RapidIO™ Interconnect Specification Part 2: Message Passing Logical Specification

4.1, 6/2017

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
4.0	No technical changes.	09/18/2016
4.1	No technical changes.	06/30/2017

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

Table of Contents

List of Figures

List of Figures

Blank page

List of Tables

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.

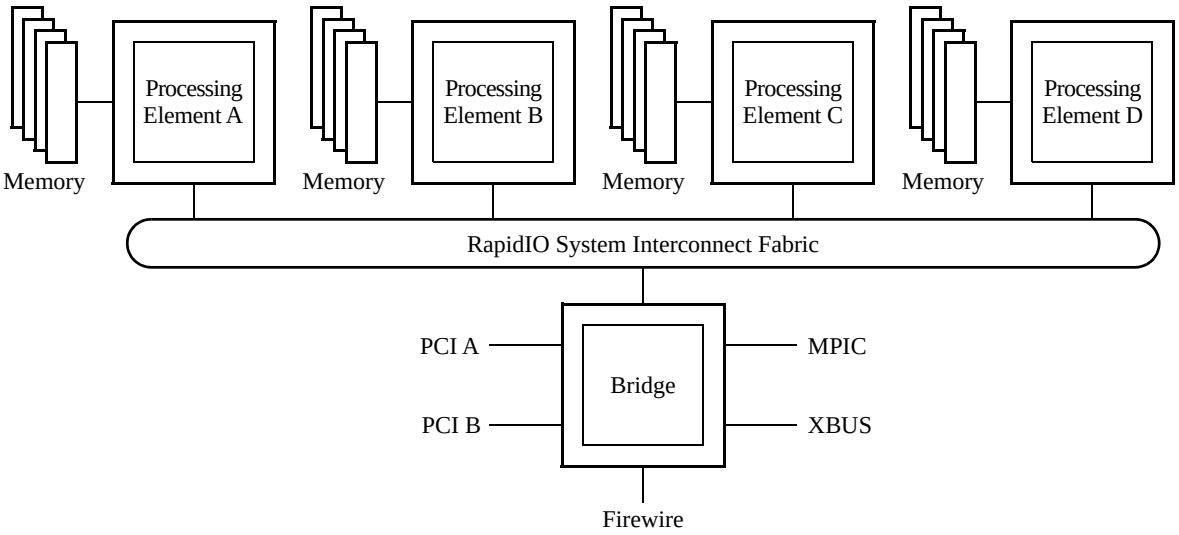


Figure 2-1. A Possible RapidIO-Based Computing System

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.

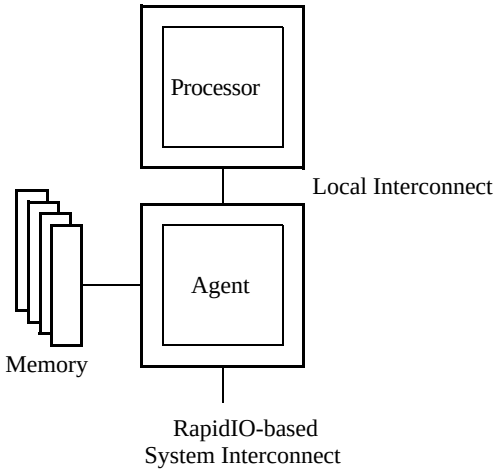


Figure 2-2. Processor-Memory Processing Element Example

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.

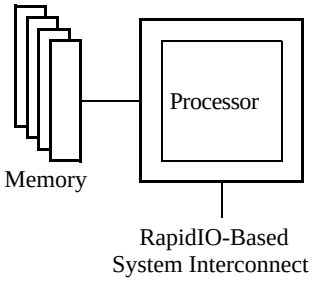


Figure 2-3. Integrated Processor-Memory Processing Element Example

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.

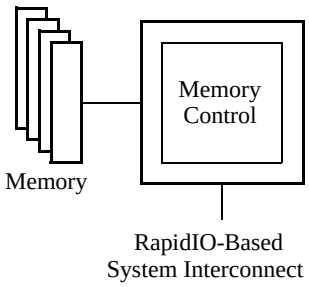


Figure 2-4. Memory-Only Processing Element Example

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.

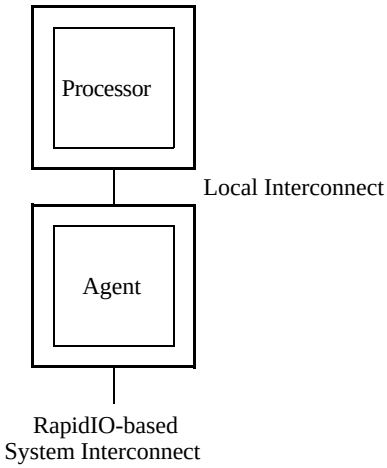


Figure 2-5. Processor-Only Processing Element Example

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*.

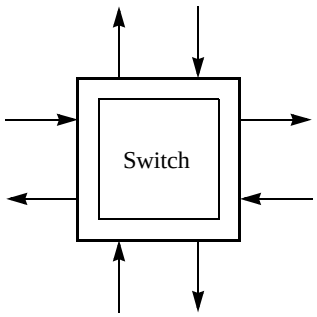


Figure 2-6. Switch Processing Element Example

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.

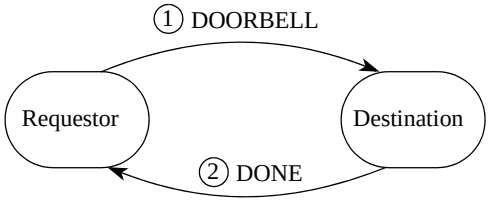


Figure 3-1. Doorbell Operation

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.

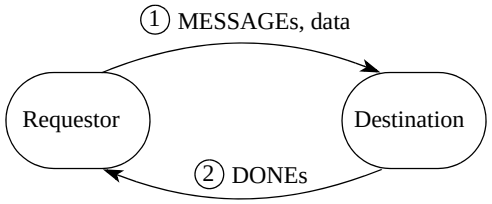


Figure 3-2. Message Operation

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)

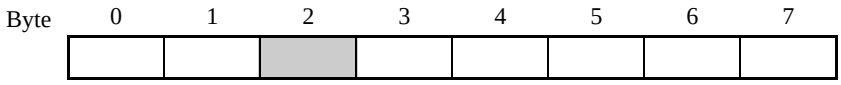
Using this information, the processing element's message passing hardware can determine that the 32 bytes contained in this part of the data message shall be put into local memory at address 0x3040.

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

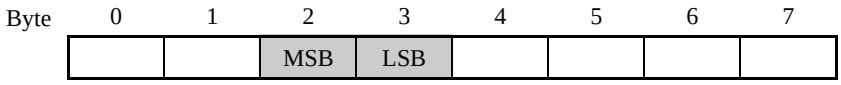
RapidIO has double-word (8-byte) aligned big-endian data payloads. This means that the RapidIO interface to devices that are little-endian shall perform the proper endian transformation at the output to format a data payload.

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.



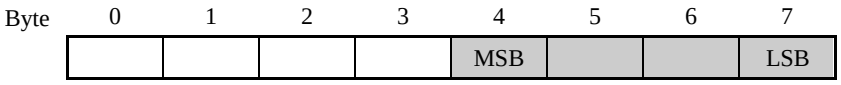
Byte address 0x0000_0002, the proper byte position is shaded.

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.

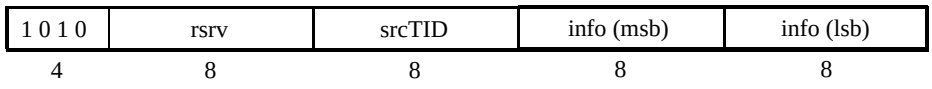


Figure 4-1. Type 10 Packet Bit Stream Format

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.

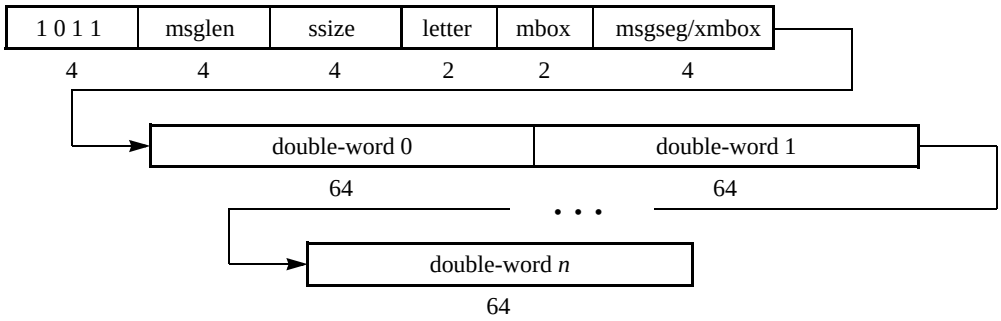


Figure 4-2. Type 11 Packet Bit Stream Format

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.

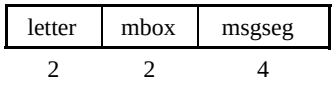


Figure 4-3. target_info Field for Message Responses

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.

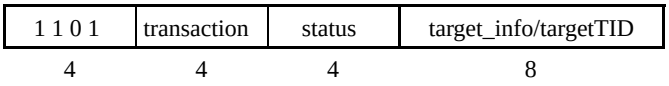


Figure 4-4. Type 13 Packet Bit Stream Format

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).

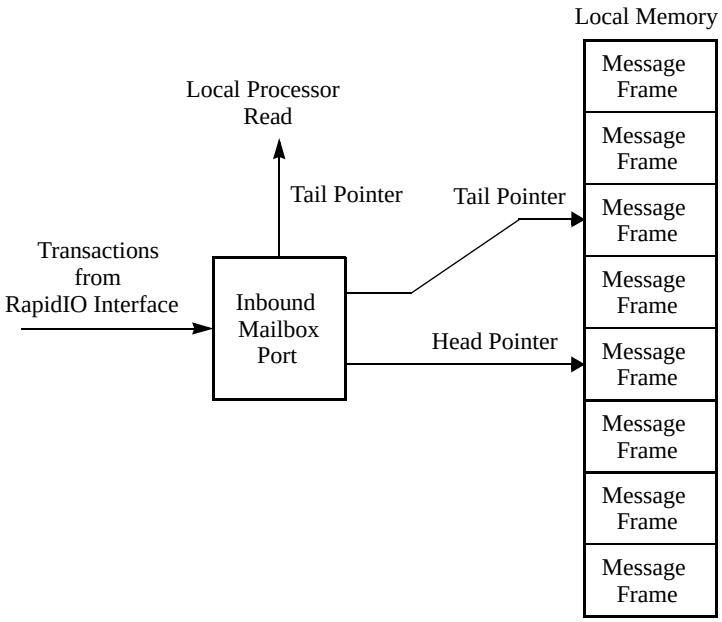


Figure A-1. Simple Inbound Mailbox Port Structure

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).

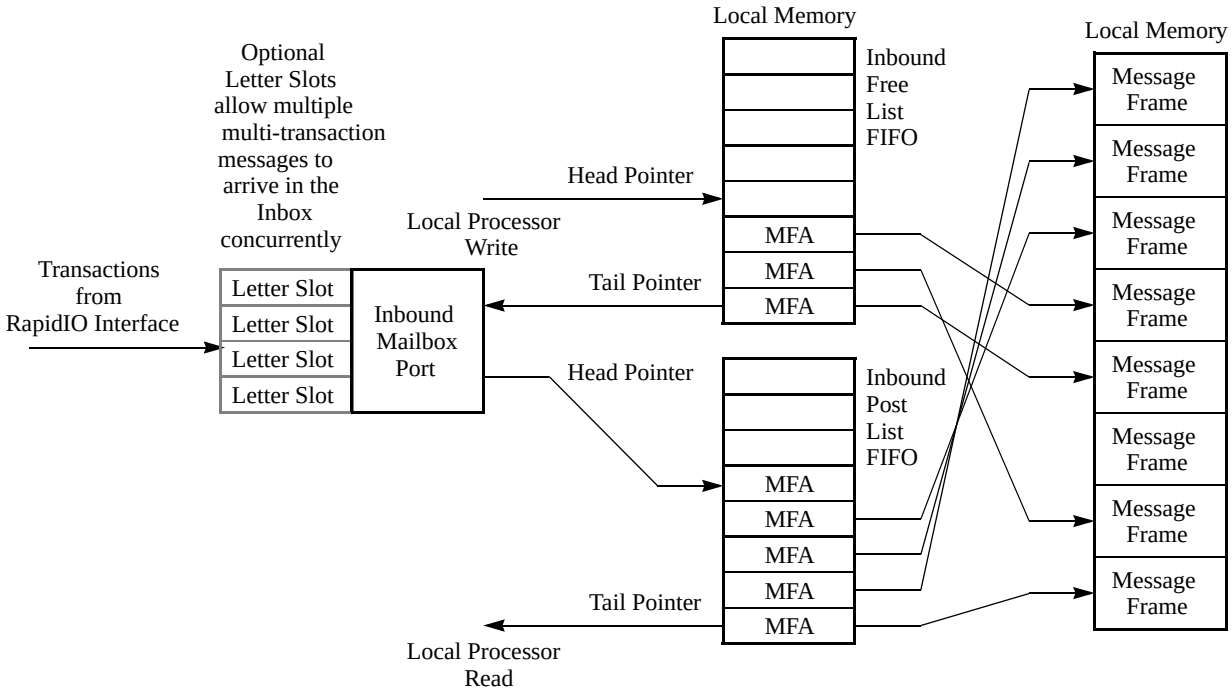


Figure A-2. Inbound Mailbox Structure

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.

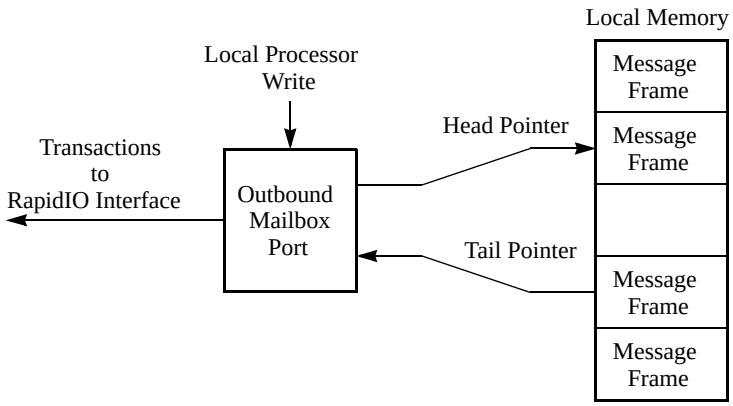


Figure A-3. Outbound Message Queue

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.

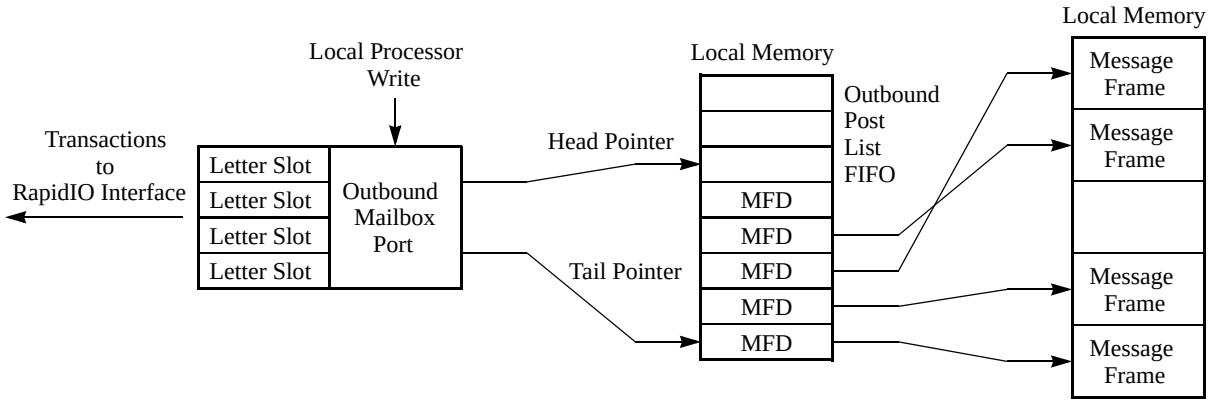


Figure A-4. Extended Outbound Message Queue

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.