

# LogiCORE™ IP Serial RapidIO v5.4

## *Getting Started Guide*

UG247 September 16, 2009





Xilinx is providing this product documentation, hereinafter “Information,” to you “AS IS” with no warranty of any kind, express or implied. Xilinx makes no representation that the Information, or any particular implementation thereof, is free from any claims of infringement. You are responsible for obtaining any rights you may require for any implementation based on the Information. All specifications are subject to change without notice. XILINX EXPRESSLY DISCLAIMS ANY WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE INFORMATION OR ANY IMPLEMENTATION BASED THEREON, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF INFRINGEMENT AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Except as stated herein, none of the Information may be copied, reproduced, distributed, republished, downloaded, displayed, posted, or transmitted in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx.

© 2006-2009 Xilinx, Inc. XILINX, the Xilinx logo, Virtex, Spartan, ISE, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.

## Revision History

The following table shows the revision history for this document.

Date	Version	Description
2/17/06	1.0	Initial Xilinx release.
7/13/06	2.0	Updated for IP1i minor release, ISE® software to v8.1i
11/15/06	3.0	Updated for IP3i minor release, added Virtex®-5 FPGA support
2/15/07	4.0	Added section on Initiator User Design, changed directory structure layout, updated tools for IP1 Jade release.
10/10/07	4.5	Updated for IP2 Jade Minor release.
3/24/08	5.0	Added support for Virtex-5 FXT FPGAs, updated tools for the ISE software v10.1 release.
06/19/08	6	Updated descriptions and images to include the new buffer, and added VHDL support.
9/18/08	6.5	Updated for ISE software Service Pack 3 release.
4/24/09	7.0	Added support for Virtex-6 devices, removed support for legacy devices. Updated tools to Xilinx ISE software v11.1.
6/24/09	7.5	Updated tools to Xilinx ISE v11.2 software.
9/16/09	8.0	Updated core to v5.4 and Xilinx ISE to v11.3.

# Table of Contents

---

<b>Schedule of Figures</b> .....	5
----------------------------------	---

## **Preface: About This Guide**

<b>Contents</b> .....	7
<b>Conventions</b> .....	7
Typographical.....	8
Online Document.....	9

## **Chapter 1: Introduction**

<b>System Requirements</b> .....	11
<b>About the Core</b> .....	11
<b>Recommended Design Experience</b> .....	12
<b>Additional Core Resources</b> .....	12
<b>Technical Support</b> .....	12
<b>Feedback</b> .....	12
RapidIO Cores .....	12
Document .....	13

## **Chapter 2: Licensing the Cores**

<b>Before you Begin</b> .....	15
<b>License Options</b> .....	15
Simulation Only .....	15
Full System Hardware Evaluation .....	16
Full .....	16
<b>Obtaining Your License Key</b> .....	16
Simulation License .....	16
Full System Hardware Evaluation License .....	16
Obtaining a Full License Key .....	16
Installing Your License File .....	17

## **Chapter 3: Customizing and Generating the Cores**

<b>Serial RapidIO Graphical User Interface</b> .....	19
Main Screen.....	20
Serial Physical Layer Configuration .....	22
Logical Layer Configuration .....	24
Buffer Reference Design Configuration .....	26
Logical Layer CAR Configuration.....	28
Logical Layer CAR Configuration.....	31
Logical Layer CSR Configuration .....	34
Serial Physical Layer CAR/CSR Configuration .....	36
Serial Physical Layer CSR Configuration .....	38
<b>Output Generation</b> .....	40

## Chapter 4: Quick Start Example Design

Overview .....	41
Generating the Cores .....	42
Implementing the Example Design .....	44
Running the Simulation .....	44
Setting up the Simulation .....	44
Functional Simulation .....	45
Creating an ISE Software Project .....	45

## Chapter 5: Detailed Example Design

Directory Structure and File Descriptions .....	47
CORE Generator Software Project .....	48
Serial RapidIO Endpoint .....	48
Directory and File Contents .....	49
Project Directory .....	49
<project directory> .....	49
Serial RapidIO Endpoint Solution .....	50
<project directory>/<srio_component name> .....	50
<srio_component name>/doc .....	51
<srio_component_name>/example_design .....	51
<srio_component_name>/example_design/chipscope .....	53
<srio_component_name>/example_design/reg_manager .....	54
<srio_component_name>/example_design/user .....	55
<srio_component_name>/implement .....	57
<srio_component_name>/implement/results .....	57
<srio_component_name>/netlists .....	57
<srio_component_name>/simulation .....	58
<srio_component_name>/simulation/functional .....	58
Serial RapidIO Endpoint Example Design Description .....	59
Initiator User Design .....	60
Target User Design .....	67
Simulation Host .....	68

# Schedule of Figures

---

## Chapter 1: Introduction

## Chapter 2: Licensing the Cores

## Chapter 3: Customizing and Generating the Cores

<i>Figure 3-1: Serial RapidIO Configuration Screen</i> .....	20
<i>Figure 3-2: Physical Layer Configuration Screen</i> .....	22
<i>Figure 3-3: Logical Layer Configuration Screen</i> .....	24
<i>Figure 3-4: Buffer Design Configuration Screen</i> .....	26
<i>Figure 3-5: Logical Layer CAR Configuration (Screen 1)</i> .....	29
<i>Figure 3-6: Logical Layer CAR Configuration (Screen 2)</i> .....	31
<i>Figure 3-7: Logical Layer CSR Configuration Screen</i> .....	34
<i>Figure 3-8: Serial Physical Layer CAR/CSR Configuration Screen</i> .....	36
<i>Figure 3-9: Serial Physical Layer CSR Configuration Screen</i> .....	38

## Chapter 4: Quick Start Example Design

<i>Figure 4-1: The RapidIO Endpoint Example Design Configuration</i> .....	42
--	----

## Chapter 5: Detailed Example Design

<i>Figure 5-1: RapidIO Endpoint Core Example Design</i> .....	59
<i>Figure 5-2: Initiator User Design</i> .....	60
<i>Figure 5-3: RapidIO Example Target Design Block Diagram</i> .....	67
<i>Figure 5-4: RapidIO Simulation Host Block Diagram</i> .....	68



# About This Guide

---

This guide contains information for generating the Xilinx LogiCORE™ IP Serial RapidIO Endpoint solution, which includes the Serial RapidIO Physical Layer, Buffer, and RapidIO Logical Layer cores. It also provides detailed information for customizing and simulating these RapidIO cores and for running the design files through implementation using Xilinx tools.

## Contents

This guide contains the following chapters:

- [Preface, “About this Guide,”](#) introduces the organization and purpose of the design guide and describes the conventions used in this document.
- [Chapter 1, “Introduction,”](#) provides information about the system requirements for running the cores recommended design experience, and references to related material including the ways to contact Xilinx for obtaining technical support and for providing feedback.
- [Chapter 2, “Licensing the Cores,”](#) provides licensing information for the Serial RapidIO Endpoint solution.
- [Chapter 3, “Customizing and Generating the Cores,”](#) describes the GUI options used to generate and customize the cores.
- [Chapter 4, “Quick Start Example Design,”](#) describes how to quickly get started using the Serial RapidIO Endpoint solution.
- [Chapter 5, “Detailed Example Design,”](#) provides a detailed explanation about the example design. It also provides information about the various output files, directory structure, purpose and contents of the implementation scripts, operation of the example target design, and the demonstration test bench.

## Conventions

This document uses the following conventions. An example illustrates each convention.

## Typographical

The following typographical conventions are used in this document

Convention	Meaning or Use	Example
Courier font	Messages, prompts, and program files that the system displays	speed grade: - 100
<b>Courier bold</b>	Literal commands you enter in a syntactical statement	<b>ngdbuild</b> design_name
<nnnnnnnnnn>	Variables in a syntax statement for which you must supply values	<project_directory>/test
<i>Italic font</i>	References to other manuals	See the <i>User Guide</i> for details.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are <i>not</i> connected.
Dark Shading	Items that are not supported or reserved	This feature is not supported
Square brackets [ ]	An optional entry or parameter. However, in bus specifications, such as <b>bus[7:0]</b> , they are required.	<b>ngdbuild</b> [option_name] design_name
Braces { }	A list of items from which you must choose one or more	<b>lowpwr</b> = {on off}
Vertical bar	Separates items in a list of choices	<b>lowpwr</b> = {on off}
Vertical ellipsis . . .	Repetitive material that has been omitted	IOB #1: Name = QOUT' IOB #2: Name = CLKIN' . . .
Horizontal ellipsis ...	Omitted repetitive material	<b>allow block</b> block_name loc1 loc2 ... locn;
Notations	The prefix '0x' or the suffix 'h' indicate hexadecimal notation	A read of address 0x00112975 returned 45524943h.
	An '_n' means the signal is active low	usr_teof_n is active low.



## Online Document

The following conventions are used in this document:

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current document	See the section “ <a href="#">Additional Resources</a> ” for details. See “ <a href="#">Title Formats</a> ” in <a href="#">Chapter 1</a> for details.
<a href="#">Blue, underlined text</a>	Hyperlink to a website (URL)	Go to <a href="http://www.xilinx.com">www.xilinx.com</a> for the latest speed files.



## Introduction

---

The complete LogiCORE™ IP RapidIO Endpoint solution consists of the Serial RapidIO Physical Layer, Buffer, and the RapidIO Logical Layer stitched together with reference designs to create a full example endpoint solution.

- The 1 lane (1x) and 4 lane (4x) Serial Physical Layer is incorporated into the Serial RapidIO Physical Layer core.
- The Logical (I/O) and Transport Layers are combined into the RapidIO Logical Layer core.

The RapidIO Endpoint example design provides the remaining logic to create a RapidIO Endpoint solution that includes a register manager reference design and an example target design. The RapidIO Endpoint example design also provides implementation and simulation scripts for the RapidIO Endpoint solution.

## System Requirements

### Windows

- Windows XP Professional 32-bit/64-bit
- Windows Vista Business 32-bit/64-bit

### Linux

- Red Hat Enterprise Linux WS v4.0 32-bit/64-bit
- Red Hat Enterprise Desktop v5.0 32-bit/64-bit (with Workstation Option)
- SUSE Linux Enterprise (SLE) desktop and server v10.1 32-bit/64-bit

### Software

- ISE® software v11.3 with applicable service pack

Check the release notes for the required service pack; ISE software service packs can be downloaded from [www.xilinx.com/xlnx/xil\\_sw\\_updates\\_home.jsp?update=sp](http://www.xilinx.com/xlnx/xil_sw_updates_home.jsp?update=sp).

## About the Core

The Serial RapidIO Endpoint solution, including the Serial RapidIO Physical Layer, Buffer, and the RapidIO Logical Layer cores are Xilinx CORE Generator™ software, included in the latest IP update. For detailed information about these cores, go to [www.xilinx.com/rapidio](http://www.xilinx.com/rapidio).

## Recommended Design Experience

The fully verified RapidIO Endpoint solution allows engineering focus on the unique user-application functions of a RapidIO design. The RapidIO Interconnect technology is a high-performance design that can be challenging to implement in any device technology.

Therefore, previous experience with building high-performance, pipelined FPGA designs using Xilinx implementation software and constraints file is recommended. The challenge to implement a complete RapidIO Endpoint design including user application functions varies, depending on the features and configuration of the application. Contact your local Xilinx representative for a closer review and estimation for your specific requirements.

## Additional Core Resources

For detailed information about and updates to the RapidIO Design Environment, Serial RapidIO Physical Layer, and the RapidIO Logical Layer cores see the following documents. These and other documents can be downloaded from the Xilinx RapidIO lounge:

[www.xilinx.com/rapidio](http://www.xilinx.com/rapidio)

- *Serial RapidIO Release Notes*
- *Serial RapidIO Data Sheet*
- *Serial RapidIO User Guide*

For updates to this document, see the *Serial RapidIO Getting Started Guide*, also available from [www.xilinx.com/rapidio](http://www.xilinx.com/rapidio).

## Technical Support

The fastest method for obtaining specific technical support for the RapidIO cores is through the [www.xilinx.com/support](http://www.xilinx.com/support) website. Questions are routed to a team of engineers with expertise in using the RapidIO cores.

Xilinx will provide technical support for use of this product as described in this guide, the and the *LogiCORE IP Serial RapidIO User Guide*. Xilinx cannot guarantee timing, functionality, or support of this product for designs that do not follow these guidelines.

## Feedback

Xilinx welcomes feedback on the RapidIO cores and documentation provided with the cores.

### RapidIO Cores

For comments or suggestions about the RapidIO cores, please submit a WebCase at [www.xilinx.com/support](http://www.xilinx.com/support), and provide the following information:

- Product name
- Core version number
- Brief explanation

## Document

If you have any comments about this document, please submit a WebCase at [www.xilinx.com/support](http://www.xilinx.com/support), and provide the following information:

- Document title
- Document number
- Page number(s) to which your comments refer
- Brief explanation

General suggestions for additions and improvements are also welcome.



# Licensing the Cores

---

The Serial RapidIO Solution consists of two Xilinx LogiCORE™ IP cores and an endpoint reference design. This chapter provides licensing instructions for the Serial RapidIO Physical Layer and the RapidIO Logical Layer cores. You must obtain the appropriate licenses before using the cores in your designs. The Serial RapidIO Physical Layer and RapidIO Logical Layer cores are provided under the terms of the [Xilinx LogiCORE Site License Agreement](#), which conforms to the terms of the [SignOnce](#) IP License standard defined by the Common License Consortium. Purchase of a Physical Layer core license includes licensing the Buffer as both are created when generating the Physical Layer.

Purchase of a core entitles you to technical support and access to updates for a period of one year. The endpoint example is a non-licensed support core that includes reference designs, as well as simulation and implementation scripts.

## Before you Begin

This chapter assumes you have installed the core using either the CORE Generator™ IP Software Update installer or by performing a manual installation after downloading the core from the web. For information about installing the core, see the RapidIO product page at [www.xilinx.com/rapidio](http://www.xilinx.com/rapidio).

## License Options

The Serial RapidIO Physical Layer and RapidIO Logical Layer cores provide three licensing options. After installing the required Xilinx ISE software and IP Service Packs, choose a license option. The endpoint example is a reference design, and is not a licensed core.

### Simulation Only

The Simulation Only Evaluation license is provided with the Xilinx CORE Generator tool. This license lets you assess core functionality with either the endpoint example design provided in the Serial RapidIO solution, or alongside your own design and demonstrates the various interfaces on the core in simulation. (Functional simulation is supported by a dynamically generated HDL structural model.)

## Full System Hardware Evaluation

The Full System Hardware Evaluation license is available at no cost and lets you fully integrate the core into an FPGA design, place-and-route the design, evaluate timing, and perform functional simulation of the RapidIO endpoint design using the RapidIO endpoint example design and demonstration test bench provided in the Serial RapidIO solution.

In addition, the license key lets you generate a bitstream from the placed and routed design, which can then be downloaded to a supported device and tested in hardware. The core can be tested in the target device for a limited time before timing out (ceasing to function), at which time it can be reactivated by reconfiguring the device.

## Full

The Full license key is available when you purchase the core and provides full access to all core functionality both in simulation and in hardware, including:

- Functional simulation support
- Full implementation support including place and route and bitstream generation
- Full functionality in the programmed device with no time outs

## Obtaining Your License Key

This section contains information about obtaining a simulation, full system hardware, and full license keys.

### Simulation License

No action is required to obtain the Simulation Only Evaluation license key; it is provided by default with the Xilinx CORE Generator software.

### Full System Hardware Evaluation License

To obtain a Full System Hardware Evaluation license, do the following:

1. Navigate to the product page for this core: [www.xilinx.com/rapidio](http://www.xilinx.com/rapidio)
2. Click Evaluate.
3. Follow the instructions to install the required Xilinx ISE® software and IP Service Packs.

### Obtaining a Full License Key

To obtain a Full license key, you must purchase a license for the core. After you purchase a license, a product entitlement is added to your Product Licensing Account on the Xilinx Product Download and Licensing site. The Product Licensing Account Administrator for your site will receive an email from Xilinx with instructions on how to access a Full license and a link to access the licensing site. You can obtain a full key through your account administrator, or your administrator can give you access so that you can generate your own keys.

Further details can be found at

[http://www.xilinx.com/products/ipcenter/ipaccess\\_fee.htm](http://www.xilinx.com/products/ipcenter/ipaccess_fee.htm).



## Installing Your License File

The Simulation Only Evaluation license key is provided with the Xilinx ISE software CORE Generator system and does not require installation of an additional license file. For the Full System Hardware Evaluation license and the Full license, an email will be sent to you containing instructions for installing your license file. Additional details about IP license key installation can be found in the ISE Design Suite Installation, Licensing and Release Notes document.



# Customizing and Generating the Cores

---

The LogiCORE™ IP Serial RapidIO Design Solution consists of four components, the Serial RapidIO Physical Layer core, Buffer core, the RapidIO Logical Layer core, and the Endpoint example design. These components are generated through the Xilinx CORE Generator™ software using a graphical user interface (GUI).

The 1 lane (1x) and 4 lane (4x) Serial Physical Layer is incorporated into the Serial RapidIO Physical Layer core. The Logical (I/O) and Transport layers are combined into the RapidIO Logical Layer core. The RapidIO Endpoint example design provides the remaining logic to create a Serial RapidIO Endpoint solution that includes a register manager reference design and an example target design.

This chapter describes the GUI options used to generate and customize the cores. When you are generating the Serial RapidIO cores, they must all be created within the same CORE Generator software project directory to work with the Endpoint Example implementation and simulation scripts.

For assistance with starting and using the Xilinx CORE Generator software, see the documentation supplied with the Xilinx ISE® software, including the *Xilinx CORE Generator Online Help*.

This information is located at [toolbox.xilinx.com/docsan/xilinx10/books/manuals.pdf](http://toolbox.xilinx.com/docsan/xilinx10/books/manuals.pdf)

## Serial RapidIO Graphical User Interface

This section describes the Xilinx CORE Generator software configurations screens provided to configure the Serial RapidIO Endpoint design.

## Main Screen

Figure 3-1 shows the Serial RapidIO CORE Generator software main configuration screen. Descriptions of the GUI options on this screen are provided in the following text.

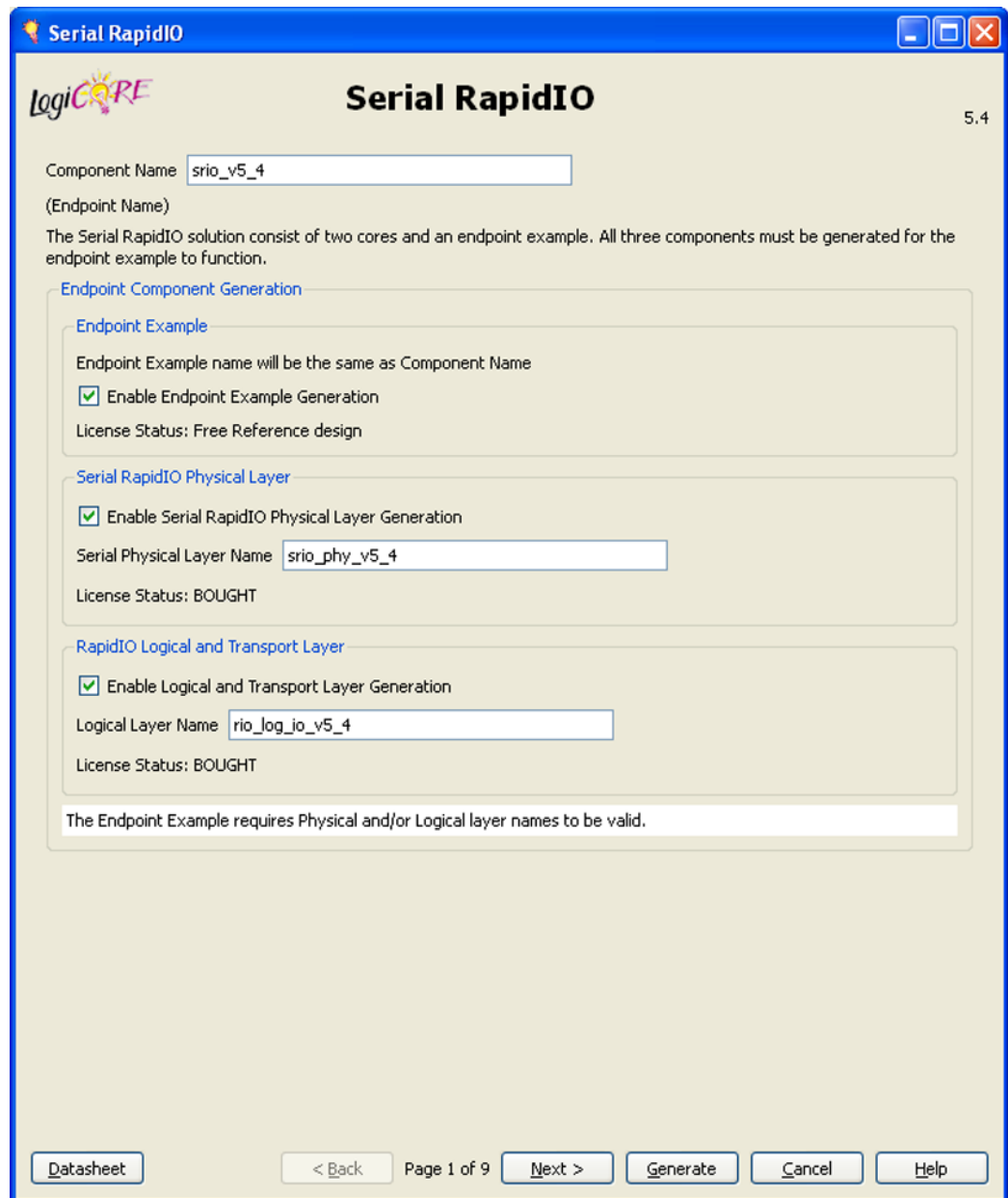


Figure 3-1: Serial RapidIO Configuration Screen

### Component Name

The component name is used as the name of the endpoint reference design as well as the base name of the output files generated for the core. Names must begin with a letter and must be composed from the following characters: a through z, 0 through 9, and "\_". The default is srio\_v5\_4.

## Enable Endpoint Example Generation

The Endpoint Example reference design is an option. Enabling this option provides the remaining logic to create a RapidIO Endpoint solution that includes a register manager reference design, an example target design, and implementation and simulation scripts. If the Endpoint Example is not generated, implementation and simulation scripts will not be generated.

## Enable Serial RapidIO Physical Layer Generation

The Physical Layer core generation is optional. Enabling generation provides a customized netlist for the Physical Layer core, and a netlist for a customized Buffer core. The Physical Layer includes an example transceiver instantiation and associated CORE Generator software files.

## Serial RapidIO Physical Layer Name

The Physical Layer name is used as the base name of the output files generated for the Physical Layer core. Names must begin with a letter and be composed from the following characters: a through z, 0 through 9, and "\_". The default is srio\_phy\_v5\_4.

## Enable Logical and Transport Layer Generation

The Logical Layer core generation is optional. Enabling generation provides a customized netlist for the Logical Layer core and associated CORE Generator software files. The Logical Layer is not meant as a stand-alone core. It is assumed a Physical Layer core will be generated as well.

## Logical Layer Name

The Logical Layer name is used as the base name of the output files generated for the logical layer core. Names must begin with a letter and be composed from the following characters: a through z, 0 through 9, and "\_". The default is rio\_log\_io\_v5\_4.

## Serial Physical Layer Configuration

Figure 3-2 shows the Serial Physical Layer Configuration screen. This screen is required when generating either the Endpoint Example Design or the Physical Layer core. The configuration options found on this screen are described in the following text.

**Serial RapidIO** 5.4

**Physical Configuration**

**System Configuration**

Total data throughput is a function of Baud rate and Port Width. Input Clock Frequency is determined from desired Baud Rate.

x4 Bandwidth	x1 Bandwidth	Per Lane Baud Rate	Input Clock Frequency
10 Gbps	2.5 Gbps	3.125 Gbaud	156.25 MHz
8 Gbps	2 Gbps	2.5 Gbaud	125 MHz
4 Gbps	1 Gbps	1.25 Gbaud	125 MHz

**Lanes Selection**

☒ x4 ☐ x1

Transfer Frequency: 3.125 Gbaud (per line)

Input Clock Frequency: 156.25 Mhz

**Implementation Options**

Configure the number of Link Requests to resend after the link time-out register expires without receiving a link response before going into the fail error state.

Additional Link Requests Before Fatal: 4

Enable to support the CRF bit for priority bumping.

☒ Support Critical Request Flow

[Datasheet](#) < Back Page 2 of 9 Next > Generate Cancel Help

Figure 3-2: Physical Layer Configuration Screen

## System Configuration

System configuration determines the ultimate port bandwidth and the required input clock frequency. See [Table 3-1](#).

**Table 3-1: Baud Rate to Input Clock Frequency**

x4 Bandwidth (Gbps)	x1 Bandwidth (Gbps)	Per Lane Baud Rate (Gbaud)	Input Clock Frequency (MHz)			
			Virtex®-6 <sup>a</sup>	Spartan®-6 <sup>b</sup>	Virtex-5 <sup>a</sup>	Virtex-4
10	2.5	3.125	156.25	N/A	156.25	312.5
8	2	2.5	125	125	125	250
4	1	1.25	125	125	125	250

- a. The Virtex-6 and Virtex-5 FPGA 3.125 cores may also be generated using a 125 MHz input clock frequency.
- b. Spartan-6 FPGAs currently only support single lane implementations.

## Lanes

Selecting x1 creates a one-lane core; selecting x4 creates a four-lane core. The default is x1 for Spartan-6 and x4 for all other families.

## Transfer Frequency

This sets the per-lane baud rate. Allowable settings are 1.25, 2.5, or 3.125 Gbps. The default transfer frequency is 3.125 Gbps. Spartan-6 devices do not support 3.125 Gbps transfer frequency, and uses a 2.5 Gbps default.

## Input Clock Frequency

**Note:** This option is not applicable for Virtex-4 FPGA designs

Select the input clock frequency. Allowable settings for 3.125 GHz line rates are 125 or 156.25 MHz. The default input clock frequency is 125 MHz.

## User Reference Clock

**Note:** This option is only applicable to Virtex-4 FPGA designs.

### Virtex-4 FPGA Designs

Select either REFCLK1 or REFCLK2 from the drop-down list to select the source of the RocketIO™ transceiver reference clock. See the [Virtex-4 FPGA RocketIO User Guide](#) to determine the appropriate clock to use. REFCLK1 is selected by default.

## Additional Link Requests Before Fatal

Use the drop-down list to select the number of link requests that are to be sent without receiving a link response prior to transiting to a fatal error state. Allowable options are 0 through 6 and Never Fatal. The default is zero. Xilinx recommends setting this value greater than zero for enhanced error recovery.

## Support Critical Request Flow

This option supports the CRF bit for priority bumping. Support Critical Request Flow is disabled by default.

## Logical Layer Configuration

Page 3, shown in [Figure 3-3](#), presents system level Implementation options which effect Logical layer functionality. Details for each option are included in this section.

**Serial RapidIO** 5.4

**Logical Layer**

**Implementation Options**

**Device IDs**

☒ Use 8 bit Device IDs  
☐ Use 16 bit Device IDs

Sets the Device IDs for this component.

Component Device ID:  (8-bit Hex)

**Doorbell and Messaging Support**

Enable support for doorbell and messaging packets. Parse packets for presentation on the Target Request interface, and build packets presented on the Initiator Request interface.

☒ Enable Doorbell and Message Packets Support

[Datasheet](#) [< Back](#) Page 3 of 9 [Next >](#) [Generate](#) [Cancel](#) [Help](#)

Figure 3-3: Logical Layer Configuration Screen



## Implementation Options

### Device IDs

Select either 8-bit or 16-bit device IDs. The core generates and consumes only those packets that match the selected device IDs. Use the 16-bit device IDs if the processing element supports large systems.

### Component Device ID

Sets the device ID for this component. Any 8 or 16-bit (depending on the selected user device ID size) hexadecimal value is allowed. The default setting is FF.

### Doorbell and Messaging Support

It is recommended Doorbell and Messaging support be included. Deselecting this option is meant to support legacy end-points which choose to build Doorbell and Message packets as undefined packet types.

## Buffer Reference Design Configuration

The Buffer design options shown in [Figure 3-4](#) provide size vs. bandwidth trade-offs. Descriptions for each option and associated cost are provided in this section.

**Serial RapidIO** 5.4

**Serial RapidIO**

**Buffer Design**

**Implementation Options**

**Buffer Configuration**

The depth of the buffer indicates how many packets will be stored. Selecting the smaller buffer sizes will reduce the resources used.

TX Buffer Depth

RX Buffer Depth

**Flow Control**

Indicates type of flow control used by the transmitter. By selecting Transmitter Controlled, the core will attempt to use transmitter controlled flow control, but switch to receiver controlled flow control if the link partner does not support transmitter controlled.

☒ Transmitter Controlled

☐ Receiver Controlled

Minimizes retry conditions through use of received buffer status and watermarks.

Packet Priority Watermarks are used to progressively limit the packet priorities that can be sent as the effective number of free buffers decreases in the link partner. If the free buffer count exceeds the watermark, packets of that priority and greater are transmitted.

Highest Priority Watermark  Range: 1..D

Medium Priority Watermark  Range: 2..E

Lowest Priority Watermark  Range: 3..F

[Datasheet](#) [< Back](#) Page 4 of 9 [Next >](#) [Generate](#) [Cancel](#) [Help](#)

Figure 3-4: Buffer Design Configuration Screen

## Implementation Options

### Buffer Configuration

You can customize the depth of the transmit and receive buffers to 8, 16, or 32. This number represents the amount of packets the buffer is capable of storing. Selecting the smaller buffer depths conserves resources, whereas maximum buffer depth yields maximum throughput.

The number of block RAM used is only affected by the depth of the buffer. Block RAM usage is listed in [Table 3-2](#) and is based on the selected Virtex FPGA device family.

**Table 3-2: Block RAM Usage by Device Family**

Buffer Depth	Virtex-6	Spartan-6	Virtex-5	Virtex-4
8	2	4	2	4
16	4	8	4	8
32	8	NA	8	NA

[Table 3-3](#) is a summary of LUT utilization for different buffer configurations. In this table, 0 represents the default configuration with an approximate LUT count of 727 for Virtex-5 devices, and 855 for Virtex-4 devices. For each parameter that is altered, more or less LUTs will be used. This number can then be added or subtracted from the approximate default value to get a rough estimate of the LUTs your configuration may use. However, this is only an estimate and should not be used as an precise value.

**Table 3-3: LUT Approximation Based on Buffer Configuration Parameters**

Parameter	Value	Virtex-6	Spartan-6	Virtex-5	Virtex-4
Flow Control	RX	-60	-51	-50	-81
	TX	0	0	0	0
TX Depth	8	-133	-73	-156	-76
	16	-75	0	-80	0
	32	0	NA	0	NA
RX Depth	8	-22	-10	-13	-14
	16	-1	0	-12	0
	32	0	NA	0	NA

## Flow Control

These options indicate the type of flow control to be used by the transmitter.

### Transmitter Controlled

Selecting this option causes the core to first attempt to use transmitter-controlled flow control, but switches to receiver-controlled if the link partner does not support it. Transmitter-controlled flow control minimizes retry conditions through the use of received buffer status and watermarks. Receiver-controlled flow control blindly transmits packets and uses the retry protocol.

## Receiver Controlled

Select this option for receiver-controlled flow control only. In this mode, packets are blindly transmitted and the retry protocol is used to control packet flow.

## Packet Priority Watermarks

Packet Priority Watermarks are used to progressively limit the packet priorities that can be sent as the effective number of free buffers decreases in the link partner. If the free buffer count exceeds the watermark, only packets of that priority and higher are transmitted. Be sure to set the watermark values below the max value of available buffers within the link partner to allow all packet priorities possibility of transmission.

Priority 3 packets will be sent until the effective number of receive buffers reaches zero.

### Highest Priority Watermark (WM2)

Used only when in Transmitter Controlled Flow Control Mode. This field establishes the minimum number of available buffer spaces required prior to sending High Priority (priority 2) packets. The watermarks must be constrained to:

$$0 < WM2 < WM1 < WM0$$

### Medium Priority Watermark (WM1)

Used only when in Transmitter Controlled Flow Control Mode. This field establishes the minimum number of available buffer spaces required prior to sending Medium Priority (priority 1) packets. The watermarks must be constrained to:

$$0 < WM2 < WM1 < WM0$$

### Smallest Priority Watermark (WM0)

Used only when in Transmitter Controlled Flow Control mode. This field establishes the minimum number of available buffer spaces required prior to sending Smallest Priority (priority 0) packets. The watermarks must be constrained to:

$$0 < WM2 < WM1 < WM0$$

## Logical Layer CAR Configuration

The Logical Layer Device Capabilities Registers (CARs) are detailed in [Figure 3-5](#). These registers are read only once implemented. Register values do not effect core behavior.

Serial RapidIO

LogiCORE
Serial RapidIO
5.4

Logical Layer

Device Capability registers (CARs)

Device Identity CAR (Offset 0x0)

Bit 0-15	0000	Device Identity
Bit 16-31	000E	Device Vendor Identity

Device Information CAR (Offset 0x4)

Bit 0-31	00000540	Device Revision Level
----------	----------	-----------------------

Assembly Identity CAR (Offset 0x8)

Bit 0-15	<input type="text" value="0000"/>	Assembly Identifier
Bit 16-31	<input type="text" value="0000"/>	Assembly Vendor Identifier

Assembly Information CAR (Offset 0xC)

Bit 0-15	<input type="text" value="0000"/>	Assembly Revision Level
Bit 16-31	0100	Extended Feature Pointer

Processing Element Features CAR (Offset 0x10)

☐ Bridge
☒ Memory
☐ Processor

Bit 0-3	0100	PE Functionality
Bit 27	0b0	Common Transport Large System Support
Bit 28	0b1	Extended Features Pointer Valid
Bit 29-31	0b001	Extended Addressing Support (Supports 34-bit addresses)

Datasheet

< Back

Page 5 of 9

Next >

Generate

Cancel

Help

Figure 3-5: Logical Layer CAR Configuration (Screen 1)

## Device Capability Registers (CARs)

### Device Identity CAR

#### Device Identity

Xilinx reserved field; currently not used.

#### Device Vendor Identity

RapidIO Trade Association assigned Xilinx vendor ID.

### Device Information CAR

#### Device Revision Level

Xilinx core revision.

- Bits 0:15 are currently reserved by Xilinx.
- Bits 16:23 represent the major revision number.
- Bits 24:27 represent the minor revision number.
- Bits 28:31 indicate the patch release.

Currently, the Device Revision Level is 0000\_0540h.

### Assembly Identity CAR

#### Assembly Identifier

Sets the type of assembly-CAR 0x08 AssyIdentity field. Any 16-bit hexadecimal value is allowed. The default setting is 0x0000. This does not affect core functionality and is stored in a configuration register.

#### Assembly Vendor Identifier

Sets the assembly vendor identity-CAR 0x08 AssyVendorIdentity field. Any 16-bit hexadecimal value is allowed. The default setting is 0x0000. This does not affect core functionality and is stored in a configuration register.

### Assembly Information CAR

#### Assembly Revision

Sets the revision for assembly-CAR 0x8 AssyRev field. Any 16-bit hexadecimal value is allowed. The default is 0x0000. This does not affect core functionality and is stored in a configuration register.

### Processing Element Features CAR

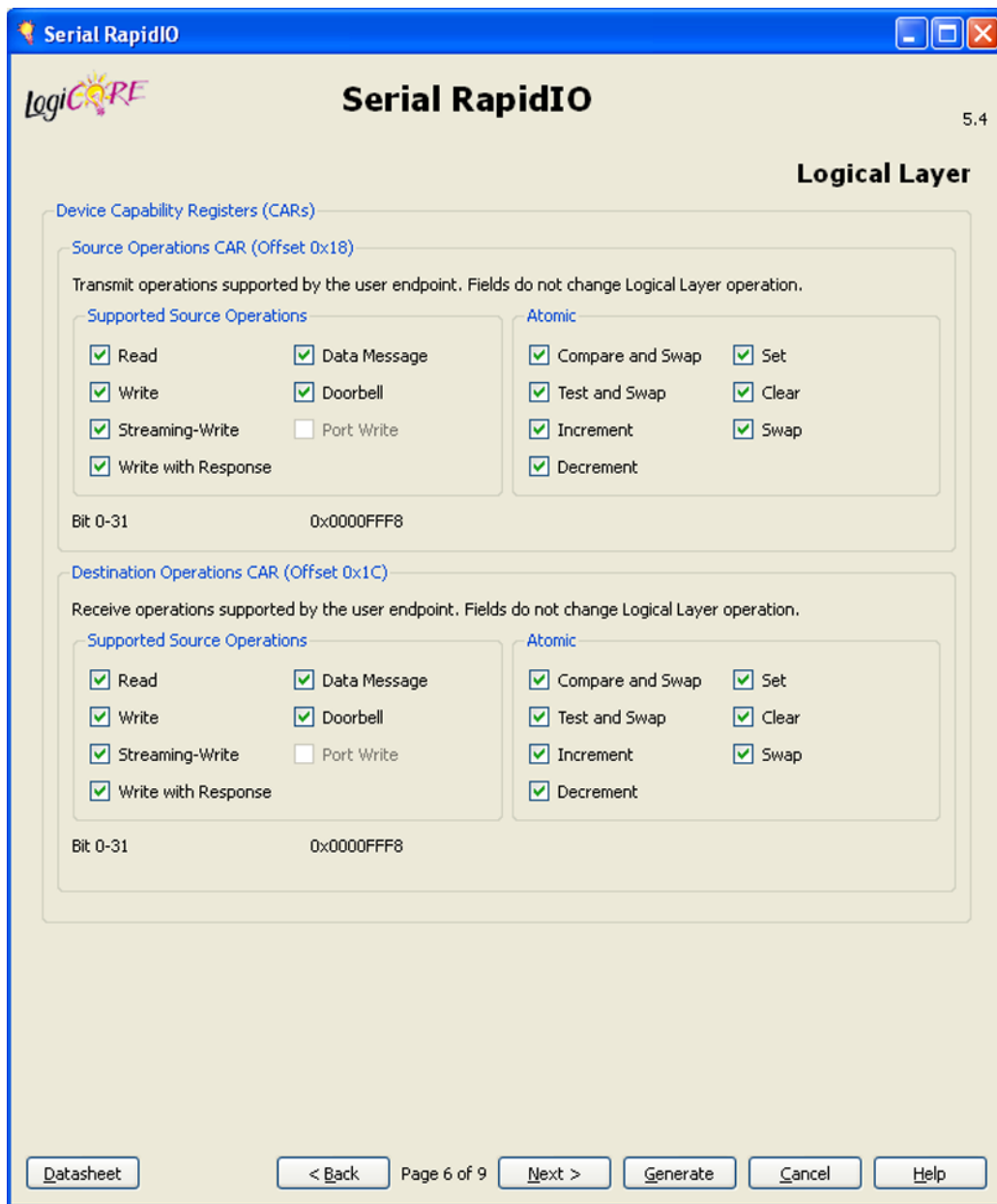
Selects the major functionality provided by the processing element. Allowable options are:

- Bridge
- Memory
- Processor

Memory is the default setting. This field does not alter core functionality.

## Logical Layer CAR Configuration

The Operations CARs, shown in [Figure 3-6](#), are meant to indicate to system software what transactions this endpoints support. These should reflect the capabilities of your design. They will not effect core functionality or size.



The screenshot shows the 'Serial RapidIO' window with the 'Logical Layer' tab selected. It displays two CAR configuration sections: 'Source Operations CAR (Offset 0x18)' and 'Destination Operations CAR (Offset 0x1C)'. Both sections include a 'Supported Source Operations' list and an 'Atomic' list. The 'Supported Source Operations' list includes: Read, Write, Streaming-Write, Write with Response, Data Message, Doorbell, and Port Write (unchecked). The 'Atomic' list includes: Compare and Swap, Test and Swap, Increment, Decrement, Set, Clear, and Swap. The 'Bit 0-31' field for both CARs is set to 0x0000FFF8. The window also features a 'Datasheet' button and navigation controls at the bottom.

**Serial RapidIO** 5.4

**Logical Layer**

**Device Capability Registers (CARs)**

**Source Operations CAR (Offset 0x18)**

Transmit operations supported by the user endpoint. Fields do not change Logical Layer operation.

**Supported Source Operations**

- ☒ Read
- ☒ Write
- ☒ Streaming-Write
- ☒ Write with Response
- ☒ Data Message
- ☒ Doorbell
- ☐ Port Write

**Atomic**

- ☒ Compare and Swap
- ☒ Test and Swap
- ☒ Increment
- ☒ Decrement
- ☒ Set
- ☒ Clear
- ☒ Swap

Bit 0-31: 0x0000FFF8

**Destination Operations CAR (Offset 0x1C)**

Receive operations supported by the user endpoint. Fields do not change Logical Layer operation.

**Supported Source Operations**

- ☒ Read
- ☒ Write
- ☒ Streaming-Write
- ☒ Write with Response
- ☒ Data Message
- ☒ Doorbell
- ☐ Port Write

**Atomic**

- ☒ Compare and Swap
- ☒ Test and Swap
- ☒ Increment
- ☒ Decrement
- ☒ Set
- ☒ Clear
- ☒ Swap

Bit 0-31: 0x0000FFF8

[Datasheet](#) [< Back](#) Page 6 of 9 [Next >](#) [Generate](#) [Cancel](#) [Help](#)

Figure 3-6: Logical Layer CAR Configuration (Screen 2)

## Source Operations CAR

**Important!** Modifications to the Source Operations CAR do not affect core operation. It is an informational register describing the capabilities of the user design.

### Supported Source Operations

Select the standard transmit operations supported by the processing element. The supported source operations are listed in the following text. All operations except Port-Write are enabled by default.

- Read
- Write
- Streaming-Write
- Write with Response
- Data Message
- Doorbell
- Port-Write

### Atomic

Select the atomic transmit operations supported by the processing element. Following is a list of supported atomic operations, and all are enabled by default.

- Compare and Swap
- Test and Swap
- Increment
- Decrement
- Set
- Clear
- Swap

## Destination Operations CAR

**Important!** Modifications to the Destination Operations CAR have no effect on core operation. It is an informational register describing the capabilities of the user design.

### Supported Destination Operations

Select the standard receive operations supported by the processing element. Following is a list of supported standard operations. All operations except Port-Write are enable by default.

- Read
- Write
- Streaming-Write
- Write with Response
- Data Message
- Doorbell
- Port-Write



## Atomic

Select the atomic receive operations supported by the processing element. Following is a list of supported atomic operations, and all are enabled by default.

- Compare and Swap
- Test and Swap
- Increment
- Decrement
- Set
- Clear
- Swap

## Logical Layer CSR Configuration

The Logical Layer Command and Status Registers (CSRs) shown in Figure 3-7 are read/write after implementation. Descriptions and impact of each register is detailed on the pages that follow.



**Serial RapidIO** 5.4

**Logical Layer**

Device Command and Status Registers (CSRs)

Local Configuration Space Base Address 1 CSR (Offset 0x5C)

Bit 0	-	Reserved
Bit 1-31	7FE00000	LCSBA

Base Device ID CSR (Offset 0x60)

Bit 0-7	-	Reserved
Bit 8-15	FF	Base Device ID
Bit 16-31	00FF	Large Base Device ID

Host Base Device ID CSR (Offset 0x68)

Bit 0-15	-	Reserved
Bit 16-31	FFFF	Host Base Device ID

Buttons: Datasheet, < Back, Page 7 of 9, Next >, Generate, Cancel, Help

Figure 3-7: Logical Layer CSR Configuration Screen

## Device Command and Status Registers (CSRs)

### Logical Configuration Space Base Address (LCSBA) CSR

Sets the local configuration register space address offset. It is a 31-bit hexadecimal value. Only the most significant 10 bits are used for Maintenance transaction address comparisons. Standard Read and Write transactions into this space are routed to the Maintenance and access the sRIO register space. The default setting is 0x7FE00000.

### Base Device ID CSR

#### Base Device ID

This is the device ID set on the Logical Layer Configuration screen. Valid only if "Use 8-bit Device IDs" was set.

#### Large Base Device ID

This is the Device ID set on the Logical Layer Configuration screen. Valid only if "Use 16-bit Device IDs" was set.

### Host Base Device ID CSR

Allows system host to identify itself by writing its Device ID to this register. Not GUI modifiable. Default Value is 0xFFFF.

## Serial Physical Layer CAR/CSR Configuration

The Serial PHY Layer Capabilities Registers (CARs) and Configuration Status Registers (CSRs) as shown in Figure 3-8 are detailed after the diagram.

**Serial RapidIO** 5.4

**Serial Physical Layer**

**Device Capability registers (CARs)**

**Processing Element Features CAR (Offset 0x10)**

Bit 25	0b1	Re-transmit Suppression Support
Bit 26	0b1	CRF Support

**Device Command and Status Registers (CSRs)**

**1x/4x LP-Serial Register Block Header CSR (Offset 0x100)**

Bit 0-15	0000	Extended Features Pointer
Bit 16-31	0001	Extended Features ID

**Port Link Time-out Control CSR (Offset 0x120)**

Bit 0-23	FFFFFF	Link Time-out Interval
Bit 24-31	-	Reserved

**Port Response Time-out Control CSR (Offset 0x124)**

Bit 0-23	FFFFFF	Response Time-out Interval
Bit 24-31	-	Reserved

**Port General Control CSR (Offset 0x13C)**

Bit 0	0b1	Host	<input checked="" type="checkbox"/>
Bit 1	0b1	Master Enable	<input checked="" type="checkbox"/>
Bit 2	0b1	Discovered	<input checked="" type="checkbox"/>
Bit 3-31	-	Reserved	

[Datasheet](#) [< Back](#) Page 8 of 9 [Next >](#) [Generate](#) [Cancel](#) [Help](#)

Figure 3-8: Serial Physical Layer CAR/CSR Configuration Screen

## Device Capability Registers (CARs)

### Processing Element Features CAR

#### Re-transmit Suppression Support

Allows the Endpoint to conditionally suppress errors due to CRC failures. Packets are marked as accepted, but dropped. The following Port Response Time-out Control CSR screen allows you to predefine packet priority levels to suppress packet retransmission.

#### CRF Support

Indicates whether the CRF bit is being used for extended priority mapping. This register reflects the support option that was selected on the Physical Layer Configuration screen ([Figure 3-2.](#))

## Device Command and Status Registers (CSRs)

### 1x/4x LP-Serial Register Block Header CSR

Sets the extended features ID. The extended features ID indicates the extended features of the core. The allowable range is any 16-bit hexadecimal value 0000h to FFFFh. The default is 0001h to reflect the endpoint functionality of the core.

### Port Link Time-out Control CSR

Time-out value the PHY uses when determining a link control symbol, such as Packet Accepted or Link Response, has been lost. When this counter expires, link protocol as defined in the RapidIO Serial PHY spec is followed. A max time-out value of FF\_FFFFh corresponds to a time-out length of approximately 3.33 seconds for a 3.125G core. For 1.25G and 2.5G cores, the max value corresponds to about 4.16s. The time-out values scale linearly.

### Port Response Time-out Control CSR

Time-out value to be used by an end-point to determine a lost packet. A max time-out value of FF\_FFFFh should correspond to a time-out length of approximately 3.33 seconds.

### Port General Control CSR

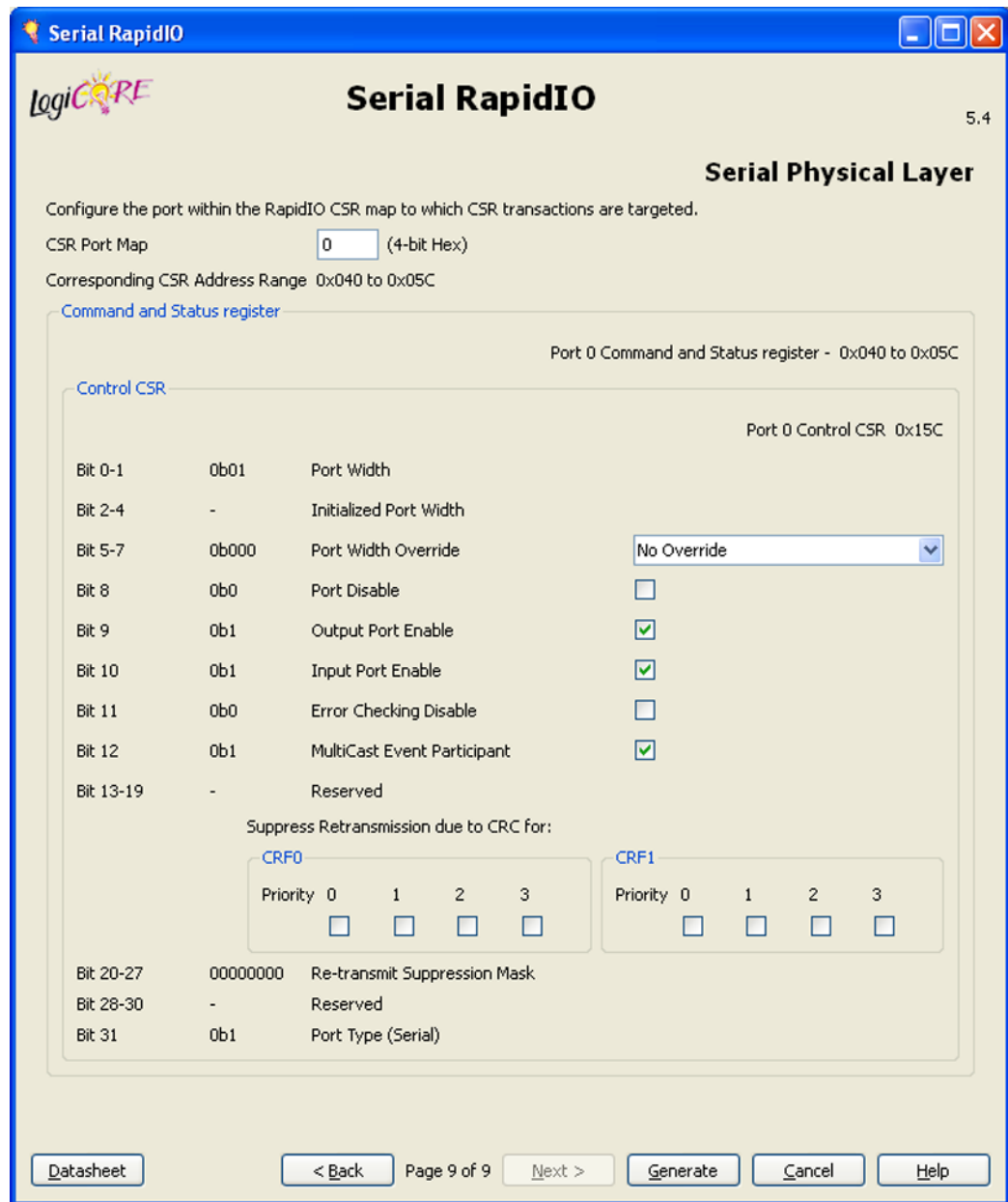
The Host bit indicates that the device is a host device. If this bit is not set, the device is an agent or a slave. This bit does not affect core functionality.

The Master Enable bit controls whether or not a device is allowed to issue requests to the system. If the Master Enable bit is not set, the device may only respond to requests.

The Discovered bit indicates that the device has been located by the processing element responsible for system configuration. This bit does not affect endpoint operation.

## Serial Physical Layer CSR Configuration

The port control CSR is Configurable using the Serial Physical Layer CSR Configuration screen (Figure 3-9). Details on each field are provided in this section.



**Serial RapidIO** 5.4

**Serial Physical Layer**

Configure the port within the RapidIO CSR map to which CSR transactions are targeted.

CSR Port Map:  (4-bit Hex)

Corresponding CSR Address Range: 0x040 to 0x05C

Command and Status register

Port 0 Command and Status register - 0x040 to 0x05C

Control CSR

Port 0 Control CSR: 0x15C

Bit	Value	Description	Configuration
Bit 0-1	0b01	Port Width	
Bit 2-4	-	Initialized Port Width	
Bit 5-7	0b000	Port Width Override	No Override
Bit 8	0b0	Port Disable	<input type="checkbox"/>
Bit 9	0b1	Output Port Enable	<input checked="" type="checkbox"/>
Bit 10	0b1	Input Port Enable	<input checked="" type="checkbox"/>
Bit 11	0b0	Error Checking Disable	<input type="checkbox"/>
Bit 12	0b1	MultiCast Event Participant	<input checked="" type="checkbox"/>
Bit 13-19	-	Reserved	

Suppress Retransmission due to CRC for:

CRF0

Priority	0	1	2	3
	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

CRF1

Priority	0	1	2	3
	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Bit 20-27	00000000	Re-transmit Suppression Mask
Bit 28-30	-	Reserved
Bit 31	0b1	Port Type (Serial)

Buttons: Datasheet, < Back, Page 9 of 9, Next >, Generate, Cancel, Help

Figure 3-9: Serial Physical Layer CSR Configuration Screen

## CSR Port MAP

Sets the port (defined in the RapidIO CSR map) to which CSR transactions are targeted. The allowable range is any single hexadecimal value of 0 through F. The default is zero (0). A change to this value modifies the PHY port CSR address offsets.

## Command and Status Register

### Port Width Override

This sets the default port width for 4x configurations. Default is No Override.

- **No Override** keeps the 4x configuration in 4x mode.
- **Force Single Lane 0** causes the 4x configuration to operate in 1x mode on Lane 0.
- **Force Single Lane 2** causes the 4x configuration to operate in 1x mode on Lane 2.

### Port Disable

Enabling this option disables port receivers and drivers (serial transceivers). When disabled, the port is unable to transmit or receive packets and control symbols. The port is enabled (unchecked) by default.

### Output Port Enable

This enables transmission on the output port. When disabled, the port can only transmit maintenance packets; control packets are unaffected. Output Port Enable is enabled by default.

### Input Port Enable

This enables reception on the input port. When disabled, the port can only receive maintenance packets. Non-maintenance packets generate packet-not-accepted (PNA) control symbols. Control symbols are unaffected. Input Port Enable is enabled by default.

### Error Checking Disable

Select this option to disable all error checking. Doing so causes all packets to be passed to the link interface, and all packets are tagged as accepted. Core behavior is undefined when this option is selected and a packet with an error is received. Error Checking is enabled (unchecked) by default.

### Multicast Event Participant

Select this option to set the core as a multicast event participant. This causes multicast event control symbols to be sent to this port. Multicast Event Participation is enabled by default.

### Re-transmit Suppression Mask

Conditionally suppress CRC errors based on packet priority. Extended priority through CRF is available only if CRF support is enabled on the Serial Physical Layer CSR Configuration screen ([Figure 3-8.](#))

## Output Generation

The output files generated from the Xilinx CORE Generator™ software are placed in the project directory. The file output list may include some or all of the following files. For complete descriptions of these output files, see [“Directory Structure and File Descriptions” in Chapter 5](#) of this guide.

- Serial RapidIO Physical Layer netlist
- RapidIO Logical Layer netlist
- Serial RapidIO Buffer netlist
- Serial RapidIO Physical Layer simulation model
- RapidIO Logical Layer simulation model
- Supporting CORE Generator software files
- Serial RapidIO release notes and other documentation
- Register Manager reference design
- Serial RapidIO endpoint example design
- Synthesis and implementation scripts
- Mentor ModelSim or Cadence IUS functional simulation scripts
- Clocking template and user constraint file



# Quick Start Example Design

---

This chapter introduces the RapidIO Endpoint reference design that is included with the Serial RapidIO Endpoint solution. The reference design demonstrates how to generate the Serial RapidIO Endpoint solution, including the Serial RapidIO Physical Layer, Buffer design and RapidIO Logical Layer cores, and illustrates using the default options in the RapidIO Endpoint example design. For detailed information about the example design, see [Chapter 5, “Detailed Example Design.”](#)

## Overview

[Figure 4-1](#) shows the Serial RapidIO Endpoint example design, which contains a simulation host model interfacing to a target design that writes and reads data to memory. Both the simulation host and the user design use the RapidIO Endpoint reference design, consisting of a Serial RapidIO Physical Layer core, Buffer core, RapidIO Logical Layer core, and Register Manager reference design. For a description of the components provided with the Serial RapidIO Endpoint, see “[Chapter 5, “Detailed Example Design.”](#)”

For detailed information about the Serial RapidIO Physical Layer and the RapidIO Logical Layer cores, see the *Serial RapidIO User Guide*.

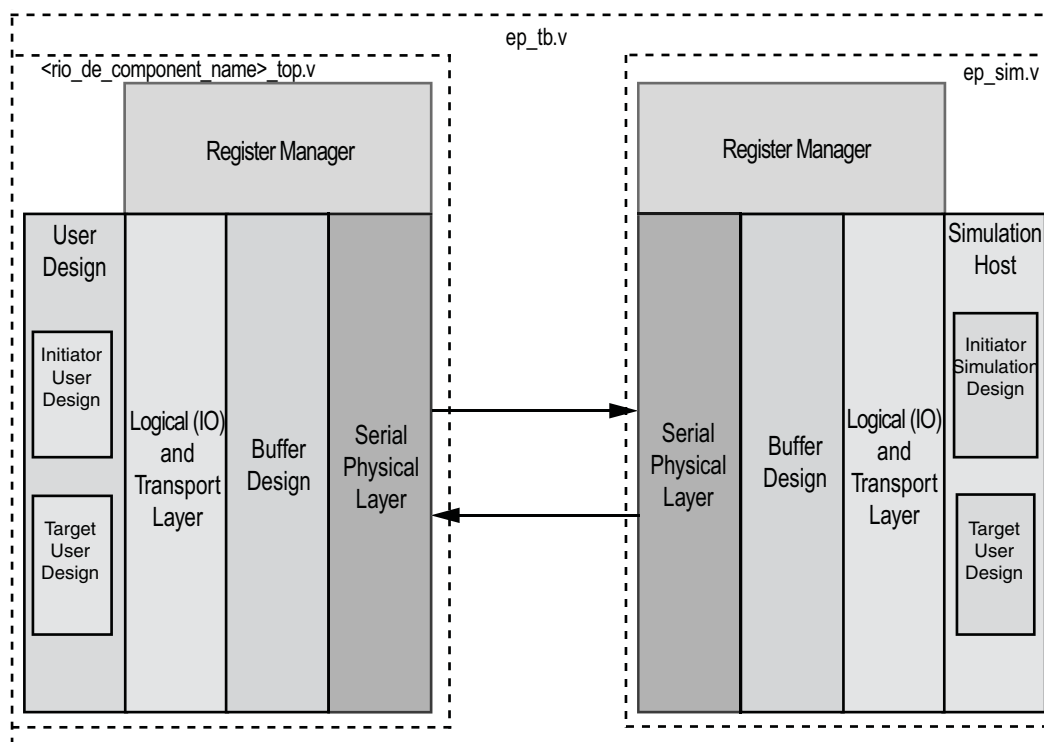


Figure 4-1: The RapidIO Endpoint Example Design Configuration

## Generating the Cores

Before you can use the Serial RapidIO Endpoint solution, the Serial RapidIO Physical Layer and RapidIO Logical Layer cores must be generated in the CORE Generator™ software within the same CORE Generator software project directory. To generate these cores, you must first create a CORE Generator software project.

To create a CORE Generator software project:

1. Start the CORE Generator software.  
For help starting the CORE Generator software, see the *Xilinx CORE Generator Guide*.
2. Choose File > New Project.
3. Type a directory name. In this example, the directory is named <project\_dir>.

4. Set the following options:

**Part Options:**

- a. From Target Architecture, select the desired Virtex® device family. Supported families include Virtex-6, Spartan-6 LXT, Virtex-5 LXT/FXT, and Virtex-4 FX devices.

**Note:** If an unsupported silicon family is selected, the RapidIO cores will not appear in the list of available cores.

**Note:** The Device, Package, and Speed Grade selected in the Part options tab have no effect on the generated cores. The core is delivered with an example UCF, targeting the Virtex-6 XC6VLX240T-1156-1, Spartan-6 XC6SLX45T-FGG484-2, Virtex-5 XC5VLX50T-FF1136-1, Virtex-5 XC5VFX70TFF1136-1, or Virtex-4 XC4VFX60-FF1152-10, depending on the chosen target architecture.

**Generation Options:**

- b. Select either VHDL or Verilog for the Design Entry.

**Note:** A VHDL reference design is not supported for Virtex-4 devices.

- c. For Vendor, select Synplicity or Other (for XST).

**Advanced Options:**

Leave the advanced options at their default values.

**To generate a Full Serial RapidIO Endpoint with default values**

1. After creating the project, locate the directory containing Serial RapidIO in the list of available cores. It should be located under Standard Bus Interfaces > RapidIO > Serial RapidIO.
2. Double-click the Serial RapidIO core.  
If the license file is not properly configured, the CORE Generator software displays an error. See [Chapter 2, "Licensing the Cores"](#) for details.
3. If a warning appears regarding the limitations of the available license, click OK.  
The Serial RapidIO customization screen appears.
4. Accept the default values on the screen, and then click Finish.

By default, the cores are named srio\_phy\_v5\_4 and rio\_log\_io\_v5\_4 and are generated into the Component Name directory within the project directory. Also generated in the Component Name directory are the supporting files for the core, including the Serial RapidIO example endpoint design. Detailed information about the files and directories delivered with the Serial RapidIO core is provided in the following chapter, ["Directory Structure and File Descriptions,"](#) page 47.

## Implementing the Example Design

After the Serial RapidIO cores are generated, the Serial RapidIO Endpoint example design can be synthesized by XST or Synplify, depending on the Vendor setting chosen in the CORE Generator software project options, and processed by the Xilinx implementation tools. The output files generated by the Serial RapidIO Endpoint solution include scripts to assist you in running synthesis and implementation.

In the implementation example that follows, `srio_v5_4` is the component name as generated by default from the Serial RapidIO Endpoint solution configuration screen. If a core is generated with a different name, substitute that core name in the following commands.

- From the CORE Generator software project directory window, type the following to implement the Serial RapidIO Endpoint example design.

### Windows

```
> cd srio_v5_4\implement
> implement.bat
```

### Linux

```
% cd srio_v5_4/implement
% ./implement.sh
```

These commands execute a script that synthesizes, builds, maps, and place-and-routes the Serial RapidIO Endpoint example design including the Serial RapidIO Physical Layer and RapidIO Logical Layer core netlists. The resulting files are placed in the results directory of the Serial RapidIO Endpoint solution, which is created by the `implement` script at runtime.

## Running the Simulation

Using the Serial RapidIO Endpoint example design (delivered as part of the Serial RapidIO Endpoint solution), you can quickly simulate and observe the behavior of the RapidIO cores.

### Setting up the Simulation

To run the gate-level simulation you must have the Xilinx Simulation Libraries compiled for your system. See the Compiling Xilinx Simulation Libraries (COMPXLIB) in the *Xilinx ISE® Synthesis and Verification Design Guide*, and the *Xilinx ISE Software Manuals and Help*. You can download these documents from:

[www.xilinx.com/support/software\\_manuals.htm](http://www.xilinx.com/support/software_manuals.htm).

The Xilinx simulation libraries must be mapped into the simulator. If the libraries are not set for your environment, go to [Answer Record 15338](#) on [www.xilinx.com/support](http://www.xilinx.com/support) for assistance compiling Xilinx simulation models and setting up the simulator environment.

Virtex-6, Spartan-6, and Virtex-5 devices require a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator. For VHDL simulation, a mixed HDL license is required.

## Functional Simulation

This section contains instructions for running a functional simulation of the Serial RapidIO Endpoint solution using Verilog. Functional simulation models for the Serial RapidIO Physical Layer and RapidIO Logical Layer cores are provided when the cores are generated. Implementing the core before simulating the functional models is not required.

In the following simulation examples, the defaults of `<project_dir>` for the CORE Generator software project directory, and `srio_v5_4` for the component name of the Serial RapidIO Endpoint solution are used. If you generate the core with a different name, substitute that component name in the following commands.

To run the functional simulation of the example design using ModelSim:

1. Launch the ModelSim simulator and set the current directory to  
`<project_dir>/srio_v5_4/simulation/functional`
2. Launch the simulation script:  
`modelsim> do simulate_mti.do`

To run the functional simulation of the example design using IUS:

1. Set the current directory to:  
`<project_dir>/srio_v5_4/simulation/functional`
2. Execute the simulation script:  
`% ./simulate_ncsim.sh`  
`> ./simulate_ncsim.bat`

The simulation script compiles the Serial RapidIO Physical Layer, Buffer, RapidIO Logical Layer functional simulation models, the Serial RapidIO Endpoint example design, and supporting simulation files. It then runs the simulation and checks ensure that it completed successfully. To observe the operation of the core, inspect the simulation transcript and the waveform.

## Creating an ISE Software Project

Located in the top level of your project directory is a `create_ise_prj.tcl` script. This script allows for easily integrating the Serial RapidIO core into an ISE software project using Project Navigator. The script creates a new project and locates all the necessary files and directories to synthesize the SRIO core. This script only supports a core generated outside of Project Navigator using the CORE Generator software with all components of the full solution selected (that is, Endpoint Example Design, Serial RapidIO Logical Layer, and Serial RapidIO Physical Layer).

To use this script, from a command prompt navigate into your project directory where this script is located. Run the command `"xtclsh create_ise_prj.tcl build_project"`. After this script executes, a `.ise` file will be located in the same directory. It can then be opened to work with the project or implement the core through XST.



## Detailed Example Design

---

This chapter provides detailed information about the Serial RapidIO Endpoint example design. It describes the output files, directory structure, purpose and contents of the implementation scripts, operation of the example target design and the demonstration test bench.

### Directory Structure and File Descriptions

The complete LogiCORE™ IP Serial RapidIO Endpoint solution consists of the following cores:

- Serial RapidIO Physical Layer
- Buffer Design
- RapidIO Logical Layer
- Serial RapidIO Endpoint Example Design

The 1 lane (1x) and 4 lane (4x) Serial Physical Layer is incorporated into the Serial RapidIO Physical Layer core. The Logical (I/O) and Transport layers are combined into the RapidIO Logical Layer core. The RapidIO Endpoint solution includes a register manager reference design, and an example user design.

The Serial RapidIO Endpoint solution also includes synthesis and implementation scripts, simulation scripts, a demonstration test bench, and supporting simulation files for the example design. For the implementation and simulation scripts to work, all cores must be generated in the same CORE Generator™ software project directory.

As illustrated in the following directory structure, the CORE Generator software project is <project\_dir>; the component name for the Serial RapidIO Physical Layer core is <phy\_component\_name>; the component name for the RapidIO Logical Layer core is <logio\_component\_name>; and the component name for the Serial RapidIO Endpoint solution is <srio\_component\_name>.

## CORE Generator Software Project



[`<project directory>`](#)

Top-level project directory for the CORE Generator software.

## Serial RapidIO Endpoint



[`<project directory>/<srio\_component name>`](#)

Contains the Serial RapidIO release notes text file.



[`<srio\_component name>/doc`](#)

Contains the Serial RapidIO Endpoint solution PDF documentation.



[`<srio\_component\_name>/example\_design`](#)

Contains the source files necessary to create the RapidIO Endpoint example design.



[`<srio\_component\_name>/example\_design/chipscope`](#)

Contains the .xco files to create the Chipscope ILA, ICON, and VIO cores and a Chipscope project file.



[`<srio\_component\_name>/example\_design/reg\_manager`](#)

Contains the source files for the example and endpoint user design.



[`<srio\_component\_name>/example\_design/user`](#)

Contains the source files for the initiator and target user designs.



[`<srio\_component\_name>/implement`](#)

Contains the supporting files for synthesis and implementation of the RapidIO example design.



[`<srio\_component\_name>/implement/results`](#)

Contains the implementation scripts that are created when the implement scripts are run.



[`<srio\_component\_name>/netlists`](#)

Contains various CORE Generator software netlists for asynchronous FIFOs that are used in implementing the example design.



[`<srio\_component\_name>/simulation`](#)

Contains the test bench and other supporting source files used to create the RapidIO simulation model.



[`<srio\_component\_name>/simulation/functional`](#)

Contains the scripts and define files for simulating the RapidIO Endpoint example design in ModelSim and Cadence IUS.



# Directory and File Contents

The Serial RapidIO Endpoint solution cores directories and their associated files are defined in this section.

## Project Directory

The Serial RapidIO cores need to be generated from the same CORE Generator software project directory so that all the files appear in the <project\_dir> directory.

### <project directory>

The following table contains the project directory files for the Serial RapidIO Physical Layer, the RapidIO Logical Layer and the Serial RapidIO Endpoint solution cores.

**Table 5-1: Project Directory**

Name	Description
<project_dir>	
<srio_component_name>.xco	Log file from CORE Generator software describing which options were used to generate the Serial RapidIO core. An XCO file is generated by the CORE Generator software for each core that it creates in the current project directory. An XCO file can also be used as an input to the CORE Generator software.
<srio_component_name>_flist.txt	A text file listing all of the output files produced when the customized Serial RapidIO cores were generated in the CORE Generator software.

## Serial RapidIO Endpoint Solution

The following tables contain the files and their descriptions specific to the Serial RapidIO Endpoint solution.

### <project directory>/<srio\_component name>

The <srio\_component\_name> directory contains the release notes text file included with the core that contains last-minute changes and or updates.

**Table 5-2: Component Name Directory**

Name	Description
<project_dir>/<srio_component_name>	
srio_readme.txt	The Serial RapidIO Endpoint core release notes text file.
create_ise_prj.tcl	A TCL script to simplify creating an ISE® software project.
<phy_component_name>.ngc	The netlist for the Serial RapidIO Physical Layer core.
<phy_component_name>.ngc	The netlist for the RapidIO Physical Layer core.
<phy_component_name>.v <phy_component_name>.vhd	The structural simulation model for the Serial RapidIO Physical Layer core. It is used to support the functional simulation of the Serial RapidIO Physical Layer core in the RapidIO Endpoint example design.
<phy_component_name>.veo <phy_component_name>.vho	The HDL template for the Serial RapidIO Physical Layer.
<logio_component_name>.ngc	The netlist for the RapidIO Logical Layer core.
<logio_component_name>.v <logio_component_name>.vhd	The structural simulation model for the RapidIO Logical Layer core. It is used to support the functional simulation of the Serial RapidIO Physical Layer core in the RapidIO Endpoint example design.
<logio_component_name>.veo <logio_component_name>.vho	The HDL template for the RapidIO Logical Layer core.
rio_buffer.ngc	The netlist for the Buffer design.
rio_buffer.v rio_buffer.vhd	The structural simulation model for the Buffer design. It is used to support the functional simulation of the Buffer in the RapidIO Endpoint example design.

Table 5-2: Component Name Directory (*Continued*)

Name	Description
rio_buffer.veo rio_buffer.vho	The HDL template for the Buffer.

[Back to Top](#)

## <srio\_component\_name>/doc

This directory contains the Serial RapidIO Endpoint solution PDF documentation that is included with the core.

Table 5-3: Doc Directory

Name	Description
<project_dir>/<srio_component_name>/doc	
srio_gsg247.pdf	The <i>Serial RapidIO Getting Started Guide</i>
srio_ds696.pdf	The <i>Serial RapidIO Data Sheet</i>
srio_ug503.pdf	The <i>Serial RapidIO User Guide</i>

[Back to Top](#)

## <srio\_component\_name>/example\_design

This directory and its subdirectories contain all the source files, aside from the Serial RapidIO Physical Layer and RapidIO Logical Layer netlists, to create the RapidIO Endpoint example design. They include the Buffer design, register manager reference design, example user design, and user constraints file.

Table 5-4: Example Design Directory

Name	Description
<project_dir>/<srio_component_name>/example_design	
<srio_component_name>_top.ucf	The user constraints file (UCF) for the RapidIO Endpoint example design.
<srio_component_name>_top.v <srio_component_name>_top.vhd	The top-level HDL file for the RapidIO Endpoint example design.
<srio_component_name>_clk.v <srio_component_name>_clk.vhd	HDL clock module for generation of reference clocks and core clocks.
rio_wrapper.v rio_wrapper.vhd	HDL wrapper file that instantiates the SRIO Physical Layer core, RapidIO Logical Layer core, Buffer design, Register Manager reference design, and transceiver wrapper. The SRIO Physical Layer core and Serial RapidIO GT wrapper are instantiated as part of the Physical Layer wrapper.

Table 5-4: Example Design Directory (Continued)

Name	Description
phy_wrapper.v phy_wrapper.vhd	Wrapper file that instantiates SRIO Physical Layer core and the SRIO GT transceiver wrapper.
rio_reset.v rio_reset.vhd	Example reset module that generates and sequences the necessary resets from a sys_reset_n.
srio_gt_wrapper_v4_1x.v	Transceiver wrapper file for Virtex®-4 FPGA 1x configurations.
srio_gt_wrapper_v4_4x.v	Transceiver wrapper file for Virtex-4 FPGA 4x configurations.
srio_gt_wrapper_v5_1x.v srio_gt_wrapper_v5_1x.vhd	Transceiver wrapper file for Virtex-5 FPGA 1x configurations.
srio_gt_wrapper_v5_4x.v srio_gt_wrapper_v5_4x.vhd	Transceiver wrapper file for Virtex-5 FPGA 4x configurations.
srio_gt_wrapper_v6_1x.v srio_gt_wrapper_v6_1x.vhd	Transceiver wrapper file for Virtex-6 FPGA 1x configurations.
srio_gt_wrapper_v6_4x.v srio_gt_wrapper_v6_4x.vhd	Transceiver wrapper file for Virtex-6 FPGA 4x configurations.
srio_gt_wrapper_s6_1x.v srio_gt_wrapper_s6_1x.vhd	Transceiver wrapper file for Spartan-6 FPGA 1x configurations.
cal_block_v1_4_1.v	Calibration block for Virtex-4 FPGA GT11 transceivers.
oplm_pma_pcs_rst_sequence.v	Module that performs the necessary reset sequencing for the Virtex-4 FPGA GT11 transceivers.
gt11_init_rx.v	Initialization state machine within the RocketIO transceiver wrappers which manages the reset sequence for the receiving Virtex-4 FPGA transceivers.
gt11_init_tx.v	Initialization state machine within the RocketIO transceiver wrappers which manages the reset sequence for the transmitting Virtex-4 FPGA transceivers.
gt11_wrapper.v	GT11 wrapper from the RocketIO transceiver Wizard for Virtex-4 FPGA configurations.
unused_mgt.v	This wrapper can be placed around any unused MGTs to alleviate static operating behavior (see <a href="#">AR #22471</a> for details).
gt11_wrapper.xco	XCO file used to generate RocketIO transceiver associated files for Virtex-4 FPGA configurations.
gtp_wrapper.v gtp_wrapper.vhd	GTP wrapper from GTP Wizard for Virtex-5 LXT/SXT or Spartan-6 LXT FPGA configurations.

Table 5-4: Example Design Directory (Continued)

Name	Description
gtp_wrapper_tile.v gtp_wrapper_tile.vhd	GTP wrapper from GTP Wizard for Virtex-5 LXT/SXT or Spartan-6 LXT FPGA configurations.
gtp_wrapper.xco gtp_wrapper_vhd.xco	XCO file used to generate gtp_wrapper.v and gtp_wrapper_tile.v files from the GTP Wizard for Virtex-5 LXT/SXT or Spartan-6 LXT FPGA configurations.
gtx_wrapper.v gtx_wrapper.vhd	GTX wrapper from GTX Wizard for Virtex-5 FXT FPGA configurations.
gtx_wrapper_tile.v gtx_wrapper_tile.vhd	GTX wrapper from GTX Wizard for Virtex-5 FXT FPGA configurations.
gtx_wrapper.xco gtx_wrapper_vhd.xco	XCO file used to generate gtx_wrapper.v and gtx_wrapper_tile.v files from the GTX Wizard for Virtex-5 FXT FPGA configurations.
gtx_wrapper.v gtx_wrapper.vhd	GTX wrapper from GTX Wizard for Virtex-6 FPGA configurations.
gtx_wrapper_gtx.v gtx_wrapper_gtx.vhd	GTX wrapper from GTX Wizard for Virtex-6 FPGA configurations.
gtx_wrapper.xco gtx_wrapper_vhd.xco	XCO file used to generate gtx_wrapper.v and gtx_wrapper_gtx.v files from the GTX Wizard for Virtex-6 FPGA configurations.

### <srio\_component\_name>/example\_design/chipscope

This directory contains the .xco files to create the Chipscope ILA, ICON, and VIO cores through CORE Generator. It also contains a Chipscope project file. This Chipscope design can be used to monitor activity in the physical layer and logical layer. It also provides the ability to initiate traffic.

Table 5-5: Chipscope Directory

Name	Description
<project_dir><srio_component_name>/example_design/chipscope	
phy_ila.xco	This .xco file is used by CORE Generator to create a Chipscope ILA core that is used to monitor activity at the Physical Layer core.
rio_ila.xco	This .xco file is used by CORE Generator to create a Chipscope ILA core that is used to monitor activity at the Logical Layer core.
srio_icon.xco	This .xco file is used by CORE Generator to create a Chipscope ICON controller that is used to control each of the various Chipscope cores.

Table 5-5: Chipscope Directory (Continued)

Name	Description
srio_vio.xco	This .xco file is used by CORE Generator to create a Chipscope VIO core that is used initiate transactions.
srio_cs_bb.v	This is a blackbox file used to instantiate the Chipscope cores.
srio.cpj	This is a Chipscope project file containing predefined configurations for each ILA and for controlling the VIO interface.

### <srio\_component\_name>/example\_design/reg\_manager

This directory contains the source files for the Register Manager reference design. The register manager is designed to process configuration packets from the RapidIO Logical Layer and Serial RapidIO Physical Layer cores. Once a configuration packet is received, the register manager determines the appropriate action to take. The source code contains comments to help explain the functionality of the code. You are not required to use this register manager in your design; however, some type of register manager will be needed.

Table 5-6: Reg Manager Directory

Name	Description
<project_dir><srio_component_name>/example_design/reg_manager	
reg_manager.v	File containing the register manager reference design that provides read and write access to the entire RapidIO configuration space through a single interface. It acts as a master on a maintenance bus connecting the configuration ports of all RapidIO layers, as well as one user agent potentially mapped into the configuration space.
reg_manager.vhd	

## <srio\_component\_name>/example\_design/user

This directory contains the source files for the Initiator and Target User reference designs. The Initiator User interfaces to the Initiator Request and Response ports of the RapidIO Logical Layer while the Target User interfaces to the Target Request and Response ports. Both User designs are used as an example on how to interface to the RapidIO Logical Layer core and demonstrate the RapidIO Endpoint in simulation or hardware. The source code contains comments to help explain the functionality of the code. You are not required to use either User design in your design.

**Table 5-7: User Directory**

Name	Description
<project_dir><srio_component_name>/example_design/user	
user_top.v user_top.vhd	Top-level User design that instantiates both Initiator and Target User designs.
target_user.v target_user.vhd	The Target User design interfaces to the Target Request and Target Response ports of the Logical Layer. It processes read and write requests. NWRITE, NWRITE_R and SWRITE requests are stored in a Block RAM. It also generates responses as necessary to NREAD and NWRITE_R commands.
initiator_user.v initiator_user.vhd	Top-level Initiator User design.
tickler.v tickler.vhd	User interface of the Initiator User design. It is used to send various packet type requests to the IREQ Generator.
ireq_generator.v ireq_generator.vhd	The Initiator Request (IREQ) Generator of the Initiator User design. It creates the requested Initiator Request packet for the Logical Layer. It also handles the generation of the SOF, EOF, and VLD control signals, populates the frame with the requested amount of data, and increments the Transaction ID.
iresp_handler.v iresp_handler.vhd	The Initiator Response (IRESP) Handler of the Initiator User design. It verifies incoming Initiator Response packets against expected responses generated from outgoing Initiator Request packets.

Table 5-7: User Directory (Continued)

initiator_bram.v initiator_bram.vhd	The Initiator block RAM (IRAM) module of the Initiator User design. It stores pre-loaded field information with different data patterns. This data is used to populate the DATA field of outgoing Initiator Request packets from the IREQ Generator module. The IRAM is also used by the IRESP Handler to verify the DATA field of incoming Initiator Response packets against the expected contents.
tid_bram.v tid_bram.vhd	The Transaction ID (TID) block RAM module of the Initiator User design. It passes information regarding expected Initiator Responses from the IREQ Generator to the IRESP Handler. The IRESP Handler will use this information to verify incoming responses.
history_bram.v history_bram.vhd	The History block RAM module of the Initiator User design. It is used to store information of outgoing Initiator Request and incoming Initiator Response packets. This history of previous transactions can be used to debug any errors. There are separate memory spaces for Initiator Request and Initiator Response packets.
fifo_16x190.v fifo_16x190.vhd	Structural simulation netlist for the FIFO used in the IREQ Generator.
target_checker.v target_checker.vhd	The Target Checker module verifies target request transactions.



## <srio\_component\_name>/implement

This directory contains the support files necessary for synthesis and implementation of the Serial RapidIO Endpoint example design with the Xilinx tools.

**Table 5-8: Implementation Directory**

Name	Description
<project_dir>/<srio_component_name>/implement	
implement.bat	A Windows batch file that processes the example design through the Xilinx tool flow.
implement.sh	A Linux shell script that processes the example design through the Xilinx tool flow.
xst.prj	The XST project file for the example design; it lists all of the source files to be synthesized. It is only available when the CORE Generator software vendor project option is set to "Other."
xst.scr	The XST script file for the example design that is used to synthesize the core, called from the implement script described previously. It is only available when the CORE Generator software vendor project option is set to "Other."
synplify.prj	The Synplicity project file for the example design; it lists all of the source files to be synthesized. It is only available when the CORE Generator software vendor project option is set to "Synplicity."

## <srio\_component\_name>/implement/results

The results directory is created when the implement scripts are ran and contains the resulting implementation files.

## <srio\_component\_name>/netlists

This directory contains various CORE Generator software netlists for asynchronous FIFO's that are used in implementing the Buffer design.

**Table 5-9: Netlists Directory**

Name	Description
<project_dir>/<srio_component_name>/netlists	
fifo_16x190.ngc	Netlist for FIFO that is used in Initiator Request Generator module of the Initiator User design.

## <srio\_component\_name>/simulation

This directory and its subdirectories contain the files necessary to simulate the Serial RapidIO Endpoint example design. This directory contains the test bench and other supporting source files used to create the RapidIO host simulation model.

**Table 5-10: Simulation Directory**

Name	Description
<project_dir>/<srio_component_name>/simulation	
ep_sim.v ep_sim.vhd	This file contains the top level RapidIO host simulation model. It instantiates the physical layer, logical layer, register manager, buffer and host application.
ep_tb.v ep_tb.vhd	Test bench for the RapidIO Endpoint example design demonstrating the example user application.
sys_clk_gen.v sys_clk_gen.vhd	This module generates the system clock.
user_sim_host.v user_sim_host.vhd	Standalone test bench for the RapidIO Endpoint example design. It sends packets that are then expected in the same order on the response.

## <srio\_component\_name>/simulation/functional

This directory contains the scripts and define files for simulating the Serial RapidIO Endpoint example design in ModelSim and IUS.

**Table 5-11: Functional Directory**

Name	Description
<project_dir>/<srio_component_name>/simulation/functional	
simulate_mti.do	A ModelSim macro file that compiles the example design sources and the structural simulation models, then runs the functional simulation to completion.
simulate_ncsim.bat	A IUS batch file for Windows that compiles the example design sources and the structural simulation models, then runs the functional simulation to completion.
simulate_ncsim.sh	A IUS shell script for Linux that compiles the example design sources and the structural simulation models, then runs the functional simulation to completion.
wave_1x_ser.sv wave_4x_ser.sv	IUS macro files that open a wave window and add key signals to the wave viewer.

## Serial RapidIO Endpoint Example Design Description

The RapidIO Endpoint example design that is delivered with the Serial RapidIO Endpoint solution helps core designers understand how to use the Serial RapidIO Physical Layer and RapidIO Logical Layer cores in a design. The RapidIO Endpoint example design is shown in [Figure 5-1](#) and contains the following:

- Register Manager reference design
- Initiator and user reference designs
- Simulation host

The example design provides helpful information for building design using the Serial RapidIO Endpoint example solution. The Register Manager converts maintenance interface of the Logical Layer and the management interfaces of the LOGIO, Buffer, and SRIO management interfaces. The Initiator and Target User reference design shows you how to develop an interface to the RapidIO Logical Layer core. The simulation host provides a starting point for developing test benches for you own design. You can also simulate and implement the example design to understand how the endpoint example design works in your own design.

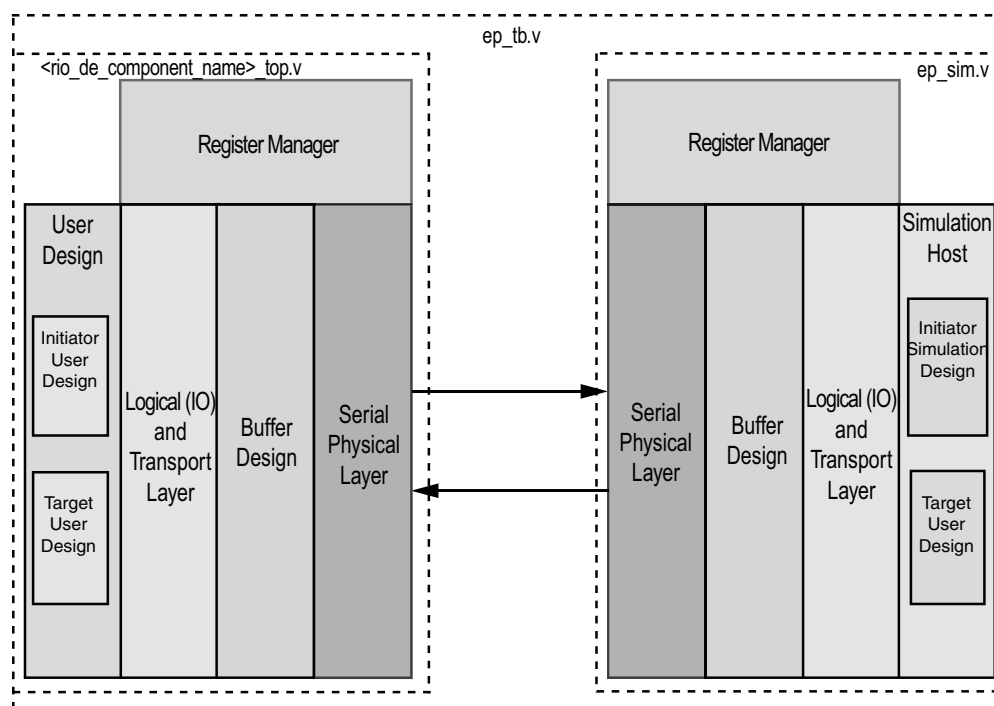


Figure 5-1: RapidIO Endpoint Core Example Design

## Initiator User Design

The Initiator User Design is an example design that interfaces to the Initiator Request and Response ports of the Logical Layer core. It is designed to behave as a Master and generate Initiator Request (IREQ) packets and verify corresponding Initiator Response (IRESP) packets. The Initiator User Design is comprised of six blocks, as shown in Figure 5-2.

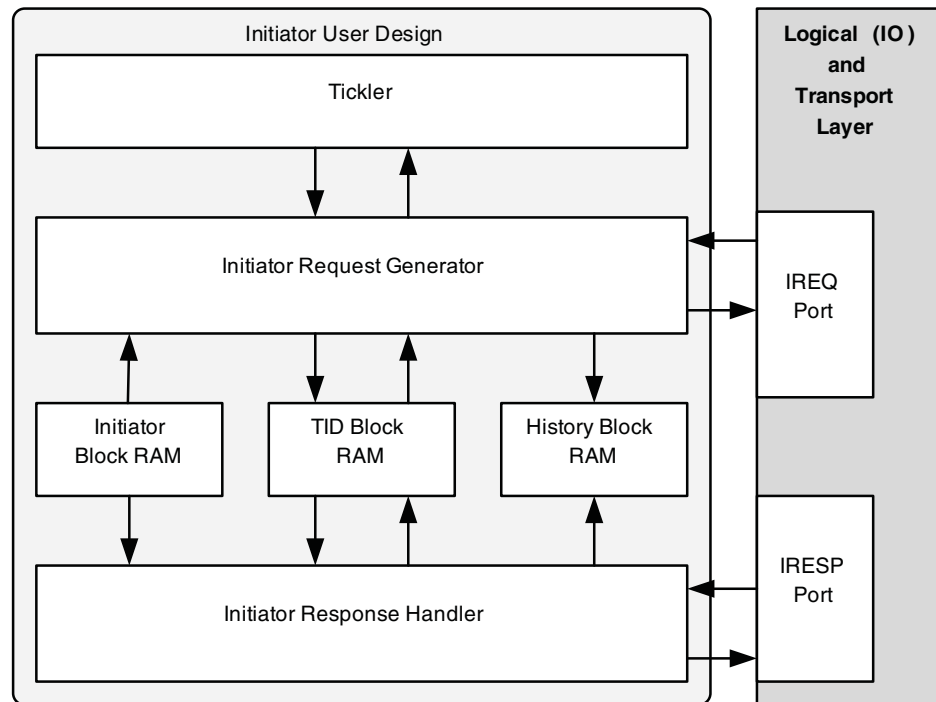


Figure 5-2: Initiator User Design

### Initiator Request Generator

The Initiator Request (IREQ) Generator is designed to aid in the generation of Initiator Request packets. It interfaces to the Tickler through LocalLink ready and valid control signals. The Tickler provides header information (such as ftype, ttype, prio, address) and data size to the IREQ Generator. The IREQ Generator then creates the requested IREQ packet for the LOGIO. It also handles the generation of the SOF, EOF, and VLD control signals, populates the frame with the requested amount of data, and increments the Transaction ID (TID).

Data is pulled either from the Initiator block RAM containing predefined data patterns or from the external Bypass Port. Pulling data from the Initiator block RAM allows test data to be easily transmitted and verified, where applicable. The Initiator block RAM is addressed using the address field of the header. The Bypass Port allows for custom data to be sent, specifically for Maintenance transactions.

The IREQ Generator passes information to the TID block RAM, which is used by the Initiator Response (IRESP) Handler for expected packet responses. The IRESP Handler uses this information to verify incoming responses. All outgoing frames (not just ones with expected responses) are written to the History block RAM and provide a record of previous transactions.

## Initiator Response Handler

The Initiator Response (IRESP) Handler is used to verify incoming IRESP packets against expected responses generated from outgoing Initiator Request (IREQ) packets. The Transaction ID (TID) or corresponding message fields are used to pull information from the TID block RAM for incoming IRESP packets. This information is used by the IRESP Handler to verify the received IRESP packet.

The IRESP Handler flags an error in any of the following situations:

- There is a data mismatch between the IRESP data and the expected data in the Initiator block RAM. The IRESP Handler can only verify data that originated from the Initiator block RAM.
- The IRESP Handler received an unexpected IRESP packet.
- An IRESP packet was received without an EOF or the EOF was late arriving; indicating that the packet is too long.
- An EOF was arrived earlier than expected for an IRESP packet indicating that the packet is shorter than expected.
- The STATUS field of an IRESP packet indicates a value other than "DONE" (4'b0000).
- The Destination ID received in the IRESP packet does not match the Device ID of the core.

## TID Block RAM

Transaction ID (TID) block RAM is used to store information of outgoing Initiator Request (IREQ) packets that are expected to generate responses. This can later be accessed by the Initiator Response (IRESP) Handler to verify incoming IRESP packets against outgoing IREQ packets. The TID block RAM is indexed with the packet TID or with the corresponding Message field if it is a Message packet.

The data stored for each entry includes an indication of whether or not to check the data, offset, and byte count of outgoing IREQ packets. The IRESP Handler uses this information to verify the data of incoming IRESP packets against the expected data in the Initiator block RAM. There is a bit indicating if the data field should be checked by the IRESP Handler. There is also a valid bit for each entry that indicates if the entry is valid and outstanding. If the IREQ Generator writes to an entry which is already valid, it is flagged as an error. If the IRESP Handler reads from an entry that is invalid, it is flagged as an error.

The TID block RAM maintains a count of the current number of valid outstanding entries in the block RAM. The counter increments when the IREQ port writes to it and is decremented when the IRESP reads from it. This outstanding packet count can be used to indicate if an expected response was never received.

## Initiator Block RAM

Initiator block RAM stores preloaded data field information. This data can be used by the Initiator Request (IREQ) Generator to populate the DATA field of outgoing IREQ packets. The Initiator block RAM data is also used by the Initiator Response (IRESP) Handler to verify the DATA field of incoming IResp packets against the expected contents.

The IRESP Handler can only verify data of incoming packets generated on outgoing IREQ packets with data from the Initiator block RAM. Therefore, it is important to keep the contents of the memory space on both ends of the link in sync to avoid erroneous data mismatches. For example, a write must occur with data from the Initiator block RAM before doing a read from the other end so that the IRESP Handler has known data to check against. For example, it is not possible to check the data of a Maintenance Read because there would be no expected data in the Initiator block RAM for the IRESP Handler to verify.

The Initiator block RAM is pre-loaded with different data patterns that can be accessed depending on the address space. The data types and their location in the block RAM address space are defined in the [Table 5-12](#).

**Table 5-12: Initiator Block RAM Data Types and Locations**

Address Range	Data Type	Init Value
0x000 – 0x01F	All 0s	0x00, 0x00, 0x00, ..., 0x00
0x020 – 0x03F	All 1s	0xFF, 0xFF, 0xFF, ..., 0xFF
0x040 – 0x05F	Alternating 0s and 1s Bytes	0x55, 0x55, 0x55, ..., 0x55
0x060 – 0x07F	Alternating 1s and 0s Bytes	0xAA, 0xAA, 0xAA, ..., 0xAA
0x080 – 0x09F	Alternating bytes of 0s and 1s	0x00, 0xFF, 0x00, ..., 0xFF
0x0A0 – 0x0BF	Alternating bytes of 1s and 0s	0xFF, 0x00, 0xFF, ..., 0x00
0x0C0 – 0x0DF	Alternating dwords of 0s and 1s	0x0000_0000_0000_0000, 0xFFFF_FFFF_FFFF_FFFF, ...
0x0E0 – 0x0FF	Alternating dwords of 1s and 0s	0xFFFF_FFFF_FFFF_FFFF, 0x0000_0000_0000_0000, ...
0x100 – 0x11F	Incrementing bytes	0x00, 0x01, 0x02, ..., 0xFF
0x120 – 0x13F	Incrementing dwords	0x0000_0000_0000_0000, 0x0000_0000_0000_0001, 0x0000_0000_0000_0002, ...
0x140 – 0x15F	Decrementing bytes	0xFF, 0xFE, 0xFD, ..., 0x00
0x160 – 0x17F	Decrementing dwords	0x0000_0000_0000_001F, 0x0000_0000_0000_001E, ..., 0x0000_0000_0000_0000
0x180 – 0x19F	Random data	0xB3, 0x4C, 0x66, 0x9D (for example)

**Table 5-12: Initiator Block RAM Data Types and Locations**

0x1A0 – 0x1BF	More random data	0x32, 0x18, 0xFF, 0x87 (for example)
0x1C0 – 0x3FF	More incrementing dwords	0x0000_0000_0000_0000, 0x0000_0000_0000_0001, 0x0000_0000_0000_0002, ...

## History Block RAM

The History block RAM stores information of outgoing Initiator Request (IREQ) and incoming Initiator Response (IRESP) packets. The IREQ and IRESP ports have separate memory spaces. This history of previous transactions can then be used to debug any failures. Although the read port exists, the logic is not currently implemented to read out the contents of the block RAM. This logic may be implemented by the user.

## Tickler

The Tickler interfaces to the Initiator Request (IREQ) Generator and generate IREQ packets. The ports of the Tickler interface to the IREQ Generator are described in [Table 5-13](#).

**Table 5-13: Tickler Interface to the IREQ Generator Ports**

Name	Width	Direction	Description
igen_vld		Out	Packet Valid Indication – indicates information presented to IREQ Generator is valid for new packet.
igen_rdy		In	IREQ Generator Ready Indication.
igen_bypass_data	[0:63]	Out	Bypass data port to allow data other than what is stored in the Initiator block RAM to be sent.
igen_stalls	[0:1]	Out	Number of cycles between packets at LOGIO IREQ interface.
igen_bypass_en		Out	Enable the Data Bypass port to use the data presented on the Bypass Data to populate the DATA field rather than contents of the Initiator block RAM. Registered at beginning of packet on ireq_gen_vld and ireq_gen_rdy and changing in the middle of packet has no effect.
igen_bypass_vld		Out	Bypass Data Valid – Indicates valid data is on Bypass Data bus.
igen_bypass_rdy		In	Bypass Data Ready – IREQ Generator is ready to accept data on Bypass Data port.

Table 5-13: Tickler Interface to the IREQ Generator Ports (*Continued*)

igen_addr	[0:33]	Out	Address of packet to be generated – Read/Write: {xamsbs[0:1], address[0:28], byte_offset[0:2]} Swrite: {xamsbs[0:1], address[0:28], 3'b0} Maintenance: {7'b0, config_offset[0:23], byte_offset[0:2]} Doorbell: N/A Message: {21'b0, iram_addr[0:9], 3'b0}
igen_size	[0:8]	Out	Number of bytes of data to transmit.
igen_ftype	[0:3]	Out	Format Type of packet to be generated.
igen_ttype	[0:3]	Out	Transaction Type of packet to be generated.
igen_prio	[0:1]	Out	Priority of packet to be generated.
igen_crf		Out	Critical Request Flow for packet to be generated.
igen_dest_id	[0:7] / [0:15]	Out	Destination ID of packet to be generated.
igen_hopcount	[0:7]	Out	Hopcount of maintenance write packets to be generated.
igen_local		Out	Local configuration indicating that the packet to be generated is a local maintenance packet.
igen_db_info	[0:15]	Out	Doorbell Info for Message packets.
igen_msglen	[0:3]	Out	Message Length – a value of 0x0 indicates a single-packet message and a value of 0xF indicates a 16-packet message.
igen_letter	[0:1]	Out	Letter for Message packets.
igen_mbox	[0:1]	Out	Mailbox for Message packets.
igen_msgseg	[0:3]	Out	Message Segment / mbox – this port is the Message Segment (msgseg) port for multiple packet messages, as determined by the ireq_gen_msglen port, and is the mbox port for single packet messages.
igen_error	[0:3]	Out	Error indications to the IREQ Generator.
g_error	[0:3]	In	Error indicators from IREQ Generator.



The Tickler and IREQ Generator interfaces communicate using the LocalLink style of ready and valid signals. When the IREQ Generator is ready and the Tickler has valid header information, the IREQ Generator generates an IREQ packet. The Tickler supports the generation of any supported packet type. The Tickler must present the header information of the request packet. It must also provide a size for packets with data so that the IREQ Generator can include the requested amount of data in the packet.

If the packet requires data, it can either be pulled from the Initiator block RAM or from the Bypass Port as determined by the `igen_bypass_en` setting. The Initiator block RAM contains predefined data patterns and is accessed by using the address header field. For the Bypass Port, data is loaded over another LocalLink style interface designed to interface to a FIFO. Specifically, the `igen_bypass_rdy` input from the IREQ Generator can be connected to the FIFOs `read_enable` port while the `igen_bypass_vld` output from the Tickler can be driven from the empty signal of the FIFO.

The Tickler can be modified to generate different packets or sequences than what is currently provided. To do this, the Tickler packet state machine can be modified to send the desired packets. Alternatively, you may remove the Tickler and build a custom interface into the IREQ Generator.

The Initiator User Design is intended to complete the RapidIO Endpoint example design and can be implemented in hardware. The user interfaces to the Initiator User Design through the Tickler interface. This is controlled through Chipscope delivered with the core. Refer to the *Serial RapidIO User Guide* ([UG503](#)) for more information on using Chipscope to generate traffic.

The resets and status LEDs are described in [Table 5-14](#) for each of the supported development platforms.

**Table 5-14: Tickler Interface to the Initiator User Design**

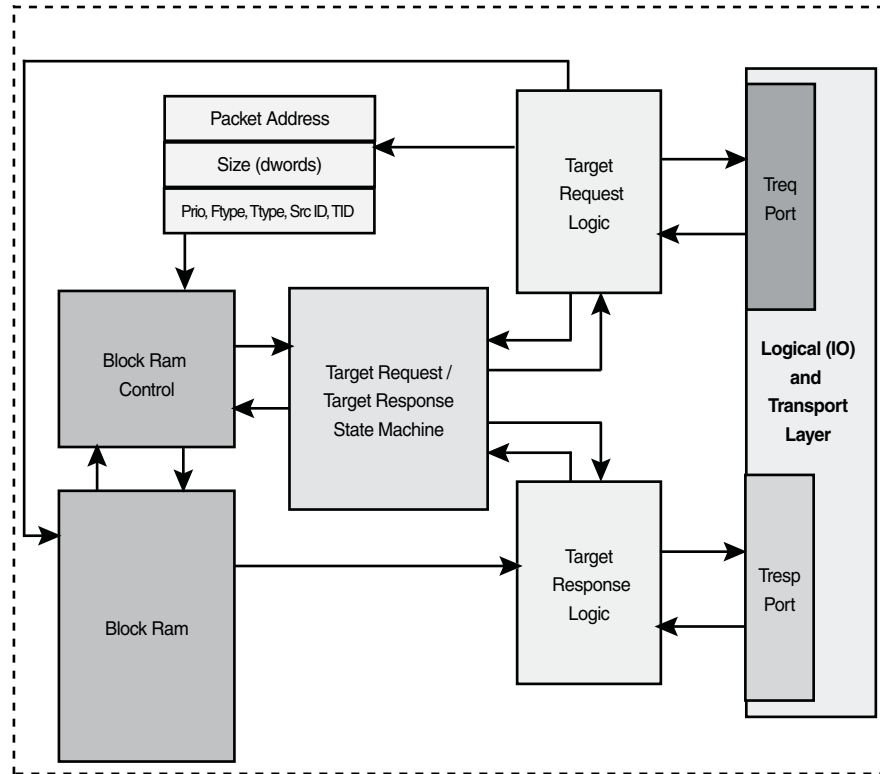
ML623	SP623	ML523	ML421	ML325	Function	Description
<b>Pushbutton</b>						
SW4	SW4	SW8	SW1	SW8	Link Reset	First issues Link Reset to the other end of the link. Once the <code>port_initialized</code> is deasserted in response to the Link Reset command, the FPGA logic is reset.
SW6	SW6	SW7	SW2	SW3	Local Reset	Resets all logic within the FPGA but does not issue a Link Reset command.
<b>LEDs</b>						
DS10	DS10	DS16	DS28	DS23	<code>mode_sel</code>	Mode Select from PHY (1 = 4x mode, 0 = 1x mode)
DS11	DS11	DS17	DS32	DS16	<code>~lnk_trdy_n</code>	Tx ready for link (1 = Tx link is ready, 0 Tx link is not ready)
DS12	DS12	DS18	DS33	DS17	<code>~lnk_rrdy_n</code>	Rx ready for link (1 = Rx link is ready, 0 = Rx link is not ready)

Table 5-14: Tickler Interface to the Initiator User Design (Continued)

ML623	SP623	ML523	ML421	ML325	Function	Description
DS13	DS13	DS19	DS34	DS18	port_initialized	Port Initialized (1 = port is initialized, 0 = port is not initialized)
DS14	DS14	DS20	DS35	DS19	lnk_porterr_n	Link Port Error (1 = port is not in error, 0 = port is in error state)
DS15	DS15	DS21	DS8	DS20	PLL Locked (Not available for Virtex-4)	Virtex-5 and Spartan-6 FPGA GT PLL Locked indicator (1 = locked, 0 = not locked).
DS16	DS16	DS22	DS8	DS21	DCM Locked	DCM Locked indicator (1 = locked, 0 = not locked)
DS17	DS17	DS23	DS10	DS22	Initiator User Error	Error indicator from Initiator User design (1 = error, 0 = no error)

## Target User Design

The Target User Design is an example target design that interfaces to the Target Request and Target Response ports of the Logical Layer. The user application portion of the target design shown in [Figure 5-3](#) is the `target_user.v` file and is created in the `<project_dir>/<srio_component_name>/example_design/user` directory when the Serial RapidIO Endpoint solution is generated through CORE Generator software.



**Figure 5-3: RapidIO Example Target Design Block Diagram**

The simulation host writes data into the block ram through the target request port on the RapidIO Logical Layer core. To read data, the simulation host will send a read request to the user application using the target request port. The user application will process the request and respond using the target response port. The simulation host uses `SWRITE`, `NWRITE`, `NWRITE_R`, and `NREAD` commands to interface to the user applications block ram.

The user application will process incoming packets on the target request port and respond if necessary. The user application will generate responses for `NWRITE_R` and `NREAD` commands.

For more information on interfacing to the RapidIO Logical Layer core, see the comments in the `user.v` file. Also, see the *LogiCORE Serial RapidIO User Guide*, `srio_ug503.pdf`, located in the

`<project_dir>/<srio_component_name>/doc`

directory after the Serial RapidIO core is generated in the CORE Generator software.

## Simulation Host

The simulation host, shown in Figure 5-4, is the `user_sim_host.v` file. It is created in the `<project_dir>/<srio_component_name>/simulation` directory when the Serial RapidIO Endpoint solution is generated through the CORE Generator software.

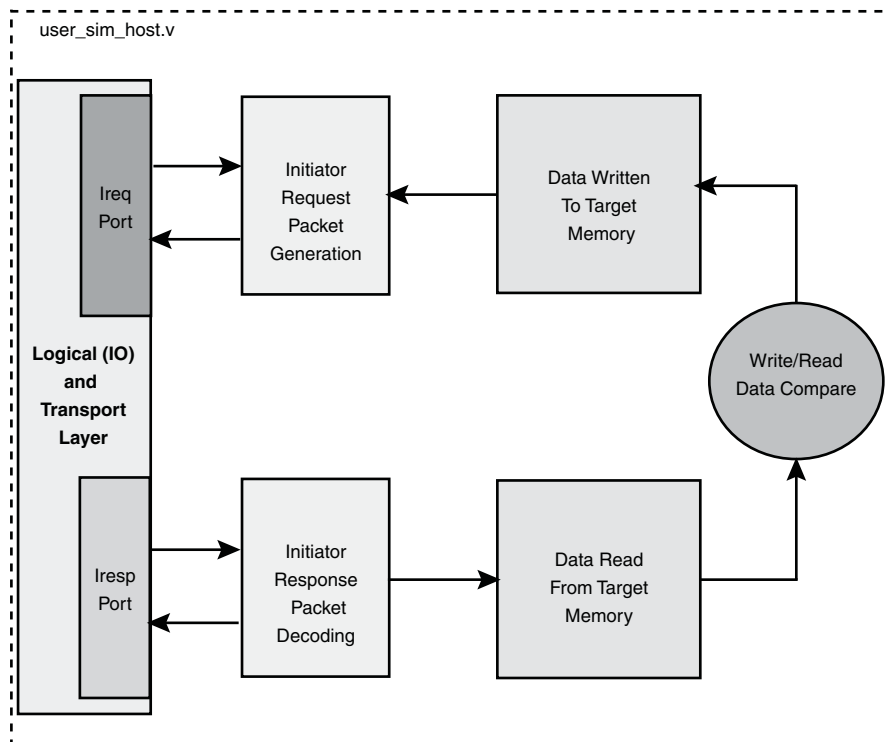


Figure 5-4: RapidIO Simulation Host Block Diagram

The simulation host initiates request to write data to the memory inside the target design user application. Data is written using `SWRITE`, `NWRITE`, or `NWRITE_R` commands. Data is then read back from the memory of the target design using the `NREAD` command and compared to the original data sent. Status and error messages are printed to the screen during operation so the user can monitor the progress of the test bench.

The Target User Design, `target_user.v`, is also instantiated in the Simulation Host. This adds target functionality to the simulation side allowing for the FPGA side to initiate requests and receive responses from the Simulation Host.