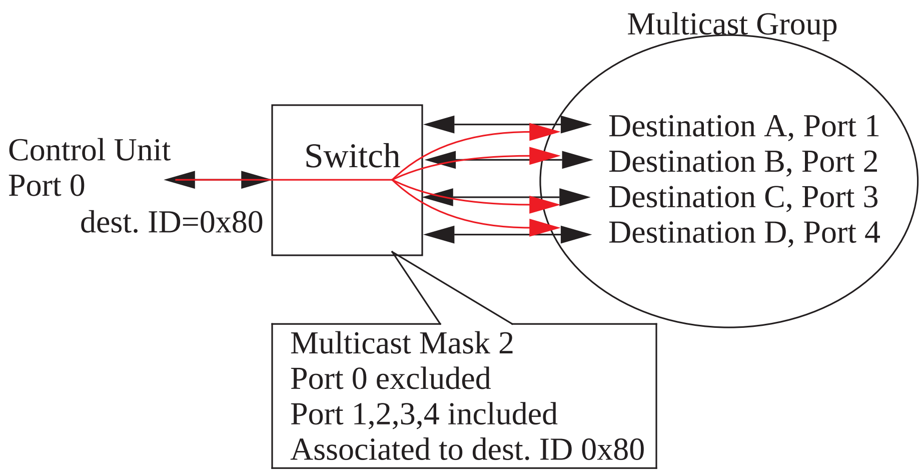


Figure 2-3. Multicast Configuration Example

Figure 2-3 shows a control unit connected to switch port 0 which needs to multicast to destinations A, B, C and D. A multicast mask, in this case arbitrarily picked as multicast mask 2, is set up to select which ports in the switch are part of the multicast group of destinations A, B, C, and D. A destination ID, in this case arbitrarily assigned 0x80, is associated with multicast mask 2 as the destination ID that the control unit should use to multicast to the multicast group. The associate operation is done using the CSRs defined in Chapter 3, “Multicast Extensions Registers”.

The defined CSRs allow a switch to associate destination IDs with multicast masks using a small number of maintenance write operations. The number of unique destination IDs that can be associated with a multicast mask is also defined in a CSR.

While each destination ID is associated with a unique multicast group, the programming model allows a destination ID to be mapped to a different multicast mask for each port on the switch. However, for each port a destination ID can be associated with at most one multicast mask. The last association operation performed for a specific port and destination ID dictates which multicast mask the destination ID is associated with. It is also possible to map a given destination ID to the same multicast mask for all ports.

A RapidIO switch may be capable of supporting large numbers of multicast groups by dedicating a sequential range of destination ID’s to an equal number of sequentially numbered multicast masks. A switch may also be designed which does not require all multicast destination IDs to be sequential. The programming model supports both of these implementations.

A packet will never be multicast back out of the port it was received on even if it is included in the multicast mask for that destination ID. This allows a group of end points which need to multicast to each other to share the same multicast mask. Packets using a multicast mask which has no egress ports selected will be dropped without error notification. A device may have implementation specific error notification in this situation, depending on system requirements.

The default state after a reset for multicast masks is that all multicast masks have no ports selected. Additionally, after reset no associations exist between any multicast group/destination ID and the multicast masks. However, implementation specific capabilities may modify the multicast mask values and associations after reset without software intervention.

For more information and examples on the use of the programming model for multicast refer to Annex B, “Multicast Applications (Informative)”.

2.4 Multicast Transaction Ordering Requirements

RapidIO packets which are in the same multicast group (the same destination ID) with the same flowID and are received on the same ingress port must be multicast on the egress ports in the same order that they were received. There are no ordering requirements between multicast packets and non-multicast packets, or between multicast packets in different multicast groups. Maintaining ordering between transactions in the same transaction request flow for a multicast group allows an application to multicast a completion flag at the end of a potentially large data transfer which was sent to the same multicast group.

Blank page

Chapter 3 Multicast Extensions Registers

3.1 Introduction

This section describes the Multicast Extensions CAR and CSR registers that allow an external processing element to determine if a switch supports the multicast extensions defined in this specification, and to manage the configuration of multicast groups for a switch processing element. This chapter only describes registers or register bits defined by this specification. Refer to the other RapidIO logical, transport, physical, and extension specifications of interest to determine a complete list of registers and bit definitions for a device. All registers are 32-bits and aligned to a 32-bit boundary. The behavior of reserved register bits and register offsets and access rules and requirements are described in the *RapidIO Part 1: Input/Output Logical Specification*.

Table 3-1. Multicast Register Map

Configuration Space Byte Offset	Register Name
0x0-C	Reserved
0x10	Processing Element Features CAR
0x14-2C	Reserved
0x30	Switch Multicast Support CAR
0x34	Reserved
0x38	Switch Multicast Information CAR
0x3C-7C	Reserved
0x80	Multicast Mask Port CSR
0x84	Multicast Associate Select CSR
0x88	Multicast Associate Operation CSR
0x8C–FC	Reserved
0x100–FFFC	Extended Features Space
0x10000–FFFFFC	Implementation-defined Space

3.2 Capability Registers (CARs)

3.2.1 Processing Elements Features CAR (Configuration Space Offset 0x10)

The Processing Elements Features CAR contains 31 processing elements features bits defined in various RapidIO specifications, as well as the Multicast Support bit, defined here.

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

Bit	Name	Reset Value	Description
0-20	-		Reserved (defined elsewhere)
21	Multicast Support	*	Support for multicast extensions 0b0 - Does not support multicast extensions 0b1 - Supports multicast extensions
22-31	-		Reserved (defined elsewhere)

* Implementation dependant

3.2.2 Switch Multicast Support CAR (Configuration Space Offset 0x30)

This register shall not be implemented if Bit 19 “Dev32 Support” of the Processing Element Features CAR is set.

The Switch Multicast Support CAR defines support for a simple multicast model and the additional limits on multicast mask resources.

Table 3-3. Bit Settings for Switch Multicast Support CAR

Bit	Name	Reset Value	Description
0	Simple_Assoc	*	Support for a simple multicast association model 0b0 - Does not support simple association 0b1 - Supports simple association If this bit is set, the Block_Assoc bit in the Switch Multicast Information CAR must also be set.
1-31	-		Reserved (defined elsewhere)

* Implementation dependant

3.2.3 Switch Multicast Information CAR (Configuration Space Offset 0x38)

This register shall not be implemented if Bit 19 “Dev32 Support” of the Processing Element Features CAR is set.

The Switch Multicast Information CAR defines the methods for associating destination IDs with multicast masks supported by a RapidIO switch device. It also defines the limits on multicast mask resources.

Table 3-4. Bit Settings for Switch Multicast Information CAR

Bits	Name	Description
0	Block_Assoc	Block association support - allows equal sized blocks of destination IDs and multicast masks to be associated with each other with a single operation rather than one at a time. 0b0 - block association is not supported 0b1 - block association is supported If the Simple_Assoc bit in the Switch Multicast Support CAR is set, this bit must also be set.
1	Per_Port_Assoc	Per ingress port association support - allows a destination ID to be associated with a multicast mask on a per-ingress port basis rather than a single association for the entire switch. 0b0 - per port association is not supported 0b1 - per port association is supported
2-15	MaxDestIDAssoc	The maximum number of destination IDs associations per multicast mask 0x0000 - 1 destination ID 0x0001 - 2 destination IDs ... 0x3FFF - 16384 destination IDs
16-31	MaxMcastMasks	The number of multicast egress port masks available. This field also defines the largest block of destination IDs that can be block associated. 0x0000 - [reserved] 0x0001 - 1 multicast mask 0x0002 - 2 multicast masks ... 0xFFFF - 65535 multicast masks

3.3 Command and Status Registers (CSRs)

3.3.1 Multicast Mask Port CSR (Configuration Space Offset 0x80)

This register shall not be implemented if Bit 19 “Dev32 Support” of the Processing Element Features CAR is set.

The Multicast Mask Port CSR allows configuration of the egress port list for each of the switch’s multicast masks.

Writing the Write_to_Verify command sets up a Mcast_Mask and Egress_Port_Num pair to verify. The presence of the specified egress port in the specified multicast mask is indicated by the Port_Present bit on a subsequent read of the register.

Writing the Add_Port or Delete_Port commands adds or deletes the specified egress port to or from the specified multicast mask.

Writing the Add_All_Ports or Delete_All_Ports commands adds or deletes all of the egress ports in the specified multicast mask.

The result of illegal values or combinations for an operation is implementation dependent. For examples of how to use this register, refer to Section 4.2, “Configuring Multicast Masks”.

Table 3-5. Bit Settings for Multicast Mask Port CSR

Bits	Name	Reset Value	Description
0-15	Mcast_Mask	0x0000	Specifies the multicast mask which is to be modified or queried as determined by the Mask_Cmd field.
16-23	Egress_Port_Num	0x00	Specifies the port number to be added, deleted, or queried with the Mask_Cmd field.
24	-	0b0	Reserved
25-27	Mask_Cmd	0b000	Specifies the mask action on a write. 0b000 - Write_to_Verify 0b001 - Add_Port 0b010 - Delete_Port 0b011 - reserved 0b100 - Delete_All_Ports 0b101 - Add_All_Ports 0b110 - reserved 0b111 - reserved
28–30	-	0b000	Reserved
31	Port_Present	0b0	Indicates the existence of the egress port and multicast mask pair as a result of the last preceding Write_to_Verify command. 0b0 - Port was not enabled as an egress port in the specified multicast mask 0b1 - Port was enabled as an egress port in the specified multicast mask. This bit is reserved on a write.

3.3.2 Multicast Associate Select CSR (Configuration Space Offset 0x84)

This register shall not be implemented if Bit 19 “Dev32 Support” of the Processing Element Features CAR is set.

This register specifies the destination ID and multicast mask number for a subsequent associate operation controlled with the Multicast Associate Operation CSR. If block association is supported, this register specifies the start of the block to associate. For examples of how this register is used, refer to Section 4.4, “Configuring Associations”.

Table 3-6. Bit Settings for Multicast Associate Select CSR

Bits	Name	Reset Value	Description
0-7	Large_DestID	0x00	Selects the most significant byte of a large transport destination ID for an association operation
8-15	DestID	0x00	Selects the destination ID for an association operation
16-31	Mcast_Mask_Num	0x0000	Selects the multicast mask number for an association operation

3.3.3 Multicast Associate Operation CSR (Configuration Space Offset 0x88)

This register shall not be implemented if Bit 19 “Dev32 Support” of the Processing Element Features CAR is set.

The Multicast Associate Operation CSR specifies three operations for associating destination IDs with multicast masks. The affected destination ID and multicast mask is specified in the Multicast Associate Select CSR. The specified operation is executed when this register is written. When this register is read and the Assoc_Cmd field it set to Write_to_Verify the specified operation is executed and the updated register state is returned. If this register is read and the Assoc_Cmd field is not set to Write_to_Verify the resulting behavior is implementation dependent. Block association operations assign associations sequentially starting with the destination ID and multicast mask specified in the Multicast Associate Select CSR.

Writing the Write_To_Verify command checks for an association between the destination ID and multicast mask specified in the Multicast Associate Select CSR. The result of the check is indicated by the state of the Assoc_Present bit on a read of this register. This command cannot be executed on a block.

Writing the Add_Assoc or Delete_Assoc command adds or deletes the association between the destination ID and the multicast mask (or block of associations, if block association is supported) specified in the Multicast Associate Select CSR.

The result of illegal values or field combinations for an association operation is implementation dependent. For examples of how this register is used, refer to Section 4.4, “Configuring Associations”.

Table 3-7. Bit Settings for Multicast Associate Operation CSR

Bits	Name	Reset Value	Description
0-15	Assoc_Blksize	0x0000	This field specifies the number of sequential DestinationIDs to be associated with an equal number of sequential multicast mask numbers if block association is supported. This field is ignored on a Write_to_Verify command. 0x0000 - one association 0x0001 - two sequential associations ... 0xFFFF - 65536 sequential associations
16-23	Ingress_Port	0x00	This field specifies the ingress port association to affect if per-port ingress association is supported
24	Large_Transport	0b0	0b0 - the association is for small transport destination IDs 0b1 - the association is for large transport destination IDs
25-26	Assoc_Cmd	0b00	This field specifies the command to execute when this register is written. 0b00 - Write_To_Verify 0b01 - reserved 0b10 - Delete_Assoc 0b11 - Add_Assoc

Table 3-7. Bit Settings for Multicast Associate Operation CSR (Continued)

Bits	Name	Reset Value	Description
27-30	-	0b0000	reserved
31	Assoc_Present	0b0	This bit contains the result of the last Write_to_Verify command executed. 0b0 - no association present 0b1 - association present This bit is reserved on write.

3.4 Switch Routing Table Register Block

A switch device which has bits 19 “Dev32 Support” and 21 “Multicast Support” set in the Processing Element Features CAR shall implement this register block. Note that this definition is a refinement of the Switch Routing Table Register block defined in RapidIO Part 3: Common Transport Specification.

3.4.1 Register Map

The register map for the routing table registers shall be as specified by Table 3-8. 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+0xBC]. Register map offset [EF_PTR+0x140] can be used for another Extended Features block.

Table 3-8. Switch Routing Table Register Map

	Block Byte Offset	Register Name
General	0x0	Routing Table Register Block Header
	0x4-0x1C	Reserved
Broadcast	0x20	Broadcast Routing Table Control CSR
	0x24	Reserved
	0x28	Broadcast Multicast Info CSR
	0x2C-0x3C	Reserved
Port 0	0x40	Port 0 Routing Table Control CSR
	0x44	Reserved
	0x48	Port 0 Multicast Info CSR
	0x4C-0x5C	Reserved

Table 3-8. Switch Routing Table Register Map

	Block Byte Offset	Register Name
Port 1	0x60	Port 1 Routing Table Control CSR
	0x64	Reserved
	0x68	Port 1 Multicast Info CSR
	0x6C-0x7C	Reserved
Ports 2-14	0x80–21C	Assigned to Port 2-14 CSRs
Port 15	0x220	Port 15 Routing Table Control CSR
	0x224	Reserved
	0x228	Port 15 Multicast Info CSR
	0x22C-0x23C	Reserved

3.4.2 Broadcast Routing Table Control CSR (Block Offset 0x20)

The use and meaning of the bits and bit fields of this register shall be as specified in Table 3-9. Unless otherwise specified, the bits and bit fields in this register are write only.

Table 3-9. Bit Settings for Broadcast Routing Table Control CSR

Bit	Name	Reset Value	Description
0-5	___		Reserved
6-7	Mask_size	see footnote[1]	A multicast mask shall consist of the number of registers indicated by this value, encoded as follows: 0b00 - One set register, one clear register (8 bytes) 0b01 - Two set registers, two clear registers (16 bytes) 0b10 - Four set registers, four clear registers (32 bytes) 0b11 - Eight set registers, eight clear registers (64 bytes) This bit field shall be read only.
8-31	___		Reserved

[1]The Mask_size reset value is implementation dependent

The following illustrates the arrangement of Multicast Mask x Set/Clear Register y CSRs for various Mask_size values.

Table 3-10. Mask 0-7 Set/Clear Registers, Mask_size = 0

Register Name	Offset
Multicast Mask 0 Set Register 0	0x000000
Multicast Mask 0 Clear Register 0	0x000004
Multicast Mask 1 Set Register 0	0x000008
Multicast Mask 1 Clear Register 0	0x00000C
Multicast Mask 2 Set Register 0	0x000010
Multicast Mask 2 Clear Register 0	0x000014
Multicast Mask 3 Set Register 0	0x000018
Multicast Mask 3 Clear Register 0	0x00001C
Multicast Mask 4 Set Register 0	0x000020
Multicast Mask 4Clear Register 0	0x000024
Multicast Mask 5 Set Register 0	0x000028
Multicast Mask 5 Clear Register 0	0x00002C
Multicast Mask 6 Set Register 0	0x000030
Multicast Mask 6 Clear Register 0	0x000034
Multicast Mask 7 Set Register 0	0x000038
Multicast Mask 7 Clear Register 0	0x00003C

Table 3-11. Mask 0-3 Set/Clear Registers, Mask_size = 1

Register Name	Offset
Multicast Mask 0 Set Register 0	0x000000
Multicast Mask 0 Set Register 1	0x000004
Multicast Mask 0 Clear Register 0	0x000008
Multicast Mask 0 Clear Register 1	0x00000C
Multicast Mask 1 Set Register 0	0x000010
Multicast Mask 1 Set Register 1	0x000014
Multicast Mask 1 Clear Register 0	0x000018
Multicast Mask 1 Clear Register 1	0x00001C
Multicast Mask 2 Set Register 0	0x000020
Multicast Mask 2 Set Register 1	0x000024
Multicast Mask 2 Clear Register 0	0x000028
Multicast Mask 2 Clear Register 1	0x00002C
Multicast Mask 3 Set Register 0	0x000030
Multicast Mask 3 Set Register 1	0x000034
Multicast Mask 3 Clear Register 0	0x000038
Multicast Mask 3 Clear Register 1	0x00003C

Table 3-12. Mask 0-1 Set/Clear Registers, Mask_size = 2

Register Name	Offset
Multicast Mask 0 Set Register 0	0x000000
Multicast Mask 0 Set Register 1	0x000004
Multicast Mask 0 Set Register 2	0x000008
Multicast Mask 0 Set Register 3	0x00000C
Multicast Mask 0 Clear Register 0	0x000010
Multicast Mask 0 Clear Register 1	0x000014
Multicast Mask 0 Clear Register 2	0x000018
Multicast Mask 0 Clear Register 3	0x00001C
Multicast Mask 1 Set Register 0	0x000020
Multicast Mask 1 Set Register 1	0x000024
Multicast Mask 1 Set Register 2	0x000028
Multicast Mask 1 Set Register 3	0x00002C
Multicast Mask 1 Clear Register 0	0x000030
Multicast Mask 1 Clear Register 1	0x000034
Multicast Mask 1 Clear Register 2	0x000038
Multicast Mask 1 Clear Register 3	0x00003C

Table 3-13. Mask 0 Set/Clear Registers, Mask_size = 3

Register Name	Offset
Multicast Mask 0 Set Register 0	0x000000
Multicast Mask 0 Set Register 1	0x000004
Multicast Mask 0 Set Register 2	0x000008
Multicast Mask 0 Set Register 3	0x00000C
Multicast Mask 0 Set Register 4	0x000010
Multicast Mask 0 Set Register 5	0x000014
Multicast Mask 0 Set Register 6	0x000018
Multicast Mask 0 Set Register 7	0x00001C
Multicast Mask 0 Clear Register 0	0x000020
Multicast Mask 0 Clear Register 1	0x000024
Multicast Mask 0 Clear Register 2	0x000028
Multicast Mask 0 Clear Register 3	0x00002C
Multicast Mask 0 Clear Register 4	0x000030
Multicast Mask 0 Clear Register 5	0x000034
Multicast Mask 0 Clear Register 6	0x000038
Multicast Mask 0 Clear Register 7	0x00003C

3.4.3 Broadcast Multicast Info CSR (Block Offset 0x28)

This register shall communicate the location of the Broadcast Multicast Mask 0 Set Register 0 CSR. The use and meaning of the bits and bit fields of this register shall be as specified in Table 3-14. This register shall be read only.

Table 3-14. Bit Settings for Broadcast Multicast Info CSR

Bit	Name	Reset Value	Description
0-7	Num_Masks	see footnote[1]	Num_Masks shall indicate the number of Broadcast Multicast Masks, encoded as follows: 0x00 - 256 Masks 0x01 - 1 Masks 0x02 - 2 Masks 0x03 - 3 Masks ... 0xFF - 255 Masks
8-21	Mask_Ptr	see footnote[2]	The Mask_Ptr value shall be the maintenance offset of the Broadcast Multicast Mask 0 Set Register 0 CSR, divided by 1024.The maintenance offset of the Broadcast Multicast Mask 0 Set Register 0 CSR shall be a 1024 byte aligned address. The Mask_Ptr value shall indicate an address in Implementation Defined register space. Writes to the Broadcast Multicast Mask registers pointed to by this register shall cause the corresponding Port n Multicast Mask registers for all ports to assume the value written.
22-31	___	0b00	Reserved

[1]The Num_Masks reset value is implementation dependent

[2]The Mask_Ptr reset value is implementation dependent

3.4.4 Port n Routing Table Control CSR (Block Offset 0x40 + (0x20 * n))

This register shall indicate the number of registers in a multicast mask for all ports whose Port n Multicast Info CSR Mask_Ptr field value is the same. The use and meaning of the bits and bit fields of this register shall be as specified in Table 3-15. Unless otherwise specified, the bits and bit fields in this register are read/write.

Table 3-15. Bit Settings for Port n Routing Table Control CSR

Bit	Name	Reset Value	Description
0-5	___		Reserved

Table 3-15. Bit Settings for Port n Routing Table Control CSR

Bit	Name	Reset Value	Description
6-7	Mask_size	see footnote[1]	A multicast mask shall consist of the number of registers indicated by this value, encoded as follows: 0b00 - One set register, one clear register (8 bytes) 0b01 - Two set registers, two clear registers (16 bytes) 0b10 - Four set registers, four clear registers (32 bytes) 0b11 - Eight set registers, eight clear registers (64 bytes) This bit field shall be read only.
8-31	___		Reserved

[1]The Mask_size reset value is implementation dependent

3.4.5 Port n Multicast Info CSR (Block Offset 0x48 + 20 * n)

This register shall communicate the location of Port n Multicast Mask 0 Register 0 CSR. The use and meaning of the bits and bit fields of this register shall be as specified in Table 3-16. This register shall be read only.

Table 3-16. Bit Settings for Port n Multicast Info CSR

Bit	Name	Reset Value	Description
0-7	Num_Masks	see footnote[1]	Communicates the number of multicast masks for this port. Num_Masks is encoded as follows: 0x00 - 256 Masks 0x01 - 1 Masks 0x02 - 2 Masks 0x03 - 3 Masks ... 0xFF - 255 Masks
8-21	Mask_Ptr	see footnote[2]	The Mask_Ptr value shall be the maintenance offset of the Port n Multicast Mask 0 Set Register 0 CSR, divided by 1024.The maintenance offset of the Port n Multicast Mask 0 Set Register 0 CSR shall be a 1024 byte aligned address. The Mask_Ptr value shall indicate an address in Implementation Defined register space.
22-31	___	0b00	Reserved

[1]The Num_Masks reset value is implementation dependent

[2]The Mask_Ptr reset value is implementation dependent

3.4.6 Broadcast Multicast Mask x Set Register y CSR (Offset = (Mask_Ptr * 0x400) + (x*8*$2^{\text{Mask\_size}}$) + (y*4))

Writes to the Broadcast Multicast Mask x Set Register y CSRs shall cause the corresponding Port n Multicast Mask x Set Register y CSRs for all ports to assume the value written. The use and meaning of the bits and bit fields of this register shall be as specified in Table 3-17. Unless otherwise specified, the bits and bit fields in this register are write only.

Table 3-17. Bit Settings for Broadcast Multicast Mask x Set Register y CSR

Bit	Name	Reset Value	Description
0-31	Mcast_ctl	All 0’s	This register controls which ports do and do not receive packets when routed according to this multicast mask. The multicast mask functionality for ports (y*32) to (y*32+31) shall be controlled by this register. Bits shall be assigned to ports sequentially as the port number increases. The lowest numbered port shall be assigned to Bit 31. Each bit shall be encoded as follows: 0b0 - Do not multicast to this port 0b1 - Multicast to this port Writing 0 to a bit shall not change the bit value. Writing 1 to a bit shall set the bit value. Bits corresponding to ports which do not exist in the device shall be reserved.

3.4.7 Broadcast Multicast Mask x Clear Register y CSR (Offset = (Mask_Ptr * 0x400) + (x * 8*2$^{\text{Mask\_size}}$) + (4*2$^{\text{Mask\_size}}$) + (y*4))

Writes to the Broadcast Multicast Mask x Clear Register y CSRs shall cause the corresponding Port n Multicast Mask x Clear Register y CSRs for all ports to assume the value written. The use and meaning of the bits and bit fields of this register shall be as specified in Table 3-18. Unless otherwise specified, the bits and bit fields in this register are write only.

Table 3-18. Bit Settings for Broadcast Multicast Mask x Clear Register y CSR

Bit	Name	Reset Value	Description
0-31	Mcast_ctl	All 0’s	This register controls which ports do and do not receive messages when routed according to this multicast mask. The multicast mask functionality for ports (y*32) to (y*32+31) shall be controlled by this register. Bits shall be assigned to ports sequentially as the port number increases. The lowest numbered port shall be assigned to Bit 31. Each bit shall be encoded as follows: 0b0 - Do not multicast to this port 0b1 - Multicast to this port Writing 0 to a bit shall not change the bit value. Writing 1 to a bit shall clear the bit value. Bits corresponding to ports which do not exist in the device shall be reserved.

3.4.8 Port n Multicast Mask x Set Register y CSR (Offset = (Mask_Ptr * 400) + (x * 8*2^{Mask_size}) + (y*4))

This register shall control the multicast behavior for all ports whose Port n Multicast Info CSR Mask_Ptr field value is the same. The use and meaning of the bits and bit fields of this register shall be as specified in Table 3-19. Unless otherwise specified, the bits and bit fields in this register are read/write.

Table 3-19. Bit Settings for Port n Multicast Mask x Set Register y CSR

Bit	Name	Reset Value	Description
0-31	Mcast_ctl	All 0’s	This register controls which ports do and do not receive messages when routed according to this multicast mask. The multicast mask functionality for ports (y*32) to (y*32+31) shall be controlled by this register. Bits shall be assigned to ports sequentially as the port number increases. The lowest numbered port shall be assigned to Bit 31. Reading this register returns the current multicast value for the assigned ports. Each bit shall be encoded as follows: 0b0 - Do not multicast to this port 0b1 - Multicast to this port Writing 0 to a bit shall not change the bit value. Writing 1 to a bit shall set the bit value. Bits corresponding to ports which do not exist in the device shall be reserved.

3.4.9 Port n Multicast Mask x Clear Register y CSR (Offset = (Mask_Ptr * 400) + (x * 8*2^{Mask_size}) + (4*2^{Mask_size}) + (y*4))

This register shall control the multicast behavior for all ports whose Port n Multicast Info CSR Mask_Ptr field value is the same. The use and meaning of the bits and bit fields of this register shall be as specified in Table 3-20. Unless otherwise specified, the bits and bit fields in this register are read/write.

Table 3-20. Bit Settings for Port n Multicast Mask x Clear Register y CSR

Bit	Name	Reset Value	Description
0-31	Mcast_ctl	All 0’s	This register controls which ports do and do not receive messages when routed according to this multicast mask. The multicast mask functionality for ports (y*32) to (y*32+31) shall be controlled by this register. Bits shall be assigned to ports sequentially as the port number increases. The lowest numbered port shall be assigned to Bit 31. Reading this register returns the current multicast value for the assigned ports. Each bit shall be encoded as follows: 0b0 - Do not multicast to this port 0b1 - Multicast to this port Writing 0 to a bit shall not change the bit value. Writing 1 to a bit shall clear the bit value. Bits corresponding to ports which do not exist in the device shall be reserved.

Chapter 4 Configuration Examples

4.1 Introduction

This chapter provides several examples of how to use the multicast programming interface. The given examples build upon each other while proceeding through the sections. References to the order of operations within the examples run from the top of a list to the bottom unless otherwise stated.

Initially assume a switch with 8 ports which supports 4 or more multicast masks with two or more destination IDs allowed per multicast group so that a total of 8 destination IDs minimum can be associated with the multicast masks. The system has the following requirements:

- Three sources of traffic (ports 0, 1, and 2) must be multicast to two destinations (ports 6 and 7).
- Three ports (ports 3, 4 and 5) need to multicast signals between each other.
- All ports occasionally need to multicast to every other port.

Assume that the switch does not require any other multicast functions and therefore multicast masks 0, 1, and 2 will be used.

4.2 Configuring Multicast Masks

This section discusses assigning an egress port list to a multicast mask.

4.2.1 Clearing Multicast Masks

Suppose that the state of the multicast masks is unknown, and therefore the masks must be cleared before being configured. In order to clear the masks the following register accesses are made. (The accesses to the Multicast Mask Port CSR can be performed in any order.)

- Remove all ports from multicast mask 0
  - — Write the value 0x0000_0040 to the Multicast Mask Port CSR
- Remove all ports from multicast mask 1
  - — Write the value 0x0001_0040 to the Multicast Mask Port CSR
- Remove all ports from multicast mask 2
  - — Write the value 0x0002_0040 to the Multicast Mask Port CSR

4.2.2 Assigning Ports to Multicast Masks

To configure mask 0 to multicast to ports 6 and 7, mask 1 to multicast to ports 3, 4 and 5, and mask 2 to multicast to every port, requires the following series of register accesses. (The accesses to the Multicast Mask Port CSR can be performed in any order.)

- Add port 6 to multicast mask 0
  - — Write the value 0x0000_0610 to the Multicast Mask Port CSR
- Add port 7 to multicast mask 0
  - — Write the value 0x0000_0710 to the Multicast Mask Port CSR
- Add port 3 to multicast mask 1
  - — Write the value 0x0001_0310 to the Multicast Mask Port CSR
- Add port 4 to multicast mask 1
  - — Write the value 0x0001_0410 to the Multicast Mask Port CSR
- Add port 5 to multicast mask 1
  - — Write the value 0x0001_0510 to the Multicast Mask Port CSR
- Add all ports to multicast mask 2
  - — Write the value 0x0002_0050 to the Multicast Mask Port CSR

4.2.3 Removing a Port from a Multicast Mask

Suppose that the device attached to port 4 needs to be removed from the system. The following register accesses are used to modify multicast masks 1 and 2 to stop port 4 from being a multicast destination. (The accesses may be performed in any order.)

- Remove port 4 from multicast mask 1
  - — Write the value 0x0001_0420 to the Multicast Mask Port CSR
- Remove port 4 from multicast mask 2
  - — Write the value 0x0002_0420 to the Multicast Mask Port CSR

4.2.4 Querying a Multicast Mask

In this section suppose that a system designer needs to determine which of the 8 ports are included in multicast mask 2. The following accesses are to be performed to provide this information. (In each case, the write operation setting up the ‘Write to Verify’ operation must be performed before the subsequent read to check the Port Present bit status. The individual multicast masks may be queried in any order.)

- Verify that port 0 is included in mask 2
  - — Write the value 0x0002_0000 to the Multicast Mask Port CSR
  - — Read the value 0x0002_0001 from the Multicast Mask Port CSR

- Verify that port 1 is included in mask 2
  - Write the value 0x0002_0100 to the Multicast Mask Port CSR
  - Read the value 0x0002_0101 from the Multicast Mask Port CSR.
- Verify that port 2 is included in mask 2
  - Write the value 0x0002_0200 to the Multicast Mask Port CSR
  - Read the value 0x0002_0201 from the Multicast Mask Port CSR
- Verify that port 3 is included in mask 2
  - Write the value 0x0002_0300 to the Multicast Mask Port CSR
  - Read the value 0x0002_0301 from the Multicast Mask Port CSR
- Verify that port 4 is not included in mask 2
  - Write the value 0x0002_0400 to the Multicast Mask Port CSR
  - Read the value 0x0002_0400 from the Multicast Mask Port CSR
- Verify that port 5 is included in mask 2
  - Write the value 0x0002_0500 to the Multicast Mask Port CSR
  - Read the value 0x0002_0501 from the Multicast Mask Port CSR
- Verify that port 6 is included in mask 2
  - Write the value 0x0002_0600 to the Multicast Mask Port CSR
  - Read the value 0x0002_0601 from the Multicast Mask Port CSR
- Verify that port 7 is included in mask 2
  - Write the value 0x0002_0700 to the Multicast Mask Port CSR
  - Read the value 0x0002_0701 from the Multicast Mask Port CSR

4.3 Simple Association

If the Simple_Assoc bit is set in the Switch Multicast Support CAR, the device supports the simple multicast programming model. This model allows for basic multicast support for devices with a limited number of multicast masks, and requires a fixed relationship between those masks and sequential multicast groups.

4.3.1 Restrictions on Block Size

If the Simple_Assoc bit is set the device has a limited number of masks. Therefore, the number of sequential associations equals the maximum number of masks.

The Assoc_BlkSize field in the Multicast Associate Operation CSR must be set to the value of (MaxMCastMasks - 1). The MaxMCastMasks field is in the Switch Multicast Information CAR.

4.3.2 Restrictions on Block Associate

If the Simple_Assoc bit is set, non-block associations are precluded.

4.3.3 Restrictions on Associations

If the Simple_Assoc bit is set the device requires a fixed relationship between the sequential mask numbers and sequential destination IDs. This must be taken into account when the masks are associated.

The Multicast Associate Select CSR is set with the Mcast_Mask_num value set to 0x0000 and the Large_DestID and DestID fields set to an integer multiple of the MaxMCastMasks value.

Hardware that sets the new Simple_Assoc CAR bit could implement a single block associate for all of the masks that it supports with the requirement that they all be sequential destination IDs.

4.4 Configuring Associations

This section describes how to associate destination IDs with multicast masks, including examples of how to use the block association and per-port association functions.

4.4.1 Basic Association

For the assumed system it is now necessary to associate a destination ID with each multicast mask from the preceding examples. How this can be accomplished may vary depending on the capabilities of the switch. For this section, assume that neither block association nor per-ingress-port association is supported by the switch.

Following upon the previous example, assume the following additional system requirements.

- the 16 bit destination ID 0x1234 needs to be associated with multicast mask 0.
- the 8 bit destination ID 0x44 needs to be associated with multicast mask 1.
- the 16 bit destination ID 0xFEED needs to be associated with multicast mask 2.

In order to accomplished the desired associations, the following register accesses are required. (The individual association operations can be performed in any order.)

- Set up the operation to associate destination ID 0x1234 with multicast mask 0
  - — Write the value 0x1234_0000 to the Multicast Associate Select CSR
- Associate destination ID 0x1234 with multicast mask 0
  - — Write the value 0x0000_00E0 to the Multicast Associate Operation CSR
- Set up the operation to associate destination ID 0x44 with multicast mask 1

   — Write the value 0x0044_0001 to the Multicast Associate Select CSR
- Associate destination ID 0x44 with multicast mask 1
   — Write the value 0x0000_0060 to the Multicast Associate Operation CSR
- Set up the operation to associate destination ID 0xFEED with multicast mask 2
   — Write the value 0xFEED_0002 to the Multicast Associate Select CSR
- Associate destination ID 0xFEED with multicast mask 2
   — Write the value 0x0000_00E0 to the Multicast Associate Operation CSR

4.4.2 Using Per-Ingress Port Association

For the associations discussed in the preceding section, if the switch supports per-ingress-port association (destination IDs are associated with multicast masks on a per ingress port basis), the required programming operations change. The associations for each multicast mask are grouped into a write to the Multicast Associate Select CSR, followed by a write to the Multicast Associate Operation CSR for each ingress port that must be aware of the association. (The writes to the Multicast Associate Operation CSR can occur in any order but must occur after the related writes to the Multicast Associate Select CSR. The individual association operations can be performed in any order.)

- Set up the operation to associate destination ID 0x1234 with multicast mask 0
   — Write the value 0x1234_0000 to the Multicast Associate Select CSR
- Associate destination ID 0x1234 with multicast mask 0 on ingress port 0
   — Write the value 0x0000_00E0 to the Multicast Associate Operation CSR
- Associate destination ID 0x1234 with multicast mask 0 on ingress port 1
   — Write the value 0x0000_01E0 to the Multicast Associate Operation CSR
- Associate destination ID 0x1234 with multicast mask 0 on ingress port 2
   — Write the value 0x0000_02E0 to the Multicast Associate Operation CSR
- Set up the operation to associate destination ID 0x44 with multicast mask 1
   — Write the value 0x0044_0001 to the Multicast Associate Select CSR
- Associate destination ID 0x44 with multicast mask 1 on ingress port 3
   — Write the value 0x0000_0360 to the Multicast Associate Operation CSR
- Associate destination ID 0x44 with multicast mask 1 on ingress port 4
   — Write the value 0x0000_0460 to the Multicast Associate Operation CSR
- Associate destination ID 0x44 with multicast mask 1 on ingress port 5
   — Write the value 0x0000_0560 to the Multicast Associate Operation CSR
- Set up the operation to associate destination ID 0xFEED with multicast mask 2
   — Write the value 0xFEED_0002 to the Multicast Associate Select CSR
- Associate destination ID 0xFEED with multicast mask 2 on ingress port 0

    - Write the value 0x0000_00E0 to the Multicast Associate Operation CSR
- Associate destination ID 0xFEED with multicast mask 2 on ingress port 1
    - Write the value 0x0000_01E0 to the Multicast Associate Operation CSR
- Associate destination ID 0xFEED with multicast mask 2 on ingress port 2
    - Write the value 0x0000_02E0 to the Multicast Associate Operation CSR
- Associate destination ID 0xFEED with multicast mask 2 on ingress port 3
    - Write the value 0x0000_03E0 to the Multicast Associate Operation CSR
- Associate destination ID 0xFEED with multicast mask 2 on ingress port 4
    - Write the value 0x0000_04E0 to the Multicast Associate Operation CSR
- Associate destination ID 0xFEED with multicast mask 2 on ingress port 5
    - Write the value 0x0000_05E0 to the Multicast Associate Operation CSR
- Associate destination ID 0xFEED with multicast mask 2 on ingress port 6
    - Write the value 0x0000_06E0 to the Multicast Associate Operation CSR
- Associate destination ID 0xFEED with multicast mask 2 on ingress port 7
    - Write the value 0x0000_07E0 to the Multicast Associate Operation CSR

4.4.3 Using Block Association

In this section assume that the switch supports block association rather than per-ingress-port association. With this feature sequential destination IDs can be quickly associated to sequential multicast masks. In order to take advantage of this feature, different destination IDs assignments are required for the system than for the preceding examples. The starting destination 0xFF00 is arbitrarily selected.

- the 16 bit destination ID 0xFF00 is used to multicast from ports 0, 1 and 2 to ports 6 and 7, so destination ID 0xFF00 needs to be associated with multicast mask 0.
- the 16 bit destination ID 0xFF01 identifies the multicast group including ports 3, 4 and 5, so destination ID 0xFF01 needs to be associated with multicast mask 1.
- the 16 bit destination ID 0xFF02 identifies the multicast group that includes all ports, so destination ID 0xFF02 needs to be associated with multicast mask 2.

Note that the number of accesses needed to accomplish the desired associations is reduced to two. (The accesses must be performed in the order given.)

- Set up the associate operation starting with destination ID 0xFF00 and multicast mask 0
    - Write the value 0xFF00_0000 to the Multicast Associate Select CSR
- Associate three sequential destination IDs starting at 0xFF00 with three sequential multicast masks starting at 0
    - Write the value 0x0002_00E0 to the Multicast Associate Operation CSR

4.4.4 Using Per-Ingress Port and Block Association

Next, if both block association and per-ingress port association are supported by the switch, then the following sequence of operations is required. (The write to the Multicast Associate Select CSR must occur before the corresponding write to the Multicast Associate Operation CSR. The individual association operations can be performed in any order.)

- Set up the associate operations starting with destination ID 0xFF00 and multicast mask 0
  - Write the value 0xFF00_0000 to the Multicast Associate Select CSR
- Associate three sequential destination IDs with three sequential multicast masks for ingress port 0
  - Write the value 0x0002_00E0 to the Multicast Associate Operation CSR
- Associate three sequential destination IDs with three sequential multicast masks for ingress port 1
  - Write the value 0x0002_01E0 to the Multicast Associate Operation CSR
- Associate three sequential destination IDs with three sequential multicast masks for ingress port 2
  - Write the value 0x0002_02E0 to the Multicast Associate Operation CSR
- Associate three sequential destination IDs with three sequential multicast masks for ingress port 3
  - Write the value 0x0002_03E0 to the Multicast Associate Operation CSR
- Associate three sequential destination IDs with three sequential multicast masks for ingress port 4
  - Write the value 0x0002_04E0 to the Multicast Associate Operation CSR
- Associate three sequential destination IDs with three sequential multicast masks for ingress port 5
  - Write the value 0x0002_05E0 to the Multicast Associate Operation CSR
- Associate three sequential destination IDs with three sequential multicast masks for ingress port 6
  - Write the value 0x0002_06E0 to the Multicast Associate Operation CSR
- Associate three sequential destination IDs with three sequential multicast masks for ingress port 7
  - Write the value 0x0002_07E0 to the Multicast Associate Operation CSR

For this example, suppose that ingress port 4 needs a second destination ID to be mapped to each of the three multicast masks and the switch also has this capability. The second destination would be added to port 4 with the following association operation. (The write to the Multicast Associate Select CSR must occur before the write to the Multicast Associate Operation CSR.

- Set up the associate operations starting with destination ID 0xFF03 and multicast mask 0
    - Write the value 0xFF03_0000 to the Multicast Associate Select CSR
- Associate three sequential destination IDs with three sequential multicast masks for ingress port 4
    - Write the value 0x0002_04E0 to the Multicast Associate Operation CSR

4.4.5 Removing a Destination ID to Multicast Mask Association

Now assume that packets from destination ID 0xFF02 on port 4 should no longer be allowed to multicast to all nodes (multicast mask 2). To remove destination ID 0xFF02 from being associated with multicast mask 2 on port 4, the following register accesses need to be performed in order.

- Set up the operation to remove the association between destination ID 0xFF02 and multicast mask 2
    - Write the value 0xFF02_0002 to the Multicast Associate Select CSR
- Remove the association between destination ID 0xFF02 and multicast mask 2 on ingress port 4
    - Write the value 0x0000_04C0 to the Multicast Associate Operation CSR

4.4.6 Querying an Association

There are three scenarios for querying destination ID to multicast mask associations in a switch. For the first scenario assume that a system designer wants to know which multicast masks are associated with destination ID 0xFF01 on port 4. Note that since a read of the Multicast Associate Operation CSR causes the last command written to be executed, that register is only written at the beginning of the sequence. (The individual associations can be queried in any order.)

- Set up the associate operations for destination ID 0xFF01 and multicast mask 0
    - Write the value 0xFF01_0000 to the Multicast Associate Select CSR
- Verify that destination ID 0xFF01 is not associated with multicast mask 0 for port 4
    - Write the value 0x0000_0480 to the Multicast Associate Operation CSR
    - Read the value 0x0000_0480 from the Multicast Associate Operation CSR
- Set up the associate operations for destination ID 0xFF01 and multicast mask 1
    - Write the value 0xFF01_0001 to the Multicast Associate Select CSR
- Verify that destination ID 0xFF01 is not associated with multicast mask 1 for port 4
    - Read the value 0x0000_0480 from the Multicast Associate Operation CSR

- Set up the associate operations for destination ID 0xFF01 and multicast mask 2
  - Write the value 0xFF01_0002 to the Multicast Associate Select CSR
- Verify that destination ID 0xFF01 is associated with multicast mask 2 for port 4
  - Read the value 0x0000_0481 from the Multicast Associate Operation CSR
- Set up the associate operations for destination ID 0xFF01 and multicast mask 3
  - Write the value 0xFF01_0003 to the Multicast Associate Select CSR
- Verify that destination ID 0xFF01 is not associated with multicast mask 3 for port 4
  - Read the value 0x0000_0480 from the Multicast Associate Operation CSR

For the second scenario assume that the system designer wants to know which destination IDs from 0xFF00 through 0xFF07 are associated with multicast mask 0 on Port 4. (The individual associations may be queried in any order.)

- Set up the associate operations for destination ID 0xFF00 and multicast mask 0
  - Write the value 0xFF00_0000 to the Multicast Associate Select CSR
- Verify that destination ID 0xFF00 is associated with multicast mask 0 for port 4
  - Write the value 0x0000_0480 to the Multicast Associate Operation CSR
  - Read the value 0x0000_0480 from the Multicast Associate Operation CSR
- Set up the associate operations for destination ID 0xFF00 and multicast mask 0
  - Write the value 0xFF01_0000 to the Multicast Associate Select CSR
- Verify that destination ID 0xFF01 is not associated with multicast mask 0 for port 4
  - Read the value 0x0000_0480 from the Multicast Associate Operation CSR
- Set up the associate operations for destination ID 0xFF00 and multicast mask 0
  - Write the value 0xFF02_0000 to the Multicast Associate Select CSR
- Verify that destination ID 0xFF02 is not associated with multicast mask 0 for port 4
  - Read the value 0x0000_0480 from the Multicast Associate Operation CSR
- Set up the associate operations for destination ID 0xFF00 and multicast mask 0
  - Write the value 0xFF03_0000 to the Multicast Associate Select CSR
- Verify that destination ID 0xFF03 is not associated with multicast mask 0 for port 4
  - Read the value 0x0000_0480 from the Multicast Associate Operation CSR

- Set up the associate operations for destination ID 0xFF00 and multicast mask 0
  - Write the value 0xFF04_0000 to the Multicast Associate Select CSR
- Verify that destination ID 0xFF04 is not associated with multicast mask 0 for port 4
  - Read the value 0x0000_0480 from the Multicast Associate Operation CSR
- Set up the associate operations for destination ID 0xFF00 and multicast mask 0
  - Write the value 0xFF05_0000 to the Multicast Associate Select CSR
- Verify that destination ID 0xFF05 is associated with multicast mask 0 for port 4
  - Read the value 0x0000_0481 from the Multicast Associate Operation CSR
- Set up the associate operations for destination ID 0xFF00 and multicast mask 0
  - Write the value 0xFF06_0000 to the Multicast Associate Select CSR
- Verify that destination ID 0xFF06 is not associated with multicast mask 0 for port 4
  - Read the value 0x0000_0480 from the Multicast Associate Operation CSR
- Set up the associate operations for destination ID 0xFF00 and multicast mask 0
  - Write the value 0xFF07_0000 to the Multicast Associate Select CSR
- Verify that destination ID 0xFF07 is not associated with multicast mask 0 for port 4
  - Read the value 0x0000_0480 from the Multicast Associate Operation CSR

For the last scenario assume that the system designer now wants to know whether or not destination ID 0xFF03 is mapped to multicast mask 3 on all ports. (The individual associations may be queried in any order.)

- Set up the associate operations for destination ID 0xFF03 and multicast mask 3
  - Write the value 0xFF03_0003 to the Multicast Associate Select CSR
- Verify that destination ID 0xFF03 is not associated with multicast mask 3 for port 0
  - Write the value 0x0000_0080 to the Multicast Associate Operation CSR
  - Read the value 0x0000_0080 from the Multicast Associate Operation CSR
- Verify that destination ID 0xFF03 is not associated with multicast mask 3 for port 1
  - Write the value 0x0000_0180 to the Multicast Associate Operation CSR
  - Read the value 0x0000_0180 from the Multicast Associate Operation CSR

- Verify that destination ID 0xFF03 is not associated with multicast mask 3 for port 2
  - Write the value 0x0000_0280 to the Multicast Associate Operation CSR
  - Read the value 0x0000_0280 from the Multicast Associate Operation CSR
- Verify that destination ID 0xFF03 is associated with multicast mask 3 for port 3
  - Write the value 0x0000_0380 to the Multicast Associate Operation CSR
  - Read the value 0x0000_0381 from the Multicast Associate Operation CSR
- Verify that destination ID 0xFF03 is not associated with multicast mask 3 for port 4
  - Write the value 0x0000_0480 to the Multicast Associate Operation CSR
  - Read the value 0x0000_0480 from the Multicast Associate Operation CSR
- Verify that destination ID 0xFF03 is not associated with multicast mask 3 for port 5
  - Write the value 0x0000_0580 to the Multicast Associate Operation CSR
  - Read the value 0x0000_0580 from the Multicast Associate Operation CSR
- Verify that destination ID 0xFF03 is not associated with multicast mask 3 for port 6
  - Write the value 0x0000_0680 to the Multicast Associate Operation CSR
  - Read the value 0x0000_0680 from the Multicast Associate Operation CSR
- Verify that destination ID 0xFF03 is not associated with multicast mask 3 for port 7
  - Write the value 0x0000_0780 to the Multicast Associate Operation CSR
  - Read the value 0x0000_0780 from the Multicast Associate Operation CSR

Blank page

Annex A End Point Considerations (Informative)

A.1 Introduction

This appendix provides implementation considerations for end points that are intended to be used in a multicast RapidIO system.

A.2 Multicast Destination ID

The transport layer requirement that processing elements be capable of accepting requests regardless of destinationID means that all end points are capable of accepting multicast traffic for all possible multicast destinationIDs. The destinationID of a multicast request may be checked by implementation-specific means.

A.3 End Point Multicast Channels

It may be valuable for an end point to have support for one or more multicast channels. Multicast channels are address ranges in RapidIO address space for which an end point may accept a multicast packet and possibly translate the RapidIO write address to another local address region. This is necessary if the recipient of a multicast transaction does not have valid address space at the address received. The size and quantity of multicast channels depend on the requirements of the application. It may also be necessary to link multicast channels to particular multicast groups.

A multicast channel valid bit can be implemented to control whether an address out-of-range error occurs for a received address which falls inside a multicast channel address range. A multicast channel enable bit can control whether an end point silently ignores the packet when an address is received which falls inside the channel address range. The enable bit allows software finer control over which end points for a particular multicast ID will actually process the multicast write without modifying switch settings in the fabric.

Blank page

Annex B Multicast Applications (Informative)

B.1 Introduction

In a multi-switch RapidIO fabric, each switch which supports multicast in the fabric will have it’s own set of multicast masks. The particular multicast mask in each switch device associated with a multicast group is very likely to have a different egress port pattern enabled, depending upon where that switch is in the switch fabric topology.

As an example, refer to the following system, where data streams entering switch A1 need to be sent to a set of destinations. There are several possible approaches to implementing this system. The first example is based on the number of different multicast groups that must be supported. Destinations are linked to a destination ID which in turn is associated with static multicast mask values. In the second example, a specific multicast mask in each switch is associated with each possible destination. The destinations are linked statically to destination IDs.

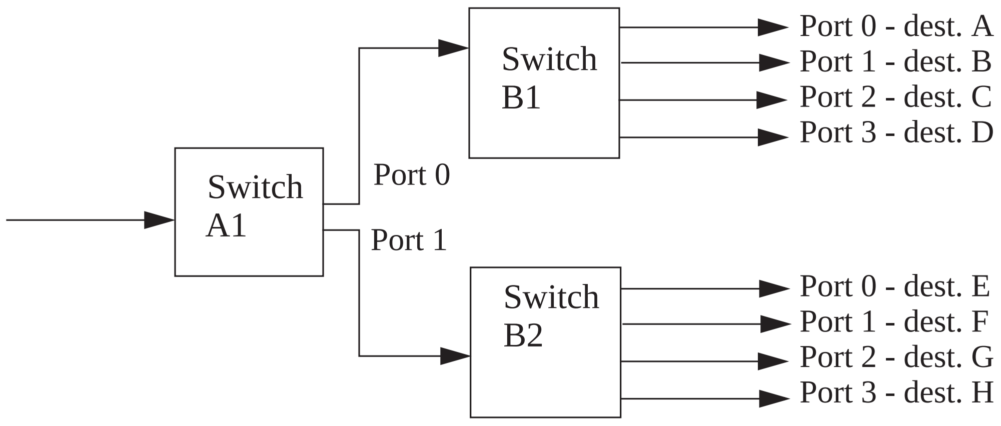


Figure 4-1. Example System using Multicast

B.2 Example 1 - Static Multicast Masks

If there are 256 combinations of destinations to receive a data stream, multicast requires 256 multicast groups, associated with 256 destination IDs. This means that an 8 bit destination ID could be used, but then there would be no destination IDs left over for control traffic in the system. As a result, this example assumes that the system needs to use 16 bit destination IDs in order to support multicast.

It is possible to use the least significant 4 bits of the 16 bit destination ID to identify which ports in Switch B1 need to be multicast to, and the next most significant 4 bits for the ports on B2. Arbitrarily selecting the value of 0x04 for the upper byte of the destination ID, then all multicast destination IDs have a format of 0x04XY, where X selects the ports in switch B2 and Y selects the ports in switch B1.

Switch A1 therefore needs two multicast masks as shown in Table 4-1.

Table 4-1. Multicast Masks for Switch A1

Multicast Mask Index	Egress Ports	Description
0	None	Associated with destination ID 0x0400, indicating that no destination is to receive this data stream. Packets multicast with destination ID of 0x0400 are dropped without notification.
1	Port 0 and port 1	Associated with destination IDs 0x04XY, where both X and Y is not 0. These represent all destination IDs which need only be multicast to both Switch B1 and switch B2.

Destination IDs of the form 0x040Y, where Y is non-zero, or 0x04X0, where X is non zero, do not have to be replicated. They can be routed directly to either port 0 (for 0x040Y) or port 1 (0x04X0) using the standard switch routing tables since there is only a single egress port.

Because Multicast Mask 1 must have 223 ((256 total) - (16 for X) - (16 for Y) - (1 for none)) destination IDs associated with it, the Switch Multicast Information CAR MaxDestIDAssociations field must contain a value of at least 222. In this particular case, the easiest internal implementation for the selection of packets to be multicast may be the use of a non-existent port in the routing table. For example, since Switch A1 has three ports, make use of a non-existent port value in the routing table to signify that the packet is subject to multicast.

Switches B1 and B2 must have 16 multicast masks, each associated with a particular combination of their egress ports 0 through 3. Each multicast mask may have 16 destination IDs associated with it, so the Switch Multicast Information CAR MaxDestIDAssociations field must contain a value of at least 15.

Table 4-2 describes which destination IDs must be associated with each multicast group for Switches B1. Note that for index 0, if the routing tables in Switch A1 are set up correctly, no packets with those multicast groups should reach switch B1.

Table 4-2. Multicast Masks for Switch B1

Multicast Mask Index	Egress Ports	Description
0	None	Associated with the following destination IDs: 0x0400 0x0410 0x0420 ... 0x04E0 0x04F0
1	Port 0	Associated with the following destination IDs: 0x0401 0x0411 0x0421 ... 0x04E1 0x04F1
2	Port 1	Associated with the following destination IDs: 0x0402 0x0412 0x0422 ... 0x04E2 0x04F2
3	Ports 1and 0	Associated with the following destination IDs: 0x0403 0x0413 ... 0x04E3 0x04F3
4	Port 2	Associated with the following destination IDs: 0x0404 0x0414 0x0424 ... 0x04E4 0x04F4
5	Ports2 and 0	Associated with the following destination IDs: 0x0405 0x0415 0x0425 ... 0x04E5 0x04F5
6	Ports2 and 1	Associated with the following destination IDs: 0x0406 0x0416 0x0426 ... 0x04E6 0x04F6

Table 4-2. Multicast Masks for Switch B1

Multicast Mask Index	Egress Ports	Description
7	Ports 2, 1 and 0	Associated with the following destination IDs: 0x0407 0x0417 0x0427 ... 0x04E7 0x04F7
8	Port 3	Associated with the following destination IDs: 0x0408 0x0418 0x0428 ... 0x04E8 0x04F8
9	Ports 3 and 0	Associated with the following destination IDs: 0x0409 0x0419 0x0429 ... 0x04E9 0x04F9
10	Ports 3 and 1	Associated with the following destination IDs: 0x040A 0x041A 0x042A ... 0x04EA 0x04FA
11	Ports 3, 1 and 0	Associated with the following destination IDs: 0x040B 0x041B 0x042B ... 0x04EB 0x04FB
12	Ports3 and2	Associated with the following destination IDs: 0x040C 0x041C 0x042C ... 0x04EC 0x04FC
13	Ports 3, 2 and 0	Associated with the following destination IDs: 0x040D 0x041D 0x042D ... 0x04ED 0x04FD

Table 4-2. Multicast Masks for Switch B1

Multicast Mask Index	Egress Ports	Description
14	Ports 3, 2 and 1	Associated with the following destination IDs: 0x040E 0x041E 0x042E ... 0x04EE 0x04FE
15	Ports 3, 2, 1 and 0	Associated with the following destination IDs: 0x040F 0x041F 0x042F ... 0x04EF 0x04FF

Table 4-2 describes which destination IDs must be associated with each multicast group for Switches B1. Note that for index 0, if the routing tables in Switch A1 are set up correctly, no packets with those multicast groups should reach switch B2.

Table 4-3. Multicast Masks for Switch B2

Multicast Mask Index	Egress Ports	Description
0	None	Associated with the following destination IDs: 0x0400 0x0401 0x0402 ... 0x040E 0x040F
1	Port 0	Associated with the following destination IDs: 0x0410 0x0411 0x0412 ... 0x041E 0x041F
2	Port 1	Associated with the following destination IDs: 0x0420 0x0421 0x0422 ... 0x042E 0x042F
3	Ports 1and 0	Associated with the following destination IDs: 0x0430 0x0431 0x0432 ... 0x043E 0x043F

Table 4-3. Multicast Masks for Switch B2

Multicast Mask Index	Egress Ports	Description
4	Port 2	Associated with the following destination IDs: 0x0440 0x0441 0x0442 ... 0x044E 0x044F
5	Ports2 and 0	Associated with the following destination IDs: 0x0450 0x0451 0x0452 ... 0x045E 0x045F
6	Ports2 and 1	Associated with the following destination IDs: 0x0460 0x0461 0x0462 ... 0x046E 0x046F
7	Ports 2, 1 and 0	Associated with the following destination IDs: 0x0470 0x0471 0x0472 ... 0x047E 0x047F
8	Port 3	Associated with the following destination IDs: 0x0480 0x0481 0x0482 ... 0x048E 0x048F
9	Ports 3 and 0	Associated with the following destination IDs: 0x0490 0x0491 0x0492 ... 0x049E 0x049F
10	Ports 3 and 1	Associated with the following destination IDs: 0x04A0 0x04A1 0x04A2 ... 0x04AE 0x04AF

Table 4-3. Multicast Masks for Switch B2

Multicast Mask Index	Egress Ports	Description
11	Ports 3, 1 and 0	Associated with the following destination IDs: 0x04B0 0x04B1 0x04B2 ... 0x04BE 0x04BF
12	Ports3 and2	Associated with the following destination IDs: 0x04C0 0x04C1 0x04C2 ... 0x04CE 0x04CF
13	Ports 3, 2 and 0	Associated with the following destination IDs: 0x04D0 0x04D1 0x04D2 ... 0x04DE 0x04DF
14	Ports 3, 2 and 1	Associated with the following destination IDs: 0x04E0 0x04E1 0x04E2 ... 0x04EE 0x04EF
15	Ports 3, 2, 1 and 0	Associated with the following destination IDs: 0x04F0 0x04F1 0x04F2 ... 0x04FE 0x04FF

It is up to the application whether either of the switch routing tables should be used for the destination IDs associated with multicast masks 1, 2, 4, and 8, as packets for these destination IDs do not have to be replicated.

Configuring each of the 16 multicast masks for switches B1 and B2 should require a maximum of 2 writes to the Multicast Mask Load CSR. Multicast masks with one or two ports require a number of register writes equal to the number of ports. Multicast masks with three egress ports to be selected should add all of the ports and then remove the port which doesn’t belong in the multicast mask, thus requiring a maximum of two register writes. The multicast mask with all ports selected requires 1 register write. Thus, to configure all 16 of the multicast masks requires a maximum of (0 + (5*1) + (10*2))=25 register write operations.

For the destination ID to multicast mask association operations for Switch B1, it would make sense to implement block association operations since this would greatly reduce the amount of effort required to associate destination IDs with multicast masks. This feature makes possible in this example to associate a sequential block of 16 destination IDs with the 16 multicast masks with only 32 register writes. Refer to Table 4.4.3, “Using Block Association,” on page 40 for details of the pair of writes required for each block of 16 destination IDs.

For the destination ID to multicast mask association operations for Switch B2 there is no pattern that leverages the programming model to speed the association of destination IDs to multicast masks. In Switch B2, it would make sense to use the regular switch routing tables rather than a multicast mask for the destination IDs associated with multicast masks 1, 2, 4 and 8 in order to minimize the number of writes required. The remaining 12 multicast groups each require 32 register write operations to complete their associations with the appropriate destination IDs, for a total of 384 writes. Designers who prefer speed of initialization over reliability may reduce this to 352 register writes by ignoring the destination IDs associated with multicast mask 0.

For switch B2, it may make sense in some systems to implement application specific configuration registers to reduce the number of operations required for configuration.

There can be significant limitations to using static multicast masks. Assume, of the 8 destinations, destinations A, B, C, D, and E are receiving one data stream using destination ID 0x041F, and destinations F, G, and H are receiving a second data stream using destination ID 0x0420.

If destination E switches wishes to change to the second data stream, two things must happen. The destination ID for the first data stream must change from 0x041F to 0x040F in order to have the proper multicast mask for switches B1 and B2, and the destination ID for the second data stream must correspondingly change from 0x0420 to 0x0430.

Because the destination IDs have changed, the switches are now allowed to reorder packets sent to destination IDs 0x041F, 0x040F, 0x0420 and 0x0430, which may change the behavior of the system in unexpected or undesirable ways.

Another issue with static multicast masks is that the latency difference for a data stream between different destinations depends upon whether the data stream is routed using the regular switch routing table or multicast through a particular switch. The different destinations will see different performance characteristics.

These characteristics could have undesirable side effects for latency and jitter sensitive applications like Voice over Internet Protocol (VoIP).

B.3 Example 2 - Linking Multicast Masks to Destination IDs

As an alternative implementation, again suppose that there are 256 possible destinations which need to be multicast, numbered 0 through 255. Each destination has a number of data streams it can receive, up to 256, which is always associated with a 16 bit destination ID of the form 0x04<destination stream>. This requires 256 multicast masks in switches A1, B1, and B2.

When a destination changes the data stream it wants to receive, the multicast masks for that data stream need to be changed. First, the multicast mask in each switch associated with the stream currently being received needs to be modified to stop multicasting to this destination. Next, the multicast mask for the new data stream needs to be modified in each switch to enable multicast to that destination.

Depending on system requirements, there are many ways to implement the multicast capabilities in this system. For example, switch A1 could always multicast all data streams to both switch B1 and switch B2. In this case, switch A1 would require 1 multicast mask that could have all 256 destination IDs associated with it. Switch B1 and B2 may receive a lot of undesired traffic in this case.

Initial programming of the multicast masks is not a requirement as with example 1. No ports should be selected in any mask after reset. Multicast masks will be modified during system operation as destinations request to receive a particular data stream. Removing the data streams from one multicast mask and adding a data stream to a multicast mask can be performed in two register writes for each switch.

The destination ID associations with multicast masks can be done far more effectively in this example if the switch devices support block associate operations. Refer to Section 3.2.3, “Switch Multicast Information CAR”, and the programming examples in Section 4.4, “Configuring Associations” for more information and examples.

Blank page

Glossary of Terms and Abbreviations

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

A

Associate, Association. A defined relationship between a destination ID and a group of end point devices, or, in a switch, a defined relationship between a destination ID and a multicast mask.

M

Multicast. The concept of sending a single message to multiple destinations in a system.

Multicast group. The group of end point devices in a system that is the target of a multicast operation.

Multicast mask. The group of egress ports in a switch that are the targets of a replicated multicast packet.

Blank page

RapidIO™ Interconnect Specification Part 12: Virtual Output Queueing Extensions Specification

3.2, 1/2016

Revision History

Revision	Description	Date
2.0	First public release	03/06/2008
2.1	No technical changes	07/09/2009
2.2	No technical changes	05/05/2011
3.0	Changed RTA contact information. Technical changes: Added 64b/67b encoding of VoQ Backpressure control symbols. Redefined Control Symbol 48 VoQ Backpressure control symbol format. Allowed the number of bits allocated to port group and port status to be programmable. Added register fields to support communication of capabilities and control of the number of port group bits in each VoQ backpressure control symbol. Note that VoQ Backpressure as defined in this specification is not backwards compatible with previous revisions of the specification.	10/11/2013
3.1	No technical changes.	09/18/2014
3.2	No technical changes.	01/28/2016

Table of Contents

Chapter 1 Introduction

1.1 Problem Illustration 9
1.2 Terminology 10
1.3 Conventions 10

Chapter 2 Overview

2.1 Congestion Message 13
2.2 Traffic Staging 14
2.3 Adding Device Independence 15
2.4 Relationship With Virtual Channels 16
2.5 Additional Queueing Considerations 16

Chapter 3 Control Symbol Format

3.1 Stype2 Control Symbol 48 19
3.2 Control Symbol 64 VoQ Backpressure 20
3.3 VoQ Backpressure per VC 22

Chapter 4 Rules

4.1 Implementation Rules 23
4.2 Rules for Generating Backpressure Control Symbols 23
4.3 Rules for Interpreting Backpressure Control Symbols 24

Chapter 5 Register Definitions

5.1 VoQ Backpressure Extended Features Block 25
5.1.1 Register Map 25
5.1.2 VoQ Backpressure Control Block Registers 26
5.1.2.1 LP-Serial VC Register Block Header 26
5.1.2.2 Port n VoQ Control Status Register 27

Table of Contents

Blank page

List of Figures

1-1 Basic Head-of-Line Blocking 9
1-2 Effective Backpressure 10
2-1 Congestion Message Information 13
2-2 Adding Egress Staging....................14
2-3 Mapping Staging Queues....................15
2-4 Associating a VC with VoQ Backpressure....................16
3-1 Control Symbol 48 Format 19
3-2 Stype2 Field Format....................19
3-3 Control Symbol 64 Format 20
3-4 Control Symbol 64 VoQ Backpressure Parameter 0 and 1 Usage....................20
3-5 Control Symbol 48 Associating a VC with VoQ Backpressure 22

List of Figures

Blank page

List of Tables

3-1 Stype2 Field Format....................19
3-2 Control Symbol 48 Example Port Status Bit Assignment....................20
3-3 VoQ Backpressure Control Symbol VC_IND Field Definition....................21
3-4 Control Symbol 64 Example Port Status Bit Assignment....................21
5-1 VoQ Register Block....................25
5-2 Bit Settings for LP-Serial Register Block Header....................26
5-3 Port n VoQ Backpressure CSR....................27
5-4 Port Status Control....................29

List of Tables

Blank page

Chapter 1 Introduction

1.1 Problem Illustration

In the basic switch model shown below, head-of-line blocking occurs when a packet for port 3 cannot be transmitted on the link because of congestion in port 2. The link effectively stalls causing the congestion in port 2 to spread to traffic on other ports. The backpressure method described here helps alleviate congestion spreading caused by transient blockages of queueing structures.

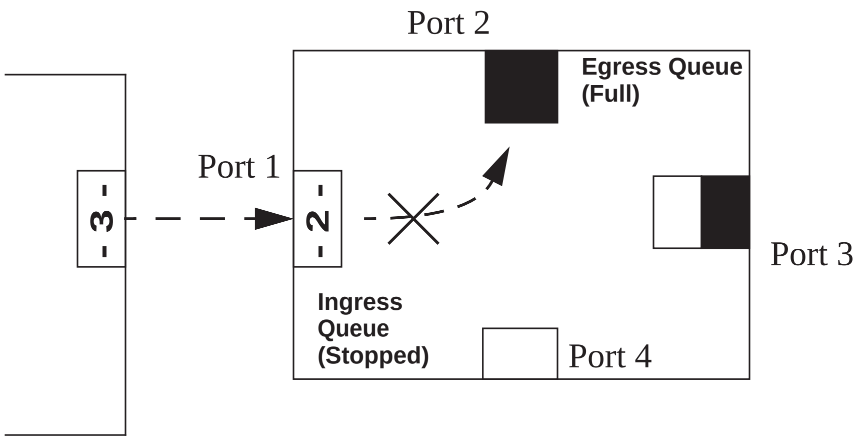


Figure 1-1. Basic Head-of-Line Blocking

The example is simplistic, but any queueing mechanism can become congested, head-of-line block, and stall the link to the upstream device. The VoQ Backpressure Process defines a *congestion message* that informs the upstream port about the congestion, allowing traffic to be sidelined (in a virtual output queue) in favor of traffic with a clear path ahead.

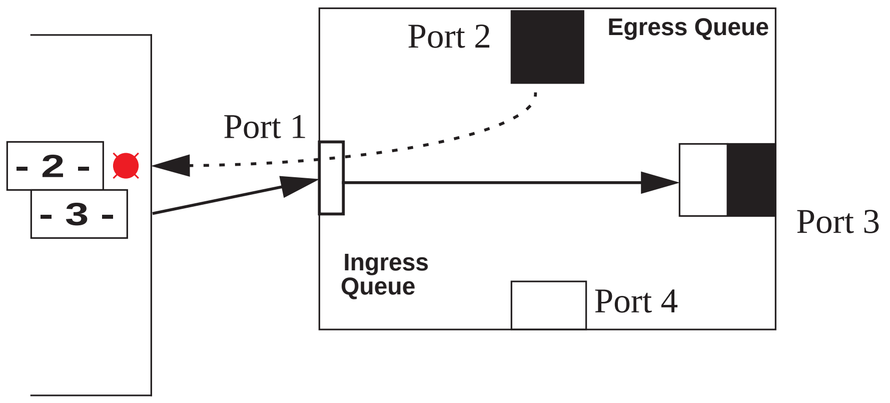


Figure 1-2. Effective Backpressure

Effective backpressure is achieved when the following elements exist:

a) The congested device can communicate congestion information to upstream devices.

b) The upstream device can segregate traffic and allow traffic to re-order based on that congestion information.

These two properties are essential. To keep the operation at the physical layer, the method described uses port identification for both the staging of traffic and the congestion status. Implementation of this specification is optional.

1.2 Terminology

Upstream Device - A device ahead of another device in the traffic flow. The upstream device is the recipient of the backpressure messages.

Downstream Device - The device later in the traffic flow. The downstream device is the originator of backpressure messages.

Port - a port is a local value associated with a specific physical interface. Every device with more than 1 port is responsible for mapping the destination ID to a local port.

Congestion Message - A bit, or group of bits indicating the congestion status of one or more ports.

Backpressure Symbol - A specific field in a RapidIO control symbol that contains the congestion message.

1.3 Conventions

All fields and message formats are described using big endian format.

||	Concatenation, used to indicate that two fields are physically associated as consecutive bits

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.
x	This value is a don’t care.
<variable>	Identifies a logical variable that may be a specific field of a register or packet or data structure.

Blank page

Chapter 2 Overview

The purpose of this backpressure method is to maintain system performance during temporary congestion caused when statistical peaks in traffic flow oversubscribe the ability of a port and its associated buffering to handle the peak load. The backpressure avoids blocking of crossing traffic that competes for common resources. As such, the scope of the message is limited to a device and its immediately adjacent upstream devices. The system may be designed such that sustained congestion will cause a cascade of backpressure messages, but the ability to avoid degradation of performance drops as the radius of affected devices increases. RapidIO has other flow control methods to manage more systemic traffic impediments. Implementation of this specification is optional.

2.1 Congestion Message

Two key considerations for the message format described here are the efficiency and latency of the message, and independence for the two devices involved in the exchange from implementation differences. The congestion message uses a packed format to convey the status of multiple ports in a single message. The contents of the congestion message are shown in Figure 2-1.

VC_IND	Port Status	Port Group

Figure 2-1. Congestion Message Information

The VC_IND indicates the virtual channel (VC0–8) that the congestion message applies to. Port Status is a bit vector that indicates the congestion status of ports, where a port is congested if its status is 1 and uncongested if its status is 0. The Port Group field identifies the subset of ports whose port status is found in this congestion message. The size of the Port Group field is programmable to support sending more or less Port Status in each message, and to support fewer or more ports on a device. The first port in the Port Status vector is the Port Group value multiplied by the port group size. The lowest numbered port in the Port Status vector occupies the least significant bit in the Port Status field. Smaller devices may be able to communicate port status in one or two messages. A congestion message is typically transmitted when at least one of the ports' status changes. The congestion message is embedded in a field in the RapidIO control symbol. The symbol containing this message is defined as the backpressure symbol.

2.2 Traffic Staging

For the congestion message to be useful, the upstream device must segregate or stage traffic prior to committing it to the RapidIO link to the downstream device, or any critical resource (like a buffer) that could block other traffic. To stage the packets, the upstream device must have knowledge about the routing configuration of the downstream device. A typical RapidIO switch will lookup incoming traffic and switch it to a port based on its destination ID. To support this backpressure method, that lookup must produce the egress port for the current device as well as the egress port for the next device. It is straightforward to align the routing tables, but it does require additional entries.

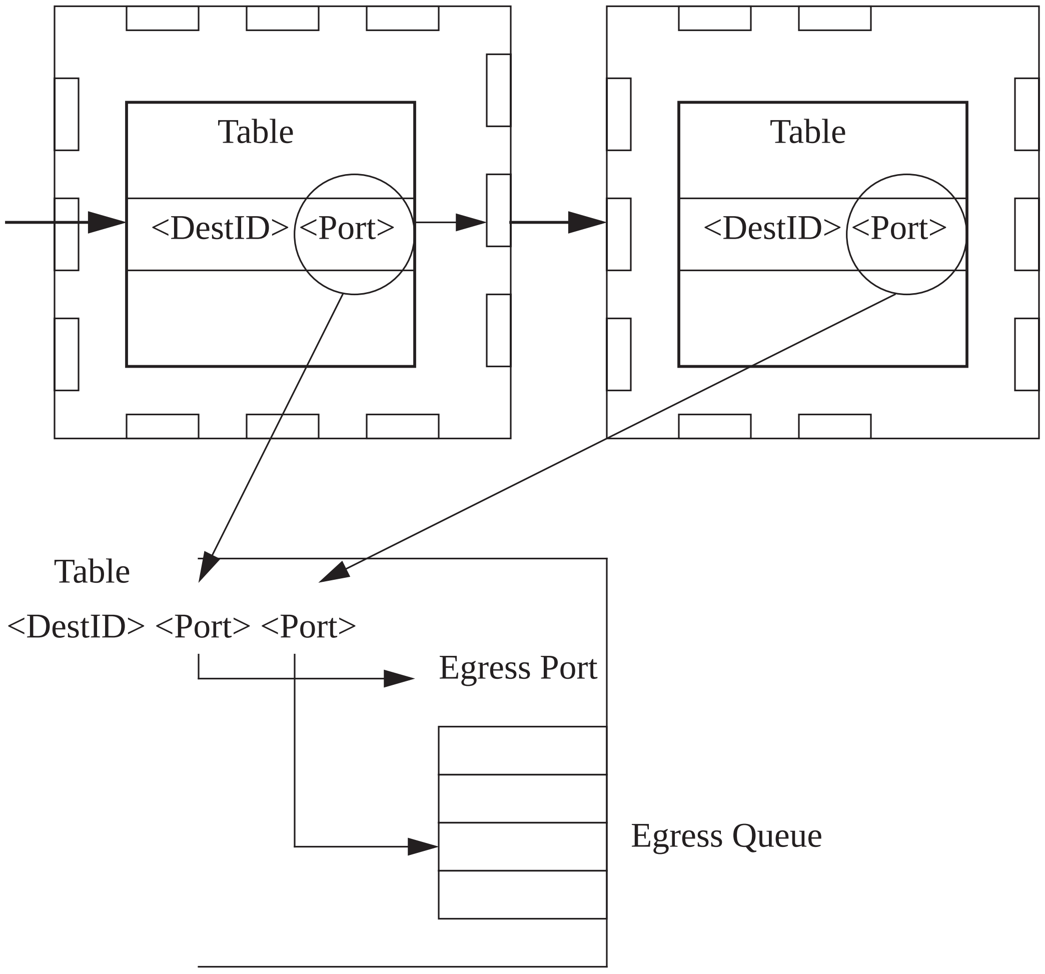


Figure 2-2. Adding Egress Staging

In the figure above, output staging is created by adding a second parameter to the routing table for the next hop port value. That value is the same value that has to be put in the downstream device’s routing table. In the upstream device it is used to identify what queue to stage the traffic in, and what queue to act on when a congestion message is received. The value is specified in an implementation-dependant fashion.

For end points, there is normally no routing table. To use this form of backpressure, the end point would need a method to associate a destination ID with the downstream egress port.

2.3 Adding Device Independence

In the basic structure above, the traffic is staged in a queue that corresponds to a port in the downstream device. A difficulty arises to match the number of queues to the number of possible ports that might exist in the next device. It is inconsistent with RapidIO's goal of allowing devices to implement cost and complexity as needed by their market to require every device to have hundreds of queues to support a maximum sized device, so an additional abstraction is required.

A device may combine traffic into fewer queues by reverse mapping the incoming port congestion message. In the figure below, the egress port supports 4 staging queues. The downstream device has 16 ports. So traffic for several ports are staged in a common queue. When a port specific message is received it is reverse mapped to the same queue that the forward lookup used to stage the data.

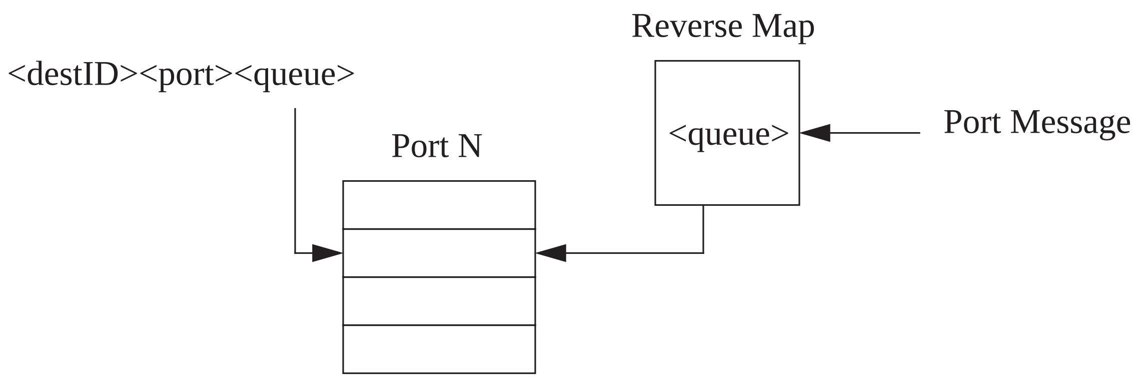


Figure 2-3. Mapping Staging Queues

In this example, as long as the forward/reverse mapping corresponds to the right downstream device's egress port, any implementation can be used, as it is all internal to a single device. This mapping requires that only enough bits for the number of queues be added to the forward table, which can be very large. In the reverse direction, the mapping can be RAM based for maximum flexibility (in the above example requiring a 256 x 2 bit RAM), or a straight decoder. This specification does not prescribe a specific method.

The only other requirement, when mapping multiple ports to a single queue, is that the queue must be shut down when any of those ports are congested. This will reduce the benefits of this backpressure method, but a significant amount of benefit is achieved with just a few queues.

2.4 Relationship With Virtual Channels

The staging method illustrated above uses queues in the output stage of the upstream device. Devices implementing multiple Virtual Channels will have additional queueing structures for the VCs. The staging queues may be before the traffic is sorted into its VC, or each VC may have a set of staging queues. When the output queueing is not tied to the implementation of VCs, the congestion message is not associated with VC operation, and the backpressure symbol can be combined with any valid combinations of RapidIO symbols.

If the output queueing is embedded within the VC structure, the VoQ congestion message can be associated with VC operation. Both message formats are described in the next section. The congestion message may be associated with a specific VC. A CSR bit is provided to enable or disable transmission of VoQ backpressure per VC.

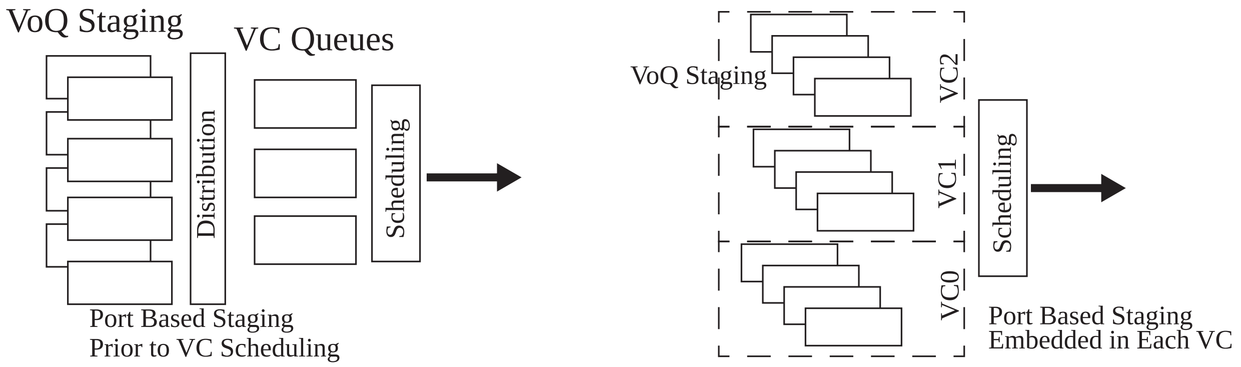


Figure 2-4. Associating a VC with VoQ Backpressure

2.5 Additional Queueing Considerations

If the downstream (originating) device is using queues in its output stage to determine congestion, then congestion messages will have to be sent to all ports that might be sources of incoming traffic.

If the downstream device is using virtual output queues, presorting traffic by egress port at the input, then it has the ability to reflect congestion status on only those ports that are receiving traffic for the congested output. Input queued switches do require N^2 queues (where N is the number of ports).

Devices using some combination of input and output queueing, or shared memory architectures, may make congestion decisions based on whatever resource allocation algorithm is being employed. It is important to consider some of the following boundary conditions:

- If the congestion message is issued with too little room in the port’s egress to account for packets that might be in flight, head-of-line blocking can still occur.

- If very small queueing structures are used, a lot of on/off chatter can occur. This is not necessarily bad as long as the additional utilization of link bandwidth is accounted for.

The generation and application of the congestion message defined in this specification will be highly dependent on the queueing and switch design of the device, and as such, is left to the implementer.

Chapter 3 Control Symbol Format

The VoQ congestion message adds a control symbol to the *RapidIO Interconnect Specification Part 6: LP-Serial Physical Layer Specification*. Refer to that specification for the definitions of control symbols, packet delimiting, and the definitions of the fields not defined here. The VoQ backpressure symbol uses the Control Symbol 48 defined for use on Baud Rate Class 2 links, and the Control Symbol 64 format defined for use on Baud Rate Class 3 links. Baud Rate Class 1 links can be designed to support the Control Symbol 64. For more information, see *RapidIO Interconnect Specification Part 6: LP-Serial Physical Layer Specification*.

3.1 Stype2 Control Symbol 48

The Control Symbol 48 is defined as follows in the *RapidIO Interconnect Specification Part 6: LP-Serial Physical Layer Specification*:

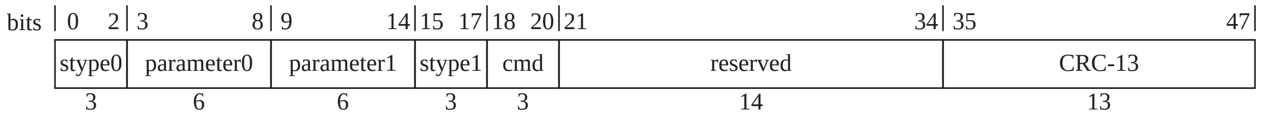


Figure 3-1. Control Symbol 48 Format

The stype2 field uses the 14 reserved bits in Control Symbol 48, and has an operation code (CMD) field and a parameter field. The VoQ backpressure symbol defines the following usage for the Stype2 field:

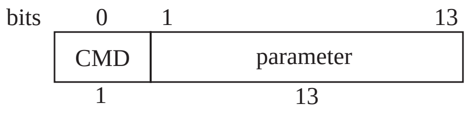


Figure 3-2. Stype2 Field Format

Table 3-1 shows the Stype2 field format definitions.

Table 3-1. Stype2 Field Format

Function	**CMD** (Bit 0)	**Parameter** (Bits 1–13)	
Reserved	0b0	Reserved	
VoQ Backpressure	0b1	Port Status Bits	Port Group

The combined size of the Port Group and Port Status Bits is 13 bits. The size of the

The Port Group field is determined by the Port n VoQ Control Status Register’s TX Port Group Size and RX Port Group Size fields. For example, suppose that the RX Port Group Size field value is 0b001, indicating that the Port Group field size is 1 bit. The Port Status bits would be assigned as shown in Table 3-2.

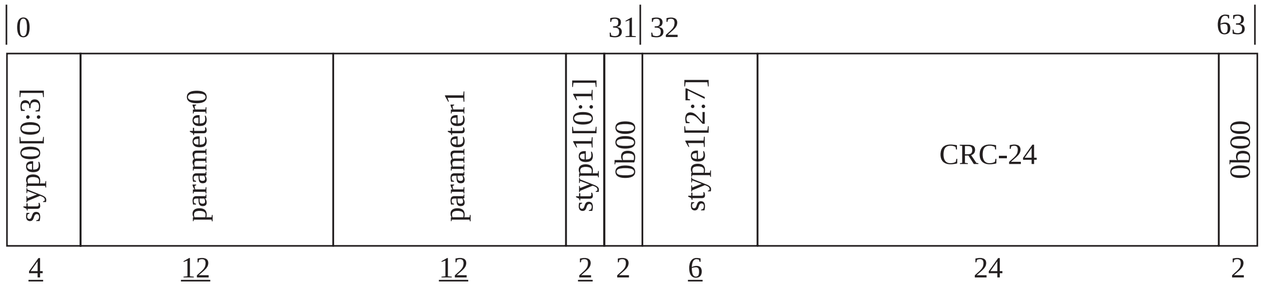
Table 3-2. Control Symbol 48 Example Port Status Bit Assignment

Port Group Value	**Port Number for Port Status Bit**											
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
0x0	11	10	9	8	7	6	5	4	3	2	1	0
0x1	23	22	21	20	19	18	17	16	15	14	13	12

3.2 Control Symbol 64 VoQ Backpressure

The Control Symbol 64 is defined as follows in the *RapidIO Interconnect Specification Part 6: LP-Serial Physical Layer Specification*:

Figure 3-3. Control Symbol 64 Format



VoQ Backpressure messages are encoded as Control Symbol 64 Stype 0 control symbols using an stype0[0:3] field value of 0b1101. The parameter0 and parameter1 fields are combined into one contiguous format that includes the VC_IND, Port Status, and Port Group fields, as shown in Figure 3-4.

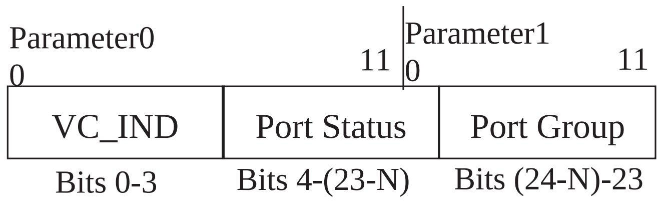


Figure 3-4. Control Symbol 64 VoQ Backpressure Parameter 0 and 1 Usage

The VC_IND value is defined as shown in Table 3-3. Reception of a reserved value in the VC_IND field shall cause the receiving endpoint to ignore the VoQ Backpressure control symbol without error.

Table 3-3. VoQ Backpressure Control Symbol VC_IND Field Definition

VC_IND Value	VC	Comments
0b0000	VC1	This encoding scheme was chosen to allow the Control Symbol 64 VC_IND value to be an extension of the Control Symbol 48 VC_Status control symbol VCID encoding.
0b0001	VC2	
0b0010	VC3	
0b0011	VC4	
0b0100	VC5	
0b0101	VC6	
0b0110	VC7	
0b0111	VC8	
0b1000	VC0	
0b1001–0b1110	Reserved	
0b11111	All VCs	This value shall be used when VoQ backpressure per VC is disabled.

The combined size of the Port Group and Port Status Bits is 20 bits. The size of the Port Group field is determined by the Port n VoQ Control Status Register’s TX Port Group Size and RX Port Group Size fields. For example, suppose that the RX Port Group Size field value is 0b100, indicating that the Port Group field size is 4 bits. The Port Status field is 16 bits long. In this case, the Port Status bits shall be assigned to ports as shown in Table 3-4.

Table 3-4. Control Symbol 64 Example Port Status Bit Assignment

Port Group Value	Port Number for Port Status Bit															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
2	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
3	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
4	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
5	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
6	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
7	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112
8	143	142	141	140	139	138	137	136	135	134	133	132	131	130	129	128
9	159	158	157	156	155	154	153	152	151	150	149	148	147	146	145	144
10	175	174	173	172	171	170	169	168	167	166	165	164	163	162	161	160
11	191	190	189	188	187	186	185	184	183	182	181	180	179	178	177	176
12	207	206	205	204	203	202	201	200	199	198	197	196	195	194	193	192
13	223	222	221	220	219	218	217	216	215	214	213	212	211	210	209	208

Table 3-4. Control Symbol 64 Example Port Status Bit Assignment

Port Group Value	Port Number for Port Status Bit															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
14	239	238	237	236	235	234	233	232	231	230	229	228	227	226	225	224
15	255	254	253	252	251	250	249	248	247	246	245	244	243	242	241	240

3.3 VoQ Backpressure per VC

Note that the VC_IND field discussed in Section 2.1 is not defined in the Control Symbol 48 VoQ backpressure message. By default, VoQ backpressure Port Status applies to all VCs on a port.

The association of VoQ backpressure with a specific port within a VC is known as VoQ Backpressure per VC. When Control Symbol 48 is used, VoQ Backpressure per VC is accomplished by transmitting a VC_Status symbol or Status symbol with every VoQ backpressure symbol. The VC_Status VCID, or VC0 for the Status symbol, identifies the specific VC that is congested for that port.

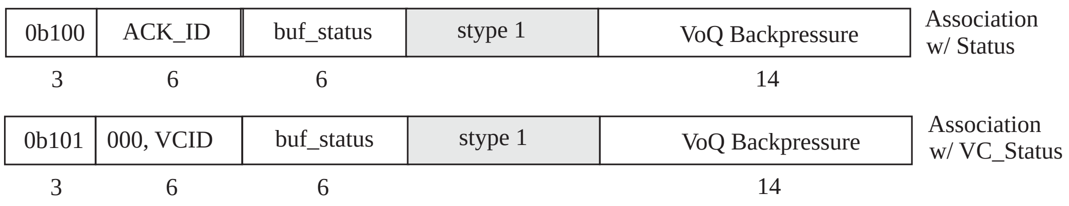


Figure 3-5. Control Symbol 48 Associating a VC with VoQ Backpressure

When Control Symbol 64 is used, VoQ Backpressure per VC is accomplished by using VC_IND field values other than 0b11111 (all VCs).

VoQ Backpressure per VC only works if both upstream and downstream devices use port staging within VC structures. If either device does not have a corresponding capability, VoQ Backpressure per VC Transmission shall be disabled in the CSR. A device with queueing within VCs shall flow control corresponding port queues in all VCs when VoQ Backpressure per VC Transmission is disabled.

When VoQ Backpressure per VC Transmission is set in the Port n VoQ Control Status Register and the stype2 field contains a VoQ Backpressure Symbol, the upstream device shall flow control just the per-port queue within the corresponding VC. When VoQ Backpressure per VC is enabled, the VoQ backpressure symbol shall be combined with a VC_Status or Status symbol.

When the VoQ Backpressure per VC Transmission field is clear, the VoQ backpressure symbol shall apply to all VCs, even if the VoQ backpressure message is combined with a VC_Status or Status control symbol.

Chapter 4 Rules

4.1 Implementation Rules

a) Implementation of VoQ backpressure is entirely optional.

b) Devices may implement VoQ backpressure on a port by port basis.

c) Devices may support only the generation of backpressure messages without the ability to honor messages, or vice versa.

d) Devices using VoQ backpressure shall support Control Symbol 48 and/or Control Symbol 64.

4.2 Rules for Generating Backpressure Control Symbols

a) The VoQ backpressure symbol shall only be transmitted to an upstream device if generation is enabled for a given port. If a congested port requests a symbol be sent to all upstream devices, only ports enabled for this feature shall actually transmit the symbol.

b) VoQ backpressure symbols may indicate congestion on any VC when VoQ Backpressure per VC is disabled.

c) VoQ backpressure symbols shall indicate congestion on a specific VC when VoQ Backpressure per VC is enabled.

d) Ports shall be grouped in order. Ports numbered 0 through N-1 shall occupy Port Group 0 in the backpressure message, ports numbered N to 2N-1 shall occupy port group 1, and so on, where N is controlled by the Port n VoQ Control Status Register's TX Port Group Size field and the number of bits available in the control symbol format.

e) The backpressure symbol shall be generated any time the status of at least one of the ports in the group changes. It is up to the implementer to define what constitutes a status change.

f) The backpressure symbol may be generated at arbitrary intervals, based on a timer. The timer may be the same timer used for VC_status, or it may be a separate timer. Use of a timer is implementation specific.

g) The backpressure symbol may be generated after link recovery to avoid orphaned congestion states.

4.3 Rules for Interpreting Backpressure Control Symbols

a) Devices shall have a mechanism to associate traffic with the downstream device’s egress port.

b) Devices shall have a mechanism to associate the incoming congestion message with traffic destined for the indicated port. All traffic identified for that port shall not be committed to a critical resource when that port is identified as congested, allowing other traffic to pass. A critical resource is any resource that can block other traffic like a link or a buffer in a VC.

c) Traffic that is still eligible for transmission is still subject to existing RapidIO ordering rules.

d) Traffic that has been segregated shall be re-introduced to the data stream with the same ordering requirements that existed when it was segregated.

e) Devices may deliberately co-mingle traffic (traffic destined to different ports) to simplify implementations. When such co-mingling loses the ability to further discriminate among the ports, any congestion for any of the ports associated with the co-mingled traffic results in all that traffic being stopped. Co-mingled traffic may only be committed to the link if all ports represented by the traffic are not congested.

Chapter 5 Register Definitions

5.1 VoQ Backpressure Extended Features Block

This section describes the registers for all RapidIO LP-Serial devices supporting virtual channels. This Extended Features register block is assigned Extended Features block ID=0x000B.

5.1.1 Register Map

Table 5-1 shows the VoQ backpressure register map for all RapidIO LP-Serial devices. The Block Offset is the offset relative to the 16-bit Extended Features Pointer (EF_PTR) that points to the beginning of the block.

The address of a byte in the block is calculated by adding the block byte offset to EF_PTR that points to the beginning of the block. This is denoted as [EF_PTR+xx] where xx is the block byte offset in hexadecimal.

Table 5-1. VoQ Register Block

Block Byte Offset	Register Name
0x0	LP-Serial Port - VoQ Backpressure Register Block Header
0x4–0x1C	Reserved
0x20	Port 0 VoQ Control Register
0x24	Port 1 VoQ Control Register
0x28	Port 2 VoQ Control Register
0x2C	Port 3 VoQ Control Register
0x30–0x418	Port *n* VoQ Control Registers
0x41C	Port 255 VoQ Control Register

5.1.2 VoQ Backpressure Control Block Registers

Multiport devices implementing VoQ backpressure shall implement one register per port, even if the port does not support backpressure. Single port end points implement the port 0 register only

5.1.2.1 LP-Serial VC Register Block Header (Block Offset 0x0)

The LP-Serial VC register block header register contains the EF_PTR to the next extended features block, and the EF_ID that identifies this as the LP-Serial virtual channel register block header.

Table 5-2. Bit Settings for LP-Serial Register Block Header

Bit	Name	Reset Value	Description
0-15	EF_PTR		Hard wired pointer to the next block in the data structure, if one exists
16-31	EF_ID	0x000B	Hard wired Extended Features ID

5.1.2.2 Port *n* VoQ Control Status Register (Block Offset - 0x20 + (4* n))

This register is used by each port to configure VoQ backpressure operation.

Table 5-3. Port n VoQ Backpressure CSR

Bit	Name	Reset Value	Description
0	VoQ Backpressure Symbol Generation Supported	see footnote[1]	0b0 = generation of VoQ backpressure is not supported by this port 0b1 = generation of VoQ backpressure supported (read-only)
1	VoQ Backpressure Symbol Reception Supported	see footnote[1]	0b0 = reception of VoQ backpressure is not supported by this port 0b1 = reception of VoQ backpressure supported (read-only)
2	VoQ Backpressure Per VC Supported	see footnote[1]	0b0 = VoQ backpressure messages shall indicate, and be interpreted to mean, that all VCs on a port are congested. 0b1 = VoQ backpressure messages shall indicate, and shall be interpreted to mean, that a specific VC on a port is congested. (read-only)
3-7	reserved	0b0	
8	Enable VoQ Symbol Generation	0b0	0b0 = No VoQ backpressure symbols will be transmitted 0b1 = VoQ backpressure symbol generation is enabled
9	Enable VoQ Participation	0b0	0b0 = this port's status will not be included in any VoQ symbols transmitted, nor cause symbols to be generated (the port's status will always be reflected as enabled). 0b1 = this port's status shall be reflected in VoQ backpressure symbols and will cause symbols to be generated
10	Port XOFF	0b0	0b0 = Port status will reflect current state of the port. 0b1 = Port status will always reflect congested (= 0b1) This field should be set to 1 if the Port n Status and Control CSR Port Unavailable bit is set for this port.
11	Enable VoQ Backpressure Per VC Transmission	0b0	0b0 = The Port Status in VoQ backpressure messages shall aggregate the congestion status of all VCs 0b1 = The Port Status in VoQ backpressure messages shall communicate the congestion status of individual VCs This field shall be reserved when VoQ Backpressure per VC Supported is 0. When this bit is set for links using Control Symbol 48, VoQ backpressure messages shall only be transmitted with VC_Status and Status control symbols.
12	Port Group Size 0 Supported	0b1	0b0 = A port group size of zero bits is not supported 0b1 = A port group size of zero bits is supported for both transmission and reception
13	Port Group Size 1 Supported	see footnote[1]	0b0 = A port group size of one bit is not supported 0b1 = A port group size of one bit is supported for both transmission and reception
14	Port Group Size 2 Supported	see footnote[1]	0b0 = A port group size of two bits is not supported 0b1 = A port group size of two bits is supported for both transmission and reception

Table 5-3. Port n VoQ Backpressure CSR

Bit	Name	Reset Value	Description
15	Port Group Size 3 Supported	see footnote[1]	0b0 = A port group size of three bits is not supported 0b1 = A port group size of three bits is supported for both transmission and reception
16	Port Group Size 4 Supported	0b1	0b0 = A port group size of four bits is not supported 0b1 = A port group size of four bits is supported for both transmission and reception
17	Port Group Size 5 Supported	see footnote[1]	0b0 = A port group size of five bits is not supported 0b1 = A port group size of five bits is supported for both transmission and reception
18	Port Group Size 6 Supported	see footnote[1]	0b0 = A port group size of six bits is not supported 0b1 = A port group size of six bits is supported for both transmission and reception
19-25	reserved	0x00	
26-28	TX Port Group Size	0x0	Current number of bits devoted to port group for transmitted VoQ Status control symbols, encoded as follows: 0x0 = No bits for port_group, all bits are port status 0x1 = One bit for port_group, remaining bits are port status 0x2 = Two bits for port_group, remaining bits are port_status ... 0x6 = Six bits for port_group, remaining bits are port_status 0x7 = Reserved This field shall be changed only when Enable VoQ Symbol Generation is cleared.
29-31	RX Port Group Size	0x0	Current number of bits devoted to port group for received VoQ Status control symbols, encoded as follows: 0x0 = No bits for port_group, all bits are port status 0x1 = One bit for port_group, remaining bits are port status 0x2 = Two bits for port_group, remaining bits are port_status ... 0x6 = Six bits for port_group, remaining bits are port_status 0x7 = Reserved This field shall be changed only when the link partner’s Enable VoQ Symbol Generation field is cleared.

[1]The reset value is implementation dependent

Symbol Generation by a port must be enabled only when the device at the other end of the link supports reception.

VoQ Backpressure per VC should be enabled only if it is supported by both connected devices. Support is defined as being able to generate and receive VoQ Backpressure per VC messages. Generating and receiving per VC VoQ Backpressure messages requires an underlying queueing structure that can segregate traffic by both VC and port.

Bits 9 and 10 combine as shown in Table 5-4.

Table 5-4. Port Status Control

Bit 9	Bit 10	Status Reflected in VoQ Backpressure Messages
0	0	Port Status is always 0b0 (will not cause symbol to be generated)
0	1	Port Status is always 0b1 (will not cause symbol to be generated)
1	0	Normal operation, state transitions cause symbols to be generated and the status is reflected in the symbol
1	1	Port Status is always 0b1 (will cause a symbol to be generated if changing from normal operation causes a state change).

With bit 9 = 0b0, toggling bit 10 will change the port’s reported state, but will not trigger any new symbols. With bit 9 = 0b1, changing from normal operation to congested or congested to normal operation will cause a symbol to be transmitted only if the state of the port changed.

Note that changing the status of the port does not necessarily imply traffic will change. That depends on the configuration of the upstream device.

RapidIO™ Interconnect Specification Annex 1: Software/System Bring Up Specification

3.2, 1/2016

RapidIO.org

Revision History

Revision	Description	Date
1.0	First release	12/17/2003
1.3	Technical changes: the following errata showings: 04-09-00020.001, 04-09-00023.001 Converted to ISO-friendly templates Revision bumped to align with the rest of the specification stack	02/23/2005
2.0	Technical changes: errata showing 06-02-00001.005	06/14/2007
2.1	Technical changes: Errata showing 08-06-00000.000	07/09/2009
2.2	No technical changes	05/05/2011
3.0	Changed RTA contact information. No technical changes	10/11/2013
3.1	No technical changes.	09/18/2014
3.2	No technical changes.	01/28/2016

Table of Contents

Chapter 1 Overview

1.1 Introduction 7
1.2 Overview 7
1.3 Scope 7
1.4 System Enumeration API 8
1.5 Terminology 8
1.6 Software Conventions 8

Chapter 2 Requirements for System Bring Up

2.1 Introduction 9
2.2 Boot Requirements 9
2.3 Enumeration Completion 10
2.4 Enumeration Time-Out 10
2.5 Function Return Codes 11

Chapter 3 Hardware Abstraction Layer

3.1 Introduction 13
3.2 Device Addressing 13
3.3 HAL Functions 14
3.3.1 Types and Definitions 14
3.3.2 rioGetNumLocalPorts 14
3.3.3 rioConfigurationRead 15
3.3.4 rioConfigurationWrite 16

Chapter 4 Standard Bring Up Functions

4.1 Introduction 17
4.2 Data Structures 17
4.3 Bring Up Functions 18
4.3.1 rioInitLib 18
4.3.2 rioGetFeatures 19
4.3.3 rioGetSwitchPortInfo 20
4.3.4 rioGetExtFeaturesPtr 21
4.3.5 rioGetNextExtFeaturesPtr 22
4.3.6 rioGetSourceOps 23
4.3.7 rioGetDestOps 24
4.3.8 rioGetAddressMode 25
4.3.9 rioGetBaseDeviceId 26
4.3.10 rioSetBaseDeviceId 27

Table of Contents

4.3.11 rioAcquireDeviceLock........ 28
4.3.12 rioReleaseDeviceLock........ 29
4.3.13 rioGetComponentTag........ 30
4.3.14 rioSetComponentTag........ 31
4.3.15 rioGetPortErrStatus........ 32

Chapter 5 Routing-Table Manipulation Functions

5.1 Introduction........ 33
5.2 Routing Table Functions........ 34
5.2.1 rioRouteAddEntry........ 34
5.2.2 rioRouteGetEntry........ 35

Chapter 6 Device Access Routine Interface

6.1 Introduction........ 37
6.2 DAR Packaging........ 37
6.3 Execution Environment........ 37
6.4 Type Definitions........ 38
6.5 DAR Functions........ 39
6.5.1 *rioDar_name*GetFunctionTable........ 39
6.5.2 rioDarInitialize........ 40
6.5.3 rioDarTerminate........ 41
6.5.4 rioDarTestMatch........ 42
6.5.5 rioDarRegister........ 43
6.5.6 rioDarGetMemorySize........ 44
6.5.7 rioDarGetSwitchInfo........ 45
6.5.8 rioDarSetPortRoute........ 46
6.5.9 rioDarGetPortRoute........ 47

Annex A System Bring Up Guidelines (Informative)

A.1 Introduction........ 49
A.2 Overview of the System Bring Up Process........ 49
A.3 System Enumeration Algorithm........ 50
A.3.1 Data Structures, Constants, and Global Variables........ 51
A.3.2 Pseudocode........ 52
A.4 System Bring Up Example........ 56

List of Figures

A-1 Example System..57

List of Figures

Blank page

Chapter 1 Overview

1.1 Introduction

This chapter provides an overview of the *RapidIO Annex 1: Software/System Bring Up Specification* document. This document assumes that the reader is familiar with the RapidIO specifications, conventions, and terminology.

1.2 Overview

The RapidIO Architectural specifications establish a framework that enables a wide variety of implementations. The *RapidIO Part 7: System and Device Inter-operability Specification* provides a standard set of device and system design solutions to support inter-operability. This document builds upon the inter-operability specification to define a standard set of software API functions for use in system bring up.

Each chapter addresses a different bring up topic. This revision of the *RapidIO Annex 1: Software/System Bring Up Specification* document covers the following issues:

Chapter 2, “Requirements for System Bring Up”

Chapter 3, “Hardware Abstraction Layer”

Chapter 4, “Standard Bring Up Functions”

Chapter 5, “Routing-Table Manipulation Functions”

Chapter 6, “Device Access Routine Interface”

Annex A, “System Bring Up Guidelines (Informative)”

1.3 Scope

Although RapidIO networks provide many features and capabilities, there are a few assumptions and restrictions that this specification relies on to simplify the bring up process and narrow the specification scope. These assumptions and restrictions are:

- Only two hosts may simultaneously enumerate a network. Two hosts may be needed on a network for fault tolerance purposes. System integrators must determine which hosts can perform this function.

- Only one host actually completes the network enumeration (this is referred to as the *winning host*). The second host must retreat and wait for the enumeration to complete or, assuming the winning host has failed, for enumeration to timeout. If a timeout occurs, the second host re-enumerates the network.
- After enumeration, other hosts in the system must passively discover the network to gather topology information such as routing tables and memory maps.

1.4 System Enumeration API

System enumeration API functions may be divided into two categories:

- Standard RapidIO functions that use hardware resources defined by the RapidIO specifications. These functions should rely on the support functions provided by the Hardware Abstraction Layer (HAL) to ensure portability between different platforms.
- Device-specific (vendor-specific) functions defined by a device manufacturer that use hardware resources outside of the scope of the RapidIO specifications. The main purpose of these functions is to provide Hardware Abstraction Layer (HAL) support to the standard RapidIO functions.

An important goal of this software API specification is to minimize the number of device-specific functions required for enumeration so that the portability of the API across hardware platforms is maximized.

1.5 Terminology

This document uses terms such as *local port, local configuration registers,* etc. to refer to hardware resources associated with a RapidIO end point device attached to (or combined with) the host processor that performs RapidIO system enumeration and initialization.

1.6 Software Conventions

To describe the software API functions, this document uses syntactic and notational conventions consistent with the C programming language. The conventions for naming functions and variables used by these APIs are outside of scope of this document.

Chapter 2 Requirements for System Bring Up

2.1 Introduction

This section describes basic requirements for system bring up and discovery. An overview of the system bring up process, including a system bring up example, is presented in Annex A, “System Bring Up Guidelines (Informative)”.

2.2 Boot Requirements

The following system state is required for proper system bring up:

After the system is powered on, the state necessary for system enumeration to occur using multiple host processors is automatically initialized as follows (These initial state requirements are specified in the *RapidIO Part 7: System and Device Inter-operability Specification*):

- System devices are initialized with the following Base Device IDs:
    - Non-boot-code and non-host device IDs are set to 0xFF (0xFFFF for 16-bit deviceID systems).
    - Boot code device IDs are set to 0xFE (0x00FE for 16-bit deviceID systems).
    - Host device IDs are set to 0x00 (0x0000 for 16-bit deviceID systems).
- Physical layer link initialization of end points is complete.
- The default routing state of all switches between the boot code device and the host device is set to route all requests for device ID 0xFE (0x00FE for 16-bit deviceID systems) to the appropriate boot code device. All response packets are routed back to the host from the boot code device.
- Any host that participates in discovery must change its destination ID to a unique ID value before starting the system initialization process. This value is used by a device’s Host Base Device ID Lock CSR to ensure only one host can manipulate a device at a time. The allowed ID values for a discovering host are 0x00 (0x0000) and 0x01 (0x0001). A host with an ID of 0x00 (0x0000) has a lower priority than a host with an ID of 0x01 (0x0001). Host devices must be configured to accept maintenance packets with a destination ID of 0xFF (0xFFFF for 16-bit deviceID systems) as well as the unique host ID.

- All host devices have their Master Enable bit (Port General Control CSR) set to 1. Switch devices do not have a Master enable bit.
- All devices will accept requests with any sourceID or destinationID value

2.3 Enumeration Completion

One or two hosts can perform system enumeration in a RapidIO network. If two hosts are present, an algorithm is needed to determine which host has the priority to proceed with enumeration. The host with the higher priority is the *winning host* and the other host is the *losing host*. The enumeration algorithm suggested in Appendix A, "System Bring Up Guidelines (Informative)," on page 49 sets priority based on the value of the power-on device ID.

Enumeration is complete when the winning host releases the lock on the losing host. It is the losing host's responsibility to detect that it has been locked by the winning host and to later detect that the lock has been released by the winning host. The methods used to release locks on nodes other than the host nodes is outside the scope of this document.

2.4 Enumeration Time-Out

As mentioned in the previous section, two hosts can be used to enumerate the RapidIO network. The algorithm in Appendix A assumes the host with the higher power-on host device ID has priority over the other host. Because of this pre-defined priority, only one host (the one with higher priority) can win the enumeration task. In this case, the losing host enters a wait state.

If the winning host fails to enumerate the entire network, the losing host's wait state times out. When this occurs, the losing host attempts to enumerate the network. In an open 8-bit deviceID system, the losing host must wait 15 seconds before timing out and restarting the enumeration task. The length of the time-out period in a closed or a 16-bit deviceID system may differ from that of an open system.

To develop the 15 second time-out value, the following assumptions are made about the network maximal size:

NUMDEV = 256 devices

NUMSWITCHES = 256 switches

NUMFTE = 256 routing table entries per switch

It is assumed that a separate maintenance write packet is required to program each routing table entry for each switch. Since we need to establish a time base for operations, we assume:

CWTime = 100 microseconds per configuration write packet

Now we can estimate that the number of configuration writes it takes to program all of the switch routing table entries is (256 switches)*(256 routing table entries), or;

=> 256*256*CWTIME microsecs =

=> ~6.6 seconds.

Given these rough approximations, a 15 second time-out value is seen as appropriate and conservative for open systems. The chosen value must be such that if a time-out were to occur, it must be guaranteed that failure HAS occurred, and hence choosing a conservative value is necessary.

2.5 Function Return Codes

The following return codes and their constant values are defined for use by the system bring up functions.

```
typedef   unsigned int        STATUS;

#define   RIO_SUCCESS                       0x0     // Success status code
#define   RIO_WARN_INCONSISTENT             0x1     // Used by
                                                    // rioRouteGetEntry—indicates
                                                    // that the routeportno returned is
                                                    // not the same for all ports
#define   RIO_ERR_SLAVE                     0x1001  // Another host has a higher
                                                    // priority
#define   RIO_ERR_INVALID_PARAMETER         0x1002  // One or more input parameters
                                                    // had an invalid value
#define   RIO_ERR_RIO                       0x1003  // The RapidIO fabric returned a
                                                    // Response Packet with ERROR
                                                    // status reported
#define   RIO_ERR_ACCESS                    0x1004  // A device-specific hardware
                                                    // interface was unable to generate
                                                    // a maintenance transaction and
                                                    // reported an error
#define   RIO_ERR_LOCK                      0x1005  // Another host already acquired
                                                    // the specified processor element
#define   RIO_ERR_NO_DEVICE_SUPPORT         0x1006  // Device Access Routine does not
                                                    // provide services for this device
#define   RIO_ERR_INSUFFICIENT_RESOURCES    0x1007  // Insufficient storage available in
                                                    // Device Access Routine private
                                                    // storage area
#define   RIO_ERR_ROUTE_ERROR               0x1008  // Switch cannot support
                                                    // requested routing
#define   RIO_ERR_NO_SWITCH                 0x1009  // Target device is not a switch
#define   RIO_ERR_FEATURE_NOT_SUPPORTED     0x100A  // Target device is not capable of
                                                    // per-input-port routing
```

Blank page

Chapter 3 Hardware Abstraction Layer

3.1 Introduction

The Hardware Abstraction Layer (HAL) provides a standard software interface to the device-specific hardware resources needed to support RapidIO system configuration transactions. Configuration read and write operations are used by the HAL functions to access RapidIO device registers. The HAL functions are accessed by the RapidIO enumeration API during system bring up.

This section describes the HAL functions and how they can be used to access local and remote RapidIO device registers. These functions must be implemented by every new device-specific host-processing element to support RapidIO system enumeration and initialization. The HAL functions assume the following:

- All configuration read and write operations support only single word (4-byte) accesses.
- As required by the device, the size of the 8-bit or 16-bit deviceID field is considered by the device implementation (see section 2.4 of the *RapidIO Part 3: Common Transport Specification* for more information).
- An enumerating processor device may have more than one RapidIO end point (local port).

3.2 Device Addressing

One purpose of the HAL is to provide a unified software interface to configuration registers in both local and remote RapidIO processing elements. This is done using a universal device-addressing scheme. Such a scheme enables HAL functions to distinguish between accesses to local and remote RapidIO end points without requiring an additional parameter. The result is that only one set of HAL functions must be implemented to support local and remote configuration operations.

All HAL functions use the **destid** and **hopcount** parameters to address a RapidIO device. The HAL reserves **destid**=0xFFFFFFFF and hopcount of 0 for addressing configuration registers within the local RapidIO end point. A **destid**= 0xFFFFFFFF and hopcount of 0 value *must* be used to address the local processing end point regardless of the actual destination ID value. This reserved combination does not conflict with the address of other RapidIO devices. The **localport** parameter is used by the HAL functions to identify a specific local port within RapidIO devices containing multiple ports.

3.3 HAL Functions

The functions that form the RapidIO initialization HAL are described in the following sections.

3.3.1 Types and Definitions

```
/* The HOST_REGS value below is a destination ID used to specify that the registers of the processor/platform on which the code is running are to be accessed. */

#define HOST_REGS 0xFFFFFFFF
```

3.3.2 rioGetNumLocalPorts

Prototype:

```
INT32 rioGetNumLocalPorts (
        void
)
```

Arguments:

None

Return Value:

0	Error
n	Number of RapidIO ports supported

Synopsis:

rioGetNumLocalPorts() returns the total number of local RapidIO ports supported by the HAL functions. The number **n** returned by this function should be equal to or greater than 1. A returned value of 0 indicates an error.

3.3.3 rioConfigurationRead

Prototype:

```
STATUS rioConfigurationRead (
        UINT8                       localport,
        UINT32                      destid,
        UINT8                       hopcount,
        UINT32                      offset,
        UINT32                      *readdata
)
```

Arguments:

localport	Local port number [IN]
destid	Destination ID of the target device [IN]
hopcount	Hop count [IN]
offset	Word-aligned (four byte boundary) offset—in bytes—of the CAR or CSR [IN]
*readdata	Pointer to storage for received data [OUT]

Return Value:

RIO_SUCCESS	The read operation completed successfully and valid data was placed into the specified location.
RIO_ERR_INVALID_PARAMETER	One or more input parameters had an invalid value.
RIO_ERR_RIO	The RapidIO fabric returned a Response Packet with ERROR status reported. Error status returned by this function may contain additional information from the Response Packet.
RIO_ERR_ACCESS	A device-specific hardware interface was unable to generate a maintenance transaction and reported an error.

Synopsis:

rioConfigurationRead() performs a configuration read transaction from CAR and/or CSR register(s) belonging to a local or remote RapidIO device. The function uses a device-specific hardware interface to generate maintenance transactions to remote devices. This hardware sends a configuration read request to the remote device (specified by **destid** and/or **hopcount**) and waits for a corresponding configuration read response. After the function receives a configuration read response it returns data and/or status to the caller. The method for accessing registers in a local device is device-specific.

A destid value of HOST_REGS and hopcount of 0 results in accesses to the local hosts RapidIO registers.

3.3.4 rioConfigurationWrite

Prototype:

```
STATUS rioConfigurationWrite (
        UINT8                           localport,
        UINT32                          destid,
        UINT8                           hopcount,
        UINT32                          offset,
        UINT32                          *writedata
)
```

Arguments:

localport	Local port number [IN]
destid	Destination ID of the target device [IN]
hopcount	Hop count [IN]
offset	Word-aligned (four byte boundary) offset—in bytes—of the CAR or CSR [IN]
*writedata	Pointer to storage for data to be written [IN]

Return Value:

RIO_SUCCESS	The write operation completed successfully.
RIO_ERR_INVALID_PARAMETER	One or more input parameters had an invalid value.
RIO_ERR_RIO	The RapidIO fabric returned a Response Packet with ERROR status reported. Error status returned by this function may contain additional information from the Response Packet.
RIO_ERR_ACCESS	A device-specific hardware interface was unable to generate a maintenance transaction and reported an error.

Synopsis:

rioConfigurationWrite() performs a configuration write transaction to CAR and/or CSR register(s) belonging to a local or remote RapidIO device. The function uses a device-specific hardware interface to generate maintenance transactions to remote devices. This hardware sends a configuration write request to the remote device (specified by **destid** and/or **hopcount**) and waits for a corresponding configuration write response. After the function receives a configuration write response it returns status to the caller. The method for accessing registers in a local device is device-specific.

A destid value of HOST_REGS and hopcount of 0 results in accesses to the local hosts RapidIO registers.

Chapter 4 Standard Bring Up Functions

4.1 Introduction

This section describes the RapidIO functions that must be implemented to support system bring up. Functions are defined only for device registers used during the RapidIO enumeration and initialization process, not for all possible RapidIO device registers. These functions can be implemented using the HAL functions. Many of the functions can also be implemented as macros that specify predefined parameters for the HAL functions. The standard RapidIO bring up functions can be combined into a library if they are implemented as a set of subroutines.

4.2 Data Structures

```
typedef ADDR_MODE UINT32;

        #define ADDR_MODE_34BIT_SUPPORT 0x1

        #define ADDR_MODE_50_34BIT_SUPPORT 0x3

        #define ADDR_MODE_66_34BIT_SUPPORT 0x5

        #define ADDR_MODE_66_50_34BIT_SUPPORT 0x7
```

4.3 Bring Up Functions

4.3.1 rioInitLib

Prototype:

```
STATUS rioInitLib (
        void
)
```

Arguments:

None

Return Value:

RIO_SUCCESS	Initialization completed successfully.
RIO_ERROR	Generic error report. Unable to initialize library.

Synopsis:

rioInitLib() initializes the RapidIO API library. No routines defined in this chapter may be called unless and until rioInitLib has been invoked. If rioInitLib returns RIO_ERROR, no routines defined in this chapter may be called.

4.3.2 rioGetFeatures

Prototype:

```
STATUS rioGetFeatures (
        UINT8                   localport,
        UINT32                  destid,
        UINT8                   hopcount,
        UINT32                  *features
)
```

Arguments:

localport	Local port number [IN]
destid	Destination ID of the processing element [IN]
hopcount	Hop count [IN]
*features	Pointer to storage containing the received features [OUT]

Return Value:

RIO_SUCCESS	The features were retrieved successfully and placed into the location specified by *features.
RIO_ERR_INVALID_PARAMETER	One or more input parameters had an invalid value.
RIO_ERR_RIO	The RapidIO fabric returned a Response Packet with ERROR status reported. Error status returned by this function may contain additional information from the Response Packet.
RIO_ERR_ACCESS	A device-specific hardware interface was unable to generate a maintenance transaction and reported an error.

Synopsis:

rioGetFeatures() uses the HAL *rioConfigurationRead()* function to read from the Processing Element Features CAR of the specified processing element. Values read are placed into the location referenced by the ***features** pointer. Reported status is similar to *rioConfigurationRead()*

A destid value of HOST_REGS and hopcount of 0 results in accesses to the local hosts RapidIO registers.

4.3.3 rioGetSwitchPortInfo

Prototype:

```
STATUS rioGetSwitchPortInfo (
        UINT8                   localport,
        UINT32                  destid,
        UINT8                   hopcount,
        UINT32                  *portinfo
)
```

Arguments:

localport	Local port number [IN]
destid	Destination ID of the processing element [IN]
hopcount	Hop count [IN]
*portinfo	Pointer to storage containing the received port information [OUT]

Return Value:

RIO_SUCCESS	The port information was retrieved successfully and placed into the location specified by *portinfo.
RIO_ERR_INVALID_PARAMETER	One or more input parameters had an invalid value.
RIO_ERR_RIO	The RapidIO fabric returned a Response Packet with ERROR status reported. Error status returned by this function may contain additional information from the Response Packet.
RIO_ERR_ACCESS	A device-specific hardware interface was unable to generate a maintenance transaction and reported an error.

Synopsis:

rioGetSwitchPortInfo() uses the HAL *rioConfigurationRead()* function to read from the Switch Port Information CAR of the specified processing element. Values read are placed into the location referenced by the ***portinfo** pointer. Reported status is similar to *rioConfigurationRead()*.

A destid value of HOST_REGS and hopcount of 0 results in accesses to the local hosts RapidIO registers.

4.3.4 rioGetExtFeaturesPtr

Prototype:

```
        STATUS rioGetExtFeaturesPtr (
                UINT8                   localport,
                UINT32                  destid,
                UINT8                   hopcount,
                UINT32                  *extfptr
)
```

Arguments:

localport	Local port number [IN]
destid	Destination ID of the processing element [IN]
hopcount	Hop count [IN]
*extfptr	Pointer to storage containing the received extended feature information [OUT]

Return Value:

RIO_SUCCESS	The extended feature information was retrieved successfully and placed into the location specified by *extfptr.
RIO_ERR_INVALID_PARAMETER	One or more input parameters had an invalid value.
RIO_ERR_RIO	The RapidIO fabric returned a Response Packet with ERROR status reported. Error status returned by this function may contain additional information from the Response Packet.
RIO_ERR_ACCESS	A device-specific hardware interface was unable to generate a maintenance transaction and reported an error.

Synopsis:

rioGetExtFeaturesPtr() uses the HAL *rioConfigurationRead()* function to read the pointer to the first entry in the extended features list from the Assembly Information CAR of the specified processing element. That pointer is placed into the location referenced by the ***extfptr** pointer. Reported status is similar to *rioConfigurationRead()*.

A destid value of HOST_REGS and hopcount of 0 results in accesses to the local hosts RapidIO registers.

Note that if the EF_PTR field of *extfptr is 0, no extended features are available.

4.3.5 rioGetNextExtFeaturesPtr

Prototype:

```
        STATUS rioGetNextExtFeaturesPtr (
                UINT8                           localport,
                UINT32                          destid,
                UINT8                           hopcount,
                UINT32                          currfptr,
                UINT32                          *extfptr
)
```

Arguments:

localport	Local port number [IN]
destid	Destination ID of the processing element [IN]
hopcount	Hop count [IN]
currfptr	Pointer to the last reported extended feature [IN]
*extfptr	Pointer to storage containing the received extended feature information [OUT]

Return Value:

RIO_SUCCESS	The extended feature information was retrieved successfully and placed into the location specified by *extfptr.
RIO_ERR_INVALID_PARAMETER	One or more input parameters had an invalid value.
RIO_ERR_RIO	The RapidIO fabric returned a Response Packet with ERROR status reported. Error status returned by this function may contain additional information from the Response Packet.
RIO_ERR_ACCESS	A device-specific hardware interface was unable to generate a maintenance transaction and reported an error.

Synopsis:

rioGetNextExtFeaturesPtr() uses the HAL *rioConfigurationRead()* function to read the pointer to the next entry in the extended features. That pointer is placed into the location referenced by the ***extfptr** pointer. Reported status is similar to *rioConfigurationRead()*.

A destid value of HOST_REGS and hopcount of 0 results in accesses to the local hosts RapidIO registers.

Note that if the EF_PTR field of *extfptr is 0, no further extended features are available. Invoking rioGetNextExtFeaturesPtr when currfptr has an EF_PTR field value of 0 will result in a return code of RIO_ERR_INVALID_PARAMETER.

4.3.6 rioGetSourceOps

Prototype:

```
STATUS rioGetSourceOps (
        UINT8                                   localport,
        UINT32                                  destid,
        UINT8                                   hopcount,
        UINT32                                  *srcops
)
```

Arguments:

localport	Local port number [IN]
destid	Destination ID of the processing element [IN]
hopcount	Hop count [IN]
*srcops	Pointer to storage containing the received source operation information [OUT]

Return Value:

RIO_SUCCESS	The source operation information was retrieved successfully and placed into the location specified by *srcops.
RIO_ERR_INVALID_PARAMETER	One or more input parameters had an invalid value.
RIO_ERR_RIO	The RapidIO fabric returned a Response Packet with ERROR status reported. Error status returned by this function may contain additional information from the Response Packet.
RIO_ERR_ACCESS	A device-specific hardware interface was unable to generate a maintenance transaction and reported an error.

Synopsis:

rioGetSourceOps() uses the HAL *rioConfigurationRead()* function to read from the Source Operations CAR of the specified processing element. Values read are placed into the location referenced by the ***srcops** pointer. Reported status is similar to *rioConfigurationRead()*.

A destid value of HOST_REGS and hopcount of 0 results in accesses to the local hosts RapidIO registers.

4.3.7 rioGetDestOps

Prototype:

```
STATUS rioGetDestOps (
        UINT8                   localport,
        UINT32                  destid,
        UINT8                   hopcount,
        UINT32                  *dstops
)
```

Arguments:

localport	Local port number [IN]
destid	Destination ID of the processing element [IN]
hopcount	Hop count [IN]
*dstops	Pointer to storage containing the received destination operation information [OUT]

Return Value:

RIO_SUCCESS	The destination operation information was retrieved successfully and placed into the location specified by *dstops.
RIO_ERR_INVALID_PARAMETER	One or more input parameters had an invalid value.
RIO_ERR_RIO	The RapidIO fabric returned a Response Packet with ERROR status reported. Error status returned by this function may contain additional information from the Response Packet.
RIO_ERR_ACCESS	A device-specific hardware interface was unable to generate a maintenance transaction and reported an error.

Synopsis:

rioGetDestOps() uses the HAL *rioConfigurationRead()* function to read from the Destination Operations CAR of the specified processing element. Values read are placed into the location referenced by the ***dstops** pointer. Reported status is similar to *rioConfigurationRead()*.

A destid value of HOST_REGS and hopcount of 0 results in accesses to the local hosts RapidIO registers.

4.3.8 rioGetAddressMode

Prototype:

```
STATUS rioGetAddressMode (
            UINT8                                   localport,
            UINT32                                  destid,
            UINT8                                   hopcount,
            ADDR_MODE                               *amode
)
```

Arguments:

localport	Local port number [IN]
destid	Destination ID of the processing element [IN]
hopcount	Hop count [IN]
*amode	Pointer to storage containing the received address mode (34-bit, 50-bit, or 66-bit address) information [OUT]

Return Value:

RIO_SUCCESS	The address mode information was retrieved successfully and placed into the location specified by *amode.
RIO_ERR_INVALID_PARAMETER	One or more input parameters had an invalid value.
RIO_ERR_RIO	The RapidIO fabric returned a Response Packet with ERROR status reported. Error status returned by this function may contain additional information from the Response Packet.
RIO_ERR_ACCESS	A device-specific hardware interface was unable to generate a maintenance transaction and reported an error.

Synopsis:

rioGetAddressMode() uses the HAL *rioConfigurationRead()* function to read from the PE Logical Layer CSR of the specified processing element. The number of address bits generated by the PE (as the source of an operation) and processed by the PE (as the target of an operation) are placed into the location referenced by the ***amode** pointer. Reported status is similar to *rioConfigurationRead()*.

A destid value of HOST_REGS and hopcount of 0 results in accesses to the local hosts RapidIO registers.

4.3.9 rioGetBaseDeviceId

Prototype:

```
STATUS rioGetBaseDeviceId (
        UINT8                           localport,
        UINT32                          *deviceid
)
```

Arguments:

localport	Local port number [IN]
*deviceid	Pointer to storage containing the base device ID [OUT]

Return Value:

RIO_SUCCESS	The base device ID information was retrieved successfully and placed into the location specified by *deviceid.
RIO_ERR_INVALID_PARAMETER	One or more input parameters had an invalid value.
RIO_ERR_RIO	The RapidIO fabric returned a Response Packet with ERROR status reported. Error status returned by this function may contain additional information from the Response Packet.
RIO_ERR_ACCESS	A device-specific hardware interface was unable to generate a maintenance transaction and reported an error.

Synopsis:

rioGetBaseDeviceId() uses the HAL *rioConfigurationRead()* function to read from the Base Device ID CSR of the local processing element (the **destid** and **hopcount** parameters used by *rioConfigurationRead()* must be set to HOST_REGS and zero, respectively). Values read are placed into the location referenced by the ***deviceid** pointer. Reported status is similar to *rioConfigurationRead()*. This function is useful only for local end-point devices.

4.3.10 rioSetBaseDeviceId

Prototype:

```
STATUS rioSetBaseDeviceId (
        UINT8                    localport,
        UINT32                   destid,
        UINT8                    hopcount,
        UINT32                   newdeviceid
)
```

Arguments:

localport	Local port number [IN]
destid	Destination ID of the processing element [IN]
hopcount	Hop count [IN]
newdeviceid	New base device ID to be set [IN]

Return Value:

RIO_SUCCESS	The base device ID was updated successfully.
RIO_ERR_INVALID_PARAMETER	One or more input parameters had an invalid value.
RIO_ERR_RIO	The RapidIO fabric returned a Response Packet with ERROR status reported. Error status returned by this function may contain additional information from the Response Packet.
RIO_ERR_ACCESS	A device-specific hardware interface was unable to generate a maintenance transaction and reported an error.

Synopsis:

rioSetBaseDeviceId() uses the HAL *rioConfigurationWrite()* function to write the base device ID in the Base Device ID CSR of the specified processing element (end point devices only). Reported status is similar to *rioConfigurationWrite()*.

A destid value of HOST_REGS and hopcount of 0 results in accesses to the local hosts RapidIO registers.

4.3.11 rioAcquireDeviceLock

Prototype:

```
STATUS rioAcquireDeviceLock (
        UINT8                   localport,
        UINT32                  destid,
        UINT8                   hopcount,
        UINT16                  hostdeviceid,
        UINT16                  *hostlockid
)
```

Arguments:

localport	Local port number [IN]
destid	Destination ID of the processing element [IN]
hopcount	Hop count [IN]
hostdeviceid	Host base device ID for the local processing element [IN]
*hostlockid	Device ID of the host holding the lock if ERR_LOCK is returned [OUT]

Return Value:

RIO_SUCCESS	The device lock was acquired successfully.
RIO_ERR_LOCK	Another host already acquired the specified processor element. ID of the device holding the lock is contained in the location referenced by the *hostlockid parameter.
RIO_ERR_INVALID_PARAMETER	One or more input parameters had an invalid value
RIO_ERR_RIO	The RapidIO fabric returned a Response Packet with ERROR status reported. Error status returned by this function may contain additional information from the Response Packet.
RIO_ERR_ACCESS	A device-specific hardware interface was unable to generate a maintenance transaction and reported an error.

Synopsis:

rioAcquireDeviceLock() tries to acquire the hardware device lock for the specified processing element on behalf of the requesting host. The function uses the HAL *rioConfigurationWrite()* function to write the requesting host device ID into the Host Base Lock Device ID CSR of the specified processing element. After the write completes, this function uses the HAL *rioConfigurationRead()* function to read the value back from the Host Base Lock Device ID CSR. The written and read values are compared. If they are equal, the lock was acquired successfully. Otherwise, another host acquired this lock and the device ID for that host is reported.

This function assumes unique host-based device identifiers are assigned to discovering hosts. For more details, refer to Annex A, “System Bring Up Guidelines (Informative)”.

A destid value of HOST_REGS and hopcount of 0 results in accesses to the local hosts RapidIO registers.

4.3.12 rioReleaseDeviceLock

Prototype:

```
STATUS rioReleaseDeviceLock (
        UINT8                       localport,
        UINT32                      destid,
        UINT8                       hopcount,
        UINT16                      hostdeviceid,
        UINT16                      *hostlockid
)
```

Arguments:

localport	Local port number [IN]
destid	Destination ID of the processing element [IN]
hopcount	Hop count [IN]
hostdeviceid	Host base device ID for the local processing element [IN]
*hostlockid	Device ID of the host holding the lock if ERR_LOCK is returned [OUT]

Return Value:

RIO_SUCCESS	The device lock was released successfully.
RIO_ERR_LOCK	Another host already acquired the specified processor element.
RIO_ERR_INVALID_PARAMETER	One or more input parameters had an invalid value
RIO_ERR_RIO	The RapidIO fabric returned a Response Packet with ERROR status reported. Error status returned by this function may contain additional information from the Response Packet.
RIO_ERR_ACCESS	A device-specific hardware interface was unable to generate a maintenance transaction and reported an error.

Synopsis:

rioReleaseDeviceLock() tries to release the hardware device lock for the specified processing element on behalf of the requesting host. The function uses the HAL *rioConfigurationWrite()* function to write the requesting host device ID into the Host Base Lock Device ID CSR of the specified processing element. After the write completes, this function uses the HAL *rioConfigurationRead()* function to read the value back from the Host Base Lock Device ID CSR. If the Device ID that is read back from the Host Base Device ID register is 0xFFFF then the lock has been released successfully.

A destid value of HOST_REGS and hopcount of 0 results in accesses to the local hosts RapidIO registers.

4.3.13 rioGetComponentTag

Prototype:

```
STATUS rioGetComponentTag (
        UINT8                       localport,
        UINT32                      destid,
        UINT8                       hopcount,
        UINT32                      *componenttag
)
```

Arguments:

localport	Local port number [IN]
destid	Destination ID of the processing element [IN]
hopcount	Hop count [IN]
*componenttag	Pointer to storage containing the received component tag information [OUT]

Return Value:

RIO_SUCCESS	The component tag information was retrieved successfully and placed into the location specified by *componenttag.
RIO_ERR_INVALID_PARAMETER	One or more input parameters had an invalid value.
RIO_ERR_RIO	The RapidIO fabric returned a Response Packet with ERROR status reported. Error status returned by this function may contain additional information from the Response Packet.
RIO_ERR_ACCESS	A device-specific hardware interface was unable to generate a maintenance transaction and reported an error.

Synopsis:

rioGetComponentTag() uses the HAL *rioConfigurationRead()* function to read from the Component Tag CSR of the specified processing element. Values read are placed into the location referenced by the ***componenttag** pointer. Reported status is similar to *rioConfigurationRead()*.

A destid value of HOST_REGS and hopcount of 0 results in accesses to the local hosts RapidIO registers.

4.3.14 rioSetComponentTag

Prototype:

```
STATUS rioSetComponentTag (
            UINT8                              localport,
            UINT32                             destid,
            UINT8                              hopcount,
            UINT32                             componenttag
)
```

Arguments:

localport	Local port number [IN]
destid	Destination ID of the processing element [IN]
hopcount	Hop count [IN]
componenttag	Component tag value to be set [IN]

Return Value:

RIO_SUCCESS	The component tag was updated successfully.
RIO_ERR_INVALID_PARAMETER	One or more input parameters had an invalid value.
RIO_ERR_RIO	The RapidIO fabric returned a Response Packet with ERROR status reported. Error status returned by this function may contain additional information from the Response Packet.
RIO_ERR_ACCESS	A device-specific hardware interface was unable to generate a maintenance transaction and reported an error.

Synopsis:

rioSetComponentTag() uses the HAL *rioConfigurationWrite()* function to write the component tag into the Component Tag CSR of the specified processing element. Reported status is similar to *rioConfigurationWrite()*.

A destid value of HOST_REGS and hopcount of 0 results in accesses to the local hosts RapidIO registers.

4.3.15 rioGetPortErrStatus

Prototype:

```
STATUS rioGetPortErrStatus (
        UINT8                   localport,
        UINT32                  destid,
        UINT8                   hopcount,
        UINT16                  extfoffset,
        UINT8                   portnum,
        UINT32                  *porterrorstatus
)
```

Arguments:

localport	Local port number [IN]
destid	Destination ID of the processing element [IN]
hopcount	Hop count [IN]
extfoffset	Offset from the previously reported extended features pointer [IN]
portnum	Port number to be accessed [IN]
*porterrorstatus	Pointer to storage for the returned value [OUT]

Return Value:

RIO_SUCCESS	The read completed successfully and valid data was placed into the location specified by *porterrorstatus.
RIO_ERR_INVALID_PARAMETER	One or more input parameters had an invalid value.
RIO_ERR_RIO	The RapidIO fabric returned a Response Packet with ERROR status reported. Error status returned by this function may contain additional information from the Response Packet.
RIO_ERR_ACCESS	A device-specific hardware interface was unable to generate a maintenance transaction and reported an error.

Synopsis:

rioGetPortErrStatus() uses the HAL *rioConfigurationRead()* function to read the contents of the Port *n* Error and Status CSR of the specified processing element. Reported status is similar to *rioConfigurationRead()*.

A destid value of HOST_REGS and hopcount of 0 results in accesses to the local hosts RapidIO registers.

Chapter 5 Routing-Table Manipulation Functions

5.1 Introduction

This section describes the RapidIO functions that must be provided to support routing tables used within the switch fabric. The RapidIO common transport specification requires implementing device-identifier-based packet routing. The detailed implementation of routing tables is beyond the scope of this specification.

The routing-table manipulation functions assume the following:

- The destination ID of the device that receives a packet routed by the switch is the *route destination ID*.
- The specific port at the route destination ID that receives a packet routed by the switch is the *route port number.*
- The software paradigm used for routing tables is a linear routing table indexed by the route destination ID.
- Switches may implement a global routing table, “per port” routing tables, or a combination of both.

5.2 Routing Table Functions

The functions defined for RapidIO routing-table manipulation are described in the following sections.

5.2.1 rioRouteAddEntry

Prototype:

```
STATUS rioRouteAddEntry (
        UINT8                   localport,
        UINT32                  destid,
        UINT8                   hopcount,
        UINT8                   tableidx,
        UINT16                  routedestid,
        UINT8                   routeportno
)
```

Arguments:

localport	Local port number (RapidIO switch) [IN]
destid	Destination ID of the processing element (RapidIO switch) [IN]
hopcount	Hop count [IN]
tableidx	Routing table index for per-port switch implementations [IN]
routedestid	Route destination ID—used to select an entry into the specified routing table [IN]
routeportno	Route port number—value written to the selected routing table entry [IN]

Return Value:

RIO_SUCCESS	The routing table entry was added successfully.
RIO_ERR_INVALID_PARAMETER	One or more input parameters had an invalid value.
RIO_ERR_RIO	The RapidIO fabric returned a Response Packet with ERROR status reported. Error status returned by this function may contain additional information from the Response Packet.
RIO_ERR_ACCESS	A device-specific hardware interface was unable to generate a maintenance transaction and reported an error.
RIO_WARN_INCONSISTENT	Used by rioRouteGetEntry—indicates that the routeportno returned is not the same for all ports.

Synopsis:

rioRouteAddEntry() adds an entry to a routing table for the RapidIO switch specified by the **destid** and **hopcount** parameters. The **tableidx** parameter is used to select a specific routing table in the case of implementations with “per port” routing tables. A value of **tableidx**=0xFFFFFFFF specifies a global routing table for the RapidIO switch. The **routeportno** parameter is written to the routing table entry selected by the **routedestid** parameter.

A destid value of HOST_REGS and hopcount of 0 results in accesses to the local hosts RapidIO registers.

5.2.2 rioRouteGetEntry

Prototype:

```
STATUS rioRouteGetEntry (
        UINT8                   localport,
        UINT32                  destid,
        UINT8                   hopcount,
        UINT8                   tableidx,
        UINT16                  routedestid,
        UINT8                   *routeportno
)
```

Arguments:

localport	Local port number (RapidIO switch) [IN]
destid	Destination ID of the processing element (RapidIO switch) [IN]
hopcount	Hop count [IN]
tableidx	Routing table index for per-port switch implementations [IN]
routedestid	Route destination ID—used to select an entry into the specified routing table [IN]
*routeportno	Route port number—pointer to value read from the selected routing table entry [OUT]

Return Value:

RIO_SUCCESS	The routing table entry was added successfully.
RIO_ERR_INVALID_PARAMETER	One or more input parameters had an invalid value.
RIO_ERR_RIO	The RapidIO fabric returned a Response Packet with ERROR status reported. Error status returned by this function may contain additional information from the Response Packet.
RIO_ERR_ACCESS	A device-specific hardware interface was unable to generate a maintenance transaction and reported an error.
RIO_WARN_INCONSISTENT	Used by rioRouteGetEntry—indicates that the routeportno returned is not the same for all ports.

Synopsis:

rioRouteGetEntry() reads an entry from a routing table for the RapidIO switch specified by the **destid** and **hopcount** parameters. The **tableidx** parameter is used to select a specific routing table in the case of implementations with “per port” routing tables. A value of **tableidx**=0xFF specifies a global routing table for the RapidIO switch. The value in the routing table entry selected by the **routedestid** parameter is read from the table and placed into the location referenced by the ***routeportno** pointer.

Reads from the global routing table may be undefined in the case where per-port routing tables exist.

A destid value of HOST_REGS and hopcount of 0 results in accesses to the local hosts RapidIO registers.

Blank page

Chapter 6 Device Access Routine Interface

6.1 Introduction

This section defines the device access routine (DAR) interface that must be provided for RapidIO device configuration. The client for this interface is the boot loader responsible for RapidIO network enumeration and initialization. By using a standard DAR interface, the firmware does not need to include knowledge of device-specific configuration operations. Thus, enumeration and initialization firmware can operate transparently with devices from many component vendors.

6.2 DAR Packaging

For each processor type supported by a DAR provider, linkable object files for DARs shall be supplied using ELF format. Device-specific configuration DARs shall be supplied using C-language source code format.

6.3 Execution Environment

The functions provided by device-specific configuration DARs must be able to link and execute within a minimal execution context (e.g., a system-boot monitor or firmware). In general, configuration DARs should not call an external function that is not implemented by the DAR, unless the external function is passed to the configuration DAR by the initialization function. Also, configuration DAR functions may not call standard C-language I/O functions (e.g., *printf*) or standard C-language library functions that might manipulate the execution environment (e.g., *malloc* or *exit*).

6.4 Type Definitions

The following type definitions are to be used by the DAR functions in Section 6.5.

```
typedef struct RDCDAR_PLAT_OPS_STRUCT {
        UINT32 specversion;
        UINT32 (*rioConfigurationRead)      (     UINT8      localport,
                                                  UINT16     destid,
                                                  UINT8      hopcount,
                                                  UINT32     offset,
                                                  UINT32     *readdata);
        UINT32 (*rioConfigurationWrite)     (     UINT8      localport,
                                                  UINT16     destid,
                                                  UINT8      hopcount,
                                                  UINT32     offset,
                                                  UINT32     *writedata);
} RDCDAR_PLAT_OPS;

typedef struct RDCDAR_OPS_STRUCT {
        UINT32 specversion;
        UINT32 (*rioDarInitialize)                (...);
        UINT32 (*rioDarTerminate)                 (...);
        UINT32 (*rioDarTestMatch)                 (...);
        UINT32 (*rioDarRegister)                  (...);
        UINT32 (*rioDarGetSwitchInfo)             (...);
        UINT32 (*rioDarSetPortRoute)              (...);
        UINT32 (*rioDarGetPortRoute)              (...);
        UINT32 (*rioDarGetMemorySize)             (...);
} RDCDAR_OPS

typedef struct RDCDAR_DATA_STRUCT {
        UINT32 databytesallocated;
        CHAR   *data;
} RDCDAR_DATA

typedef struct RDCDAR_SWITCH_INFO_STRUCT {
        BOOL   useslutmodel;
        BOOL   separatelutperinputport;
        UINT32 maxlutentries;
} RDCDAR_SWITCH_INFO
```

6.5 DAR Functions

The functions that must be provided for a RapidIO device-specific configuration DAR are described in the following sections. For the *rioDar_name*GetFunctionTable functions, the *rioDar_name* portion of the function name shall be replaced by an appropriate name for the implemented driver.

6.5.1 *rioDar_name*GetFunctionTable

Prototype:

```
UINT32 rioDar_nameGetFunctionTable(
            UINT32                          specversion,
            RDCDAR_OPS_STRUCT               *darops,
            UINT32                          maxdevices,
            UINT32                          *darspecificdatabytes
)
```

Arguments:

specversion	Version number of the DAR interface specification indicating the caller's implementation of the type definition structures [IN]
*darops	Pointer to a structure of DAR functions that are allocated by the caller and filled in by the called function (see Section 6.4) [OUT]
maxdevices	Maximum expected number of RapidIO devices that must be serviced by this configuration DAR [IN]
*darspecificdatabytes	Number of bytes needed by the DAR for the DAR private data storage area [OUT]

Return value:

RIO_SUCCESS	On successful completion

Synopsis:

rioDar_name*GetFunctionTable()* is called by a client to obtain the list of functions implemented by a RapidIO device-specific configuration DAR module. It shall be called once before enumerating the RapidIO network.

The **specversion** parameter is the version number defined by the revision level of the specification from which the DAR type definition structures are taken (see Section 6.4).

The **maxdevices** parameter is an estimate of the maximum number of RapidIO devices in the network that this DAR must service. The DAR uses this estimate to determine the size required for the DAR private data storage area. The storage size is returned to the location referenced by the ***darspecificdatabytes** pointer. After the client calls this function, the client shall allocate a DAR private data storage area of a size no less than that indicated by ***darspecificdatabytes**. The client shall provide that private data storage area to *rioDarInitialize()*.

6.5.2 rioDarInitialize

Prototype:

```
UINT32 rioDarInitialize (
        UINT32                  specversion,
        UINT32                  maxdevices,
        RDCDAR_PLAT_OPS         *platops,
        RDCDAR_DATA             *privdata
)
```

Arguments:

specversion	Version number of the DAR interface specification indicating the caller’s implementation of the type definition structures [IN]
maxdevices	Maximum expected number of RapidIO devices that must be serviced by this configuration DAR [IN]
*platops	Pointer to a structure of platform functions for use by the DAR (see Section 6.4) [IN]
*privdata	Pointer to structure containing DAR private data area (see Section 6.4) [IN/OUT]

Return value:

RIO_SUCCESS	On successful completion

Synopsis:

rioDarInitialize() is called by a client to initialize a RapidIO device-specific configuration DAR module. This function shall be called once after calling the rioDar_name*GetFunctionTable()* functions and before enumerating the RapidIO network.

The **specversion** parameter is the version number defined by the revision level of the specification from which the DAR type definition structures are taken (see Section 6.4).

The **maxdevices** parameter is an estimate of the maximum number of RapidIO devices in the network that this DAR must service. The **maxdevices** value must be equal to the value used in the corresponding rioDar_name*GetFunctionTable()* function call. The client is responsible for allocating the structure referenced by ***privdata**. The client is also responsible for allocating a DAR private data storage area at least as large as that specified by the rioDar_name*GetFunctionTable()* call. The client must initialize the structure referenced by ***privdata** with the number of bytes allocated to the DAR private data storage area and with the pointer to the storage area. After calling *rioDarInitialize()*, the client may not deallocate the DAR private data storage area until after the *rioDarTerminate()* function has been called.

6.5.3 rioDarTerminate

Prototype:

```
UINT32 rioDarTerminate (
        RDCDAR_DATA                 *privdata
)
```

Arguments:

*privdata	Pointer to structure containing DAR private data area (see Section 6.4) [IN/OUT]

Return value:

RIO_SUCCESS	On successful completion

Synopsis:

rioDarTerminate() is invoked by a client to terminate a RapidIO device-specific configuration DAR module. This function shall be called once after all use of the DAR services is completed. After calling this function, the client may deallocate the DAR private data storage area in the structure referenced by ***privdata**.

6.5.4 rioDarTestMatch

Prototype:

```
UINT32 rioDarTestMatch (
        RDCDAR_DATA                 *privdata,
        UINT8                       localport,
        UINT32                      destid,
        UINT8                       hopcount
)
```

Arguments:

*privdata	Pointer to structure containing DAR private data area (see Section 6.4) [IN/OUT]
localport	Local port number used to access the network [IN]
destid	Destination device ID for the target device [IN]
hopcount	Number of switch hops needed to reach the target device [IN]

Return value:

RIO_SUCCESS	Device DAR does provide services for this device
RIO_ERR_NO_DEVICE_SUPPORT	Device DAR does not provide services for this device.

Synopsis:

rioDarTestMatch() is invoked by a client to determine whether or not a RapidIO device-specific configuration DAR module provides services for the device specified by **destid**. The DAR interrogates the device (using the platform functions supplied during DAR initialization), examines the device identity and any necessary device registers, and determines whether or not the device is handled by the DAR.

The DAR does not assume that a positive match (return value of 0) means the DAR will actually provide services for the device. The client must explicitly register the device with *rioDARregister()* if the client will be requesting services.

A destid value of HOST_REGS and hopcount of 0 results in accesses to the local hosts RapidIO registers.

6.5.5 rioDarRegister

Prototype:

```
UINT32 rioDarRegister (
        RDCDAR_DATA                  *privdata,
        UINT8                        localport,
        UINT32                       destid,
        UINT8                        hopcount,
)
```

Arguments:

*privdata	Pointer to structure containing DAR private data area (see Section 6.4) [IN/OUT]
localport	Local port number used to access the network [IN]
destid	Destination device ID for the target device [IN]
hopcount	Number of switch hops needed to reach the target device [IN]

Return value:

RIO_SUCCESS	Device DAR successfully registered this device.
RIO_ERR_NO_DEVICE_SUPPORT	Device DAR does not provide services for this device.
RIO_ERR_INSUFFICIENT_RESOURCES	Insufficient storage available in DAR private storage area

Synopsis:

rioDarRegister() is invoked by a client to register a target device with a RapidIO device-specific configuration DAR. The client must call this function once for each device serviced by the DAR. The client should first use the *rioDarTestMatch()* function to verify that the DAR is capable of providing services to the device.

A destid value of HOST_REGS and hopcount of 0 results in accesses to the local hosts RapidIO registers.

6.5.6 rioDarGetMemorySize

Prototype:

```
UINT32 rioDarGetMemorySize (
        RDCDAR_DATA                 *privdata,
        UINT8                       localport,
        UINT32                      destid,
        UINT8                       hopcount,
        UINT32                      regionix,
        UINT32                      *nregions,
        UINT32                      *regbytes[2],
        UINT32                      *startoffset[2]
)
```

Arguments:

*privdata	Pointer to structure containing DAR private data area (see Section 6.4) [IN/OUT]
localport	Local port number used to access the network [IN]
destid	Destination device ID for the target device [IN]
hopcount	Number of switch hops needed to reach the target device [IN]
regionix	Index of the memory region being queried (0, 1, 2, 3, ...) [IN]
*nregions	Number of memory regions provided by the target device [OUT]
*regbytes	Size (in bytes) of the queried memory region [OUT]
*startoffset	Starting address offset for the queried memory region [OUT]

Return value:

RIO_SUCCESS	Device DAR successfully returned memory size information for the target device.
RIO_ERR_NO_DEVICE_SUPPORT	Device DAR could not determine memory size information for the target device.

Synopsis:

rioDarGetMemorySize() is invoked by a client to determine the number of, the sizes of, and the offsets for the memory regions supported by a RapidIO target device. The function is intended to support the mapping of PCI or other address windows to RapidIO devices. If the **regionix** parameter is greater than the number of regions provided by the device (***nregions**), the DAR should return a value of zero for the ***regbytes** and ***startoffset** parameters, and indicate a “successful” (0) return code.

rioDarGetMemorySize always returns at least one region. The first index, index 0, always refers to the region controlled by the Local Configuration Space Base Address Registers.

The client must register the target device with the RapidIO device-specific configuration DAR before calling this function.

A destid value of HOST_REGS and hopcount of 0 results in accesses to the local hosts RapidIO registers.

6.5.7 rioDarGetSwitchInfo

Prototype:

```
UINT32 rioDarGetSwitchInfo (
        RDCDAR_DATA              *privdata,
        UINT8                    localport,
        UINT32                   destid,
        UINT8                    hopcount,
        RDCDAR_SWITCH_INFO       *info
)
```

Arguments:

*privdata	Pointer to structure containing DAR private data area (see Section 6.4) [IN/OUT]
localport	Local port number to be used to access network [IN]
destid	Destination device ID to reach target switch device [IN]
hopcount	Number of switch hops to reach target switch device [IN]
*info	Pointer to switch information data structure (see Section 6.4) [OUT]

Return value:

RIO_SUCCESS	Device DAR successfully retrieved the information for RDCDAR_PLAT_OPS_STRUCT.
RIO_ERR_NO_DEVICE_SUPPORT	Insufficient switch routing resources available.
RIO_ERR_NO_SWITCH	Target device is not a switch.

Synopsis:

rioDarGetSwitchInfo() is invoked by a client to retrieve the data necessary to initialize the RDCDAR_SWITCH_INFO structure.

The client must register the target device with the RapidIO device-specific configuration DAR before calling this function.

A destid value of HOST_REGS and hopcount of 0 results in accesses to the local hosts RapidIO registers.

6.5.8 rioDarSetPortRoute

Prototype:

```
UINT32 rioDarSetPortRoute (
        RDCDAR_DATA          *privdata,
        UINT8                localport,
        UINT32               destid,
        UINT8                hopcount,
        UINT8                tableidx,
        UINT16               routedestid,
        UINT8                routeportno
)
```

Arguments:

*privdata	Pointer to structure containing DAR private data area (see Section 6.4) [IN/OUT]
localport	Local port number to be used to access network [IN]
destid	Destination device ID to reach target switch device [IN]
hopcount	Number of switch hops to reach target switch device [IN]
inport	Target switch device input port [IN]
tableidx	Routing table index for per-port switch implementations [IN]
routedestid	Route destination ID—used to select an entry into the specified routing table [IN]
routeportno	Route port number—value written to the selected routing table entry [IN]

Return value:

RIO_SUCCESS	Device DAR successfully modified the packet routing configuration for the target switch device.
RIO_ERR_NO_DEVICE_SUPPORT	Insufficient switch routing resources available.
RIO_ERR_ROUTE_ERROR	Switch cannot support requested routing.
RIO_ERR_NO_SWITCH	Target device is not a switch.
RIO_ERR_FEATURE_NOT_SUPPORTED	Target device is not capable of per-input-port routing.

Synopsis:

rioDarSetPortRoute() is invoked by a client to modify the packet routing configuration for a RapidIO target switch device.

The client must register the target device with the RapidIO device-specific configuration DAR before calling this function.

A destid value of HOST_REGS and hopcount of 0 results in accesses to the local hosts RapidIO registers.

6.5.9 rioDarGetPortRoute

Prototype:

```
UINT32 rioDarGetPortRoute (
        RDCDAR_DATA               *privdata,
        UINT8                     localport,
        UINT32                    destid,
        UINT8                     hopcount,
        UINT8                     tableidx,
        UINT16                    routedestid,
        UINT8                     *routeportno
)
```

Arguments:

*privdata	Pointer to structure containing DAR private data area (see Section 6.4) [IN/OUT]
localport	Local port number to be used to access network [IN]
destid	Destination device ID to reach target switch device [IN]
hopcount	Number of switch hops to reach target switch device [IN]
tableidx	Routing table index for per-port switch implementations [IN]
routedestid	Route destination ID—used to select an entry into the specified routing table [IN]
*routeportno	Route port number—pointer to value read from the selected routing table entry [OUT]

Return value:

RIO_SUCCESS	Device DAR successfully modified the packet routing configuration for the target switch device.
RIO_ERR_NO_DEVICE_SUPPORT	Insufficient switch routing resources available.
RIO_ERR_ROUTE_ERROR	Switch cannot support requested routing.
RIO_ERR_NO_SWITCH	Target device is not a switch.

Synopsis:

rioDarGetPortRoute() is invoked by a client to read the packet routing configuration for a RapidIO target switch device.

The client must register the target device with the RapidIO device-specific configuration DAR before calling this function.

A destid value of HOST_REGS and hopcount of 0 results in accesses to the local hosts RapidIO registers.

Blank page

Annex A System Bring Up Guidelines (Informative)

A.1 Introduction

The *RapidIO Annex 1: Software/System Bring Up Specification* defines a standard set of software API functions for use in system enumeration and initialization. These API functions enable up to two RapidIO hosts to cooperatively enumerate and configure a RapidIO network.

This appendix is provided as a reference model for the system bring up process. An algorithm is presented that enables up to two cooperating host processors in a Rapid IO system to enumerate the entire network, set up a route to every system node, and enable the booting software to start the next boot-process phase. The actual implementation of the algorithm used to bring up a RapidIO network can vary greatly from this model in both capability and complexity.

A.2 Overview of the System Bring Up Process

This section presents a high-level overview of the system bring up process.

1. The system is powered on. Refer to Chapter 2, “Requirements for System Bring Up” for the system power-on requirements.
2. The host processor fetches the initial boot code (if necessary). If two processors are present, both can fetch the initial boot code.
3. The system exploration and enumeration algorithm is started. The algorithm for this process is outlined in Section A.3 on page 50.
4. All devices have been enumerated and stored in the device database, and routes have been set up between the host device and all end point devices. The enumeration process may optionally choose to do the following:
    a) Compute and configure optimal routes between the host device and end point devices, and between different end point devices.
    b) Configure the switch devices with the optimal route information.
    c) Store the optimal route and alternate route information in the device database.
5. The address space is mapped.

The host may access the network across a host-RapidIO bridge or host-PCI bridge. The address-space mapping across this bridge must be done when devices are enumerated and stored in the device database. This allows the address of a found device to be retrieved later and presented to the device access routines during operating system (OS) initialization. The pseudocode for this process is as follows:

```
ACQUIRE the host bridge address-space requirement
MAP the address space into a host address partition X
FOR every device in the database
    IF the component is a RapidIO device
        ACQUIRE the device’s address-space requirement
        MAP the address space into a new host address partition
        EXPAND the partition X window to cover the new partition
        UPDATE the device database with the new host address
    ELSE IF the component is a PCI bridge
        ACQUIRE the bridge’s PCI bus ID
        ACQUIRE the bridge’s address-space requirement
        // All devices that appear behind this PCI bridge must have their address spaces
        // mapped within the region specified for this bridge.
        MAP the address space into a new host address partition
        EXPAND the partition X window to cover the new partition
        UPDATE the device database with the new host address
    ENDIF
ENDFOR
```

After discovery has been concluded, it is expected that the majority of systems will then attempt to load in a software image from a boot device.

A.3 System Enumeration Algorithm

The system enumeration algorithm is designed for use by one or two host processors. The outline of the algorithm is as follows:

1. Access the RapidIO network. This step may involve generating special transaction cycles to ensure that the RapidIO network is accessible.
2. Discover the host and assign a device ID to it.
3. Discover the neighbor, if present.
4. If necessary, repeat the previous step recursively to discover additional devices.
5. Clear up.

When a host begins exploring, it must acquire the Host Base Device ID Lock before it can proceed. Once acquired, it can set its device ID and discover its neighbor (if necessary).

If two hosts are used, both can execute the enumeration algorithm. However, only one host (the one with higher priority) can win the enumeration task. The losing host enters a wait state. The guidelines for prioritizing hosts to enumerate the network and restarting enumeration should the winning host fail to complete the task are described in Chapter 2, “Requirements for System Bring Up,” on page 9.

The enumeration algorithm described below sets priority based on the value of the power-on device ID. The winning host is the device with the higher power-on host device ID. The losing host has the lower power-on host device ID. The losing host enters a wait state until the winning host completes enumeration or until the wait state times out.

The prioritization mechanism never results in a deadlock if the priorities of both host processors are unique. The enumeration process is initially performed in parallel by both hosts until they meet at a device. When a meeting occurs, prioritization guarantees one winning host—the other host retreats (enters a wait state).

The enumeration algorithm described below uses a recursive, depth-first graph traversal to discover the network. It may be possible to improve the algorithm using non-recursive or breadth-first graph traversal. However, those improvements and optimizations are implementation dependent and beyond the scope of this document.

A.3.1 Data Structures, Constants, and Global Variables

This section outlines the data structures, constants, and global variables used by the system enumeration algorithm pseudocode.

The example system is composed of only 8 bit capable devices.

Data Structures

```
struct rioRouteTable {

        // The switch routing table is implemented as a linear routing table for destination IDs. The table is
        // indexed using the destination ID and the table index range is equal to the maximum destination ID
        // value. The value of a table entry indicates the output port number used to route messages for the
        // destination ID. The table entry default value is implementation dependent. Table entries must be
        // initialized to support FLASH memory accesses. The algorithm pseudocode described in this
        // document assumes the device ID is equal to the RapidIO protocols destination ID. This assignment
        // is not a general requirement.

        UINT8   LFT[MAX_DEVICEID];
}

struct rioSwitch {

        …

        UINT16 SwitchIdentity;                  // Switch Identity
        UINT16 hopCount;                        // Hop Count to reach this switch
        UINT16 DeviceID;                        // Associated Device ID in the path to this switch
        struct rioRouteTable RouteTable;        // Switch Routing Table

        …

}
```

Constants

```
        RIO_GEN_DFLT_DID 0x00FFFFFF             // RIO_GEN_DFLT_DID is the general default device
                                                // ID assigned to non-host and non-boot code end
                                                // points
        RIO_BOOT_DFLT_DID 0x0000FFFE            // RIO_BOOT_DFLT_DID is the default device ID
                                                // assigned to boot code devices
        RIO_HOST_DFLT_DID 0x00000000            // RIO_HOST_DFLT_DID is the default device ID
                                                // assigned to host devices
```

Global Variables

```
        UINT16 DeviceID = 0;                    // Currently available Device ID to be assigned to the
```

```
                                          // end point device
UINT16 SwitchID = 0;                      // Currently available Switch ID. This is used
                                          // internally by the to index
                                          // switches that have been discovered.

                                          // The following global arrays are used to store device
                                          // information
                                          // collected from rioGetFeatures and
                                          // rioGetSwitchPortInfo. They are
                                          // also used to store the hopCount and DeviceID
                                          // assigned to switches.

struct rioSwitch Switches[MAX_SWITCHES];
```

A.3.2 Pseudocode

This section outlines the detailed pseudocode for the system enumeration algorithm.

```
//**********************************************************************
// System enumeration and initialization using the power-on device ID as the hostDeviceID
// —Discover the host first
// —Discover the host's neighbor recursively

STATUS rioSystemEnumerate (hostDeviceID)
{
    // Discover the host first.
    status = rioEnumerateHost (hostDeviceID);

    if (status == ERR_SLAVE) {
         rioClearUp (hostDeviceID);
         return ERR_SLAVE;
    }

    // Discover the host neighbor
    status = rioEnumerateNeighbor (hostDeviceID, hopCount = 1);

    if (status == ERR_SLAVE) {
         rioClearUp (hostDeviceID);
         return ERR_SLAVE;
    }

    // If the code advances to this point successfully, the host must acquire the
    // HostBaseDeviceIdLock for all devices in the system. When this is done, the Discovered bit
    // Master Enable bit, etc. can be set for all devices.

}   // end rioSystemEnumerate

//**********************************************************************
// System Delay
// —Wait for other host to release the lock

rioDelay () {
}   // end rioDelay

//**********************************************************************
// Host enumeration and initialization

STATUS rioEnumerateHost (hostDeviceID)
```

```
{
    // Try to acquire the lock
    rioAcquireDeviceLock (0, hostDeviceID, 0, hostDeviceID);

    while (HostBaseDeviceIdLockCSR.HostBaseDeviceID < hostDeviceID) {
         // Delay for a while
         rioDelay ();

         // Retry lock acquisition
         rioAcquireDeviceLock (0, hostDeviceID, 0, hostDeviceID, &lockingHost);
    }

    // Check to see if there is a master with a larger host device ID
    if (HostBaseDeviceIdLock.HostBaseDeviceID > hostDeviceID) {
         // Release the current lock
         rioReleaseDeviceLock (0, hostDeviceID, 0, hostDeviceID);

         return ERR_SLAVE;
    }

    // Lock has been acquired so enumeration can begin

    // Assign the default host ID to the host
    rioSetBaseDeviceId (0, hostDeviceID, hostDeviceID);

    // Increment the available device ID
    if (DeviceID == hostDeviceID) {
         DeviceID ++;
    }

    return RIO_SUCCESS;
}   // end rioEnumerateHost

//************************************************************************
// Neighbor enumeration

STATUS rioEnumerateNeighbor (hostDeviceID, hopCount)
{
    // The host has already discovered this node if it currently owns the lock
    rioGetCurHostLock (0, 0, 0, &owner_device_id);
    if (owner_device_id == hostDeviceID) {
         return RIO_SUCCESS;
    }

    // Try to acquire the lock
    rioAcquireDeviceLock (0, RIO_GEN_DFLT_DID, hopCount, hostDeviceID, &lockingHost);

    while (HostBaseDeviceIdLockCSR.HostBaseDeviceID < hostDeviceID) {
         // Delay for a while
         rioDelay ();

         // Retry lock acquisition
         rioAcquireDeviceLock(0, RIO_GEN_DFLT_DID, hopCount, hostDeviceID,
              &lockingHost);
    }

```

```
// Check to see if there is a master with a larger host device ID
if (HostBaseDeviceIdLock.HostBaseDeviceID > hostDeviceID) {
    return ERR_SLAVE;
}

// Lock has been acquired so enumeration can begin

// Check Source Operation CAR and Destination Operation CAR to see if a Device ID can be
// assigned

rioGetSourceOps (0, RIO_GEN_DFLT_DID, hopCount, &SourceOperationCAR);
rioGetDestOps (0, RIO_GEN_DFLT_DID, hopCount, &DestinationOperationCAR);

if ( (SourceOperationCAR.Read || Write || Atomic) &&
    (DestinationOperationCAR.Read || Write || Atomic)) {

    // Set the device ID
    rioSetBaseDeviceId (0, RIO_GEN_DFLT_DID, DeviceID);

    // Increment the available device ID
    DeviceID ++;
    if (DeviceID == hostDeviceID) {
        DeviceID ++;
    }
}

// Check to see if the device is a switch
rioGetFeatures (0, RIO_GEN_DFLT_DID, hopCount, &ProcessingElementFeatureCAR);
if (ProcessingElementFeatureCAR.Switch == TRUE) {

    // Read the switch information
    rioGetSwitchPortInfo (0, RIO_GEN_DFLT_DID, hopCount,
        &SwitchPortInformationCAR);

    // Record the switch device identity
    Switches[SwitchID].SwitchIdentity = DeviceIdentityCAR.DeviceIdentity;

    // Bookkeeping for the current switch ID
    curSwitchID = SwitchID;

    // Increment the available switch ID
    SwitchID ++;

    // Initialize the current switch routing table to add entries for all previously discovered
    // devices so that they are routed correctly. Start with the host device ID (0x00) and end with
    // DeviceID-1.
    for (each deviceID in [0..DeviceID-1]) {
        rioRouteAddEntry (0, RIO_GEN_DFLT_DID, hopCount, RIO_GEN_DFLT_DID,
                deviceID,
            SwitchPortInformationCAR.PortNumber, NULL);
    }

    // Synchronize the current switch routing table with the global table
    for (each deviceID in [0.. DeviceID-1]) {
        Switches[curSwitchID].RouteTable.LFT[deviceID] =
            SwitchPortInformationCAR.PortNumber;
    }

    // Update the hopCount to reach the current switch
    Switches[curSwitchID].HopCount = hopCount;

    for (each portNum in SwitchPortInformationCAR.PortTotal) {
        if (SwitchPortInformationCAR.PortNumber == portNum) {
            continue;
        }

        // Bookkeeping for the current available device ID
        curDeviceID = DeviceID;

        rioGetPortErrStatus (0, RIO_GEN_DFLT_DID, hopCount,
            &PortErrorStatusCSR[portNum]);

        // Check if it is possible to have a neighbor
        if (PortErrorStatusCSR[portNum].PortUninitialized == TRUE) {
            continue;
        }

        else if (PortErrorStatusCSR[portNum].PortOK == TRUE) {

            // Check if it is an enumeration boundary port
            if (PortControlCSR[portNum].PortEnumerationBoundary == TRUE) {
                continue;
            }
            rioRouteAddEntry(0, RIO_GEN_DFLT_DID, hopCount, RIO_GEN_DFLT_DID, 0,
                portNumber, NULL);

            // Discover the neighbor recursively
            if (status = rioEnumerateNeighbor(hopCount + 1) != RIO_SUCCESS) {
                return status;
            }

            // If more than one end point device was found, update the current switch routing table
            // entries beginning with the curDeviceID entry and ending with the DeviceID-1
            // entry.
            if (DeviceID > curDeviceID) {
                for (each deviceID in [curDeviceID..DeviceID-1]) {
                    rioRouteAddEntry(0, RIO_GEN_DFLT_DID, hopCount, deviceID,
                        portNumber);
                }

                // Synchronize the current switch routing table with the global table
                for (each deviceID in [curDeviceID..DeviceID-1]) {
                    Switches[curSwitchID].RouteTable.LFT[deviceID] = portNumber;
                }

                // Update the associated Device ID in the path.
                Switches[curSwitchID].DeviceID = curDeviceID;
            }   // end if
        }   // end else if
    }   // end for
} // end if (ProcessingElementFeatureCAR.Switch == TRUE)

return RIO_SUCCESS;

}   // end rioEnumerateNeighbor

// ************************************************************************
// System clear up
// —Reset the previously acquired lock because a master exists elsewhere. Use hostDeviceID to
// reset the lock

STATUS rioClearUp (hostDeviceID) {

// Clear the host lock
if (hostDeviceID > DeviceID –1) {
    rioReleaseDeviceLock (0, hostDeviceID, 0, hostDeviceID);
}

// Clear the discovered end point device lock
while (DeviceID >= 1) {
    rioReleaseDeviceLock (0, DeviceID-1, 0, hostDeviceID);
    DeviceID --;
}

// Clear the discovered switch device lock
while (SwitchID >= 1) {
    rioReleaseDeviceLock (0, Switches[SwitchID–1].DeviceID,
                          Switches[SwitchID-1].hopCount, hostDeviceID);
    SwitchID --;
}

return RIO_SUCCESS;
}   // end rioClearUp
```

A.4 System Bring Up Example

This section walks-through a system bring up example. The system described in this example is shown in Figure A-1.

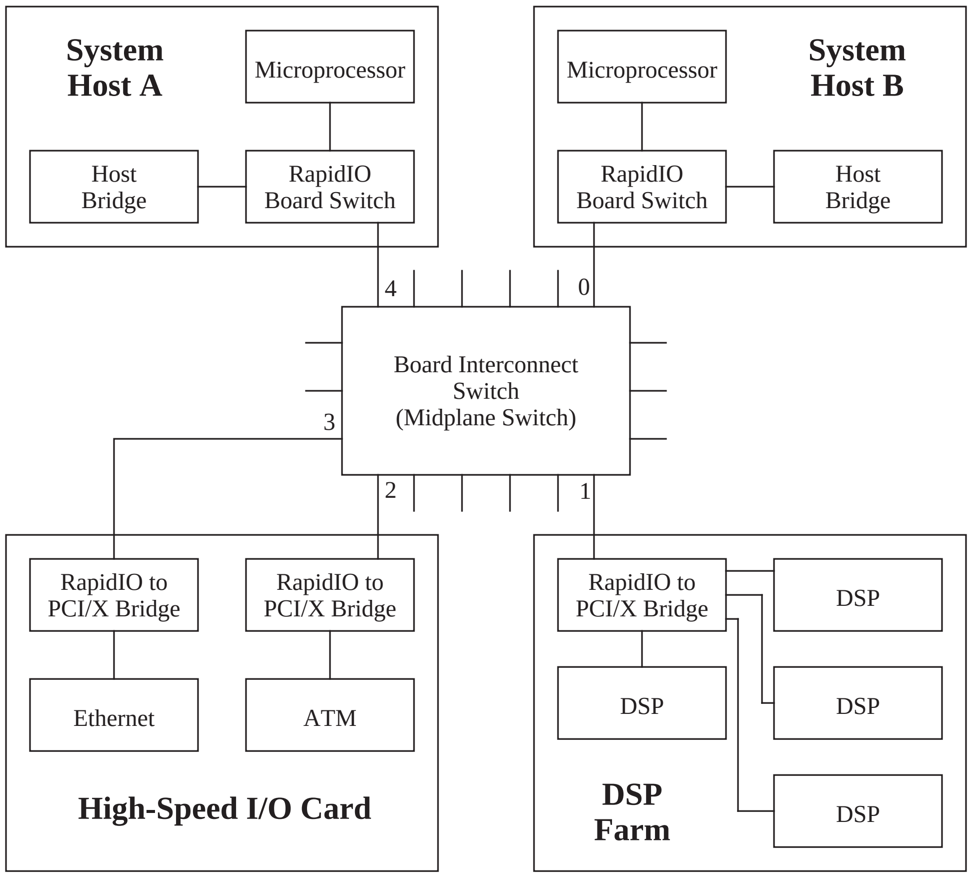
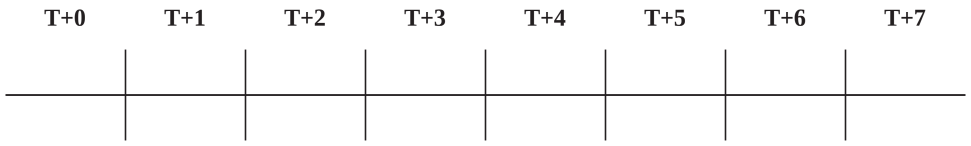


Figure A-1. Example System

Referring to Figure A-1, system Host A is preloaded with device ID 0x00 and system Host B is preloaded with device ID 0x01. Host A is configured to accept maintenance packets with destination IDs of 0x00 and 0xFF. Host B is configured to accept maintenance packets with destination IDs of 0x01 and 0xFF. System Bring Up advances through time slots along the following timeline:



The time slots shown above are defined as follows:

- **T+0:** Host A begins RapidIO enumeration.
- **T+1:** Host B begins RapidIO enumeration and Host A continues RapidIO enumeration.
- **T+2:** Host B discovers another host in the system (Host A) and waits.
- **T+3:** Host A discovers a higher priority host in the system (Host B) and retreats.

- **T+4:** Host B assumes sole enumeration of the system.
- **T+5:** Host B enumerates the PE on switch port 1.
- **T+6:** Host B enumerates the PEs on switch ports 2, 3 and 4.
- **T+7:** System enumeration is complete.

The following describes the actions taken during each time slot in more detail:

Time T+0

Host A attempts to acquire the lock from its Host Base Device ID Lock CSR by writing 0x00 to the CSR. Host A confirms it has acquired the lock when it reads the value of 0x00 (the host device ID) from the Lock CSR. Host A continues by reading the Processing Element Features CAR and adding the information from the CAR to its RapidIO device database. Host A updates its Base Device ID CSR with the host device ID (0x00).

Time T+1

Host B attempts to acquire the lock from its Host Base Device ID Lock CSR by writing 0x01 to the CSR. Host B confirms it has acquired the lock when it reads the value of 0x01 (the host device ID) from the Lock CSR. Host B continues by reading the Processing Element Features CAR and adding the information from the CAR to its RapidIO device database. Host B updates its Base Device ID CSR with the host device ID (0x01).

Host A begins neighbor enumeration. It attempts to acquire the lock from the Host Base Device ID Lock CSR of the Board Interconnect Switch. A maintenance write of the host device ID (0x00), the destination device ID (0xFF), and the hop count (0) is issued for the Lock CSR. Host A confirms it has acquired the lock when it reads the value of 0x00 (the host device ID) from the Lock CSR.

Time T+2

Host B begins neighbor enumeration. It attempts to acquire the lock from the Host Base Device ID Lock CSR of the Board Interconnect Switch. A maintenance write of the host device ID (0x01), the destination device ID (0xFF), and the hop count (0) is issued for the Lock CSR. However, after Host B issues a maintenance read from the Lock CSR it finds that the device was already locked by host device ID 0x00. Because Host B has a higher priority than the current lock holder (0x01 is greater than 0x00), Host B spins in a delay loop and repeatedly attempts to acquire the lock.

Time T+3

Host A continues neighbor enumeration. It issues a maintenance read cycle to the Device Identity CAR of the Board Interconnect Switch and looks for a matching entry in the device database. Device configuration continues because no match is found (Host A has not enumerated the device). Host A reads the Source Operations and Destination Operations CARs for the device. It is determined that the device does not support read/write/atomic operations and does not require a device ID. Host A reads the Processing Element Feature CAR for the device and determines that it is a switch element.

Because the device is a switch, Host A reads the Switch Port Information CAR and records the device identity in the switch database. Next, Host A adds a set of entries to the switch's routing table. For each previously discovered device ID, an entry is created containing a target ID (0xFF), hop count (0), and the route port number (from the Switch Port Information CAR). The switch database is updated with the same routing information. Host A reads the Port Error Status CSR for switch port 0, verifying that it is possible for the port to have a neighbor PE. An entry is created in the switch’s routing table containing target ID (0xFF), hop count (0), and the route port number (0).

Host A continues neighbor enumeration using a hop count of 1. It attempts to acquire the lock from the Host Base Device ID Lock CSR of the neighbor PE on port 0. A maintenance write of the host device ID (0x00), the destination device ID (0xFF), and the hop count (1) is issued for the Lock CSR. However, after Host B issues a maintenance read from the Lock CSR it finds that the device was already locked by host device ID 0x01. Because Host A has a lower priority than the current lock holder (0x00 is less than 0x01), Host A retreats. It begins the process of backing out all enumeration and configuration changes it has made.

Host A checks its device and switch databases to find all host locks it obtained within the system (System Host A and the Board Interconnect Switch). It issues a maintenance write transaction to their Host Base Device ID Lock CSRs to release the locks.

Time T+4

As Host B spins in its delay loop, it attempts to acquire the lock from the Host Base Device ID Lock CSR of the Board Interconnect Switch. A maintenance write of the host device ID (0x01), the destination device ID (0xFF), and the hop count (0) is issued for the Lock CSR. Because Host A released the lock, Host B is able to confirm it has acquired the lock when it reads the value of 0x01 from the Lock CSR.

Host B continues neighbor enumeration. It issues a maintenance read cycle to the Device Identity CAR of the Board Interconnect Switch and looks for a matching entry in the device database. Device configuration continues because no match is found (Host B has not enumerated the device). Host B reads the Source Operations and Destination Operations CARs for the device. It is determined that the device does not support read/write/atomic operations and does not require a device ID. Host B reads the Processing Element Feature CAR for the device and determines that it is a switch element.

Because the device is a switch, Host B reads the Switch Port Information CAR and records the device identity in the switch database. Next, Host B adds a set of entries to the switch's routing table. For each previously discovered device ID, an entry is created containing a target ID (0xFF), hop count (0), and the route port number (from the Switch Port Information CAR). The switch database is updated with the same routing information. Host B reads the Port Error Status CSR for switch port 0, verifying that it is possible for the port to have a neighbor PE. An entry is created in the switch's routing table containing target ID (0xFF), hop count (0), and the route port number (0). Host B detects that it is attached to port 0. Because Host B has already been enumerated, neighbor enumeration continues on the next port.

Time T+5

Host B reads the Port Error Status CSR for switch port 1, verifying that it is possible for the port to have a neighbor PE. An entry is created in the switch's routing table containing target ID (0xFF), hop count (0), and the route port number (1).

Host B continues neighbor enumeration using a hop count of 1. It attempts to acquire the lock from the Host Base Device ID Lock CSR of the neighbor PE on port 1. A maintenance write of the host device ID (0x01), the destination device ID (0xFF), and the hop count (1) is issued for the Lock CSR. Host B confirms it has acquired the lock when it reads the value of 0x01 from the Lock CSR.

Host B issues a maintenance read cycle to the Device Identity CAR of the DSP Farm and looks for a matching entry in the device database. Device configuration continues because no match is found (Host B has not enumerated the device). Host B reads the Source Operations and Destination Operations CARs for the device. It is determined that the device supports read/write/atomic operations. A maintenance write is used to update the Base Device ID CSR with the value of 0x00 (the first available device ID). DeviceID is incremented and compared with the Host B device ID. Because they are equal, deviceID is assigned the next available device ID.

Time T+6

The process described in the previous step (Time T+5) is repeated on switch ports 2–4. Device IDs 0x02, 0x03, and 0x04 are assigned to the PEs on switch ports 2, 3 and 4, respectively.

Time T+7

Host A detects that its Host Base Device Lock CSR has been acquired by another host device, indicating it has been enumerated. Host A can initiate passive discovery to build a local system database.

Glossary of Terms and Abbreviations

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

A

Application programming interface (API.). A standard software interface that promotes portability of application programs across multiple devices.

C

Capability registers (CARs). High-speed memory containing recently accessed data and/or instructions (subset of main memory) associated with a processor.

Command and status registers (CSRs). A set of registers that allows a processing element to control and determine the status of another processing element's internal hardware.

D

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.

Discovery. The passive exploration of a RapidIO network fabric. This process involves walking an already enumerated RapidIO fabric to determine network topology and resource allocations.

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

E

End point. A processing element which is the source or destination of transactions through a RapidIO fabric.

End point device. A processing element which contains end point functionality.

Enumeration. The active exploration of a RapidIO network fabric. This process involves configuring device identifiers and maintaining proper host locking.

H

Hardware abstraction layer (HAL). A a standard software interface to device-specific hardware resources.

I

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

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.

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.

S

Sender. The RapidIO interface output port on a processing element.

Source. The origin of a packet on the RapidIO interconnect, also referred to as an initiator.

Switch. A multiple port processing element that directs a packet received on one of its input ports to one of its output ports.

T

Target. The termination point of a packet on the RapidIO interconnect, also referred to as a destination.

Transaction. A specific request or response packet transmitted between end point devices in a RapidIO system.

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.

RapidIO™ Interconnect Specification Annex 2: Session Management Protocol Specification

3.2, 1/2016

RapidIO.org

Revision History

Revision	Description	Date
2.0	First release	06/14/2007
2.0	Public release	03/06/2008
2.1	No technical changes	07/09/2009
2.1	Removed confidentiality markings for public release	08/13/2009
2.2	No technical changes	05/05/2011
3.0	Changed RTA contact information. No technical changes	10/11/2013
3.1	No technical changes.	09/18/2014
3.2	No technical changes.	01/28/2016

Questions regarding RapidIO.org, specifications, or membership should be forwarded to:

RapidIO.org
8650 Spicewood Springs #145-515
Austin, TX 78759
512-827-7680 Tel.

Table of Contents

Chapter 1 Overview

1.1 Introduction........ 11
1.2 Overview........ 11
1.3 Features of the Session Management Protocol........ 12
1.4 Contents........ 12
1.5 Terminology........ 13
1.6 Conventions........ 14
1.7 Useful References........ 14

Chapter 2 Managing Data Streams

2.1 Introduction........ 15
2.2 System Example........ 15
2.3 Establishing Data Streams........ 16
2.4 Data Streaming System Configurations........ 17

Chapter 3 Session Management Operation

3.1 Introduction........ 19
3.2 Initialization of Session Management Advertisement CSRs........ 19
3.3 Contacting a Participating End point........ 20
3.4 Establishing Conduits........ 21
3.4.1 Master/Slave Configuration Conduit Establishment........ 22
3.4.2 Peers Configuration Conduit Establishment........ 23
3.4.3 Conduit Establishment Algorithm........ 24
3.5 Management Messages........ 26
3.5.1 Session Management Message Types........ 26
3.5.1.1 REQUEST........ 26
3.5.1.2 ADVERTISE........ 26
3.5.1.3 OPEN........ 27
3.5.1.4 ACCEPT........ 27
3.5.1.5 REFUSE........ 27
3.5.1.6 FLOW-CONTROL........ 27
3.5.1.7 DATA........ 27
3.5.1.8 CLOSE........ 27
3.5.1.9 STATUS........ 28
3.5.2 Message Header Fields........ 28
3.5.2.1 Command Header Field: <CMD><VER>........ 28
3.5.2.2 SourceID and DestID........ 28
3.5.2.3 Protocol Identifier: <ProtoID>........ 29
3.5.2.4 Class of Service: <COS>........ 29

Table of Contents

3.5.2.5 Stream Identifier: <StreamID> 29
3.5.3 Session Management Protocol Attributes 30
3.5.3.1 VENDOR Attribute 31
3.5.3.2 DATA_OFFSET_VENDOR Attribute 31
3.5.3.3 DATA_OFFSET Attribute 32
3.5.3.4 REQUEST_RETRY_PERIOD Attribute 32
3.5.3.5 REQUEST_TIMEOUT_PERIOD Attribute 32
3.5.3.6 FLOW_CONTROL_XON_TIMEOUT_PERIOD Attribute 33
3.5.3.7 OPEN_MESSAGE_NUMBER Attribute 33
3.5.3.8 CONDUIT_STREAM Attribute 33
3.5.3.9 DATA_HEADER_FORMAT Attribute 34
3.5.3.10 CONVEYANCE Attribute 34
3.5.3.11 Other Attributes 34
3.6 Message Sequence Examples 35
3.6.1 Stream Initiation 35
3.6.2 Refusal to Initiate a Stream 35
3.6.3 Stream Shutdown 36
3.6.4 Uses of the STATUS command 37
3.6.5 Use of the FLOW_CONTROL Command 38
3.7 Session Management Error Conditions and Recovery 39
3.7.1 Message Loss 39
3.7.2 Session Management Protocol Congestion Management 40
3.7.3 Session Management Protocol Non-Compliance 40
3.8 Rules for Session Management 41
3.8.1 Optional Features 41
3.8.2 Attribute Related Rules 41
3.8.3 Rules Related to Virtual Stream Status 42
3.8.4 Rules Related to Vendor-Specific Commands 42
3.8.5 Rules Related to Reserved Fields 43
3.9 Notes on Optional Features and Inter-Operability 43
3.9.1 Optional Attributes 43
3.9.2 REQUEST and ADVERTISE 44

Chapter 4 Message Format Descriptions

4.1 Introduction 45
4.2 Control Message Formats 45
4.2.1 REQUEST 45
4.2.2 ADVERTISE 46
4.2.3 OPEN 47
4.2.4 ACCEPT 48
4.2.5 REFUSE 49
4.2.6 FLOW_CONTROL 49
4.2.7 CLOSE 50
4.2.8 STATUS 51
4.2.9 User Defined 52

Table of Contents

4.3 Data Formats 53
4.3.1 DATA Message Format, MAILBOX 53
4.3.2 DATA1 Message Format, Large PDU 54
4.3.3 DATA2 Message Format 54
4.3.4 DATA3 Zero-length DATA header 54
4.3.5 Data Streaming 55

Chapter 5 Registers

5.1 Introduction 57
5.2 Session Management Protocol Extended Features Register Block 58
5.2.1 Session Management Protocol Register Block Header (Block Offset 0x0) 58
5.2.2 Session Management Protocol Register Write Enable CSR (Block Offset 0x4) 59
5.2.3 Session Management Advertisement CSR (Block Offset 0x8) 60
5.2.4 Session Management Attribute Range CSR (Block Offset 0xC) 61
5.2.5 Session Management Protocol Attributes 0-508 CSRs (Block Offset 0x10-0x7F8) 63
5.3 Component Tag CSR Session Management Protocol Advertisement 64

Chapter 6 Vendor-Defined Protocols

6.1 ProtoID 67
6.2 Attributes 67
6.2.1 VENDOR attribute 67
6.2.2 PROTOCOL_NAME attribute 67
6.2.3 Other attributes 67
6.3 Other Requirements for Vendor-Defined Protocols 67

Chapter 7 Ethernet Encapsulation

7.1 ProtoID 69
7.2 Attributes 69
7.2.1 MTU Attribute 69
7.2.2 CONVEYANCE Attribute 69
7.2.3 MAC_ADDRESS Attribute 69
7.3 Other Requirements of Ethernet Encapsulation 69
7.3.1 Dropped Messages 70
7.3.2 Broadcast 70
7.3.2.1 Broadcast With Multicast Extensions 70
7.3.2.2 Broadcast Without Multicast Extensions 70

Table of Contents

7.3.2.3 Vendor defined Broadcast Server 70
7.3.3 Ingress/Egress Nodes 70

List of Figures

1-1 Data Streaming....11
2-1 Example of a RapidIO-Based Networking System 15
2-2 Stream Process....16
3-1 Normal Stream Initiation 35
3-2 Use of REFUSE Command 35
3-3 Normal Stream Shutdown....36
3-4 Use of the STATUS Command 37
3-5 Use of the FLOW_CONTROL Command 38

List of Figures

Blank page

List of Tables

2-1 Data Streaming System Configurations....17
3-1 StreamID Assignments....29
3-2 System Management Protocol Attribute Sizes....30
3-3 Vendor-Specific Attribute Ranges....30
3-4 System Management Protocol Attribute Sizes....31
3-5 System Management Protocol Attribute Sizes....31
3-6 DATA_OFFSET Attribute Format....32
3-7 DATA_HEADER_FORMAT Attribute Values....34
3-8 Ethernet Encapsulation Conveyance....34
4-1 REQUEST Message Format....45
4-2 ADVERTISE Message Format - Protocol Attributes....46
4-3 ADVERTISE Message Format - Protocol List....47
4-4 OPEN Message Format....48
4-5 ACCEPT Message Format....48
4-6 REFUSE Message Format....49
4-7 FLOW_CONTROL Message Format....50
4-8 CLOSE Message Format....50
4-9 STATUS Message Format....51
4-10 Status Bit Values....51
4-11 USERDEFINED Message Format....52
4-12 DATA Message Format....53
4-13 DATA Message Format....54
4-14 DATA Message Format....54
5-1 Bit Settings for Session Management Protocol Register Block Header....58
5-2 Bit Settings for Session Management Protocol Register Write Enable Register....59
5-3 Bit Settings for Session Management Protocol Advertisement Register....60
5-4 Bit Settings for Session Management Attribute Range Register....61
5-5 Bit Settings for Session Management Protocol Attributes 0-508 Registers....63
5-6 Component Tag CSR Bit Usage....64

List of Tables

Blank page

Chapter 1 Overview

1.1 Introduction

The Session Management Protocol permits system software to establish, manage, and remove *virtual streams* as defined in *RapidIO Interconnect Specification Part 10: Data Streaming Logical Specification Rev. 2.1*. The data streaming protocol allows data of any format to be transported between two end points. The Session Management Protocol provides a method for two end points to negotiate the characteristics of the data stream and assign those characteristics so the receiving entity can use the appropriate software layers upon receiving the data stream.

1.2 Overview

A stream, is a unidirectional connection between two end points. Bidirectional traffic is carried over two streams, one in each direction.

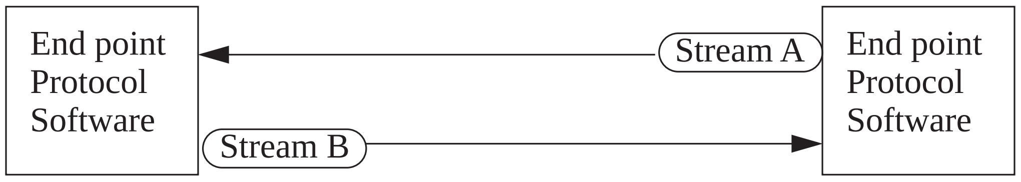


Figure 1-1. Data Streaming

The end points must have a common understanding of what the data within the stream is, and associate the stream with the right end point process. One of the principles of the data streaming protocol is the use of *virtual streams*. A virtual stream contains a generic tag called the *virtual stream ID (VSID)*. The VSID is assigned by the destination, based on the destination’s method for decoding, and may be a software or a hardware feature.

VSIDs are unique between any pair of source and destinations. The Session Management Protocol allows a source and destination to discover what protocols the two can use to communicate, assign a VSID to carry that protocol, establishing an open stream. Once open, a stream can carry any number of protocol data units (PDUs) until the stream is not longer needed. The Session Management Protocol is then used to close the stream.

The Session Management Protocol is intended to manage streams of communication. When data is passed using Type 9 (Data-Streaming Class) as the conveyance, the streamID used for Session Management Protocol is the same as the streamID used for Type 9 data traffic. When the data are passed using Type 11 (Message Class) as the conveyance, the streamID used for Session Management Protocol can be encapsulated in the command header of the DATA or DATA1 commands, as defined in Section 4.3.

1.3 Features of the Session Management Protocol

The Session Management Protocol provides the following features:

- A method to contact the session management software running on any end point.
- A method to discover which protocols an end point is capable of receiving.
- A method to discover and assign streams.
- A method to manage the status of streams.
- A method to close active streams.
- A method to handle errors and to handle protocol violations.

1.4 Contents

Following are the contents of the *RapidIO Interconnect Specification Annex 2: Session Management Protocol Specification Rev. 2.1*:

- Chapter 1, “Overview,” is an overview of the Session Management Protocol specification.
- Chapter 2, “Managing Data Streams,” introduces system issues such as discovery and transport configurations.
- Chapter 3, “Session Management Operation,” describes the set of messages defined by the protocol and their use.
- Chapter 4, “Message Format Descriptions,” contains the packet format definitions for the session management messages.
- Chapter 5, “Registers,” describes the visible register set that allows an external processing element to discover and contact a session management process on another end point.
- Chapter 6, “Vendor-Defined Protocols,” describes how to specify vendor-defined protocol attributes.
- Chapter 7, “Ethernet Encapsulation,” contains the specific use case requirement for tunneling Ethernet using this protocol and data streaming, as well as requirements for vendor-specific attributes.

1.5 Terminology

The following terms are used in this document. The terms are consistent with their usage in *RapidIO Interconnect Specification Part 10: Data Streaming Logical Specification*.

Refer to the Glossary at the back of this document for additional definitions.

Class of service - (cos) a term used to describe different treatment (quality of service) for different data streams. Support for class of service is provided by a class of service field in the data streaming protocol. The class of service field is used in the virtual stream ID and in identifying a virtual queue.

The value of the cos field is not defined by this specification, but is implementation dependent. The requirements for this field are that every message with a given cos must be transmitted with equal priority.

Conduit - A bidirectional data path, consisting of one stream for data transfer in each direction.

Conveyance - The RapidIO logical layer protocol used to transmit and receive data within a stream. This specification defines the use of RapidIO Type 11 (Message Class) and Type 9 (Data-Streaming Class) conveyances.

Egress - Egress is the device or node where traffic exits the system. The egress node also becomes the destination for traffic out of the RapidIO fabric. The terms egress and destination may or may not be used interchangeably when considering a single end to end connection.

Ingress - Ingress is the device or node where traffic enters the system. The ingress node also becomes the source for traffic into the RapidIO fabric. The terms ingress and source may or may not be used interchangeably when considering a single end to end connection.

Process - When a node communicates with a remote, some element of execution is responsible for managing the data communication. In this specification such element of execution is referred to as a process. No implication is intended regarding the internal structure of the operating system or other system organization.

Protocol Data Unit - (PDU) A self contained unit of data transfer comprised of data and protocol information that defines the treatment of that data.

Virtual Stream ID - (VSID) an identifier comprised of several fields in the protocol to identify individual data streams. When using Type 9 (Data-Streaming Class) as the conveyance for data transfers, the VSID is encapsulated in the Type 9 protocol. When using Type 11 (Message Class) as the conveyance for data transfers, the VSID is encapsulated in fields in the DATA or DATA1 Session Management Protocol commands.

StreamID - a specific field in the data streaming protocol that is combined with the data stream's transaction request flow ID and the source ID or destination ID from the underlying packet transport fabric to form the virtual stream ID.

Segment - A portion of a PDU.

Segmentation - a process by which a PDU is transferred as a series of smaller segments.

Segmentation context - Information that allows a receiver to associate a particular packet with the correct PDU.

Suspect - A node which, for some reason, is not behaving according to the specification. See "Section 3.7, Session Management Error Conditions and Recovery" on page 39 for more information.

1.6 Conventions

All fields and message formats are described using big endian format.

||	Concatenation, used to indicate that two fields are physically associated as consecutive bits
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.
x	This value is a don't care.
<variable>	Identifies a logical variable that may be a specific field of a register or packet or data structure.

1.7 Useful References

RapidIO Interconnect Specification Part 2: Message Passing Logical Specification

RapidIO Interconnect Specification Part 3: Common Transport Specification

RapidIO Interconnect Specification Part 10: Data Streaming Logical Specification

Chapter 2 Managing Data Streams

2.1 Introduction

Data streaming provides a common layer in RapidIO that allows any protocol to simultaneously share the same physical transport with any and all other protocols. The system can be comprised of many end points, supporting multiple protocols, and utilizing a variety of hardware and software acceleration features. Using data streaming and this management protocol, any standard or proprietary protocol data format can be tunneled on a RapidIO fabric.

2.2 System Example

Figure 2-1 shows an example of a system using two methods to perform file transfers, one interworking with an Ethernet bridge and one tunneling FTP directly.

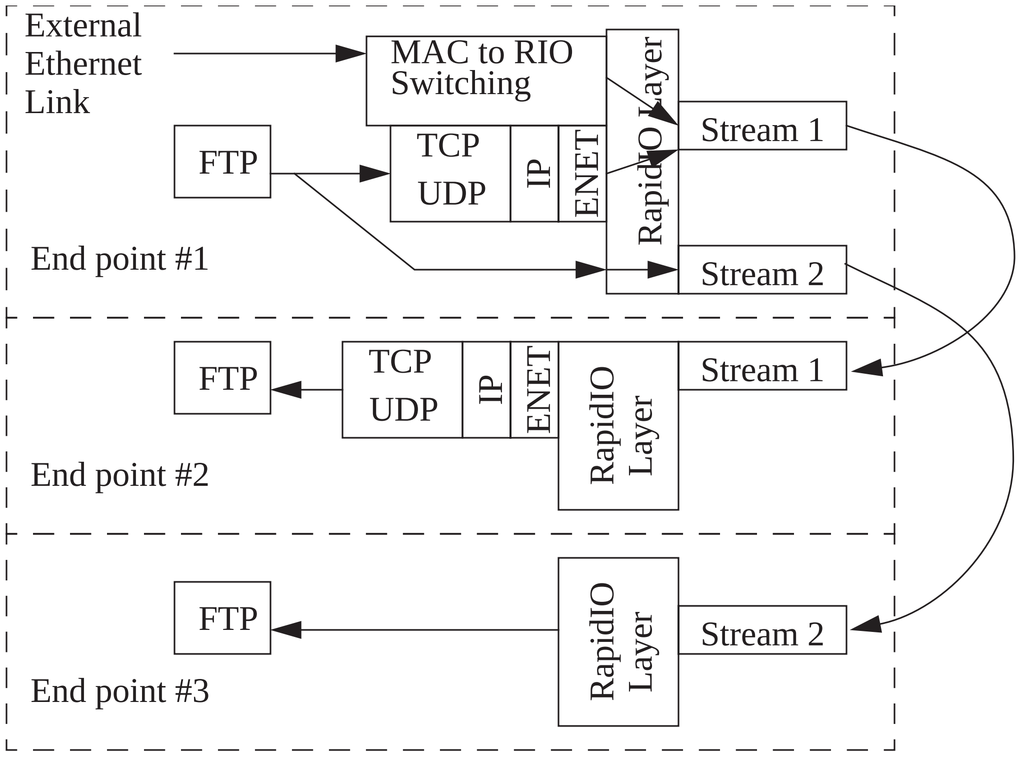


Figure 2-1. Example of a RapidIO-Based Networking System

2.3 Establishing Data Streams

The data streaming process is separated into two phases, with the management phase separated from the data phase. All the information about the stream is exchanged separately from the actual data transfer. Once established, a stream can support many data transfers. The data transfer only contains the information necessary to recover the original data. The received data must then be *associated* with an end process.

The method of association makes use of both the streamID and class of service (cos) to identify the stream. For example, the streamID can be linked directly to a given process which receives information for that stream. The cos may be used to determine the real time behavior necessary for the data, for example, guaranteed latency of *X* for responses.

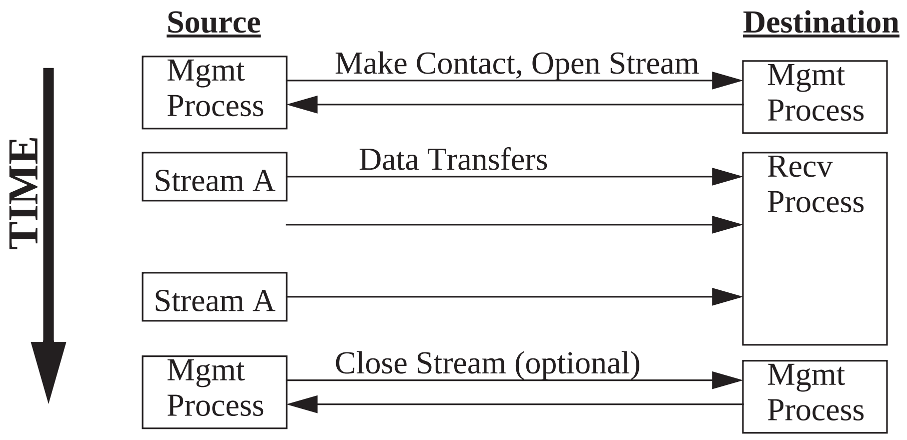


Figure 2-2. Stream Process

The Session Management Protocol begins with a discovery process. That discovery process detects which end points can perform session management negotiations. The discovery process is described in “Section 3.3, Contacting a Participating End point” on page 20.

Once two management end points are connected, they exchange a series of Session Management Protocol messages to discover the data streaming capabilities of the other. If both have the right capabilities, then one or more streams are opened for a given protocol.

With an open stream data can be transferred at any time. A stream may be persistent, existing even though there is no data to transfer at any given time.

Any stream may be closed by either end point, however most streams should remain open as long as the hardware and software might need to transfer data. If an end point is shutting down, or is terminating the process that handles data on that stream, it may close a stream.

As previously described, streams are unidirectional. However, many systems have requirements that communication be bidirectional. A bidirectional data path is created out of two streams, one in each direction. Such a bidirectional data path is called a conduit. When there is a requirement for bidirectional data transfer, one system may initiate the connection. The recipient of the initial negotiation is expected to start negotiation for the connection in the other direction. The resulting pair of streams should be grouped together, so that whenever one stream of the conduit is opened or closed, the other stream is treated the same. For more information, see “Section 3.4, Establishing Conduits” on page 21.

2.4 Data Streaming System Configurations

RapidIO Interconnect Specification Part 10: Data Streaming Logical Specification defines a logical layer for streaming data transfer. Hardware designed to this specification can transfer data using hardware or software resources to encapsulate data. The data streaming protocol uses a segmentation and reassembly protocol to manage variable sized PDUs.

Other methods may be used to transfer variable sized PDUs as long as they include the elements of segmentation and reassembly and contain a virtual stream ID. This specification also defines a method to encapsulate data using the *RapidIO Interconnect Specification Part 2: Message Passing Logical Layer Specification*.

The management protocol may also be run over a number of conveyances. The messages may be embedded in a predesignated stream in the data streaming logical layer, or it may be run over the message passing logical layer, even if the data is on the data streaming logical layer. Additional conveyances may be available as vendor-specific conveyances or in future versions of the RapidIO specifications. Table 2-1 shows the system configurations that are currently defined by this specification.

Table 2-1. Data Streaming System Configurations

Management	Data Transfers
Messaging (Type 11)	Messaging (Type 11)
Messaging (Type 11)	Data Streaming (Type 9)
Data Streaming (Type 9)	Messaging (Type 11)
Data Streaming (Type 9)	Data Streaming (Type 9)

The protocol first establishes how to contact the management process. Once contacted, the management processes then exchange the necessary information to transfer the data.

When a single node supports Session Management Protocol on multiple conveyances, the initial information required to establish a Session Management Protocol connection on each conveyance must be put in a separate Session Management Protocol Extended Features Register Block, as described in “Section 5.2, Session Management Protocol Extended Features Register Block” on page 58.

Therefore, multiple Session Management Protocol Extended Features Register Blocks may be required.

Chapter 3 Session Management Operation

3.1 Introduction

This chapter describes the RapidIO Session Management Protocol. The protocol consists of a sequence of messages to establish capabilities, open streams, manage streams, and close streams. The protocol includes methods for handling abnormal conditions.

3.2 Initialization of Session Management Advertisement CSRs

Before any management messages can be exchanged, an end point must first establish which end points support data streams, and discover how to contact their management process. The Session Management Protocol allows each end point to use its own resources as needed for the process of data streaming.

Participating end points place information in the *Session Management Advertisement CSR*, indicating that it supports the session management protocol, and identifies which conveyance to use to contact the management process. Legacy devices may also advertise participation in this protocol using the *Component TAG CSR* (see *RapidIO Interconnect Specification Part 3: Common Transport Specification* and “Section 5.3, Component Tag CSR Session Management Protocol Advertisement” on page 64).

On hardware power-up and on hardware reset, the Session Management Advertisement CSR and the Component Tag CSR must be initialized by hardware to indicate non-participation in Session Management Protocol. During software initialization, implementations conforming to this specification must indicate participation in the Session Management Protocol by modifying the Session Management Advertisement CSR, or optionally by modifying the Component Tag CSR if the Session Management Protocol Extended Features Register Block is not available.

The Session Management Advertisement CSRs, defined in “Section 5.2, Session Management Protocol Extended Features Register Block” on page 58, may be initialized by a processor which is part of the device implementing the Session Management Advertisement CSRs, by a processor remote from the device, or through a combination of the two. For example, protocol support related attributes could be initialized by the local processor, while system related attributes such as message timeout values could be initialized by a remote processor.

The Session Management Advertisement CSRs include a number of registers to facilitate initialization by local and remote processors. These are:

- “Section 5.2.2, Session Management Protocol Register Write Enable CSR (Block Offset 0x4)” on page 59
- “Section 5.2.4, Session Management Attribute Range CSR (Block Offset 0xC)” on page 61

The Session Management Protocol Register Write Enable CSR implements mutual exclusion between processors attempting to write to the attribute registers. If the Session Management Protocol Register Write Enable CSR is locked, other processors must respect the lock and not attempt to modify the attribute registers.

The Session Management Attribute Range CSR indicates how many attributes have been initialized. It also supports encoding up to 16 stages of initialization, to allow sequencing of the attribute initialization process.

For more information, refer to the definitions of the named registers.

3.3 Contacting a Participating End point

The *Session Management Advertisement CSR* contains the following information:

<Conveyance> indicates which conveyance can be used for the Session Management Protocol

If the conveyance is Type 11 (messaging) then the CSR has the following information:

- <Mailbox ID> identifies a mailbox dedicated for the reception of management messages.

If the conveyance is Type 9 (streaming) then the CSR has the following information:

- <StreamID> identifies a stream dedicated for the reception of management messages.
- <COS> identifies the class of service dedicated for the reception of management messages.

The conveyance used to create and manage all streams is specified in the Session Management Advertisement CSR, or in the Component Tag CSR if the Session Management Protocol Extended Features Register Block is not available. Unless otherwise specified, the conveyance used to transmit DATA and FLOW_CONTROL messages for a stream is the same as the conveyance used to create and manage all streams. Specification of other conveyances for DATA and FLOW_CONTROL traffic is performed by including the *CONVEYANCE* attribute as described in Section 3.5.3.10.

A special case may be needed when a system is required to create and manage streams using both Type 11 (messaging) and Type 9 (streaming). This will occur on mixed systems, where some nodes provide support for only Type 11 conveyance and other nodes provide support for only Type 9 conveyance. When necessary, this can be accomplished by use of the Session Management Advertisement CSR to contain contact information for Type 9 management, and using the Component Tag CSR to contain contact information for Type 11 management. Other configurations may be possible, but are implementation specific.

End points have two options to determine whether a remote node participates in this protocol. First, they may choose an implementation-specific mechanism, such as a built-in table, to check only for specified remote systems. Second, they may scan management space for all remote nodes, collecting information on participating end points. In either case, the end point is then expected to contact remote end points as appropriate, asking for their capabilities. See Chapter 5, “Registers”, for the bit definitions.

3.4 Establishing Conduits

Conduits consist of a pair of unidirectional streams for transferring data for a single protocol between two end points. In every conduit, one stream transfers protocol data in one direction while the other stream transfers protocol data in the other direction.

Conduits are established using the same command set as unidirectional streams. The Session Management Protocol requires that the sender initiate the process of opening a stream. One consequence of these two facts is that for conduits to be established, both end points are required to send an OPEN command, and there must be a mechanism to link the two streams into a single conduit. Linking the two streams into a conduit is accomplished by use of the *CONDUIT_STREAM* attribute during OPEN/ACCEPT negotiation. The *CONDUIT_STREAM* attribute is a 32-bit attribute, as described in Section 3.5.3.8. The data associated with this attribute consists of two streamIDs involved in the conduit.

One difficulty arises, based on whether a known end point is required to establish the conduit based on some external criteria, or whether it is possible for either end point to initiate establishment of the conduit.

The simple case, where one end point initiates establishment of the conduit, is referred to as a master/slave configuration. In this case, the master may send an OPEN message at any time to initiate establishment of the conduit, but the slave may only send the corresponding OPEN message after receiving the OPEN message from the master.

The more general case is referred to as a peers configuration. In this case, either node may send an OPEN message at any time to initiate establishment of the conduit. The peers configuration is more complex, because there must be a mechanism to handle the case when the two OPEN messages are in transmit at the same time.

It should be noted that an implementation capable of handling the peers configuration is capable of handling the master/slave configuration. For this reason, the algorithm for the peers configuration is presented in Section 3.4.3, but no algorithms for master/slave configuration are provided.

3.4.1 Master/Slave Configuration Conduit Establishment

In the master/slave configuration, one end point, the master, is always responsible for initiating conduit establishment. The other end point, the slave, completes conduit establishment in response to the establishment of the first conduit stream. The algorithm running on the master may not be identical to the algorithm running on the slave.

The master begins to establish the conduit by creating a local structure to contain information about the conduit. The first RapidIO transaction that the master makes is sending an OPEN message[1] containing the *CONDUIT_STREAM* attribute. The *CONDUIT_STREAM* attribute value is 0xFFFFFFFF in the first OPEN message when establishing a conduit, indicating that no previous negotiation has already been performed for this protocol between these two end points.

Upon receipt of an OPEN with *CONDUIT_STREAM* specified, the slave creates a local record containing information about the conduit. It responds to the OPEN with an ACCEPT message specifying a newly created streamID, which we refer to as 0xSSSS and *CONDUIT_STREAM* attribute value of 0xFFFFSSSS indicating that the slave, the sender of the ACCEPT message, will receive data on streamID 0xSSSS for this conduit.

To complete conduit creation, the slave then sends an OPEN message with the *CONDUIT_STREAM* attribute set to 0xFFFFSSSS to the master. The first field is the streamID that the slave will use to transmit data on, which has not yet been established. The second field is the streamID that the slave will use to receive data on, which has been assigned in the prior ACCEPT message.

The *CONDUIT_STREAM* attribute data contains the first field set to 0xFFFF and the second field set to a valid streamID. The fact that the *CONDUIT_STREAM* attribute is specified indicates that the OPEN message is related to a conduit. The value 0xFFFF in the first field indicates that the slave, the sender of the OPEN message, does not already know the streamID to use for transmitting data related to the conduit. The value of 0xSSSS in the second field indicates that the slave, the sender of the OPEN message, does already know the streamID to use for received data related to the conduit, and that the streamID is 0xSSSS.

The master, on receiving the ACCEPT message, reads the *CONDUIT_STREAM*

[1]REQUEST and ADVERTISE messages may have been previously exchanged. As these details add no relevant information to the discussion of establishing a conduit, they have been omitted.

attribute data, extracts the second field and finds 0xSSSS, and searches through its internal data for a conduit with matching streamID. The criteria for determining that the streamID is a match includes, but may not be limited to, the following tests. The matching streamID from the master's internal data must be part of a conduit. It must have the same value as specified, 0xSSSS. Finally, it must be the streamID that the master uses to transmit data on that conduit. If no such data record is found, the master may respond with REFUSE. However, when the master does find the data record associated with the conduit, it responds to the OPEN with ACCEPT, and updates its internal data structures. The ACCEPT message specifies a streamID on which the master will receive data, 0xMMMM, as well as the *CONDUIT_STREAM* attribute. The data contained in the *CONDUIT_STREAM* attribute consists of 0xSSSSMMMM.

The slave, on receiving the ACCEPT, searches through its internal data containing incomplete conduit records, finds a matching record, and updates its internal data structures.

3.4.2 Peers Configuration Conduit Establishment

In some distributed and/or reliable systems, it is necessary to allow each end of a conduit to attempt, simultaneously, to establish a conduit. The difference between the peers configuration and the master/slave configuration is that, in the peers configuration, an additional check is required before attempting to complete the conduit.

Assume that node M and node P are attempting to simultaneously create a conduit for a protocol. It is possible that node P receives node M's OPEN request before node P transmits is own OPEN request, and vice versa. When P receives the OPEN request, it checks to see if it has an outstanding OPEN request for establishing the same conduit. In this case, Node P does not, so the peers algorithm degenerates into the master/slave algorithm. Node P sends an ACCEPT response for node M's OPEN request, followed by node P's OPEN request to complete the creation of the conduit.

If node P receives node M's OPEN request after node P has transmitted its own OPEN request to node M, then when node P checks for outstanding OPEN requests for the creation of a conduit with the specified protocol with the other node, it will find one. Node P will respond with an ACCEPT message specifying the streamID that node P will receive data on using the *CONDUIT_STREAM* attribute, and will note that stream against its own attempt to establish the conduit. When node P receives the ACCEPT response for its own OPEN request for the conduit, both the transmit and receive stream IDs for the conduit will be known. Exactly the same procedure occurs on node M, so the conduit is established.

3.4.3 Conduit Establishment Algorithm

The following algorithm consists of three entry points, representing the procedure to call when processing incoming OPEN and ACCEPT messages, and the procedure for application code to initiate the procedure to establish a conduit. These are called process_incoming_open(), process_incoming_accept(), and create_conduit(), respectively. When working together, these three procedures create conduits.

There are several user-supplied procedures called from this algorithm. The function names should be self-explanatory, with the exception of find_partial_conduit(). This procedure searches through internal data structures for a conduit structure matching the specified remote nodeID and protocol, and with the streamIDs matching in the following manner. If the streamID specified in the call is 0xFFFF, then any streamID matches. If the streamID specified in the call is not 0xFFFF but the streamID in the local structure is 0xFFFF, then the streamID matches. If neither is 0xFFFF, then the values must be identical for them to match.

The algorithm specified here can be implemented for systems using a single execution thread and polled mode I/O. It does not indicate critical sections, which must be mutually exclusive. For multi-threaded OS implementations, the implementer must supply a locking mechanism to prevent concurrent access by other processes and/or interrupt service routines, and identify which portions of the algorithm need to be protected in their particular environment.

```
process_incoming_open(message)
        remote = get_sender(message)
        proto = get_protocol(message)
        rem_xmit = get_sender_transmit(message)
        rem_rcv = get_sender_receive(message)
        conduit = find_partial_conduit(remote, proto, rem_xmit, rem_rcv)
        if ( not found(conduit) )
                conduit = create_new_conduit(remote, proto)
                local_xmit = rem_rcv
                local_rcv = rem_xmit
                update_conduit_transmit(conduit, local_xmit)
        else
                local_xmit = get_conduit_transmit(conduit)
                local_rcv = get_conduit_receive(conduit)
        streamID = alloc_new_stream()
        update_conduit_receive(conduit, streamID)
        response = create_new_message()
        put_local_receive(response, streamID)
        put_local_transmit(response, local_xmit)
```

```
            send_accept(remote, response, streamID)
            if ( local_xmit == 0xFFFF )
                response = create_new_message()
                put_local_receive(response, streamID)
                put_local_transmit(response, 0xFFFF)
                update_conduit_flag(conduit, OPEN_SENT)
                send_open(remote, response)

process_incoming_accept(message)
        remote = get_sender(message)
        proto = get_protocol(message)
        rem_xmit = get_sender_transmit(message)
        rem_rcv = get_sender_receive(message)
        local_xmit = rem_rcv
        conduit = find_partial_conduit(remote, proto, rem_xmit, rem_rcv)
        if ( not found(conduit) )
            conduit = create_new_conduit(remote, proto)
            local_rcv = rem_xmit
            update_conduit_receive(conduit, local_rcv)
        else
            local_rcv = get_conduit_receive(conduit)
            if ( local_rcv == 0xFFFF )
                local_rcv = rem_xmit
                update_conduit_receive(conduit, local_rcv)
        update_conduit_transmit(conduit, local_xmit)
        flag = get_conduit_flag(conduti)
        if ( not flagIsSet(flag, OPEN_SENT) )
            response = create_new_message()
            put_local_receive(response, local_rcv)
            put_local_transmit(response, 0xFFFF)
            update_conduit_flag(conduit, OPEN_SENT)
            send_open(remote, response)

create_conduit(remote, proto)
        conduit = find_partial_conduit(remote, proto, 0xFFFF, 0xFFFF)
        if ( not found(conduit) )
            conduit = create_new_conduit(remote, proto)
```

```
update_conduit_receive(conduit, 0xFFFF)
update_conduit_transmit(conduit, 0xFFFF)
message = create_new_message()
put_local_transmit(message, 0xFFFF)
put_local_receive(message, 0xFFFF)
update_conduit_flag(conduit, OPEN_SENT)
send_open(remote, message)
```

3.5 Management Messages

The Session Management protocol consists of the following messages. The formats of the messages are defined in Chapter 4, “Message Format Descriptions,” on page 45.

Message names in this document use only capital letters to avoid confusion with non-message related use of the terms for message names.

Attribute names in this document are italicized and in uppercase, with individual words separated by underscore, to avoid confusion with non-attribute related use of the terms for attribute names.

3.5.1 Session Management Message Types

The following subsections list the messages used in the Session Management Protocol.

3.5.1.1 REQUEST

A REQUEST message is used to request information related to the protocols supported by the remote. The REQUEST can be for a list of supported protocols, or a list of attributes associated with a particular protocol.

For more information, refer to “Section 4.2.1, REQUEST” on page 45.

3.5.1.2 ADVERTISE

An ADVERTISE message is the response to a REQUEST message. If the REQUEST message specified that a list of protocols supported should be returned, the ADVERTISE message contains only a list of protocols.

If the REQUEST message specified a particular protocol, and the recipient of the REQUEST supported the protocol, the ADVERTISE message contains all required and optional attributes related to the protocol.

Optional attributes and vendor defined attributes may be negotiated. See Section 3.5.3 for discussion on this topic.

For more information, refer to “Section 4.2.2, ADVERTISE” on page 46.

3.5.1.3 OPEN

An OPEN message is sent to attempt to open a session on a target. The attributes specified in the OPEN message may or may not be acceptable by the target.

For more information, refer to “Section 4.2.3, OPEN” on page 47.

3.5.1.4 ACCEPT

An ACCEPT message is sent if the OPEN message is successful. The ACCEPT message specifies the stream to be used to identify the session, as well as the other attribute values which will govern the session.

OPEN messages may be sent to attempt to open multiple sessions with the same target and attributes. A target may optionally support only a single session for a given protocol and source, in which case attempts to OPEN multiple sessions will result in ACCEPT responses which all specify the same stream.

For more information, refer to “Section 4.2.4, ACCEPT” on page 48.

3.5.1.5 REFUSE

A REFUSE message is sent if the OPEN message is unsuccessful. The REFUSE message contains all the attributes in the OPEN request, in order to allow the OPEN sender to differentiate which OPEN request was refused.

For more information, refer to “Section 4.2.5, REFUSE” on page 49.

3.5.1.6 FLOW-CONTROL

FLOW-CONTROL messages start or stop transmission of DATA messages. It is not necessary for any implementation to send FLOW-CONTROL messages, however FLOW-CONTROL messages must always be supported when received.

For more information, refer to “Section 4.2.6, FLOW_CONTROL” on page 49.

3.5.1.7 DATA

DATA messages are used to transfer data using the message passing logical layer. There are several DATA messages, each with a different header format for use in different hardware and/or software environments.

For more information, refer to “Section 4.3, Data Formats” on page 53.

3.5.1.8 CLOSE

CLOSE messages are used to terminate the existence of a stream between a source and destination.

NOTE:

The stream may or may not exist at either the source or destination. After reception of a CLOSE message, the stream specified must no longer be used. After sending a CLOSE message, the system must be able to receive, without error, messages in transit at the time the CLOSE was sent. Such messages may be dropped.

When the stream is part of a conduit, the conduit should be closed. The means of determining that a stream is part of a conduit is implementation specific.

For more information, refer to “Section 4.2.7, CLOSE” on page 50.

3.5.1.9 STATUS

STATUS messages can be used to query status information related to a specified stream. In this form of the STATUS command, a streamID is included in the message, and other parts of the command indicate that the purpose is to query the status of the stream.

STATUS message are also used to provide status information, related either to a specified stream or to a specified command. A STATUS message is sent in three situations: as a response to a STATUS message querying status for a specific stream, indicating the current status of the stream; as a response to a CLOSE message, indicating that the CLOSE was successful, and as a response to an illegal, unknown, or malformed command, indicating that the command was not understood.

Note that the STATUS response to a STATUS query can include a query of the same stream.

For more information, refer to “Section 4.2.8, STATUS” on page 51.

3.5.2 Message Header Fields

All Session Management Protocol messages begin with a *command* header field, followed by one or more additional fields. The header fields and arrangement of the header fields are fixed for each message type.

3.5.2.1 Command Header Field: <CMD><VER>

The command header field consists of two octets, a command value denoted <CMD>, and a version value denoted <VER>. Each command value corresponds to one of the messages laid out in Section 3.5.1 on page 26.

<VER> for all commands described in this version of the specification, except the DATA1 command, must be set to a numeric value of 0x01. Receivers of commands must check <VER>. If a receiver receives an unrecognized message or a message other than STATUS with unknown <VER>, then the receiver must respond with a STATUS command with the Command_Unknown bit set. Recipients must not respond to STATUS messages with unknown <VER>.

3.5.2.2 SourceID and DestID

SourceID, denoted <SourceID>, is the RapidIO destination ID of the device which transmitted the message. DestID, denoted <DestID>, is the RapidIO destination ID for the device which should receive this message.

Both <SourceID> and <DestID> are 2 octets in size.

3.5.2.3 Protocol Identifier: <ProtoID>

The protocol identifier is used to specify what encapsulated protocol is being referenced by the Session Management message. The <ProtoID> is always 16 bits in size.

Ethernet encapsulation, as described below, uses protocol ID 0x0102.

Proprietary protocols use 0x0101 for all proprietary protocols. Different protocols are distinguished by protocol attributes, as described in Chapter 6, “Vendor-Defined Protocols”.

Protocol ID 0xFFFF is reserved for special usage within messages. Refer to the definition of individual message types for the use of this special value, if any.

3.5.2.4 Class of Service: <COS>

The Class of Service field can be used to specify the priority of a given stream or virtual stream. The value of the COS field is not defined by this spec, but can be implementation dependent. The requirements for this field are that every message with a given COS value must be transmitted with equal priority.

Class of Service, or <COS>, is always one octet in size.

3.5.2.5 Stream Identifier: <StreamID>

The stream identifier is the value used to identify the particular session for a given protocol between a <SourceID> and <DestID>. The combination of <SourceID><DestID><ProtoID><StreamID> must always be unique in the system.

The <StreamID> is always two octets in size.

StreamIDs are assigned by the recipient of an OPEN message. The value of the StreamID must conform to the criteria shown in Table 3-1.

Table 3-1. StreamID Assignments

StreamID value	Usage
0x0000 - 0xDFFF	Available for applications
0xE000 - 0xEFFF	Vendor specific
0xF000 - 0xFFFE	Reserved
0xFFFF	Invalid

StreamIDs marked “available for applications” in Table 3-1 must be opened with an OPEN message. StreamIDs with vendor specific values and reserved values are pre-defined and may be used without explicitly opening a stream. Definitions of reserved StreamIDs are defined in protocol specific chapters.

3.5.3 Session Management Protocol Attributes

Protocol attributes are relevant to individual virtual streams, and not to the session management protocol. Protocol attributes specify information about the stream, about the data transferred within the stream, or about how the data is to be represented on egress.

Each session management protocol attribute is encoded and made available to the remote system. Each attribute is encoded into an attribute field. Regardless of the data being identified, the attribute field always consists of a 64-bit (8-octet) value, consisting of two fields. The first field, Attribute ID, contains an identifier of the attribute. The second field, Attribute Value, contains the value associated with the attribute on a particular system. The fixed size allows for consistent parsing of attributes between multiple different protocols. It also allows attributes to be ignored if they are not understood by a given implementation. Another reason to have fixed size attributes is that it will simplify the implementation of hardware support for these attributes.

Attribute IDs can have one of three sizes: 8-bits, 16-bits, and 32-bits. The size of each attribute can be determined based on the first octet of the Attribute ID. The remaining bits in the attribute encoding are available for the Attribute Value. Attribute ID sizes are pre-defined, independent of protocol. Table 3-3 lists the sizes of each Attribute ID for all protocols.

Table 3-2. System Management Protocol Attribute Sizes

Attribute ID size	Attribute ID value
8 bits	0x00 - 0x7F
16 bits	0x8000 - 0xEFFF
32 bits	0xF0000000 - 0xFFFFFFFF

Attribute IDs marked as vendor specific are available for use by vendors for their own purposes. Table 3-3 shows the attribute ID ranges available for vendor use. All attribute ID ranges not explicitly assigned as general attributes, protocol-specific attributes, or vendor-specific attributes, are reserved and must not be used.

Table 3-3. Vendor-Specific Attribute Ranges

Attribute ID size	Vendor-Specific Range
8-bits	0x78-0x7F
16-bits	0xEF00 - 0xEFFF
32-bits	0xFE000000 - 0xFFFFFFFF

There are minimal ordering requirements for protocol attributes. All required attributes must occur ahead of all optional or vendor specific attributes. The order of any attributes required by a protocol may be specified by the protocol. Optional attributes may occur in any order. All optional attributes must occur after all required attributes, and before any vendor specific attributes. If vendor specific attributes are used, the first vendor specific attribute must begin with attribute 0x00, which identifies the vendor associated with the vendor specific attributes. Any further ordering requirements of vendor specific attributes may be defined by the vendor.

All implementations of the Session Management Protocol must include complete support for all standard attributes for every protocol supported. In the event that an optional or vendor specific attribute is not understood by an implementation, the session must not be opened.

3.5.3.1 *VENDOR* Attribute

The *VENDOR* attribute is an 8-bit attribute ID (0x00) with a 56-bit value. Only one *VENDOR* attribute may be specified in any single command.

The attribute value for the *VENDOR* attribute must conform to one of two formats:

OUI format: The three-octet OUI for the vendor defining the protocol may be assigned to the second, third, and fourth octets of the attribute, and the remaining four octets set to zero.

Table 3-4. System Management Protocol Attribute Sizes

AttributeID	OUI #1	OUI #2	OUI #3	Zero	Zero	Zero	Zero
0x01	0x00	0xA0	0x1E	0x00	0x00	0x00	0x00

NAME format: A seven-octet value, where every octet must contain a non-zero value. This format may be assigned to a seven-character representation of the ASCII value of the vendor company name.

Table 3-5. System Management Protocol Attribute Sizes

AttributeID	Char #1	Char #2	Char #3	Char #4	Char #5	Char #6	Char #7
0x01	0x52	0x61	0x70	0x69	0x64	0x49	0x4F

Vendors should use the OUI format if an OUI is available. If NAME format is used, the vendor is responsible for insuring that the choice of NAME does not conflict with any existing *VENDOR* attribute value.

NOTE:

The values specified above are for example only, and should not be used.

3.5.3.2 *DATA_OFFSET_VENDOR* Attribute

The *DATA_OFFSET_VENDOR* attribute is an 8-bit attribute ID (0x02) with a 56-bit value. The value must conform to the formats specified for the *VENDOR* attribute.

The *DATA_OFFSET_VENDOR* attribute identifies the vendor associated with the contents of the *DATA_OFFSET* Attribute.

The *DATA_OFFSET_VENDOR* attribute is optional, however, if the *DATA_OFFSET_VENDOR* attribute is specified, then the *DATA_OFFSET* attribute must follow it.

3.5.3.3 *DATA_OFFSET* Attribute

The *DATA_OFFSET* attribute is specified with the 16-bit attribute ID 0x8003, leaving six octets of data.

The *DATA_OFFSET* attribute specifies the number of octets of data that will be appended to the header of a Data message for a given stream. These octets are known as the Offset octets. The Offset octets can be used to convey information such as TCP/IP offload information, packet classification, and other vendor specific features. The remaining five octets in the *DATA_OFFSET* attribute are available for vendor specific information, labelled <VendorN>. The format of the *DATA_OFFSET* Attribute is as follows:

Table 3-6. *DATA_OFFSET* Attribute Format

attributeID	**Offset**	**Vendor0**	**Vendor1**	**Vendor2**	**Vendor3**	**Vendor4**
16-bits	8-bits	8-bits	8-bits	8-bits	8-bits	8-bits
0x8003	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF

DATA_OFFSET must be present if the *DATA_OFFSET_VENDOR* is specified.

If the *DATA_OFFSET_VENDOR* and *DATA_OFFSET* attributes are not included in an Open request, the default number of Offset octets is 0.

If the *DATA_OFFSET_VENDOR* and *DATA_OFFSET* attributes are present, the *DATA_OFFSET* attribute must follow the *DATA_OFFSET_VENDOR* attribute.

3.5.3.4 *REQUEST_RETRY_PERIOD* Attribute

The *REQUEST_RETRY_PERIOD* attribute is specified with the 32 bit attribute ID 0xF0000000.

The *REQUEST_RETRY_PERIOD* attribute indicates the time period, in microseconds, after which an REQUEST, OPEN, CLOSE, or STATUS request which has not received a response should be sent again.

The *REQUEST_RETRY_PERIOD* attribute default value is 250 milliseconds.

3.5.3.5 *REQUEST_TIMEOUT_PERIOD* Attribute

The *REQUEST_TIMEOUT_PERIOD* attribute is specified with the 32 bit attribute ID 0xF0000001.

The *REQUEST_TIMEOUT_PERIOD* attribute indicates the time period, in microseconds, after which an REQUEST, OPEN, CLOSE, or STATUS request which has not received a response should be judged to have failed.

The *REQUEST_TIMEOUT_PERIOD* attribute default value is 1 second.

3.5.3.6 *FLOW_CONTROL_XON_TIMEOUT_PERIOD* Attribute

The FLOW_CONTROL_XON TIMEOUT attribute is specified with the 32 bit attribute ID 0xF0000002.

The FLOW_CONTROL_XON TIMEOUT attribute value indicates the time period, in microseconds, from the time a FLOW CONTROL XOFF request is received until another FLOW CONTROL XOFF/XON message must be received. In the event that a FLOW CONTROL XOFF/XON message is not received in the timeout interval, transmission of the XOFF'ed stream should resume. This behavior is designed to detect the loss of FLOW CONTROL XON messages.

The FLOW_CONTROL_XON TIMEOUT attribute default value is 1 second.

3.5.3.7 *OPEN_MESSAGE_NUMBER* Attribute

The *OPEN_MESSAGE_NUMBER* attribute is specified with the 32 bit attribute ID 0xF0000003.

The *OPEN_MESSAGE_NUMBER* attribute is used by the originator of an OPEN message to identify the response to the OPEN request. This allows the originator to have multiple parallel OPEN requests in flight, and to be able to match responses to the requests.

The *OPEN_MESSAGE_NUMBER* attribute value is used in an implementation specific manner.

3.5.3.8 *CONDUIT_STREAM* Attribute

The *CONDUIT_STREAM* attribute is specified with the 32 bit attribute ID 0xF0000004.

The *CONDUIT_STREAM* attribute is used by both sides, in OPEN and ACCEPT messages, when creating a bidirectional conduit. The 32 bit value associated with the *CONDUIT_STREAM* attribute consists of the two stream IDs associated with the conduit. The first sixteen bits are used to indicate the stream ID of the stream which the sender of the OPEN or ACCEPT will use to transmit data on. The last sixteen bits are used to indicate the stream ID of the stream which the sender of the OPEN or ACCEPT will use to receive data on. The value of 0xFFFF for either field indicates that the specified stream ID was not known at the time the OPEN or ACCEPT message was sent.

See "Section 3.4, Establishing Conduits" on page 21 for more information, and examples of usage.

3.5.3.9 *DATA_HEADER_FORMAT* Attribute

The *DATA_HEADER_FORMAT* attribute is specified with the 32 bit attribute ID 0xF0000005.

The *DATA_HEADER_FORMAT* attribute is used in an ACCEPT message to indicate which DATA header should be used for DATA messages. The value is the numeric value of the DATA command, that is 0x06 indicates DATA Message Format, MAILBOX, as described in “Section 4.3.1, DATA Message Format, MAILBOX” on page 53, 0x09 indicates DATA1 message format for large PDUs, and so on.

Note that a special case exists for the DATA2 command. The implementation-specific value in the DATA2 header precedes the DATA2 command in the attribute value field, as shown in Table 3-7.

Table 3-7. *DATA_HEADER_FORMAT* Attribute Values

	8-bits	**8-bits**	**8-bits**	**8-bits**	**8-bits**	**8-bits**	**8-bits**	**8-bits**
DATA	0xF0	0x00	0x00	0x05	Reserved	Reserved	0x00	0x06
DATA1	0xF0	0x00	0x00	0x05	Reserved	Reserved	Implementation-Specific	0x09
DATA2	0xF0	0x00	0x00	0x05	Reserved	Reserved	0x00	0x0A
DATA3	0xF0	0x00	0x00	0x05	Reserved	Reserved	0x00	0x0B

When the *DATA_HEADER_FORMAT* attribute is not specified, the DATA command (0x06) must be used.

3.5.3.10 *CONVEYANCE* Attribute

The transmission channel is assigned with the 16-bit attribute ID 0x8001, leaving six octets for channel information. The channel is encoded in the first sixteen bits, leaving 32 bits for channel-specific information. The values for the conveyance are shown in Table 3-8.

Note that if a vendor-specific channel is used, the *VENDOR* attribute must be specified.

Table 3-8. Ethernet Encapsulation Conveyance

Channel (16 bits)	**Channel-specific information (32 bits)**
0x0000 = reserved	N/A
0x0001 = message	0x0000_00nn; nn indicates a mailbox
0x0002 - 0xFEFF = reserved	N/A
0xFF00 - 0xFFFF = vendor-specific	Vendor-defined

3.5.3.11 Other Attributes

For all protocols, system vendors may choose to define additional vendor-specific attributes not defined by the protocol. If additional protocol-specific attributes are used, the list of attributes must include the *VENDOR* attribute.

3.6 Message Sequence Examples

This section presents flow diagrams, to illustrate some typical uses of the Session Management command set.

3.6.1 Stream Initiation

Figure 3-1 shows the message sequence for initiation of a stream.

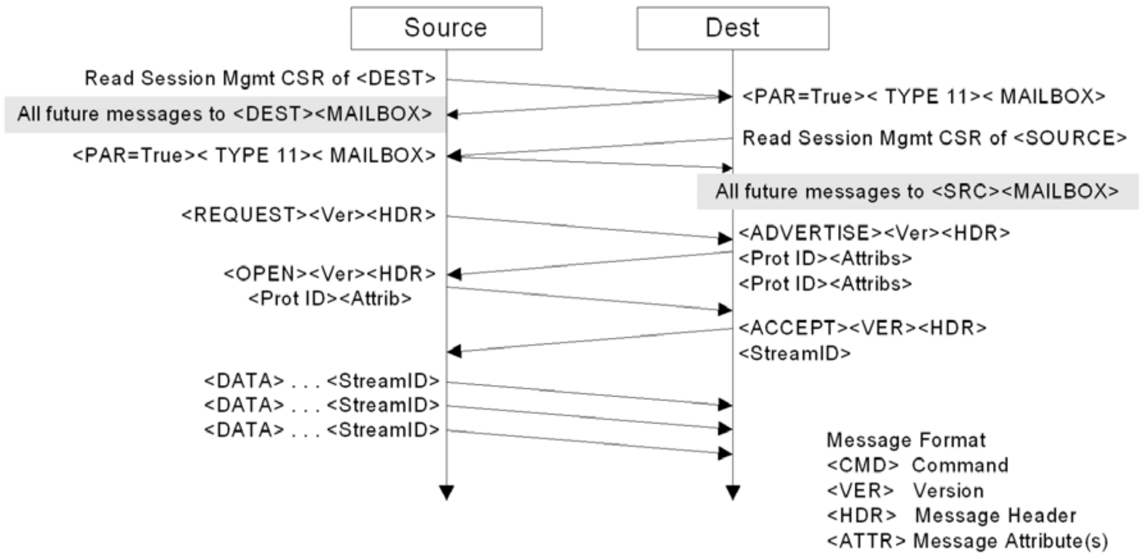


Figure 3-1. Normal Stream Initiation

3.6.2 Refusal to Initiate a Stream

Figure 3-2 shows the usage of a REFUSE command, when the recipient of an OPEN command does not allow the stream to be created.

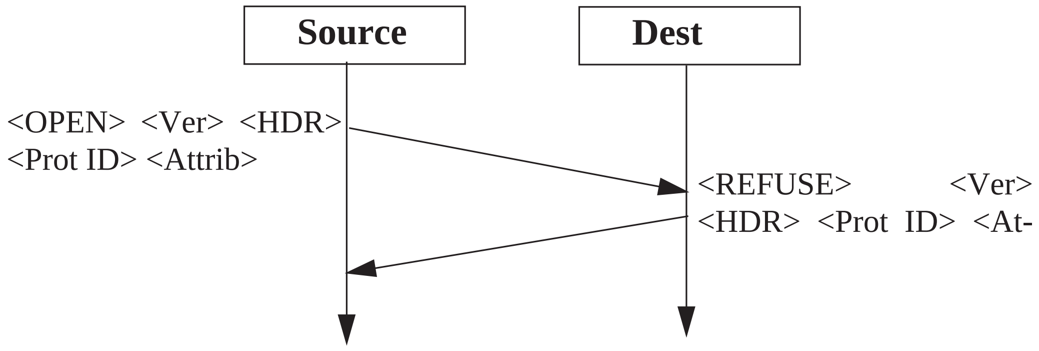


Figure 3-2. Use of REFUSE Command

3.6.3 Stream Shutdown

Figure 3-3 shows the sequence of commands for use during stream shutdown, when the initiator sends the CLOSE command. The receiver may also initiate the shutdown procedure.

Note that after the CLOSE message has been received, subsequent data must be discarded.

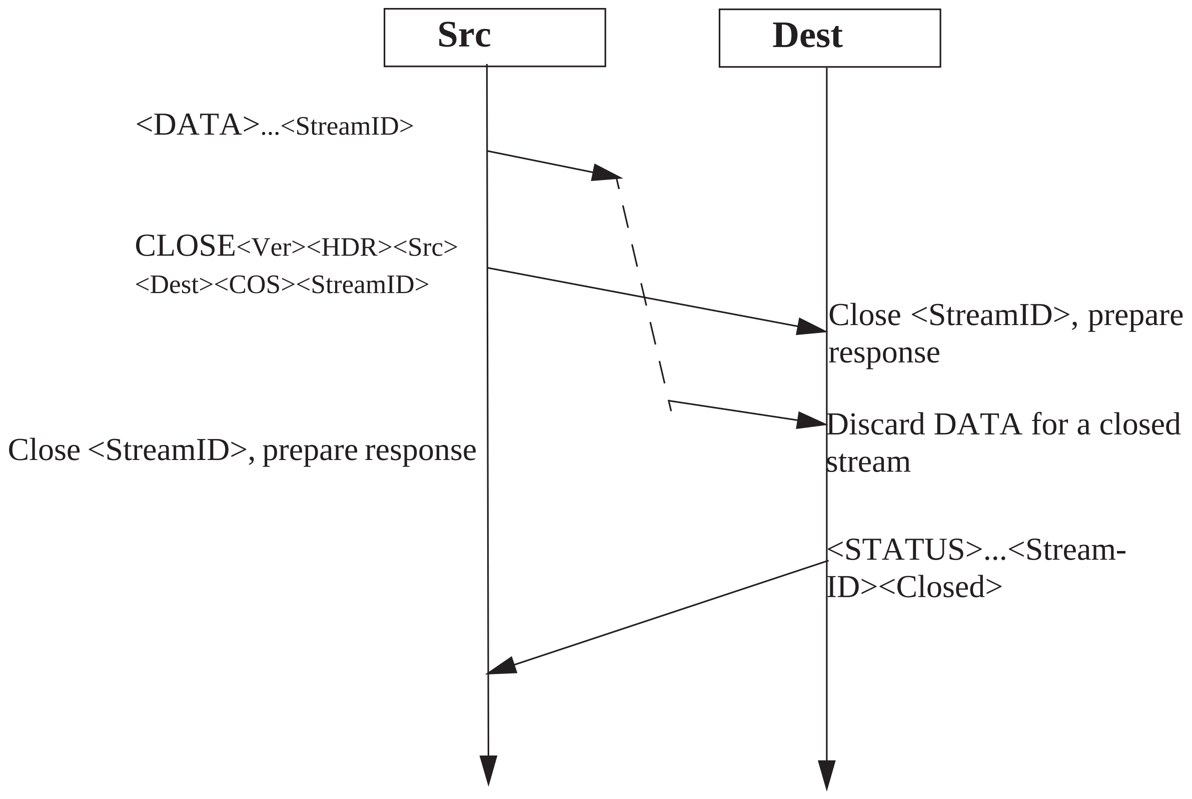

Src
Dest
<DATA>...<StreamID>
CLOSE<Ver><HDR><Src><Dest><COS><StreamID>
Close <StreamID>, prepare response
Discard DATA for a closed stream
Close <StreamID>, prepare response
<STATUS>...<StreamID><Closed>

Figure 3-3. Normal Stream Shutdown

3.6.4 Uses of the STATUS command

Figure 3-4 demonstrates several uses of the STATUS command.

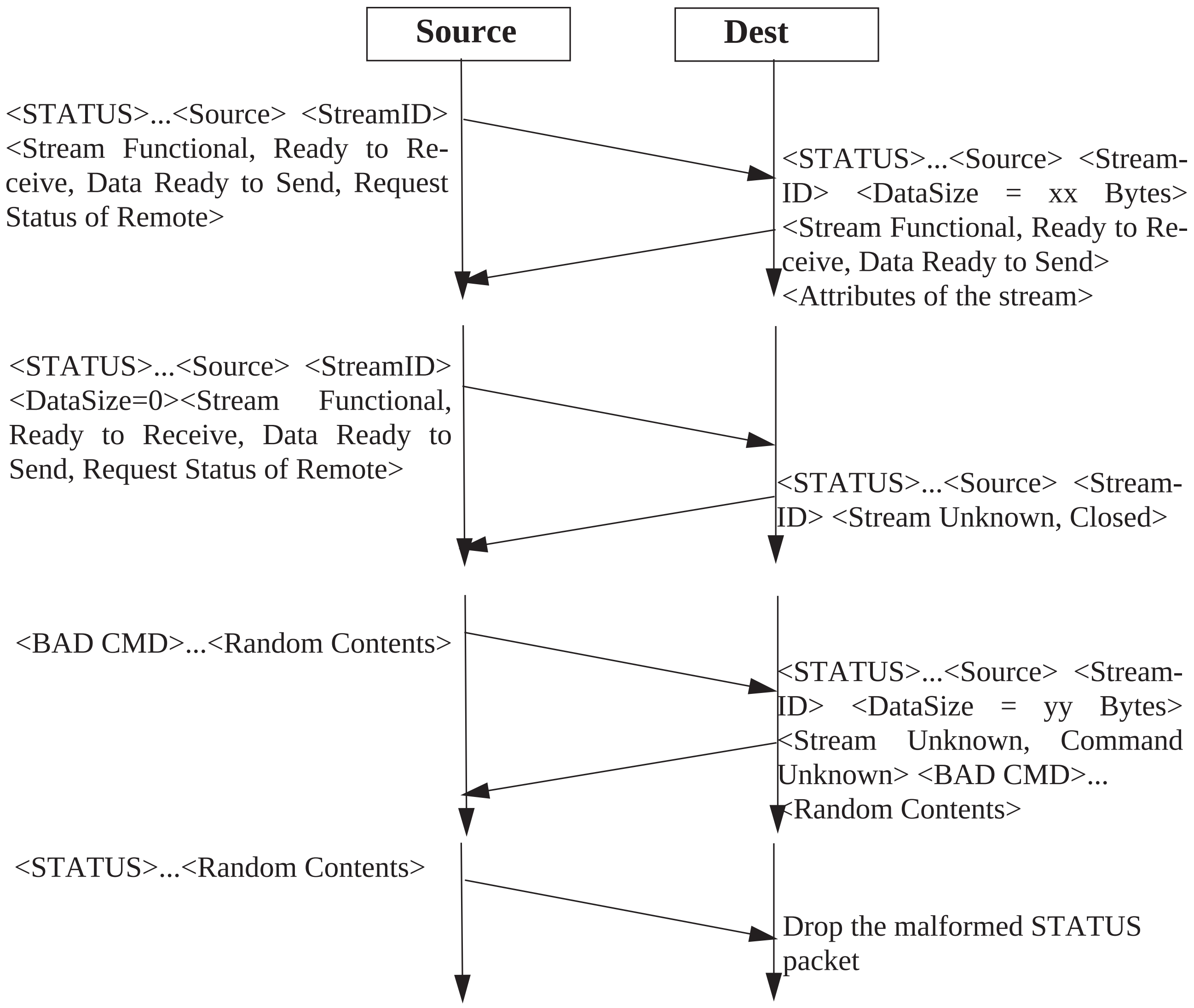

Source
Dest
<STATUS>...<Source> <StreamID> <Stream Functional, Ready to Receive, Data Ready to Send, Request Status of Remote>
<STATUS>...<Source> <StreamID> <DataSize = xx Bytes> <Stream Functional, Ready to Receive, Data Ready to Send> <Attributes of the stream>
<STATUS>...<Source> <StreamID> <DataSize=0><Stream Functional, Ready to Receive, Data Ready to Send, Request Status of Remote>
<STATUS>...<Source> <StreamID> <Stream Unknown, Closed>
<BAD CMD>...<Random Contents>
<STATUS>...<Source> <StreamID> <DataSize = yy Bytes> <Stream Unknown, Command Unknown> <BAD CMD>... <Random Contents>
<STATUS>...<Random Contents>
Drop the malformed STATUS packet

Figure 3-4. Use of the STATUS Command

There are four examples of the use of a Status query and response contained in Figure 3-4. The first message sent is a well formed STATUS request for information on a known VSID. The response to this message includes the state of the VSID, as well as all of the attributes used to OPEN the VSID. The second example is similar to the first, but for a VSID which is not open. The third example shows the use of a STATUS message to respond to a malformed packet. The last example shows the discard of a malformed STATUS command.

3.6.5 Use of the FLOW_CONTROL Command

Figure 3-5 shows sample usage of the FLOW_CONTROL command.

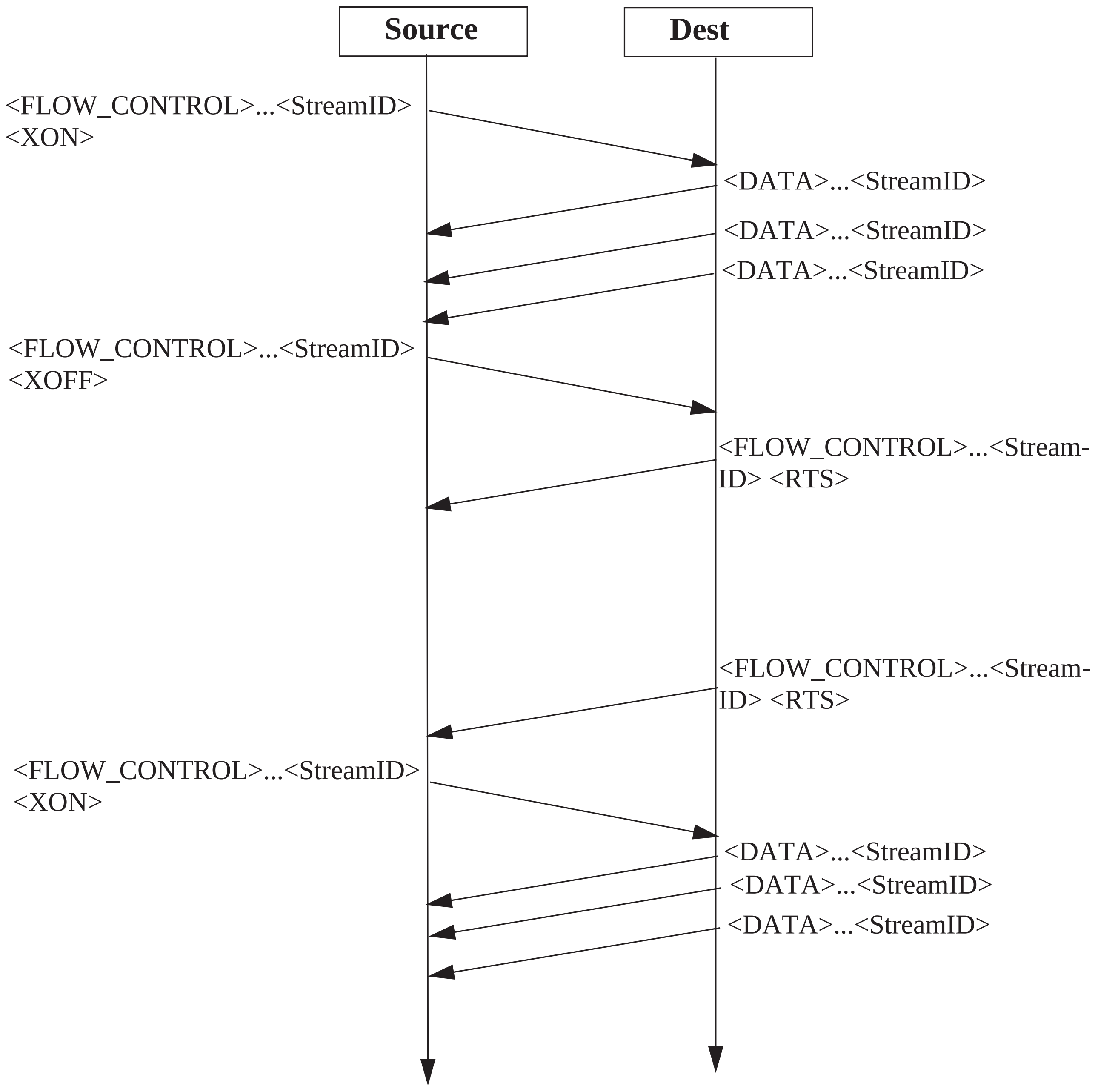

Source
Dest
<FLOW_CONTROL>...<StreamID> <XON>
<DATA>...<StreamID>
<DATA>...<StreamID>
<DATA>...<StreamID>
<FLOW_CONTROL>...<StreamID> <XOFF>
<FLOW_CONTROL>...<StreamID> <RTS>
<FLOW_CONTROL>...<StreamID> <RTS>
<FLOW_CONTROL>...<StreamID> <XON>
<DATA>...<StreamID>
<DATA>...<StreamID>
<DATA>...<StreamID>

Figure 3-5. Use of the FLOW_CONTROL Command

FLOW_CONTROL packets are used to start and stop the transmission of data for a VSID, and to inform receivers that data is available to be transmitted. The above example shows a transmission being started (<XON>), stopped (<XOFF>), and the receiver being informed that there is data to be sent. The VSID is then started (<XON>), and data transmission begins anew.

3.7 Session Management Error Conditions and Recovery

Error conditions which the Session Management Protocol is designed to handle are:

- Message Loss
- Message Congestion at Source or Destination
- Session Management Protocol Non-compliance

Under some situations, some operating systems may need to drop messages. It is this source of message loss that the Session Management Protocol is designed to deal with. Message loss is not likely to occur in RapidIO fabrics and hardware when reliable communication channels are used. Session Management Protocol must not be used over unreliable channels.

RapidIO fabrics may experience congestion. Many mechanisms are available to manage congestion within RapidIO systems, including those designed in the Session Management Protocol.

The Session Management Protocol is designed to allow deployment of reliable systems in the face of software defects. The scope of defects is generally limited to non-responsiveness or poorly formed responses. Pathological software defects and malicious intent may still result in Session Management Protocol failure.

Non-compliance is tested in several ways. First, whenever illegal values are specified in individual commands, the system receiving the command must respond with an indication of error. Second, compliant software systems must be able to be configured into a validation mode, in which fields are tested for validity whenever possible and all reserved fields are tested to insure that the contents are zero.

3.7.1 Message Loss

Message loss for the Session Management Protocol REQUEST, OPEN, CLOSE and STATUS request messages is detectable through timeouts on ADVERTISE, ACCEPT, REFUSE, and STATUS responses.

Session Management control messages may be dropped due to lack of resources by the receiver. In this case, the transmitter can detect the fact that the message has been lost through a response timeout, which will trigger retransmission of the original request. A timeout period limits the number of retries which can be attempted.

Both the interval between retries, and the overall timeout period, are negotiated using attributes in the OPEN command. These attributes are the *REQUEST_RETRY_PERIOD* attribute, and the *REQUEST_TIMEOUT_PERIOD* attribute, respectively. By default, a REQUEST, OPEN, CLOSE or STATUS request should be retried once every 250 msec, with an overall timeout period of 1 second.

FLOW_CONTROL XOFF message loss is detected through continued transmission of DATA on the stream which was turned off. In this case, the implementation may send additional FLOW_CONTROL XOFF messages, or if the congestion becomes severe, send a CLOSE request for the stream or conduit.

FLOW_CONTROL XON message loss is detected through timeouts on reception of a FLOW_CONTROL XON after the reception of a FLOW_CONTROL XOFF. The timeout period is set using the FLOW_CONTROL_XON TIMEOUT attribute in the OPEN message for the stream. To support a wide variation in timeout periods and data patterns, it may be necessary to handle to receive repeated FLOW_CONTROL XOFF messages for a stream.

If DATA message loss is allowed to occur in a system, it may be handled by the application or by the Session Management Protocol implementation.

Once message transmission has been timed out, the target of the message and all intervening nodes should be deemed suspect by the system.

In the event that no response is received in reply to a STATUS command within the reply delay period, the streamID specified in the STATUS command must be silently closed locally, and the remote node must be considered suspect.

For information on dealing with suspect nodes, refer to Chapter 3.7.3, “Session Management Protocol Non-Compliance,” on page 40.

3.7.2 Session Management Protocol Congestion Management

The Session Management Protocol is designed to avoid congestion conditions.

It is strongly recommended that messaging hardware implement a mechanism allowing ACCEPT, CLOSE, FLOW_CONTROL and STATUS messages to be sent and received with higher priority than messages containing commands OPEN, REQUEST, ADVERTISE, REFUSE, DATA and USERDEFINED. This allows the mechanisms for avoiding and managing congestion to operate in the presence of congestion.

The relative priority within the set of ACCEPT, CLOSE, FLOW_CONTROL and STATUS is implementation specific, with equality being the norm. Similarly, the relative priority within the set of OPEN, REQUEST, ADVERTISE, REFUSE, DATA, and USERDEFINED messages is implementation specific, with equality being the norm.

3.7.3 Session Management Protocol Non-Compliance

Session Management Protocol Non-Compliance is the term used to indicate one of two conditions. It can indicate that a target node is not operating in strict compliance with the Session Management Protocol and/or the timeout values used by the transmitter. It can also indicate that a target node is making illegal or non-specified use of reserved fields. In both cases, the target node is judged to be suspect (unreliable) by the Session Management Protocol implementation.

When a target node is judged to be suspect, the local system should follow the procedures, if any, defined for abnormal behavior for each command as described in Chapter 4, “Message Format Descriptions,” on page 45. System recovery actions may be initiated. The system recovery actions are outside the scope of this specification.

3.8 Rules for Session Management

This section describes restrictions and conditions during use of the Session Management Protocol.

3.8.1 Optional Features

This document describes a fully functional model, in which RapidIO end points can probe whether remote end points participate, send queries to discover what protocols each end point supports, and establish conduits and/or Virtual Streams with which to communicate. This functionality is designed for inter-operability of software, independent of the choice of OS.

In closed systems, the system designer may design the system so that each end point uses hard-coded information about all the remote end points with which it needs to communicate. In this case, the full functionality described in this document may not be necessary. Within such a closed system the use of the Component Tag CSR should be considered optional; however, a Session Management Protocol Register Extension Block, if available in the hardware, is not optional. The commands REQUEST and ADVERTISE are also optional. Designers of such systems should keep in mind that the rules below related to Attribute order may still place restrictions on the required order of Attributes to the OPEN command, and design the system accordingly.

Even in systems not intended as closed, the REQUEST and ADVERTISE commands are optional. No system may refuse to establish a connection based solely on the fact that no REQUEST command had been previously received and a corresponding ADVERTISE command sent in response. Systems which do not implement support for REQUEST and ADVERTISE will respond to REQUEST with STATUS, indicating Command_Unknown. When implementations do include support for REQEUEST and ADVERTISE, this also allows the system to attempt faster startup, trying OPEN first with the preferred protocol and attributes, and only fall back on the REQUEST / ADVERTISE mechanism in case of failure.

3.8.2 Attribute Related Rules

Implementations may choose to view Attributes as ordered lists. Therefore, any given implementation may refuse to open a Virtual Stream if the Attributes are not in the same order as presented in the ADVERTISE command. To ensure inter-operability, OPEN commands should maintain the order of Attributes that was used in the ADVERTISE command.

When advertising attributes, there may be cases when multiple values for a specific Attribute are provided for a single protocol block, resulting in duplicate copies of the Attribute. In this case, the initiator may remove the duplicate copies in order to select a specific value, or it may leave all values in place. However, by the time the ACCEPT command is sent, all duplicate copies must be removed. Therefore, if an OPEN command is received, containing duplicate copies of any attribute, the recipient must remove the duplicates and determine a specific value for every required Attribute.

In some cases, duplicate copies of Attributes may indicate that the receiver can receive a range of values. In this case, there will always be exactly two copies of the Attribute, indicating the minimum and maximum values. A system receiving an ADVERTISE command with this condition may attempt to OPEN a Virtual Stream with an intermediate value. If the receiver responds with REFUSE, then the sender must not attempt any other intermediate value for any such attribute, but restrict itself to the values specified. A single attempt at intermediate values may be attempted. It is not necessary for implementations to exhaustively check attribute ranges.

In all systems, vendor-specific VSIDs may be handled in vendor-specific manner. In this case, the participants are not required to go through the establishment protocol with OPEN, ACCEPT, and REFUSE, but may be defined by the vendor to be available for immediate DATA messages.

3.8.3 Rules Related to Virtual Stream Status

If an OPEN message is received for a protocol and remote end point, where an existing Virtual Stream is already open, the recipient may choose to either re-send the existing VSID, or to create a new VSID for a second instance of the protocol connection.

In the case that the recipient of an OPEN message responds with an ACCEPT command containing the existing VSID, the ACCEPT should be followed by a STATUS message indicating that the stream is functional and indicating other status information as appropriate. If any DATA or FLOW_CONTROL command is received, which specifies a VSID which the recipient does not understand, the recipient must respond with a STATUS command indicating that the VSID is unknown. Upon receipt of such a STATUS command, the node should, at a time deemed appropriate by the system designer, check all other open VSIDs to verify that they are functional. This is intended to handle the condition where a node is rebooted and the Virtual Streams need to be closed and/or reestablished.

3.8.4 Rules Related to Vendor-Specific Commands

If any USERDEFINED command is received, which the recipient does not understand, the receiver must respond with a STATUS command indicating that the USERDEFINED command is unknown.

3.8.5 Rules Related to Reserved Fields

Several fields in the structures described by this specification are marked reserved. On transmission, these fields must be filled with zeros.

Implementations of Session Management Protocol conforming to this specification must be able to be configured into a validation mode. When configured in this mode, the receiver must test all reserved fields for zero-filled content, and reject the received command if not zero-filled. The sender of such a message is thereafter to be treated as suspect (unreliable). Normal error handling is used in such a case, or if no other error handling is specified, a STATUS message must be returned, indicating Command_Unknown.

For higher performance, implementations of Session Management Protocol conforming to this specification may be able to be configured into a non-validation mode, in which reserved fields are ignored. Other error handling must not be disabled when the implementation is configured in non-validation mode.

3.9 Notes on Optional Features and Inter-Operability

For full inter-operability, an implementation must support all the features of this specification, including the optional features. There are conditions in which lack of optional features may restrict functionality and inter-operability. This section lists a sample of some potential consequences of not implementing all defined features, though this list is neither complete nor comprehensive. It is included as a warning of some consequences of design decisions during implementation.

3.9.1 Optional Attributes

Every implementation may choose to implement vendor specific attributes. If any vendor specific attribute is used, it should be optional. If not, then it is unlikely that other systems, which may not understand the vendor-specific attribute, will be able to inter-operate with the implementation.

For example, if the *DATA_OFFSET_VENDOR* and *DATA_OFFSET* attributes are required, then the system will not be able to communicate with other implementations. However, if the *DATA_OFFSET_VENDOR* and *DATA_OFFSET* attributes are used but not required, then the implementation may operate more efficiently with other systems using the feature, but will continue to inter-operate with systems not implementing that feature.

3.9.2 REQUEST and ADVERTISE

The REQUEST and ADVERTISE message types are listed as optional. This is true in two senses. First, no system is required to use REQUEST or ADVERTISE when attempting to open a connection. Second, no system is required to recognize incoming REQUEST and ADVERTISE messages, but may respond to them with STATUS messages indicating Command_Unknown.

However, any arbitrary node, which wants to establish a connection with a node that does not recognize incoming REQUEST messages, may not be able to determine the appropriate attributes to use for the desired protocol. Without a listing of the available attributes from an ADVERTISE message, it is not possible reliably to make a connection. Such systems would not inter-operate.

Furthermore, if an implementation that does not support REQUEST and ADVERTISE messages also chooses to view attributes as an ordered list, then even in the case that some external agent provided the attributes and values to use, then the two systems may still not be inter-operable due to ordering restrictions.

Chapter 4 Message Format Descriptions

4.1 Introduction

This chapter contains the definition of the data streaming packet format.

4.2 Control Message Formats

All message formats are given in big endian format - the most significant octet is on the left of each field.

4.2.1 REQUEST

A REQUEST message is used to request information related to protocols supported by the remote. There are two primary variants to the REQUEST message: first, to request a list of protocols supported by the remote; second, to request attributes of a specified protocol. These two variants are distinguished by the contents of the ProtoID field. A value containing all ones, 0xffff, indicates a request for a list of protocols without attributes. Protocol specific values of ProtoID are described in "Section 3.5.2.3, Protocol Identifier: <ProtoID>" on page 29.

Table 4-1. REQUEST Message Format

	Octet 0	**Octet 1**	**Octet 2**	**Octet 3**
Word 0	<CMD=0x01>	<VER>	<SourceID>	<SourceID>
Word 1	<DestID>	<DestID>	<COS>	Reserved
Word 2	<ProtoID>	<ProtoID>	<NumAttrib>	<NumAttrib>
Word 3	Reserved	Reserved	Reserved	Reserved

The form of REQUEST, which is used to request the attributes for a specified protocol, may limit the request by including a list of required attributes. The <NumAttrib> field contains the number of attributes listed, or zero if no attributes are listed. Upon receipt of a REQUEST message, the receiver should respond only with attributes that match the attributes included in the REQUEST message.

All the header fields in a REQUEST message other than <NumAttrib> are described in Chapter 3.5.2, "Message Header Fields," on page 28.

4.2.2 ADVERTISE

An ADVERTISE message is sent in response to a REQUEST message, to identify supported protocols or attributes of a specified protocol, depending on the contents of the REQUEST message. The two variants are distinguished by the value of the <A> bit in the message header.

Table 4-2. ADVERTISE Message Format - Protocol Attributes

3	2	1	0
<CMD=0x02>	<VER>	<SourceID>	<SourceID>
<DestID>	<DestID>	<S+A+Count>	<Count=M>
<ProtoID_1>	<ProtoID_1>	<numAttributes_1=N1>	<numAttributes_1=N1>
<Attribute_1>	<Attribute_1>	<Attribute_1>	<Attribute_1>
<Attribute_1>	<Attribute_1>	<Attribute_1>	<Attribute_1>
<Attribute_2>	<Attribute_2>	<Attribute_2>	<Attribute_2>
<Attribute_2>	<Attribute_2>	<Attribute_2>	<Attribute_2>
.....			
<Attribute_N1>	<Attribute_N1>	<Attribute_N1>	<Attribute_N1>
<Attribute_N1>	<Attribute_N1>	<Attribute_N1>	<Attribute_N1>
<ProtoID_2>	<ProtoID_2>	<numAttributes_2=N2>	<numAttributes_2=N2>
<Attribute_1>	<Attribute_1>	<Attribute_1>	<Attribute_1>
<Attribute_1>	<Attribute_1>	<Attribute_1>	<Attribute_1>
<Attribute_2>	<Attribute_2>	<Attribute_2>	<Attribute_2>
<Attribute_2>	<Attribute_2>	<Attribute_2>	<Attribute_2>
.....			
<Attribute_N2>	<Attribute_N2>	<Attribute_N2>	<Attribute_N2>
<Attribute_N2>	<Attribute_N2>	<Attribute_N2>	<Attribute_N2>
.....			
<ProtoID_M>	<ProtoID_M>	<numAttributes_2=NM>	<numAttributes_2=NM>
<Attribute_1>	<Attribute_1>	<Attribute_1>	<Attribute_1>
<Attribute_1>	<Attribute_1>	<Attribute_1>	<Attribute_1>
<Attribute_2>	<Attribute_2>	<Attribute_2>	<Attribute_2>
<Attribute_2>	<Attribute_2>	<Attribute_2>	<Attribute_2>
.....			
<Attribute_NM>	<Attribute_NM>	<Attribute_NM>	<Attribute_NM>
<Attribute_NM>	<Attribute_NM>	<Attribute_NM>	<Attribute_NM>

Table 4-3. ADVERTISE Message Format - Protocol List

3	**2**	**1**	**0**
<CMD=0x02>	<VER>	<SourceID>	<SourceID>
<DestID>	<DestID>	<1+0+Count>	<Count>
<ProtoID_1>	<ProtoID_1>	<ProtoID_2>	<ProtoID_2>
	.....		
<ProtoID_Count>	<ProtoID_Count>	Reserved	Reserved
Reserved	Reserved	Reserved	Reserved

<S>: (1 bit) indicates whether or not the requested protocol is supported. If the value of <S> is 1, then the protocol is supported. If the value of <S> is 0, then the protocol is not supported, and the <A> bit must be set to zero (0).

<A>: (1 bit) indicates whether or not the ADVERTISE message includes attributes or whether it is a simple list of protocols. Note that if <S> is set, then the <A> bit is invalid and must contain the value zero (0). <A>=1 indicates that the format follows Table 4-2. <A>=0 indicates that the format follows Table 4-3.

If <A>=0, then the message contains a list of <ProtoID> values, as shown in Table 4-3. Note that the final Reserved fields shown in the table indicate padding to a multiple of 8 octets, and that there may be zero to seven such Reserved octets, and not exactly the six (6) octets shown in the Table 4-3. The Reserved octets must contain zero (0) values.

The valid values of S+A are 0b00, 0b10, and 0b11. The value of 0b01 is reserved.

If <S>=0, then <Count> is invalid and must be set to zero (0). If <S>=1, then <Count> is valid. <Count> is a 14 bit value the number of <ProtoID>’s included in the ADVERTISE message. Note that this field does not indicate octet length, but rather the number of protocols.

If <A>=0, then the message contains a list of <ProtoID> values, padded out to a multiple of 8 octets.

If <A>=1, then the message contains a list of Attributes associated with the <ProtoID>, padded out to the nearest multiple of 8 octets. The number of Attributes in the message is contained in the <numAttributes> field. Note that a value of 0, meaning that no Attributes are supported for this <ProtoID>, is valid.

4.2.3 OPEN

An OPEN message is used to request that the remote system create a stream or virtual stream suitable for the remote system to receive data on. The OPEN message specifies the protocol which is to be carried on the stream or virtual stream, as well as any protocol attributes that need to be used.

Table 4-4. OPEN Message Format

3	2	1	0
<CMD=0x03>	<VER>	<SourceID>	<SourceID>
<ProtoID>	<ProtoID>	<numAttributes=N>	<numAttributes=N>
<Attribute_1>	<Attribute_1>	<Attribute_1>	<Attribute_1>
<Attribute_1>	<Attribute_1>	<Attribute_1>	<Attribute_1>
<Attribute_2>	<Attribute_2>	<Attribute_2>	<Attribute_2>
<Attribute_2>	<Attribute_2>	<Attribute_2>	<Attribute_2>
.....			
<Attribute_N>	<Attribute_N>	<Attribute_N>	<Attribute_N>
<Attribute_N>	<Attribute_N>	<Attribute_N>	<Attribute_N>

Note that the packet format specifies the SourceID, but no DestID. When the OPEN message is received, the receiver uses it’s own nodeID as the recipient of traffic, and the SourceID specified in the message is the information that the recipient needs to have in order to pass control traffic.

The OPEN request must have the *OPEN_MESSAGE_NUMBER* attribute as the first attribute in its attribute list.

4.2.4 ACCEPT

An ACCEPT message is used to inform a requestor that a stream or virtual stream is now open and that the requestor can send data traffic using the StreamID specified in the ACCEPT message.

Table 4-5. ACCEPT Message Format

3	2	1	0
<CMD=0x04>	<VER>	<DestID>	<DestID>
<ack-type>	<COS>	<StreamID>	<StreamID>
<ProtoID>	<ProtoID>	<numAttributes=N>	<numAttributes=N>
<Attribute_1>	<Attribute_1>	<Attribute_1>	<Attribute_1>
<Attribute_1>	<Attribute_1>	<Attribute_1>	<Attribute_1>
<Attribute_2>	<Attribute_2>	<Attribute_2>	<Attribute_2>
<Attribute_2>	<Attribute_2>	<Attribute_2>	<Attribute_2>
.....			
<Attribute_N>	<Attribute_N>	<Attribute_N>	<Attribute_N>
<Attribute_N>	<Attribute_N>	<Attribute_N>	<Attribute_N>

“ack type” values:

- 0: normal ACK
- 1-255: reserved

For ACCEPT and REFUSE messages, the DestID is the nodeID of the receiver, which is the system sending the ACCEPT or REFUSE message.

The only attribute required in the ACCEPT message is the *OPEN_MESSAGE_NUMBER* attribute from the OPEN. Other attributes from the REQUEST message may optionally be copied to the ACCEPT message.

4.2.5 REFUSE

A REFUSE message is sent in response to an OPEN request, if the requestee cannot create a stream or virtual stream with the protocol and attributes specified in the OPEN request.

Table 4-6. REFUSE Message Format

3	2	1	0
<CMD=0x05>	<VER>	<DestID>	<DestID>
<nack-type>	0xFF	0xFF	0xFF
<ProtoID>	<ProtoID>	<numAttributes=N>	<numAttributes=N>
<Attribute_1>	<Attribute_1>	<Attribute_1>	<Attribute_1>
<Attribute_1>	<Attribute_1>	<Attribute_1>	<Attribute_1>
<Attribute_2>	<Attribute_2>	<Attribute_2>	<Attribute_2>
<Attribute_2>	<Attribute_2>	<Attribute_2>	<Attribute_2>
.....			
<Attribute_N>	<Attribute_N>	<Attribute_N>	<Attribute_N>
<Attribute_N>	<Attribute_N>	<Attribute_N>	<Attribute_N>

“nack type” values:

- 0: normal NACK
- 1-255: reserved

For ACCEPT and REFUSE messages, the <DestID> is the nodeID of the receiver, which is the system sending the ACCEPT or REFUSE message.

The only attribute required in the REFUSE message is the *OPEN_MESSAGE_NUMBER* attribute from the OPEN. Other attributes from the REQUEST message may optionally be copied to the REFUSE message.

4.2.6 FLOW_CONTROL

A FLOW_CONTROL message is used to regulate traffic and manage traffic congestion. There are three types of flow control behavior which can be specified. First, use the XON command to enable data traffic. Second, use the XOFF command to suspend data traffic. In the case that data traffic is suspended, RTS can be used to inform the stream owner (data receiver) that data is available to be sent.

Table 4-7. FLOW_CONTROL Message Format

3	2	1	0
<CMD=0x07>	<VER>	<COS>	<Flow_Control>
<SourceID>	<SourceID>	<StreamID>	<StreamID>
<ProtoID>	<ProtoID>	Reserved	Reserved
Reserved	Reserved	Reserved	Reserved

The default state when a stream is opened is for traffic to be enabled. No explicit FLOW_CONTROL command XON needs to be sent in the default case.

Once an RTS message is sent, additional RTS messages must not be sent to indicate that additional data is available. A single RTS message is sufficient. Because no explicit response to RTS is required, the sender of a FLOW_CONTROL command RTS may retry the message at 250 msec intervals, until 1 second has elapsed.

Flow_Control values:

- XON: 0x01
- XOFF: 0x00
- RTS: 0xff

No node is required to initiate FLOW-CONTROL. However, all RapidIO nodes must accept and handle FLOW-CONTROL commands coming from the remote. Note that it is possible to receive a FLOW-CONTROL command with XON flow when no previous FLOW-CONTROL command with XOFF flow has been received, since the FLOW-CONTROL XOFF command could have been lost.

4.2.7 CLOSE

A CLOSE message is used to terminate a stream or virtual stream. Either endpoint may initiate the CLOSE behavior. In the event that the stream or virtual stream is part of a conduit, both streams or virtual streams must be closed at the same time. The means of determining that a stream is part of a conduit is implementation specific.

Table 4-8. CLOSE Message Format

3	2	1	0
<CMD=0x08>	<VER>	<SourceID>	<SourceID>
<DestID>	<DestID>	<COS>	Reserved
<StreamID>	<StreamID>	Reserved	Reserved
Reserved	Reserved	Reserved	Reserved

NOTE: both SourceID and DestID must be specified in order to allow both endpoints to initiate a shutdown. Regardless of which endpoint initiates the CLOSE, the SourceID indicates the node sending data.

When receiving a CLOSE message, the receiver of the CLOSE must reply with a STATUS message indicating that the StreamID has been closed.

4.2.8 STATUS

A STATUS message is used to request the status of a stream or virtual stream, to report the status of a stream or virtual stream, and to indicate certain error conditions such as illegal commands. The status bits indicate the status of the stream and/or the reason for sending the STATUS message.

Table 4-9. STATUS Message Format

3	2	1	0
<CMD=0x10>	<VER>	<COS>	<DataSize>
<SourceID>	<SourceID>	<StreamID>	<StreamID>
<Mailbox>	<Reserved>	<CmdID>	<CmdVersion>
<Status>	<Status>	<Status>	<Status>
<ContextSpecificData>	<ContextSpecificData>	<ContextSpecificData>	<ContextSpecificData>
<ContextSpecificData>	<ContextSpecificData>	<ContextSpecificData>	<ContextSpecificData>
.....			
<ContextSpecificData>	<ContextSpecificData>	<ContextSpecificData>	<ContextSpecificData>
<ContextSpecificData>	<ContextSpecificData>	<ContextSpecificData>	<ContextSpecificData>

Table 4-10. Status Bit Values

Status Bit	Meaning
0x00000001	Stream Unknown
0x00000002	Stream Functional
0x00000004	Ready to Receive
0x00000008	Data Ready to Send
0x0ffffff0	Reserved
0x10000000	Error
0x20000000	Closed
0x40000000	Command Unknown
0x80000000	Request Status of Remote

A STATUS message consists of two parts. The initial header fields, as shown in Table 4-9, and context specific data. The contents of the context specific data depends on the reason for sending the STATUS message. In all cases, the <DataSize> field is used to indicate he number of 8-octet words contained in the <ContextSpecificData> fields. That is, the number of octets in the <ContextSpecificData> field is eight times the value of <DataSize>.

If the STATUS message is sent in response to an invalid, unknown, or malformed command, then the original command is sent in the <ContextSpecificData> fields and the Command_Unknown flag bit is set. If the STATUS message is sent in response to a STATUS message with the Request_Status_of_Remote bit set, then the <ContextSpecificData> fields should be set to a copy of the ProtoID and attributes used to create the StreamID or virtual StreamID. If the STATUS message is sent in order to request the status of a StreamID, the <ContextSpecificData> fields are not used, and <DataSize> must be set to zero.

In the case that the Stream_Unknown bit is set in the status field, the only two valid flag bits are Stream_Unknown and Closed. Other flags are implicitly Reserved, and must be set to zero by the sender.

When configured in validation mode, the receiver must test the value of the Reserved bits of the Status field, including the bits explicitly marked as Reserved and the bits which are implicitly Reserved if the Stream_Unknown bit is set.

The receiver must not respond to an illegally formed STATUS command with another STATUS command. If not well-formed, the STATUS command must be ignored. Such as system should indicate the reception of an illegal STATUS message with a message printed to the console or other implementation specific error reporting mechanism, but the method and format to indicate this error is beyond the scope of this specification.

When a STATUS message is received with the ’Request_Status_of_Remote’ bit set, a STATUS message must be sent back to the originator for the streamID requested. The STATUS message which is sent back must not have the ’Request_Status_of_Remote’ bit set.

The STATUS message will only include <ContextSpecificData> fields when a STATUS message is sent as a response for an unrecognized/unsupported/malformed message or in response to a STATUS message.

4.2.9 User Defined

A number of message IDs are reserved for application specific commands.

If a system receives a USERDEFINED command which is not understood, it must respond with a STATUS message, with the Command Unknown bit set to indicate that it does not understand the command.

Table 4-11. USERDEFINED Message Format

3	2	1	0
<CMD=0xF0 to 0xFF>	<VER>	<COS>	Reserved
<SourceID>	<SourceID>	<StreamID>	<StreamID>
<user-defined data>	<user-defined data>	<user-defined data>	<user-defined data>
<user-defined data>	<user-defined data>	<user-defined data>	<user-defined data>

4.3 Data Formats

4.3.1 DATA Message Format, MAILBOX

DATA message format is used to send virtual Type 9 data (streams, see *RapidIO Interconnect Specification Part 10: Data Streaming Logical Specification*) when the only conveyance provided by the hardware is Type 11 (messages). The <Mailbox> field is set to the mailbox ID used by the receiver of the data. <COS> is defined in Table 3.5.2.4, "Class of Service: <COS>," on page 29.

Table 4-12. DATA Message Format

3	2	1	0
<CMD=0x06>	<VER>	<Mailbox>	<COS>
<Reserved>	<Reserved>	<SourceID>	<SourceID>
<S+E+00 +Length(4bits)>	<Length>	<StreamID/PDU-Length>	<StreamID/PDU-Length>
Payload Data octet	Payload Data octet	Payload Data octet	Payload Data octet
.....			
Payload Data octet	Payload Data octet	Payload Data octet	Payload Data octet

In the case that data is received for a StreamID unknown to the receiver, then the receiver must respond with a STATUS message with the Closed bit set to indicate that the stream is no longer open. It may also send STATUS messages to all other StreamIDs for the same remote system, whether the StreamID is for transmitted traffic or for received traffic.

The S and E bits are used to segment transfers larger than the maximum PDU length over multiple messages. The S bit indicates that this is the first segment in a transfer. The E bit indicates that this is the last segment in a transfer. If neither the S or the E bit is set, then this is one of the middle segments in a multi-segment transfer. If a transfer fits within a single segment, then both the S and E bits are set. PDUs in a multi-segment transfer must be sent in order: Start segment, middle segments, End segment.

The length field contains the size of the data payload included in this message excluding the encapsulation header. The StreamID/PDU-Length field contains either the StreamID of the message or the length of the data payload excluding the encapsulation header. If either the S or E bit is set, then the StreamID/PDU-Length field contains the StreamID. If neither S nor E is set, then the StreamID/PDU-Length field contains PDU-Length, which includes the total size of all RapidIO messages making up this packet, exclusive of encapsulation headers.

4.3.2 DATA1 Message Format, Large PDU

DATA1 message format is used in situations where large amounts of data need to be transferred as a block. All fields have the same meaning as in the DATA header.

Table 4-13. DATA Message Format

3	2	1	0
<CMD=0x09>	<VER>	<Mailbox>	<COS>
<Reserved>	<Reserved>	<SourceID>	<SourceID>
<S+E+Length(6bits)>	<Length>	<Length>	<Length>
<StreamID/PDU-Length >	<StreamID/PDU-Length >	<Reserved/PDU-Length>	<Reserved/PDU-Length>
Payload Data octet	Payload Data octet	Payload Data octet	Payload Data octet
.....			
Payload Data octet	Payload Data octet	Payload Data octet	Payload Data octet

All fields in this header format are used in the same manner as the DATA command. However, the capacity of this format allows transfers of up to 4 gigabytes of data.

Underlying hardware may place limits on the size of messages, such as only allowing data to be transferred in multiples of eight octets. In this case, padding must be used where necessary to fill the size constraints. If padding is used, the data must contain zeros, and the pad octets are not included in the <S+E+00+Length>, <Length>, <StreamID/PDU-Length>, and <Reserved/PDU-Length> fields.

4.3.3 DATA2 Message Format

The DATA2 command is available for use when the full information contained in the DATA and DATA1 headers is not required. Note that use of this format restricts the packet size to the size limited by hardware, so that the maximum PDU size is 16380 octets, but may be smaller due to hardware constraints.

Table 4-14. DATA Message Format

3	2	1	0
<CMD=0x0A>	<Implementation-Specific>	<S+E+Length (6 bits)>	<Length>

The Implementation-Specific field is available for use by implementers. Note that the DATA2 message format is only used when the DATA Header attribute is used during stream creation. The value to be used in this field is the seventh octet of the DATA Header attribute, preceding the DATA command value.

4.3.4 DATA3 Zero-length DATA header

When using some conveyances, it may be possible to fully segregate traffic based on streamID or other information. In this case, no header is required. For the purposes of determining the DATA header format, this can be considered as CMD=0x0B, though the actual command value of 0x0B must never be transmitted.

This header format is used for hardware assisted data streaming, as defined in *RapidIO Interconnect Specification Part 10: Data Streaming Logical Specification*.

4.3.5 Data Streaming

When the data is carried by the Data Streaming Protocol is uses a logical layer packet format defined in *RapidIO Interconnect Specification Part 10: Data Streaming Logical Specification*. In this case, no additional header needs to be used.

It is strongly recommended that the RapidIO specific information necessary for software to compose and respond to Data messages is made accessible to software through implementation specific means by hardware that supports type 9 packets.

Blank page

Chapter 5 Registers

5.1 Introduction

Before sending session management protocol messages, it is necessary to know if the target node supports session management protocol, and the messaging method used by the target node. The target node advertises this information using registers.

A Session Management Protocol register extension block may be used to advertise a target nodes session management protocol parameters. This method is the recommended approach.

Devices which are capable of accepting session management protocol messages may advertise this fact in the Component Tag CSR. The Component Tag CSR may be used to advertise session management protocol parameters only if a Session Management Protocol register extension block is not present in a device. Only devices whose Destination Operations CAR and Source Operations CAR indicate support for Data Message and/or Data Streaming transactions may advertise support for the Session Management Protocol using the Component Tag CSR.

The use of the Component Tag CSR is deprecated for new devices.

5.2 Session Management Protocol Extended Features Register Block

Where Reserved fields are used in the following structures, the value must be set to zero. Implementations configured in validation mode should check these fields when first reading the Session Management Protocol Extended Features Register Block, and indicate the presence of an illegal Session Management Protocol Extended Features Register with a message to the console or other implementation specific error reporting mechanism, but the method and format to indicate this error is beyond the scope of this specification.

Note that there should be an instance of the Session Management Protocol Extended Features Register Block for every conveyance supported by the endpoint.

5.2.1 Session Management Protocol Register Block Header (Block Offset 0x0)

The Session Management Protocol Register Block Header contains the EF_PTR to the next extended features block and the EF_ID that identifies this as the Session Management Protocol Register Block Header.

Table 5-1. Bit Settings for Session Management Protocol Register Block Header

Bit	Name	Reset Value	Description
0-15	EF_PTR		Hard wired pointer to the next block in the data structure, if one exists
16-31	EF_ID	0x000C	Hard wired Extended Features ID

5.2.2 Session Management Protocol Register Write Enable CSR (Block Offset 0x4)

The Session Management Protocol Advertisement register is used allow an external RapidIO entity write access to the Session Management Protocol registers, which otherwise are read only.

The operation of this register is identical to the Host Base Device ID CSR specified in Part 2 Common Transport Specification:

- When the Lock_Val is 0xFFFF, all other registers in this block are read only.
- Writing to this register when Lock_Val is 0xFFFF sets the Lock_Val field to the value written.
- When the Lock_Val field is not 0xFFFF, all registers in this block are writable. Implementation specific checking may be done on the write transactions to this block.
- When the Lock_Val field is not 0xFFFF, writing the value of the Lock_Val field to this register resets the Lock_Val field to 0xFFFF.
- When the Lock_Val field is not 0xFFFF, writing a value different from the Lock_Val field to this register does not affect the value of the Lock_Val field.
- Writing 0xFFFF to the Lock_Val field when the Lock_Val field is 0xFFFF has no effect.

Table 5-2. Bit Settings for Session Management Protocol Register Write Enable Register

Bit	Name	Reset Value	Description
0-15	Reserved	0x0000	Not Used
16-31	Lock_Val	0xFFFF	See description above

5.2.3 Session Management Advertisement CSR (Block Offset 0x8)

The Session Management Protocol Advertisement register is used to indicate whether or not the node supports the session management protocol and, if so, how to send session management protocol messages to the node.

All of the fields of this register are read-only using RapidIO maintenance transactions, but may be written by the local processing element.

Table 5-3. Bit Settings for Session Management Protocol Advertisement Register

Bit	Name	Reset Value	Description
0-3	Conveyance	Impl. Spec.	Identifies which conveyance this Session Management Protocol register block applies to: 0x0 - Messaging (Type 11) 0x1 - Data Streaming (Type 9) 0x2-0xE - Reserved 0xF - Not Supported Write Protected by Session Management Protocol Register Write Enable register.
4-31	Conveyance_Info	Impl. Dep.	Conveyance Info This field communicates conveyance specific information for reception of Session Management Protocol messages. If Conveyance = type 11, the field format is 0x00000nn where nn represents the mailbox ID, formatted according to the definition in Part 2. If Conveyance = type 9, the field format is 0x0ccssss where: cc - Class of Service (COS) value to be used ssss - StreamID to be used Write Protected by Session Management Protocol Register Write Enable register.

5.2.4 Session Management Attribute Range CSR (Block Offset 0xC)

The Session Management Protocol Advertisement register is used to indicate whether or not the node supports the session management protocol and, if so, how to send session management protocol messages to the node.

All of the fields of this register are read-only using RapidIO maintenance transactions.

Table 5-4. Bit Settings for Session Management Attribute Range Register

Bit	Name	Reset Value	Description
0-7	—	0x00	reserved
8-15	Sess_Mgmt_Max_Attr	Impl. Dep.	This field contains the maximum number of attributes. Each attribute takes up 8 octets. 0 - No Session Management Protocol Attribute Registers 1 - 2 Session Management Protocol Attribute Registers 2 - 4 Session Management Protocol Attribute Registers 3 - 6 Session Management Protocol Attribute Registers ... n - 2n Session Management Protocol Attribute Registers ... 0xFE - 508 Session Management Protocol Attribute Registers 0xFF - Reserved This field is Read Only from the RapidIO interface, but may be writable by the local Processing Element.
16-19	—	0x0	reserved

Table 5-4. Bit Settings for Session Management Attribute Range Register

Bit	Name	Reset Value	Description
20-23	Sess_Mgmt_Init_Stage	Impl. Dep.	This field indicates progress through the register initialization sequence, with a standard value indicating that initialization is complete. 0x0 - Initialization of registers complete 0x1-0xF - Initialization of registers incomplete, initialization stage X in progress. Write Protected by Session Management Protocol Register Write Enable register.
24-31	Sess_Mgmt_Num_Attr	Impl. Dep.	This field contains the number of valid attributes following this register. Each attribute takes up 8 octets. 0 - No Session Management Protocol Attribute Registers 1 - 2 Session Management Protocol Attribute Registers 2 - 4 Session Management Protocol Attribute Registers 3 - 6 Session Management Protocol Attribute Registers ... n - 2n Session Management Protocol Attribute Registers ... 0xFE - 508 Session Management Protocol Attribute Registers 0xFF-0xFFFE - Reserved 0xFFFF - Session Management Protocol Attribute Registers not initialized Write Protected by Session Management Protocol Register Write Enable register.

5.2.5 Session Management Protocol Attributes 0-508 CSRs (Block Offset 0x10-0x7F8)

The number of valid Session Management Protocol Attribute registers is indicated by the Sess_Mgmt_Num_Attr field of the Session Management Protocol Advertisement register.

The number of Session Management Protocol Attributes registers is implementation specific. Up to 508 Session Management Protocol Attributes registers can exist.

It is recommended that at least 8 Session Management Protocol Attributes registers be implemented.

Table 5-5. Bit Settings for Session Management Protocol Attributes 0-508 Registers

Bit	Name	Reset Value	Description
0-63	Attribute Data	Impl Dep	Attribute specification data, formatted as per the attributes defined in Section 3.5.3 and in the protocol-specific chapters.

5.3 Component Tag CSR Session Management Protocol Advertisement

If a Session Management Protocol Register Block Header does not exist within the registers for a device, the device may advertise support for the Session Management Protocol using the Component Tag CSR, as described in Table 5-6.

Only devices whose Destination Operations CAR and Source Operations CAR indicate support for Data Message and/or Data Streaming transactions may advertise support for the Session Management Protocol using the Component Tag CSR.

If Bit 0 of the Component Tag CSR contains a value of 0, then the node does not conform to this specification. In this case, this specification puts no requirements on the remaining bits of the Component Tag CSR.

If Bit 0 contains a value of 1, then the node must not use the first 16 bits of this register for any purpose other than advertising support for Session Management Protocol messages.

Table 5-6. Component Tag CSR Bit Usage

Bit	Field Name	Description
0	Sess_Mgmt_Sup	Session Management Protocol Support This bit indicates whether or not a processing element supports the Session Management Protocol. 0b0 - This processing element does not support the Session Management Protocol 0b1 - This processing element does support the Session Management Protocol

Table 5-6. Component Tag CSR Bit Usage

Bit	Field Name	Description
1-2	Sess_Mgmt_Chan	Session Management Protocol Channel This bit indicates what logical layer to use, and how to interpret the Channel Information below: 0b00 = Use Message Passing (type 11) 0b01 = Use Data Streaming (type 9) 0b10 = Reserved 0b11 = Expansion This field is only valid if Sess_Mgmt_Sup is 0b1.
2-15	Sess_Mgmt_Chan_Info	Session Management Protocol Channel Info This field communicates channel specific information for reception of Session Management Protocol messages. If Session Management Channel = type 11: bits 2-15 = 0b00000nnnnnnnn where the low 8 bits are the mailbox ID (formatted according to the definition in Part 2) If Session Management Channel = type 9: bits 2-5 = 0bx0000nnnnnnnn where: x = 1 = StreamID MSB = 0xFF x = 0 = StreamID MSB = 0x00 n = StreamID LSB The StreamID is the Stream to use to embed the management messages. The COS field in the VSID should be the highest priority COS supported by the interface. ex: 0b100001111000 = StreamID = 0xFFF0 This field is only valid if Sess_Mgmt_Sup is 1.

The Sess_Mgmt_Chan bits indicate the conveyance to use for Session Management Protocol traffic. The value of 0b11, Expansion, is used to indicate that a reserved, expanded Sess_Mgmt_Chan field, larger than two bits, is to be used. In this case, Sess_Mgmt_Chan_Info will consist of fewer bits.

Blank page

Chapter 6 Vendor-Defined Protocols

6.1 ProtoID

The ProtoID value for vendor-defined protocols is 0x0101.

6.2 Attributes

Two protocol attributes are required for vendor-defined protocols. These two attributes distinguish a vendor-defined protocol from all other vendor-defined protocols. The attributes are the *VENDOR* attribute, and the *PROTOCOL-NAME* attribute.

6.2.1 *VENDOR* attribute

The VENDOR attribute, described in “Section 3.5.3.1, VENDOR Attribute” on page 31, is required. It must be the first attribute listed.

6.2.2 *PROTOCOL_NAME* attribute

The *PROTOCOL-NAME* attribute is an 8-bit attribute ID (0x01) with a 56-bit value. The *PROTOCOL-NAME* attribute is required for all vendor-defined protocols, and must immediately follow the *VENDOR* attribute. The format of the attribute value is defined by the vendor. The sole restriction is that the value defined by the vendor must uniquely identify the protocol in that it differentiates the protocol from all other protocols defined by the vendor. It may consist of an ASCII string or a numeric value, at the discretion of the vendor defining the protocol.

6.2.3 Other attributes

Other attributes may be defined by the vendor.

6.3 Other Requirements for Vendor-Defined Protocols

Vendors wishing to make their protocols available may choose to create an RFC describing attributes and other issues related to their protocol.

Blank page

Chapter 7 Ethernet Encapsulation

7.1 ProtoID

The protocol ID value for Ethernet encapsulation is 0x0102.

7.2 Attributes

Several protocol attributes are required: *MTU*, *CONVEYANCE*, and *MAC_ADDRESS*. No ordering restrictions are placed on these attributes by the specification, though implementations may impose ordering restrictions. Additional optional protocol attributes exist, depending on the system configuration.

7.2.1 MTU Attribute

The MTU is assigned with the 16-bit attributed ID 0x8002, leaving six octets of data for the MTU value. Only the lowest two octets should be used, allowing a maximum MTU of 64 KBytes.

7.2.2 *CONVEYANCE* Attribute

The *CONVEYANCE* attribute, as described in “Section 3.5.3.10, CONVEYANCE Attribute” on page 34, is required for Ethernet encapsulation.

7.2.3 *MAC_ADDRESS* Attribute

The MAC address is specified with the 16-bit attribute ID 0x8003, leaving six octets of data for the MAC address.

This protocol is intended for use where RapidIO nodes and Ethernet nodes may co-exist on the same virtual Ethernet segment. Therefore, MAC addresses are required to conform to industry standards. This includes the requirement that the first three octets of the MAC address should be a valid OUI (Organizationally Unique Identifier) assigned by IEEE.

7.3 Other Requirements of Ethernet Encapsulation

There are a number of additional requirements for Ethernet encapsulation. These requirements are discussed in the following sections.

7.3.1 Dropped Messages

In Ethernet, it is appropriate for a packet to be dropped if there are difficulties sending it, since higher level protocols handle retransmission. Although this differs from the normal goal of RapidIO, that is, to have reliable transmission, there are some conditions in which it is appropriate to drop Ethernet encapsulation messages.

If a node has received an XOFF from the remote system using Ethernet encapsulation at the time a new message is to be sent, the node must retain at least one pending message for transmission, however it may choose to drop additional messages. The pending message(s) should be the most recent message(s), and older messages should be dropped.

7.3.2 Broadcast

7.3.2.1 Broadcast With Multicast Extensions

If multicast capabilities are available in the RapidIO switches, they should be used for broadcast messages. The value of the destination ID for ethernet encapsulation should be 0xFE for 8-bit IDs or 0xFFFE for 16-bit IDs.

7.3.2.2 Broadcast Without Multicast Extensions

If multicast capabilities are not available, broadcast messages must be sent by unicast transmission. The reserved StreamID value 0xF000 may be used for transmission of broadcast traffic. Note that no OPEN message is required for this StreamID, so no dedicated resources are required to be permanently allocated in order to receive broadcast traffic. Receivers of broadcast traffic know that it is a broadcast message, by reading the StreamID of incoming messages.

7.3.2.3 Vendor defined Broadcast Server

Vendors may wish to define a vendor-specific protocol for use by a broadcast server, thereby eliminating the requirement for multiple unicast message transmission by other nodes. Such broadcast servers must not re-transmit incoming broadcast messages received on the broadcast StreamID 0xF000.

7.3.3 Ingress/Egress Nodes

Ingress/Egress nodes may be configured as switches or as routers. If configured as routers, ingress and egress traffic are handled at a higher level protocol, and not the subject of this protocol.

If ingress/egress nodes are configured as switches, the node must forward ingress and egress traffic. This is handled as a layer 2 switch, the behavior of which is well understood and not defined in this protocol.

When Ethernet-over-RapidIO traffic is transmitted outside the RapidIO fabric, the egress system must be able to be configured to forward broadcast packets to the external network and forward external broadcast packets into the RapidIO fabric. This can be done by forwarding RapidIO messages received on StreamID 0xF000 to ethernet interfaces. Incoming broadcast packets coming from Ethernet interfaces must be sent using the RapidIO broadcast mechanism.

Blank page

Glossary of Terms and Abbreviations

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

B

Bridge. A processing element that connects one computer bus to another, allowing a processing element on one bus to access an processing element on the other.

C

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

Class of service (cos). A term used to describe different treatment (quality of service) for different data streams. Support for class of service is provided by a class of service field in the data streaming protocol. The class of service field is used in the virtual stream ID and in identifying a virtual queue.

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.

Conduit. A bidirectional data transfer mechanism consisting of two streams or virtual streams, one for communication in each direction.

Conveyance. A communication channel, e.g. mailbox, stream, shared memory mechanism, etc.

D

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.

Double-word. An eight octet or 64 bit quantity, aligned on eight octet boundaries.

E

Egress. Egress is the device or node where traffic exits the system. The egress node also becomes the destination for traffic out of the RapidIO fabric. The terms egress and destination may or may not be used interchangeably when considering a single end to end connection.

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.

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.

H

Half-word. A two octet or 16 bit quantity, aligned on two octet boundaries.

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

I

Ingress. Ingress is the device or node where traffic enters the system. The ingress node also becomes the source for traffic into the RapidIO fabric. The terms ingress and source may or may not be used interchangeably when considering a single end to end connection.

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

I/O. Input-output.

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.

PDU. Protocol Data Unit, the OSI term for a packet.

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.

Protocol Attributes. Information relevant to communication using a particular protocol, or to data transferred within the context of a connection using that protocol.

R

Receiver. The RapidIO interface input port on a processing element.

S

SAR. Segmentation and Reassembly, a mechanism for encapsulating a PDU within multiple packets.

Segmentation. A process by which a PDU is transferred as a series of smaller *segments*.

Session Management Protocol. The protocol specified in this document, used for negotiation of communication sessions and optionally for data transfers.

Sequence. Sequentially ordered, uni-directional group of messages that constitute the basic unit of data delivered from one end point to another.

StreamID. A specific field in the data streaming protocol that is combined with the data stream's transaction request flow ID and the sourceID or destinationID from the underlying packet transport fabric to form the virtual stream ID.

Suspect. A communication partner, which may not fully conform to the session management protocol.

Source. The origin of a packet on the RapidIO interconnect, also referred to as an initiator.

Switch. A multiple port processing element that directs a packet received on one of its input ports to one of its output ports.

T

Target. The termination point of a packet on the RapidIO interconnect, also referred to as a destination.

Transaction. A specific request or response packet transmitted between end point devices in a RapidIO system.

Transaction request flow. A sequence of transactions between two processing elements that have a required completion order at the destination processing element. There are no ordering requirements between transaction request flows.

V

Virtual Stream ID (VSID). An identifier comprised of several fields, used to identify individual data streams. When using Type 9 (streaming) as the conveyance for data transfers, the VSID is encapsulated in the Type 9 protocol. When using Type 11 (messaging) as the conveyance for data transfers, the VSID is encapsulated in fields in the DATA or DATA1 Session Management Protocol commands.

W

Word. A four octet or 32 bit quantity, aligned on four octet boundaries.