Freescale Semiconductor

AN2753
Rev. 1, 11/2004

RapidIO Bring-Up Procedure on PowerQUICC III™

by Lorraine McLuckie and Colin Cureton
NCSD Platforms, East Kilbride

The MPC8540 and MPC8560 PowerQUICC III™ processors have an 8-bit parallel RapidIO interface. This document provides guidance in the basic use of this interface; detailing the setup of local and remote processors starting with basic verification and discovery, then describing booting over RapidIO and conducting simple memory tests.

Most of the guidance in this document applies to any RapidIO enabled PowerQUICC III system. However, the concepts are illustrated by a few specific examples.

Contents

1.Introduction 1
1.1. Glossary 2
2.Background Information 3
2.1. RapidIO. 3
2.2. Using RapidIO on MPC8540 and MPC8560 6
2.3. Example Target Hardware 8
3.Bring-up Procedure 11
3.1. Configure the Local Processor 12
3.2. Discover Other Devices in the System 20
3.3. Enable Access to Remote Configuration Space 27
3.4. Boot over RapidIO 30
3.5. Enable Memory Reads and Writes 36
4.Output from Example Application 41
4.1. Point-to-Point Example. 41
4.2. Fabric Example. 45
5.References 54
6.Revision History 54
Appendix A. Notes on Maintenance Transactions 55

1 Introduction

This document is designed to provide assistance to engineers wishing to use the RapidIO interface on the MPC8540 and MPC8560 PowerQUICC III processors. It summarizes some basic aspects of the RapidIO specification, and the use of the RapidIO interfaces on these processors. It also details a basic procedure to bring up simple RapidIO systems; including the setup of the local processor, simplified discovery, booting over RapidIO, and the execution of simple memory reads/writes to the remote processors.

Section 2, “Background Information,” provides background information regarding the RapidIO specifications, the implementation and usage of the RapidIO interface on the MPC8540/60, and information regarding the example systems, which are used to illustrate the steps of the procedure outlined in Section 3, “Bring-up Procedure.”

Section 3, “Bring-up Procedure,” describes a procedure for bringing up a basic MPC8540/60-based RapidIO system. Both point-to-point and fabric-based systems are considered.

Section 4 provides example output from an application that follows the procedure outlined in Section 3, “Bring-up Procedure,” executed on the example hardware systems described in Section 2, “Background Information.”

Appendix A contains additional information about maintenance transactions, paying particular attention to the calculation of the maintenance offset used

1.1 Glossary

The following acronyms and terms are used throughout this application note.

Table 1. Glossary of Terms

Term	Description
ATMU	Address translation and mapping unit
BAR	Base address register
CAR	Capability attribute register
CCSR	Configuration, control and status registers
CCSRBAR	CCSR base address register
CSR	Capability status register
DDR SDRAM	Double data rate SDRAM
DIDCAR	Device identity capability register
DMA	Direct memory access
EEPROM	Electrically erasable programmable read only memory
HBDIDLCSR	Host base device lock ID CSR
I/O	Input/Output
JTAG	Joint test access group
LAW	Local area window
LP-LVDS	Link protocol, low voltage differential signaling
MAS	MMU assist register
MMU	Memory management unit
PowerQUICC III	MPC85xx networking and communications processor
R/W	Read/Write
RIO	RapidIO
RIW	RapidIO inbound window
ROW	RapidIO outbound window
SBTG	Software bring-up technical group

Table 1. Glossary of Terms (continued)

Term	Description
SDRAM	Synchronous dynamic random access memory
TLB	Translation lookaside buffer

2 Background Information

This section provides background information relevant to the bring-up procedure. Section 2.1, “RapidIO,” provides a summary of the relevant aspects of the RapidIO specification. Section 2.2, “Using RapidIO on MPC8540 and MPC8560,” describes the implementation of RapidIO on MPC8540/60 and considerations for its use. Section 2.3, “Example Target Hardware,” provides information regarding the example systems which will be used to illustrate the procedure.

2.1 RapidIO

The RapidIO interconnect architecture is a high-performance packet-switched interconnect technology. It addresses the high-performance embedded industry's need for reliability, increased bandwidth, and faster bus speeds in an intra-system interconnect. The RapidIO interconnect allows chip-to-chip and board-to-board communications at performance levels scaling to 10 Gigabits per second and beyond.

This section summarizes some basic information about the RapidIO specification. Additional information can be found in the *RapidIO Interconnect Specification*.[1]

2.1.1 RapidIO Layers

The RapidIO specification is written in a layered manner, to ensure flexibility and that changes to one layer do not necessarily affect other layers.

The layers of the RapidIO architecture are as follows:

- Logical layer—Defines the overall protocol and packet formats, the types of transaction that can be carried out with RapidIO and how addressing is handled.
- Transport layer—Provides the necessary route information for a packet to move from one point to another.
- Physical layer—Contains the device level interface such as packet transport mechanisms, flow control, electrical characteristics, and low level error management.

The logical layer is split into two categories depending on the system model in use. The first part is the input/output logical specification which defines the basic system architecture of RapidIO. The other is the message passing logical specification which enables distributed I/O processing. This note concentrates on the input/output logical specification.

2.1.2 Processing Element Models

There are several types of devices that can be used on a RapidIO based system. This note describes only two device model types: the integrated processor-memory processing element and the switch processing element.

2.1.2.1 Integrated Processor-Memory Processing Element

One form of the processor-memory processing element is a fully integrated component that is designed to connect to a RapidIO interconnect system as shown in Figure 1. This type of device integrates a memory system and other support logic with a processor core on the same piece of silicon, or within the same package, and is one example of a RapidIO end-point. In this note the processing element is an MPC8540/60 device.

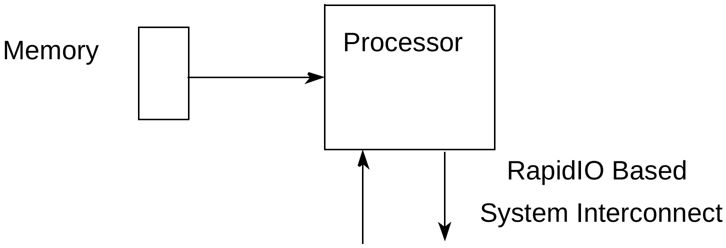


Figure 1. Integrated Processor-Memory Processing Element

2.1.2.2 Switch Processing Element

A switch processing element is a device that allows communication with other processing elements through a switch. A switch may be used to connect a variety of RapidIO compliant processing elements. Behavior of the switches, and the interconnect fabric in general, is addressed in the *RapidIO Common Transport Specification*.[1] In this note the switch processing element used is the Tundra Tsi500™ 4-port switch.[3]

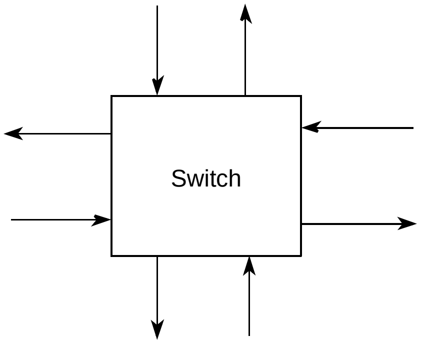


Figure 2. Switch Processing Element Model

2.1.2.3 RapidIO Transactions

In the logical layer, RapidIO defines a broad range of transaction types, ranging from simple transactions to access an agent device's memory space, to user defined and implementation dependent transactions. Throughout this note, three different classes of request transactions and their associated responses are used, where a request is a transaction that is issued by a processing element to accomplish some activity on a remote processing element. The three transactions that are used in this note are detailed below. These sections detail the logical layer aspects of these transaction; no consideration is given to the transport or physical layer interactions.

2.1.2.3.1 Maintenance Transaction

A maintenance transaction is a special system support request. It is used by a processing element to read or write data to capability attribute registers (CARs) and capability status registers (CSRs), defined in the RapidIO specification.[1] Maintenance requests do not contain addresses, but use a maintenance offset into the CAR/CSR block to specify which register is to be read or written.

Maintenance transactions are used for system initialization and discovery and for altering system configuration.

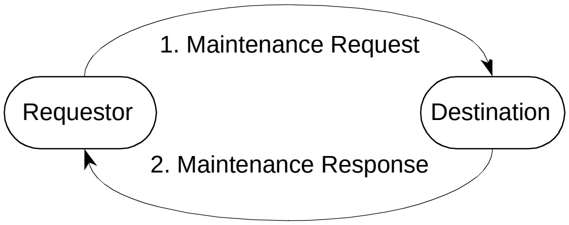


Figure 3. Maintenance Transaction

2.1.2.3.2 NREAD Transactions

A read transaction, consisting of the NREAD and RESPONSE transactions, is used by a processing element to read data from a specified address. The data is returned in a response packet and is of the size requested.

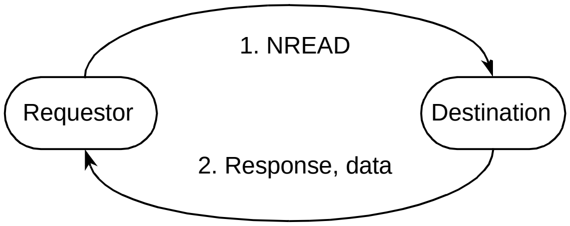


Figure 4. NREAD Transaction

2.1.2.3.3 NWRITE/NWRITE_R Transactions

The write transaction is used by a processing element to write data to a specified address. The write transaction can take several forms including NWRITE and NWRITE_R.

The NWRITE request allows data writes, with no expected response.

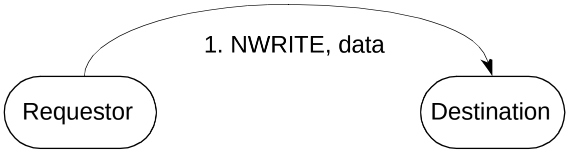


Figure 5. NWRITE Transaction

The NWRITE_R (write with response) transaction is identical to the NWRITE transaction except that it requires a response to notify the sender that the write has completed at the destination. This transaction is useful for guaranteeing read-after-write and write-after-write ordering through a system that can reorder transactions.

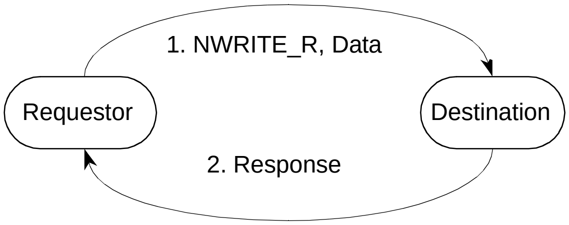


Figure 6. NWRITE_R Transaction

2.2 Using RapidIO on MPC8540 and MPC8560

This section provides information regarding the implementation of RapidIO on MPC8540/60 devices. It also provides information on other parts of the MPC8540/60 devices (for example, the MMU), which although not part of the RapidIO interface, must be considered in the use of RapidIO. Further information can be found in the references.[2]

The MPC8540/60 has an 8-bit parallel RapidIO interface that conforms to the *Parallel RapidIO Interconnect Specification.*[1] This RapidIO interface can operate at up to 500 MHz. It has a theoretical unidirectional peak bandwidth at 8 Gbps, and a theoretical bi-directional peak of 16 Gbps.

The RapidIO controller on the MPC8540/60 is partitioned into inbound and outbound blocks. These blocks are further divided into three implementation layers which correspond loosely to the logical, common transport and physical layers of the RapidIO interconnect specification. Users access the RapidIO interface mainly at the logical layer and transport layers. The user does not control the physical layer directly.

The RapidIO implementation on MPC8540/60 also has a messaging unit which implements Part II of *Parallel RapidIO Interconnect Specification.*[1] See Freescale application note AN2741, *Using the RapidIO Messaging Unit on PowerQUICC III* for more information.

2.2.1 Mapping Memory to the RapidIO Interface

The accesses to the RapidIO interface are memory mapped. Therefore, as well as considering the specifics of the RapidIO interface, it is also necessary to examine the way in which memory accesses are re-directed to the RapidIO interface.

2.2.1.1 Memory Management Unit

The MPC8540/60 supports demand paged virtual memory, as well as other memory management schemes that depend on precise control of effective-to-physical address translation and flexible memory protection. To achieve this, the memory management unit (MMU) makes use of software managed translation lookaside buffers (TLBs) that support variable-size pages, with per-page properties and permissions.

The MMU is not part of the RapidIO interface. However, it is noted here, as it must be made aware of any area of memory which is translated to any interface, including RapidIO. Therefore before being able to bring up a RapidIO system, a TLB entry must exist to cover the area of the memory map which is used for RapidIO transactions. This entry may be initialized by a bootloader, or a debugger. However, if a TLB entry has not been initialized to cover the region of the memory map to be used for RapidIO, then it is the responsibility of the bring-up software to do this.

The TLB is initialized using the MMU assist registers (MAS0–MAS4 and MAS6). The MMU assist registers are initialized with information about the TLB entry (for example the required address range, size, and permissions), and the **tlbwe** instruction is used to write that information into the TLB.

2.2.1.2 Local Memory Map and Local Access Windows

The local memory map on the MPC8540/60 refers to the 32-bit address space seen by the processor and the internal DMA engines as they access memory and I/O space. Within this map there exists memory mapped configuration, control and status registers as well as all memory accessed by the DDR SDRAM, local bus memory controllers, and other interfaces.

All addresses used by the system with the exception of configuration space (mapped by CCSRBAR), and the on-chip SRAM regions (mapped by L2SRBAR), must be mapped by a local access window (LAW).

The local area windows are not specific to RapidIO. However they are noted here as a LAW must exist to cover the area to be used by RapidIO. This may be initialized by a bootloader, or a debugger. However, if a LAW has not been initialized to cover the region of the memory map to be used for RapidIO then it is the responsibility of the bring-up software to do this.

A set of eight local access windows is used to define the local memory map. Each local window maps a region of memory to a particular target interface, such as the DDR SDRAM controller or the RapidIO controller. No address translation is performed by the local access windows. Each window can be configured in size from 4 Kbytes to 2 GBytes.

2.2.2 RapidIO ATMU (Address Translation and Mapping Unit)

The RapidIO controller on the MPC8540/60 is split into outbound and inbound blocks. The outbound block uses RapidIO outbound windows (ROWs) to translate an address from the local address space to that of RapidIO. The inbound block uses RapidIO inbound windows (RIWs) to translate an address from the external address space of RapidIO to the local address space understood by the internal interfaces.

2.2.2.1 RapidIO Outbound ATMU Windows

RapidIO outbound ATMU windows (ROWs) perform the mapping from the internal 32-bit address space to the 34-bit address space of RapidIO. ROWs also attach attributes such as transaction type and priority level. There are nine RapidIO outbound ATMU windows (0–8). Window 0 is always enabled and is used as the default window if the address does not match one of the other eight.

The default RapidIO Outbound Window is defined by two registers: the translation address register (ROWTAR*n*) and the attributes register (ROWAR*n*). The other ROWs also have a base address register (ROWBAR*n*).

The ROWBAR register contains the base address of the window in the local memory address space. The ROWTAR contains the RapidIO base address to which these transactions are translated. The ROWAR register contains information about the window (for example, its size) and the types of transactions that are generated by it (for example, NWRITE, NWRITE_R, or maintenance write).

The ROWTAR register takes one of two formats depending on the type of transaction to be executed. For regular transactions (such as NREAD, NWRITE, and NWRITE_R) the ROWTAR contains the device ID of the target, and bits 0–21 of the translated address. For maintenance transactions, which do not have an address, the ROWTAR contains the device ID of the target, the hopcount, and the upper bits of the maintenance offset into the CAR/CSR block.

In regular transactions, the RapidIO address is created by concatenating the translation address in the ROWTAR with the transaction offset (that is, the offset of the transaction from the base address of the ROW). With the maintenance transactions, the maintenance offset is created from a combination of the 12-bit configuration field in the ROWTAR and the transaction offset. Further information on the operation of the maintenance window can be found in Appendix A

2.2.2.2 RapidIO Inbound ATMU Windows

RapidIO Inbound ATMU windows (RIWs) perform the address translation from the 34-bit external RapidIO address space to the 32-bit internal address space. These inbound ATMU windows also attach attributes such as transaction type and target interface to the transaction. There are five inbound ATMU windows (0–4). Window 0 is always enabled and is used as the default window if the address does not match one of the other four windows.

The default RapidIO inbound window is defined by two registers: the translation address register (RIWTAR*n*) and the attributes register (RIWAR*n*). The other RIWs also have a base address register (RIWBAR*n*).

The RIWBAR register contains the base address of the window in RapidIO address space. The RIWTAR contains the local base address to which these transactions are translated. The RIWAR register contains information about the window (size, for example) and the types of transactions which are generated by it (for example, snoop or not snoop local processor).

2.2.2.3 Inbound ATMU Local Configuration Space Window

There is an additional inbound window that can be used by external RapidIO devices to access the local configuration, control and status registers (CCSR) memory.[2] The base address of this 1 Mbyte window is defined in the local configuration space base address 1 command and status register (LBSBA1CSR). Incoming RapidIO reads and writes, which match this window, are redirected, at the same offset, into the CCSR area. This local configuration space window has the highest priority for incoming translation.

2.3 Example Target Hardware

In Section 3, “Bring-up Procedure,” this document outlines a procedure to bring up a basic RapidIO system. Examples illustrate the steps in that procedure. This section gives information about the systems referenced in the examples.

It is assumed that these systems either run a bootloader and then the bring-up application is downloaded and run from this bootloader, or the systems are initialized by a debugger and the code is downloaded and run across JTAG. It is therefore assumed that most of the setup and initialization of the system (for example, CCSRBAR, DDR, and Flash) has already been completed. However, no assumptions are made regarding the initialization of the RapidIO interface.

In both of these examples the agent processors boot over RapidIO. However, it should be noted that this is not a requirement for RapidIO agents. In some system, the agent may have its own non-volatile memory from which to boot.

2.3.1 Point-to-Point System

The simplest system that can be built using RapidIO consists of two RapidIO end-points directly connected. A point-to-point system such as this is used to illustrate the examples in this application note. This system consists of two MPC8540/60 devices communicating over 8-bit parallel RapidIO.

In this system, one of the MPC8540/60 processors is configured as a RapidIO host, processor 1. This is the processor on which the bring-up application is executed. The other processor in the system, processor 2, is configured as a RapidIO agent. The host processor has a Flash device attached to its local bus, containing the boot image for the host and agent processors. The agent device is configured to boot across RapidIO and is in boot hold off mode, which prevents the core from booting until the processor has been configured by an external master, in this case the host.

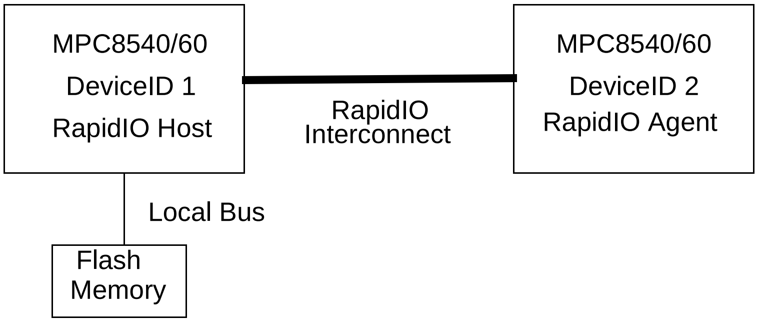


Figure 7. Point-to-Point Example System

The system architecture and configuration are fixed. So there is no requirement for a bring-up application for this system to satisfy all the requirements of the RapidIO Trade Association SBTG discovery process.[4] A software application could take advantage of the fixed nature of the system and the resulting assumptions below, to greatly simplify the discovery and bring-up process.

- There is only one host in the system.
- The host has a RapidIO device ID of 0x1.
- Both processors are MPC8540/60 devices.
- The agent device has a default RapidIO device ID of 0xFF.
- After discovery, processor 2 has a RapidIO device ID of 0x2.

The Flash device on the host processor contains one boot image for both processors. The 4 Mbyte boot image is at address 0xFFC0_0000–0xFFFF_FFFF.

2.3.2 Fabric-Based System

In many applications, RapidIO endpoints do not communicate point-to-point as described in the previous section, but through a RapidIO switch fabric. The fabric-based system used to illustrate the examples in this application note consists of four MPC8540/60 devices connected to a Tundra Tsi500, four port 8-bit parallel RapidIO switch.[3]

In this system, one of the MPC8540/60 processors is configured as a RapidIO host, processor 1. It is the host processor on which an application to bring up the system is run. This host processor has a Flash device attached to its local bus, containing the boot images for the host and agent processors.

The remaining three processors (processors 2, 3 and 4) do not have any Flash associated with them, and must be booted across RapidIO, accessing the contents of the host's Flash device. To achieve this, the processors are configured as RapidIO agents, are configured to boot across RapidIO, and are configured in CPU boot holdoff mode which prevents the core from booting until the processor has been configured by an external master.

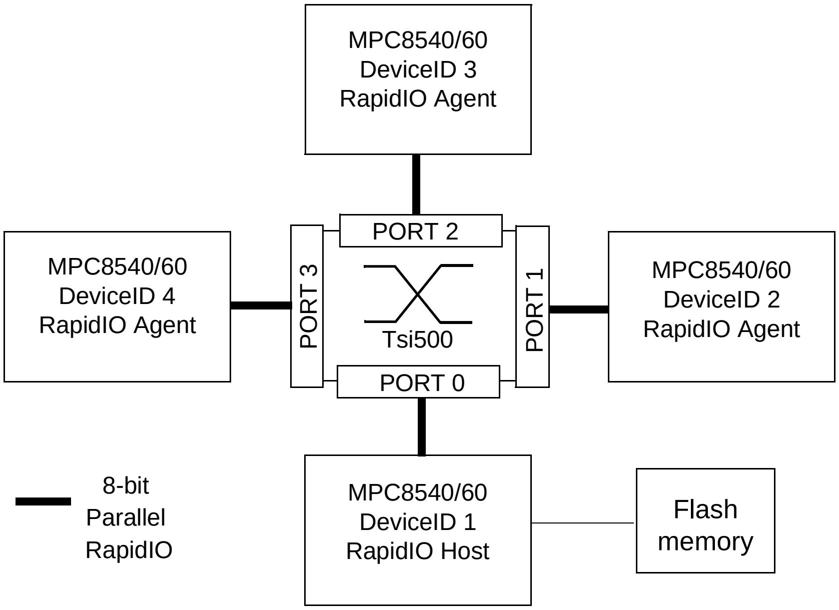


Figure 8. Fabric-Based Example Hardware and Configuration

The system architecture and configuration are fixed. So there is no requirement for a bring-up application on this system to satisfy all the requirements of the RapidIO SBTG discovery process.[4] An application could take advantage of the fixed nature of the system, and the resulting assumptions listed below to greatly simplify the discovery and bring-up process.

Assumptions:

1. There is only one host in the system
2. The host has a RapidIO device ID of 1
3. The host is connected to Port0 of the Tsi500 switch
4. There is only one switch in the system, and it is a Tsi500
5. All other processors are MPC8540/60 devices
6. The agents all have default RapidIO device ID of 0xFF

After the discovery process:

7. The agent on port 1 has RapidIO device ID 2
8. The agent on port 2 has RapidIO device ID 3
9. The agent on port 3 has RapidIO device ID 4

The Flash device on the host processor contains two boot images. The host’s 4 Mbyte boot image is at addresses 0xFFC0_0000–0xFFFF_FFFF, the agents' 4 Mbyte boot image is at addresses 0xFF80_0000–0xFFBF_FFFF.

3 Bring-up Procedure

This section describes a procedure which may be followed to bring up basic RapidIO capability on a MPC8540/60 system. This procedure may not apply to all possible systems, but an application using this procedure was created and tested on both of the example hardware systems described in Section 2.3, “Example Target Hardware.” In both cases, this procedure was executed on the host to discover and bring up the other elements in the system.

In the descriptions which follow, the steps are organized into 5 basic groups, each covered in a separate section.

Section 3.1, “Configure the Local Processor,” covers the steps required to configure and check the local processor before any attempt is made to generate RapidIO traffic, namely: set up a TLB entry, set up a local area window, check training, and set up a maintenance window.

Section 3.2, “Discover Other Devices in the System,” covers the discovery of all the other elements of the system. This is achieved using only maintenance transactions. This section includes steps to identify adjacent devices, set up switch (if required), and discover other endpoints. It is important to note that this section assumes a limited and fixed configuration system (as described in Section 2.3, “Example Target Hardware”). The procedure described in this section does not satisfy all the requirements of the RapidIO SBTG documentation.[4]

Section 3.3, “Enable Access to Remote Configuration Space,” describes the steps that must be taken to enable the local processor (host) to access the local configuration space of the remote device (agent). This consists of steps to set up the incoming window on the agent that re-directs RapidIO transactions to the local configuration space, and sets up an outbound window on the host to map outgoing transactions onto that window.

Section 3.4, “Boot over RapidIO,” describes how an agent can be configured to permit it to boot over RapidIO (that is, the agent boots from Flash attached to host). This includes steps to set up an inbound window on the host to capture the incoming boot reads, and to configure and release the agent to execute its boot procedure.

Section 3.5, “Enable Memory Reads and Writes,” describes how the host and agent could be set up to enable host access to the agent’s memory space. This section describes steps to set up the outbound window on the host and set up the inbound window on the agent.

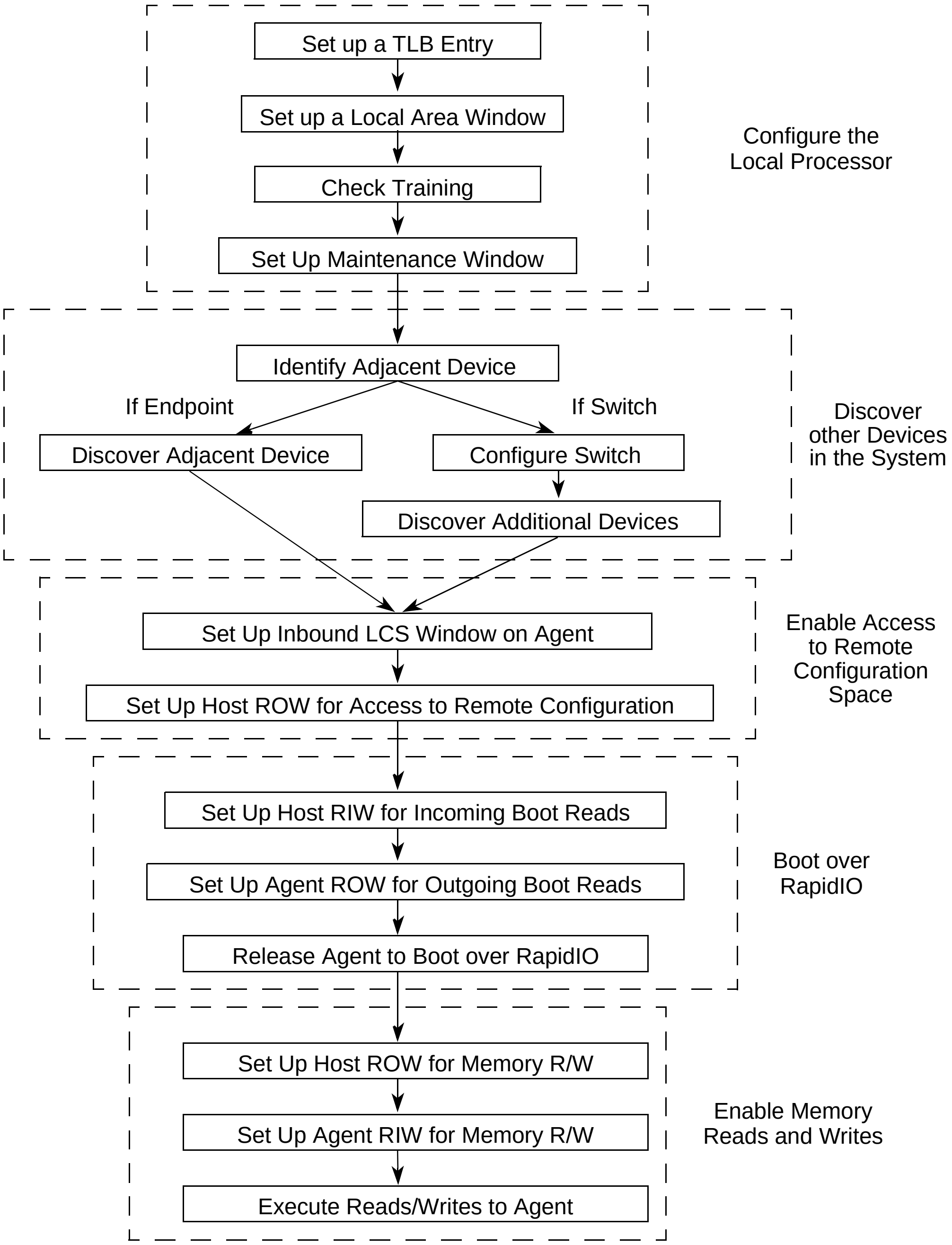

Set up a TLB Entry
Set up a Local Area Window
Check Training
Set Up Maintenance Window
Configure the Local Processor
Identify Adjacent Device
If Endpoint
If Switch
Discover Adjacent Device
Configure Switch
Discover Additional Devices
Discover other Devices in the System
Set Up Inbound LCS Window on Agent
Set Up Host ROW for Access to Remote Configuration
Enable Access to Remote Configuration Space
Set Up Host RIW for Incoming Boot Reads
Set Up Agent ROW for Outgoing Boot Reads
Release Agent to Boot over RapidIO
Boot over RapidIO
Set Up Host ROW for Memory R/W
Set Up Agent RIW for Memory R/W
Execute Reads/Writes to Agent
Enable Memory Reads and Writes

Figure 9. Outline of Procedure for Simple RapidIO System Bring-Up

3.1 Configure the Local Processor

Before any attempt can be made to generate RapidIO traffic, there are a number of steps that must be completed on the local (host) processor.

3.1.1 Set up a TLB Entry

The first step in the procedure involves setting up a TLB entry to cover the area of memory used for RapidIO accesses. This may or may not be necessary, depending on the configuration set by the bootloader or debugger.

In the example below, there is the minimum configuration for a 256 Mbyte entry, covering address range 0xC000_0000–0xCFFF_FFFF. This area is used throughout the examples to access RapidIO. In this example, entry 3 of TLB1 is used; however any unused entry in TLB1 (some entries are used by bootloader/debugger configuration to cover CCSR, Flash, DDR and so forth) could be used for this purpose.

The TLB entry is created by initializing the MAS0–MAS4 registers with information regarding the area of memory, then loading that information into the TLB.

MAS registers are special purpose registers, and should be initialized using the **mtspr** instruction. For example, to load the value required into MAS0 (SPR 0x270), the following sequence may be required.

```
asm(“lis 3, 0x1003”);
asm(“ori 3, 3, 0x0000”);
asm(“mtspr 0x270, 3”);
```

3.1.1.1 Initialize MAS0

	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
Field	—			TLB SEL	—								ESEL			
Setting	0x1				0x00								0x3			

	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
Field	—															NV
Setting	0x000												0x0			

Figure 10. MAS0 Setting for RapidIO Mapping

MAS0 contains the MMU Read/Write and replacement control. The settings in the example have the following definitions:

Table 2. MAS0 Field Descriptions and Settings

Bits	Name	Description and Setting
35	TLBSEL	Selects TLB for access 1 TLB1
44–47	ESEL	Entry select. Number of entry in selected array to be used for **tlbwe**. This field is also updated on TLB error exceptions (misses), and **tlbsx** hit and miss cases. 0011 This becomes entry 3.
63	NV	Next victim. Next victim bit value to be written to TLB0[NV] on execution of **tlbwe**. This field is also updated on TLB error exceptions (misses), **tlbsx** hit and miss cases and on execution of **tlbre**. 0 A next victim has not been identified.

3.1.1.2 Initialize MAS1

	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
Field	V	IPROT	—						TID							
Setting	0x8				0x0				0x00							

	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
Field	—			TS	TSIZE				—							
Setting	0x0				0x9				0x00							

Figure 11. MAS1 Setting for RapidIO Window

MAS1 contains the descriptor context and configuration control. The settings in the example have the following definitions:

Table 3. MAS1 Field Descriptions and Settings

Bits	Name	Description and Setting
32	V	TLB valid bit 1 This TLB entry is valid.
33	IPROT	Invalidate protect. Set to protect this TLB entry from invalidate operations due the execution of **tlbiva**[**x**] (TLB1 only). 0 Entry is not protected from invalidation
40–47	TID	Translation identity. An 8-bit field that defines the process ID for this TLB entry. All zeros. It is a global entry and can be used by any process.
51	TS	Translation space. This bit is compared with the IS or DS fields of the MSR (depending on the type of access) to determine if this TLB entry may be used for translation. 0 Bit is not set.
52–55	TSIZE	Translation size. Defines the TLB entry page size. 1001 This entry has a page size of 256 Mbytes

3.1.1.3 Initialize MAS2

	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
Field	EPN															
Setting	0xC000															

	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
Field	EPN				—		SHAREN	—		X0	X1	W	I	M	G	E
Setting	0x0				0x008											

Figure 12. MAS2 Setting for RapidIO Window

MAS2 contains the effective page number and page attributes. The settings have the following definitions:

Table 4. MAS2 Field Descriptions and Settings

Bits	Name	Description and Setting
32–51	EPN	Effective page number. Depending on page size, only the bits associated with a page boundary are valid. Bits that represent offsets within a page are ignored and should be cleared. 0xC0000 Effective address of this space starts at address 0xC000_0000
54	SHAREN	Enables cache fills to use shared cache state for this page. 0 Cache fills do not use shared cache for this page
57	X0	Implementation-dependent page attribute
58	X1	Implementation-dependent page attribute
59	W	Write-through 0 This page is considered write-back with respect to the caches in the system.
60	I	Caching-inhibited 1 The page is considered caching-inhibited. All loads and stores to the page bypass the caches and are performed directly to main memory.
61	M	Memory coherence required 0 Memory coherence is not required.
62	G	Guarded 0 Accesses to this page are not guarded and can be performed before it is known if they are required by the sequential execution model.
63	E	Endianness. Determines endianness for the corresponding page. 0 The page is accessed in big-endian byte order.

3.1.1.4 Initialize MAS3

	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
Field	RPN															
Setting	0xC000															

	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
Field	RPN				—		U0-U3				UX	SX	UW	SW	UR	SW
Setting	0x0				0x03F											

Figure 13. MAS3 Setting for RapidIO Window

MAS3 contains the real page number and access control. The settings in the example have the following definitions:

Table 5. MAS3 Field Descriptions and Settings

Bits	Name	Description and Setting
32–51	RPN	Real page number. The real address of this space starts at 0xC000_0000. 0xC000
54–57	U0-U3	User attribute bits. Associated with a TLB entry and can be used by system software. All zeros
58	UX	Permission bit. 1 User mode has execute permission.
59	SX	Permission bit. 1 Supervisor mode has execute permission.
60	UW	Permission bit. 1 User mode has write permission.
61	SW	Permission bit. 1 Supervisor mode has write permission.
62	UR	Permission bit. 1 User mode has read permission.
63	SR	Permission bit. 1 Supervisor mode has read permission.

3.1.1.5 Load Information into the TLB

The final step is to execute a **sync** instruction to ensure that all the MAS registers have been written, a **tlbwe** instruction to load the information into the TLB, and a final **sync** to ensure that the TLB entry has been created before continuing. For example:

```
asm(“sync”);
asm(“tlbwe”);
asm(“sync”);
```

3.1.2 Set Up a Local Area Window

The next step is to set up a local area window (LAW) to re-direct memory accesses within a certain address range to the RapidIO interface. This may or may not be necessary, depending on the configuration set by the bootloader or debugger.

In the example, the LAWBAR and LAWAR registers are configured to ensure that a 256 Mbytes block of memory space, covering address range 0xC000_0000–0xCFFF_FFFF, is re-directed to the RapidIO interface. The LAW registers are memory mapped within the CCSR area, and therefore can be read and written using standard memory reads and writes.

3.1.2.1 Set LAWBAR

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—												BASE_ADDR			
Setting	0x000												0xC			

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	BASE_ADDR															
Setting	0x0000															

Figure 14. LAWBAR Register for RapidIO Window

The local area window base address (LAWBAR) sets the base address of this window. The setting in the example has the following definition:

Table 6. LAWBAR Field Description and Setting

Bits	Name	Description and Setting
12–31	BASE_ADDR	Identifies the 20 most-significant address bits of the base of local access window *n*. 0xC000 The real address of this space starts at 0xC000_0000.

3.1.2.2 Set LAWAR

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	EN	—							TRGT_IF				—			
Setting	0x80								0xC				0x0			

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—										SIZE					
Setting	0x00								0x1B							

Figure 15. LAWAR Record for RapidIO Window

The local area window attributes register (LAWAR) enables this window, sets the interface that the transactions are directed to, and sets the size of the window. The settings in the example have the following definitions:

Table 7. LAWAR Field Descriptions and Settings

Bits	Name	Description and Setting
0	EN	1 Window is enabled.
8–11	TRGT_IF	Identifies the target interface ID when a transaction hits in the address range defined by this window. 1100 Target interface is RapidIO
26–31	SIZE	Identifies the size of the window from the starting address. 0x1B Window is 256 Mbytes.

3.1.3 Check Training

Before attempting to create any RapidIO traffic, it is necessary to ensure that the RapidIO interface has trained successfully with the adjacent device.[4] This is determined by examination of the port error and status command and status register (PESCSR).

The PESCSR[PP] bit defines input clock toggling and the PESCSR[PO] bit defines port initialization. Both of these bits are set if the RapidIO port has trained correctly. If these bits are not set, no further attempts should be made to use the RapidIO interface.

3.1.4 Set Up Maintenance Window

The next step in the bring-up process is the creation of a RapidIO outbound window to generate RapidIO maintenance transactions.

In the example, the ROWBAR, ROWTAR and ROWAR registers are configured to initialize a 4 Mbyte RapidIO window at the very bottom of the area of the RapidIO space, that is, 0xC000_0000–0xC040_0000. Reads and writes within this window then generate RapidIO maintenance read and write transactions.

3.1.4.1 Set ROWBAR

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—												BADD			
Setting	0x000												0xC			

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	BADD															
Setting	0x0000															

Figure 16. Maintenance Window ROWBAR Register

The base address register (ROWBAR), defines the start address of the window. The settings in the example have the following definitions:

Table 8. ROWBAR Field Description and Setting

Bits	Name	Description and Setting
12–31	BADD	Base address of outbound window. Source address that is the starting point for the outbound translation window. 0xC0000 Window starts at address 0xC000_0000.

3.1.4.2 Set ROWAR

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	EN	—			TFLOV		PCI	—					RDTYP			
Setting	0x80								0x0				0x7			

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	WRTYP				—						SIZE					
Setting	0x7				0x0				0x15							

Figure 17. Maintenance Window ROWAR Settings

The attributes register defines some attributes of the transactions that are created by this window; including the priority and transaction types. It also defines the size of the window. The settings in the example have the following definitions:

Table 9. ROWAR Field Descriptions and Settings

Bits	Name	Description and Setting
0	EN	Window address translation enable. 1 This RapidIO outbound window is enabled.
4–5	TFLOV	Transaction flow level priority of transaction 00 Lowest priority transaction request flow.
6	PCI	Follow PCI transaction ordering rules 0 Do not follow PCI transaction ordering rules.
12–15	RDTYP	Read transaction type. 0111 Transaction type to run on RapidIO interface if access is a read.
16–19	WRTYP	Write transaction type. 0111 Transaction type to run on RapidIO interface if access is a write.
26–31	SIZE	Outbound window size. 0x15 4 Kbytes

3.1.4.3 Set ROWTAR

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—		TRGTID								—		HOP_COUNT			
Setting	0x3FC												0x0			

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	HOP_COUNT				CFG_OFFSET											
Setting	0x0				0x000											

Figure 18. Maintenance Window ROWTAR Settings

The RapidIO outbound window translation address register (ROWTAR) defines the starting address in RapidIO space for hits within this window. The ROWTAR takes different forms for different transaction types. For maintenance transactions, it is of the form shown above. In the example, the settings have the following definitions:

Table 10. ROWTAR Field Descriptions and Settings

Bits	Name	Description and Setting
2–9	TRGTID	0xFF Target ID for RapidIO Packet
12–19	HOP_COUNT	All zeros. Hop count of maintenance transaction
20–31	CFG_OFFSET	All zeros. Upper bits of maintenance offset.

NOTE

This is just an initial value for the ROWTAR. Users must ensure that the ROWTAR contains the correct destination ID, hopcount and offset value for their transaction.

For further information on the operation of the maintenance window, refer to Appendix A

3.2 Discover Other Devices in the System

This application note does not cover all the requirements of the discovery process described in the RapidIO specifications.[4] It provides examples of simplified discovery processes that could be used with simple systems (that is, with only one host, and not more than one switch). Two target hardware systems, described in Section 2.3, “Example Target Hardware,” are used to illustrate the process.

3.2.1 Identify Adjacent Device

The first RapidIO transaction that should be executed is a maintenance read to the device identity capability register (DIDCAR) of the adjacent device. The DIDCAR, at maintenance offset 0x00, contains a device identity field and a device vendor identity field. The RapidIO consortium maintains a list of the assigned values. Reading this register therefore enables the identification of the adjacent device. This is achieved by executing a read instruction within the area covered by the maintenance window. The attributes of this maintenance read are as follows:

> Destination ID = 0xFF. The RapidIO specifications state that non-boot-code and non-host devices should initially have a device ID of 0xFF.[4]
>
> Hopcount = 0x00. If adjacent device is a switch, it must be accessed with hopcount 0.
>
> Maintenance offset = 0x00

This returned DIDCAR value should then be compared to the list of DIDCAR values of all the devices which are recognized by the application. In the examples, this consists of PowerQUICC III processor and Tsi500 switch only. If it is determined that the adjacent device is another processor (for example, a MPC8540/60) then the application can discover that processor directly. If it is determined that the adjacent device is a switch (for example a Tsi500 switch) then this switch must be configured correctly before proceeding with the discovery of the processors beyond it.

3.2.2 Discover other Devices—Point-to-Point

This section describes the procedure which could be followed if the adjacent device is a RapidIO endpoint.

3.2.2.1 Read Back the HBDIDLCSR of Adjacent Device

Each RapidIO end point has a host base device lock ID CSR (HBDIDLCSR) register, at Maintenance Offset 0x68, which contains the device ID of the element in the system which is responsible for its configuration. This provides a locking mechanism and a host should only proceed with the configuration of an agent device if it gains the lock.

The HBDIDLCSR can be read by executing a maintenance read with the following attributes:

Destination ID = 0xFF, This agent device has a Device ID of 0xFF.

Hopcount = 0xXX, Hopcount is ‘don’t care’ as there are no switches in the system.

Maintenance Offset = 0x68, HBDIDLCSR is at offset 0x68.

If the agent has not been locked, then the default value of this HBDIDLCSR is 0x0000_FFFF. If this HBDIDLCSR is read back as any other value, it indicates that the device has already been discovered, and the mechanism to deal with this situation is beyond the scope of this application note.

3.2.2.2 Attempt to Lock HBDIDLCSR of Adjacent Device

Assuming that the lock for the device has not already been taken, the processor running this application should identify itself as the device responsible for initializing the agent. To achieve this it should copy its own device ID (determined by reading its own, memory mapped, BDIDCSR) into the HBDIDLCSR of the agent device. This maintenance write would have the following attributes:

Destination ID = 0xFF, This agent device has a Device ID of 0xFF.

Hopcount = 0xXX, Hopcount is ‘don’t care’ as there are no switches in the system.

Maintenance offset = 0x68, HBDIDLCSR is at offset 0x68.

3.2.2.3 Confirm That the Adjacent Device Has Accepted Lock.

The HBDIDLCSR should be read back to confirm that it has been updated to the host's device ID. If not, the lock has not been accepted, the host should not proceed any further with the configuration of this device.

The attributes of this maintenance read are:

Destination ID = 0xFF, This agent device has Device ID of 0xFF.

Hopcount = 0xXX, Hopcount is ‘don’t care’ as there are no switches in the system.

Maintenance offset = 0x68, HBDIDLCSR is at offset 0x68.

If the lock has not been accepted, it suggests that there is another host in the system, which is beyond the scope of this example. Refer to interoperability specification for information regarding the bring-up requirements for multi-host systems.[4]

3.2.2.4 Update the Device ID

It is assumed that the agent device has an initial device ID of 0xFF. Once the agent device has been discovered by the host, it must be allocated a permanent device ID. This is achieved by writing the desired device ID value into the base device ID command and status register (BDIDCSR), at configuration offset 0x60.

In this example, it is assumed that the host device has a device ID of 0x01. The host allocates the agent a device ID of 0x02. The maintenance write to achieve this update has the following attributes:

Destination ID = 0xFF, This agent device has Device ID of 0xFF.

Hopcount = 0xXX, Hopcount is ‘don’t care’ as there are no switches in the system.

Maintenance Offset = 0x60, BDIDCSR is at offset 0x60.

3.2.2.5 Read Back from Updated Device ID

Once the device ID has been set, it should be verified that the device can be accessed using that device ID as the destination. The ROWTAR must be updated with the new device ID (as shown below) and some known value (for example the BDIDCSR) should be read back from the agent. Software should confirm that this returns the expected value.

The maintenance read should have the following attributes:

Destination ID = 0x02, This agent device has Device ID of 0x02.

Hopcount = 0xXX, Hopcount is ‘don’t care’ as there are no switches in the system.

Maintenance offset =0x60, BDIDCSR is at offset 0x60.

3.2.3 Discover Other Devices—Fabric

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—		TRGTID								—		HOP_COUNT			
Setting	0x008												0x0			

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	HOP_COUNT				CFG_OFFSET											
Setting	0x0				0x000											

If there is a switch in a system, the discovery process is slightly different, as the switch must be configured before the devices beyond it are discovered. The following section details the process which was followed for Tsi500 switch. Steps may need to be added or removed for other RapidIO switches.

3.2.3.1 Check Switch Port

It is necessary to determine to which port of the switch the device running the bring-up application is connected.

In the Tsi500 switch example, this is achieved by reading the switch port information CAR. A read to this register returns the port number from which it was read. In the target hardware example the host processor responsible for this discovery process is attached to Port 0.

The maintenance read has the following attributes:

Destination ID = 0xXX, ID is ‘don’t care’ when accessing first switch in the system.

Hopcount = 0x00, Transaction applies to first switch.

Maintenance offset = 0x14, Offset of switch port information CAR.

3.2.3.2 Read Back the HBDIDLCSR of Switch

Each RapidIO device has a register which contains the device ID of the element in the system which is responsible for its configuration. This provides a locking mechanism and a device should only proceed with the configuration of a device if it gains the lock.

The HBDIDLCSR of the switch can be read using a maintenance read with the following attributes:

Destination ID = 0xXX, ID is ‘don’t care’ when accessing first switch in the system.

Hopcount = 0x00, Transaction applies to first switch

Maintenance offset = 0x68, Offset to HBDIDLCSR

If this device has not yet been discovered, the default value for the HBDIDLCSR is 0xFFFF.

3.2.3.3 Attempt to Lock HBDIDLCSR of Switch

To identify the processor running this application as the device responsible for initializing the switch, its device ID should be written into the switch’s HBDIDLCSR. This is achieved using a maintenance write with the following attributes:

Destination ID = 0xXX, ID is ‘don’t care’ when accessing first switch in the system.

Hopcount = 0x00, Transaction applies to first switch.

Maintenance offset = 0x68, Offset to HBDIDLCSR

3.2.3.4 Confirm that Switch has Accepted Lock

The HBDIDLCSR register should be read back to confirm that it has been updated to the host's device ID. If not, the lock has not been accepted, the host should not proceed any further with the configuration of the switch.

If the lock is not accepted, it suggests that there is another host in the system, which is beyond the scope of this example. Refer to SBTG specification for information regarding the bring-up requirements for multi-host systems.[4]

The HBDIDLCSR can be read using a maintenance read with the following attributes:

Destination ID = 0xXX, ID is ‘don’t care’ when accessing first switch in the system.

Hopcount = 0x00, Transaction applies to first switch.

Maintenance offset = 0x68, Offset to HBDIDLCSR

3.2.3.5 Route Responses Back to Host

To ensure that responses issued by the processors beyond the switch are correctly routed to the requesting processor, it is necessary to initialize the routing information of the switch with that information.

In the example, the processor executing the bring-up software has a device ID of 1, and is attached to port 0. Therefore it is necessary to update the routing tables to indicate that all packets for device ID 1 are routed to port 0. Indeed, because the example hardware has a fixed configuration and a fixed device number allocation, it is possible to update the routing tables with all the required information, even before the devices have been discovered.

In the Tsi500 switch example, each port has its own routing table, which is applied to all the packets arriving at that port. Each routing table has 32, 32-bit entries. Each 32-bit entry routes the packets for 8 different destination IDs (one nibble per destination ID). For example, DESTID_0_LKUP contains the routing information for all packets with destination IDs 0–7.

The figure below shows a configuration that routes all packets with a destination ID of 1 to port 0, all those with a destination ID of 2 to port 1, all those with a destination ID of 3 to port 2, and all those with a destination ID of 4 to port 3. In this example, packets with destination ID 0,5,6 or 7 are not routed, and generate an error if received on this port.

<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td></tr>
<tr><td>Field</td><td colspan="4">DEST_ID_PORT_07</td><td colspan="4">DEST_ID_PORT_06</td><td colspan="4">DEST_ID_PORT_05</td><td colspan="4">DEST_ID_PORT_04</td></tr>
<tr><td>Setting</td><td colspan="4">0x8</td><td colspan="4">0x8</td><td colspan="4">0x8</td><td colspan="4">0x3</td></tr>
<tr><td></td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td>Field</td><td colspan="4">DEST_ID_PORT_03</td><td colspan="4">DEST_ID_PORT_02</td><td colspan="4">DEST_ID_PORT_01</td><td colspan="4">DEST_ID_PORT_00</td></tr>
<tr><td>Setting</td><td colspan="4">0x2</td><td colspan="4">0x1</td><td colspan="4">0x0</td><td colspan="4">0x8</td></tr>
</table>

Figure 19. DESTID_0_LKUP Setting

The routing table for each of the four ports must be updated with this information using four maintenance write transactions. These maintenance transactions have the following attributes:

Destination ID = 0xXX, ID is ‘don’t care’ when accessing first switch in the system.

Hopcount = 0x00, Transaction applies to first switch.

Maintenance offset = 0x10A00 (port0), 0x30A00 (port1), 0x50A00 (port2) and 0x70A00 (port4)

3.2.3.6 Discover the Devices beyond the Tsi500 Switch

The next step is to identify which devices are connected to the other ports of the RapidIO switch. If any of these devices is another switch, or another RapidIO host then the full RapidIO discovery procedure must be executed.[4] The example scenario is simplified by the assumption that there is only one switch, and all other processors are configured as RapidIO agents.

The following steps should be executed for each of the ports on the switch (with the exception of the port to which the processor running this application is attached). In the example, this is repeated for the Tsi500 ports 1, 2 and 3.

NOTE

In the Tsi500 switch, the registers for each port have maintenance offsets 0x10xxx (port0), 0x30xxx (port1), 0x50xxx (port2) and 0x70xxx(port3).

3.2.3.6.1 Check Training on Port *N*

Ensure that the port under investigation (port *n*) is trained. If it is not trained, then the remaining steps are omitted for this iteration.

In the Tsi500 switch example, each of the four ports has a port training status register. These registers can be examined to determine if each port has successfully completed the RapidIO training procedure with an adjacent device.[1]

This maintenance transaction has the following attributes:

Destination ID = 0xXX, ID is ‘don’t care’ when accessing first switch in the system.

Hopcount = 0x00, Transaction applies to first switch.

Maintenance offset =0x30068 (port1), 0x50068 (port2) or 0x70068 (port4)

3.2.3.6.2 Route Packets for Device ID 0xFF to Port *N*

The agents initially have a device ID of 0xFF. The routing table must be updated to ensure that packets with this destination are directed to the port under investigation.

In the Tsi500 switch example, the routing table for port 0, which contains the routing information for packets arriving at Port 0, must be updated. Entry DESTID_248_LKUP contains the routing information for destination IDs 0xF8 - 0xFF. It must be updated to direct the packets with destination ID of 0xFF to the Tsi500 port currently under investigation (port *n*).

For example, to direct packets arriving at port 0, with destination ID of 0xFF to port 3, the port 0 DESTID_248_LKUP register should be updated as shown.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	DEST_ID_PORT_FF				DEST_ID_PORT_FE				DEST_ID_PORT_FD				DEST_ID_PORT_FC			
Setting	0x3				0x8				0x8				0x8			

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	DEST_ID_PORT_FB				DEST_ID_PORT_FA				DEST_ID_PORT_F9				DEST_ID_PORT_F8			
Setting	0x8				0x8				0x8				0x8			

Figure 20. Port 0 DESTID_248_LKUP Setting while Discovering Device on Port 3

In this example, the routing information for packets with unexpected destination IDs is set to the default value of 0x8. These packets are not routed and the Tsi500 switch reports an error if they are received on this port.

This routing information update is achieved with a maintenance transaction with the following attributes:

Destination ID = 0xXX, ID is ‘don’t care’ when accessing first switch in the system.

Hopcount = 0x00, Transaction applies to first switch.

Maintenance offset = 0x10A7C (port0)

3.2.3.6.3 Read Back the DIDCAR of Device on Port *N*

The routing is now in place to permit access to the device which is connected to port *n* of the switch. The initial read should be to the DIDCAR of that device, to identify what device it is.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—		TRGTID								—		HOP_COUNT			
Setting	0x3FC												0xF			

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	HOP_COUNT				CFG_OFFSET											
Setting	0xF				0x000											

Figure 21. ROWTAR Setting for Accessing DIDCAR of Device on Port *N*

The read of the DIDCAR can be achieved with a maintenance read with the following attributes:

Destination ID = 0xFF, Accessing the agent with destination ID 0xFF

Hopcount=0xFF, Transaction bypasses switch.

Maintenance offset = 0x00, Offset to DIDCAR

3.2.3.6.4 Read Back the HBDIDLCSR of Device on Port *N*

Each RapidIO device has a register which contains the device ID of the element in the system which is responsible for its configuration. This provides a locking mechanism and a device should only proceed with the configuration of a device if it gains the lock.

The HBDIDLCSR can be read using a maintenance read with the following attributes:

Destination ID = 0xFF, Accessing the agent with destination ID 0xFF

Hopcount = 0xFF, Transaction bypasses switch.

Maintenance offset =0x68, Offset to HBDIDLCSR

If this device has not yet been discovered, the default value for the HBDIDLCSR is 0xFFFF.

3.2.3.6.5 Attempt to Lock HBDIDLCSR of Device on Port *N*

To identify the processor running this application as the device responsible for initializing this device, its device ID should be written into the HBDIDLCSR of the device on Port *n*. This is achieved using a maintenance write with the following attributes:

Destination ID = 0xFF, Accessing the agent with destination ID 0xFF

Hopcount = 0xFF, Transaction bypasses switch.

Maintenance offset = 0x68, Offset to HBDIDLCSR

3.2.3.6.6 Confirm that Device on Port *N* Has Accepted Lock

The HBDIDLCSR register should be read back to confirm that it has been updated to the host's device ID. If not, the lock has not been accepted, and the host should not proceed any further with the configuration of the device on this port.

If the lock is not accepted, it suggests that there is another host in the system, which is beyond the scope of this example. Refer to SBTG specification for information regarding the bring-up requirements for multi-host systems.[4]

The HBDIDLCSR can be read using a maintenance read with the following attributes:

Destination ID = 0xFF, Accessing the agent with destination ID 0xFF

Hopcount = 0xFF, Transaction bypasses switch.

Maintenance Offset = 0x68, Offset to HBDIDLCSR

3.2.3.6.7 Update the Device ID

It is assumed that all the agents have an initial device ID of 0xFF. Once they have been discovered by the host, the host must allocate a permanent device ID. This is achieved by writing the desired device ID value into the base device ID command and status register (BDIDCSR).

In the example, it is assumed that the host device, connected to Port 0, has a device ID of 0x01. The host allocates the device IDs of the other devices as follows: the device connected to port 1 is allocated a device ID of 0x02, the device connected to port 2 is allocated a device ID of 0x03 and the device connected to port 3 is allocated a device ID of 0x04.

The BDIDCSR can be updated by executing a maintenance write with the following attributes:

Destination ID = 0xFF, Accessing the agent with destination ID 0xFF
Hopcount = 0xFF, Transaction bypasses switch.
Maintenance Offset = 0x60, Offset to BDIDCSR

3.2.3.6.8 Update the Routing Table, All Ports

If the systems were operating with dynamic allocation of device IDs, the routing tables would have to be updated with that information at this stage in the procedure. However, in this example, there is a fixed routing table, which was initialized in Section 3.2.3.5, "Route Responses Back to Host," therefore there is no action necessary here.

3.2.3.6.9 Read Back from Updated Device ID

Once the device ID has been set, it should be verified that the device can be accessed using that device ID as the destination. That is, update the ROWTAR with the new device ID (example shown below) and read from the BDIDCSR.

The read of the BDIDCSR should be executed with the following attributes:

Destination ID = 0x02, 0x03 or 0x04, Accessing the agent with updated Device ID
Hopcount = 0xFF, Transaction bypasses switch.
Maintenance Offset = 0x60, Offset to BDIDCSR

Software should confirm that this read returns the expected value.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—		TRGTID								—		HOP_COUNT			
Setting	0x00C												0xF			

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	HOP_COUNT				CFG_OFFSET											
Setting	0xF				0x000											

Figure 22. ROWTAR for Accessing BDIDCSR on Device 3

3.3 Enable Access to Remote Configuration Space

There are two steps required to access the local configuration (CCSR) space on a remote processor: on the remote processor set up the LCSBA1CSR to accept RapidIO transactions in a certain address range, and on the local processor set up an outbound window with the correct address translation into the same range.

NOTE

Once the remote access to the remote configuration space has been enabled, it is assumed that all further accesses to the agent devices are by this method, rather than by maintenance transactions.

3.3.1 Setup Inbound LCS Window on Agent

In the example, the LCSBA1CSR of each of the processors is set up to redirect RapidIO arriving at the 1 Mbyte window, starting at RapidIO address 0x0_0100_0000. This is achieved by setting the LCSBA1CSR register as shown below.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—	LCSBA														—
Setting	0x0020															

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—															
Setting	0x0000															

Figure 23. LCSBA1CSR Settings

The settings in the example have the following definitions:

LCSBA = 0x0010, The 1 Mbyte LCS window starts at RapidIO address 0x0100_0000. (LCSBA contains the most significant 14 bits of the 34-bit address, but the top 2 extended bits are ignored in this example.)

The update of the LCSBA1CSR on the agent can be achieved using a maintenance write with the following attributes:

Destination ID = 0x02, 0x03 or 0x04, Depending on which agent is being accessed

Hopcount = 0xFF, Transaction bypasses switch

Maintenance offset = 0x5C, Offset to LCSBA1CSR

3.3.2 Setup Outbound Window on Host

A RapidIO outbound window should be set up on the host for each of the agent processors, to map outbound transactions into this address space, with the appropriate destination ID.

For example, a 1 Mbytes outbound window with base address 0xC400_0000 could be initialized on processor 1 to map to the CCSR locations on processor 2. This would be achieved as shown below.

3.3.2.1 ROWBAR

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—												BADD			
Setting	0x000												0xC			

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	BADD															
Setting	0x4000															

Figure 24. Remote CCSR ROWBAR Settings

The base address register (ROWBAR), defines the start address of the window. The settings in the example have the following definitions:

Table 11. ROBAR Field Description and Setting

Bits	Name	Description and Setting
12–31	BADD	Base address of outbound window. Source address that is the starting point for the outbound translation window. 0xC4000, Window starts at address 0xC400_0000

3.3.2.2 ROWAR

<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th></tr>
<tr><td>Field</td><td>EN</td><td colspan="3">—</td><td colspan="2">TFLOV</td><td>PCI</td><td colspan="5">—</td><td colspan="4">RDTYP</td></tr>
<tr><td>Setting</td><td colspan="8">0x80</td><td colspan="4">0x0</td><td colspan="4">0x4</td></tr>
</table>

<table>
<tr><th></th><th>16</th><th>17</th><th>18</th><th>19</th><th>20</th><th>21</th><th>22</th><th>23</th><th>24</th><th>25</th><th>26</th><th>27</th><th>28</th><th>29</th><th>30</th><th>31</th></tr>
<tr><td>Field</td><td colspan="4">WRTYP</td><td colspan="6">—</td><td colspan="6">SIZE</td></tr>
<tr><td>Setting</td><td colspan="4">0x4</td><td colspan="4">0x0</td><td colspan="8">0x13</td></tr>
</table>

Figure 25. Remote CCSR Window ROWAR Settings

The attributes register defines attributes of the transactions which are created by this window. This includes the priority and transaction types. It also defines the size of the window. The settings in the example have the following definitions:

Table 12. ROWAR Descriptions and Settings

Bits	Name	Description and Setting
0	EN	Window address translation enable. Note that for ROWAR0 this bit is read-only and hardwired to 1. 1 Address translation enabled
4–5	TFLOV	Transaction flow level priority of transaction 00 Lowest priority transaction request flow
6	PCI	Follow PCI transaction ordering rules 0 Do not follow PCI transaction ordering rules.
12–15	RDTYP	Read transaction type. Transaction type to run on RapidIO interface if access is a read. 0100 NREAD
16–19	WRTYP	Write transaction type. Transaction type to run on RapidIO interface if access is a write. 0100 NWRITE
26–31	SIZE	Outbound window size. Outbound window size n which is the encoded 2^{n+1}-byte window size. 0x13 Window is 1 Mbyte

3.3.2.3 ROWTAR

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—		TRGTID								TREXAD		TRAD			
Setting	0x008												0x0			

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	TRAD															
Setting	0x1000															

Figure 26. Remote CCSR ROWTAR*n* Settings

The settings in the example have the following definitions:

Table 13. ROWTAR*n* Descriptions and Settings

Bits	Name	Description and Setting
2–9	TRGTID	Target ID for RapidIO packet 0x02 Transactions have destination ID of 2.
10–11	TREXAD	Translation extended address of outbound window 00 The 2 extended address bits are both zero.
12–31	TRAD	Translation address of outbound window 0x01000 Outbound transactions start at address 0x0_0100_0000.

3.3.3 Confirm Access to Remote LCS

The software should then verify that the agent's local configuration space can be read in this manner. For example, with the ROW on the host and the LCS window on agent 2 set up as shown in the figures above, the following accesses should return the following values.

Reading the host processor's address 0xC400_0000 should return the CCSRBAR of processor 2.

Reading the host processor's address 0xC40C_0000 should return the DIDCAR of processor 2.

Reading the host processor's address 0xC40C_005C should return the LCSBA1CSR of processor 2.

3.4 Boot over RapidIO

If a device is to boot over RapidIO, the cfg_rom_loc configuration signal must be configured to direct accesses to the boot vector and default boot ROM region to the RapidIO interface, and the cfg_cpu_boot configuration signal must be configured to put the processor in boot holdoff mode

Given these configurations, there is a sequence of steps which must be completed by the host to configure the system and initiate the booting of the agent processor.

3.4.1 Prepare the Host Processor for Incoming Boot Reads

The host processor must provide an inbound RapidIO window that accepts the boot code reads arriving over RapidIO and re-directs them to the appropriate area of Flash memory.

With opportunities to adjust window sizes, translate addresses on the outgoing window of the agent and on the incoming window of the host, there are many different configurations that would permit the agent to boot over RapidIO. This section details one possibility.

The default boot ROM address range is 8 Mbytes; however in this example, the complete boot image for one processor is contained within 4 Mbytes. Therefore, a 4 Mbytes RapidIO inbound window captures the boot reads coming from the agent processors. It is assumed that there is no translation of these reads on the outbound window of the agent, therefore they appear in the RapidIO address range 0x0_FFC0_0000–0x0_FFFF_FFFF. In the inbound window these reads must be re-directed to the appropriate area of Flash.

3.4.1.1 RIWBAR

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—										BEXAD		BADD			
Setting	0x00								0x0				0xF			

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	BADD															
Setting	0xFC00															

Figure 27. RIWBAR Settings on Host for Incoming Boot Reads

The RapidIO inbound window base address register (RIWBAR) contains the base RapidIO address of the incoming window. The settings in the example have the following definitions:

Table 14. RIWBAR Field Descriptions and Settings

Bits	Name	Description and Setting
10–11	BEXAD	Base extended address of inbound window. 00 Extended bits in the base address are both 0
12–31	BADD	Base address of inbound window. Source address that is the starting point for the inbound translation window. 0xF_FC00 Base address bits 2–21 are 0xF_FC00

The combined meaning of these two fields is that the base address of the inbound RapidIO window is 0x0_FFC0_0000.

3.4.1.2 RIWAR

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	EN	—							TGINT				RDTYP			
Setting	0x8				0x0				0xF				0x4			

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	WRTYP				—						SIZE					
Setting	0x0				0x0				0x15							

Figure 28. RIWAR Settings on Host for Incoming Boot Reads

The RapidIO inbound window attributes register (RIWAR) defines the window size and other attributes for the translation. The settings in the example have the following definitions:

Table 15. RIWAR Field Descriptions and Settings

Bits	Name	Description and Setting
0	EN	Window address translation enable 1 Address translation enabled
8–11	TGINT	Target interface 1111 Incoming transactions are re-directed to the local memory
12–15	RDTYP	Read transaction type. Transaction type to run if access is a read 0100 Do not snoop local processor
16–19	WRTYP	Write transaction type. Transaction type to run if access is a write. 0000 Reserved (effectively makes this window read only)
26–31	SIZE	Inbound window size. 0x15 4 Mbyte window

3.4.1.3 RIWTAR

The RapidIO inbound window translation address register (RIWTAR) contains the translation address for the incoming transactions. The setting of the RIWTAR is different for the two scenarios.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—												TRAD			
Setting	0x000												0xF			

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	TRAD															
Setting	0xFC00															

Figure 29. RIWTAR Settings on Host for Incoming Boot Reads (Point-to-Point)

In the point-to-point system, the 4 Mbyte image required to boot the agent is at local address 0xFFC0_0000–0xFFFF_FFFF. Therefore the base address of the translation is 0xFFC00000.

The RIWTAR defines the translation on the address of the access:

Table 16. RIWTAR Field Description and Setting

Bits	Name	Description and Setting
12–31	TRAD	Translation address of inbound window. System address that represents the starting point of the inbound translated address. 0xF_FC00, translated window starts at 0xFFC0_0000

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—												TRAD			
Setting	0x000												0xF			

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	TRAD															
Setting	0xF800															

Figure 30. RIWTAR Settings on Host for Incoming Boot Reads (Fabric)

In the fabric example, the boot image for the agent is at local address 0xFF80_0000–0xFFBF_FFFF. Therefore, all transactions arriving at the incoming window should be shifted down the memory map by 4 Mbytes, and the base address of the translation is 0xFF80_0000.

The RIWTAR defines the translation on the address of the access:

TRAD = 0xF_F800, translated window starts at 0xFF80_0000

3.4.2 Configure the Agent Processor

This step configures and releases a single processor, to enable it to boot. In order for this step to be possible, the access to the internal memory map of the agent through the LCSBA1CSR, as described in Section 3.3, "Enable Access to Remote Configuration Space," must be enabled. This is due to the need to access registers in the agent processor that are not part of the RapidIO interface and are therefore not available through maintenance transactions.

3.4.2.1 Configure the Agent's RapidIO Outbound Window

The boot location configuration signal cfg_rom_loc, determines that all accesses to the default boot location are directed to the RapidIO interface. Before releasing the agent to boot, it is necessary to create an outbound RapidIO window to direct all those accesses to the host processor, at the correct RapidIO address.

In this particular example the default outbound RapidIO window is used for this purpose. The default outbound window does not have a base address, as it accepts all outgoing transactions which do not map to any other outbound window. It does have attribute and translation registers which must be initialized.

3.4.2.1.10 ROWTAR

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—		TRGTID								TREXAD		TRAD			
Setting	0x004												0x0			

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	TRAD															
Setting	0x0000															

Figure 31. ROWTAR on Agent for Boot Reads

The settings in the example have the following definitions:

Table 17. ROWTAR Field Descriptions and Settings

Bits	Name	Description
2–9	TRGTID	Target ID for RapidIO packet 0x01 Transactions have destination ID of 1 (host).
10–11	TREXAD	Translation extended address of outbound window 00 The 2 extended address bits are both zero
12–31	TRAD	Translation address of outbound window All zeros. There is no translation on the outbound transactions.

This results in the outbound transactions being directed to the host processor, with no address translation. For example, the first boot read comes out of the agent processor with a RapidIO address of 0x0_FFFF_FFFC.

3.4.2.1.11 ROWBAR

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	EN	—			TFLOV		PCI	—					RDTYP			
Setting	0x80								0x0				0x4			

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	WRTYP				—						SIZE					
Setting	0x4				0x0				0x1F							

The settings in the example have the following definitions:

Table 18. ROWBAR Field Descriptions and Settings

Bits	Name	Description and Setting
0	EN	Window address translation enable 1 This is default window, always enabled, read-only bit
4–5	TFLOV	Transaction flow level priority of transaction 00 Transactions have low priority.
6	PCI	Follow PCI transaction ordering rules 0 PCI transaction ordering is not followed.
12–15	RDTYP	Read transaction type. Transaction type to run on RapidIO interface if access is a read. 0100 Reads within this window generate NREAD transactions.
16–19	WRTYP	Write transaction type. Transaction type to run on RapidIO interface if access is a write. 0100 Writes within this window generate NWRITE transactions
26–31	SIZE	Window size 0x17 This window is 16 Mbytes

3.4.2.2 Create LAW on Agent for the Boot Area on RapidIO

This step may or may not be required, depending on the operation of the bootloader program which is being executed over RapidIO.

The mapping of the local area window (LAW) has priority over the default boot location set up by cfg_rom_loc. Therefore, if the agent's bootloader program sets up a LAW which covers the same area as the address range used for booting, then booting over RapidIO fails after the instructions to create that LAW have been executed.

Assuming that the bootloader does not use the highest priority LAW (LAW0), then this LAW should be used to direct the appropriate area of memory to the RapidIO interface. If the bootloader uses LAW0 for its own memory setup it has to be amended to permit booting over RapidIO. If the bootloader does not create a LAW which covers the same area of the local map as the boot instructions, then the setting up of this LAW is not required; the cfg_boot_loc is enough to re-direct the boot operations to the correct area.

In this example, the local area window is set up to cover the top 16 Mbytes of memory (0xFF00_0000–0xFFFF_FFFF).

3.4.2.2.12 LAWBAR

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—												BASE_ADDR			
Setting	0x000												0xF			

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	BASE_ADDR															
Setting	0xF000															

Figure 32. Agent LAWBAR Setting for RapidIO Window

The local area window base address (LAWBAR) sets the base address of this window. The settings in the example have the following definitions:

BASE_ADDR = 0xFF000 indicates that the start address of this window is 0xFF00_0000.

3.4.2.2.13 LAWAR

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	EN	—							TRGT_IF				—			
Setting	0x80								0xC				0x0			

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—										SIZE					
Setting	0x00								0x17							

Figure 33. Agent LAWAR Settings for RapidIO Window

The local area window attributes register (LAWAR) enables this window, sets the interface that the transactions are directed to and sets the size of the window. The settings in the example have the following definitions:

Table 19. LAWAR Field Descriptions and Settings

Bits	Name	Description and Setting
0	EN	Window address translation enable 1 This window is enabled.
8–11	TRGT_IF	Identifies the target interface ID when a transaction hits in the address range defined by this window. 1100 Target interface is RapidIO
26–31	SIZE	Window size 0x17 Window is 16 Mbytes.

3.4.2.3 Provide Agent Processor with Access to the RapidIO Bus

The processors which are configured as RapidIO agents are not enabled to issue RapidIO requests into the system. An agent's PGCCSR[ME],master enable bit, must be set by the host before the booting process is initiated. This must be accessed through the memory mapped LCSBA1CSR mechanism described in Section 3.3, "Enable Access to Remote Configuration Space."

3.4.2.4 Enable the CPU

The processor was configured in boot holdoff mode which means the agent's CPU has been prevented from booting. To initiate the booting process the host processor must set the agent's EEBPCR[CPU_EN],CPU port enable bit, to permit the CPU to start executing its boot sequence. This must be accessed through the memory mapped LCSBA1CSR mechanism described in Section 3.3, "Enable Access to Remote Configuration Space."

Once this CPU port enable bit has been set, the agent should boot from the host's Flash memory.

3.5 Enable Memory Reads and Writes

Now that the processors have been booted, many simple tests can be run to check functionality and benchmark RapidIO performance. This note describes how memory reads and writes can be executed between the host processor and processor 2, information can be extrapolated for other tests.

Before the host can access the agent's memory, more ATMU windows have to be initialized. A RapidIO outbound window must be set up on the host to direct normal reads and writes to the agent processor, and a RapidIO inbound window must be set up on the agent to capture these accesses and direct them to the appropriate area of the agent's memory.

This example sets up a 16 Mbyte outbound window on the host and a 16 Mbyte inbound window on agent 2. These windows direct accesses to the host's memory (in the range 0xC100_0000–0xC1FF_FFFF) to the agent's memory (in the range 0x0100_0000–0x01FF_FFFF). This area of agent memory was chosen to protect the lowest 16 Mbytes of agent memory, which contain interrupt vectors and so forth.

To achieve this mapping, the registers should be set up as shown next.

3.5.1 Host Setup

A RapidIO outbound window is set up on the host to direct local accesses in the range 0xC100_0000–0xC1FF_FFFF to processor 2 with RapidIO address 0x0_0000_0000–0x0_00FF_FFFF.

3.5.1.1 ROWBAR

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—												BADD			
Setting	0x000												0xC			

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	BADD															
Setting	0x1000															

Figure 34. Remote Memory ROWBAR Settings

The base address register (ROWBAR), defines the start address of the window. The settings in the example have the following definitions:

Table 20. ROWBAR Field Description and Setting

Bits	Name	Description and Setting
12–31	BADD	Base address of outbound window. Source address that is the starting point for the outbound translation window. 0xC1000 Window starts at address 0xC100_0000

3.5.1.2 ROWAR

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	EN	—			TFLOV		PCI	—					RDTYP			
Setting	0x80								0x0				0x4			

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	WRTYP				—						SIZE					
Setting	0x4				0x0				0x17							

Figure 35. Remote Memory Window ROWAR Settings

The attributes register defines attributes of the transactions which are created by this window. This includes the priority and transaction types. It also defines the size of the window. The settings in the example have the following definitions:

Table 21. ROWAR Field Descriptions and Settings

Bits	Name	Description and Setting
0	EN	Window address translation enable 1 This RapidIO outbound window is enabled.
4–5	TFLOV	Transaction flow level priority of transaction 00 The transactions are low priority.
6	PCI	Follow PCI transaction ordering rules 0 PCI transaction ordering is not followed.
12–15	RDTYP	Read transaction type. Transaction type to run on RapidIO interface if access is a read. 0100 Reads to this window generate RapidIO NREAD transactions.
16–19	WRTYP	Write transaction type. Transaction type to run on RapidIO interface if access is a write. 0100 writes to this window generates RapidIO NWRITE transactions
26–31	SIZE	Outbound window size. 0x17 This window is 16 Mbytes

3.5.1.3 ROWTAR

<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td></tr>
<tr><td>Field</td><td colspan="2">—</td><td colspan="8">TRGTID</td><td colspan="2">TREXAD</td><td colspan="4">TRAD</td></tr>
<tr><td>Setting</td><td colspan="12">0x008</td><td colspan="4">0x0</td></tr>
<tr><td></td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td>Field</td><td colspan="16">TRAD</td></tr>
<tr><td>Setting</td><td colspan="16">0x0000</td></tr>
</table>

Figure 36. Remote Memory ROWTAR Settings

The settings in the example have the following definitions:

Table 22. ROWTAR Field Descriptions and Settings

Bits	Name	Description and Setting
2–9	TRGTID	Target ID for RapidIO packet 0x02 Transactions have destination ID of 2.
10–11	TREXAD	Translation extended address of outbound window 00 The 2 extended address bits are both zero
12–31	TRAD	Translation address of outbound window All zeros. Outbound transactions start at address 0x0_0000_0000

3.5.2 Agent Setup

A 16 Mbyte RapidIO inbound window is set up on the agent to capture RapidIO transactions with the address range 0x0_0000_0000–0x000FF_FFFF and redirect them to the agent’s local memory space in the address range 0x0100_0000–0x01FF_FFFF.

3.5.2.1 RIWBAR

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—										BEXAD		BADD			
Setting	0x00								0x0				0x0			

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	BADD															
Setting	0x0000															

Figure 37. RIWBAR Settings on Host for Incoming Boot Reads

The settings in the example have the following definitions:

Table 23. RIWBAR Field Descriptions and Settings

Bits	Name	Description and Setting
10–11	BEXAD	Base extended address of inbound window. 00 Extended bits in the base address are both 0
12–31	BADD	Base address of inbound window. Source address that is the starting point for the inbound translation window. All zeros

The combined meaning of these two fields is that the base address of the inbound RapidIO window is 0x0_0000_0000.

3.5.2.2 RIWAR

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	EN	—							TGINT				RDTYP			
Setting	0x8				0x0				0xF				0x5			

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	WRTYP				—						SIZE					
Setting	0x5				0x0				0x17							

Figure 38. RIWAR Settings on Host for Incoming Boot Reads

The settings in the example have the following definitions:

Table 24. RIWAR Descriptions and Settings

Bits	Name	Description and Setting
0	EN	Window address translation enable 1 Address translation enabled
8–11	TGINT	Target interface 1111 Incoming transactions are re-directed to the local memory
12–15	RDTYP	Read transaction type. Transaction type to run if access is a read. 0101 Snoop local processor
16–19	WRTYP	Write transaction type. Transaction type to run if access is a write. 0101 Received write. Snoop local processor
26–31	SIZE	Inbound window size. Inbound window size *N* which is the encoded 2^{n+1} bytes window size. 01_0111 16 Mbyte window

3.5.2.3 RIWTAR

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—												TRAD			
Setting	0x000												0x0			

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	TRAD															
Setting	0x1000															

Figure 39. RIWTAR Settings on Host for Incoming Boot Reads (Point-to-Point)

The RIWTAR defines the translation on the address of the access:

Table 25. RIWTAR Description and Setting

Bits	Name	Description and Setting
12–31	TRAD	Translation address of inbound window. System address that represents the starting point of the inbound translated address. 0x01000 Translated window starts at 0x0100_0000

3.5.3 Confirming Operation

After the agent and host are configured, tests can be run to verify the ability to read/write memory between the two processors. These tests can take various forms.

For a simple test, requiring transactions to have 32 bits of payload or less, simple reads and writes to the 16 Mbyte address range 0xC100_0000–0xC1FF_FFFF on the host result in RapidIO transactions to processor 2. When processor 2 receives those requests they are translated to the 16 Mbyte address range 0x0100_0000–0x01FF_FFFF. To confirm correct operation, these memory locations can be examined directly on processor 2, or read back and compared with the value written.

If the test requires larger transactions, the DMA engine can be used. The same address mapping applies, that is, if the DMA source/destination is set up as 0xC100_0000, this generates RapidIO read/writes to processor 2. Using the DMA can generate RapidIO reads and writes with payload greater than 32 bits. Therefore, this provides a more efficient way of passing large amounts of data between devices and is useful for benchmarking.

4 Output from Example Application

This section contains example text output from an application to bring up basic RapidIO systems, using the procedure described previously. Section 4.1, "Point-to-Point Example," contains a sample output, obtained when the application was executed on the point-to-point system described in Section 2.3.1, "Point-to-Point System." Section 4.2, "Fabric Example," contains a sample output, obtained when the application was executed on the fabric based system described in Section 2.3.2, "Fabric-Based System."

4.1 Point-to-Point Example

```
## Starting application at 0x00040004 ...

Check LAWs before setting up

lawbar0 = 0x00000000, lawar0 = 0x00000000

lawbar1 = 0x00000000, lawar1 = 0x80f0001b

lawbar2 = 0x000fe000, lawar2 = 0x80400018

lawbar3 = 0x000fd000, lawar3 = 0x8040000c

lawbar4 = 0x00000000, lawar4 = 0x00000000

lawbar5 = 0x00000000, lawar5 = 0x00000000

lawbar6 = 0x00000000, lawar6 = 0x00000000

lawbar7 = 0x00000000, lawar7 = 0x00000000

Completed the setup of TLBs and lawbars

lawbar5 = 0x000c0000, lawar5 = 0x80c0001b

This device is configured as a RIO host

This device has RIO device ID 0x00000001

RapidIO port is trained OK

Set up the outbound rio maintenance window

rowbar1 = 0x000c0000, rowar1 = 0x80077015, rowtar1 = 0x3fc00000

 ---- rio_local_config() completed successfully ----
```

```
Identified adjacent device as MPC8540/60
Executing the point-to-point discovery process
Original HBDIDL =0x0000ffff
Updated HBDIDL =0x00000001
Original deviceID = 0x000000ff
Updated deviceID  = 0x00000002

 ---- rio_discovery() completed successfully ----

Created a local ROW for CCSR access to proc 2
rowbar5 = 0x000c4000, rowar5 = 0x80044013, rowtar5 = 0x00801000
Read back data from device 2 through lcsbacsr
didcar= 0x00020002
lcsbacsr = 0x00200000

 ---- set up remote LCS access to all trained processors ----

Set up inbound window on host to re-direct incoming rio to 'safe' area
riwbar1 = 0x00000000, riwar1 = 0x80f55017, riwtar1 = 0x00001000
 ---- Provided LCSBASCR and memory access back to local processor ----

Options :
b = boot another processor
h = print this list of options
q = quit this application
r = read rioport information
t = run one of the tests
Attempting to boot processor 2
Booting processor 2
Before booting :
ccsrbar = 0x000ff700
eebpcr = 0x00000000
agent LAWBAR0 = 0x000ff000
agent LAWAR0  = 0x80c00017
agent LAWBAR1 = 0x00000000
agent LAWAR1  = 0x00000000
agent LAWBAR2 = 0x00000000
agent LAWAR2  = 0x00000000
agent LAWBAR3 = 0x00000000
agent LAWAR3  = 0x00000000
agent LAWBAR4 = 0x00000000
agent LAWAR4  = 0x00000000
agent LAWBAR5 = 0x00000000
agent LAWAR5  = 0x00000000
agent LAWBAR6 = 0x00000000
agent LAWAR6  = 0x00000000
agent LAWBAR7 = 0x00000000
agent LAWAR7  = 0x00000000
Press any key to continue
AFTER BOOT Read back data from device 2  through lcsbacsr
didcar= 0x00020002
lcsbacsr = 0x00200000
bdidcsr= 0x00020000
ccsrbar = 0x000e0000
eebpcr = 0x01000000
agent LAWBAR0 = 0x000ff000
agent LAWAR0  = 0x80c00017
agent LAWBAR1 = 0x00000000
agent LAWAR1  = 0x80f0001a
agent LAWBAR2 = 0x000fe000
agent LAWAR2  = 0x8040000c
agent LAWBAR3 = 0x000fd000
agent LAWAR3  = 0x8040000c
agent LAWBAR4 = 0x00000000
agent LAWAR4  = 0x00000000
agent LAWBAR5 = 0x00000000
agent LAWAR5  = 0x00000000
agent LAWBAR6 = 0x00000000
agent LAWAR6  = 0x00000000
agent LAWBAR7 = 0x00000000
agent LAWAR7  = 0x00000000
Now setup window to access 16MBytes memory of device 2
rowbar2=0x000c1000, rowtar2=0x00800000, rowar2=0x80044017
Set up IB window on agent to permit host to access its memory
Agent riwbar1=0x00000000, riwtar1=0x00001000, riwar1=0x80f55017
>T. Which test do you want to run ?
A = execute 100 32bit writes to agent, read back and compare
B = DMA transfer 64kbytes block to agent, read back and compare
Running test A on processor 2
Running test A for deviceid 2
Executing 100 32-bit writes into agent memory
Executing 100 32-bit reads, and comparing
Test completed successfully
>T. Which test do you want to run ?
A = execute 100 32bit writes to agent, read back and compare
B = DMA transfer 64kbytes block to agent, read back and compare
Running test B on processor 2
Running test B for deviceid 2
About to DMA transfer 65536 bytes to the agent
DMA transfer of 65536 bytes to the agent has completed
DMA transfer back from host has completed
Comparing data sent with data received
Test completed successfully
>R. Read RIO information for which device number  1 or 2 ? 1
Rio port information from device 1
didcar   = 0x00020002       bdidcsr = 0x00010000
lcsbacsr = 0x00200000       pescsr  = 0x0002000a
plascsr  = 0x07000003       pgccsr  = 0xe0000000
predr    = 0x00000000       pnfedr  = 0x00000000
row0 :                 tar=0x00000000, ar=0x8004401f
row1 : bar=0x000c0000, tar=0x008ff000, ar=0x80077015
row2 : bar=0x000c1000, tar=0x00800000, ar=0x80044017
row3 : bar=0x00000000, tar=0x00000000, ar=0x0004401f
row4 : bar=0x00000000, tar=0x00000000, ar=0x0004401f
row5 : bar=0x000c4000, tar=0x00801000, ar=0x80044013
```

```
row6 : bar=0x00000000, tar=0x00000000, ar=0x0004401f

row7 : bar=0x00000000, tar=0x00000000, ar=0x0004401f

row8 : bar=0x00000000, tar=0x00000000, ar=0x0004401f

riw0 :                 tar=0x00000000, ar=0x8004401f

riw1 : bar=0x00000000, tar=0x00001000, ar=0x80f55017

riw2 : bar=0x00000000, tar=0x00000000, ar=0x0004401f

riw3 : bar=0x00000000, tar=0x00000000, ar=0x0004401f

riw4 : bar=0x000ffc00, tar=0x000ffc00, ar=0x80f50015

>R. Read RIO information for which device number 1 or 2 ? 2

Rio port information from device 2

didcar   = 0x00020002       bdidcsr = 0x00020000

lcsbacsr = 0x00200000       pescsr  = 0x0002020a

plascsr  = 0x01000003       pgccsr  = 0x40000000

predr    = 0x00000000       pnfedr  = 0x00000000

row0 :                 tar=0x00400000, ar=0x8004401f

row1 : bar=0x00000000, tar=0x00000000, ar=0x0004401f

row2 : bar=0x00000000, tar=0x00000000, ar=0x0004401f

row3 : bar=0x00000000, tar=0x00000000, ar=0x0004401f

row4 : bar=0x00000000, tar=0x00000000, ar=0x0004401f

row5 : bar=0x00000000, tar=0x00000000, ar=0x0004401f

row6 : bar=0x00000000, tar=0x00000000, ar=0x0004401f

row7 : bar=0x00000000, tar=0x00000000, ar=0x0004401f

row8 : bar=0x00000000, tar=0x00000000, ar=0x0004401f

riw0 :                 tar=0x00000000, ar=0x8004401f

riw1 : bar=0x00000000, tar=0x00001000, ar=0x80f55017

riw2 : bar=0x00000000, tar=0x00000000, ar=0x0004401f

riw3 : bar=0x00000000, tar=0x00000000, ar=0x0004401f

riw4 : bar=0x00000000, tar=0x00000000, ar=0x0004401f
```

4.2 Fabric Example

```
## Starting application at 0x00040004 ...

Check LAWs before setting up

lawbar0 = 0x00000000, lawar0 = 0x00000000

lawbar1 = 0x00000000, lawar1 = 0x80f0001b

lawbar2 = 0x00080000, lawar2 = 0x8000001c
```

```
lawbar3 = 0x000ff000, lawar3 = 0x80400017
lawbar4 = 0x000e2000, lawar4 = 0x80000017
lawbar5 = 0x00000000, lawar5 = 0x00000000
lawbar6 = 0x00000000, lawar6 = 0x00000000
lawbar7 = 0x00000000, lawar7 = 0x00000000
Completed the setup of TLBs and lawbars
lawbar5 = 0x000c0000, lawar5 = 0x80c0001b
This device is configured as a RIO host
This device has RIO device ID 0x00000001
RapidIO port is trained OK
Set up the outbound rio maintenance window
rowbar1 = 0x000c0000, rowar1 = 0x80077015, rowtar1 = 0x3fc00000
 ---- rio_local_config() completed successfully ----

Identified adjacent device as Tsi500
Executing the fabric version of the discovery code
swport = 0x00000400
Original Tsi500 HBDIDL =0x0000ffff
Updated HBDIDL on Tsi500  =0x00000001

Examining port 1 of the TSI500
Port 1 training reg = 0x00000800,  This port is not trained

Examining port 2 of the TSI500
Port 2 training reg = 0xff800800,  Trained successfully
Changed the Port 0 Tsi500_DESTID_248_LKUP to 0x22222222
Device attached to port 2, DIDCAR=0x00030002
Original HBDIDL =0x0000ffff
Updated HBDIDL =0x00000001
Original deviceID = 0x000000ff
Updated deviceID  = 0x00000003

Examining port 3 of the TSI500
Port 3 training reg = 0xff800800,  Trained successfully
Changed the Port 0 TSI500_DESTID_248_LKUP to 0x33333333
```

```
Device attached to port 3, DIDCAR=0x00030002
Original HBDIDL =0x0000ffff
Updated HBDIDL =0x00000001
Original deviceID = 0x000000ff
Updated deviceID  = 0x00000004

 ---- rio_discovery() completed successfully ----

Created a local ROW for CCSR access to proc 3
rowbar6 = 0x000c5000, rowar6 = 0x80044013, rowtar6 = 0x00c01000
Read back data from device 3 through lcsbacsr
didcar= 0x00030002
lcsbacsr = 0x00200000

Created an local ROW for CCSR access to proc 4
rowbar7 = 0x000c6000, rowar7 = 0x80044013, rowtar7 = 0x01001000
Read back data from device 4 through lcsbacsr
didcar= 0x00030002
lcsbacsr = 0x00200000

 ---- set up remote LCS access to all trained processors ----

Set up inbound window on host to re-direct incoming rio to 'safe' area
riwbar1 = 0x00000000, riwar1 = 0x80f55017, riwtar1 = 0x00001000
 ---- Provided LCSBASCR and memory access back to local processor ----

Options :
b = boot another processor
h = print this list of options
q = quit this application
s = read back info from tsi500
r = read rioport information
t = run one of the tests
B. Boot processor 2, 3, or 4 ?  3
Booting processor 3
```

```
Before booting :

ccsrbar = 0x000ff700

eebpcr = 0x00000000

agent LAWBAR0 = 0x000ff000

agent LAWAR0   = 0x80c00017

agent LAWBAR1 = 0x00000000

agent LAWAR1   = 0x00000000

agent LAWBAR2 = 0x00000000

agent LAWAR2   = 0x00000000

agent LAWBAR3 = 0x00000000

agent LAWAR3   = 0x00000000

agent LAWBAR4 = 0x00000000

agent LAWAR4   = 0x00000000

agent LAWBAR5 = 0x00000000

agent LAWAR5   = 0x00000000

agent LAWBAR6 = 0x00000000

agent LAWAR6   = 0x00000000

agent LAWBAR7 = 0x00000000

agent LAWAR7   = 0x00000000

Press any key to continue

AFTER BOOT Read back data from device 3  through lcsbacsr

didcar= 0x00030002

lcsbacsr = 0x00200000

bdidcsr= 0x00030000

ccsrbar = 0x000e0000

eebpcr = 0x01000000

agent LAWBAR0 = 0x000ff000

agent LAWAR0   = 0x80c00017

agent LAWBAR1 = 0x00000000

agent LAWAR1   = 0x80f0001c

agent LAWBAR2 = 0x00080000

agent LAWAR2   = 0x80400017

agent LAWBAR3 = 0x000ff000

agent LAWAR3   = 0x80400017

agent LAWBAR4 = 0x000e2000
agent LAWAR4  = 0x80000017
agent LAWBAR5 = 0x000c0000
agent LAWAR5  = 0x80c0001c
agent LAWBAR6 = 0x00000000
agent LAWAR6  = 0x00000000
agent LAWBAR7 = 0x00000000
agent LAWAR7  = 0x00000000
Now setup window to access 16MBytes memory of device 3
rowbar3=0x000c2000, rowtar3=0x00c00000, rowar3=0x80044017
Set up IB window on agent to permit host to access its memory
Agent riwbar1=0x00000000, riwtar1=0x00001000, riwar1=0x80f55017
>B. Boot processor 2, 3, or 4 ?  4
Booting processor 4
Before booting :
ccsrbar = 0x000ff700
eebpcr = 0x00000000
agent LAWBAR0 = 0x000ff000
agent LAWAR0  = 0x80c00017
agent LAWBAR1 = 0x00000000
agent LAWAR1  = 0x00000000
agent LAWBAR2 = 0x00000000
agent LAWAR2  = 0x00000000
agent LAWBAR3 = 0x00000000
agent LAWAR3  = 0x00000000
agent LAWBAR4 = 0x00000000
agent LAWAR4  = 0x00000000
agent LAWBAR5 = 0x00000000
agent LAWAR5  = 0x00000000
agent LAWBAR6 = 0x00000000
agent LAWAR6  = 0x00000000
agent LAWBAR7 = 0x00000000
agent LAWAR7  = 0x00000000
Press any key to continue
AFTER BOOT Read back data from device 4  through lcsbacsr
didcar= 0x00030002
```

```
lcsbacsr = 0x00200000
bdidcsr= 0x00040000
ccsrbar = 0x000e0000
eebpcr = 0x01000000
agent LAWBAR0 = 0x000ff000
agent LAWAR0  = 0x80c00017
agent LAWBAR1 = 0x00000000
agent LAWAR1  = 0x80f0001c
agent LAWBAR2 = 0x00080000
agent LAWAR2  = 0x80400017
agent LAWBAR3 = 0x000ff000
agent LAWAR3  = 0x80400017
agent LAWBAR4 = 0x000e2000
agent LAWAR4  = 0x80000017
agent LAWBAR5 = 0x000c0000
agent LAWAR5  = 0x80c0001c
agent LAWBAR6 = 0x00000000
agent LAWAR6  = 0x00000000
agent LAWBAR7 = 0x00000000
agent LAWAR7  = 0x00000000
Now setup window to access 16MBytes memory of device 4
rowbar4=0x000c3000, rowtar4=0x01000000, rowar4=0x80044017
Set up IB window on agent to permit host to access its memory
Agent riwbar1=0x00000000, riwtar1=0x00001000, riwar1=0x80f55017
>T. Which test do you want to run ?
A = execute 100 32bit writes to agent, read back and compare
B = DMA transfer 64kbytes block to agent, read back and compare
A. On which device do you want to run test A : 2, 3, or 4 ?3
Running test A for deviceid 3
Executing 100 32-bit writes into agent memory
Executing 100 32-bit reads, and comparing
Test completed successfully
>T. Which test do you want to run ?
A = execute 100 32bit writes to agent, read back and compare
B = DMA transfer 64kbytes block to agent, read back and compare
```

```
B. On which device do you want to run test B: 2, 3, or 4 ?3

Running test B for deviceid 3

About to DMA transfer 65536 bytes to the agent

DMA transfer of 65536 bytes to the agent has completed

DMA transfer back from host has completed

Comparing data sent with data received

Test completed successfully

>T. Which test do you want to run ?

A = execute 100 32bit writes to agent, read back and compare

B = DMA transfer 64kbytes block to agent, read back and compare

A. On which device do you want to run test A : 2, 3, or 4 ?4

Running test A for deviceid 4

Executing 100 32-bit writes into agent memory

Executing 100 32-bit reads, and comparing

Test completed successfully

>T. Which test do you want to run ?

A = execute 100 32bit writes to agent, read back and compare

B = DMA transfer 64kbytes block to agent, read back and compare

B. On which device do you want to run test B: 2, 3, or 4 ?4

Running test B for deviceid 4

About to DMA transfer 65536 bytes to the agent

DMA transfer of 65536 bytes to the agent has completed

DMA transfer back from host has completed

Comparing data sent with data received

Test completed successfully

>R. Read RIO information for which device number  1,2,3 or 4 ? 1

Rio port information from device 1

didcar   = 0x00030002       bdidcsr = 0x00010000

lcsbacsr = 0x00200000       pescsr  = 0x0002000a

plascsr  = 0x02000002       pgccsr  = 0xe0000000

predr    = 0x00000000       pnfedr  = 0x00000000

row0 :                 tar=0x00000000, ar=0x8004401f

row1 : bar=0x000c0000, tar=0x010ff000, ar=0x80077015

row2 : bar=0x00000000, tar=0x00000000, ar=0x0004401f

row3 : bar=0x000c2000, tar=0x00c00000, ar=0x80044017
```

```
row4 : bar=0x000c3000, tar=0x01000000, ar=0x80044017
row5 : bar=0x00000000, tar=0x00000000, ar=0x0004401f
row6 : bar=0x000c5000, tar=0x00c01000, ar=0x80044013
row7 : bar=0x000c6000, tar=0x01001000, ar=0x80044013
row8 : bar=0x00000000, tar=0x00000000, ar=0x0004401f
riw0 :                 tar=0x00000000, ar=0x8004401f
riw1 : bar=0x00000000, tar=0x00001000, ar=0x80f55017
riw2 : bar=0x00000000, tar=0x00000000, ar=0x0004401f
riw3 : bar=0x00000000, tar=0x00000000, ar=0x0004401f
riw4 : bar=0x000ffc00, tar=0x000ff800, ar=0x80f50015
>R. Read RIO information for which device number  1,2,3 or 4 ? 3
Rio port information from device 3
didcar   = 0x00030002       bdidcsr = 0x00030000
lcsbacsr = 0x00200000       pescsr  = 0x0002000a
plascsr  = 0x01000003       pgccsr  = 0x40000000
predr    = 0x00000000       pnfedr  = 0x00000000
row0 :                 tar=0x00400000, ar=0x8004401f
row1 : bar=0x00000000, tar=0x00000000, ar=0x0004401f
row2 : bar=0x00000000, tar=0x00000000, ar=0x0004401f
row3 : bar=0x00000000, tar=0x00000000, ar=0x0004401f
row4 : bar=0x00000000, tar=0x00000000, ar=0x0004401f
row5 : bar=0x00000000, tar=0x00000000, ar=0x0004401f
row6 : bar=0x00000000, tar=0x00000000, ar=0x0004401f
row7 : bar=0x00000000, tar=0x00000000, ar=0x0004401f
row8 : bar=0x00000000, tar=0x00000000, ar=0x0004401f
riw0 :                 tar=0x00000000, ar=0x8004401f
riw1 : bar=0x00000000, tar=0x00001000, ar=0x80f55017
riw2 : bar=0x00000000, tar=0x00000000, ar=0x0004401f
riw3 : bar=0x00000000, tar=0x00000000, ar=0x0004401f
riw4 : bar=0x00000000, tar=0x00000000, ar=0x0004401f
>R. Read RIO information for which device number  1,2,3 or 4 ? 4
Rio port information from device 4
didcar   = 0x00030002       bdidcsr = 0x00040000
lcsbacsr = 0x00200000       pescsr  = 0x0002000a
plascsr  = 0x05000007       pgccsr  = 0x40000000
```

```
predr    = 0x00000000       pnfedr   = 0x00000000
row0 :                 tar=0x00400000, ar=0x8004401f
row1 : bar=0x00000000, tar=0x00000000, ar=0x0004401f
row2 : bar=0x00000000, tar=0x00000000, ar=0x0004401f
row3 : bar=0x00000000, tar=0x00000000, ar=0x0004401f
row4 : bar=0x00000000, tar=0x00000000, ar=0x0004401f
row5 : bar=0x00000000, tar=0x00000000, ar=0x0004401f
row6 : bar=0x00000000, tar=0x00000000, ar=0x0004401f
row7 : bar=0x00000000, tar=0x00000000, ar=0x0004401f
row8 : bar=0x00000000, tar=0x00000000, ar=0x0004401f
riw0 :                 tar=0x00000000, ar=0x8004401f
riw1 : bar=0x00000000, tar=0x00001000, ar=0x80f55017
riw2 : bar=0x00000000, tar=0x00000000, ar=0x0004401f
riw3 : bar=0x00000000, tar=0x00000000, ar=0x0004401f
riw4 : bar=0x00000000, tar=0x00000000, ar=0x0004401f
>S. Print Switch info
                       Port0      Port1      Port2    Port3
Training registers : 0xff800800 0x00000800 0xff800800 0xff800800
Routing 0 registers: 0xfff3210f 0xfff3210f 0xfff3210f 0xfff3210f
IB Packet count    : 0x007c11f3 0x00000000 0x003e0857 0x003e06ab
IB Read pkt cnt 0  : 0x000003a4 0x00000000 0x003e068d 0x003e04e1
IB Read pkt cnt 1  : 0x00000000 0x00000000 0x00000005 0x00000005
IB Read pkt cnt 2  : 0x00000000 0x00000000 0x00000000 0x00000000
IB Read pkt cnt 3  : 0x00000000 0x00000000 0x00000000 0x00000000
IB Write pkt cnt 0 : 0x000002e5 0x00000000 0x00000000 0x00000000
IB Write pkt cnt 1 : 0x00000000 0x00000000 0x00000003 0x00000003
IB Write pkt cnt 2 : 0x00000000 0x00000000 0x00000000 0x00000000
IB Write pkt cnt 3 : 0x00000000 0x00000000 0x00000000 0x00000000
Non recov err      : 0x00000000 0x00000000 0x00000000 0x00000000
Recoverable err    : 0x00000000 0x00000000 0x00000000 0x00000000
Inbound err stat 3 : 0x00000000 0x00000000 0x00000000 0x00000000
Outbound err stat 1: 0x00000000 0x00000000 0x00000000 0x00000000
```

5 References

1. RapidIO Trade Association, *RapidIO Interconnect Specification*, Rev. 1.2, 6/2002.
2. *MPC8540 PowerQUICC III Integrated Host Processor Reference Manual*, Rev. 1, 7/2004.
   or *MPC8560 PowerQUICC III Integrated Communications Processor Reference Manual*, Rev. 1, 7/2004.
3. *Tsi500 Parallel RapidIO Multi-port Switch User Manual*, April 2004.
4. *RapidIO Interconnect Specification Annex I: Software/System Bring Up Specification*, Rev. 1.0, 12/2003.
5. Freescale application note AN2741, *Using the RapidIO Messaging Unit on PowerQUICC III*, Rev. 1, 8/2004.

6 Revision History

Table 26 provides a revision history for this application note.

Table 26. Document Revision History

Revision	Date	Substantive Change(s)
1	11/15/2004	Initial release

Appendix A Notes on Maintenance Transactions

A.1 Overview

Throughout this document, all maintenance transactions are described as a list of maintenance parameters (for example, destination ID, hopcount, and maintenance offset), but no attempt is made to describe how these maintenance transactions are created. A full description of the operation of the maintenance window, and the generation of maintenance transactions is described in this appendix. In addition, it describes the operation of maintenance transactions in a multi-switch environment.

A.2 Terminology

In this Appendix, specific terminology is used to describe the different aspects of the maintenance transaction.

NOTE

This table contains clarifications of terms specific to this Appendix. It should not be assumed that this is commonly known or used terminology.

Table 27. Maintenance Transaction Terminology

Term	Description
Maintenance offset	This is the logical offset of a maintenance transaction. The maintenance offset closely resembles the way in which the RapidIO specification represents the offsets to the CAR/CSR block. Maintenance offsets are always 16-bit word aligned. For example: The RapidIO specification describes the source operations CAR as offset 0x18, word 0. This document would describe that as maintenance offset 0x18. The RapidIO specification describes the destination operations CAR as offset 0x18, word 1. This document would describe that as maintenance offset 0x1C. The maintenance offset is represented by 24-bits (although 2 LSBs are always 0). Therefore, the maximum maintenance offset is 0x7F_FFFC.
Configuration offset	This is a 21-bit field within the RapidIO maintenance packet. This is a double-word pointer and is equivalent to the maintenance offset shifted 3 bits to the right. For Example: For accesses to the source operations CAR (maintenance offset 0x18), the configuration offset field in the RapidIO maintenance transaction would be 0x03. For accesses to the destination operations CAR (maintenance offset 0x1C), the configuration offset field in the RapidIO maintenance transaction would also be 0x03.
Word pointer	This is a single bit field in the maintenance packet that defines which word to access within the double word indicated by the configuration offset. For Example: For accesses to the source operations CAR (maintenance offset 0x18), the word pointer in the RapidIO maintenance transaction would be 0x0. For accesses to the destination operations CAR (maintenance offset 0x1C), the word pointer in the RapidIO maintenance transaction would be 0x1.

Table 27. Maintenance Transaction Terminology (continued)

Term	Description
CFG_OFFSET	This is a field within the ROWTAR for maintenance transactions. A variable number of bits (depending on the size of the window) is taken from this field to build the configuration offset.
Transaction offset	This is the offset of the untranslated address from the base address of the window. For example: If the base address of the maintenance window is 0xC000_0000, then an access to address 0xC000_001C has a transaction offset of 0x1C.

A.3 Example 1: 4 Mbytes Maintenance Window

In this example (used in the main document) the maintenance window is set up to 4 Mbytes, which permits a transaction offset of 0x0–0x3FFFFF.

In a 4 Mbytes maintenance window, the bottom 22 bits of the maintenance offset are determined from the transaction offset, and only the 2 MSBs of the CFG_OFFSET field are significant.

To allow for access to maintenance offsets greater than 0x3F_FFFC, the MSBs of the offset must be masked off from the calculation of the transaction address, and placed correctly within the CFG_OFFSET field.

Maintenance transactions within a 4 Mbytes window can be achieved using the following macros.

```
/******************************************************************************

MACRO definitions

******************************************************************************/

#define MAINT_READ_4M(device_id, hopcount, offset , ptrvalue) \

  rioport->rowtar1 = ((device_id) << ROW_DEV_ID_Shift) \

  |((hopcount) << ROW_HOPCNT_Shift) | ((offset) >> 12); \

  asm ("sync");\

  ptrvalue = *((volatile uint32*)(START_OF_RIO_WINDOW + ((offset)& 0x3FFFFF)));\

  asm ("sync");


#define MAINT_WRITE_4M(device_id, hopcount, offset , value) \

  rioport->rowtar1 = ((device_id) << ROW_DEV_ID_Shift) \

  |((hopcount) << ROW_HOPCNT_Shift) | ((offset) >> 12); \

  asm ("sync");\

  *((volatile uint32 *) (START_OF_RIO_WINDOW + ((offset) & 0x3FFFFF))) = value; \

  asm ("sync");
```

A maintenance read to destination ID 0x02, hopcount 0xFF, and maintenance offset 0x68, could then be achieved using the macro as follows.

```
uint32 value_read;

MAINT_READ_4M (0x02, 0xFF, 0x68, value_read);
```

A maintenance write of the value 0xFFF3_210F to destination ID 0xFF, hopcount 0x00, maintenance offset 0x70A00 can be achieved using the macro as follows.

```
MAINT_WRITE_4M (0xFF, 0x00, 0x70A00, 0xFFF3210F);
```

A.4 Example 2: 4 Kbytes Maintenance Window

In this example the maintenance window is only 4 Kbytes, which permits a transaction offset of 0x0–0xFFF.

In a 4 Kbytes maintenance window, the bottom 12 bits of the maintenance offset are determined from the transaction offset, and all 12 bits of the CFG_OFFSET field are significant.

To allow for access to maintenance offsets greater than 0xFFC, the MSBs of the offset must be masked off from the calculation of the transaction offset, and placed correctly within the CFG_OFFSET field.

Maintenance transactions within a 4 Kbytes window can be achieved using the following macros.

```
/*****************************************************************************

MACRO definitions

*****************************************************************************/

#define MAINT_READ_4k(device_id, hopcount, offset , ptrvalue) \

  rioport->rowtar1 = ((device_id) << ROW_DEV_ID_Shift) \

  |((hopcount) << ROW_HOPCNT_Shift) | ((offset) >> 12); \

  asm ("sync");\

  ptrvalue = *((volatile uint32 *) (START_OF_RIO_WINDOW + ((offset)& 0xFFF)));\

  asm ("sync");


#define MAINT_WRITE_4k(device_id, hopcount, offset , value) \

  rioport->rowtar1 = ((device_id) << ROW_DEV_ID_Shift) \

  |((hopcount) << ROW_HOPCNT_Shift) | ((offset) >> 12); \

  asm ("sync");\

  *((volatile uint32 *) (START_OF_RIO_WINDOW + ((offset) & 0xFFF))) = value; \

  asm ("sync");
```

A maintenance read to destination ID 0x02, hopcount 0xFF, maintenance offset 0x68, could then be achieved using the macro as follows.

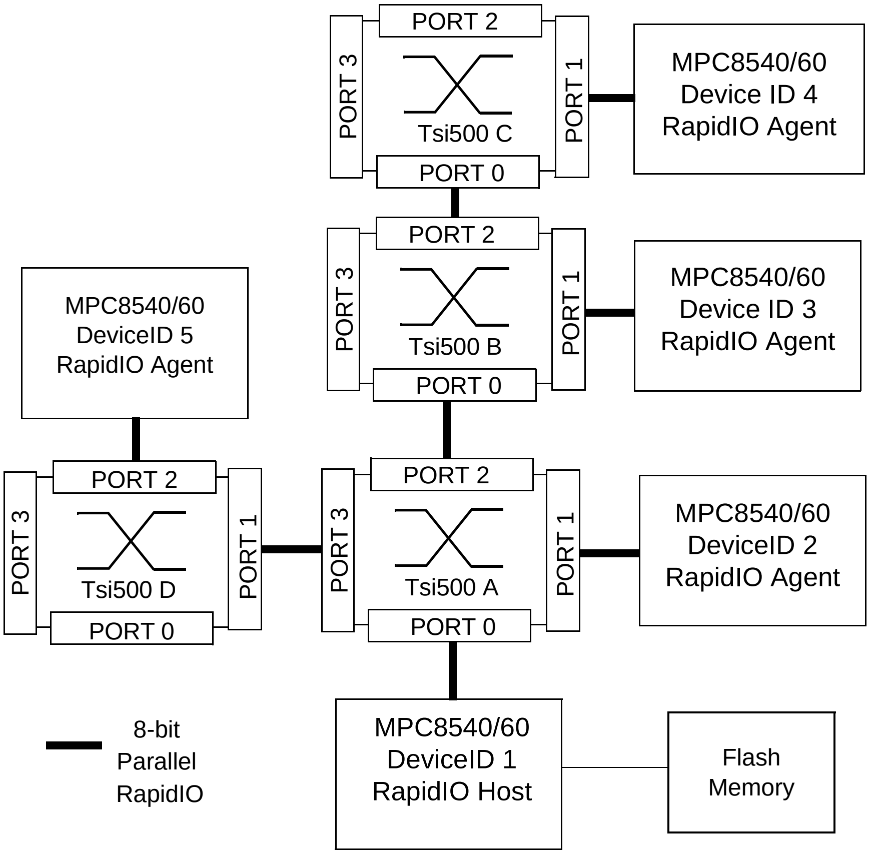
```
uint32 value_read;

MAINT_READ_4k (0x02, 0xFF, 0x68, value_read);
```

A maintenance write of the value 0xfff3210f to destination ID 0xFF, hopcount 0x00, maintenance offset 0x70a00 can be achieved using the macro as follows.

```
MAINT_WRITE_4k (0xFF, 0x00, 0x70A00, 0xFFF3210F);
```

A.5 Maintenance Transactions within Multi-Switch Systems

This document assumes that a system consists of no more that one RapidIO switch. For multi-switch systems, consideration must be given to the way in which the maintenance window can be used to direct accesses to the different switches. This section describes how the destination ID and hopcount are both used to determine which switch is accessed.



When the hopcount of a transaction is greater than 0, the switch uses the destination ID to determine which port the request is forwarded to, and then decrements the hopcount by one.

In the example above, assume that the routing tables on all switches are set up to direct transactions to the correct RapidIO endpoints. Table 28 shows the resulting switch for a given hopcount and destination ID.

Table 28. Maintenance Transactions and Resultant Accessed Switch

Hopcount	Destination ID	Switch Accessed
0x00	any	switch A
0x01	0x3 or 0x4	switch B
0x02	0x4	switch C
0x01	0x5	switch D

THIS PAGE INTENTIONALLY LEFT BLANK

How to Reach Us:

Home Page:
www.freescale.com

email:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
(800) 521-6274
480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Technical Information Center
3-20-1, Minami-Azabu, Minato-ku
Tokyo 106-0047 Japan
0120 191014
+81 3 3440 3569
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate,
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor
Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
(800) 441-2447
303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@
hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters which may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals” must be validated for each customer application by customer’s technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

AN2753
Rev. 1
11/2004

freescale™
semiconductor