Freescale Semiconductor
Application Note

AN2932
Rev. 0, 12/2005

Serial RapidIO Bring-Up Procedure on PowerQUICC™ III

by Lorraine McLuckie and Colin Cureton
NCSD Platforms, East Kilbride

The MPC8548 PowerQUICC™ III processor features a 1x/4x serial RapidIO interface. This document provides guidelines for basic use of this interface, starting with local and remote processors setup with basic verification and discovery. It then describes booting over RapidIO and conducting simple data transfer tests.

Most of this document applies to any PowerQUICC III system enabled for serial RapidIO. However, the concepts are illustrated with a few device-specific examples.

Contents

1 Introduction 2
2 RapidIO Basics 3
3 Bring-up Procedure 8
4 Output from Example Application 39
5 References 48

Appendix A:
Notes on Maintenance Transactions 49
A.1 Terminology 49
A.2 Example 1: 4 Mbyte Maintenance Window 50
A.3 Example 2: 4 Kbyte Maintenance Window 50
A.4 Maintenance Transactions Within Multi-Switch Systems 51

1 Introduction

This document assists engineers in using the serial RapidIO interface on the MPC8548 PowerQUICC III processor. After summarizing basic aspects of the RapidIO specification, it describes a procedure to bring up simple RapidIO systems, including setting up the local processor, simplified discovery, booting over RapidIO, and executing simple memory reads/writes to the remote processors.

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 access 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
SDRAM	Synchronous dynamic random access memory
SPR	Special-Purpose Register
SRIO	Serial RapidIO
TLB	Translation lookaside buffer

2 RapidIO Basics

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. This section summarizes information from the RapidIO specification. For details, consult the *RapidIO Interconnect Specification* [1].

2.1 RapidIO Layers

The RapidIO specification is written in layers, as follows, to ensure flexibility and modularity so that changes to one layer do not necessarily affect other layers:

- *Logical layer*. Defines the overall protocol and packet formats, the types of RapidIO transactions, and addressing. The logical layer is split into several categories depending on the system model:
    - Input/output logical specification that defines the basic system architecture of RapidIO.
    - Message passing logical specification that enables distributed I/O processing.
- *Transport layer*. Describes routing as packets move from one point to another.
- *Physical layer*. Defines the device-level interface such as packet transport mechanisms, flow control, electrical characteristics, and low-level error management.

This application note concentrates on the input/output logical specification. The RapidIO block implements the 1x/4x serial RapidIO physical layer.

2.2 Processing Element Models

Several types of devices can be used in a RapidIO based system. Only two device model types are described here: the integrated processor-memory processing element and the switch processing element.

2.2.1 Integrated Processor-Memory Processing Element

One form of the processor-memory processing element is a fully integrated component that connects 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 MPC8548 device.

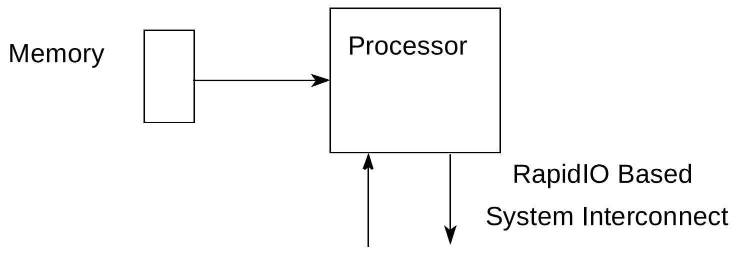


Figure 1. Integrated Processor-Memory Processing Element

2.2.2 Switch Processing Element

A switch processing element allows communication with other processing elements through a switch. A switch can 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 Tsi568 Serial RapidIO switch [3].

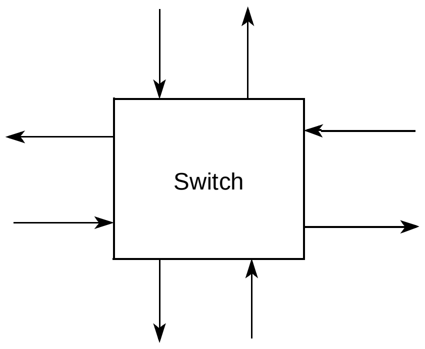


Figure 2. Switch Processing Element Model

2.2.3 CAR/CSR Block

The RapidIO specification defines a block of capability attributes registers (CARs) and command and status registers (CSRs) on each RapidIO device. These CAR/CSR registers are accessed using maintenance transactions, and they are used to configure a device across RapidIO. Each register has a maintenance offset used in the maintenance transaction to select the register to be accessed. Table 2 shows an extract from the list of CAR/CSR registers and their offsets. For details, consult the RapidIO specifications [1] or the *MPC8548E PowerQUICC III™ Integrated Host Processor Family Reference Manual*.

Table 2. Extract from the table of CAR/CSRs

Offset	Register Name
0x00	Device Identity CAR
0x04	Device Information CAR
0x08	Assembly Identity CAR
0x0C	Assembly Information CAR
0x10	Processing Element Features CAR
0x14	Switch Port Information CAR

2.2.4 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. Each RapidIO transaction has a source and destination ID. The RapidIO specification defines two transport modes: large and small. In a small transport systems, source and destination IDs are 8 bits, and therefore a system can consist of up to 256 devices. In a large transport system, with 16-bit source and destination IDs, up to 65,536 devices can be supported. In this discussion, there are three different classes of requests and their associated responses. A request is a transaction issued by a processing element to accomplish an activity on a remote processing element.

The three transactions types are as follows:

- *Maintenance transaction*. A special system support request. by a processing element to read or write data to CARs and CSRs, as defined in the RapidIO specification [1, 4]. Instead of addresses, maintenance requests use an offset into the CAR/CSR block to specify the register to be read or written. Maintenance transactions are used for system initialization and discovery and for altering system configuration.

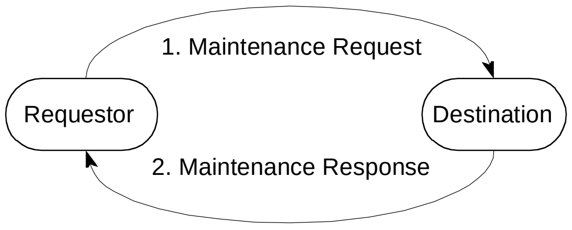


Figure 3. Maintenance Transaction

- *NREAD transaction*. A processing element uses a read transaction, consisting of the NREAD request and corresponding RESPONSE, to read data from a specified address. The data is returned in a response packet and is of the size requested.

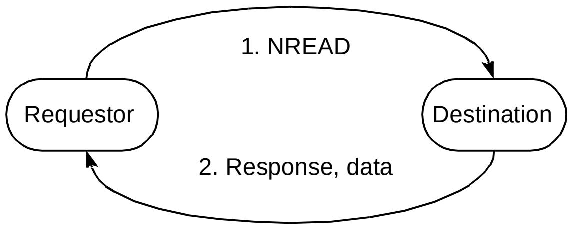


Figure 4. NREAD Transaction

- *NWRITE/NWRITE_R transaction*. A processing element uses a write transaction 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, as shown in Figure 5.

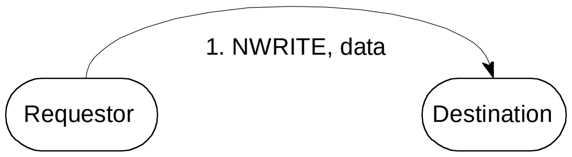


Figure 5. NWRITE Transaction

NWRITE_R (write with response) transaction. Identical to the NWRITE transaction except that it requires a response to notify the sender that the write has completed at the destination, as shown in Figure 6. This transaction is useful for guaranteeing read-after-write and write-after-write ordering through a system that can reorder transactions.

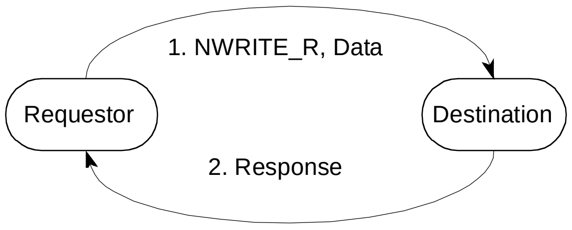


Figure 6. NWRITE_R Transaction

2.3 Serial RapidIO on the MPC8548

This section describes the implementation of RapidIO on MPC8548 devices, including other MPC8548 modules (for example, the MMU) that must be considered in the use of RapidIO. The MPC8548 1x/4x serial interface conforms to the *RapidIO Interconnect Specification* [1]. This RapidIO interface operates at up to 3.125Gbaud (unidirectional data rate 2.5Gbps, per lane).

The MPC8548 RapidIO controller is partitioned into inbound and outbound blocks, which are further divided into three implementation layers that loosely correspond to the logical, common transport, and physical layers of the RapidIO interconnect specification. Users access the RapidIO interface mainly at the logical and transport layers. The user does not directly control the physical layer.

The MPC8548 RapidIO implementation has a messaging unit that implements Part II of *RapidIO Interconnect Specification*.[1]. **F**or details, see Freescale application note AN2923, *Using the Serial RapidIO Messaging Unit on PowerQUICC III*[5]. Other Freescale devices implement the 8-bit parallel RapidIO, and there are separate application notes on the operation of these devices [6, 7].

2.3.1 Mapping Memory to the RapidIO Interface

Because accesses to the RapidIO interface are memory-mapped, we must consider not only the specifics of the RapidIO interface but also the way memory accesses are redirected to the RapidIO interface.

2.3.1.1 Memory Management Unit

The MPC8548 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. The memory management unit (MMU) uses software managed translation lookaside buffers (TLBs) that support variable-size pages, with per-page properties and permissions.

Although the MMU is not part of the RapidIO interface, it must be notified of any memory area that is translated to any interface, including RapidIO. Before RapidIO transactions are attempted, there must be a TLB entry to cover the area of the memory map used for RapidIO transactions. This entry can be initialized by a bootloader or a debugger. However, if a TLB entry is not initialized to cover the RapidIO region of the memory map, the bring-up software must do this.The TLB is initialized by the MMU assist (MAS) registers, which are initialized with information on the TLB entry (for example the required address range, size, and permissions). The **tlbwe** instruction is used to write the information into the TLB.

2.3.1.2 Local Memory Map and Local Access Windows

The MPC8548 local memory map refers to the address space from the processor and the internal DMA engines as they access memory and I/O space. In this map are memory-mapped configuration, control and status registers, and all memory accessible to the DDR SDRAM, local bus memory controllers, and other interfaces.

All addresses used by the system except configuration space (mapped by CCSRBAR), and the internal SRAM regions (mapped by L2SRBAR), must be mapped by a local access window (LAW). The local access windows are not specific to RapidIO, but they must cover the area of memory used to generate RapidIO transactions. The LAW is initialized by the bootloader, the debugger, or the bring-up software. If a LAW is not used, the bring-up software must cover the region of the memory map used for RapidIO. A set of 10 local access windows 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. The LAWs do not handle address translation. Each window can be configured from 4 Kbytes to 32 GBytes.

2.3.2 RapidIO ATMU (Address Translation and Mapping Unit)

The MPC8548 RapidIO controller is partitioned 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 for the internal interfaces.

2.3.2.1 RapidIO Outbound ATMU Windows

RapidIO outbound ATMU windows (ROWs) map the 36-bit internal address space to the 34-bit RapidIO address space. 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 the default window if the address does not match one of the other eight.

Each window can be divided into 2 or 4 segments of equal size. Each segment assigns attributes and the device ID for an outbound transaction, but each segment within a window translates to the same translation address. Each segment can be further sub-divided into 2, 4, or 8 subsegments of equal size. These subsegments allow a single segment to target a number of numerically adjacent target device IDs, using the same attributes and the same address translation.

NOTE

Errata in early MPC8548 silicon prevents the use of segmented windows. Therefore, although the correct operation of the segments is described in this document, they are not used in the example software.

The default RapidIO outbound window is defined by three registers:

- Translation address register (ROWTAR*n)*. Contains the RapidIO base address to which these transactions are translated. ROWTAR takes one of two formats depending on the type of transaction. For regular transactions (such as NREAD, NWRITE, and NWRITE_R) it contains the device ID of the target and bits 0–21 of the translated address. For maintenance transactions without an address, ROWTAR contains the device ID of the target, the hopcount, and the upper bits of the maintenance offset into the CAR/CSR block.
- Attributes register (ROWAR*n*). Contains information on the window (for example, its size) and the types of transactions that are generated by it (for example, NWRITE, NWRITE_R, or maintenance write).
- Base address register (ROWBAR*n*). Contains the base address of the window in the local memory address space. There is no ROWBAR for the default window.

In addition, each window has a set of segment registers (ROWS*x*R*n*) to define the attributes of each segment. For details on these registers, and their bit definitions, see Section 3, “Bring-up Procedure.”

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

2.3.2.2 RapidIO Inbound ATMU Windows

RapidIO inbound ATMU windows (RIWs) translate addresses from the 34-bit external RapidIO address space to the 36-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 the default window if the address does not match one of the other four windows.

The default RapidIO inbound window is defined by three registers:

- Translation address register (RIWTAR*n*).Contains the local base address to which the transactions are translated.
- Attributes register (RIWAR*n*). Contains information on the window (size, for example) and what should be done with the transactions received through it (for example, snoop or not snoop local processor).
- Base address register (RIWBAR*n*). Contains the base address of the window in RapidIO address space. There is no RIWBAR for the default window.

2.3.2.3 Inbound ATMU Local Configuration Space Window

An additional inbound window 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 that 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.

2.4 Example Target Hardware

This section describes the system referenced in the examples presented in Section 3, “Bring-up Procedure.” The system either runs a bootloader and then the bring-up application is downloaded and run from this bootloader, or the system is initialized by a debugger and the code is downloaded and run across JTAG. Here, it is assumed that most system setup and initialization (for example, CCSRBAR, DDR, and Flash) is complete. No assumptions are made about the initialization of the RapidIO interface.

The fabric-based system in the example has an AMC carrier card featuring a Tundra Tsi568 serial RapidIO switch[3] and a number of MPC8548 AMC cards. The Tsi568 is a 16 port (x1) or 8 port (x4) RapidIO switch. In the example hardware, only three ports are routed to the AMC connectors. Any SRIO-capable AMC cards can be inserted into the AMC connectors. For simplicity, we assume that only MPC8548 boards are to be inserted. In the flexible hardware architecture depicted in Figure 7, several MPC8548 devices can connect to a single Tsi568. Only one of these processors can be configured as a RapidIO host, and this device has a device ID of 0x0. All other processors in the system are configured as RapidIO agents and have an initial device ID of 0xFF. The agent devices can be configured to boot from their local Flash memory devices or from RapidIO. The boot image on the Flash memory device connected to the host processor can be used to boot the host and/or the agent devices. This 4 Mbyte boot image is at addresses 0xFFC0_0000–0xFFFF_FFFF.

3 Bring-up Procedure

The procedure to bring up basic RapidIO capability on a MPC8548 system may not apply to all systems. An application using this procedure was created and tested on the hardware system described in Section 2.4, “Example Target Hardware.” This procedure executed on the host to discover and bring up the other elements in the system.

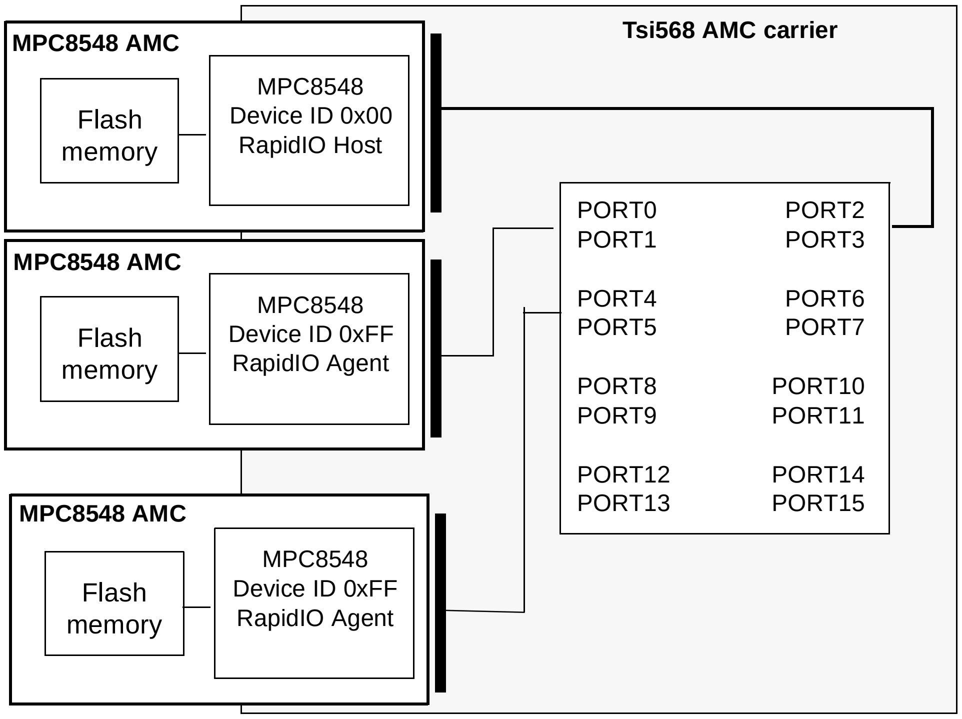


Figure 7. Fabric-Based Hardware and Example Configuration

The subsections that follow are organized around five overall steps in the bring-up procedure:

1. Configure the local processor.

   Configure and check the local processor before generating any RapidIO traffic. Set up a TLB entry, set up a local area window, check lane synchronization and alignment, and set up a maintenance window.

2. Discover all other devices in the system.

   Using only maintenance transactions, identify adjacent devices, set up a switch, and discover other endpoints. Assume limited flexibility in the configuration of the system (as described in Section 2.4, "Example Target Hardware"). This step does not satisfy all requirements of the RapidIO SBTG documentation [4].

3. Enable access to remote configuration space.

   Enable the local processor (host) to access the local configuration space of the remote device (agent). Set up the incoming window on the agent that redirects RapidIO transactions to the local configuration space, and set up an outbound window on the host to map outgoing transactions onto that window.

4. Boot over RapidIO.

   Configure an agent to boot over RapidIO, that is, to boot from Flash memory attached to host. Set up an inbound window on the host to capture the incoming boot reads, and configure and release the agent to execute its boot procedure.

5. Enable memory reads and writes.

   Enable host access to agent memory space. Set up the outbound window on the host and the inbound window on the agent.

Restrictions on software operation that cause the system to halt and report the findings to the user are as follows:

- This software operates on a small transport mode RapidIO system.
- All devices are MPC8548 or Tsi568 only.

- No more than one Tsi568 can be found.
- The software runs on a host with device ID 0.
- All other processors in the system are agents with an initial device ID of 0xFF.
- The software allocates device ID 1 and 2 to any other processors it discovers.
- The software does not find any more than two agent devices.

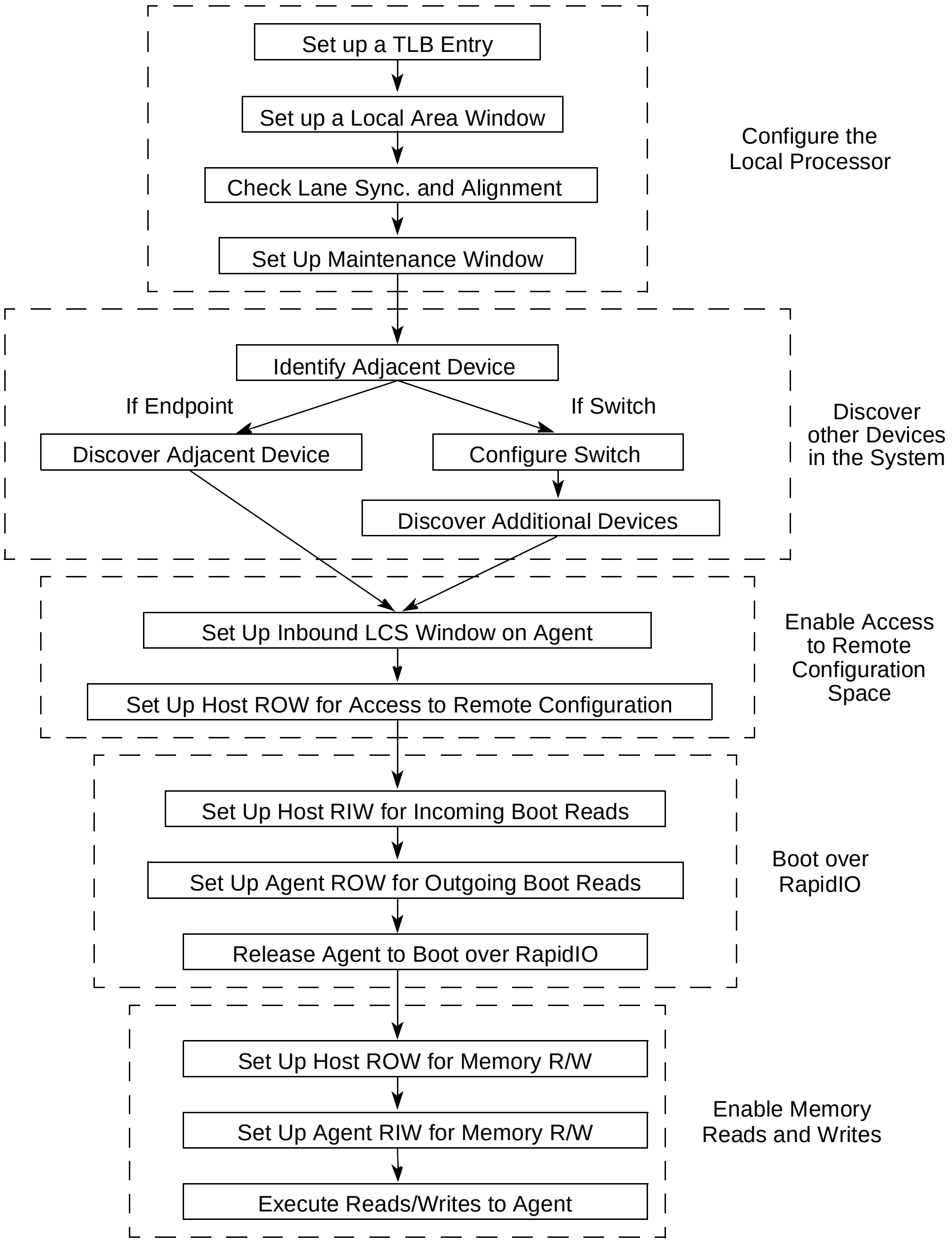

Set up a TLB Entry
Set up a Local Area Window
Check Lane Sync. and Alignment
Set Up Maintenance Window
Configure the Local Processor
Identify Adjacent Device
If Endpoint
If Switch
Discover Adjacent Device
Configure Switch
Discover Additional Devices
Discover other Devices in the System
Set Up Inbound 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 8. Procedure for Bringing Up a Simple RapidIO System

3.1 Configure the Local Processor

Before RapidIO traffic can be generated, several steps must be completed on the local (host) processor.

3.1.1 Set up a TLB Entry

The first step is to set up a TLB entry to cover the area of memory used for RapidIO accesses, which may or may not be necessary, depending on the configuration set by the bootloader or debugger. This example presents the minimum configuration for a 256 Mbyte entry covering address range 0x0_C000_0000–0x0_CFFF_FFFF. This area is used throughout the examples to access RapidIO. Entry 3 of TLB1 is used, but any unused entry in TLB1 can be used (some entries are used by bootloader/debugger configuration to cover CCSR, Flash, DDR and so on).

The TLB entry is created by initializing the relevant MAS registers with information on the area of memory and then loading this information into the TLB. MAS registers are special-purpose registers that are initialized using the **mtspr** instruction. For example, to load the value required into MAS0 (SPR 0x270), use the following sequence:

```
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 9. MAS0 Setting for RapidIO Mapping

MAS0 contains the MMU Read/Write and replacement control. Table 3 lists the settings for the example.

Table 3. 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 10. MAS1 Setting for RapidIO Window

MAS1 contains the descriptor context and configuration control. The settings in the example have the definitions listed in Table 4.

Table 4. 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 to 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				—					X0	X1	W	I	M	G	E
Setting	0x0				0x008											

Figure 11. MAS2 Setting for RapidIO Window

MAS2 contains the effective page number and page attributes. The settings in this example have the definitions listed in Table 5.

Table 5. 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
57	X0	Implementation-dependent page attribute 0
58	X1	Implementation-dependent page attribute 0
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	SR
Setting	0x0				0x03F											

Figure 12. MAS3 Setting for RapidIO Window

MAS3 contains the real page number and access control. The settings in the example have the definitions listed in Table 6.

Table 6. MAS3 Field Descriptions and Settings

Bits	Name	Description and Setting
32–51	RPN	Real page number. 0xC0000 The real address of this space starts at 0x0_C000_0000. The higher order bits of the 36-bit effective page number are contained in MAS7.
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 Initialize MAS7

	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
Field	-															
Setting	0x0000															

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

Figure 13. MAS7 Setting for RapidIO Window

MAS3 contains the real page number and access control. The settings in the example have the definitions listed in Table 7.

Table 7. MAS7 Field Descriptions and Settings

Bits	Name	Description and Setting
60-63	RPN	Higher order bits of the Real page number. 0x0 The real address of this space starts at 0x0_C000_0000.

3.1.1.6 Load Information into the TLB

Finally, execute a **sync** instruction to ensure that all the MAS registers are written, a **tlbwe** instruction to load the information into the TLB, and a final **sync** to ensure that the TLB entry is created. For example:

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

3.1.2 Set Up a Local Area Window

We set up a local area window (LAW) to redirect memory accesses within a certain address range to the RapidIO interface. Whether this step is necessary depends on the configuration set by the bootloader or debugger. In the example, the LAWBAR and LAWAR registers are configured to ensure that a 256 Mbyte block of memory space covering address range 0x0_C000_0000–0x0_CFFF_FFFF is redirected to the RapidIO interface. The LAW registers are memory-mapped within the CCSR area, so they 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	0x00								0x0C							

	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 8. LAWBAR Field Description and Setting

Bits	Name	Description and Setting
12–31	BASE_ADDR	Identifies the 24 most-significant address bits of the 36-bit base address of local access window *n*. 0x0C0000 The address of this space starts at 0x0_C000_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 to which the transactions are directed, and sets the size of the window. The settings in the example have the definitions presented in Table 9

Table 9. 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. 0xC Target interface is RapidIO.
26–31	SIZE	Identifies the size of the window from the starting address. 0x1B Window size is 256 Mbytes.

3.1.3 Check Lane Synchronization and Alignment

Before creating any RapidIO traffic, we must ensure that the RapidIO interface is successfully synchronized with the adjacent device,[4] which is indicated in the error and status command and status register (ESCSR). The ESCSR[PO] bit indicates that the input and output ports are initialized and are exchanging error free control symbols with the attached device. This bit must be set before the RapidIO interface is used. To determine whether the RapidIO port is initialized in 1x or 4x mode, we can check the control command and status register (CCSR). The CCSR[IPW] field indicates the width of the initialized RapidIO port.

3.1.4 Set Up Maintenance Window

After checking synchronization and initialization, we create a RapidIO outbound window to generate RapidIO maintenance transactions. In this example, a single maintenance window is created and used to issue maintenance reads and writes to all the elements of the system. 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, 0x0_C000_0000–0x0_C040_0000. Reads and writes within this window then generate RapidIO maintenance read and write transactions.

In maintenance transactions, the lower-order bits of the maintenance offset are taken from the offset into the maintenance window, but the higher-order bits are taken from the contents of the ROWTAR. Refer to Appendix A for details on the operation of the maintenance windows and the calculation of the maintenance offsets.

3.1.4.1 Set ROWBAR

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—								BEXTADD				BADD			
Setting	0x00								0x0				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 are defined in Table 10.

Table 10. ROWBAR Field Description and Setting

Bits	Name	Description and Setting
8-11	BEXTADD	Window base extended address, bits 0–3 of 36-bit base address 0x0 Extended addressing is not used
12–31	BADD	Base address of outbound window, bits 4-23 of the 36-bit base address. Source address that is the starting point for the outbound translation window. 0xC0000 Window starts at address 0x0_C000_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	—	NSEG		NSSEG		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				0x13							

Figure 17. Maintenance Window ROWAR Settings

The attributes register defines attributes of the transactions created by this window; including the priority and transaction types. It also defines the size of the window. The settings in the example are defined in Table 11.

Table 11. 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.

Table 11. ROWAR Field Descriptions and Settings (continued)

Bits	Name	Description and Setting
6	PCI	Follow PCI transaction ordering rules 0 Do not follow PCI transaction ordering rules.
8-9	NSEG	Number of segments in this window 0 Single-segment (normal) window.
10-11	NSSEG	Number of subsegments per segment 0 No subsegments.
12–15	RDTYP	Read transaction type. Transaction type to run on RapidIO interface if access is a read. 0x7 Maintenance read.
16–19	WRTYP	Write transaction type. Transaction type to run on RapidIO interface if access is a write 0x7 Maintenance write.
26–31	SIZE	Outbound window size. 0x13 One Mbyte.

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 in this window. The ROWTAR takes different forms for different transaction types. For maintenance transactions, it is of the form shown here. In the example, the settings are defined as shown in Table 12.

Table 12. 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 details 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 requirements of the discovery process described in the RapidIO specifications [4]. It provides examples of simplified discovery processes for use with simple systems (that is, with only one host, and not more than one switch). The target hardware system is described in Section 2.4, “Example Target Hardware.”

3.2.1 Identify Adjacent Device

The first RapidIO transaction to 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. Therefore, reading this register enables the identification of the adjacent device. This maintenance transaction is achieved by executing a read instruction within the area covered by the maintenance window. The attributes of this 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. Adjacent device should initially be accessed with hopcount 0.
- Maintenance offset = 0x00.

The returned DIDCAR value should then be compared to the list of DIDCAR values of all devices recognized by the application. If the adjacent device is a processor (for example, a MPC8548), the application should directly discover that processor. If the adjacent device is a switch (for example a Tsi568 switch), this switch must be correctly configured before proceeding with the discovery of the processors beyond it. For the example hardware, the adjacent device is a Tsi568 serial RapidIO switch.

3.2.2 Initial Configuration of Switch

This section describes the configuration of the Tsi568 RapidIO switch. Steps may need to be added, removed, or changed for other RapidIO switches.

3.2.2.1 Check Switch Port

We must determine the number of ports on the switch and the port to which the device running the bring-up application is connected. In the Tsi568 switch example, we read the switch port information CAR(RIO_SW_PORT). See Figure 19. A read from this register returns the number of ports on the device and the number of the port from which this register was read. The maintenance read has the following attributes:

- Destination ID = 0xXX, ID is ‘don’t care’ when accessing the first switch in the system.
- Hopcount = 0x00, Transaction applies to first switch.
- Maintenance offset = 0x14 Switch port information CAR is at offset 0x14.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	Reserved															
Setting	0x0000															

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	PORT_TOTAL								PORT_NUM							
Setting	0x10								Undefined.							

Figure 19. Format of Tsi568 RIO_SW_PORT Register

3.2.2.2 Read Back the Switch HBDIDLCSR

Each RapidIO device has a register (HBDIDLCSR) containing the device ID of the element in the system that configures it. See Figure 20. This register provides a locking mechanism, and an element should proceed to configure a device only 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 the first switch in the system.
- Hopcount = 0x00, Transaction applies to first switch
- Maintenance offset = 0x68. HBDIDLCSR is at offset 0x68

If the agent is not locked, the default value of this HBDIDLCSR is 0x0000_FFFF. If a read of HBDIDLCSR yields any other value, the device is already locked, and the mechanism to deal with this situation is beyond the scope of this application note.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—															
Setting	0x0000															

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

Figure 20. Format of the HBDIDLCSR

3.2.2.3 Lock HBDIDLCSR of Switch

To identify the processor running this application as the device that initializes the switch, its device ID is written into the switch HBDIDLCSR using a maintenance write with the following attributes:

- Destination ID = 0xXX, ID is ‘don’t care’ when accessing the first switch in the system.
- Hopcount = 0x00, Transaction applies to first switch.
- Maintenance offset = 0x68, Offset to HBDIDLCSR

3.2.2.4 Confirm that Switch Has Accepted Lock

The HBDIDLCSR is read to confirm that it is updated with the ID of the host device. If not, the lock was not accepted, and the host should not proceed to configure the switch. Rejection of the lock suggests that there is another host in the system, which is beyond the scope of this discussion. Refer to the interoperability specification for 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 the first switch in the system.
- Hopcount = 0x00, Transaction applies to first switch.
- Maintenance offset = 0x68, Offset to HBDIDLCSR

3.2.2.5 Route Responses Back to Host

To ensure that responses issued by the processors beyond the switch are correctly routed to the requesting processor, we must initialize the routing information of the switch with that information. (Maintenance transactions to the switch are always routed back to the port from which they were received). The Tsi568 maintains an independent look-up table for each port of the switch, so we can create different routing information on a per-port basis. However, it can also update all routing tables simultaneously using an indirect read/write mechanism. In this example, the routing tables for all ports are identical, and the indirect read/write mechanism is used. Figure 21 and Figure 22 show the RIO_ROUTE_CFG_DESTID and the RIO_ROUTE_CFG_PORT registers, which are used to update the routing tables for the whole device.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	-															
e.g	0x0000															
	16	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
Field	LRG_CFG_DEST_ID								CFG_DEST_ID							
e.g.	0x00								0x00							

Figure 21. RIO_ROUTE_CFG_DESTID

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	-															
e.g.	0x0000															
	16	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
Field	-								PORT							
e.g.	0x00								0x02							

Figure 22. RIO_ROUTE_CFG_PORT

For example, if the host processor was connected to port 2, all packets bound for device ID 0 should be directed to port 2. The routing tables are updated with this information in 2 maintenance writes. First, write the device ID to be updated, in this case 0x00, to the RIO_ROUTE_CFG_DESTID using a maintenance write with the following attributes.

- Destination ID = 0xXX, ID is ‘don’t care’ when accessing the first switch in the system.
- Hopcount = 0x00, Transaction applies to first switch.
- Maintenance offset = 0x70

Then write the number of the port to which packets for this device should be directed (in this case 0x02), to the RIO_ROUTE_CFG_PORT using a maintenance write with the following attributes.

- Destination ID = 0xXX, ID is ‘don’t care’ when accessing the first switch in the system.
- Hopcount = 0x00, Transaction applies to first switch.
- Maintenance offset = 0x74

3.2.3 Discover the Devices Beyond the Tsi568 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 it is beyond the scope of this simple application. Further details of how to discover systems with multiple hosts, or multiple switches can be found in the interoperability specification.[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).

3.2.3.1 Check Lane Synchronization and Alignment on Port *N*

Ensure that the port under investigation (port *n*) is initialized. If it is not initialized, then the remaining steps are omitted for this iteration. In the Tsi568 switch example, each of the ports has an error and status CSR (SPn_ERR_STAT). SPn_ERR_STAT[PORT_OK] can be examined to determine if each port has successfully completed the RapidIO lane synchronization and alignment 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 = 0x158 (port0), 0x178 (port1), 0x198 (port2),....0x338 (port15)

3.2.3.2 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 using the indirect routing update mechanism described in Section 3.2.2.5, “Route Responses Back to Host.” A value of 0xFF is written to the RIO_ROUTE_CFG_DESTID and the port number is written to RIO_ROUTE_CFG_PORT. For example, packets with a destination ID of 0xFF can be routed to port 4 by two maintenance writes. A maintenance write of 0xFF to RIO_ROUTE_CFG_DESTID sets the destination ID for which the routing is to be changed:

- Destination ID = 0xXX, ID is ‘don’t care’ when accessing the first switch in the system.
- Hopcount = 0x00, Transaction applies to first switch.
- Maintenance offset = 0x70.

A maintenance write of *N* to RIO_ROUTE_CFG_PORT updates the port to which these packets will be routed.

- Destination ID = 0xXX, ID is ‘don’t care’ when accessing the first switch in the system.
- Hopcount = 0x00, Transaction applies to first switch.
- Maintenance offset = 0x74.

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

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

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

The read of the DIDCAR is a maintenance read with the following attributes:

- Destination ID = 0xFF, accessing the agent with a destination ID of 0xFF.
- Hopcount=0xFF, transaction bypasses switch.
- Maintenance offset = 0x00, offset to DIDCAR.

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

Each RapidIO device has a register (HBDIDCSR) 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 is read using a maintenance read with the following attributes:

- Destination ID = 0xFF, accessing the agent with a destination ID of 0xFF
- Hopcount = 0xFF, transaction bypasses switch.
- Maintenance offset = 0x68, offset to HBDIDLCSR

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

3.2.3.5 Lock HBDIDLCSR of Device on Port *N*

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

- Destination ID = 0xFF, accessing the agent with a destination ID of 0xFF
- Hopcount = 0xFF, transaction bypasses switch.
- Maintenance offset = 0x68, offset to HBDIDLCSR

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

The HBDIDLCSR register is read to confirm that it is updated with the host device ID. If not, the lock was not accepted, and the host should not further configure the device on this port. Rejection of the lock suggests that there is another host in the system, which is beyond the scope of this example. Refer to the SBTG specification for 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 a destination ID of 0xFF.
- Hopcount = 0xFF, transaction bypasses switch.
- Maintenance Offset = 0x68, offset to HBDIDLCSR

3.2.3.7 Update the Device ID

All the agents have an initial device ID of 0xFF. After the host discovers them, the host must allocate a permanent device ID by writing into the base device ID command and status register (BDIDCSR). See Figure 24. In the example, the host device, connected to Port 0, has a device ID of 0x00. The host allocates device IDs in ascending order to the agents it discovered. That is, the first agent to be discovered becomes Device ID 1, the next to be discovered becomes Device ID 2, and so on. In the BDIDCSR, the BDID field contains the device ID required in small transport systems, and LBDID contains the additional device ID bits required in a large transport system. The BDIDCSR can be updated by executing a maintenance write with the following attributes:

- Destination ID = 0xFF, accessing the agent with a destination ID of 0xFF.

- Hopcount = 0xFF, transaction bypasses switch.
- Maintenance Offset = 0x60, offset to BDIDCSR

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

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

Figure 24. Format of the BDIDCSR

3.2.3.8 Update the Routing Table, All Ports

After the BDIDCSR is updated with the new device ID, the routing table of the switch must be updated to reflect this information using the indirect write mechanism described in Section 3.2.2.5, “Route Responses Back to Host.” The device ID allocated in the previous step should be written to the RIO_ROUTE_CFG_DESTID, and the number of the port to which it connects should be written to RIO_ROUTE_CFG_PORT.

3.2.3.9 Read Back from Updated Device ID

When the device ID is set, we verify that the device can be accessed using that device ID as the destination. That is, update the ROWTAR with the new device ID and read from the BDIDCSR with the following attributes:

- Destination ID = 0x02, 0x03, and so on. 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 25. ROWTAR for Accessing BDIDCSR on Device 3

3.3 Enable Access to Remote Configuration Space

To access the local configuration (CCSR) space on a remote processor, complete the following steps:

1. On the remote processor, set up the LCSBA1CSR to accept RapidIO transactions in a certain address range.
2. On the local processor, set up an outbound window with the correct address translation into the same range.

NOTE

When the remote access to the remote configuration space is enabled, all further accesses to the agent devices occur by this method rather than by maintenance transactions.

3.3.1 Set Up Inbound LCS Window on Agent

We set up the LCSBA1CSR of each of the processor to redirect RapidIO arriving at the 1 Mbyte window, starting at RapidIO address 0x0_0100_0000, to the CCSR area by setting the LCSBA1CSR as described here.

	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 26. LCSBA1CSR Settings

In the example, LCSBA = 0x0010 and the 1 Mbyte LCS window starts at RapidIO address 0x0_0100_0000. (LCSBA contains the most significant 14 bits of the 34-bit address). The update of the LCSBA1CSR on the agent is achieved using a maintenance write with the following attributes:

- Destination ID = 0x02, 0x03, and so on, depending on which agent is accessed.
- Hopcount = 0xFF, transaction bypasses switch.
- Maintenance offset = 0x5C, Offset to LCSBA1CSR.

3.3.2 Set Up Outbound Window on Host

To map outbound transactions into the address space with the appropriate destination device ID, a RapidIO outbound window is set up on the host for each agent processor. For each device found, a 1 MByte window is set up to map the outbound transactions to the LCSBAR on the agent. In the example, a 1 MByte window starting at address 0xc110000 is used to map RapidIO transactions to the LCS space of deviceID 1.

NOTE

We can also target multiple devices with the same translation address via a single, segmented window. However, errata in early MPC8548 silicon prevents the use of segments and sub-segments for targeting the same address space on multiple devices.

3.3.2.1 ROWBAR

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	BADD															
Setting	0x1100															

Figure 27. ROWBAR Settings for Remote CCSR Access

The base address register (ROWBAR), defines the start address of the window. The settings in the example are defined in Table 13.

Table 13. ROWBAR Field Description and Setting

Bits	Name	Description and Setting
8-11	BEXTADD	Window base extended address. Contains bits 0-3 of 36-bits address. 0x0 Extended addressing not used.
12–31	BADD	Base address of outbound window. Source address that is the starting point for the outbound translation window. 0xC1100, Window starts at address 0x0_C110_0000.

3.3.2.2 ROWAR

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	EN	—			TFLOV		PCI	—	NSEG		NSSEG		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	0x5				0x0				0x13							

Figure 28. Remote CCSR Window ROWAR Settings

The attributes register defines attributes of the transactions created by this window. This includes the priority and transaction types. It also defines the size of the window. The settings in the example are defined in Table 14.

Table 14. 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.

Table 14. ROWAR Descriptions and Settings

Bits	Name	Description and Setting
6	PCI	Follow PCI transaction ordering rules. 0 Do not follow PCI transaction ordering rules.
8-9	NSEG	Number of segments in this window. 00 Single segment.
10-11	NSSEG	Number of sub-segments in this window. 00 No sub-segments.
12–15	RDTYP	Read transaction type. Transaction type to run on RapidIO interface if access is a read. 0x4 NREAD.
16–19	WRTYP	Write transaction type. Transaction type to run on RapidIO interface if access is a write. 0x5 NWRITER.
26–31	SIZE	Outbound window size. Outbound window size n, which is the encoded 2^{n+1}-byte window size. 0x13 Window size is 1Mbyte.

3.3.2.3 ROWTAR

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	LTGTID		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	0x1000															

Figure 29. ROWTAR*n* Settings for Remote CCSR Access

The settings in the example are defined in Table 15.

Table 15. ROWTAR*n* Descriptions and Settings

Bits	Name	Description and Setting
0–1	LTGTID	Corresponds to bits 6-7 of device ID in large transport systems. 0x00 This application note assumes small transport system.
2–9	TRGTID	Target ID for RapidIO packet. 0x01 Transactions have destination ID of 1.
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 verify that the local configuration space of the agent can be read in this way. For example, with the ROW on the host and the LCS window on agent 1 set up as shown in the preceding figures, the accesses should return the following values:

- Reading host processor address 0x0_C110_0000 returns the CCSRBAR of processor 1.
- Reading host processor address 0x0_C11C_0000 returns the DIDCAR of processor 1.
- Reading host processor address 0x0_C11C_005C returns the LCSBA1CSR of processor 1.

3.4 Boot over RapidIO

In the example hardware, some agents can be configured to boot from their local Flash devices, and others can be configured to boot over RapidIO. This section describes how to boot over RapidIO. A device that is to boot over RapidIO must be configured as follows:

- The cfg_rom_loc configuration signal is configured to direct accesses to the boot vector and default boot ROM region to the RapidIO interface
- The cfg_cpu_boot configuration signal is configured to put the processor in boot hold-off mode so that it does not attempt to boot until it is configured by an external processor.

With these configurations, the host completes a series of steps to configure the system and initiate booting of the agent processor.

3.4.1 Prepare the Host Processor for Incoming Boot Reads

The host processor provides an inbound RapidIO window that accepts the boot code reads over RapidIO and redirects them to the appropriate area of Flash memory. With opportunities to adjust window sizes and translate addresses on the outgoing window of the agent and the incoming window of the host, many different configurations permit the agent to boot over RapidIO. This section details only one possibility.

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				0x0			

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	BADD															
Setting	0x0020															

Figure 30. RIWBAR Settings on Host for Incoming Boot Reads

The RapidIO inbound window base address register (RIWBAR) contains the RapidIO base address of the incoming window. The settings in the example are defined in Table 16.

Table 16. 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. 0x0_0200 Base address are 0x0_0020_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				0x5			

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

Figure 31. 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 are defined in Table 17.

Table 17. 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. 0x5 Snoop local processor.
16–19	WRTYP	Write transaction type. Transaction type to run if access is a write. 0x0 Reserved (effectively makes this window read only).
26–31	SIZE	Inbound window size. 0x17 Window size is 16 Mbytes.

3.4.1.3 RIWTAR

The RapidIO inbound window translation address register (RIWTAR) contains the translation address for the incoming transactions.

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

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

Figure 32. RIWTAR Settings on Host for Incoming Boot Reads

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

Table 18. RIWTAR Field Description and Setting

Bits	Name	Description and Setting
8-9	TREXAD	Top bits of the 36-bit local address.
12–31	TRAD	Translation address of inbound window. System address that represents the starting point of the inbound translated address. 0xF_F000, translated window starts at 0x0_FF00_0000.

3.4.2 Configure the Agent Processor

A single processor is configured and released so that it can boot. 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 because of 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 RapidIO Outbound Window of the Agent

The boot location configuration signal cfg_rom_loc, directs all accesses to the default boot location to the RapidIO interface. Before the agent is released to boot, an outbound RapidIO window must be created to direct all accesses to the host processor at the correct RapidIO address.

3.4.2.2 ROWBAR

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

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

Figure 33. Remote CCSR ROWBAR Settings

The base address register (ROWBAR) defines the start address of the window. The settings in the example are defined in Table 19.

Table 19. ROWBAR Field Description and Setting

Bits	Name	Description and Setting
8–11	BEXTADD	Window base extended address. Contains bits 0–3 of 36-bits address. 0x0 Extended addressing not used.
12–31	BADD	Base address of outbound window. Source address that is the starting-point for the outbound translation window. 0xFF000, Window starts at address 0x0_FF00_0000.

3.4.2.2.1 ROWTAR

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	TRAD															
Setting	0x2000															

Figure 34. ROWTAR on Agent for Boot Reads

The settings in the example are defined in Table 20.

Table 20. ROWTAR0 Field Descriptions and Settings

Bits	Name	Description
0–1	LTGTID	Bits 6–7 of target ID in large transport systems only. 00 This application assumes small transport system.
2–9	TRGTID	Target ID for RapidIO packet. 0x00 Transactions have destination ID of 0 (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. 0x020000. Translation address 0x0_0200_0000.

3.4.2.2.2 ROWAR

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	EN	—			TFLOV		PCI	—	NSEG		NSSEG		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	0x5				0x0				0x17							

The settings in the example are defined in Table 21.

Table 21. ROWAR 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.
8-9	NSEG	Number of segments in this window. 00 Single-segment window.
10-11	NSSEG	Number of subsegments. 00 one subsegment (and one device ID) per segment.
12–15	RDTYP	Read transaction type. Transaction type to run on RapidIO interface if access is a read. 0x4 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. 0x5 Writes within this window generate NWRITE_R transactions.
26–31	SIZE	Window size. 0x17 Window size is 16 Mbytes.

3.4.2.3 Create LAW on Agent for the Boot Area on RapidIO

Whether it is necessary to create a LAW on the agent depends on the operation of the bootloader program executed over RapidIO. The mapping of the LAW has priority over the default boot location set up by cfg_rom_loc. Therefore, if the bootloader program of the agent sets up a LAW that covers the same area as the address range used for booting, booting over RapidIO fails after the instructions execute to create that LAW.

If the bootloader does not use the highest-priority LAW (LAW0), 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, the bootloader must be amended to permit booting over RapidIO. If the bootloader does not create a LAW that covers the same area of the local map as the boot instructions, the LAW does not have to be set up; the cfg_boot_loc is enough to redirect the boot operation to the correct area. In this example, the local area window is set up to cover addresses 0x0_FF00_0000–0x0_FFFF_FFFF.

3.4.2.3.3 LAWBAR0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	—								BASE_ADDR							
Setting	0x00								0x0F							

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

Figure 35. Agent LAWBAR0 Setting for RapidIO Window

The local area window base address 0 (LAWBAR0) sets the base address of this window. The settings in the example are defined in Table 22.

Table 22. LAWBAR0 Field Descriptions and Settings

Bits	Name	Description and Setting
12–31	BASE_ADDR	Identifies the 24 most-significant address bits of the 36-bit base address of local access window *n*. 0x0FF000 The address of this space starts at 0x0_FF00_0000.

3.4.2.3.4 LAWAR0

	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 36. Agent LAWAR0 Settings for RapidIO Window

The local area window attributes register 0 (LAWAR0) enables this window, sets the interface to which the transactions are directed, and sets the size of the window. The settings in the example are defined in Table 23.

Table 23. LAWAR0 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. 0xC Target interface is RapidIO.
26–31	SIZE	Window size 0x17 Window size is 16 Mbytes.

3.4.2.4 Provide Agent Processor with Access to the RapidIO Bus

The processors configured as RapidIO agents are not enabled to issue RapidIO requests into the system. The host must set the agent's GCCSR[M], master bit, before booting is initiated. This bit is accessed through the memory-mapped LCSBA1CSR described in Section 3.3, "Enable Access to Remote Configuration Space."

3.4.2.5 Enable the CPU

The processor is configured in boot hold-off mode to prevent the agent CPU from booting. To initiate the booting process, the host processor must set the agent's EEBPCR[CPU_EN], CPU port enable bit. This bit is accessed through the memory-mapped LCSBA1CSR described in Section 3.3, "Enable Access to Remote Configuration Space." When this CPU port enable bit is set, the agent boots from the host Flash memory.

3.5 Enable Memory Reads and Writes

After the processors have booted, many simple tests can be run to check functionality and benchmark RapidIO performance. Before the host can access the agent's memory, more ATMU windows must be initialized. A RapidIO outbound window is set up on the host to direct normal reads and writes to the agent processors, and a RapidIO inbound window is set up on the agent to capture these accesses and direct them to the appropriate area of the agent's memory.

For every device found, a 4 MByte outbound window is created on the host, including a window to map transactions to deviceID 0, which permits loopback testing (through switch). Each device is then programmed with a corresponding inbound window to accept these transactions and redirect them to the device's local memory. In this example, the memory area of each agent starts at address 0x0_0100_0000.

NOTE

Multiple devices with the same translation should also be targetable with a single, segmented outbound window. However, errata in early MPC8548 silicon prevents the use of segments and sub-segments for targeting the same address space on multiple devices.

3.5.1 Host Setup

A RapidIO outbound window is set up on the host to direct local accesses:

- In the range 0x0_C600_0000–0x0_C63F_FFFF to processor 0.
- In the range 0x0_C640_0000–0x0_C67F_FFFF to processor 1.

And so on, all with RapidIO address 0x0_0000_0000–0x0_003F_FFFF.

3.5.1.1 ROWBAR

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Field	BADD															
Setting	0x6000															

Figure 37. Remote Memory ROWBAR Settings

The base address register (ROWBAR) defines the start address of the window. The settings in the example are defined in Table 24.

Table 24. ROWBAR Field Description and Setting

Bits	Name	Description and Setting
8-11	BEXTADD	Bits 0-3 of the 36-bit address. 0x0 Not set in this example.
12–31	BADD	Base address of outbound window. Source address that is the starting point for the outbound translation window. 0xC6000 Window starts at address 0x0_C600_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	—	NSEG		NNSEG		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	0x5				0x0				0x15							

Figure 38. Remote Memory Window ROWAR Settings

The attributes register defines attributes of the transactions created by this window, including the priority and transaction types. It also defines the size of the window. The settings in the example are defined in Table 25.

Table 25. 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.

Table 25. ROWAR Field Descriptions and Settings (continued)

Bits	Name	Description and Setting
6	PCI	Follow PCI transaction ordering rules. 0 PCI transaction ordering is not followed.
8-9	NSEG	Number of segments. 00 Single Segment window.
10-11	NSSEG	Number of sub-segments. 00 No sub-segments.
12–15	RDTYP	Read transaction type. Transaction type to run on RapidIO interface if access is a read. 0x4 Reads to this window generate RapidIO NREAD transactions.
16–19	WRTYP	Write transaction type. Transaction type to run on RapidIO interface if the access is a write. 0x5 Writes to this window generate RapidIO NWRITE_R transactions.
26–31	SIZE	Outbound window size. 0x15 Window size is 4 Mbytes.

3.5.1.3 ROWTAR

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	LTGTID		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 39. Remote Memory ROWTAR Settings

The settings in the example are defined in Table 26.

Table 26. ROWTAR Field Descriptions and Settings

Bits	Name	Description and Setting
0-1	LTGTID	Bits 6–7 of targetID for large transport systems only. 00 Small transport system.
2–9	TRGTID	Target ID for RapidIO packet 0x01 Target device ID 1.
10–11	TREXAD	Translation extended address of outbound window. 00 The two 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 4 Mbyte RapidIO inbound window is set up on the agent to capture RapidIO transactions with the address range 0x0_0000_0000–0x0_003F_FFFF and redirect them to the agent's local memory space in the address range 0x0_0100_0000–0x0_013F_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 40. RIWBAR Settings on Host for Incoming Boot Reads

The settings in the example are defined in Table 27.

Table 27. 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				0x15							

Figure 41. RIWAR Settings on Host for Incoming Boot Reads

The settings in the example are defined in Table 28.

Table 28. 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. 0x15 Window size is 4 Mbyte.

3.5.2.3 RIWTAR

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

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

Figure 42. RIWTAR Settings on Host for Incoming Boot Reads (Point-to-Point)

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

Table 29. RIWTAR Description and Setting

Bits	Name	Description and Setting
8-11	TREXAD	Translation extended address. Bits 0–3 of the 36-bit address. 0x0 extended addressing not used in this example.
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 0x0_0100_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 that requires transactions to have 32 bits of payload or less, simple reads and writes to the 4 Mbyte address range 0x0_C680_0000–0x0_C6BF_FFFF on the host result in RapidIO transactions to processor 2. When processor 2 receives the requests, they are translated to the 4Mbyte address range 0x0100_0000–0x013F_FFFF. To confirm correct operation, these memory locations can be directly examined 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 0x0_C680_0000, this generates RapidIO read/writes to processor 2. Using the DMA can generate RapidIO reads and writes with a payload greater than 32 bits, providing a more efficient way to pass large amounts of data between devices that is convenient 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.

```
## Starting application at 0x00040004 ...
Check LAWs before setting up
lawbar0 = 0x00000000, lawar0 = 0x00000000
lawbar1 = 0x00000000, lawar1 = 0x80f0001c
lawbar2 = 0x000f0000, lawar2 = 0x8040001b
lawbar3 = 0x000a0000, lawar3 = 0x8020001c
lawbar4 = 0x000e3000, lawar4 = 0x80200017
lawbar5 = 0x000c0000, lawar5 = 0x80c0001c
lawbar6 = 0x00000000, lawar6 = 0x00000000
lawbar7 = 0x00000000, lawar7 = 0x00000000
Completed the setup of TLBs and lawbars
lawbar5 = 0x000c0000, lawar5 = 0x80c0001b

Checked DIDCAR of local device is MPC8548
This device is configured as a RIO host
This device initially has RIO device ID 0x00000000
This device has been allocated RIO device ID 0x00000000
RapidIO port is trained OK
RapidIO port trained 4x

Set up the outbound rio maintenance window
rowbar1 = 0x000c0000, rowar1 = 0x80077013, rowtar1 = 0x3fc00000
 ---- rio_local_config() completed successfully ----

Identified adjacent device as Tsi568
Executing the fabric version of the discovery code
read back from SW_PORT = 0x00001002
The switch has 16 ports
This host device is attached to port 2
```

```
Original Tsi568 HBDIDL =0x0000ffff
Updated HBDIDL on Tsi568  =0x00000000
Routed all packets for deviceID 0x00 to port 2
Confirmed route back to host through switch, read bdidcsr

Examine each port on the Tsi568, 'discover' any devices

Examining port 0  Trained successfully
Trained 4x
All packets for DeviceID = 0xFF, now to port 0
Examine didcar of device attached to port 0
Found another MPC8548
Original HBDIDL =0x0000ffff
Updated HBDIDL =0x00000000
Original deviceID = 0x000000ff
Allocated deviceID = 0x00000001
All packets for DeviceID = 0x01, now to port 0
Confirmed read operation using new device id

Examining port 2  Trained successfully
Trained 4x
This port number == host port number. Discovered myself!

Examining port 4  This port is not trained
Examining port 5  This port is not trained
Examining port 6  Trained successfully
Trained 4x
All packets for DeviceID = 0xFF, now to port 6
Examine didcar of device attached to port 6
Found another MPC8548
Original HBDIDL =0x0000ffff
Updated HBDIDL =0x00000000
Original deviceID = 0x000000ff
Allocated deviceID = 0x00000002
All packets for DeviceID = 0x02, now to port 6
```

```
Confirmed read operation using new device id


Examining port 8  This port is not trained

Examining port 9  This port is not trained

Examining port 10  This port is not trained

Examining port 11  This port is not trained

Examining port 12  This port is not trained

Examining port 13  This port is not trained

Examining port 14  This port is not trained

Examining port 15  This port is not trained

Discovered 2 other devices

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

Set up ROWs of local processor for access through LCS window

Allow access back to host, through switch


ROW2 is for LCS access back to device 0

rowbar2 = 0x000c1000, rowar2 = 0x80045013, rowtar2 = 0x00001000

write 0x00200000 to LCSBAR of processor 0.

Initialize pointers to the registers, accessed through LCS

srio_device[0].imm = 0xc1000000

srio_device[0].srio = 0xc10c0000

srio_device[0].ecm = 0xc1000000

Confirmed correct operation of LCS by comparing to maintenance read


ROW3 is for LCS access back to device 1

rowbar3 = 0x000c1100, rowar3 = 0x80045013, rowtar3 = 0x00401000

write 0x00200000 to LCSBAR of processor 1.

Initialize pointers to the registers, accessed through LCS

srio_device[1].imm = 0xc1100000

srio_device[1].srio = 0xc11c0000

srio_device[1].ecm = 0xc1100000

Confirmed correct operation of LCS by comparing to maintenance read


ROW4 is for LCS access back to device 2

rowbar4 = 0x000c1200, rowar4 = 0x80045013, rowtar4 = 0x00801000
```

```
write 0x00200000 to LCSBAR of processor 2.
Initialize pointers to the registers, accessed through LCS
srio_device[2].imm = 0xc1200000
srio_device[2].srio = 0xc12c0000
srio_device[2].ecm = 0xc1200000
Confirmed correct operation of LCS by comparing to maintenance read
 ---- set up remote LCS access to all trained processors ----

ROW5 is for memory access to device 0
rowbar5 = 0x000c6000, rowar5 = 0x80045015, rowtar5 = 0x00000000

ROW6 is for memory access to device 1
rowbar6 = 0x000c6400, rowar6 = 0x80045015, rowtar6 = 0x00400000

ROW7 is for memory access to device 2
rowbar7 = 0x000c6800, rowar7 = 0x80045015, rowtar7 = 0x00800000

Check boot status of device 1
Looks like the device has already been booted
Set up access to device 1's memory space
Set up IB window on agent to permit host to access its memory
Agent riwbar1=0x00000000, riwtar1=0x00001000, riwar1=0x80f55015
&srio_device[1].srio->riwbar1 = 0xc11d0dc8
&srio_device[1].srio->riwtar1 = 0xc11d0dc0
set up memory access to device 1

Check boot status of device 2
don't think this has been booted
srio_device[device_id].imm->im_gur.porbmsr = 0x03310000
Host: riwbar2 = 0x00002000, riwar2 = 0x80f50017, riwtar2 = 0x000ff000
agent device rowbar6 = 0x000ff000
agent device rowar6 = 0x80045017
agent device rowtar6 = 0x00002000
Before booting:
didcar= 0x00120002
```

```
lcsbacsr = 0x00200000

bdidcsr= 0x00020000

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= 0x00120002

lcsbacsr = 0x00200000

bdidcsr= 0x00020000

ccsrbar = 0x000e0000

eebpcr = 0x01000000

agent LAWBAR0 = 0x000ff000

agent LAWAR0  = 0x80c00017

agent LAWBAR1 = 0x00000000

agent LAWAR1  = 0x80f0001c

agent LAWBAR2 = 0x000f0000

agent LAWAR2  = 0x8020001c

agent LAWBAR3 = 0x000a0000

agent LAWAR3  = 0x8020001c
agent LAWBAR4 = 0x000e3000
agent LAWAR4  = 0x80200017
agent LAWBAR5 = 0x000c0000
agent LAWAR5  = 0x80c0001c
agent LAWBAR6 = 0x00000000
agent LAWAR6  = 0x00000000
agent LAWBAR7 = 0x00000000
agent LAWAR7  = 0x00000000
Set up access to device 2's memory space
Set up IB window on agent to permit host to access its memory
Agent riwbar1=0x00000000, riwtar1=0x00001000, riwar1=0x80f55015
&srio_device[2].srio->riwbar1 = 0xc12d0dc8
&srio_device[2].srio->riwtar1 = 0xc12d0dc0
setup memory access to device 2
Set up inbound window on host to re-direct incoming rio to 'safe' area
riwbar1 = 0x00000000, riwar1 = 0x80f55015, riwtar1 = 0x00001000
 ---- Provided LCSBASCR and memory access back to local processor ----

Host processor is deviceid 0
Discovered 2 agent processors
With deviceID =  1   2
Options :
h = print this list of options
q = quit this application
s = read back info from tsi568
r = SRIO setup and status
t = run one of the NREAD/NWRITE tests
R. Read RIO information for which device number?
0
Rio port information from device 0
didcar   = 0x00120002       bdidcsr = 0x00000000
predr    = 0x00000000       pnfedr  = 0x00000000
pccsr   = 0x50600001       pescsr  = 0x00100002
row0 :                tar=0x00000000, ar=0x80044023
row1 : bar=0x000c0000, tar=0x008ff000, ar=0x80077013
```

```
row2 : bar=0x000c1000, tar=0x00001000, ar=0x80045013
row3 : bar=0x000c1100, tar=0x00401000, ar=0x80045013
row4 : bar=0x000c1200, tar=0x00801000, ar=0x80045013
row5 : bar=0x000c6000, tar=0x00000000, ar=0x80045015
row6 : bar=0x000c6400, tar=0x00400000, ar=0x80045015
row7 : bar=0x000c6800, tar=0x00800000, ar=0x80045015
row8 : bar=0x00000000, tar=0x00000000, ar=0x00044023
riw0 :                 tar=0x00000000, ar=0x80044021
riw1 : bar=0x00000000, tar=0x00001000, ar=0x80f55015
riw2 : bar=0x00002000, tar=0x000ff000, ar=0x80f50017
riw3 : bar=0x00000000, tar=0x00000000, ar=0x00044021
riw4 : bar=0x00000000, tar=0x00000000, ar=0x00044021
>R. Read RIO information for which device number ?
1
Rio port information from device 1
didcar    = 0x00120002       bdidcsr = 0x00010000
predr     = 0x00000000       pnfedr  = 0x00000000
pccsr   = 0x50600001       pescsr  = 0x00000002
row0 :                 tar=0x00000000, ar=0x80044023
row1 : bar=0x00000000, tar=0x00000000, ar=0x00044023
row2 : bar=0x00000000, tar=0x00000000, ar=0x00044023
row3 : bar=0x00000000, tar=0x00000000, ar=0x00044023
row4 : bar=0x00000000, tar=0x00000000, ar=0x00044023
row5 : bar=0x00000000, tar=0x00000000, ar=0x00044023
row6 : bar=0x00000000, tar=0x00000000, ar=0x00044023
row7 : bar=0x00000000, tar=0x00000000, ar=0x00044023
row8 : bar=0x00000000, tar=0x00000000, ar=0x00044023
riw0 :                 tar=0x00000000, ar=0x80044021
riw1 : bar=0x00000000, tar=0x00001000, ar=0x80f55015
riw2 : bar=0x00000000, tar=0x00000000, ar=0x00044021
riw3 : bar=0x00000000, tar=0x00000000, ar=0x00044021
riw4 : bar=0x00000000, tar=0x00000000, ar=0x00044021
>R. Read RIO information for which device number ?
2
Rio port information from device 2
```

```
didcar   = 0x00120002       bdidcsr = 0x00020000
predr    = 0x00000000       pnfedr  = 0x00000000
pccsr   = 0x50600001       pescsr  = 0x00000002
row0 :                 tar=0x00000000, ar=0x80044023
row1 : bar=0x00000000, tar=0x00000000, ar=0x00044023
row2 : bar=0x00000000, tar=0x00000000, ar=0x00044023
row3 : bar=0x00000000, tar=0x00000000, ar=0x00044023
row4 : bar=0x00000000, tar=0x00000000, ar=0x00044023
row5 : bar=0x00000000, tar=0x00000000, ar=0x00044023
row6 : bar=0x000ff000, tar=0x00002000, ar=0x80045017
row7 : bar=0x00000000, tar=0x00000000, ar=0x00044023
row8 : bar=0x00000000, tar=0x00000000, ar=0x00044023
riw0 :                 tar=0x00000000, ar=0x80044021
riw1 : bar=0x00000000, tar=0x00001000, ar=0x80f55015
riw2 : bar=0x00000000, tar=0x00000000, ar=0x00044021
riw3 : bar=0x00000000, tar=0x00000000, ar=0x00044021
riw4 : bar=0x00000000, tar=0x00000000, ar=0x00044021
>S. Print Switch info
port       err_status     ctl
port  0      0x00000002    0x50600001
port  1      0x00000001    0x00600001
port  2      0x00000002    0x50600001
port  3      0x00000001    0x00600001
port  4      0x00000001    0x50600001
port  5      0x00000001    0x00600001
port  6      0x00000002    0x50600001
port  7      0x00000001    0x00600001
port  8      0x00000001    0x50600001
port  9      0x00000001    0x00600001
port 10      0x00000001    0x50600001
port 11      0x00000001    0x00600001
port 12      0x00000001    0x50600001
port 13      0x00000001    0x00600001
port 14      0x00000001    0x50600001
port 15      0x00000001    0x00600001
```

```
>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
On which device do you want to run test a ?
loopback test to self, through switch
Running test A for deviceid 0
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
On which device do you want to run test a ?
Running test A for deviceid 1
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
On which device do you want to run test a ?
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
On which device do you want to run test b ?
loopback test to self, through switch
Running test B for deviceid 0
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
On which device do you want to run test b ?
Running test B for deviceid 1
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
On which device do you want to run test b ?
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
```

5 References

1. RapidIO Trade Association, *RapidIO Interconnect Specification*, Rev. 1.2, 6/2002.
2. *MPC8548E PowerQUICC III Integrated Host Processor Family Reference Manual, MPC8548ERM, Rev1, 7/2005.*
3. *Tsi568 user manual*. Tundra document 80B8000_MA001_.
4. *RapidIO Interconnect Specification Annex I: Software/System Bring Up Specification*, Rev. 1.0, 12/2003.
5. Freescale application note AN2923, *Using the Serial RapidIO Messaging Unit on PowerQUICC III.*
6. Freescale application note AN2753, RapidIO Bringup Procedure on PowerQUICC III.
7. Freescale application note AN2741, Using the RapidIO Messaging Unit on PowerQUICC III.

Appendix A Notes on Maintenance Transactions

In this document, all maintenance transactions are described as a list of maintenance parameters (for example, destination ID, hopcount, and maintenance offset), but there is no description of how these maintenance transactions are created. Therefore, this appendix fully describes the operation of the maintenance window and the generation of maintenance transactions. It also describes the operation of maintenance transactions in a multi-switch environment.

A.1 Terminology

Table 30 lists the terms used to describe different aspects of the maintenance transaction. This table clarifies terms specific to this appendix. Do not assume that this is commonly known or used terminology.

Table 30. Maintenance Transaction Terminology

Term	Description
Maintenance offset	The logical offset of a maintenance transaction. The maintenance offset closely resembles the way 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. In this application note, it is described as maintenance offset 0x18. The RapidIO specification describes the destination operations CAR as offset 0x18, word 1. In this application note, it is described 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	A 21-bit field within the RapidIO maintenance packet. It 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 is 0x03. For accesses to the destination operations CAR (maintenance offset 0x1C), the configuration offset field in the RapidIO maintenance transaction is also 0x03.
Word pointer	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 is 0x0. For accesses to the destination operations CAR (maintenance offset 0x1C), the word pointer in the RapidIO maintenance transaction is 0x1.
CFG_OFFSET	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	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.2 Example 1: 4 Mbyte 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 Mbyte 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 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 Mbyte window use 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, can then use 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 uses the macro as follows.

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

A.3 Example 2: 4 Kbyte Maintenance Window

A 4 Kbyte maintenance window permits a transaction offset of 0x0–0xFFF. 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 use 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.

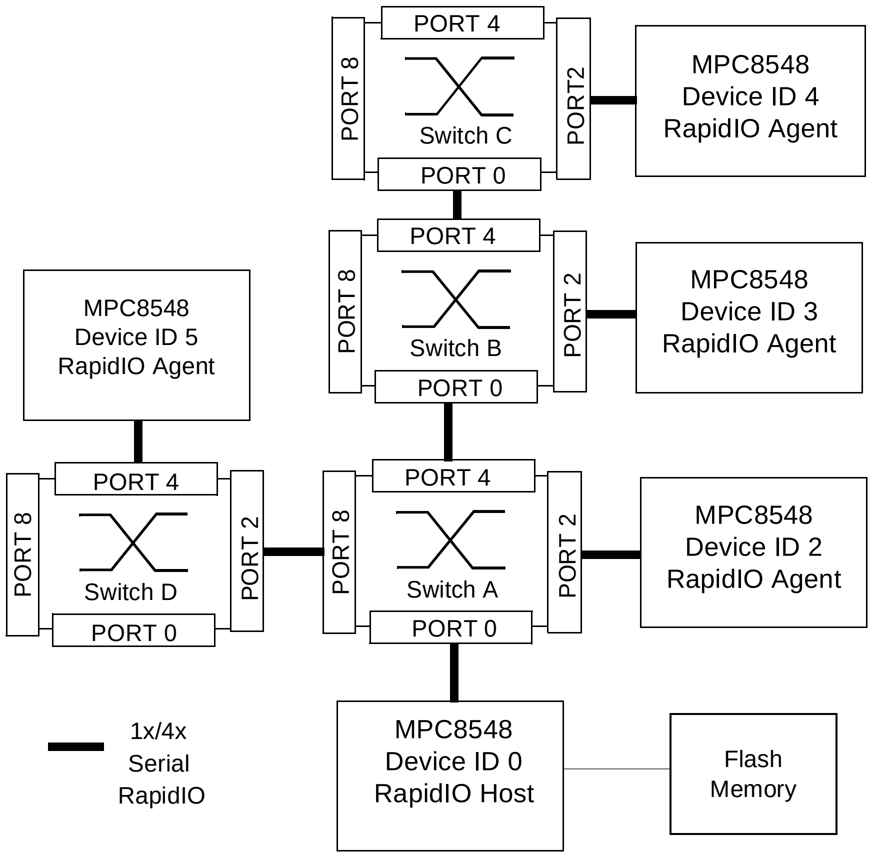
```
uint32 value_read;
MAINT_READ_4k (0x02, 0xFF, 0x68, value_read);
```

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.4 Maintenance Transactions Within Multi-Switch Systems

In this document, a system consists of no more than one RapidIO switch. For multi-switch systems, the way the maintenance window can be used to direct accesses to different switches must be considered. This section describes how the destination ID and hopcount determine which switch is accessed.



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

In the example here, the routing tables on all switches are set up to direct transactions to the correct RapidIO endpoints. Table 31 shows the resulting switch for a given hopcount and destination ID. If larger hopcounts are used, the switches are bypassed and the maintenance transactions are directed to the endpoints.

Table 31. 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

NOTES:

NOTES:

NOTES:

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations not listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GMBH
Technical Information Center
Schatzbogen 7
81829 München, 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.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
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

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Order No.: AN2932
Rev. 0
12/2005