WIND RIVER

Wind River® Simics®
RapidIO

TECHNOLOGY GUIDE

4.6

Revision 4081
Date 2012-11-16

Corporate Headquarters
Wind River
500 Wind River Way
Alameda, CA 94501-1153
U.S.A.

Toll free (U.S.A.): 800-545-WIND
Telephone: 510-748-4100
Facsimile: 510-749-2010

For additional contact information, see the Wind River Web site:
`www.windriver.com`
For information on how to contact Customer Support, see:
`www.windriver.com/support`

Contents

Chapter 1

Overview

1.1 Introduction

RapidIO is a packet-switched system level interconnect designed for tightly coupled devices.

Note: The RapidIO support in Simics is not part of the standard Simics product and may require additional licenses.

1.2 RapidIO in Simics

In Simics, RapidIO devices are expected to implement the `rapidio_v3`, `rapidio_v4` or `rapidio_v5` interface. They cover the same set of RapidIO operations, but `rapidio_v4` and `rapidio_v5` can handle latency in read operations. `rapidio_v5` also supports data streaming and transport class.

The Simics interfaces are defined on the transactional level, and does not model the low-level packets. Currently it is not possible to model failing transactions. The complete API documentation can be found in section 5.1, *Interfaces*.

If the device has more than one connection, the device should define one *port* for each connection. The device should also have an attribute that points to the connected device, to permit easy configuration.

To connect more than two devices to the same bus, a RapidIO switch is needed. As the RapidIO switches are not covered by the standard, there are no generic RapidIO switch. There is however a *Tundra Tsi500* modeled, which has four ports. Larger networks can be created by cascading multiple switches together.

There is also a simple dummy rapidio device that can be used for access logging.

Chapter 2

Conformance to RapidIO Standard

The latest RapidIO specification can be found at the RapidIO Trade Association, `www.rapidio.org/specs/current/`. The specifications are organized in *Logical*, *Transport*, and *Physical Specifications*, as seen in Figure 2.1. The Simics abstraction layer is on Logical Specifications, so everything below that is hidden. Simics supports a subset of version 1.2, 1.3, 2.0 and 2.1 of the standard, lacking support for Data Streaming (see Section 2.5). The Simics interfaces also lacks support for error injection. In the following sections, the mapping between each RapidIO operation and Simics interface function calls is described in detail.

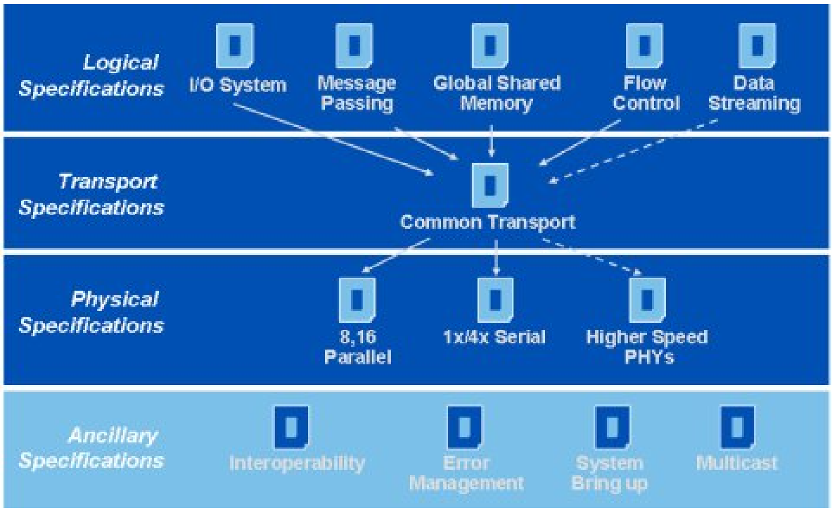

Figure 2.1: RapidIO Specification Hierarchy

2.1 I/O System

Part 1 of the specification describes I/O operations. There are a number different transaction types.

Read Operations

Consist of a NREAD type 2 packet followed by a DONE type 13 packet. Corresponds in the rapidio_v5 interface to a transaction_request(op=RapidIO_Read) followed by a transaction_response() call back

Write, Streaming Write, and Write-With-Response Operations

Consist of a NWRITE, SWRITE or NWRITE_R type 5 or type 6 packet, optionally followed by a DONE type 13 packet. In Simics, all writes are done with a call to transaction_request(op=RapidIO_Write). In Simics, the write will always succeed, but the interface require a transaction_response() callback with the same id and a dummy data packet of the same length as in the request.

Atomic (Read-Modify-Write) Operations

Consist of an ATOMIC type 5 packet, followed by a DONE type 13 packet. In Simics, it corresponds to a transaction_request(op=RapidIO_*op*) call, where *op* is either Increment, Decrement, Set, Clear, Test_and_Swap, Compare_and_Swap or Swap. The receiver responds with a transaction_response() call.

Maintenance Operations

Consist of a MAINTENANCE type 8 packet, followed by a MAINTENANCE type 8 reply packet. In Simics, CSR register reads are done by a read_register_request()/read_register_responce() pair. Writes are performed with write_register() call without any response (it will always succeed). The third type of maintenance operations, port-writes, are done with the port_write() function.

2.2 Message Passing

Part 2 of the specification describes the Message Passing extension of RapidIO.

Doorbell Operations

Typically used for interrupts. Consist of a DOORBELL type 10 packet followed by a DONE type 13 packet. Corresponds in the rapidio_v5 interface to a doorbell() call, without any reply (it always succeeds).

Data Message Operations

A message can be split up on up to sixteen MESSAGE type 11 packets followed by DONE type 13 packets. In Simics, the whole message is transmitted in one piece via a call to the deliver_message() interface function, and no reply is needed.

2.3 Shared Memory

Part 5 of the standard describes cache coherency protocols. It consists of a number of requests of type 1, 2 and 5 packets. They are not modeled in the rapidio_v5 interface. Systems with shared memory benefit from mapping the shared memory-space directly into the physical memory space of each processor instead.

The requests are mentioned here for completeness: READ_OWNER, READ_TO_OWN_OWNER, IO_READ_OWNER, READ_HOME, READ_TO_OWN_HOME, IO_READ_HOME, DKILL_HOME, IKILL_HOME, TLBIE, TLBSYNC, IREAD_HOME, FLUSH, IKILL_SHARER, DKILL_SHARER and CASTOUT. The responses of type 13 can be DONE, DATA_ONLY, NOT_OWNER, RETRY, INTERVENTION, DONE_INTERVENTION or ERROR.

2.4 Flow Control

Part 9 of the specification describes flow control, carried out by XON/XOFF type 7 packets. These will only be added to the Simics functional interface if the need arises from customers.

To be able to calculate the *FlowID* of a RapidIO packet, the priority can be extracted from the transport_class parameter of each interface function. Priority fields are passed in the physical layer, see specification part 4 and 6.

2.5 Data Streaming

Part 10 describes the possibility to setup a virtual stream between two RapidIO endpoints. Sending is done with DATA STREAMING type 9 packets. Data streaming is only supported by the rapidio_v5 interface, using the stream_data function.

Chapter 3

Simulation Models

Simics implements for the following RapidIO-related classes. For more detailed information about them, see to chapter 6.

3.1 Simple RapidIO Device

Dummy RapidIO device.

3.2 MPC8548 RapidIO Port

The mpc8548-rapidio device implements the functionality of the RapidIO Controller integrated on the MPC8548.

3.3 Tundra Tsi500 RapidIO Switch

Tundra Tsi500 4-port RapidIO switch. Supports 8-bit addressing.

Chapter 4

Programming Guide

4.1 First steps

4.1.1 Introduction

In this step-by-step guide we will start an MPC8548CDS board, and create a new RapidIO device to use on the board.

4.1.2 Create Workspace

First, create a workspace for your development if you have not done so yet. Workspaces are described in the *Model Builder User's Guide*.

```
joe@computer: ~$ [simics]/bin/workspace-setup ~/simics-workspace
Setting up Simics workspace directory: /home/joe/simics-workspace
[..]
joe@computer: ~$ cd simics-workspace
joe@computer: simics-workspace$
```

Try to start and boot Linux on the MPC8548CDS board.

```
joe@computer: simics-workspace$ ./simics targets/mpc8548cds/mpc8548cds-rapidio.simics
[...]
simics> continue
```

You should arrive to the shell prompt after a while. Check that there are no RapidIO devices connected.

```
# ls /sys/bus/rapidio/devices
#
```

You can now quit this Simics session.

4.1.3 Create a DML device

Start with a DML skeleton.

```
joe@computer: simics-workspace$ [simics]/bin/workspace-setup --dml-device my_rio_dev
[..]
joe@computer: simics-workspace$ make
[..]
joe@computer: simics-workspace$
```

The skeleton should compile cleanly. Now, open `modules/my_rio_dev/my_rio_dev.dml` in your favorite editor. Insert the following lines somewhere in the file:

```
import "rapidio-device.dml";

bank regs {
    parameter byte_order = "big-endian";
    is arch_io_registers;
    is arch_message_registers;
    is arch_transport_registers;
}
```

Save the file and rerun "make".

4.1.4 Connect a device the to machine

Launch the mpc8548cds configuration again but pass a command line parameter to set it as master. After that, create an instance of the **my_rio_dev** class you just built, and connect it using the "peer" attributes.

```
joe@computer: simics-workspace$ ./simics
[...]
simics> $kernel_cmdline = "riohdid=0"
simics> run-command-file targets/mpc8548cds/mpc8548cds-rapidio.simics
simics> @SIM_create_object("my_rio_dev", "riodev0", [])
simics> riodev0->peer = mpc8548cds.soc.rapidio
simics> mpc8548cds.soc.rapidio->peer = riodev0
simics> continue
```

You should arrive to the shell prompt after a while. Check that Linux found your RapidIO device.

```
# ls /sys/bus/rapidio/devices
00:e:ffff
#
```

4.2 RapidIO devices in DML

4.2.1 Overview

Note: This chapter assumes a basic knowledge of the RapidIO standard, how RapidIO is modeled in Simics and how to program in DML. Refer to the *Model Builder User's Guide* for Simics modeling information.

Creating a minimal RapidIO device is very easy: you just need to import the file `rapidio-device.dml` in your device. It will define the following:

- A `peer` *connect* object to let the device be connected to another device.
- Implement `rapidio_v3` interface with default functions that log things.
- Several templates to add RapidIO capabilities to the device with minimal effort.

4.2.2 register bank templates

All RapidIO device should at least implement the I/O register set. Names and offsets for the I/O registers are defined in the *arch_io_registers* template.

Other optional register sets are *arch_message_registers*, *arch_transport_registers* and more. A typical device will look like this:

```
bank regs {
    parameter byte_order = "big-endian";
    is arch_io_registers;
    is arch_message_registers;
    is arch_transport_registers;

    register device_id     {
        parameter hard_reset_value = 0x0500000d;
    }
}
```

For the contents of all the predefined templates, see section 4.2.4.

4.2.3 Utility methods

`rapidio-device.dml` also contains the glue code for incoming and outgoing transactions. These are the functions that it defines for you:

$peer.deliver_message(uint16 destination, uint16 mbox, dbuffer_t *data)

Sends a message to the specified destination.

$peer.memory_operation(uint16 destination, physical_address_t addr, dbuffer_t *buf, rapidio_operati

Makes a memory operation on the destination.

$peer.doorbell(uint16 target, uint16 data)
Sends a doorbell message to the peer, with specified target and data.

$peer.read_register(uint16 target, uint8 hopcount, int reg_no) -> (uint32 data)
Reads a maintenance register at the target. The target device is identified with an (ID, hopcount) pair.

$peer.write_register(uint16 target, uint8 hopcount, int reg_no, uint32 data)
Writes a maintenance register at the target.

These functions are you supposed to overload with something useful:

method get_device_id() -> (uint16 src_id)
Should return the device ID to use for outgoing requests.

method on_memory_operation(physical_address_t addr, dbuffer_t ∗buf, rapidio_operation_t op)

This function is called when an incoming memory operation occurs.

method on_doorbell(uint16 target, uint16 source, uint16 data)
This function is called when an incoming doorbell transaction occurs.

method on_deliver_message(uint16 mbox, dbuffer_t ∗data)
This function is called when an incoming message transaction occurs.

4.2.4 Full register templates

Here is a full listing of the templates defined.

```
template arch_io_registers {
    /* should be implemented by all */
    parameter io_regbase default 0;

    /* Capabilities registers */
    register device_id         size 4 @ $io_regbase + 0x00 is (arch_read_only)
        "Device identity capability register";
    register device_info       size 4 @ $io_regbase + 0x04 is (arch_read_only)
        "Device information capability register";
    register assembly_id       size 4 @ $io_regbase + 0x08
        "Assembly identity capability register";
    register assembly_info     size 4 @ $io_regbase + 0x0C
        "Assembly information capability register" {
        field assy_rev [0:15]  "Assembly revision level";
        field ef_ptr  [16:31]  "Pointer to first ext.feat.";
    }
    register pe_features       size 4 @ $io_regbase + 0x10 is (arch_read_only)
        "Processing element features capability register";
    register switch_info       size 4 @ $io_regbase + 0x14 is (arch_read_only)
```

```
        "Switch port information capability register";
}

template arch_message_registers {
    /* Valid for transport endpoints */
    register src_operations    size 4 @ $io_regbase + 0x18 is (arch_read_only)
        "Source operations capability register";
    register dst_operations    size 4 @ $io_regbase + 0x1C is (arch_read_only)
        "Destination operations capability register";
    register write_port_status size 4 @ $io_regbase + 0x44 is (arch_read_only)
        "Port-write and doorbell command and status register";
    register pe_ll_status      size 4 @ $io_regbase + 0x4C
        "Processing element logical layer control command and status register";
    register base1_status_hi   size 4 @ $io_regbase + 0x58 is (arch_read_write)
        "Local configuration space base address 1 command and status register";
    register base1_status      size 4 @ $io_regbase + 0x5C is (arch_read_write)
        "Local configuration space base address 1 command and status register";
}

template arch_transport_registers {
    /* Valid for transport endpoints */
    parameter trans_regbase default 0;
    register base_device_id   size 4 @ $trans_regbase + 0x60 "Base Device ID";
    register host_base_device_id   size 4 @ $trans_regbase + 0x68 "Host Base Device I
        parameter hard_reset_value = 0xffff;
        method write(value) {
            if (0) {
                /* This expression generates code that compiles incorrectly
                   on gcc 3.2 and 3.3 */
                value &= 0xffff;
            } else {
                value[31:16, le] = 0;
            }
            if ($this == 0xffff && value != 0xffff) {
                log "info", 3: "Device id %d locked me.", value;
                $this = value;
            } else if ($this != 0xffff && $this == value) {
                log "info", 3: "Device id %d unlocked me.", value;
                $this = 0xffff;
            }
        }
    }
    register component_tag   size 4 @ $trans_regbase + 0x6c is (arch_read_write)
        "Component Tag";
    register conf_dest_id    size 4 @ $trans_regbase + 0x70
```

```
        "Standard Route Configuration Destination ID Select";
    register conf_output_port size 4 @ $trans_regbase + 0x74
        "Standard Route Configuration Port Select";
    register default_output_port size 4 @ $trans_regbase + 0x78
        "Standard Route Default Port";
}

template arch_port_maintenance_registers {
    parameter switch_regbase default 0x100;
    register port_block_header size 4 @ $switch_regbase + 0x00 {
        field ef_ptr [0:15] is (read_only);
        field ef_id  [16:31] is (read_only);
    }
    register port_link_timeout   size 4 @ $switch_regbase + 0x20 {
        parameter hard_reset_value = 0xffffff00;
    }
    register port_general_control size 4 @ $switch_regbase + 0x3c;
}

template arch_per_port_maintenance_registers {
    parameter port_no default $i;
    register link_maintenance_request size 4
             @ $switch_regbase + 0x40 + $port_no * 0x20;
    register link_maintenance_response size 4
             @ $switch_regbase + 0x44 + $port_no * 0x20;
    register port_local_ackid_status size 4
             @ $switch_regbase + 0x48 + $port_no * 0x20;
    register port_error_and_status size 4
             @ $switch_regbase + 0x58 + $port_no * 0x20 {
        is write_1_clears;
    }
    register port_control size 4
             @ $switch_regbase + 0x5c + $port_no * 0x20;
}

template arch_error_management_registers {
    parameter error_regbase default undefined;
    register error_block_header size 4 @ $error_regbase + 0x00 {
        field ef_ptr [0:15] is (read_only);
        field ef_id  [16:31] is (read_only);
    }
    register layer_error_detect       size 4 @ $error_regbase + 0x08;
    register layer_error_enable       size 4 @ $error_regbase + 0x0c;
    register layer_capture_address_hi size 4 @ $error_regbase + 0x10;
    register layer_capture_address    size 4 @ $error_regbase + 0x14;
    register layer_capture_device_id  size 4 @ $error_regbase + 0x18;
    register layer_capture_control    size 4 @ $error_regbase + 0x1c;
    register portwrite_target_id      size 4 @ $error_regbase + 0x28;
    register packet_ttl               size 4 @ $error_regbase + 0x2c;
}

template arch_per_port_error_management_registers {
    parameter port_no default $i;
    register port_error_detect size 4
             @ $error_regbase + 0x40 + $port_no * 0x40;
    register port_error_rate_enable size 4
             @ $error_regbase + 0x44 + $port_no * 0x40;
    register port_capture_attributes size 4
             @ $error_regbase + 0x48 + $port_no * 0x40;
    register port_capture_symbol size 4
             @ $error_regbase + 0x4c + $port_no * 0x40;
    register port_capture_packet_1 size 4
             @ $error_regbase + 0x50 + $port_no * 0x40;
    register port_capture_packet_2 size 4
             @ $error_regbase + 0x54 + $port_no * 0x40;
    register port_capture_packet_3 size 4
             @ $error_regbase + 0x58 + $port_no * 0x40;
    register port_error_rate size 4
             @ $error_regbase + 0x68 + $port_no * 0x40;
    register port_error_rate_threshold size 4
             @ $error_regbase + 0x6c + $port_no * 0x40;
}
```

Chapter 5

RapidIO API

Below is a description of the RapidIO interface used by devices in Simics. For documentation of the complete Simics API, refer to the *Simics Reference Manual*.

5.1 Interfaces

5.1.1 rapidio_v3

Implemented By

mpc8548-rapidio, rapidio_simple_device, tsi500

Description

Execution Context

Instruction Context for all methods.

5.1.2 rapidio_v4

Implemented By

mpc8548-rapidio

Description

Execution Context

Instruction Context for all methods.

5.1.3 rapidio_v5

Implemented By

mpc8548-rapidio

Description

Execution Context

Instruction Context for all methods.

Chapter 6

Class Details

Following is a list of all configuration classes used to implement RapidIO in Simics. For documentation of other classes, refer to the *Simics Reference Manual*.

6.1 Classes

6.1.1 rapidio_simple_device

Provided By
rapidio-simple-device

Interfaces Implemented
conf_object, io_memory, log_object, rapidio_v3

Ports
regs (int_register, io_memory), trigger_dma (signal)

Description
Dummy RapidIO device.

Attributes

peer
Optional attribute; **read/write** access; type: `[os]`, `object`, or `nil`. The connected RapidIO switch or device

Command List

Commands defined by interface conf_object
break-hap, get-attribute-list, get-interface-list, get-interface-port-list, list-attributes, list-interfaces, log, log-group, log-level, log-size, log-type, trace-hap, unbreak-hap, untrace-hap, wait-for-log

6.1.2 mpc8548-rapidio

Provided By
MPC8548-devices

Interfaces Implemented
conf_object, frequency_listener, io_memory, log_object, rapidio_v3, rapidio_v4, rapidio_v5

Ports
HRESET (signal), SRESET (signal), blocked_ccsr (int_register, io_memory), outbound0 (int_register, io_memory), outbound1 (int_register, io_memory), outbound2 (int_register, io_memory), outbound3 (int_register, io_memory), outbound4 (int_register, io_memory), outbound5 (int_register, io_memory), outbound6 (int_register, io_memory), outbound7 (int_register, io_memory), outbound8 (int_register, io_memory), regs (int_register, io_memory)

Description
The mpc8548-rapidio device implements the functionality of the RapidIO Controller integrated on the MPC8548.

Attributes

ccb_frequency
Optional attribute; **read/write** access; type: `[os]`, `object`, or `nil`. CCB (platform frequency) dispatcher

ccsr_space
Required attribute; **read/write** access; type: `[os]` or `object`. Reference to the internal register memory space.

doorbell_irq_target
Required attribute; **read/write** access; type: `[os]` or `object`. Interrupt target for doorbell inbound interrupts

have_special_local_memory
Pseudo attribute; **read-only** access; type: `boolean`. True if this SRIO controller has special treatment of target ID 0xf for inbound windows.

inbound_irq_target
Required attribute; **read/write** access; type: `[o|[os]{2}]`. Interrupt target for inbound interrupts

inbound_space
Required attribute; **read/write** access; type: `[os]` or `object`. A separate memory space to store the inbound mappings

info
Required attribute; **read/write** access; type: `[os]` or `object`. SoC info object

outbound_doorbell_irq_target

Required attribute; **read/write** access; type: `[os]` or `object`. Interrupt target for doorbell outbound interrupts

outbound_irq_target

Required attribute; **read/write** access; type: `[o|[os]{2}]`. Interrupt target for outbound interrupts

outbound_space

Required attribute; **read/write** access; type: `[os]` or `object`. Physical memory space mapping for system.

outstanding_requests

Optional attribute; **read/write** access; type: `[[ioii]|[ioi]*]`. Map from ID of the outstanding request to (cpu, cookie, len).

peer

Optional attribute; **read/write** access; type: `[os]`, `object`, or `nil`. The connected RapidIO switch or device

phys_space

Required attribute; **read/write** access; type: `[os]` or `object`. Physical memory space mapping for system.

port_irq_target

Required attribute; **read/write** access; type: `[os]` or `object`. Interrupt target for inbound port write/error interrupts

regs_AACR

Optional attribute; **read/write** access; type: `integer`. Accept-All configuration Register

regs_SLCSR

Optional attribute; **read/write** access; type: `integer`. Serial Link Status Register

Command List

Commands defined by interface conf_object

break-hap, get-attribute-list, get-interface-list, get-interface-port-list, list-attributes, list-interfaces, log, log-group, log-level, log-size, log-type, trace-hap, unbreak-hap, untrace-hap, wait-for-log

Commands

info	print information about the device
status	print status of the device

Command Descriptions

<mpc8548-rapidio>.info

Synopsis
<mpc8548-rapidio>.info

Description
Print detailed information about the configuration of the device.

<mpc8548-rapidio>.status

Synopsis
<mpc8548-rapidio>.status

Description
Print detailed information about the current status of the device.

6.1.3 tsi500

Provided By
tsi500

Interfaces Implemented
conf_object, log_object

Ports
Port0 (rapidio_v3), Port1 (rapidio_v3), Port2 (rapidio_v3), Port3 (rapidio_v3), regs (int_register)

Description
Tundra Tsi500 4-port RapidIO switch. Supports 8-bit addressing.

Attributes

devices
Optional attribute; **read/write** access; type: `[o|[os]|n{4}]`. Receivers on the bus. Must implement the rapidio_v3 interface.

Command List

Commands defined by interface conf_object
break-hap, get-attribute-list, get-interface-list, get-interface-port-list, list-attributes, list-interfaces, log, log-group, log-level, log-size, log-type, trace-hap, unbreak-hap, untrace-hap, wait-for-log

Commands

info	print information about the device
status	print status of the device

Command Descriptions

<tsi500>.info

Synopsis
<tsi500>.info

Description
Print detailed information about the configuration of the device.

<tsi500>.status

Synopsis
<tsi500>.status

Description
Print detailed information about the current status of the device.

Index