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

Rev. 1.3, 06/2005

RapidIO Trade Association

Revision History

Revision	Description	Date
1.1	First public release	03/08/2001
1.2	Technical changes: incorporate Rev. 1.1 errata rev. 1.1.1, errata 3	06/26/2002
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
1.3	Removed confidentiality markings for public release	06/07/2005

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

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

Table of Contents

Chapter 1 Overview

Chapter 2 System Models

Chapter 3 Operation Descriptions

Table of Contents

Chapter 4
Packet Format Descriptions

Chapter 5
Input/Output Registers

Table of Contents

Table of Contents

List of Figures

List of Figures

List of Tables

List of Tables

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.

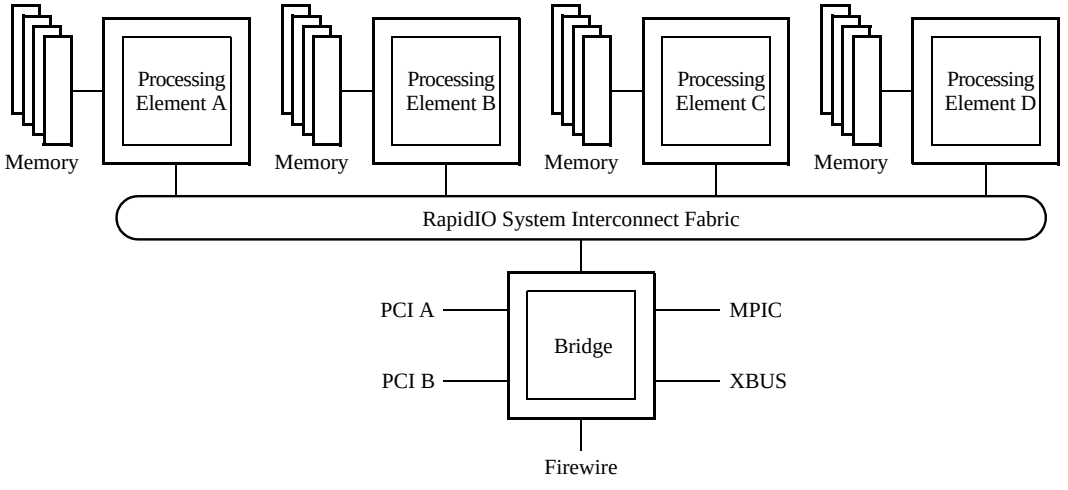


Figure 2-1. A Possible RapidIO-Based Computing System

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.

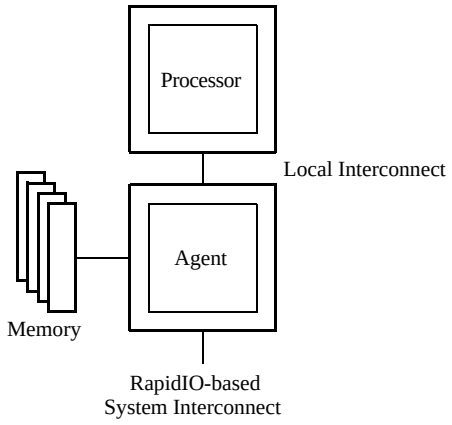


Figure 2-2. Processor-Memory Processing Element Example

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.

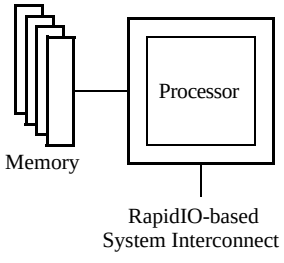


Figure 2-3. Integrated Processor-Memory Processing Element Example

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.

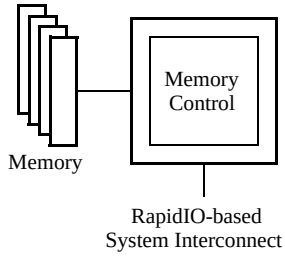


Figure 2-4. Memory-Only Processing Element Example

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.

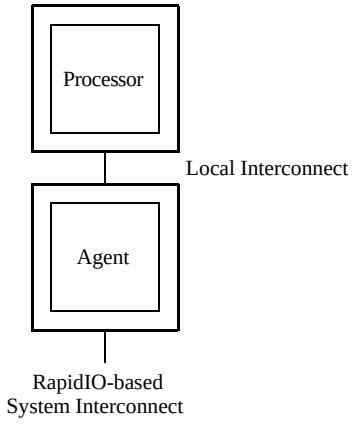


Figure 2-5. Processor-Only Processing Element Example

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.*

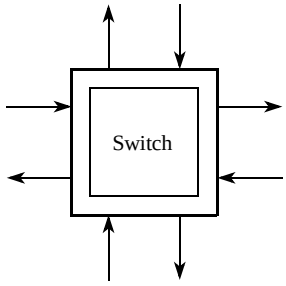


Figure 2-6. Switch Processing Element Example

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

There may be multiple transaction request flows between a given source and destination pair. When multiple flows exist between a source and destination pair,

the flows are distinguished by a flow indicator (flowID). 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.

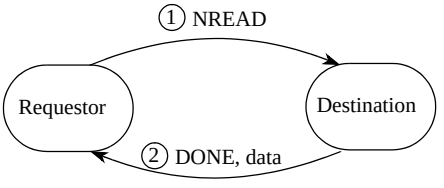


Figure 3-1. Read Operation

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.

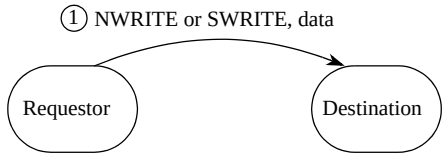


Figure 3-2. Write and Streaming-Write Operations

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.

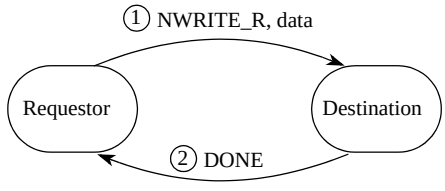


Figure 3-3. Write-With-Response Operation

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.

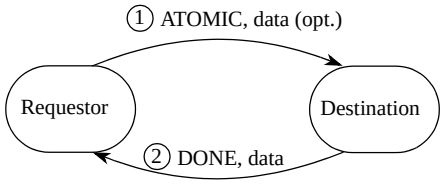


Figure 3-4. Atomic (Read-Modify-Write) Operation

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.

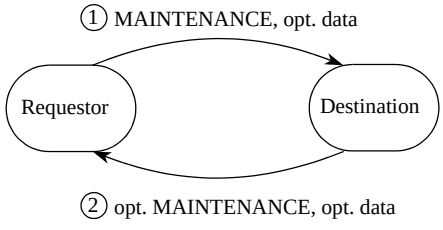
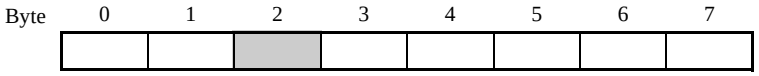


Figure 3-5. Maintenance Operation

3.5 Endian, Byte Ordering, and Alignment

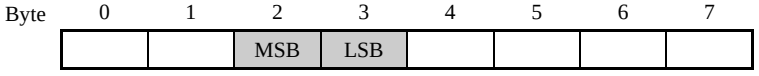
RapidIO has double-word (8-byte) aligned big-endian data payloads. This means that the RapidIO interface to devices that are little-endian shall perform the proper endian transformation to format a data payload.

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.



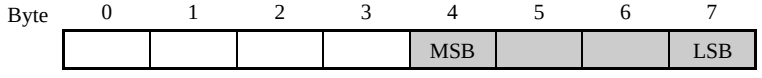
Byte address 0x0000_0002, the proper byte position is shaded.

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.

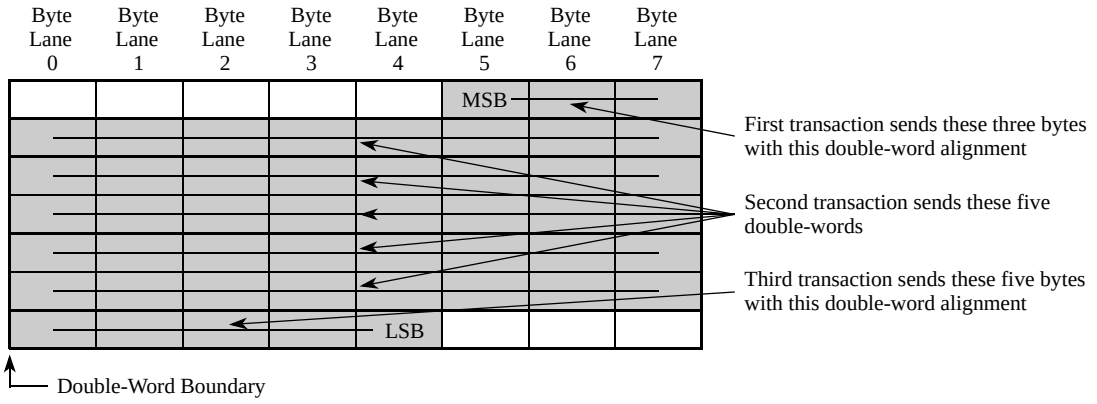


Figure 3-9. Data Alignment Example

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 1's to specified address	Section 4.1.5
	ATOMIC clear	Read-write 0's 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.

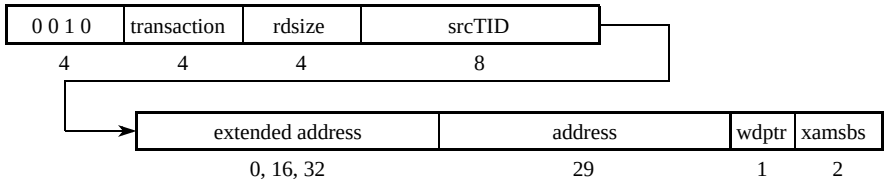


Figure 4-1. Type 2 Packet Bit Stream Format

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.

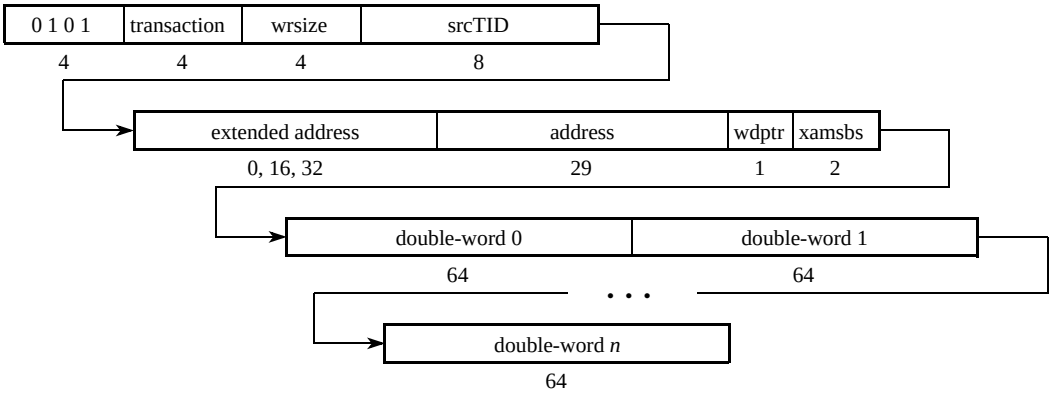


Figure 4-2. Type 5 Packet Bit Stream Format

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.

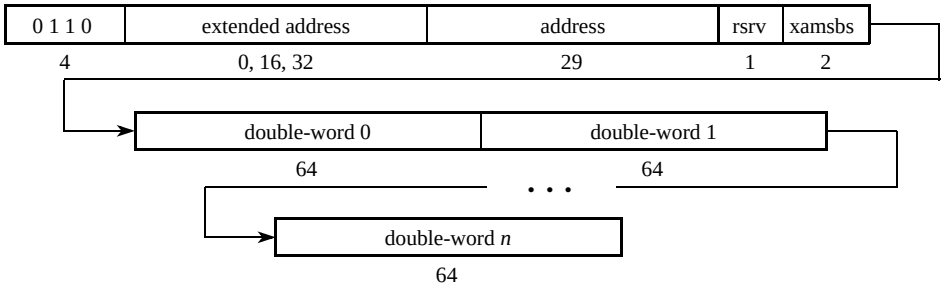


Figure 4-3. Type 6 Packet Bit Stream Format

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.

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.

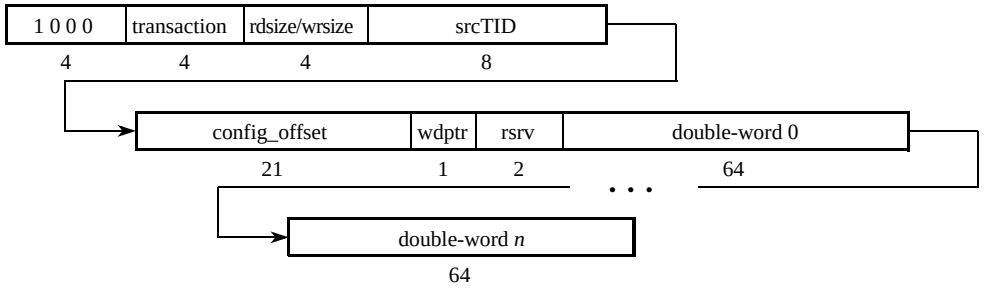


Figure 4-4. Type 8 Request Packet Bit Stream Format

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

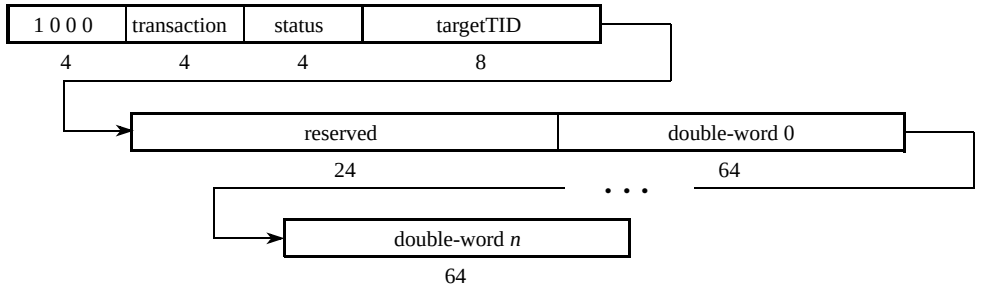


Figure 4-5. Type 8 Response Packet Bit Stream Format

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.

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.

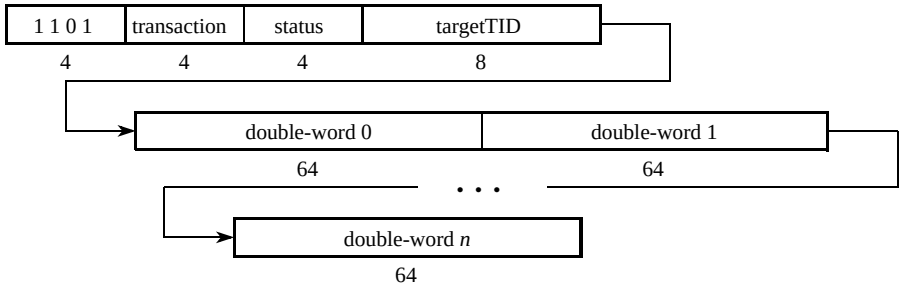


Figure 4-6. Type 13 Packet Bit Stream Format

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.

Blank page

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

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

Configuration Space Byte Offset	Register Name
0x18	Source Operations CAR
0x1C	Destination Operations CAR
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 and Bit Behavior

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

Table 5-2. Configuration Space Reserved Access Behavior

Byte Offset	Space Name	Item	Initiator behavior	Target behavior
0x0–3C	Capability Register Space (CAR Space - this space is read-only)	Reserved bit	read - ignore returned value[1]	read - return logic 0
			write -	write - ignored
		Implementation-defined bit	read - ignore returned value unless implementation-defined function understood	read - return implementation-defined value
			write -	write - ignored
		Reserved register	read - ignore returned value	read - return logic 0s
			write -	write - ignored

Table 5-2. 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.

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.

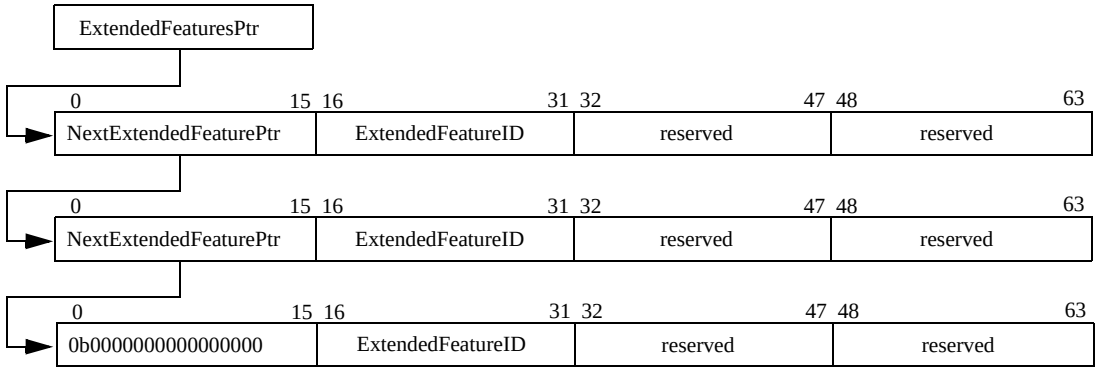


Figure 5-1. Example Extended Features Data Structure

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 the RapidIO Trade Association.

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

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

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

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

Table 5-7. Bit Settings for Processing Element Features CAR (Continued)

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

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

Table 5-9. Bit Settings for Source Operations CAR (Continued)

Bit	Field Name	Description
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

Table 5-10. Bit Settings for Destination Operations CAR (Continued)

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

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.

Blank page

Blank page