LogiCORE IP Serial RapidIO Gen2 Endpoint v1.5

Product Guide

PG007 July 25, 2012

XILINX®

Table of Contents

SECTION I: SUMMARY

Introduction

The LogiCORE™ IP Serial RapidIO Gen2 Endpoint Solution (SRIO Gen2 Endpoint) comprises a highly flexible and optimized Serial RapidIO Gen2 Physical Layer and a Serial RapidIO Gen2 Logical (I/O) and Transport Layer. This IP solution is provided in netlist form with supporting example design code. The SRIO Gen2 Endpoint supports 1x, 2x, and 4x lane widths. It comes with a configurable buffer design, reference clock module, reset module, and configuration fabric reference design. The SRIO Gen2 Endpoint uses AXI4-Stream interfaces for high-throughput data transfer and AXI4-Lite interfaces for the configuration (maintenance) interfaces.

Features

- Designed to *RapidIO Interconnect Specification rev. 2.2*
- Supports 1x, 2x and 4x operation with the ability to train down to 1x from 2x or 4x
- Supports per-lane speeds of 1.25, 2.5, 3.125, 5.0, and 6.25 Gbaud

Logical Layer

- Concurrent Initiator and Target operations
- Doorbell and Message support
- Dedicated port for maintenance transactions
- Simple handshaking mechanism to control data flow using standard AXI4-Lite and AXI4-Stream interfaces
- Programmable source ID on all outgoing packets
- Optional large system support for 16-bit Device IDs

Buffer

- Independently configurable TX and RX Buffer depths of 8, 16, or 32 packets
- Support for independent clocks
- Optional TX Flow Control support

Physical Layer

- Configurable IDLE1/IDLE2 sequence support
- Supports critical request flow
- Support for multicast events

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family[1]	Virtex-7, Kintex-7, Virtex-6
Minimum Supported Speed Grades	See Table 2-1.
Supported User Interfaces	AXI4-Stream, AXI4-Lite
Resources	See Table 2-2.
Provided with Core	
Design Files	Vivado: Encrypted RTL ISE: NGC Netlist
Example Design	Configuration Fabric Design
Test Bench	Verilog
Constraints File	UCF, XDC
Simulation Model	Vivado: Encrypted Verilog ISE: Verilog Structural
Supported S/W Driver	N/A
Tested Design Flows[2]	
Design Entry	ISE v14.2 Vivado Design Suite v2012.2
Simulation[3]	Mentor Graphics ModelSim
Synthesis	XST Vivado Synthesis
Support	
Provided by Xilinx @ www.xilinx.com/support	

1. For a complete listing of supported devices, see the release notes for this core.
2. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.
3. Requires a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator.

Overview

The RapidIO Interconnect Architecture, designed to be compatible with the most popular integrated communications processors, host processors, and networking digital signal processors, is a high-performance, packet-switched, interconnect technology. It addresses the high-performance embedded industry's need for reliability, increased bandwidth, and faster bus speeds in an intra-system interconnect.

The RapidIO standard is defined in three layers: *logical*, *transport* and *physical*. The logical layer defines the overall protocol and packet formats. This is the information necessary for endpoints to initiate and complete a transaction. The transport layer provides the route information necessary for a packet to move from endpoint to endpoint. The physical layer describes the device-level interface specifics such as packet transport mechanisms, flow control, electrical characteristics, and low-level error management. This partitioning provides the flexibility to add new transaction types to the logical specification without requiring modification to the transport or physical layer specifications.

- For more information about the RapidIO core, go to www.xilinx.com/rapidio
- For more information about the RapidIO standards and specifications, go to www.rapidio.org

System Overview

The SRIO Gen2 Endpoint is comprised of the following:

- A Serial RapidIO Gen2 top-level wrapper (<srio_component_name>) containing:
    - Serial RapidIO Gen2 Physical Layer (PHY)
    - Serial RapidIO Gen2 Logical (I/O) and Transport Layer (LOG)
    - Serial RapidIO Gen2 Buffer Design (BUF)
- Reference design for clocking, resets, and configuration accesses

The SRIO Gen2 Endpoint is shown in Figure 1-1.

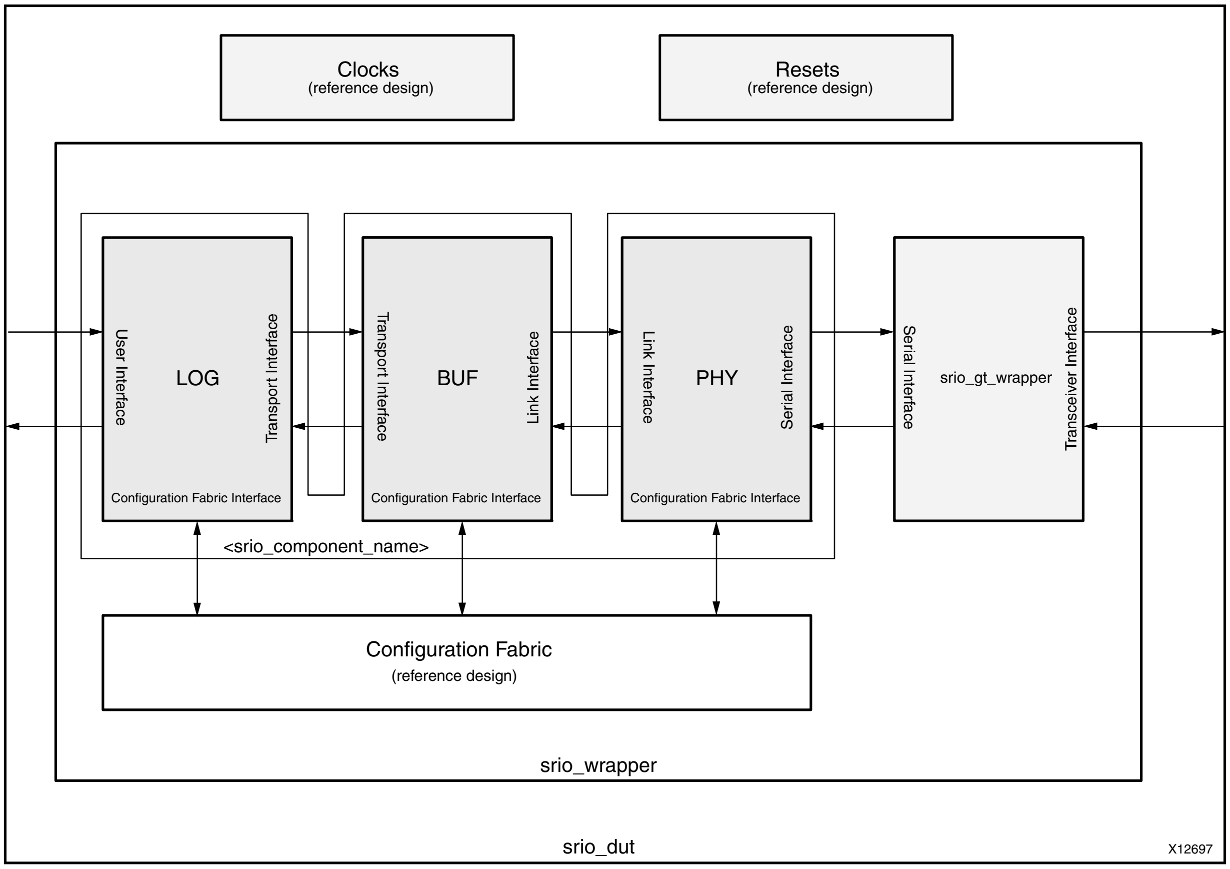


Figure 1-1: **Serial RapidIO System Overview**

The SRIO Gen2 Endpoint is delivered through a layered approach. The <srio_component_name> wrapper contains the LOG, BUF, and PHY. The wrapper presents all the ports from these sub-cores, but ties off any unused ports. This allows customers to use the same wrapper for various configurations of the core, such as the full core or just the PHY.

The *srio_wrapper* integrates the *<srio_component_name>* wrapper, the *srio_gt_wrapper*, and configuration fabric reference design. The *<srio_component_name>* wrapper provides all the ports of the LOG, BUF, and PHY, and the srio_wrapper connects them. Another wrapper level, the srio_dut wrapper, contains the clock and reset modules. These modules are for customers who want to integrate an entire SRIO Gen2 Endpoint into their design.

Although not shown in Figure 1-1, the srio_example_top wrapper includes all the components described previously in addition to an example design. This is used for testing and demonstration purposes, both in simulation and hardware.

Applications

The SRIO Gen2 Endpoint is well suited for control and data operations in communication and embedded systems requiring high-speed I/O with low latency. Typical applications include:

- Wireless Base Stations as interconnect on Channel Cards or Radio Equipment controller
- DSP farms for image & signal processing which is ideal for multi-processor communication interconnect
- Scientific, military, and industrial equipment
- High-availability enterprise storage as reliable, low latency, and high bandwidth memory interface
- Edge Networking for multimedia data compression

Unsupported Features

The following feature is not supported:

- Train down to lane-R (redundant lane). The redundant lane is lane 1 in a x2 configuration, and is lane 2 in a x4 configuration.

Licensing

This Xilinx LogiCORE IP module is provided under the terms of the Xilinx Core License Agreement. The module is shipped as part of the Vivado Design Suite and ISE Design Suite software. For full access to all core features in simulation and in hardware, you must purchase a license for the core. Contact your local Xilinx sales representative for information about pricing and availability.

For more information, please visit the Serial RapidIO Gen2 product page.

Information about other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information on pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

Recommended Design Experience

Although the SRIO Gen2 Endpoint is fully verified, the challenge associated with implementing a complete design varies depending on the configuration and functionality of the application. For best results, previous experience building high performance, pipelined FPGA designs using Xilinx implementation software and user constraints files (UCF) is recommended.

Contact your local Xilinx representative for a closer review and estimation for your specific requirements.

Product Specification

The SRIO Gen2 Endpoint is presented as three sub-cores (provided through the <srio_component_name> wrapper) combined into a single solution using the srio_wrapper module. The wrapper provides a high-level, low maintenance interface for most use models while allowing control of sub-components where necessary.

This chapter gives a basic, functional overview for each sub-core and interface including signal lists and register definitions. Not all the signals listed in the following sections come out of the srio_wrapper.

Standards Compliance

The Serial RapidIO Gen2 Physical Layer (PHY), Serial RapidIO Gen2 Logical Layer (LOG), and Serial RapidIO Gen2 Buffer (BUF) are designed according the *RapidIO Interconnect Specification rev. 2.2 (*RapidIO Specification*)*. Although working knowledge of the RapidIO Specification is not required to use the SRIO Gen2 Endpoint, it may be necessary to reference the specifications for details outside of the scope of this guide. This guide references portions of the RapidIO Specification when necessary.

The RapidIO Specifications can be found at www.rapidio.org/specs/current. The following is a list of the chapters of the *RapidIO Interconnect Specification rev 2.2* specification that directly relate to the SRIO Gen2 Endpoint:

- ***Part 1: Input/Output System Logical*** – Specifies functionality of the Serial RapidIO Gen2 Logical (I/O) and Transport Layer.
- ***Part 2: Message Passing Logical*** – Specifies functionality of the Serial RapidIO Gen2 Logical (I/O) and Transport Layer when Doorbell and Message parsing is enabled.
- ***Part 3: Common Transport*** – Specifies functionality of the Serial RapidIO Gen2 Logical (I/O) and Transport Layer.

Part 6: Serial Physical Layer – Specifies functionality of the Serial RapidIO Gen2 Physical Layer and the Serial RapidIO Gen2 Buffer.

Performance

Table 2-1 shows the minimum speed grades for each supported device.

Table 2-1: **Minimum Supported Speed Grade Details**

Link Width	Performance per Lane (Gb/s)	Virtex-6	7 Series
1x	1.25 / 2.5 / 3.125	1	1
	5.0	1	1
	6.25	2	1
2x	1.25 / 2.5 / 3.125	1	1
	5.0	1	1
	6.25	2	1
4x	1.25 / 2.5 / 3.125	1	1
	5.0	2	2
	6.25	3	3

Resource Utilization

Resources required for the SRIO Gen2 Endpoint have been estimated in Table 2-2. These values were generated using the Xilinx CORE Generator™ tools v13.4. They are derived from post-synthesis reports, and might change during MAP and PAR.

Table 2-2: **Core Resources Used**

Logical Layer (LOG)				
	Virtex-6		7 Series	
LUTs	2400		2350	
Flip Flops	2350		2350	
Slices(1)	1000		1000	
Buffer (BUF)				
	Virtex-6		**7 Series**	
	Min(2)	**Max(3)**	**Min(2)**	**Max(3)**
LUTs	750	750	650 (600)	800
Flip Flops	1100	1200	1150	1200
Slices(1)	400	450	500	500
Block RAM	2	8	2	8

Table 2-2: **Core Resources Used *(Cont'd)***

Physical Layer (PHY)										
	Virtex-6					**7 Series**				
	IDLE1		**IDLE2**		**Both**	**IDLE1**		**IDLE2**		**Both**
	1x	**4x**	**1x**	**4x**	**2x**	**1x**	**4x**	**1x**	**4x**	**2x**
LUTs	2500	3300	3050	5850	4400	2350	3150	3050	5800	4350
Flip Flops	2400	3600	2800	5050	3600	2350	3600	2800	5050	3600
Slices(1)	1200	1700	1650	2700	2000	1300	1850	1550	2700	2000
BUFG	4	4	4	4	4	5(4)	5(4)	5(4)	5(4)	5(4)
Transceivers	1	4	1	4	2	1	4	1	4	2

Configuration Fabric Reference Design		
	Virtex-6	**7 Series**
LUTs	150	150
Flip Flops	150	150
Slices(1)	100	100

Other		
	Virtex-6	**7 Series**
DCM/PLL/ MMCM	1	1

1. Slice count values are only an estimate. The exact number of slices depends on user core configuration and the level of resource sharing with adjacent logic.
2. Buffer configured with 8-deep Tx and Rx buffers, synchronous clock support, and Rx flow control.
3. Buffer configured with 32-deep Tx and Rx buffers, asynchronous clock support, and Tx/Rx flow control.
4. For certain configurations, an additional BUFG is used for Kintex-7 Initial ES Silicon.

Top-Level Wrapper

The srio_wrapper module (`srio_wrapper.v`) bundles each component of the SRIO Gen2 Endpoint, including the reference design, to provide a packaged solution around which to design. Figure 2-1 provides a basic block diagram of how each piece fits into the wrapper module, and a general view of the data interaction between each piece of the srio_wrapper design.

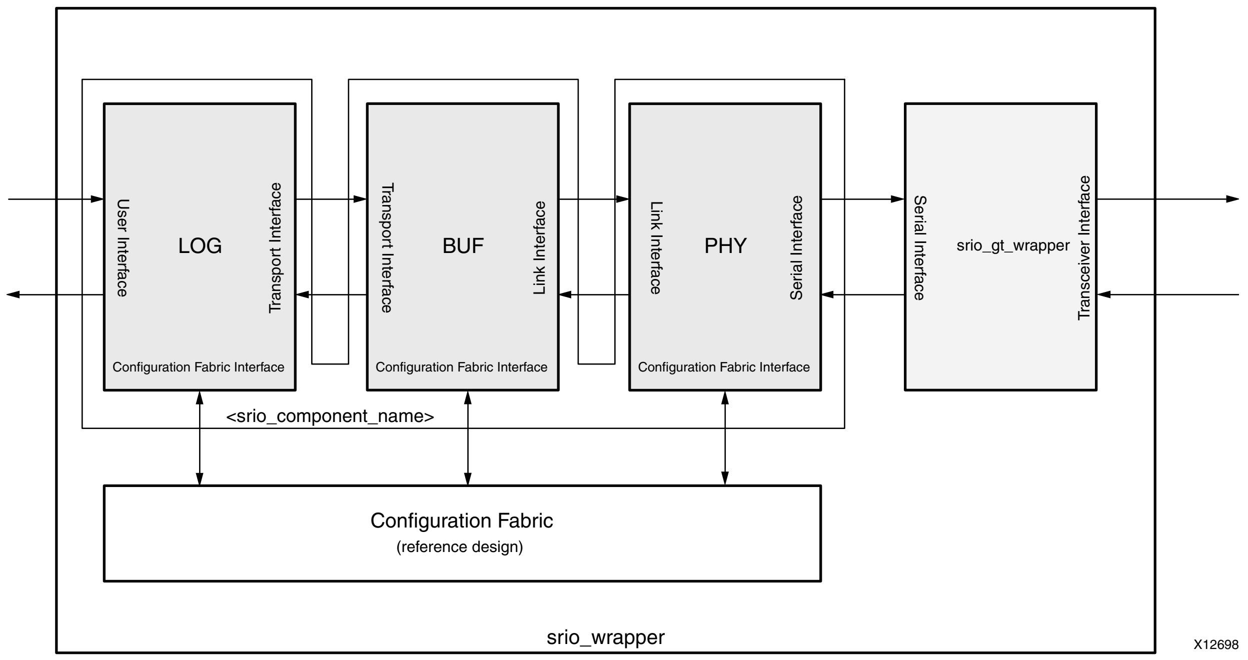


Figure 2-1: **Top-Level Wrapper Block Diagram**

Port Descriptions

This section details the interfaces on each of the three sub-cores of the SRIO Gen2 Endpoint, and the interfaces for the modules in the reference design.

Logical Layer (LOG) Interfaces

The Logical Layer (LOG) is partitioned into several modules that control the concatenation and parsing of transmit and receive packets. The LOG has three interfaces:

- User Interface
- Transport Interface
- Configuration Fabric Interface

Figure 2-2 shows the ports associated with each of the LOG's interfaces. In Figure 2-2, solid arrowheads represent AXI4-Stream ports, and open arrowheads represent AXI4-Lite ports.

Note: Port names and descriptions are from the LOG point of view.

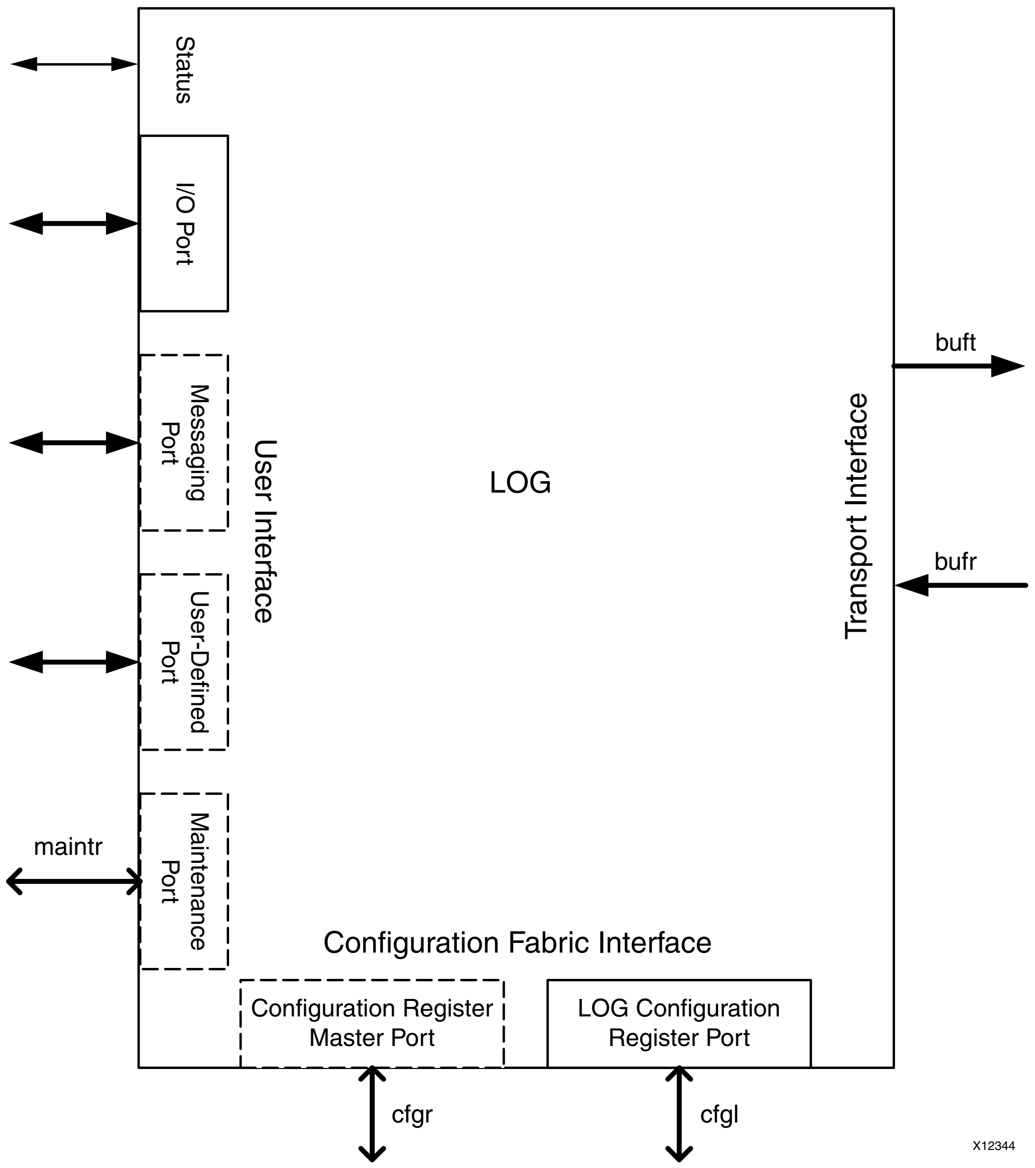


Figure 2-2: **Logical Layer (LOG) Interfaces**

The User Interface contains ports from which a packet can be issued or consumed. The number of ports and the transaction types associated with each port can be configured when the core is generated. Configuration read and write accesses can also be initiated from these ports to the configuration registers that reside in this SRIO Gen2 Endpoint device or to a remote device. These interfaces are fed out through the Serial RapidIO wrapper and are used for packet generation and consumption.

The Transport Interface contains two ports, Receive and Transmit, and is designed to be connected with a RapidIO-compliant Physical Layer or buffering application. This interface is invisible from outside the wrapper module.

The Configuration Fabric Interface contains two ports: the Configuration Master port and the LOG Configuration Register port. The Configuration Master issues reads and writes to the local configuration space via the Configuration Fabric. The LOG Configuration Register port is a slave interface for reads and writes to any configuration registers that are defined as part of the Logical or Transport Layers. The Configuration Fabric decodes the address of reads or writes from the Configuration Bus Master and passes them to the Configuration Register Ports of the LOG, PHY and BUF. This interaction is kept completely within the *srio_wrapper* module. If the user chooses to directly access the maintenance traffic by selecting the AXI4-Stream interface instead of the AXI4-Lite interface, the Configuration Register Master port will not exist.

Clock and Reset Interface

Table 2-3 lists the signals associated with the clock and resets for the LOG layer.

Table 2-3: **LOG Clock and Reset Interface Signal List**

Signal	Directio n	Description
log_lcl_log_clk	Output	Clock for the LOG. In example design, log_clk depends on line rate and link width (a core trained down from Nx to 1x still uses the Nx clock rate). See Clocking in Chapter 3 for more information. ***Note:*** This signal is the same as log_clk, which is the name used in other areas of this document.
log_rst	Input	Reset for LOG. Must deassert synchronously to log_clk. See Resets in Chapter 3.
log_lcl_cfg_clk	Input	Configuration Register Interface clock. If the AXI4-Lite Maintenance Port and the Configuration Fabric reference design are in use, this must be equivalent to log_clk. Otherwise, this clock is independent of log_clk. ***Note:*** This signal is the same as cfg_clk, which is the name used in other areas of this document.
log_lcl_cfg_rst	Input	Configuration Register Interface reset. Clears LOG registers to default values. Must deassert synchronously to cfg_clk. ***Note:*** This signal is the same as cfg_rst, which is the name used in other areas of this document.

User Interfaces

The User Interface contains a set of I/O ports and the following optional ports:

- Messaging Port
- Maintenance Port
- User-Defined Port

These interfaces are available from the srio_wrapper level. Each transaction type is assigned to a particular port. Typically, any supported I/O transactions such as NWRITEs, NWRITE_Rs, SWRITEs, NREADs, and RESPONSEs (not including MAINTENANCE responses) are transmitted or received on the I/O port. MESSAGE transactions (if supported) can be assigned to either the Messaging port or the I/O port. DOORBELL transactions use the I/O port regardless of whether the Messaging port is present. If the Maintenance port is enabled, all Maintenance packets are expected on the Maintenance port. If a transaction is User-Defined, an unsupported type, or does not have an assigned port, it will use the User-Defined port (when the User-Defined port is disabled, received packets that do not correspond to another port will be dropped).

I/O Port

The I/O port can be configured in one of two styles: Condensed I/O or Initiator/Target. The signals available depend on the style selected during core generation.

The I/O port is built from AXI4-Stream channels. Two packet formats are available: HELLO or SRIO Stream. All channels in the I/O port must use the same packet format, which is selected when the core is generated. See Chapter 3, Designing with the Core for more information on port usage.

Condensed I/O

The Condensed I/O port style reduces the number of channels used to transmit and receive I/O packets. There is one AXI4-Stream channel used to transmit all packet types associated with the I/O port (`iotx`). Similarly, there is one channel used for all received I/O port packets (`iorx`). Figure 2-3 shows the Condensed I/O port.

Note: Port names and descriptions are from the LOG point of view.

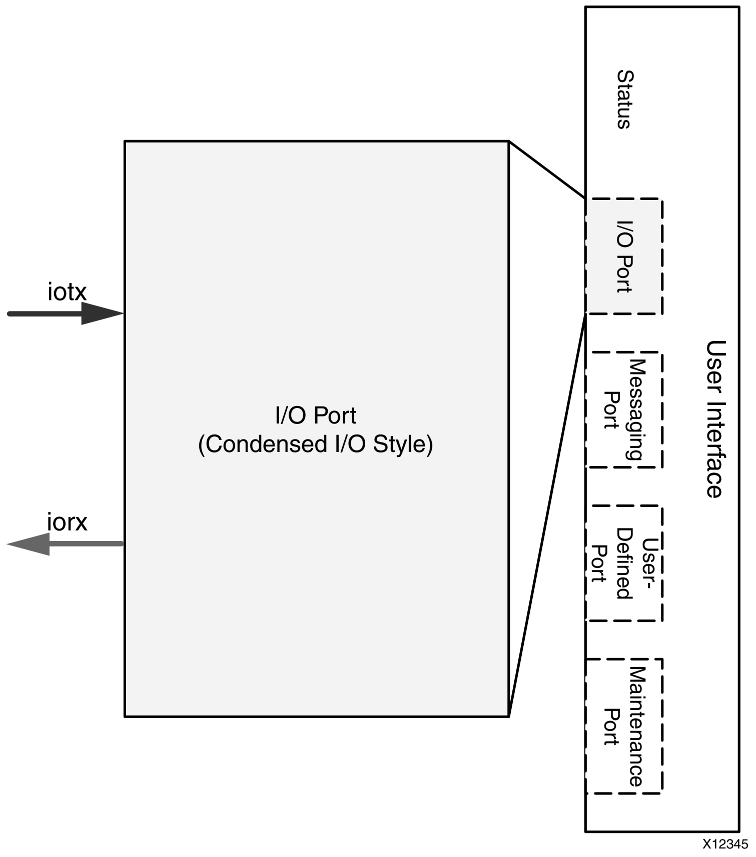


Figure 2-3: **Condensed I/O Port**

Table 2-4 lists the signals associated with the Condensed I/O port.

Table 2-4: **Condensed I/O Port Signal List**

Signal	Direction	Description
s_axis_iotx_tvalid	Input	Indicates that the information on the channel is valid.
s_axis_iotx_tready	Output	Handshaking signal. Indicates that the data from the source is accepted (if valid).
s_axis_iotx_tdata[63:0]	Input	Packet header and data.
s_axis_iotx_tkeep[7:0]	Input	Byte qualifier that indicates whether the content of the associated byte of data is valid. If port is configured to use HELLO format, this must be tied to 8'hFF. For ports configured to use SRIO Stream format, this input should be set to 8'hFF except when tlast is asserted. Bit 7 corresponds to the most significant byte of data (tdata[63:56]), and bit 0 corresponds to the least significant byte (tdata[7:0]).
s_axis_iotx_tlast	Input	Indicates the last beat of a packet.

Table 2-4: **Condensed I/O Port Signal List *(Cont'd)***

Signal	Direction	Description
s_axis_iotx_tuser[31:0]	Input	HELLO Format: Valid on the first beat of a packet, this signal consists of the Source ID (31:16) and Destination ID (15:0) for the packet. If using 8-bit Device IDs, the most significant byte of each ID should be padded with 0's. The Source ID portion of the signal will be tied to the deviceid signal within the srio_wrapper module. SRIO Stream Format: In this format, tuser will only be 8 bits wide. Bit 1 is used to set the Critical Request Flow (CRF) flag for the packet, and should be tied to zero if CRF support is disabled. All other bits are reserved. On subsequent beats within a packet, this field is reserved.
m_axis_iorx_tvalid	Output	Indicates that the information on the channel is valid.
m_axis_iorx_tready	Input	Handshaking signal. Indicates that the data from the source is accepted (if valid).
m_axis_iorx_tdata[63:0]	Output	Packet header and data.
m_axis_iorx_tkeep[7:0]	Output	Byte qualifier that indicates whether the content of the associated byte of data is valid. If port is configured to use HELLO format, this will be tied to 8'hFF. For ports configured to use SRIO Stream format, this output will be set to 8'hFF except when tlast is asserted. Bit 7 corresponds to the most significant byte of data (tdata[63:56]) and bit 0 corresponds to the least significant byte (tdata[7:0]).
m_axis_iorx_tlast	Output	Indicates the last beat of a packet.
m_axis_iorx_tuser[31:0]	Output	HELLO Format: Valid on the first beat of a packet, this signal consists of the Source ID (31:16) and Destination ID (15:0) for the packet. If using 8-bit Device IDs, the most significant byte of each ID will be padded with 0's. SRIO Stream Format: In this format, tuser will only be 8 bits wide. Bit 1 will be set if the Critical Request Flow (CRF) flag for the packet was set. All other bits are reserved. On subsequent beats within a packet, this field is reserved.

Initiator/Target

The Initiator/Target port style allows separation of transactions intended for remote devices (placed on Initiator port) from transactions targeting the local endpoint (placed on Target port).

As shown in Figure 2-4, there are four AXI4-Stream channels for I/O transactions when using the Initiator/Target port style. In Figure 2-4, request channels are shown in black and response channels in grey.

Note: Port names and descriptions are from the LOG point of view.

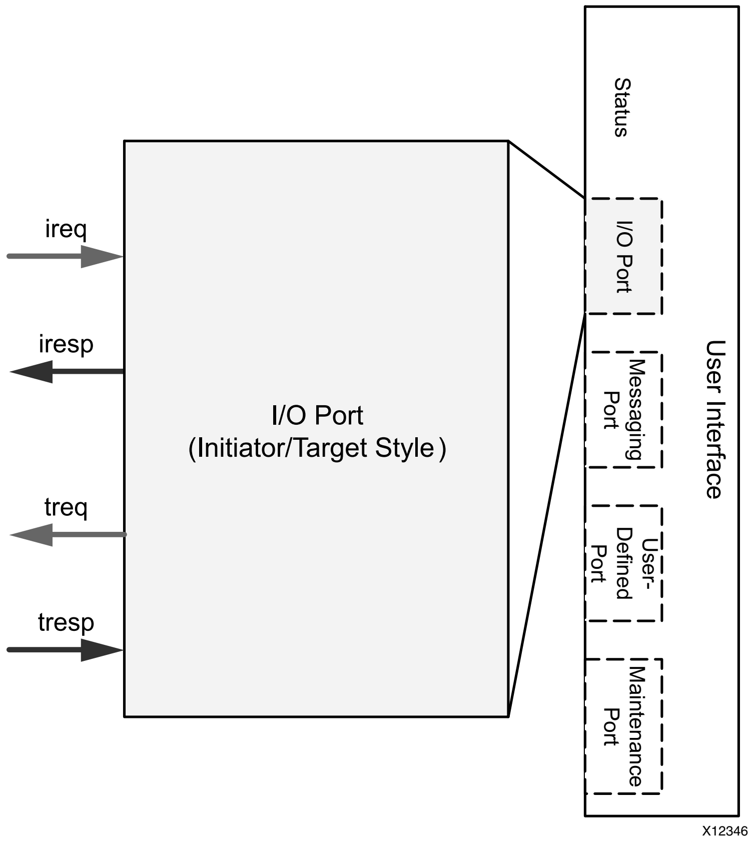


Figure 2-4: **Initiator/Target Port**

Requests generated by the local endpoint are placed on the Initiator Request (`ireq`) channel to be transmitted on the link. Responses received from a remote device are presented on to the user the Initiator Response (`iresp`) channel.

Requests originating from a remote device which are received by the core are presented to the user on the Target Request (`treq`) channel. Responses to said requests, which are generated by the user, are placed on the Target Response (`tresp`) channel.

Table 2-5 shows the signals associated with the Initiator/Target port.

Table 2-5: **Initiator/Target Port Signal List**

Signal	Direction	Description
s_axis_ireq_tvalid	Input	Indicates that the information on the interface is valid.
s_axis_ireq_tready	Output	Handshaking signal. Indicates that the data from the source is accepted (if valid).
s_axis_ireq_tdata[63:0]	Input	Packet header and data.

Table 2-5: **Initiator/Target Port Signal List *(Cont'd)***

Signal	Direction	Description
s_axis_ireq_tkeep[7:0]	Input	Byte qualifier that indicates whether the content of the associated byte of data is valid. If port is configured to use HELLO format, this must be tied to 8'hFF. For ports configured to use SRIO Stream format, this input should be set to 8'hFF except when tlast is asserted. Bit 7 corresponds to the most significant byte of data (tdata[63:56]), and bit 0 corresponds to the least significant byte (tdata[7:0]).
s_axis_ireq_tlast	Input	Indicates the last beat of a packet.
s_axis_ireq_tuser[31:0]	Input	HELLO Format: Valid on the first beat of a packet, this signal consists of the Source ID (31:16) and Destination ID (15:0) for the packet. If using 8-bit Device IDs, the most significant byte of each ID should be padded with 0's. The Source ID portion of the signal will be tied to the deviceid signal within the srio_wrapper module. SRIO Stream Format: In this format, tuser will only be 8 bits wide. Bit 1 is used to set the Critical Request Flow (CRF) flag for the packet, and should be tied to zero if CRF support is disabled. All other bits are reserved. On subsequent beats within a packet, this field is reserved.
m_axis_iresp_tvalid	Output	Indicates that the information on the interface is valid.
m_axis_iresp_tready	Input	Handshaking signal. Indicates that the data from the source is accepted (if valid).
m_axis_iresp_tdata[63:0]	Output	Packet header and data.
m_axis_iresp_tkeep[7:0]	Output	Byte qualifier that indicates whether the content of the associated byte of data is valid. If port is configured to use HELLO format, this will be tied to 8'hFF. For ports configured to use SRIO Stream format, this output will be set to 8'hFF except when tlast is asserted. Bit 7 corresponds to the most significant byte of data (tdata[63:56]) and bit 0 corresponds to the least significant byte (tdata[7:0]).
m_axis_iresp_tlast	Output	Indicates the last beat of a packet.
m_axis_iresp_tuser[31:0]	Output	HELLO Format: Valid on the first beat of a packet, this signal consists of the Source ID (31:16) and Destination ID (15:0) for the packet. If using 8-bit Device IDs, the most significant byte of each ID should be padded with 0's. SRIO Stream Format: In this format, tuser will only be 8 bits wide. Bit 1 will be set if the Critical Request Flow (CRF) flag for the packet was set. All other bits are reserved. On subsequent beats within a packet, this field is reserved.
m_axis_treq_tvalid	Output	Indicates that information on the interface is valid.
m_axis_treq_tready	Input	Handshaking signal. Indicates that the data from the source is accepted (if valid).
m_axis_treq_tdata[63:0]	Output	Packet header and data.

Table 2-5: **Initiator/Target Port Signal List *(Cont'd)***

Signal	Direction	Description
m_axis_treq_tkeep[7:0]	Output	Byte qualifier that indicates whether the content of the associated byte of data is valid. If port is configured to use HELLO format, this will be tied to 8'hFF. For ports configured to use SRIO Stream format, this output will be set to 8'hFF except when tlast is asserted. Bit 7 corresponds to the most significant byte of data (tdata[63:56]), and bit 0 corresponds to the least significant byte (tdata[7:0]).
m_axis_treq_tlast	Output	Indicates the last beat of a packet.
m_axis_treq_tuser[31:0]	Output	HELLO Format: Valid on the first beat of a packet, this signal consists of the Source ID (31:16) and Destination ID (15:0) for the packet. If using 8-bit Device IDs, the most significant byte of each ID should be padded with 0's. SRIO Stream Format: In this format, tuser will only be 8 bits wide. Bit 1 will be set if the Critical Request Flow (CRF) flag for the packet was set. All other bits are reserved. On subsequent beats within a packet, this field is reserved.
s_axis_tresp_tvalid	Input	Indicates that the information on the interface is valid.
s_axis_tresp_tready	Output	Handshaking signal. Indicates that the data from the source is accepted (if valid).
s_axis_tresp_tdata[63:0]	Input	Packet header and data.
s_axis_tresp_tkeep[7:0]	Input	Byte qualifier that indicates whether the content of the associated byte of data is valid. If port is configured to use HELLO format, this must be tied to 8'hFF. For ports configured to use SRIO Stream format, this input should be set to 8'hFF except when tlast is asserted. Bit 7 corresponds to the most significant byte of data (tdata[63:56]) and bit 0 corresponds to the least significant byte (tdata[7:0]).
s_axis_tresp_tlast	Input	Indicates the last beat of a packet.
s_axis_tresp_tuser[31:0]	Input	HELLO Format: Valid on the first beat of a packet, this signal consists of the Source ID (31:16) and Destination ID (15:0) for the packet. If using 8-bit Device IDs, the most significant byte of each ID should be padded with 0's. The Source ID portion of the signal will be tied to the deviceid signal within the srio_wrapper module. SRIO Stream Format: In this format, tuser will only be 8 bits wide. Bit 1 is used to set the Critical Request Flow (CRF) flag for the packet, and should be tied to zero if CRF support is disabled. All other bits are reserved. On subsequent beats within a packet, this field is reserved.

Messaging Port

The messaging port is an optional interface (messages can also be combined onto the I/O Ports, treated as write transactions, via CORE Generator settings). A separate Messaging port follows the Initiator/Target style.

The Initiator/Target port style allows separation of transactions targeting remote devices from transactions targeting the local endpoint. Figure 2-5 details the Messaging port.

Note: Port names and descriptions are from the LOG point of view.

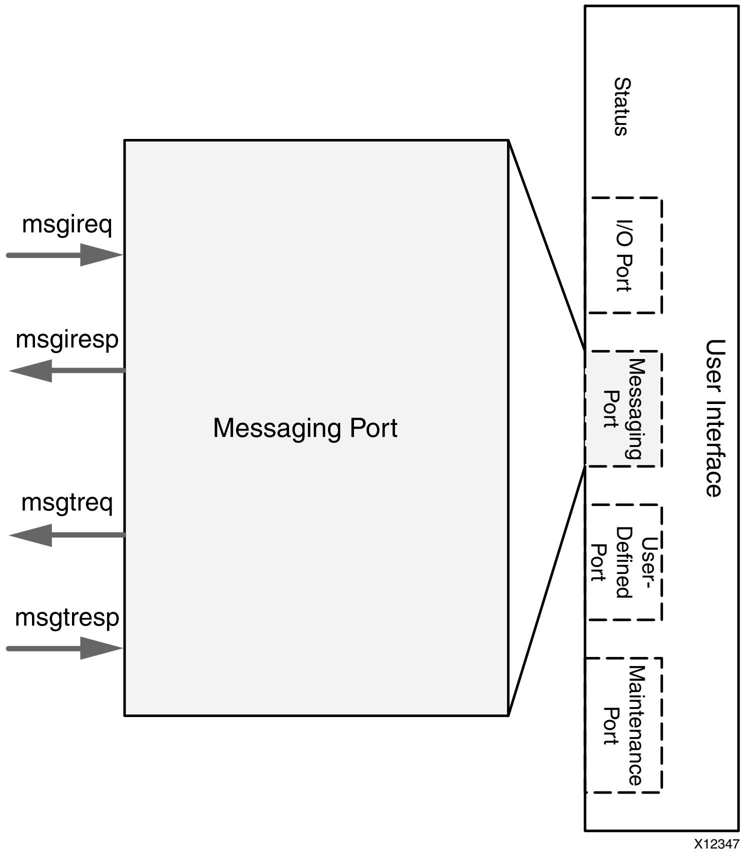


Figure 2-5: **Messaging Port**

Requests generated by the local endpoint are placed on the Message Initiator Request (`msgireq`) port to be transmitted on the link. Responses received from a remote device are presented on the Message Initiator Response (`msgiresp`) port.

Requests originating from a remote device that are received by the Serial RapidIO core are presented on the Message Target Request (`msgtreq`) port. Responses to these requests, which are generated by the user, are placed on the Message Target Response (`msgtresp`) port.

Table 2-6 shows the signals associated with the Messaging port.

Table 2-6: **Messaging Port Signal List**

Signal	Direction	Description
s_axis_msgireq_tvalid	Input	Indicates that the information on the interface is valid.
s_axis_msgireq_tready	Output	Handshaking signal. Indicates that the data from the source is accepted (if valid).

Table 2-6: **Messaging Port Signal List *(Cont'd)***

Signal	Direction	Description
s_axis_msgireq_tdata[63:0]	Input	Packet header and data.
s_axis_msgireq_tkeep[7:0]	Input	Byte qualifier that indicates whether the content of the associated byte of data is valid. For HELLO ports, this must be tied to 8'hFF.
s_axis_msgireq_tlast	Input	Indicates the last beat of a packet.
s_axis_msgireq_tuser[31:0]	Input	On the first beat of a packet, this signal consists of the Source ID (31:16) and Destination ID (15:0) for the packet. If using 8-bit Device IDs, the most significant byte of each ID should be padded with 0s. On subsequent beats within a packet, this field is reserved.
m_axis_msgiresp_tvalid	Output	Indicates that the information on the interface is valid.
m_axis_msgiresp_tready	Input	Handshaking signal. Indicates that the data from the source is accepted (if valid).
m_axis_msgiresp_tdata[63:0]	Output	Packet header and data.
m_axis_msgiresp_tkeep[7:0]	Output	Byte qualifier that indicates whether the content of the associated byte of data is valid. For HELLO ports, this must be tied to 8'hFF.
m_axis_msgiresp_tlast	Output	Indicates the last beat of a packet.
m_axis_msgiresp_tuser[31:0]	Output	On the first beat of a packet, this signal consists of the Source ID (31:16) and Destination ID (15:0) for the packet. If using 8-bit Device IDs, the most significant byte of each ID should be padded with 0s. On subsequent beats within a packet, this field is reserved.
m_axis_msgtreq_tvalid	Output	Indicates that the information on the interface is valid.
m_axis_msgtreq_tready	Input	Handshaking signal. Indicates that the data from the source is accepted (if valid).
m_axis_msgtreq_tdata[63:0]	Output	Packet header and data.
m_axis_msgtreq_tkeep[7:0]	Output	Byte qualifier that indicates whether the content of the associated byte of data is valid. For HELLO ports, this must be tied to 8'hFF.
m_axis_msgtreq_tlast	Output	Indicates the last beat of a packet.
m_axis_msgtreq_tuser[31:0]	Output	On the first beat of a packet, this signal consists of the Source ID (31:16) and Destination ID (15:0) for the packet. If using 8-bit Device IDs, the most significant byte of each ID should be padded with 0s. On subsequent beats within a packet, this field is reserved.
s_axis_msgtresp_tvalid	Input	Indicates that the information on the interface is valid.
s_axis_msgtresp_tready	Output	Handshaking signal. Indicates that the data from the source is accepted (if valid).
s_axis_msgtresp_tdata[63:0]	Input	Packet header and data.

Table 2-6: **Messaging Port Signal List *(Cont'd)***

Signal	Direction	Description
s_axis_msgtresp_tkeep[7:0]	Input	Byte qualifier that indicates whether the content of the associated byte of data is valid. For HELLO ports, this must be tied to 8'hFF.
s_axis_msgtresp_tlast	Input	Indicates the last beat of a packet.
s_axis_msgtresp_tuser[31:0]	Input	On the first beat of a packet, this signal consists of the Source ID (31:16) and Destination ID (15:0) for the packet. If using 8-bit Device IDs, the most significant byte of each ID should be padded with 0s. On subsequent beats within a packet, this field is reserved.

User-Defined Port

The User-Defined port is an optional port and has two AXI4-Stream channels, where one channel is used for the transmit direction and one channel is used for the receive direction. The User-Defined port only uses the SRIO Stream format. Figure 2-6 shows the User-Defined port.

Note: Port names and descriptions are from the LOG point of view.

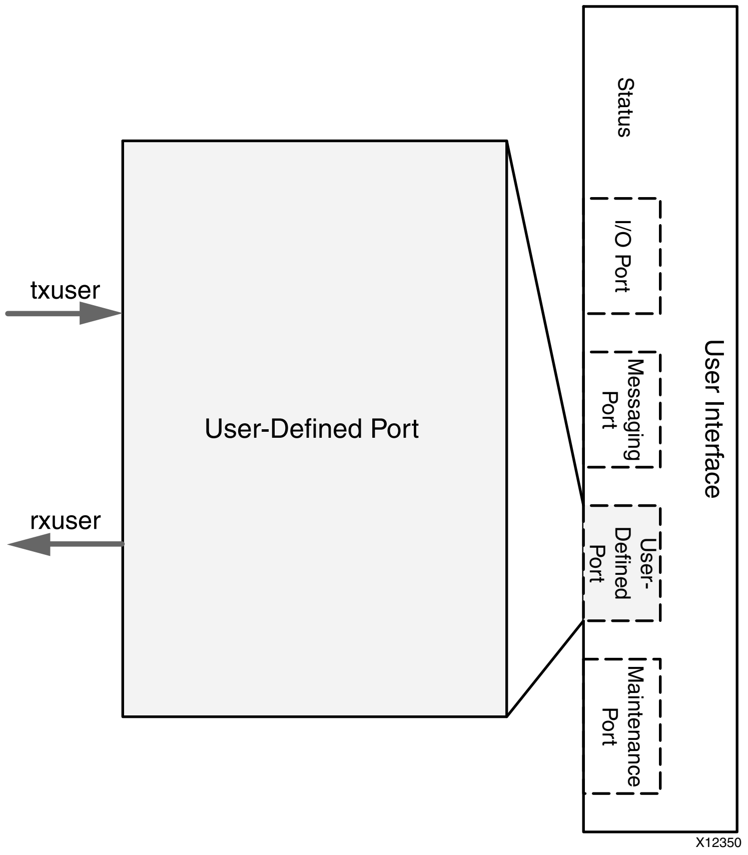


Figure 2-6: **User-Defined Port**

Table 2-7 lists the signals associated with the User-Defined port.

Table 2-7: **User-Defined Port Signal List**

Signal	Direction	Description
s_axis_usrtx_tvalid	Input	Indicates that the information on the channel is valid.
s_axis_usrtx_tready	Output	Handshaking signal. Indicates that the data from the source is accepted (if valid).
s_axis_usrtx_tdata[63:0]	Input	Packet header and data.
s_axis_usrtx_tkeep[7:0]	Input	Byte qualifier that indicates whether the content of the associated byte of data is valid. For SRIO Stream format, this input should be set to 8'hFF except when tlast is asserted. Bit 7 corresponds to the most significant byte of data (tdata[63:56]) and bit 0 corresponds to the least significant byte (tdata[7:0]).
s_axis_usrtx_tlast	Input	Indicates the last beat of a packet.
s_axis_usrtx_tuser[7:0]	Input	In SRIO Stream Format, tuser will only be 8 bits wide. Bit 1 is used to set the Critical Request Flow (CRF) flag for the packet, and should be tied to zero if CRF support is disabled. All other bits are reserved. On subsequent beats within a packet, this field is reserved.
m_axis_usrrx_tvalid	Output	Indicates that the information on the channel is valid.
m_axis_usrrx_tready	Input	Handshaking signal. Indicates that the data from the source is accepted (if valid).
m_axis_usrrx_tdata[63:0]	Output	Packet header and data.
m_axis_usrrx_tkeep[7:0]	Output	Byte qualifier that indicates whether the content of the associated byte of data is valid. For SRIO Stream format, this output will be set to 8'hFF except when tlast is asserted. Bit 7 corresponds to the most significant byte of data (tdata[63:56]) and bit 0 corresponds to the least significant byte (tdata[7:0]).
m_axis_usrrx_tlast	Output	Indicates the last beat of a packet.
m_axis_usrrx_tuser[7:0]	Output	In SRIO Stream Format, tuser will only be 8 bits wide. Bit 1 will be set if the Critical Request Flow (CRF) flag for the packet was set. All other bits are reserved. On subsequent beats within a packet, this field is reserved.

Maintenance Port

If the Maintenance port is enabled, it can be configured to be used as an AXI4-Lite Interface (recommended), or as AXI4-Stream. The AXI4-Lite interface allows the user application to target either the local or remote configuration space, whereas the AXI4-Stream port only provides access to remote transactions. In addition, the AXI4-Stream requires additional user logic in order to forward received maintenance requests to the configuration space.

AXI4-Lite Maintenance Port

Figure 2-7 shows the AXI4-Lite channels associated with the AXI4-Lite Maintenance port. Requests are communicated on the channels shown in black, and AXI4-Lite responses are returned on the channels shown in grey.

Note: Port names and descriptions are from the LOG point of view.

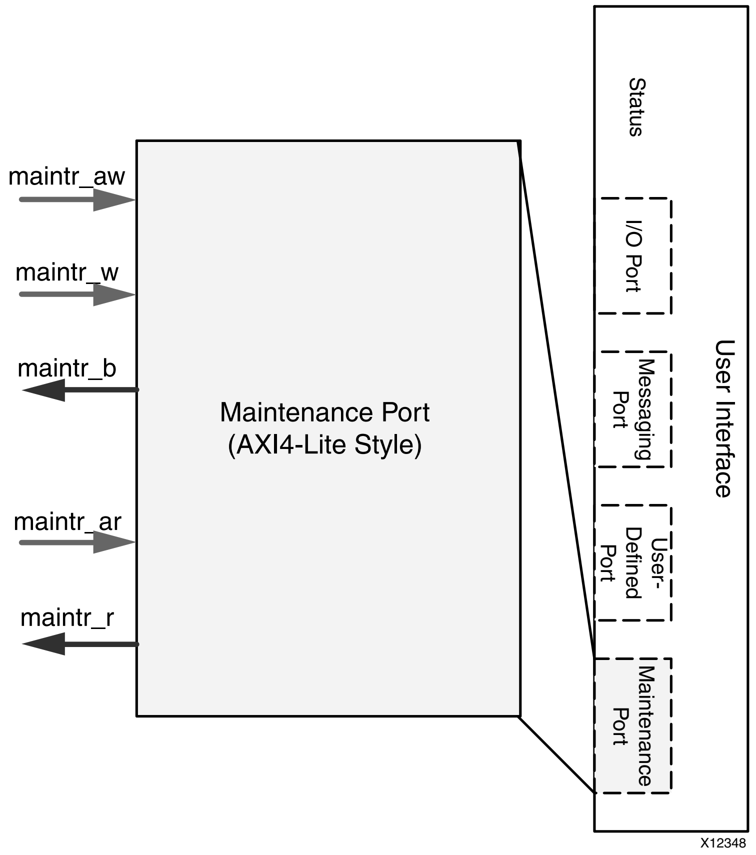


Figure 2-7: **AXI4-Lite Maintenance Port**

Each channel has an independent ready/valid handshake. The signal list for the Maintenance port is shown in Table 2-8.

Table 2-8: **Maintenance Port Signal List**

Signal	Direction	Description
s_axi_maintr_rst	Input	Reset for the maintr interface. Clears outstanding packets. Intended to be used for link timeouts.
s_axi_maintr_awvalid	Input	Indicates that the write address is valid.
s_axi_maintr_awready	Output	Handshaking signal. Indicates that the write address is accepted (if valid).
s_axi_maintr_awaddr[31:0]	Input	Write address. • [31:24] - Hop count + 1 (8'h00 for local write) • [23:0] - Configuration offset for write

Table 2-8: **Maintenance Port Signal List**

Signal	Direction	Description
s_axi_maintr_wvalid	Input	Indicates that the write data is valid.
s_axi_maintr_wready	Output	Handshaking signal. Indicates that the write data is accepted (if valid).
s_axi_maintr_wdata[31:0]	Input	Write data.
s_axi_maintr_bvalid	Output	Indicates that the write response is valid.
s_axi_maintr_bready	Input	Handshaking signal. Indicates that the write response is accepted (if valid).
s_axi_maintr_bresp[1:0]	Output	Write response. • 2'b00 - OK • 2'b10 - Err • 2'bx1 - Reserved
s_axi_maintr_arvalid	Input	Indicates that the read address is valid.
s_axi_maintr_arready	Output	Handshaking signal. Indicates that the read address is accepted (if valid).
s_axi_maintr_araddr[31:0]	Input	Read address. • [31:24] - Hop count + 1 (8'h00 for local read) • [23:0] - Configuration offset for read
s_axi_maintr_rvalid	Output	Indicates that the read response is valid.
s_axi_maintr_rready	Input	Handshaking signal. Indicates that the read response is accepted (if valid).
s_axi_maintr_rresp[1:0]	Output	Read response. • 2'b00 - OK • 2'b10 - Err • 2'bx1 - Reserved
s_axi_maintr_rdata[31:0]	Output	Read Data.

Status

Table 2-9 lists the signals that provide status information to the user.

Table 2-9: **Status Signal List**

Signal	Direction	Description
deviceid[15:0]	Output	Current value stored in the Base DeviceID CSR (offset 0x60).
port_decode_error	Output	Indicates an usupported transaction was received, and was dropped because the User-Defined port was not enabled.

Configuration Fabric Interface

The Configuration Fabric Interface contains two ports: the Configuration Master port and the LOG Configuration Register port. The Configuration Master issues reads and writes to the local configuration space (for the LOG, BUF, and PHY) via the Configuration Fabric. The

LOG Configuration Register port is a slave interface for reads and writes to the registers defined as part of the Logical or Transport Layers. Table 2-10 lists the signals associated with the Configuration Master and LOG Configuration Register ports.

Table 2-10: **LOG Configuration Fabric Interface Signal List**

Signal	Direction	Description
s_axi_cfgl_awvalid	Input	Indicates that the write address is valid.
s_axi_cfgl_awready	Output	Handshaking signal. Indicates that the write address is accepted (if valid).
s_axi_cfgl_awaddr[23:0]	Input	Write address.
s_axi_cfgl_wvalid	Input	Indicates that the write data is valid.
s_axi_cfgl_wready	Output	Handshaking signal. Indicates that the write data is accepted (if valid).
s_axi_cfgl_wdata[31:0]	Input	Write data.
s_axi_cfgl_wstrb[3:0]	Input	Byte qualifier that indicates whether the content of the associated byte of data is valid.
s_axi_cfgl_bvalid	Output	Indicates that the write response is valid.
s_axi_cfgl_bready	Input	Handshaking signal. Indicates that the write response is accepted (if valid).
s_axi_cfgl_arvalid	Input	Indicates that the read address is valid.
s_axi_cfgl_arready	Output	Handshaking signal. Indicates that the read address is accepted (if valid).
s_axi_cfgl_araddr[23:0]	Input	Read address.
s_axi_cfgl_rvalid	Output	Indicates that the read response is valid.
s_axi_cfgl_rready	Input	Handshaking signal. Indicates that the read response is accepted (if valid).
s_axi_cfgl_rdata[31:0]	Output	Read Data.
m_axi_cfgr_awvalid	Output	Indicates that the write address is valid.
m_axi_cfgr_awready	Input	Handshaking signal. Indicates that the write address is accepted (if valid).
m_axi_cfgr_awaddr[23:0]	Output	Write address.
m_axi_cfgr_awprot[2:0]	Output	Tied to 0.
m_axi_cfgr_wvalid	Output	Indicates that the write data is valid.
m_axi_cfgr_wready	Input	Handshaking signal. Indicates that the write data is accepted (if valid).
m_axi_cfgr_wdata[31:0]	Output	Write data.
m_axi_cfgr_wstrb[3:0]	Output	Byte qualifier that indicates whether the content of the associated byte of data is valid.
m_axi_cfgr_bvalid	Input	Indicates that the write response is valid.
m_axi_cfgr_bready	Output	Handshaking signal. Indicates that the write response is accepted (if valid).

Table 2-10: **LOG Configuration Fabric Interface Signal List *(Cont'd)***

Signal	Direction	Description
m_axi_cfgr_bresp[1:0]	Input	Write response. • 2'b00 - OK • 2'b10 - Err • 2'bx1 - Reserved
m_axi_cfgr_arvalid	Output	Indicates that the read address is valid.
m_axi_cfgr_arready	Input	Handshaking signal. Indicates that the read address is accepted (if valid).
m_axi_cfgr_araddr[23:0]	Output	Read address.
m_axi_cfgr_arprot[2:0]	Output	Tied to 0.
m_axi_cfgr_rvalid	Input	Indicates that the read response is valid.
m_axi_cfgr_rready	Output	Handshaking signal. Indicates that the read response is accepted (if valid).
m_axi_cfgr_rresp[1:0]	Input	Read response. • 2'b00 - OK • 2'b10 - Err • 2'bx1 - Reserved
m_axi_cfgr_rdata[31:0]	Input	Read Data.

Transport Interface

The Transport Interface contains a transmit and receive port, and connects to either a buffering layer or directly to the RapidIO-compliant Physical Layer. Table 2-11 lists the signals associated with the Transport Interface from the LOG point of view.

Table 2-11: **LOG Transport Interface Signal List**

Signal	Direction	Description
m_axis_buft_tvalid	Output	Indicates that the information on the channel is valid.
m_axis_buft_tready	Input	Handshaking signal. Indicates that the data from the source is accepted (if valid).
m_axis_buft_tdata[63:0]	Output	Packet header and data.
m_axis_buft_tkeep[7:0]	Output	Byte qualifier that indicates whether the content of the associated byte of data is valid. This output will be set to 8'hFF except when tlast is asserted. Bit 7 corresponds to the most significant byte of data (tdata[63:56]) and bit 0 corresponds to the least significant byte (tdata[7:0]).
m_axis_buft_tlast	Output	Indicates the last beat of a packet.
m_axis_buft_tuser[7:0]	Output	On the first beat of a packet, Bit 4 should be set if the transaction is a response. Bit 1 will be set if the Critical Request Flow (CRF) flag for the packet was set. All other bits are reserved. On subsequent beats within a packet, this field is reserved.

Table 2-11: **LOG Transport Interface Signal List**

Signal	Direction	Description
s_axis_bufr_tvalid	Input	Indicates that the information on the channel is valid.
s_axis_bufr_tready	Output	Handshaking signal. Indicates that the data from the source is accepted (if valid).
s_axis_bufr_tdata[63:0]	Input	Packet header and data.
s_axis_bufr_tkeep[7:0]	Input	Byte qualifier that indicates whether the content of the associated byte of data is valid. This input should be set to 8'hFF except when tlast is asserted. Bit 7 corresponds to the most significant byte of data (tdata[63:56]) and bit 0 corresponds to the least significant byte (tdata[7:0]).
s_axis_bufr_tlast	Input	Indicates the last beat of a packet.
s_axis_bufr_tuser[7:0]	Input	On the first beat of a packet, Bit 3 should be set to indicate this is the start of the packet. Bit 1 is used to set the Critical Request Flow (CRF) flag for the packet, and should be tied to zero if CRF support is disabled. All other bits are reserved. On subsequent beats within a packet, this field is reserved.
log_lcl_response_only	Input	This signal indicates that the LOG should only transmit response transactions. If a request is in progress when this signal asserts, the request may be completed. ***Note:*** This signal is the same as response_only, which is the name used in other areas of this document.
log_lcl_maint_only	Input	This signal indicates that the LOG should only transmit maintenance transactions. If a request is in progress when this signal asserts, the request may be completed. ***Note:*** This signal is the same as maint_only, which is the name used in other areas of this document.

Buffer Design (BUF) Interfaces

The Buffer Design (BUF) generated with the PHY is provided to account for transmit and receive packet buffering. The BUF is necessary for guaranteed packet delivery and flow control operations. Xilinx provides a configurable buffer solution that allows for trade-offs between system performance and resource requirements.

The transmit buffer is responsible for queuing outgoing transactions and managing the flow of these packets over the link interface into the PHY. TX and RX Buffer sizes are configurable through the CORE Generator software to values of 8, 16 or 32 packets in depth. The TX buffer is a store and forward buffer designed for a low packet-to-packet latency to maximize streaming throughput. The transmit buffer must hold each packet until it has been successfully received by the link partner device, at which point the packet is released to make room for additional packets. In the case where multiple unsent packets accumulate in the buffer, which often happens when flow control occurs, the BUF will reorder packets based on type and priority with response packets issued first followed by requests. For more information on flow control and packet reordering, see Chapter 3, Designing with the Core.

Additionally the BUF handles clock crossing when necessary. This domain crossing logic is recommended for all multi-lane cores since the PHY clock is dynamic at start-up and during traindown scenarios. This allows user logic to run at a consistent and known rate. Clock domain crossing logic can be optionally added or removed when generating the core through the CORE Generator software.

The receive buffer acts as a FIFO for storing and forwarding data into the LOG receive path. The receive buffer also has domain crossing logic which allows the logical layer/user design and the PHY to run at different clock rates. As with the transmit buffer design, this logic is recommended for multi-lane cores.

All interfaces for the BUF are connected internally to the srio_wrapper module and are invisible from outside the wrapper layer.

Note: Port names and descriptions are from the BUF point of view.

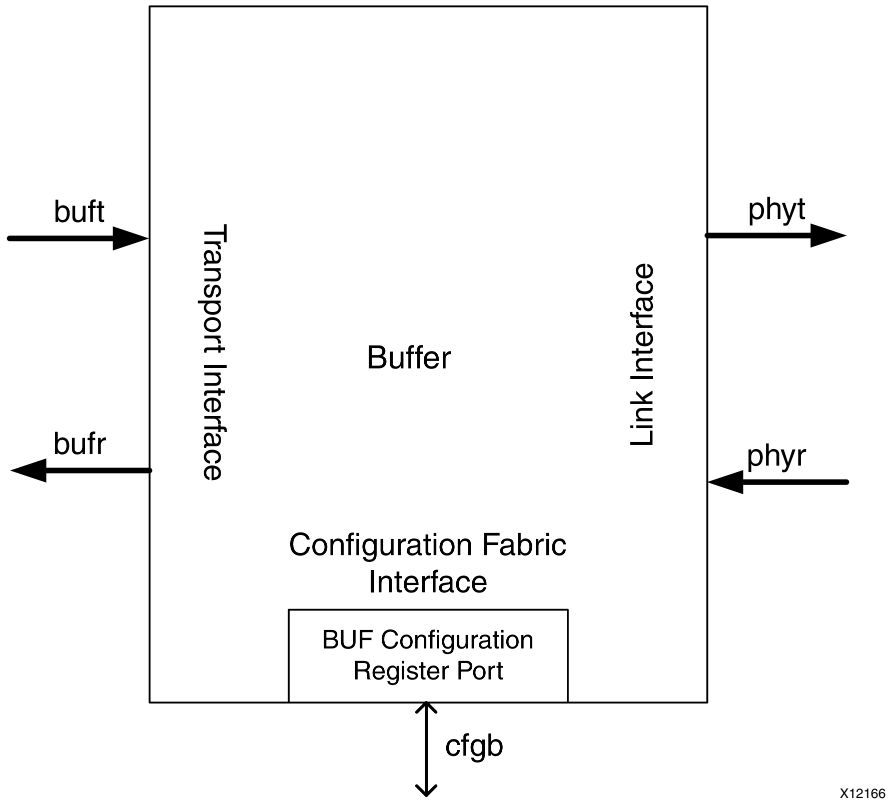


Figure 2-8: **Buffer Layer (BUF) Interfaces**

As shown in Figure 2-8, there are two AXI4-Stream channels in each direction on both the LOG and PHY sides of the BUF. There is also an AXI4-Lite interface to the Configuration Fabric, which allows access to the BUF configuration space.

Clock and Reset Interface

Table 2-12 lists the associated signals for the clock and reset interface for the BUF layer.

Table 2-12: **BUF Clock and Reset Interface Signal List**

Signal	Direction	Description
buf_lcl_log_clk	Input	Clock for the LOG. ***Note:*** This signal is the same as log_clk, which is the name used in other areas of this document.
buf_lcl_phy_clk	Input	Clock for the PHY. ***Note:*** This signal is the same as phy_clk, which is the name used in other areas of this document.
buf_rst	Input	Reset for BUF. Must deassert synchronously to phy_clk. See Resets in Chapter 3 for more information.
buf_lcl_cfg_clk	Input	Configuration Register Interface clock. If the AXI4-Lite Maintenance Port and the Configuration Fabric reference design are in use, this must be equivalent to log_clk. Otherwise, this clock is independent of log_clk. ***Note:*** This signal is the same as cfg_clk, which is the name used in other areas of this document.
buf_lcl_cfg_rst	Input	Configuration Register Interface reset. Clears BUF registers to default values. Must deassert synchronously to cfg_clk. ***Note:*** This signal is the same as cfg_rst, which is the name used in other areas of this document.

Transport Interface

Table 2-13 lists the signals associated with the BUF Transport Interface.

Table 2-13: **BUF Transport Interface Signal List**

Signal	Direction	Description
s_axis_buft_tvalid	Input	Indicates that the information on the channel is valid.
s_axis_buft_tready	Output	Handshaking signal. Indicates that the data from the source is accepted (if valid).
s_axis_buft_tdata[63:0]	Input	Packet header and data.
s_axis_buft_tkeep[7:0]	Input	Byte qualifier that indicates whether the content of the associated byte of data is valid. This output will be set to 8'hFF except when tlast is asserted. Bit 7 corresponds to the most significant byte of data (tdata[63:56]) and bit 0 corresponds to the least significant byte (tdata[7:0]).
s_axis_buft_tlast	Input	Indicates the last beat of a packet.

Table 2-13: **BUF Transport Interface Signal List *(Cont'd)***

Signal	Direction	Description
s_axis_buft_tuser[7:0]	Input	On the first beat of a packet, Bit 4 should be set if the transaction is a response. If Bit 1 is high, the Critical Request Flow (CRF) flag for the packet will be set (if CRF Support is enabled). All other bits are reserved. On subsequent beats within a packet, this field is reserved.
m_axis_bufr_tvalid	Output	Indicates that the information on the channel is valid.
m_axis_bufr_tready	Input	Handshaking signal. Indicates that the data from the source is accepted (if valid).
m_axis_bufr_tdata[63:0]	Output	Packet header and data.
m_axis_bufr_tkeep[7:0]	Output	Byte qualifier that indicates whether the content of the associated byte of data is valid. This input should be set to 8'hFF except when tlast is asserted. Bit 7 corresponds to the most significant byte of data (tdata[63:56]) and bit 0 corresponds to the least significant byte (tdata[7:0]).
m_axis_bufr_tlast	Output	Indicates the last beat of a packet.
m_axis_bufr_tuser[7:0]	Output	On the first beat of a packet, Bit 3 will be set to indicate this is the start of packet. Bit 1 reflects the value of the Critical Request Flow (CRF) flag for the packet, if CRF support is enabled. All other bits are reserved. On subsequent beats within a packet, this field is reserved.
buf_lcl_response_only	Output	This signal indicates that the transmit buffer can only accept responses. If a request is in progress when this signal asserts, the request may be completed. ***Note:*** This signal is the same as response_only, which is the name used in other areas of this document.

Link Interface

Table 2-14 lists the signals associated with the BUF Link Interface.

Table 2-14: **BUF Link Interface Signal List**

Signal	Direction	Description
m_axis_phyt_tvalid	Output	Indicates that the information on the channel is valid.
m_axis_phyt_tready	Input	Handshaking signal. Indicates that the data from the source is accepted (if valid).
m_axis_phyt_tdata[63:0]	Output	Packet header and data.
m_axis_phyt_tkeep[7:0]	Output	Byte qualifier that indicates whether the content of the associated byte of data is valid. This input should be set to 8'hFF except when tlast is asserted. Bit 7 corresponds to the most significant byte of data (tdata[63:56]) and bit 0 corresponds to the least significant byte (tdata[7:0]).
m_axis_phyt_tlast	Output	Indicates the last beat of a packet.

Table 2-14: **BUF Link Interface Signal List *(Cont'd)***

Signal	Direction	Description
m_axis_phyt_tuser[7:0]	Output	On the first beat of a packet, Bit 1 reflects the value of the Critical Request Flow (CRF) flag for the packet, if CRF support is enabled. All other bits are reserved. On the last beat of a packet, Bit 0 is used as a source discontinue. If Bit 0 is high on the tlast transfer, the packet should be discarded.
s_axis_phyr_tvalid	Input	Indicates that the information on the channel is valid.
s_axis_phyr_tready	Output	Handshaking signal. Indicates that the data from the source is accepted (if valid).
s_axis_phyr_tdata[63:0]	Input	Packet header and data.
s_axis_phyr_tkeep[7:0]	Input	Byte qualifier that indicates whether the content of the associated byte of data is valid. This output will be set to 8'hFF except when tlast is asserted. Bit 7 corresponds to the most significant byte of data (tdata[63:56]) and bit 0 corresponds to the least significant byte (tdata[7:0]).
s_axis_phyr_tlast	Input	Indicates the last beat of a packet.
s_axis_phyr_tuser[7:0]	Input	If Bit 1is high on the first beat of a packet, the Critical Request Flow (CRF) flag for the packet will be set (if CRF Support is enabled). All other bits are reserved. On the last beat of a packet, Bit 0 is used as a source discontinue. If Bit 0 is high on the tlast transfer, the packet will be dropped in the BUF.
buf_lcl_master_enable	Input	Reflects the value of the Master Enable bit from the Port General Control CSR in the physical layer configuration space. When set, the BUF can forward requests to the PHY. If this input is low, the BUF can only forward responses. ***Note:*** This signal is the same as master_enable, which is the name used in other areas of this document.
buf_lcl_idle2_selected	Input	When asserted, the PHY is operating in IDLE2 mode and using long control symbols (therefore ackIDs are 6 bits). ***Note:*** This signal is the same as idle2_selected, which is the name used in other areas of this document.
buf_lcl_phy_rewind	Input	This signal indicates that the PHY is requesting a rewind. The BUF must monitor the next_fm signal to determine which packet to transmit next. ***Note:*** This signal is the same as phy_rewind, which is the name used in other areas of this document.
buf_lcl_phy_next_fm	Input	This bus is used to communicate the next packet to send, indexed by ackID. This signal increments for each packet, except in a rewind scenario. ***Note:*** This signal is the same as phy_next_fm, which is the name used in other areas of this document.

Table 2-14: **BUF Link Interface Signal List *(Cont'd)***

Signal	Direction	Description
buf_lcl_phy_last_ack	Input	This bus is used to communicate the ackID for the last acknowledge control symbol received. This frees up slots in the buffer by allowing the BUF to clear the corresponding packet. If this signal increments by more than one, multiple packet may be cleared. ***Note:*** This signal is the same as phy_last_ack, which is the name used in other areas of this document.
buf_lcl_phy_rcvd_buf_stat	Input	This bus is used to communicate the last received buffer status from a status control symbol received from the link partner. ***Note:*** This signal is the same as phy_rcvd_buf_stat, which is the name used in other areas of this document.
buf_lcl_phy_buf_stat	Output	This bus is used to communicate the status of the Receive Buffer. This bus will never equal all ones, which indicates receiver flow control. Nor will the lower five bits be all ones, which may also be perceived as receiver flow control. It will always hold an estimation of number of free buffer locations. When the buffer is very lean (more than 30 free packet locations), the buf_stat value will provide the value of 30. ***Note:*** This signal is the same as phy_buf_stat, which is the name used in other areas of this document.
buf_lcl_tx_flow_control	Output	Indicates that the transmit buffer is in transmitter-controlled flow control mode. ***Note:*** This signal is the same as tx_flow_control, which is the name used in other areas of this document.

BUF Configuration Fabric Interface

Table 2-15 lists the signals associated with the Configuration Fabric Interface.

Note: All the signal names in Table 2-15 are from the BUF point of view.

Table 2-15: **BUF Configuration Fabric Interface Signal List**

Signal	Direction	Description
s_axi_cfgb_awvalid	Input	Indicates that the write address is valid.
s_axi_cfgb_awready	Output	Handshaking signal. Indicates that the write address is accepted (if valid).
s_axi_cfgb_awaddr[23:0]	Input	Write address.
s_axi_cfgb_wvalid	Input	Indicates that the write data is valid.
s_axi_cfgb_wready	Output	Handshaking signal. Indicates that the write data is accepted (if valid).
s_axi_cfgb_wdata[31:0]	Input	Write data.
s_axi_cfgb_wstrb[3:0]	Input	Byte qualifier that indicates whether the content of the associated byte of data is valid.

Table 2-15: **BUF Configuration Fabric Interface Signal List**

Signal	Direction	Description
s_axi_cfgb_bvalid	Output	Indicates that the write response is valid.
s_axi_cfgb_bready	Input	Handshaking signal. Indicates that the write response is accepted (if valid).
s_axi_cfgb_arvalid	Input	Indicates that the read address is valid.
s_axi_cfgb_arready	Output	Handshaking signal. Indicates that the read address is accepted (if valid).
s_axi_cfgb_araddr[23:0]	Input	Read address.
s_axi_cfgb_rvalid	Output	Indicates that the read response is valid.
s_axi_cfgb_rready	Input	Handshaking signal. Indicates that the read response is accepted (if valid).
s_axi_cfgb_rdata[31:0]	Output	Read Data.

Physical Layer (PHY) Interfaces

The Physical Layer (PHY) handles link training, initialization and protocol. Included in this effort is insertion of CRC and the acknowledgement identifier into outgoing packets. The PHY also interfaces to the serial transceivers.

These transceivers are provided as an external instantiation to the core to ease customer use models. However, the PHY connections to the MGTs are abstracted by two wrapper modules, the PHY wrapper and the srio_wrapper. As such, the interface into the transceivers and also BUF interface are hidden from outside of the srio_wrapper.

Note: Port names and descriptions are from the PHY point of view.

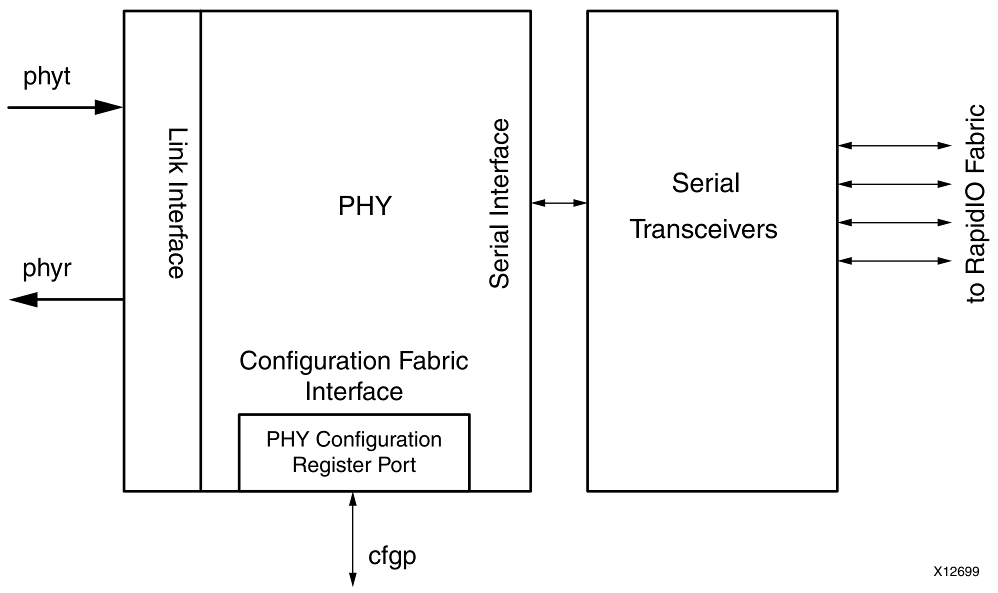


Figure 2-9: **Physical Layer (PHY) Interfaces**

As shown in Figure 2-9, there is an AXI4-Stream channel in each direction on the BUF side of the PHY. There is also an AXI4-Lite interface to the Configuration Fabric, which allows access to the PHY configuration space. There is also a serial interface on the transceiver side of the PHY.

Clock and Reset Interface

Table 2-16 lists the associated signals for the Clock and Reset Interface of the PHY layer.

Table 2-16: **PHY Clock and Reset Interface Signal List**

Signal	Direction	Description
phy_lcl_phy_clk	Input	Frequency depends on serial transfer frequency and initialized link width. If the core trains down to 1x mode, phy_clk must change to the 1x rate. See Clocking in Chapter 3 for more information. ***Note:*** This signal is the same as phy_clk, which is the name used in other areas of this document.
phy_rst	Input	Reset for PHY. Must deassert synchronously to phy_clk. See Resets in Chapter 3 for more information.
gt_pcs_clk	Input	Clock signal for Serial Interface. Must be phase aligned to phy_clk. The clock rate is based on the transfer frequency of the core, and is half of the gt_clk: • 1.25 G - 31.25 MHz • 2.5 G - 62.5 MHz • 3.125 G - 78.13 MHz • 5 G - 125 MHz • 6.25 G - 156.MHz
gt_pcs_rst	Input	Reset for Serial Interface. Must deassert synchronously to gt_pcs_clk. See Resets in Chapter 3 for more information.
phy_lcl_cfg_clk	Input	Configuration Register Interface clock. If the AXI4-Lite Maintenance Port and the Configuration Fabric reference design are in use, this must be equivalent to log_clk. Otherwise, this clock is independent of log_clk. ***Note:*** This signal is the same as cfg_clk, which is the name used in other areas of this document.
phy_lcl_cfg_rst	Input	Configuration Register Interface reset. Clears PHY registers to default values. Must deassert synchronously to cfg_clk. ***Note:*** This signal is the same as cfg_rst, which is the name used in other areas of this document.

Link Interface

Table 2-17 lists the signals associated with the PHY Link Interface.

Table 2-17: **PHY Link Interface Signal List**

Signal	Direction	Description
s_axis_phyt_tvalid	Input	Indicates that information on the channel is valid.
s_axis_phyt_tready	Output	Handshaking signal. Indicates that the data from the source is accepted (if valid).
s_axis_phyt_tdata[63:0]	Input	Packet header and data.
s_axis_phyt_tkeep[7:0]	Input	Byte qualifier that indicates whether the content of the associated byte of data is valid. This input should be set to 8'hFF except when tlast is asserted. Bit 7 corresponds to the most significant byte of data (tdata[63:56]) and bit 0 corresponds to the least significant byte (tdata[7:0]).
s_axis_phyt_tlast	Input	Indicates the last beat of a packet.
s_axis_phyt_tuser[7:0]	Input	On the first beat of a packet, Bit 1 reflects the value of the Critical Request Flow (CRF) flag for the packet, if CRF support is enabled. All other bits are reserved. On the last beat of a packet, Bit 0 is used as a source discontinue. If Bit 0 is high on the tlast transfer, the packet should be discarded.
m_axis_phyr_tvalid	Output	Indicates that the information on the channel is valid.
m_axis_phyr_tready	Input	Handshaking signal. Indicates that the data from the source is accepted (if valid).
m_axis_phyr_tdata[63:0]	Output	Packet header and data.
m_axis_phyr_tkeep[7:0]	Output	Byte qualifier that indicates whether the content of the associated byte of data is valid. This output will be set to 8'hFF except when tlast is asserted. Bit 7 corresponds to the most significant byte of data (tdata[63:56]) and bit 0 corresponds to the least significant byte (tdata[7:0]).
m_axis_phyr_tlast	Output	Indicates the last beat of a packet.
m_axis_phyr_tuser[7:0]	Output	If Bit 1is high on the first beat of a packet, the Critical Request Flow (CRF) flag for the packet will be set (if CRF Support is enabled). All other bits are reserved. On the last beat of a packet, Bit 0 is used as a source discontinue. If Bit 0 is high on the tlast transfer, the packet will be dropped in the BUF.
srio_host	Output	Reflects the value of the Host bit from the Port General Control CSR in the physical layer configuration space. When set, the endpoint is the system host device.

Table 2-17: **PHY Link Interface Signal List *(Cont'd)***

Signal	Direction	Description
phy_lcl_master_enable	Output	Reflects the value of the Master Enable bit from the Port General Control CSR in the physical layer configuration space. When set, the BUF can forward requests to the PHY. If this input is low, the BUF can only forward responses. ***Note:*** This signal is the same as master_enable, which is the name used in other areas of this document.
phy_lcl_idle2_selected	Output	When asserted, the PHY is operating in IDLE2 mode and using long control symbols (therefore ackIDs are 6 bits). ***Note:*** This signal is the same as idle2_selected, which is the name used in other areas of this document.
phy_lcl_phy_rewind	Output	This signal indicates that the PHY is requesting a rewind. The BUF must monitor the next_fm signal to determine which packet to transmit next. ***Note:*** This signal is the same as phy_rewind, which is the name used in other areas of this document.
phy_lcl_phy_next_fm	Output	This bus is used to communicate the next packet to send, indexed by ackID. This signal increments for each packet, except in a rewind scenario. ***Note:*** This signal is the same as phy_next_fm, which is the name used in other areas of this document.
phy_lcl_phy_last_ack	Output	This bus is used to communicate the ackID for the last acknowledge control symbol received. This frees up slots in the buffer by allowing the BUF to clear the corresponding packet. If this signal increments by more than one, multiple packet may be cleared. ***Note:*** This signal is the same as phy_last_ack, which is the name used in other areas of this document.
phy_lcl_phy_rcvd_buf_stat	Output	This bus is used to communicate the last received buffer status from a status control symbol received from the link partner. ***Note:*** This signal is the same as phy_rcvd_buf_stat, which is the name used in other areas of this document.

Table 2-17: **PHY Link Interface Signal List *(Cont'd)***

Signal	Direction	Description
phy_lcl_phy_buf_stat	Input	This bus is used to communicate the status of the Receive Buffer. This bus will never equal all ones, which indicates receiver flow control. Nor will the lower five bits be all ones, which may also be perceived as receiver flow control. It will always hold an estimation of number of free buffer locations. When the buffer is very lean (more than 30 free packet locations), the buf_stat value will provide the value of 30. ***Note:*** This signal is the same as phy_buf_stat, which is the name used in other areas of this document.
phy_lcl_tx_flow_control	Input	Indicates that the transmit buffer is in transmitter-controlled flow control mode. ***Note:*** This signal is the same as tx_flow_control, which is the name used in other areas of this document.

Serial Interface

Table 2-18 lists the signals associated with the Serial Interface.

Note: LW = link width.

Table 2-18: **PHY Serial Interface Signal List**

Signal	Direction	Description
gttx_data[32*LW-1:0]	Output	Transmit data to the serial transceivers.
gttx_charisk[4*LW-1:0]	Output	Per-byte K-character indicator to the serial transceivers. If bit 0 is asserted, gttx_data[7:0] is a K-character, and so on.
gttx_inhibit[LW-1:0]	Output	Per-lane inhibit for the serial transceivers. If bit 0 is asserted, the transmitter of lane0 is disabled, and so on.
gtrx_data[32*LW-1:0]	Input	Receive data from the serial transceivers.
gtrx_charisk[4*LW-1:0]	Input	Per-byte K-character indicator from the serial transceivers. If bit 0 is asserted, gtrx_data[7:0] is a K-character, and so on.
gtrx_chariscomma[4*LW-1:0]	Input	Per-byte comma indicator from the serial transceivers. If bit 0 is asserted, gtrx_data[7:0] is a comma character, and so on.
gtrx_disperr[4*LW-1:0]	Input	Per-byte disparity error indicator from the serial transceivers. If bit 0 is asserted, gtrx_data[7:0] had a disparity error, and so on.
gtrx_notintable[4*LW-1:0]	Input	Per-byte not-in-table error indicator from the serial transceivers. If bit 0 is asserted, gtrx_data[7:0] had an 8b/10b decode error, and so on.
gtrx_chanbondseq[LW-1:0]	Input	Per-lane channel bonding sequence indicator from the serial transceivers. If bit 0 is asserted, lane0 received a channel bonding sequence, and so on.
gtrx_chanisaligned[LW-1:0]	Input	Per-lane alignment indicator from the serial transceivers. If bit 0 is asserted, lane0 is aligned with the channel bonding master, and so on.

Table 2-18: **PHY Serial Interface Signal List *(Cont'd)***

Signal	Direction	Description
gtrx_chanbonden	Output	Enable channel bonding in the serial transceivers.
gtrx_reset_req	Input	The serial transceivers require a reset, for example due to an elastic buffer underflow or overflow.
gtrx_reset	Output	Reset for the serial transceivers.
gtrx_reset_done[LW-1:0]	Input	Per-lane indicator that the reset sequence is complete for the serial transceiver. If bit 0 is asserted, lane0 is finished with the reset process, and so on.

Control and Status Interface

Table 2-19 list the signals associated with the PHY Control/Status Interface.

Table 2-19: **PHY Control and Status Interface Signal List**

Signal	Direction	Description
sim_train_en	Input	This signal reduces the timers for intitialization. This is used only for simulation purposes.
phy_mce	Input	This is a single-cycle input signal that instructs the PHY to send an MCE control signal.
phy_link_reset	Input	As long as this signal is asserted, the PHY will send Link Reset control symbols.
force_reinit	Input	This signal forces the PHY to reintialize the link.
link_initialized	Output	Indicates the link has initialized. Specifically, at least 15 Status control symbols have been sent and 8 error-free Status control symbols have been received.
phy_rcvd_mce	Output	Indicates the PHY has received an MCE control symbol from the link partner.
phy_rcvd_link_reset	Output	Indicates the PHY has received at least four consecutive, error-free Link Reset control symbols. (Note: Status control symbols and Idles are permitted to be interspersed between these Link Reset control symbols.)
port_error	Output	Indicates the port has received an unrecoverable error and is in the error state. This signal reflects the value of the Port Error bit in the Port n Error and Status CSR in the Physical Layer configuration.
port_initialized	Output	Indicates the port has initialized. This signal reflects the value of the Port Uninitialized bit in the Port n Error and Status CSR in the Physical Layer configuration
mode_1x	Output	Indicates the link has trained down to 1x.
idle_selected	Output	Indicates that either IDLE1 or IDLE2 has been selected.
port_timeout[23:0]	Output	Reflects the value of the Timeout Value field in the Port Link Timeout Control CSR.
srio_host	Output	Reflects the value of the Host bit from the Port General Control CSR in the Physical Layer configuration

Table 2-19: **PHY Control and Status Interface Signal List *(Cont'd)***

Signal	Direction	Description
phy_lcl_maint_only	Output	This signal indicates that the LOG should only transmit maintenance transactions. If a request is in progress when this signal asserts, the request may be completed. ***Note:*** This signal is the same as maint_only, which is the name used in other areas of this document.
rx_lane_r	Output	Indicates the port has trained down to the redundancy lane.

Configuration Fabric Interface

Table 2-20 lists the signals associated with the Configuration Fabric Interface.

Table 2-20: **PHY Configuration Fabric Interface Signal List**

Signal	Direction	Description
s_axi_cfgp_awvalid	Input	Indicates that the write address is valid.
s_axi_cfgp_awready	Output	Handshaking signal. Indicates that the write address is accepted (if valid).
s_axi_cfgp_awaddr[23:0]	Input	Write address.
s_axi_cfgp_wvalid	Input	Indicates that the write data is valid.
s_axi_cfgp_wready	Output	Handshaking signal. Indicates that the write data is accepted (if valid).
s_axi_cfgp_wdata[31:0]	Input	Write data.
s_axi_cfgp_wstrb[3:0]	Input	Byte qualifier that indicates whether the content of the associated byte of data is valid.
s_axi_cfgp_bvalid	Output	Indicates that the write response is valid.
s_axi_cfgp_bready	Input	Handshaking signal. Indicates that the write response is accepted (if valid).
s_axi_cfgp_arvalid	Input	Indicates that the read address is valid.
s_axi_cfgp_arready	Output	Handshaking signal. Indicates that the read address is accepted (if valid).
s_axi_cfgp_araddr[23:0]	Input	Read address.
s_axi_cfgp_rvalid	Output	Indicates that the read response is valid.
s_axi_cfgp_rready	Input	Handshaking signal. Indicates that the read response is accepted (if valid).
s_axi_cfgp_rdata[31:0]	Output	Read Data.

Serial Transceivers Interfaces

Table 2-21 lists the signals associated with the interfaces for the Serial Transceivers module in the phy_wrapper.

Note: Port names and descriptions are from the serial transceiver point of view. LW = link width.

Table 2-21: **Serial Transceivers Interface Signal List**

Signal	Direction	Description
Clocks and Reset		
refclk	Input	Reference clock for the transceivers, and is dependent on the line rate. See Clocking in Chapter 3 for more information on the supported reference clock frequencies.
gt_clk	Input	Clock signal for Transceiver Interface. Must be phase aligned to phy_clk. The clock rate is based on the transfer frequency of the core: 1.25G - 62.5 Mhz 2.5G - 125 Mhz 3.125G - 156.25 Mhz 5G - 250 Mhz 6.25G - 312.5 Mhz
gt_pcs_clk	Input	Clock signal for Serial Interface. Must be phase aligned to phy_clk. The clock rate is based on the transfer frequency of the core, and is half of gt_clk: • 1.25G - 31.25 Mhz • 2.5G - 62.5 Mhz • 3.125G - 78.13 Mhz • 5G - 125 Mhz • 6.25G - 156.25 Mhz
gt_pcs_rst	Input	Reset for Serial Interface. Must deassert synchronously to gt_pcs_clk. See Resets in Chapter 3 for more information.
clk_lock	Input	Indicates the clocks are valid.
Serial Interface		
gttx_data[32*LW-1:0]	Input	Transmit data to the serial transceivers.
gttx_charisk[4*LW-1:0]	Input	Per-byte K-character indicator to the serial transceivers. If bit 0 is asserted, gttx_data[7:0] is a K-character, and so on.
gttx_inhibit[LW-1:0]	Input	Per-lane inhibit for the serial transceivers. If bit 0 is asserted, the transmitter of lane0 is disabled, and so on.
gtrx_data[32*LW-1:0]	Output	Receive data from the serial transceivers.
gtrx_charisk[4*LW-1:0]	Output	Per-byte K-character indicator from the serial transceivers. If bit 0 is asserted, gtrx_data[7:0] is a K-character, and so on.
gtrx_chariscomma[4*LW-1:0]	Output	Per-byte comma indicator from the serial transceivers. If bit 0 is asserted, gtrx_data[7:0] is a comma character, and so on.
gtrx_disperr[4*LW-1:0]	Output	Per-byte disparity error indicator from the serial transceivers. If bit 0 is asserted, gtrx_data[7:0] had a disparity error, and so on.
gtrx_notintable[4*LW-1:0]	Output	Per-byte not-in-table error indicator from the serial transceivers. If bit 0 is asserted, gtrx_data[7:0] had an 8b/10b decode error, and so on.

Table 2-21: **Serial Transceivers Interface Signal List *(Cont'd)***

Signal	Direction	Description
gtrx_chanbondseq[LW-1:0]	Output	Per-lane channel bonding sequence indicator from the serial transceivers. If bit 0 is asserted, lane0 received a channel bonding sequence, and so on.
gtrx_chanisaligned[LW-1:0]	Output	Per-lane alignment indicator from the serial transceivers. If bit 0 is asserted, lane0 is aligned with the channel bonding master, and so on.
gtrx_reset_req	Output	The serial transceivers require a reset, for example due to an elastic buffer underflow or overflow.
gtrx_reset	Input	Reset for the serial transceivers.
gtrx_reset_done[LW-1:0]	Output	Per-lane indicator that the reset sequence is complete for the serial transceiver. If bit 0 is asserted, lane0 is finished with the reset process, and so on.
Transceiver Interface		
srio_rxnN	Input	Half of the transceiver receive differential pair. There is a pair for each lane generated in the core. N reflects the lane number.
srio_rxpN	Input	Half of the transceiver receive differential pair. There is a pair for each lane generated in the core. N reflects the lane number.
srio_txnN	Output	Half of the transceiver transmit differential pair. There is a pair for each lane generated in the core. N reflects the lane number.
srio_txpN	Output	Half of the transceiver transmit differential pair. There is a pair for each lane generated in the core. N reflects the lane number.

Configuration Fabric Reference Design Interfaces

The Configuration Fabric reference design manages the accesses to the configuration spaces of each of the sub-cores. The configuration modules in each of the sub-cores are slaves on the configuration bus (an AXI-Lite interface), and the Configuration Fabric module is the bus master. Reads and writes derived from Maintenance transactions that were issued either locally or from the link partner are presented to the Configuration Fabric via the Configuration Master port in the LOG. Table 2-22 lists the signals associated with the Configuration Fabric reference design.

Note: Port names and descriptions are from the Configuration Fabric Reference Design point of view.

Table 2-22: **Configuration Fabric Reference Design Signal List**

Signal	Direction	Description
cfg_clk	Input	Configuration Register Interface clock. If the AXI4-Lite Maintenance Port and the Configuration Fabric reference design are in use, this must be equivalent to log_clk.
cfg_rst	Input	Configuration Register Interface reset. Must deassert synchronously to cfg_clk.
LOG Master Port		
cfgr_awvalid	Input	Indicates that the write address is valid.
cfgr_awready	Output	Handshaking signal. Indicates that the write address is accepted (if valid).
cfgr_awaddr[23:0]	Input	Write address.
cfgr_awprot[2:0]	Input	Tied to 0.
cfgr_wvalid	Input	Indicates that the write data is valid.
cfgr_wready	Output	Handshaking signal. Indicates that the write data is accepted (if valid).
cfgr_wdata[31:0]	Input	Write data.
cfgr_wstrb[3:0]	Input	Byte qualifier that indicates whether the content of the associated byte of data is valid.
cfgr_bvalid	Output	Indicates that the write response is valid.
cfgr_bready	Input	Handshaking signal. Indicates that the write response is accepted (if valid).
cfgr_bresp[1:0]	Input	Write response. • 2'b00 - OK • 2'b10 - Err • 2'bx1 - Reserved
cfgr_arvalid	Input	Indicates that the read address is valid.
cfgr_arready	Output	Handshaking signal. Indicates that the read address is accepted (if valid).
cfgr_araddr[23:0]	Input	Read address.
cfgr_arprot[2:0]	Input	Tied to 0.
cfgr_rvalid	Output	Indicates that the read response is valid.
cfgr_rready	Input	Handshaking signal. Indicates that the read response is accepted (if valid).
cfgr_rresp[1:0]	Input	Read response. • 2'b00 - OK • 2'b10 - Err • 2'bx1 - Reserved
cfgr_rdata[31:0]	Output	Read Data.
LOG Slave Port		
cfgl_awvalid	Output	Indicates that the write address is valid.

Table 2-22: **Configuration Fabric Reference Design Signal List**

Signal	Direction	Description
cfgl_awready	Input	Handshaking signal. Indicates that the write address is accepted (if valid).
cfgl_awaddr[23:0]	Output	Write address.
cfgl_awprot[2:0]	Output	Tied to 0.
cfgl_wvalid	Output	Indicates that the write data is valid.
cfgl_wready	Input	Handshaking signal. Indicates that the write data is accepted (if valid).
cfgl_wdata[31:0]	Output	Write data.
cfgl_wstrb[3:0]	Output	Byte qualifier that indicates whether the content of the associated byte of data is valid.
cfgl_bvalid	Input	Indicates that the write response is valid.
cfgl_bready	Output	Handshaking signal. Indicates that the write response is accepted (if valid).
cfgl_arvalid	Output	Indicates that the read address is valid.
cfgl_arready	Input	Handshaking signal. Indicates that the read address is accepted (if valid).
cfgl_araddr[23:0]	Output	Read address.
cfgl_arprot[2:0]	Output	Tied to 0.
cfgl_rvalid	Input	Indicates that the read response is valid.
cfgl_rready	Output	Handshaking signal. Indicates that the read response is accepted (if valid).
cfgl_rdata[31:0]	Input	Read Data.
BUF Slave Port		
cfgb_awvalid	Output	Indicates that the write address is valid.
cfgb_awready	Input	Handshaking signal. Indicates that the write address is accepted (if valid).
cfgb_awaddr[23:0]	Output	Write address.
cfgb_awprot[2:0]	Output	Tied to 0.
cfgb_wvalid	Output	Indicates that the write data is valid.
cfgb_wready	Input	Handshaking signal. Indicates that the write data is accepted (if valid).
cfgb_wdata[31:0]	Output	Write data.
cfgb_wstrb[3:0]	Output	Byte qualifier that indicates whether the content of the associated byte of data is valid.
cfgb_bvalid	Input	Indicates that the write response is valid.
cfgb_bready	Output	Handshaking signal. Indicates that the write response is accepted (if valid).
cfgb_arvalid	Output	Indicates that the read address is valid.
cfgb_arready	Input	Handshaking signal. Indicates that the read address is accepted (if valid).

Table 2-22: **Configuration Fabric Reference Design Signal List**

Signal	Direction	Description
cfgb_araddr[23:0]	Output	Read address.
cfgb_arprot[2:0]	Output	Tied to 0.
cfgb_rvalid	Input	Indicates that the read response is valid.
cfgb_rready	Output	Handshaking signal. Indicates that the read response is accepted (if valid).
cfgb_rdata[31:0]	Input	Read Data.
PHY Slave Port		
cfgp_awvalid	Output	Indicates that the write address is valid.
cfgp_awready	Input	Handshaking signal. Indicates that the write address is accepted (if valid).
cfgp_awaddr[23:0]	Output	Write address.
cfgp_awprot[2:0]	Output	Tied to 0.
cfgp_wvalid	Output	Indicates that the write data is valid.
cfgp_wready	Input	Handshaking signal. Indicates that the write data is accepted (if valid).
cfgp_wdata[31:0]	Output	Write data.
cfgp_wstrb[3:0]	Output	Byte qualifier that indicates whether the content of the associated byte of data is valid.
cfgp_bvalid	Input	Indicates that the write response is valid.
cfgp_bready	Output	Handshaking signal. Indicates that the write response is accepted (if valid).
cfgp_arvalid	Output	Indicates that the read address is valid.
cfgp_arready	Input	Handshaking signal. Indicates that the read address is accepted (if valid).
cfgp_araddr[23:0]	Output	Read address.
cfgp_arprot[2:0]	Output	Tied to 0.
cfgp_rvalid	Input	Indicates that the read response is valid.
cfgp_rready	Output	Handshaking signal. Indicates that the read response is accepted (if valid).
cfgp_rdata[31:0]	Input	Read Data.

Register Space

The Register Space definitions are distributed throughout the RapidIO Specification. Table 2-23 shows the structure of the Register Space, and which SRIO Gen2 Endpoint sub-core contains each piece of address space.

Table 2-23: **Register Space**

Configuration Space Byte Offset	RapidIO Specification Register Space	SRIO Gen2 Endpoint
0x00-0x3C	Capability Register Space	LOG(1)
0x040-0xFC	Command and Status Register Space	LOG
0x0100-0xFFFC	Extended Features Space	PHY
0x010000-0xFFFFFC	Implementation-defined Space	BUF, LOG(2)

1. The Processing Element Features CAR implemented in the LOG contains bits defined in Part 6: LP-Serial Physical Layer Specification of the RapidIO Specification.
2. The Maintenance Request Information Register, which only exists when the AXI4-Lite Maintenance port is being used, is located in the LOG. It exists behind the User Maintenance interface and cannot be accessed by another AXI4 master (only via maintenance request transactions or requests received from the link partner).

Capability Register Space

The registers in the Capability Register Space are largely defined in Parts 1-3 of the RapidIO Specification (though the Processing Element Features CAR contains bits defined in Part 6: LP-Serial Physical Layer Specification of the RapidIO Specification). These registers are Read-Only, and are implemented with the Command and Status Registers in the LOG.

Table 2-24: **Capability Register Map**

Configuration Space Byte Offset	Register Name
0x00	Device Identity CAR
0x04	Device Information CAR
0x08	Assembly Identity CAR
0x0C	Assembly Information CAR
0x10	Processing Element Features CAR
0x14	Switch Port Information CAR
0x18	Source Operations CAR
0x1C	Destination Operations CAR
0x20-0x3C	Reserved

Device Identity CAR

This CAR defines the Device ID and Device Vendor ID. This CAR is not user-accessible and cannot be changed. Reads to this register return a value dependent on the device targeted with the CORE Generator software.

Table 2-25: **Device Identity CAR (Offset 0x0)**

Bits	Field	Value	Access	Description
[31:16]	Device ID	See Description Column	R	Device ID Virtex®-7: 16'h0370 Virtex-6: 16'h0360 Kintex™-7: 16'h0270
[15:0]	Device Vendor ID	16'h000E	R	RIO TA assigned Vendor ID for Xilinx

Device Information CAR

This CAR defines additional information about the device such as the Device Revision Label, Major Revision, Minor Revision, and patch version. All other bits are reserved.

Table 2-26: **Device Information CAR (Offset 0x4)**

Bits	Field	Value	Access	Description
[31:20]	Reserved	12'b0	R	
[19:16]	Device Revision Label	4'h2	R	Device Revision Label. 4'h2. Indicates that core was designed to *RapidIO Interconnect Specification rev. 2.2.*
[15:12]	Reserved	4'h0	R	
[11:8]	Major Revision	4'h1	R	Updated with each major release
[7:4]	Minor Revision	4'h5	R	Updated with each official core release
[3:0]	Patch	4'h0	R	Updated for bug fixes only

Assembly Identity CAR

This CAR defines the vendor that manufactured the assembly or the subsystem containing the device, Assembly Vendor ID, and the type of assembly, Assembly ID. Reads to this register return the value set within the CORE Generator software.

Table 2-27: **Assembly Identity CAR (Offset 0x8)**

Bits	Field	Value	Access	Description
[31:16]	Assembly ID	CORE Generator	R	Customer Modifiable Field
[15:0]	Assembly Vendor ID	CORE Generator	R	Customer Modifiable Field

Assembly Information CAR

This CAR defines more information about the assembly, such as the assembly revision number and the pointer to the first entry in the extended features list. Reads to this register return the value set within the CORE Generator software.

Table 2-28: **Assembly Information CAR (Offset 0xC)**

Bits	Field	Value	Access	Description
[31:16]	Assembly Revision	CORE Generator	R	Customer Modifiable Field
[15:0]	Extended Features Pointer	CORE Generator	R	Pointer to the PHY Register space (default 16’h0100)

Processing Elements Features CAR

This CAR defines the functionality provided by this endpoint. The RapidIO Specification gives you the option to define this as a Bridge, Memory, Processor or Switch element. The LOG Interface supports Bridge, Memory and Processor implementations. Large System support is also defined in this register. When bit 4 is set, the endpoint will use 16-bit Device IDs. This option is enabled through the CORE Generator software GUI. All other bits are reserved.

Table 2-29: **Processing Elements Features CAR (Offset 0x10)**

Bits	Field	Value	Access	Description
31	Bridge	CORE Generator	R	Endpoint used as a bridge to another interface
30	Memory	CORE Generator	R	Endpoint has physically addressable storage space
29	Processor	CORE Generator	R	Endpoint has a local processor that runs code
28	Switch	1'b0	R	The SRIO Gen2 Endpoint does not currently support switch functionality
27	Multiport	1'b0	R	The SRIO Gen2 Endpoint does not currently support multiple ports
[26:7]	Reserved	20'b0	R	
6	Implementation-defined	1'b0	R	
5	CRF Support	CORE Generator	R	Support for critical request flow indicator.
4	Common Transport Large System Support	CORE Generator	R	Endpoint supports 16-bit Device IDs
3	Extended Features	1'b1	R	Endpoint has extended features. Set to 1 indicating the extended features pointer is valid and points to PHY register space
[2:0]	Extended Addressing Support	3'b001	R	Endpoint supports 34-bit addressing mode

Switch Port Information CAR

This CAR defines the switching capabilities. This register is only valid if the Switch bit (bit 28) of the Processing Element Features CAR is set. However, switch functionality is not currently supported, and this bit is tied to 0.

Source Operations CAR

This CAR defines the set of RapidIO logical operations that can be issued by this endpoint. The state of these bits do not affect functionality. Reads to register return the values for the parameters set in the CORE Generator software. All other bit locations including Reserved bits return 0.

Table 2-30: **Source Operations CAR (Offset 0x18)**

Bit	Field	Value	Access	Description
[31:18]	Reserved	14'b0	R	
[17:16]	Implementation -defined	2'b0	R	
15	Read	CORE Generator	R	Core can support a read (NREAD) operation
14	Write	CORE Generator	R	Core can support a write (NWRITE) operation
13	Streaming-write	CORE Generator	R	Core can support a streaming-write (SWRITE) operation
12	Write-with-resp onse	CORE Generator	R	Core can support a write-with-response (NWRITE_R) operation
11	Data Message	CORE Generator	R	Core can support a data message operation
10	Doorbell	CORE Generator	R	Core can support a doorbell operation
9	Atomic (compare-and-s wap)	CORE Generator	R	Core can support an Atomic compare-and-swap transaction
8	Atomic (test-and-swap)	CORE Generator	R	Core can support an atomic test-and-swap transaction
7	Atomic (increment)	CORE Generator	R	Core can support an Atomic increment transaction
6	Atomic (decrement)	CORE Generator	R	Core can support an Atomic decrement transaction
5	Atomic (set)	CORE Generator	R	Core can support an Atomic set transaction.
4	Atomic (clear)	CORE Generator	R	Core can support an Atomic clear transaction.
3	Atomic (swap)	CORE Generator	R	Core can support an Atomic swap transaction.

Table 2-30: **Source Operations CAR (Offset 0x18) *(Cont'd)***

Bit	Field	Value	Access	Description
2	Port-write	1'b0	R	Port-write not supported by the SRIO Gen2 Endpoint.
[1:0]	Implementation -defined	2'b0	R	

Destination Operations CAR

This CAR defines the set of RapidIO logical operations that can be supported by this endpoint. Reads to register return the values for the parameters set in the CORE Generator software. All other bit locations including Reserved bits will return 0.

Table 2-31: **Destination Operations CAR (Offset 0x1C)**

Bit	Field Name	Value	Access	Description
[31:18]	Reserved	14'b0	R	
[17:16]	Implementation-defined	2'b0	R	
15	Read	CORE Generator	R	Core can support a read (NREAD) operation
14	Write	CORE Generator	R	Core can support a write (NWRITE) operation
13	Streaming-write	CORE Generator	R	Core can support a streaming-write (SWRITE) operation
12	Write-with-response	CORE Generator	R	Core can support a write-with-response (NWRITE_R) operation
11	Data message	CORE Generator	R	Core can support a data message operation
10	Doorbell	CORE Generator	R	Core can support a doorbell operation
9	Atomic (compare-and-swap)	CORE Generator	R	Core can support an Atomic compare-and-swap transaction
8	Atomic (test-and-swap)	CORE Generator	R	Core can support an atomic test-and-swap transaction
7	Atomic (increment)	CORE Generator	R	Core can support an Atomic increment transaction
6	Atomic (decrement)	CORE Generator	R	Core can support an Atomic decrement transaction
5	Atomic (set)	CORE Generator	R	Core can support an Atomic set transaction.
4	Atomic (clear)	CORE Generator	R	Core can support an Atomic clear transaction.
3	Atomic (swap)	CORE Generator	R	Core can support an Atomic swap transaction.

Table 2-31: **Destination Operations CAR (Offset 0x1C) *(Cont'd)***

Bit	Field Name	Value	Access	Description
2	Port-write	1'b0	R	Port-write not supported by the SRIO Gen2 Endpoint.
[1:0]	Implementation-defined	2'b0	R	

Command and Status Register Space

The registers in the Command and Status Register Space are defined in Parts 1-3 of the RapidIO Specification. These registers are implemented with the Capability Registers in the LOG. The Configuration Fabric module should route all transactions targeting this space to the LOG Configuration Register port.

Table 2-32: **Command and Status Register Map**

Configuration Space Byte Offset	Register Name
0x40-0x48	Reserved
0x4C	Processing Element Logical Layer CSR
0x50-0x54	Reserved
0x58	Local Configuration Space Base Address 0 CSR
0x5C	Local Configuration Space Base Address 1 CSR
0x60	Base DeviceID CSR
0x64	Reserved
0x68	Host Base DeviceID Lock CSR
0x6C	Component Tag CSR
0x70-0xFC	Reserved

Processing Element Logical Layer Control CSR

This CSR is used for general command and status information for this interface. The RapidIO Specification defines one field in this register that identifies the ability to support 34, 50 or 64 bit addressing. This interface supports 34 bit addressing only. All other bits are reserved. Reads to this register return the value 0000_0001h. Writes to this register are ignored.

Table 2-33: **Processing Element Logical Layer CSR (Offset 0x4C)**

Bits	Field	Reset Value	Access	Description
[31:3]	Reserved	29'b0	R	
[2:0]	Extended Addressing Control	3'b001	R	Endpoint supports 34-bit addressing

Local Configuration Space Base Address 0 CSR

This CSR is the configuration space high-base address register. It specifies the most significant bytes of the local physical address offset for this endpoint's configuration register space if the local address space is greater than 34-bits. Reads to this register return 0000_0000h. Writes to this register are ignored.

Table 2-34: **Local Configuration Space Base Address 1 CSR (Offset 0x58)**

Bits	Field	Reset Value	Access	Description
[31:0]	Reserved	32'b0	R	Endpoint only supports 34-bit addressing

Local Configuration Space Base Address 1 CSR

This CSR is the configuration space low-base address register. It specifies the local physical address offset for this endpoint's configuration register space. This allows the configuration space to be physically mapped in endpoint memory. This register allows access to the configuration space through normal read and write operations rather than maintenance operations. For more information about LCSBA functionality, see Local Configuration Space Base Address Accesses, page 102.

The default value of this register must be set through the CORE Generator software. When the device is operational, the BAR address value may be overwritten by a maintenance write operation or a write operation that "hits" the base address value located in this register. The new BAR value is written to offset `0x5Ch` in the configuration space. Once the register is written, its new value will be compared against incoming reads and writes to determine if they are intended for the configuration space.

The RapidIO Specification defines the size of the configuration space as 16 Megabytes. The size limits the addressable locations of the configuration space within the memory of the endpoint. For this reason, only bits [30:21] are writable and are used to make the address comparison with normal read and write operations. Bits[20:0] are read-only and are always set to zero.

Table 2-35: **Local Configuration Space Base Address 1 CSR (Offset 0x5C)**

Bits	Field	Reset Value	Access	Description
31	Reserved	1'b0	R	Endpoint only supports 34-bit addressing
[30:21]	LCSBA	CORE Generator	RW	Local Configuration Space Base Address, corresponds to bits 34:25 of a new address. An incoming address targeting this address space is redirected to the maintenance space.
[20:0]	LCSBA (unused)	21'h00_0000	R	Unused portion of the LCSBA

Base Device ID CSR

This CSR contains the base Device ID value for this endpoint. For 8-bit Device ID, bits [23:16] are valid. If Large System Support is enabled, then the 16-bit Device ID occupies bits [15:0]. The default value of this register must be set through the CORE Generator software. All other bits are reserved. Reads to reserved bits return 0 and writes are ignored.

Table 2-36: **Base DeviceID CSR (Offset 0x60)**

Bits	Field	Reset Value	Access	Description
[31:24]	Reserved	8'h00	R	
[23:16]	Base_deviceID	CORE Generator	RW	8-bit Device ID for an endpoint in a small transport system
[15:0]	Large_base_deviceID	CORE Generator	RW	16-bit Device ID for an endpoint in a large transport system

Host Base Device ID Lock CSR

This CSR contains the base Device ID value for the Host that is responsible for initializing this endpoint. This register is a write-once register which provides a lock function. Reads after reset return a default value of 0000_FFFFh. This register can be written once after reset. All subsequent writes will be ignored except when bits [15:0] of the written value matches the value contained in the register field (both bytes must be written simultaneously for the comparison to occur), in which case it will be reset to 0000_FFFFh. Writes of 16'hFFFF to the Host_base_deviceID field will not lock the CSR.

Table 2-37: **Host Base Device ID Lock CSR (Offset 0x68)**

Bits	Field	Reset Value	Access	Description
[31:16]	Reserved	16'h0000	R	
[15:0]	Host_base_deviceI D	16'hFFFF	RW once	Device ID of the system host. Field is writable when value is 16'hFFFF. Otherwise writing the current value will reset to 16'hFFFF to allow for a host base Device ID change.

Component Tag CSR

This CSR contains a component tag value for this endpoint and can be assigned by software during initialization. Reads after reset will return a default value of 0000_0000h. The value can be changed through a maintenance write operation or a normal write operation with the configuration space base address offset 0x6C.

Table 2-38: **Component Tag CSR (Offset 0x6C)**

Bits	Field	Reset Value	Access	Description
[31:0]	Component_Tag	32'h0000_0000	RW	Component Tag to be set by software at initialization.

Extended Features Space

The Extended Features Space (implemented in the PHY) is unique because it has multiple sub-spaces.

The RapidIO Physical Layer Specification defines Extended Features (EF) blocks, which are allowed by the specification to reside anywhere in the Extended Features space. The PHY will hold two Extended Features blocks: LP-Serial and LP-Serial Lane.

Extended Features Block Offsets

Although the specification allows for the Extended Features blocks to be located anywhere within the EF data structure, the PHY restricts their locations and ordering. By default, the LP-Serial Extended Features block is at offset 0x0100 and the LP-Serial Lane Extended Features block is at offset 0x0400.

The first register in each block (called the Block Header) contains the Extended Features Block ID and a pointer to the next Extended Features block (EF_PTR). The final EF_PTR will point to offset 0x0, indicating the end of the Extended Features data structure.

LP-Serial Extended Features Block Registers

The registers in the LP-Serial Extended Features block are defined in Part 6 of the RapidIO Specification. The default offset of the LP-Serial Extended Features block is 0x0100. Each register's byte offset is in addition to the block offset (for example, with the default offset in place, a read to address 0x0120 will access the Port Link Timeout CSR).

Table 2-39: **LP-Serial Register Map**

Configuration Space Byte Offset	Register Name
0x0	LP-Serial Register Block Header
0x04-0x1C	Reserved
0x20	Port Link Timeout CSR
0x24	Port Response Timeout CSR
0x28-0x38	Reserved
0x3C	Port General Control CSR
0x40	Port *n* Maintenance Request CSR
0x44	Port *n* Maintenance Response CSR
0x48	Port *n* Local ackID CSR
0x4C-0x50	Reserved
0x54	Port *n* Control 2 CSR
0x58	Port *n* Error and Status CSR

Table 2-39: **LP-Serial Register Map *(Cont'd)***

Configuration Space Byte Offset	Register Name
0x5C	Port *n* Control CSR
0x60-0xFC	Reserved

LP-Serial Register Block Header

This register contains the extended features pointer (EF_PTR) to the next extended features block and the EF_ID that identifies this as the generic endpoint block header.

Table 2-40: **LP-Serial Register Block Header (Offset 0x0)**

Bits	Field	Reset Value	Access	Description
[31:16]	Extended Features Pointer	CORE Generator	R	Pointer to the next extended features block, if one exists.
[15:0]	Extended Features ID	CORE Generator	R	Hard-wired extended features ID. Value of 0'h0002 if software assisted error recovery is enabled, 0'h0001 otherwise.

Port Link Timeout CSR

The port link timeout control register contains the timeout timer value for all ports on a device. This timeout is for link events such as sending a packet to receiving the corresponding acknowledgement, and sending a link-request to receiving the corresponding link-response. The settings are a linear progression from 0 to max.

Table 2-41: **Port Link Timeout CSR (Offset 0x20)**

Bits	Field	Reset Value	Access	Description
[31:8]	Timeout Value	CORE Generator	RW	Link timeout value from packet sent to acknowledgement receipt and Link-Request to Link-Response interval.
[7:0]	Reserved	8'b0	R	

Table 2-42 provides the multiply factor for the timeout value. Multiply the timeout value with the multiply factor to calculate the timeout value in seconds.

Table 2-42: **Timeout Calculation**

Line Rate (Gbps)	Multiply Factor (ns)
1.25	224
2.5	240
3.125	192
5.0	256
6.25	205

Port Response Timeout CSR

The port response timeout control register contains the time-out timer count for all ports on a device. This timeout is for sending a request packet to receiving the corresponding response packet. Table 2-42 provides the multiply factor for the timeout value. Multiply the timeout value with the multiply factor to calculate the timeout value in seconds.

Table 2-43: **Port Response Timeout CSR (Offset 0x24)**

Bits	Field	Reset Value	Access	Description
[31:8]	Timeout Value	CORE Generator	RW	Timeout from sending a Request packet to receiving a response packet.
[7:0]	Reserved	8'b0	R	

Port General Control CSR

The bits accessible through this CSR apply to all ports on a device. There is a single copy of each such bit per device.

Table 2-44: **Port General Control CSR (Offset 0x3C)**

Bits	Field	Reset Value	Access	Description
31	Host	CORE Generator	RW	Device responsible for system exploration.
30	Master Enable	CORE Generator	RW	Device is allowed to issue requests into the system. If set to 0, core will not accept requests from user.
29	Discovered	CORE Generator	RW	Device has been located by the PE responsible for system exploration.
[28:0]	Reserved	29'b0	R	

Port *n* Link Maintenance Request CSR

A write to this register generates a link-request control symbol on the corresponding port. This register is available only if the software assisted error recovery option is selected in the GUI.

Table 2-45: **Port *n* Link Maintenance Request CSR (Offset 0x40)**

Bits	Field	Reset Value	Access	Description
[31:3]	Reserved	29’b 0	R	
[2:0]	Command	3’b0	RW	Command that is sent in the link-request control symbol.

Port *n* Link Maintenance Response CSR

This register returns the ackID_status and port_status of the link-response control symbol that was last received in response to the link-request that was sent out by the corresponding port. This register is available only if the software assisted error recovery option is selected in the GUI.

Table 2-46: **Port *n* Link Maintenance Response CSR (Offset 0x44)**

Bits	Field	Reset Value	Access	Description
31	Response_valid	1'b0	RC	If the corresponding link request causes a link response, this bit indicates that the link response has been received and the status fields are valid. If the corresponding link request does not cause a link response, this bit indicates the link request has been sent.
[30:11]	Reserved	20'b 0	R	
[10:5]	ackID_status	6'b0	R	ackID_status field in the link response control symbol.
[4:0]	link_status	5'b0	R	link_status field in the link-response control symbol.

Port *n* Local ackID CSR

This register returns the status of the local ackID for both the input and output sides of the corresponding port. This register is available only if the software assisted error recovery option is selected in the GUI.

Table 2-47: **Port *n* Local ackID CSR (Offset 0x48)**

Bits	Field	Reset Value	Access	Description
31	Clear_outstanding_ackIDs	1'b0	RW	This bit is always 0 when read. When written, this bit clears all outstanding unacknowledged received packets.
30	Reserved	1'b0	R	
[29:24]	Inbound_ackID	6'b0	RW	Expected ackID value in the next received packet. ***Note:*** This bit is read-only when bit 31 is set.
[23:14]	Reserved	10'b 0	R	
[13:8]	Outstanding_ackID	6'b0	R	Expected ackID value in the next received packet accepted control symbol.
[7:6]	Reserved	2'b0	R	
[5:0]	Outbound_ackID	6'b0	RW	Value of the next transmitted ackID. A write to this register can force retransmission of outstanding unacknowledged packets.

Port *n* Error and Status CSR

These registers are accessed when a local processor or an external device wishes to examine the port error and status information.

Table 2-48: **Port *n* Error and Status CSR (Offset 0x58)**

Bits	Field	Reset Value	Access	Description
31	Idle Sequence 2 Support	CORE Generator	R	Indicates whether the port supports idle sequence 2 for baud rates of less than 5.5 Gbaud.
30	Idle Sequence 2 Enable	CORE Generator	R	Controls whether idle sequence 2 is enabled for baud rates less than 5.5 Gbaud. Read-only if idle sequence 2 is not enabled, or if the core is configured for greater than 5.5 Gbaud. • 1'b0 - idle sequence 1 enabled • 1'b1 - idle sequence 2 enabled
29	Idle Sequence	CORE Generator	R	Indicates which idle is in use. • 1'b0 - idle sequence 1 is active • 1'b1 - idle sequence 2 is active
28	Reserved	1'b0	R	
27	Flow Control Mode	CORE Generator	R	Indicates which flow control mode is in use. • 1'b0 - RX flow control • 1'b1 - TX flow control
[26:21]	Reserved	6'b0	R	
20	Output Retry-encountered	1'b0	RW1C	Indicates an Output Retry has been encountered in the past. Write 1'b1 to clear this bit.
19	Output Retried	1'b0	R	Indicates an Output Retry was the last response received. Set when a Packet-Retry Control Symbol is received. Cleared when a Packet-Accepted to Packet-not-Accepted is received.
18	Output Retry-Stopped	1'b0	R	Output port is in the Retry-Stopped state.
17	Output Error-encountered	1'b0	RW1C	Output port has received an output error in the past. Write 1'b1 to clear this bit.
16	Output Error-stopped	1'b0	R	Output port is currently in error and awaiting a valid Link Response control symbol to clear it.
[15:11]	Reserved	5'b0	R	
10	Input Retry-stopped	1'b0	R	Port is currently in the Input Retry-stopped state awaiting a Restart from Retry to clear it.

Table 2-48: **Port *n* Error and Status CSR (Offset 0x58) *(Cont'd)***

Bits	Field	Reset Value	Access	Description
9	Input Error-encountered	1'b0	RW1C	Port has encountered an input error in the past. Write 1'b1 to clear this bit.
8	Input Error-stopped	1'b0	R	Port is in an Input Error-stopped state awaiting a Link Request to clear the error state.
[7:3]	Reserved	5'b0	R	
2	Port Error	1'b0	RW1C	Port has encountered an unrecoverable error on the input port and/or the output port.
1	Port OK	1'b0	R	Input and Output ports are initialized, and the core is exchanging error free control symbols with the link partner.
0	Port Un-initialized	1'b1	R	Port is un-initialized.

Port *n* Control CSR

This register contains the control register bits for individual ports on a processing element.

Table 2-49: **Port *n* Control CSR (Offset 0x5C)**

Bits	Field	Reset Value	Access	Description
[31:30]	Port Width	CORE Generator	R	Width of Device: • 2'b00 - 1x • 2'b01 - 4x • 2'b10 - 2x • 2'b11 - Reserved
[29:27]	Initialized Port Width	3'b000	R	Initialized Port Width: • 3'b000 - 1x, lane 0 • 3'b001 - 1x, lane R • 3'b010 - 4x • 3'b011 - 2x • 3'b100 - 3'b111 - Reserved
[26:24]	Port Width Override	CORE Generator	RW	Override for Initialized Port Width: • 3'b000 - No Override • 3'b001 - Reserved • 3'b010 - Force 1x, lane 0 • 3'b011 - Force 1x, lane R • 3'b100 - 3'b111 - Reserved The core will ignore writes with invalid values.
23	Port Disable	CORE Generator	RW	When set, serial transceivers are disabled.

Table 2-49: **Port *n* Control CSR (Offset 0x5C) *(Cont'd)***

Bits	Field	Reset Value	Access	Description
22	Output Port Enable	CORE Generator	RW	When cleared, endpoint can only send maintenance packets.
21	Input Port Enable	CORE Generator	RW	When cleared, port can only receive maintenance packets, all other ftypes incur a packet-not-accepted response.
20	Error Checking Disable	CORE Generator	RW	Disables RapidIO error checking. When set, core behavior is undefined as all errors may propagate.
19	Multi-cast Event Participant	1'b1	R	Device can accept multi-cast packets.
18	Reserved	1'b0	R	
17	Enumeration Boundary	CORE Generator	RW	An enumeration boundary aware system will not enumerate past a port with this bit set.
16	Reserved	1'b0	R	
[15:14]	Extended Port Width Override	12'h000	R	The SRIO Gen2 Endpoint does not support 8x or 16x port widths, so this field is tied to 0.
[13:12]	Extended Port Width Support	2'b0	R	The SRIO Gen2 Endpoint does not support 8x or 16x port widths, so this field is tied to 0.
[11:4]	Implementation-defined	8'b0	R	
[3:1]	Reserved	3'b0	R	
0	Port Type	1'b1	R	Serial Port

LP-Serial Lane Extended Features Block Registers

The registers in the LP-Serial Lane Extended Features block are defined in Part 6 of the RapidIO Specification. The default offset of the LP-Serial Lane Extended Features block is 0x0400. Each register's byte offset is in addition to the block offset (for example, with the default offset in place, a read to address 0x0410 will access the Lane 0 Status 0 CSR).

Table 2-50: **LP-Serial Lane Register Map**

Configuration Space Byte Offset	Register Name
0x0	LP-Serial Lane Register Block Header
0x4-0xC	Reserved
0x10	Lane 0 Status 0 CSR
0x14	Lane 0 Status 1 CSR
0x18-0x2C	Reserved
0x30	Lane 1 Status 0 CSR

Table 2-50: **LP-Serial Lane Register Map**

Configuration Space Byte Offset	Register Name
0x34	Lane 1 Status 1 CSR
0x38-0x4C	Reserved
0x50	Lane 2 Status 0 CSR
0x54	Lane 2 Status 1 CSR
0x58-0x6C	Reserved
0x70	Lane 3 Status 0 CSR
0x74	Lane 3 Status 1 CSR
0x78-0xFC	Reserved

LP-Serial Lane Register Block Header

This register contains the extended features pointer (EF_PTR) to the next extended features block and the EF_ID that identifies this as the generic endpoint block header.

Table 2-51: **LP-Serial Lane Register Block Header (Offset 0x0)**

Bits	Field	Reset Value	Access	Description
[31:16]	Extended Features Pointer	CORE Generator	R	Pointer to the next extended features block, if one exists
[15:0]	Extended Features ID	16'h000D	R	Hard-wired extended features ID of LP-Serial Lane Extended Features Block

Lane *n* Status 0 CSRs

This register contains status information about the local lane transceiver. There is one register per lane, based on the link width of the generated core (for example, 1x cores only have the Lane 0 Status 0 CSR at offset 0x10). All bits in the register are read-only.

Table 2-52: **Lane *n* Status 0 CSRs (Offsets 0x10, 0x30. 0x50, 0x70)**

Bits	Field	Reset Value	Access	Description
[31:24]	Port Number	8'h00	R	Port number to which lane is assigned. Hard-wired to 8'b0
[23:20]	Lane Number	4'hn	R	Lane number. Hard wired to 4'h0 for Lane 0 (offset 0x30), 4'h1 for Lane 1 (offset 0x50), etc.
19	Transmitter Type	1'b0	R	Tied to 1'b0, indicating short-run transmitter.
18	Transmitter Mode	1'b0	R	Tied to 1'b0, indicating short-run transmitter operating mode.
[17:16]	Receiver Type	2'b0	R	Tied to 2'b0, indicating short-run receiver.

Table 2-52: **Lane *n* Status 0 CSRs (Offsets 0x10, 0x30. 0x50, 0x70)**

Bits	Field	Reset Value	Access	Description
15	Receiver Input Inverted	1'b0	R	The SRIO Gen2 Endpoint does not include receiver polarity detection/correction. Tied to 1'b0.
14	Receiver Trained	1'b0	R	Indicates the state of the rcvr_trained signal. When set, indicates that the adaptive equalizers controlled by the lane receiver are trained.
13	Receiver Lane Sync	1'b0	R	Indicates the state of the lane_sync signal.
12	Receiver Lane Ready	1'b0	R	Indicates the state of the lane_ready signal.
[11:8]	8B/10B Decoding Errors	4'h0	RC	Indicates the number of 8B/10B decoding errors that have been detected for this lane since the register was last read. This field is reset to 4'h0 when the register is read (errors during the read may be dropped). If >15 errors are received before the register is cleared, the register will hold the value of 4'hF.
7	lane_sync State Change	1'b0	RC	When set, indicates that there has been a change in the value of the lane_sync signal. Resets to 1'b0 when the register is read.
6	rcvr_trained State Changed	1'b0	RC	When set, indicates that there has been a change in the value of the rcvr_trained signal. Resets to 1'b0 when the register is read.
[5:4]	Reserved	2'b0	R	
3	Status 1 CSR Implemente d	CORE Generator	R	Indicates whether or not the Status 1 CSR is implemented. The SRIO Gen2 Endpoint implements the Status 1 CSR for all lanes if IDLE2 sequences are enabled in the GUI. 1'b0 - The Status 1 CSR is not implemented for this lane. 1'b1 - The Status 1 CSR is implemented for this lane.
[2:0]	Status 2-7 CSRs Implemente d	3'b000	R	The SRIO Gen2 Endpoint does not implement any of the implementation-defined Status 2-7 CSRs.

Lane *n* Status 1 CSRs

This register is implemented only if the lane supports the IDLE2 sequence. It contains status information about the connected port, which is collected from the CS Fields and CS Markers of the received IDLE2 sequences. There is one register per lane, based on the link width of the generated core (for example, 1x cores only have the Lane 0 Status 1 CSR at offset 0x14).

Table 2-53: **Lane *n* Status 1 CSRs (Offset 0x14, 0x34, 0x54, 0x74)**

Bits	Field	Reset Value	Access	Description
31	IDLE2 Received	1'b0	RW1C	Indicates whether an IDLE2 has been received by the lane since the bit was last reset. The bit can be reset by writing to it with a value of 1'b1.
30	IDLE2 Information Current	1'b0	R	When set, this bit indicates that the information in the register was collected from the CS Field and CS Marker of an IDLE2 sequence received without detected errors, and that the lane_sync signal has remained asserted since the last CS Field and CS Marker were received.
29	Values Changed	1'b0	RC	When set, this bit indicates whether any of the other bits in the register have changed. It is reset to 1'b0 when read.
28	Implementati on-defined	1'b0	R	
27	Connected Port Lane Receiver Trained	1'b0	R	Connected port lane receiver trained.
[26:24]	Received Port Width	3'b0	R	Port width of connected port.
[23:20]	Lane Number in Connected Port	4'b0	R	Lane number within connected port.
[19:18]	Connected Port Transmit Emphasis Tap(-1) Status	2'b0	R	Tap(-1) status. • 2'b00 - Tap(-1) not implemented • 2'b01 - Tap(-1) at minimum emphasis • 2'b10 - Tap(-1) at maximum emphasis • 2'b11 - Tap(-1) at intermediate emphasis setting
[17:16]	Connected Port Transmit Emphasis Tap(+1) Status	2'b0	R	Tap(+1) status. • 2'b00 - Tap(+1) not implemented • 2'b01 - Tap(+1) at minimum emphasis • 2'b10 - Tap(+1) at maximum emphasis • 2'b11 - Tap(+1) at intermediate emphasis setting
15	Connected Port Scrambling/ Descramblin g Enabled	1'b0	R	When set, indicates that scrambling/descrambling is enabled on the connected port.
[14:0]	Reserved	15'b0	R	

Implementation-defined Space

The RapidIO Specification does not define the behavior of the registers in the Implementation-defined Space. The BUF CSRs reside within this space starting at offset 0x010000, and are contained in the BUF. The Maintenance Request Information Register at offset 0x10100 is located in the LOG. This register only exists when the AXI4-Lite Maintenance port is being used. It exists behind the User Maintenance Interface and cannot be accessed by another AXI4 master (only via maintr transactions or requests received from the link partner).

Watermarks CSR

This register contains the watermark values used to determine packet priority allowance when in transmitter controlled flow control. It is mandatory that this register exhibits the condition 64 > WM0 > WM1 > WM2 > 0 at all times to ensure proper operation. It is recommended that the size of the buffer be taken into account when assigning the watermark values. Too high of a watermark value for a smaller buffer could cause the core to lock up. If there is concern about the size of the buffer, safe watermark values are WM0=3, WM1=2, WM2=1. It is also recommended that traffic flow is halted prior to writing this register.

Table 2-54: **Watermarks CSR (Offset 0x0)**

Bits	Fields	Reset Value	Access	Description
[31:22]	Reserved	10'h000	R	
[21:16]	WM2	CORE Generator	RW	Watermark 2 defines the minimum buffer space necessary within the link partner to transmit a priority 2 packet. WM1 > WM2 > 0
[15:14]	Reserved	2'b000	R	
[13:8]	WM1	CORE Generator	RW	Watermark 1 defines the minimum buffer space necessary within the link partner to transmit a priority 1 packet. WM0 > WM1 > WM2
[7:6]	Reserved	2'b000	R	
[5:0]	WM0	CORE Generator	RW	Watermark 0 defines the minimum buffer space necessary within the link partner to transmit a priority 0 packet. 64 > WM0 > WM1

Buffer Control CSR

This register contains status information for the buffer and allows control of Buffer features.

Table 2-55: **Buffer Control CSR (Offset 0x4)**

Bits	Field	Reset Value	Access	Description
31	RX Flow Control Only	CORE Generator	R	BUF was generated with RX flow control only and does not support TX flow control.
30	Unified Clock	CORE Generator	R	BUF was generated as a unified clock implementation and does not contain clock synchronization logic. Care must be taken to ensure the LOG and PHY clocks are the same. This option should not be used if the core will encounter a traindown situation.
29	TX Flow Control	CORE Generator	R	When asserted to 1'b1, indicates the BUF is currently operating in TX flow control mode.
28	TX Request Reorder	CORE Generator	R	When asserted to 1'b1, indicates the Transmit Buffer has been configured to allow reordering of requests.
27	Reserved	1’b0	R	
[26:16]	TX Size	CORE Generator	R	Number of total unacknowledged packets the TX Buffer can hold. Possible values are 8, 16 or 32.
15	Force RX Flow Control	CORE Generator	RW	Force a TX Flow Control enabled core to use RX flow control. This bit is not writable if the core was generated with the RX Flow Control Only option (in which case, bit 31 will be set).
[14:8]	Reserved	7'h00	R	
[7:0]	RX Size	CORE Generator	R	Number of maximum-sized packets the RX Buffer can hold. Possible values are 8, 16 or 32.

Maintenance Request Information Register

This CSR is used to populate the fields of remote requests sent via the AXI4-Lite Maintenance port. The register is stored within the main LOG rather than with the Serial RapidIO Registers accessible through the Configuration Register Interface. To set the Destination ID, TID, priority, or CRF independently for each outgoing maintenance request, this register must be written before each remote maintenance request.

Table 2-56: **Maintenance Request Information Register (Offset 0x0)**

Bits	Field	Reset Value	Access	Description
[31:24]	Request TID	8'b0	RW	The TID that will be used for the next outgoing maintenance request. The value will increment after each request is sent.
[23:19]	Reserved	5'b0	R	

Table 2-56: **Maintenance Request Information Register (Offset 0x0)**

Bits	Field	Reset Value	Access	Description
[18:17]	Request Priority	2'b01	RW	Priority that will be used for outgoing maintenance requests.
[16]	Request CRF	1'b0	RW	CRF value that will be used for outgoing maintenance requests. This bit is read-only if the CRF support is disabled in the GUI.
[15:0]	Request Destination ID	16'b0	RW	DestID that will be used for outgoing maintenance requests.

Designing with the Core

This chapter includes guidelines and additional information to make designing with the core easier.

Xilinx Solution

The Xilinx multi-core approach to Serial RapidIO ensures flexibility while still offering the ease of a drop-in solution. This section details the SRIO Gen2 Endpoint, the sub-cores included and the reference design provided.

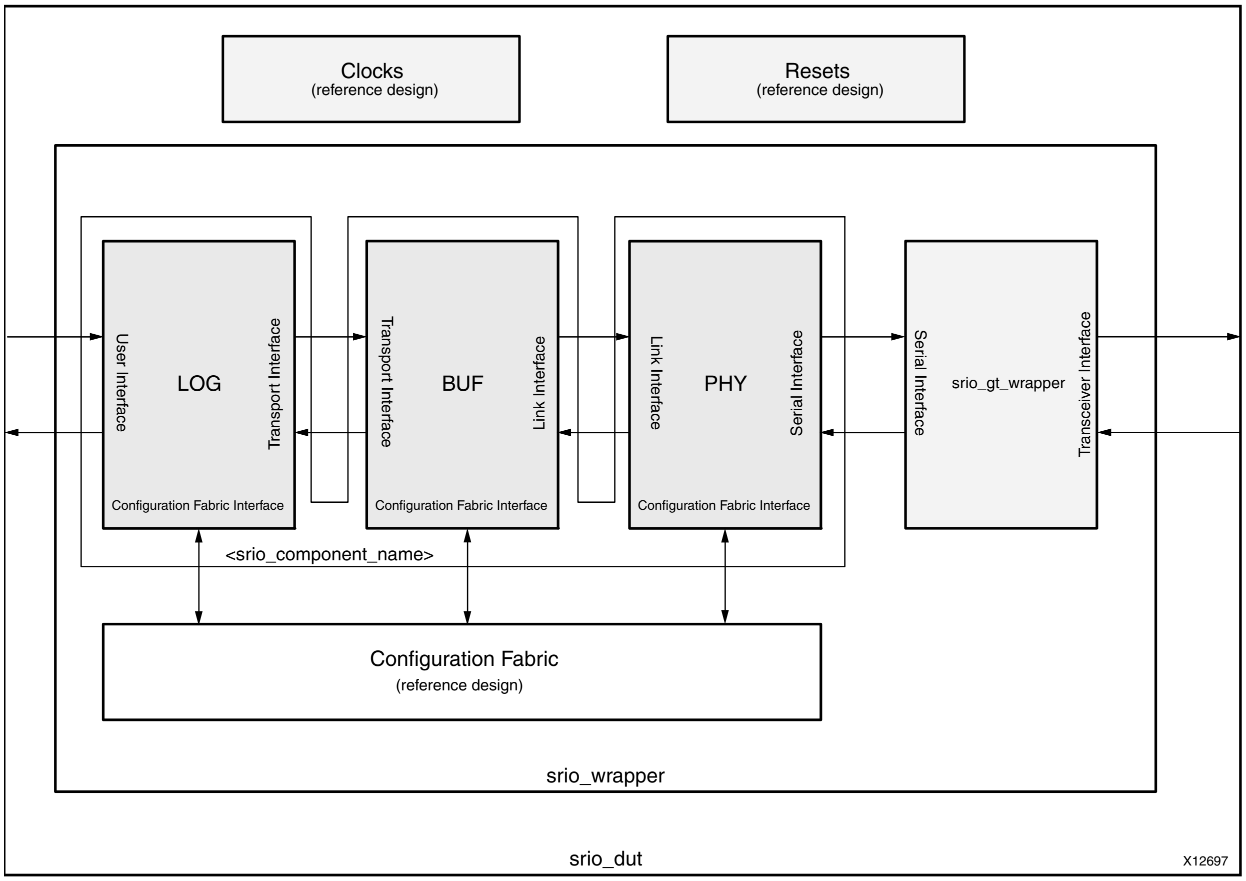


Figure 3-1: **SRIO Gen2 Endpoint Gen2 System Overview**

Figure 3-1 shows how the SRIO Gen2 Endpoint is structured. It is possible to use the full solution as highlighted in this chapter or to interface into the core at the BUF or PHY. The modularity of the SRIO Gen2 Endpoint allows users to swap out the Xilinx offering with a custom design at any of those layers.

Each section within this chapter details functionality the advanced user may need to access and expectations as RapidIO packets proceed through each level.

General Design Guidelines

Transaction Types

The RapidIO Specification defines several transaction types. Each transaction type performs a different function. Transaction support for the core is set through the CORE Generator GUI.

Table 3-1 lists the defined transaction types and indicates which LOG ports the transactions belong with. A transaction is considered unsupported if it is not one of the defined transaction types below or if support for the transaction was not enabled in the GUI. If a received transaction is not supported, it will be presented on the User-Defined port. If the User-Defined port does not exist, the transaction will be dropped within the core and the `port_decode_error` signal will assert.

Table 3-1: **Supported Transaction Types and Corresponding Ports**

Packet Type	FTYPE	TTYPE	Transmit Port	Receive Port	Description
NREAD	0010	0100	• Condensed I/O: iotx • Initiator/Target: ireq	• Condensed I/O: iorx • Initiator/Target: treq	Basic read request transaction. Request does not have a data payload. Results in a response with data.
NWRITE	0101	0100	• Condensed I/O: iotx • Initiator/Target: ireq	• Condensed I/O: iorx • Initiator/Target: treq	Basic write operation. Request has a data payload. Does not result in a response.
NWRITE_R	0101	0101	• Condensed I/O: iotx • Initiator/Target: ireq	• Condensed I/O: iorx • Initiator/Target: treq	Basic write operation. Request has a data payload. Results in a response with no data.
SWRITE	0110	N/A	• Condensed I/O: iotx • Initiator/Target: ireq	• Condensed I/O: iorx • Initiator/Target: treq	Streaming write operation (uses less header fields than NWRITE). Request has a data payload. Does not result in a response.

Table 3-1: **Supported Transaction Types and Corresponding Ports *(Cont'd)***

Packet Type	FTYPE	TTYPE	Transmit Port	Receive Port	Description
DOORBELL	1010	N/A	• Condensed I/O: iotx • Initiator/Target: ireq	• Condensed I/O: iorx • Initiator/Target: treq	Very short message between processing elements. Request has no data payload. Results in a response with no data.
MESSAGE	1011	N/A	• If Separate Messaging port is used: msgireq • If Messages are combined with I/O, ∘ Condensed I/O: iotx ∘ Initiator/Target: ireq	• If Separate Messaging port is used: msgtreq • If Messages are combined with I/O, ∘ Condensed I/O: iorx ∘ Initiator/Target: treq	Messaging operation - typically used by processors in distributed memory system machines. Request has no data payload. Results in a message response (with no data).
ATOMIC with no payload	0010	1100 1101 1110 1111	• Condensed I/O: iotx • Initiator/Target: ireq	• Condensed I/O: iorx • Initiator/Target: treq	Read-modify-write operation. Request does not have a data payload. Results in a response with data.
ATOMIC with payload	0101	1100 1101 1110	• Condensed I/O: iotx • Initiator/Target: ireq	• Condensed I/O: iorx • Initiator/Target: treq	Read-modify-write operation. Request has a data payload. Results in a response with data.
RESPONSE without data	1101	0000	• Condensed I/O: iotx • Initiator/Target: tresp	• Condensed I/O: iorx • Initiator/Target: iresp	Response transaction with no data payload.
RESPONSE with data	1101	1000	• Condensed I/O: iotx • Initiator/Target: tresp	• Condensed I/O: iorx • Initiator/Target: iresp	Response transaction with a data payload.
MESSAGE RESPONSE	1101	0001	• If Separate Messaging port is used: msgtresp • If Messages are combined with I/O, ∘ Condensed I/O: iotx ∘ Initiator/Target: tresp	• If Separate Messaging port is used: msgiresp • If Messages are combined with I/O, ∘ Condensed I/O: iorx ∘ Initiator/ Target: iresp	Response to a MESSAGE transaction. Does not contain a data payload.

Table 3-1: **Supported Transaction Types and Corresponding Ports *(Cont'd)***

Packet Type	FTYPE	TTYPE	Transmit Port	Receive Port	Description
MAITNENANCE READ REQUEST	1000	0000	• If AXI4-Lite Maintenance port is used: maintr_ar • If Direct Maintenance Access is selected: maintreq	• If AXI4-Lite Maintenance port is used, received transactions are handled by the core. • If Direct Maintenance Access is selected: maintrx	Configuration space read transaction. Does not contain a data payload. Results in MAINTENANCE READ RESPONSE.
MAITNENANCE WRITE REQUEST	1000	0001	• If AXI4-Lite Maintenance port is used: maintr_w and maintr_aw • If Direct Maintenance Access is selected: maintreq	• If AXI4-Lite Maintenance port is used, received transactions are handled by the core. • If Direct Maintenance Access is selected: maintrx	Configuration space write transaction. Contains a data payload. Results in MAINTENANCE WRITE RESPONSE.
MAITNENANCE READ RESPONSE	1000	0010	• If AXI4-Lite Maintenance port is used, responses are generated by the core. • If Direct Maintenance Access is selected: maintresp	• If AXI4-Lite Maintenance port is used: maintr_r • If Direct Maintenance Access is selected: maintrx	Configuration space read transaction. Does not contain a data payload. Results in MAINTENANCE READ RESPONSE.
MAITNENANCE WRITE RESPONSE	1000	0011	• If AXI4-Lite Maintenance port is used, responses are generated by the core. • If Direct Maintenance Access is selected: maintresp	• If AXI4-Lite Maintenance port is used: maintr_b • If Direct Maintenance Access is selected: maintrx	Configuration space write transaction. Contains a data payload. Results in MAINTENANCE WRITE RESPONSE.

Logical Layer AXI4-Stream Serial RapidIO Interface Usage

The User Interface ports (with the possible exception of the Maintenance port) are built from AXI4-Stream interfaces.

Interface Overview

The AXI4-Stream protocol uses a ready/valid handshake to transfer information from a master to a slave component. It uses a last beat indicator to designate packet boundaries.

There is also a signal with byte enable information, indicating which of the bytes in the current data beat are good. Finally, there are data and user signals which carry the actual packet information.

HELLO Packet Format

In order to simplify packet construction, the user interface ports can be configured to use the Header Encoded Logical Layer Optimized (HELLO) format. This format allows for standardization of header field placement across packet types. It also segments the header and data into separate transfers on the interface. This leads to simpler control logic and allows data to be aligned to transfer boundaries for easier data management.

Certain fields are unnecessary or contain modified content for some transaction types. A more detailed view of the HELLO-formatted header based on the packet's FTYPE is shown in Figure 3-2.

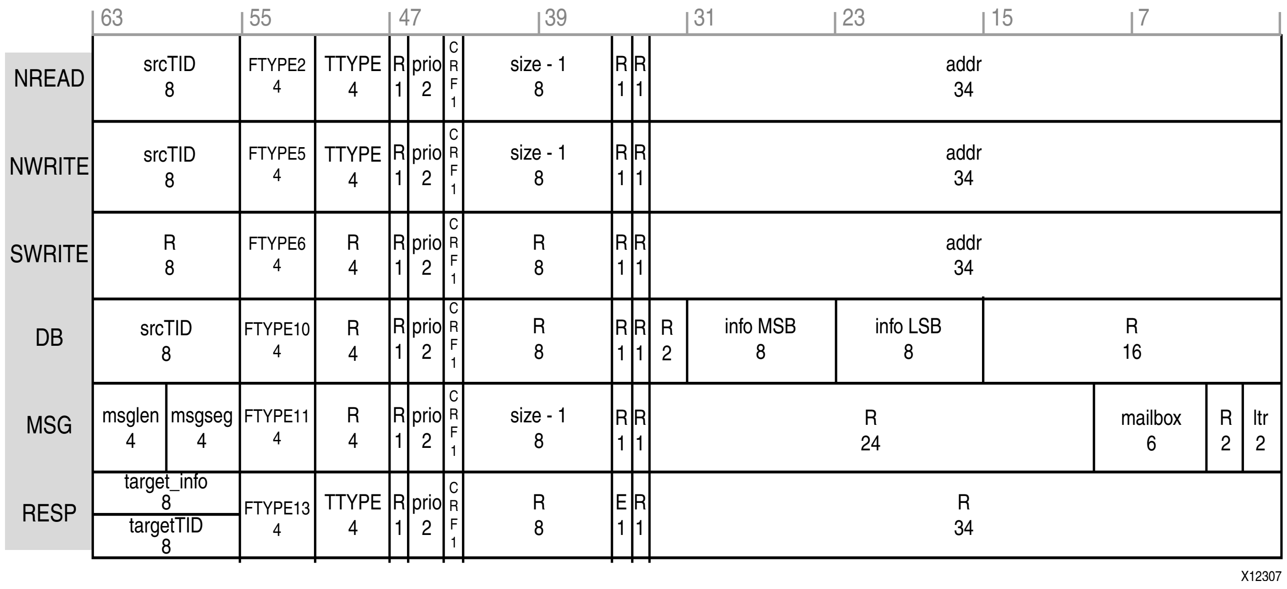


Figure 3-2: **HELLO FTYPE Packet Format**

Table 3-2 explains the HELLO fields in more detail.

Table 3-2: **HELLO Format Detail**

Field	Bits	Description
TID	[63:56]	The Transaction ID for the packet. The RapidIO Specification allows for only one outstanding packet with a given TID and Source/Destination ID pair at any given time.
FTYPE	[55:52]	The Transaction Class for the packet. Supported FTYPEs for HELLO format are 2, 5, 6, A, B, and D. The core may only support a subset of these, based on selections made when the core was generated.

Table 3-2: **HELLO Format Detail *(Cont'd)***

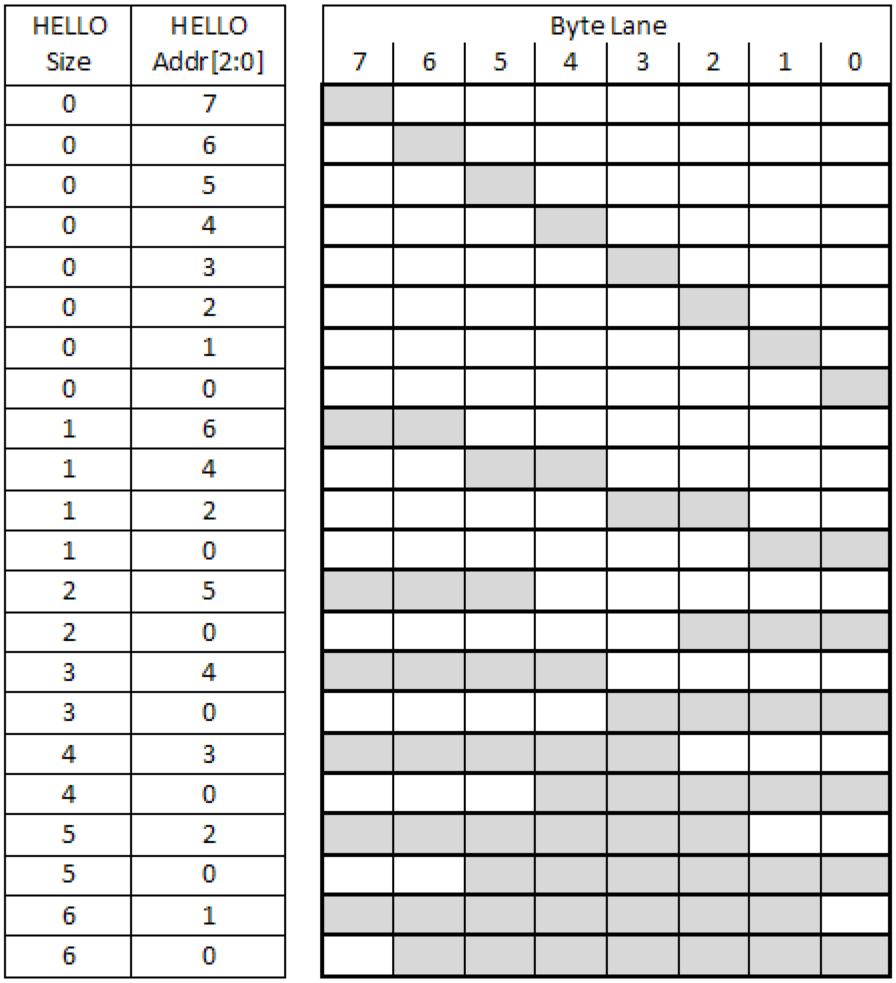
Field	Bits	Description
TTYPE	[51:48]	Transaction Type for the packet. Only used for FTYPE 2, 4 and 13 to define additional functionality within the Transaction Class.
priority	[46:45]	Priority for the packet. Request packets must use priorities 0-2 only. Response packet priority should be request priority +1.
CRF	[44]	Critical Request Flow flag for the packet.
size	[43:36]	Data payload size in bytes, minus one. Limitations exist; read this section thoroughly.
error	[35]	When set, indicates that the packet has an error status. Reserved, except for RESPONSE packets.
address	[33:0]	Byte address for the transaction. Limitations exist; read this section thoroughly.
info	[31:16]	Info field; applies to DOORBELL packets only.
msglen-1	[63:60]	The number of packets making up a MESSAGE sequence; applies to MESSAGEs only.
msgseg-1	[59:56]	The message segment represented by this packet; applies to MESSAGEs only. Reserved for single-segment messages.
mailbox	[9:4]	Mailbox targeted by the transaction; applies only to MESSAGE packets. Upper four bits are reserved except for single-segment messages.
letter	[1:0]	Letter field for MESSAGE packets only. Indicates a slot within a mailbox.

The status of a response will be truncated to a single bit. If the status of a received packet is DONE, the HELLO Error (E) field will be zero. Any other response status will result in the Error bit being set.

The size field in the HELLO packet is the number of bytes in the transfer minus one (valid range is 0 to 255, which corresponds to a true size of 1 to 256 bytes). The size and address fields must correspond to valid size, address, and wdptr fields for the corresponding RapidIO packet type. This leads to some constraints for these values.

Sub-DWORD Accesses

For transfers of fewer than eight bytes, the address and size are used to determine the active byte lanes for the transfer (the TKEEP bus must be tied to all ones). Only combinations resulting in a valid rdsize/wrsize and wdptr are allowed. Figure 3-3 shows HELLO address and size combinations that result in valid byte lanes (highlighted in gray).



HELLO Size	HELLO Addr[2:0]	Byte Lane 7	6	5	4	3	2	1	0
0	7	▨							
0	6		▨						
0	5			▨					
0	4				▨				
0	3					▨			
0	2						▨		
0	1							▨	
0	0								▨
1	6	▨	▨						
1	4			▨	▨				
1	2					▨	▨		
1	0							▨	▨
2	5	▨	▨	▨					
2	0						▨	▨	▨
3	4	▨	▨	▨	▨				
3	0					▨	▨	▨	▨
4	3	▨	▨	▨	▨	▨			
4	0				▨	▨	▨	▨	▨
5	2	▨	▨	▨	▨	▨	▨		
5	0			▨	▨	▨	▨	▨	▨
6	1	▨	▨	▨	▨	▨	▨	▨	
6	0		▨	▨	▨	▨	▨	▨	▨

Figure 3-3: **Valid Size and Address Combinations for Sub-DWORD accesses**

Large Accesses

LOG transfers longer than one DWORD (eight bytes) that start on a byte offset other than zero must be broken into multiple transfers including sub-DWORD transactions for unaligned segments. Alternatively, reads may be handled by increasing the read size to the next supported size and extracting the data necessary from the corresponding response.

Accordingly, the three least significant bits of the address must be zero for transactions that are one DWORD or larger. The RapidIO Specification defines a subset of supported sizes in the range from 1 to 256 bytes for use in read and write transactions. Requests greater than a DWORD should be rounded up to the next closest supported value. The valid HELLO sizes for read and write transactions are: 7, 15, 31, 63, 95 (reads only), 127, 159 (reads only), 191 (reads only), 223 (reads only), and 255.

For write sizes intermediate to these sizes, only the necessary amount of data to send should be provided to the core before the channel's `tlast` signal asserts. Only the provided data will be sent. Similarly, the user may be provided write data that falls short of the advertised size. The available data should be written and the transfer should be presumed complete.

The RapidIO Specification does not support data transfer sizes of greater than 256 bytes, and the LOG does not break large transactions into smaller segments. Failure to comply with this requirement may lead to a fatal link error in which the link partner continually retries the oversized packet.

The HELLO header is presented on the first transfer on the User Interface ports. If there is data associated with the transaction, it will be presented on subsequent beats. The Source and Destination IDs for the packet are placed on the user signal and sampled on the first transfer for each packet. The user signal is ignored after the first beat.

Figure 3-4 shows a typical transfer with data on one of the User Interface ports. This particular transfer has a data payload of four DWORDs (32 bytes). On the interface, it takes five total cycles including the transfer of the header.

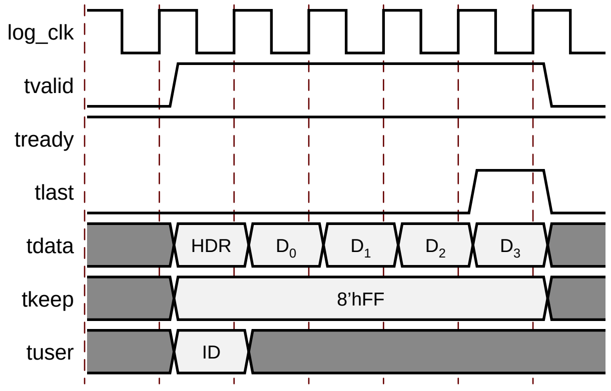


Figure 3-4: **Basic HELLO Packet Transfer on User Interface**

Figure 3-5 shows a more complicated traffic flow. First, there are two back-to-back single cycle (header-only) packets. The packet boundaries are indicated by the assertion of the TLAST signal. After the single-cycle packets, the master waits one cycle before starting the next packet. In the third packet, both the master and slave each stall the interface on a different cycle (by deasserting TVALID and TREADY respectively). This packet had two

DWORDs of payload and thus only three active cycles on the interface, but took a total of five clock cycles due to the stalls.

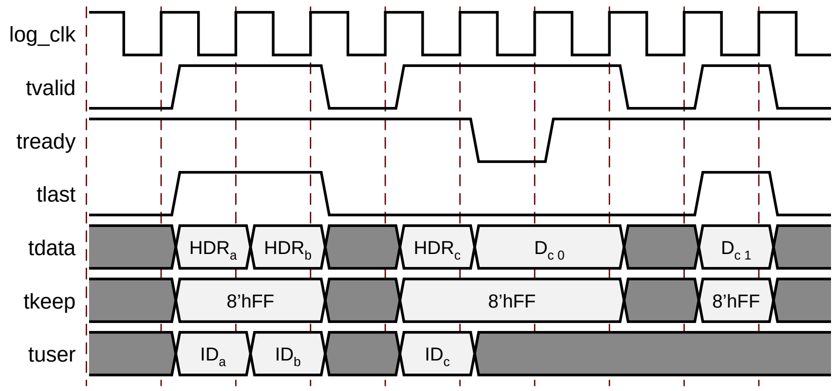


Figure 3-5: **Advanced HELLO Packet Transfer on User Interface**

SRIO Stream Packet Format

User Interface ports can be configured to use the SRIO Stream format for maximum control. In this format, packets are presented fully formed as defined in the RapidIO Specification including all Logical/Transport layer fields (plus the priority which the specification defines as a Physical Layer field). The data bytes must be reversed before presentation to the LOG in this format in order to comply with AXI4-Stream protocol. Packets on the Link Interface (between the BUF and the PHY) and Transport Interface (between the LOG and the BUF) use the SRIO Stream format.

Figure 3-6 shows a typical transfer with data on one of the User Interface ports. This particular transfer has a data payload of five DWORDs (32 bytes). On the interface, it takes five total cycles and the applicable CRF/Response bits are set on the first cycle.

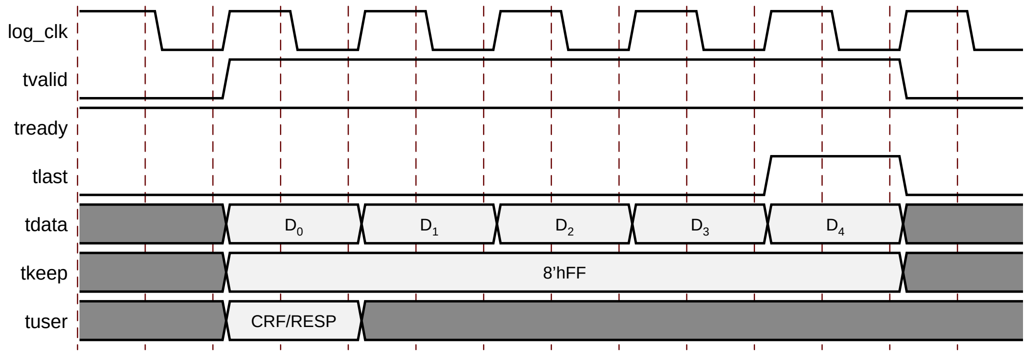


Figure 3-6: **Basic SRIO Stream Packet Transfer**

Figure 3-7 and Figure 3-8 show a more complicated traffic flow. First, in Figure 3-7 there is a single cycle packet. Next, the master waits one cycle before sending two back-to-back 2-DWORD packets. The packet boundaries are indicated by the assertion of the TLAST signal. In the packet in Figure 3-8, both the master and slave each stall the interface on a different cycle (by deasserting TVALID and TREADY respectively). This packet had three DWORDs and thus only three active cycles on the interface, but took a total of five clock cycles due to the stalls.

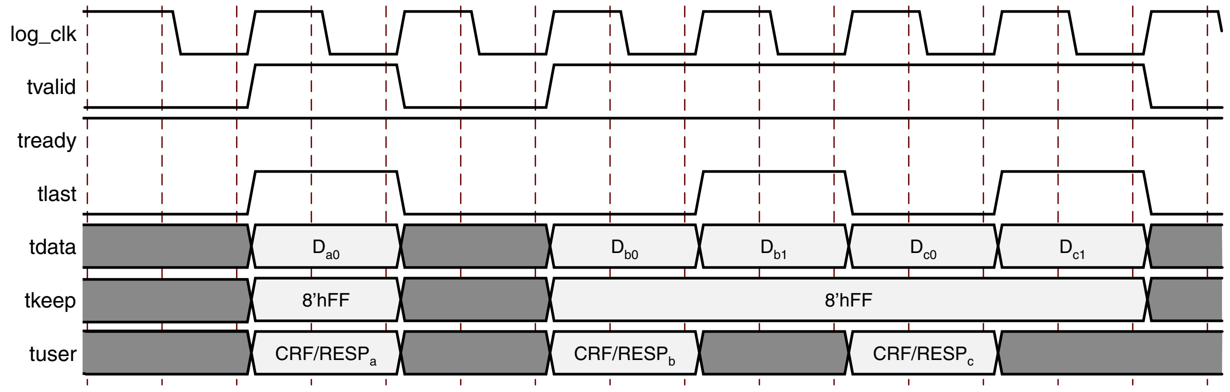


Figure 3-7: **Advanced SRIO Stream Packet Transfer: Part 1**

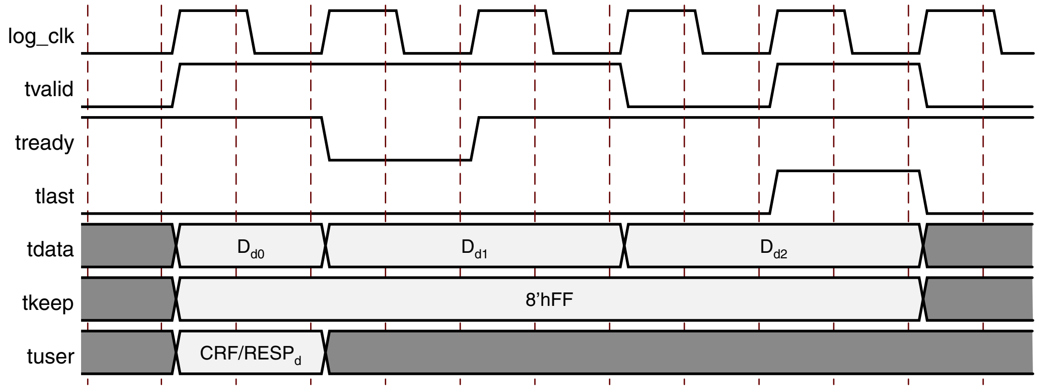


Figure 3-8: **Advanced SRIO Stream Packet Transfer: Part 2**

Accessing the Configuration Space

Each processing element connected through the RapidIO fabric has capability registers (CARs) and command and status registers (CSRs). These can be accessed to determine or set the capabilities, configuration, and status of the device. The configuration registers are defined in the *RapidIO Specification* for each layer. See Chapter 2, Product Specification for detailed information about these registers in the Xilinx Configuration Space.

The registers are 32-bits. All configuration register read and write accesses are performed in word (4-byte) increments. The SRIO Gen2 Endpoint does not support other sizes for Maintenance transactions, including LCSBA transactions. Read and write accesses to reserved register offsets terminate normally and do not result in an error condition. Similarly, writes to CARs (read-only registers) terminate normally and do not result in an error condition.

Using the AXI4-Lite Maintenance Port

Maintenance Write Example

For write transactions, both the write address and write data must be transferred on their respective channels on the Maintenance port before the write is forwarded. When a response is received by the LOG, the Maintenance block will return a status on the write response channel. The Maintenance block only accepts one write at a time (new address and data will not be accepted until the response is transferred to the user).

Figure 3-9 is a timing diagram showing an example of two write transactions on the Maintenance port. Notice that since the address and data are on separate channels, they can be transferred on the port at any time with respect to one another.

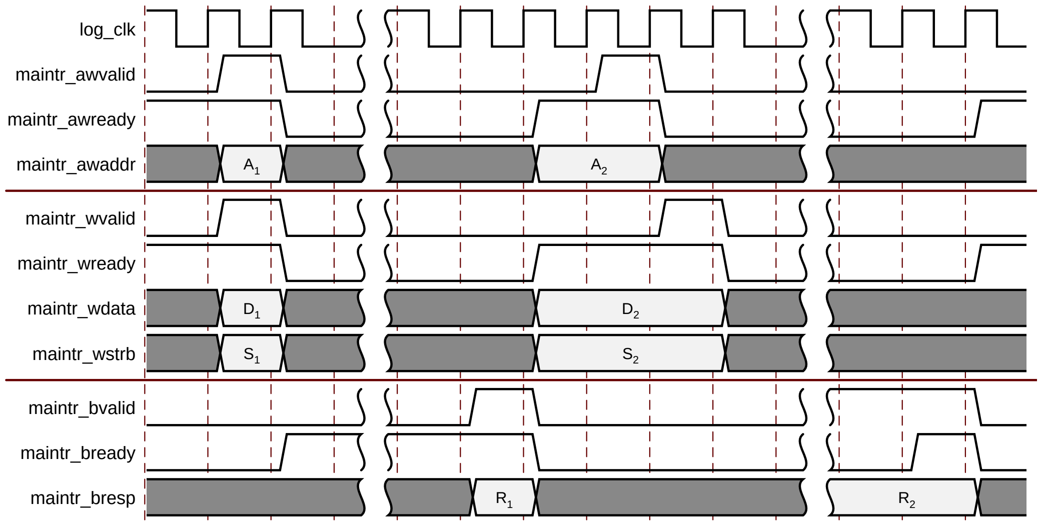


Figure 3-9: **User Maintenance Write Transaction Example**

Maintenance Read Example

A read transaction is forwarded immediately once the read address is transferred on the Maintenance port. When the response is received by the LOG, the Maintenance block will return a status on the read response channel. The Maintenance block only accepts one read at a time (a new address will not be accepted until the response is transferred to the user).

Figure 3-10 is a timing diagram showing an example of a read transaction on the Maintenance port.

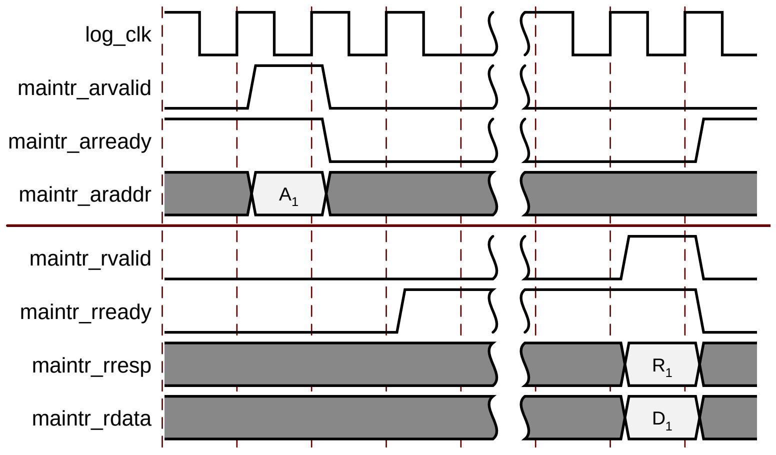


Figure 3-10: **User Maintenance Read Transaction Example**

Remotely Accessing the Xilinx Configuration Space

Description

The primary method for accessing the CARs and CSRs that control the LOG, BUF, and PHY functionality from a remote device is through maintenance packets. This includes both maintenance read and write transactions. The configuration space can also be accessed through regular read and write operations, rather than maintenance transactions. This occurs when a NREAD or NWRITE hits the address defined within the Local Configuration Space Base Address (LCSBA) CSR. Specifically, when the two *xamsbs* bits and the eight most significant bits of the address equal the value in the LCSBA CSR, the read or write request is then treated like a maintenance transaction.

To write to the configuration space of the endpoint, the remote endpoint generates a MAINTENANCE WRITE REQUEST. If the AXI4-Lite Maintenance Ports are in use, the write will be presented on the cfgr interface. Then, the Configuration Fabric that is part of the reference design will route the write to the appropriate configuration space based on the address of the write. The AXI4-Lite slaves that interface to the configuration registers will process the write and return a response. After completing the write, the LOG will issue a MAINTENANCE WRITE RESPONSE back to the remote endpoint indicating the status of the write.

To read from the configuration space of the endpoint, the remote endpoint generates a MAINTENANCE READ REQUEST. If the AXI4-Lite Maintenance Ports are in use, the read will be presented on the CFGR AXI-4-Lite interface. Then, the Configuration Fabric that is part of the reference design will route the read to the appropriate configuration space based on the address of the read (see Table 2-23).

The AXI4-Lite configuration slave will process the read and return a response with data and status information. After issuing the reads, the LOG will issue a single MAINTENANCE READ RESPONSE back to the remote endpoint indicating the status of the read and containing the requested data.

Interface

There are no signals required to consume an incoming MAINTENANCE WRITE or READ REQUEST, nor to generate the corresponding MAINTENANCE WRITE or READ RESPONSE when using the AXI4-Lite Maintenance interfaces and the Configuration Fabric reference design delivered with the SRIO Gen2 Endpoint. If the provided reference design is not used, the user is responsible for decoding Maintenance transactions and routing them to the AXI4-Lite Configuration Register Interfaces contained within each sub-core.

Remote Maintenance Access Detail

This example details the path followed by a remote MAINTENANCE WRITE transaction when the AXI4-Lite Maintenance Interfaces are in use. The remote endpoint issues a MAINTENANCE REQUEST (ftype=MAINTENANCE and ttype=WRITE REQUEST or READ REQUEST) transaction generated from the remote endpoint. The status is returned through a MAINTENANCE RESPONSE (ftype=MAINTENANCE and ttype=WRITE RESPONSE or READ RESPONSE). Figure 3-11 shows the flow of the transactions through the RapidIO system. The various transfer points shown in this illustration are explained in the following section.

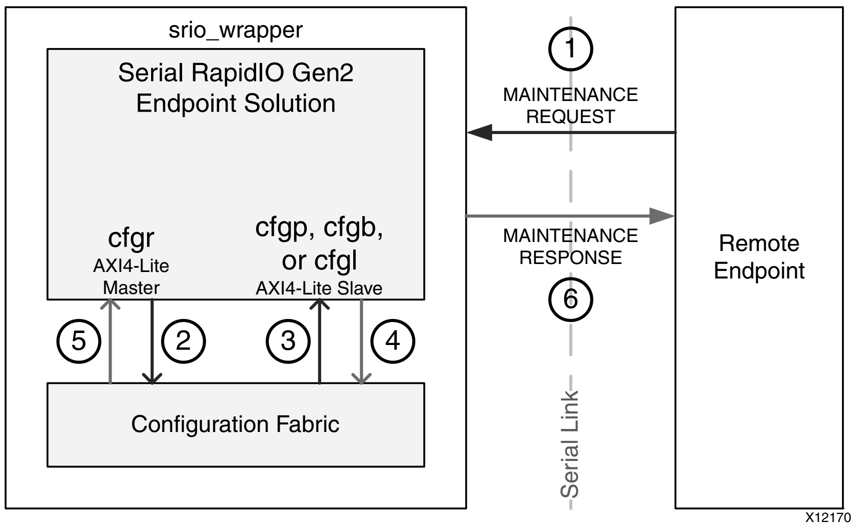


Figure 3-11: **REMOTE MAINTENANCE Transaction Flow**

1. Remote endpoint sends MAINTENANCE REQUEST to Local Endpoint

   The remote endpoint initiates the configuration register access by sending a MAINTENANCE REQUEST across the link.

2. LOG presents read or write to Configuration Fabric

The MAINTENANCE REQUEST passes through the various components of the SRIO Gen2 Endpoint within the local endpoint. The request is then decoded in the LOG and AXI4-Lite Configuration Bus on the read or write channel as appropriate for the transaction type. The Configuration Fabric accepts the request on the CFGR interface.

3. Configuration Fabric presents read or write to appropriate configuration space

   The Configuration Fabric reference design sends each 32-bit configuration access to a configuration slave interface (LCFG - LOG, BCFG - BUF, or PCFG - PHY) interface. The selected configuration slave depends on the configuration space offset used.

4. Configuration slave returns AXI4-Lite response

   The targeted configuration register is read or written as directed by the Configuration Fabric, then it returns an AXI-Lite response with the status (and the data in the case of a read).

5. Configuration Fabric returns response to LOG

   The Configuration Fabric passes the response including the status of the transaction (and data in the case of a read) back to the LOG.

6. Local Endpoint returns MAINTENANCE RESPONSE to the Remote Endpoint

   The LOG forms the MAINTENANCE RESPONSE, using information from the request packet and from the response returned by the configuration fabric. The MAINTENANCE RESPONSE packet is passed through the SRIO Gen2 Endpoint across the link to the requesting remote endpoint to complete the maintenance request-response transaction. This response includes a status indicating whether the request was processed successfully and data for unerrored reads.

Special Considerations for Accessing the Xilinx Configuration Space Remotely

If the SRIO Gen2 Endpoint receives a transaction of an unsupported type (LCSBA hit for any packet types except NREAD, NWRITE, and NWRITE_R), it will return an error response. In the case of an errored write, the core will not update the configuration register with the write data from the request. An error response will be also returned for MAINTENANCE REQUESTs or LCSBA hits with a request size of anything but one word (four bytes).

Using SRIO Gen2 Endpoint to Access Local Configuration Space

Description

The method for accessing the CARs and CSRs that control the LOG, BUF, and PHY functionality from within the local device is through local maintenance packets. This includes both maintenance read and write transactions. The AXI4-Lite Maintenance port only allows one outstanding read and one outstanding write at any given time. Particularly remote requests can take a significant amount of time to complete. Traffic flow must be planned accordingly.

Interface

The user will generate the local configuration request on the Maintenance port (see Maintenance Port in Chapter 2 for more details).

Local Configuration Access Example

This example details the process for reading or writing to a local configuration register using the AXI4-Lite Maintenance Interface. Figure 3-12 shows the flow of the transactions through the local endpoint.

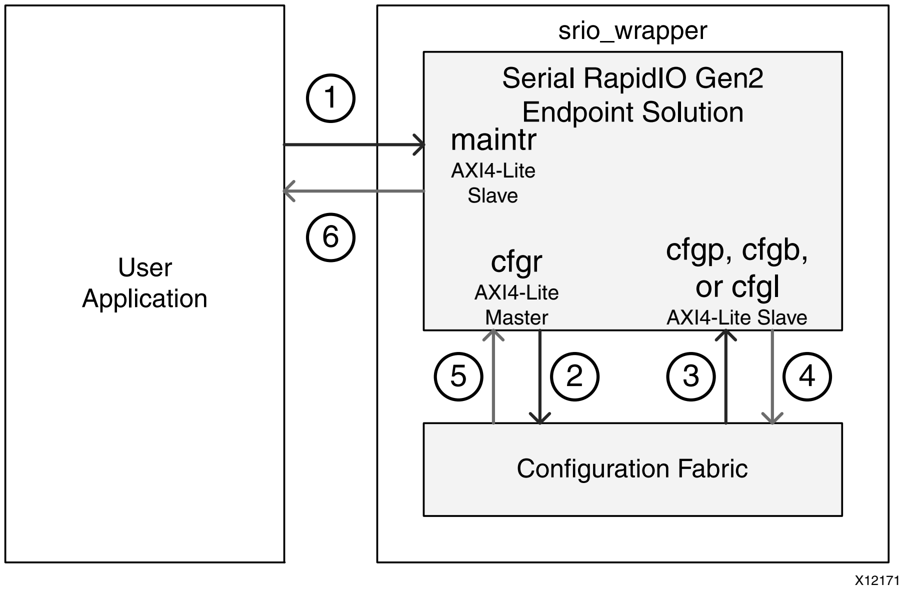


Figure 3-12: **Local MAINTENANCE Transaction Flow**

1. User generates the request

   To initiate the access to the local configuration space, the user creates an AXI4-Lite read or write on the Maintenance port (see Using the AXI4-Lite Maintenance Port).

   When setting up the address for the write, the offset of the register is the address of that register. For example, the Base Device ID CSR resides at offset 0x60. This address is passed along through Read Address or Write Address Channel on the Maintenance port. However, configuration offsets are only 24 bits while the `maintr_awaddr` and `maintr_araddr` inputs are 32 bits. The upper eight bits are used to set the hop count for the transaction. So for a local write, address bits [31:24] should be zero. For maintenance packets, configuration offsets are placed on `maintr_awaddr[23:0]` or `maintr_araddr[23:0]` for writes and reads, respectively. Note that because registers are only addressed on a word (four-byte) basis, the lower two bits of address will be ignored.

The user presents the address on the appropriate address input, then asserts VALID for the channel. For write accesses, 32 bits of data are also presented by the user on the write data channel while asserting the maintr_wvalid signal. Once the address (and data in the case of a write) has been accepted into the core, the LOG will process the transaction.

2. LOG presents transaction to Configuration Fabric

   When the Configuration Fabric Interface is available, the transaction will be presented to the Configuration Fabric through the cfgr AXI4-Lite Master.

3. Configuration Fabric forwards transaction to Configuration Registers

   The Configuration Fabric decodes the address of the transaction and presents it to the appropriate Configuration Register Interface.

4. Configuration Register response returned to Configuration Fabric

   After the register is read or updated, a response will be presented back from the Configuration Register Interface to the Configuration Fabric.

5. Configuration Fabric returns response to LOG

   The Configuration Fabric forwards the response back to the LOG on the cfgr interface.

6. Response provided to user

   The LOG will return a response including status information (and data for reads) to the User on the appropriate Response Channel of the Maintenance port. See Using the AXI4-Lite Maintenance Port for more information.

Using SRIO Gen2 Endpoint to Access Remote Configuration Space

Description

The primary method for reading or writing configuration registers of a remote device connected through the RapidIO fabric is through maintenance transactions. This includes both maintenance read and write transactions. The configuration space can also be accessed through regular read and write operations, rather than maintenance transactions. This occurs when a NREAD, NWRITE, or SWRITE hits the address defined within the Local Configuration Space Base Address (LCSBA) CSR. Specifically, when the two *xamsbs* bits and the eight most significant bits of the address equal the value in the LCSBA CAR, the read or write request is then treated like a maintenance transaction. LCSBA transactions targeting remote configuration space must be performed using the I/O ports.

The process for accessing remote configuration space with MAINTENANCE transactions using the AXI4-Lite Maintenance port is very similar to the process used to access the local configuration space with the SRIO Gen2 Endpoint.

Interface

The user will generate the local configuration request on the Maintenance port (see Maintenance Port in Chapter 2 description for more details).

Remote Configuration Access Example

This example details the process for reading from or writing to a remote endpoint configuration register using a MAINTENANCE REQUEST (ftype=MAINTENANCE and ttype=READ REQUEST or WRITE REQUEST) transaction from the local endpoint. It also tracks the corresponding MAINTENANCE RESPONSE (ftype=MAINTENANCE and ttype=READ RESPONSE or WRITE RESPONSE) back to the requesting local endpoint. Figure 3-13 shows the flow of the transactions through the RapidIO system.

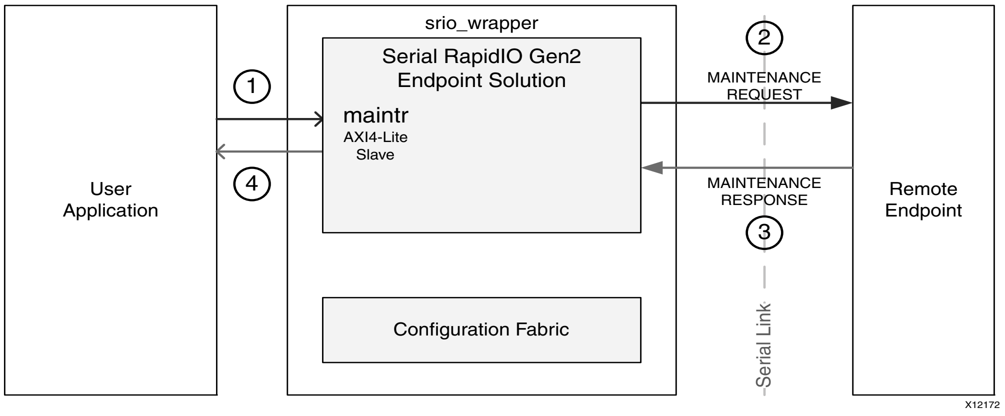


Figure 3-13: **MAINTENANCE Transaction Flow**

1. User generates MAINTENANCE request

To initiate the access to the remote configuration space, the user creates an AXI4-Lite read or write on the Maintenance port (see Using the AXI4-Lite Maintenance Port).

When setting up the address for the write, the offset of the register is the address of that register. For example, the Base Device ID CSR resides at offset 0x60. This address is passed along through the Read Address or Write Address Channel on the Maintenance port. However, configuration offsets are only 24 bits while the Maintenance port address fields are 32 bits. The upper byte of the address is used to convey hop count information. If the upper eight bits of address are greater than zero, the LOG will send the transaction to the remote endpoint. It will subtract one from that 8-bit value and use that number as the hop count in the MAINTENANCE REQUEST. Since switches do not normally have their own Device ID, it would not be possible to route maintenance packets to them. To circumvent this, hop count is used to specify the number of switches (or hops) between the source and destination devices. When a switch does not have a Device ID associated with it, it examines the hop count field of an incoming maintenance packet. If it is zero, the maintenance packet is for that switch, and the switch acts upon it. If it is not zero, the switch decrements the hop count by one and passes the maintenance packet along to the next device. If targeting an endpoint, the hop count should be set to 0xFF because endpoints do not check the hop count when determining if a transaction is intended for it. This is also an easy method to ensure an intermediate switch does not accidently grab the transaction for itself.

Note that because the LOG only supports word (four-byte) Maintenance Requests, the lowest two bits of address will be ignored.

The MAINTENANCE REQUEST will use the value stored in the Base Device ID CSR as the Source ID for the transaction. This value is also available on the deviceid output from the LOG. The priority, CRF, and Destination ID fields will be populated with information from the Maintenance Request Information Register (see Maintenance Request Information Register in Chapter 2). To override these values, that register must be written before the remote request is presented to the LOG. The TID used for the Request is also stored in the Maintenance Request Information Register. The LOG starts with a TID of 0 and increments it for every remote request that it generates. The TID that will be used on the next request can be read in this register, or can be overwritten with a register write. The process for accessing the Maintenance Request Information Register is the same as that described in Using SRIO Gen2 Endpoint to Access Local Configuration Space.

2. MAINTENANCE REQUEST transmitted to remote device

The LOG will form the MAINTENANCE REQUEST packet, which will be forwarded across the link into the RapidIO system.

3. Remote device sends MAINTENANCE RESPONSE to SRIO Gen2 Endpoint

The targeted device will create a MAINTENANCE RESPONSE and send it back to the SRIO Gen2 Endpoint.

4. Response returned to User

Once the MAINTENANCE RESPONSE is received by the LOG, it will check the TID against that of the request. If there is a match, the status of the response will be returned to the user on the appropriate Maintenance Response channel. For a read response, the data from the MAINTENANCE RESPONSE will also be returned on the read response channel of the Maintenance port. See the Using the AXI4-Lite Maintenance Port for signaling information.

Special Considerations for Accessing Remote Configuration Space with the AXI4-Lite Maintenance Port

The LOG allows four-byte configuration accesses only. There can be only one outstanding read and one outstanding write access at any given time (local or remote). The LOG will drop MAINTENANCE responses with TIDs that do not match that of the outstanding request. This is done silently and is implemented this way for AXI4-Lite compliance.

- A reset is provided on the Maintenance port so that a timeout can be implemented to prevent a lost or improperly formed response from locking up the port indefinitely. When `maintr_rst` is asserted, outstanding maintenance packets will be cancelled. A typical application will use the link_timeout output (which reflects the value stored in the Link Timeout CSR) to set the timeout counter which is only active when a request is outstanding and clears to 0 when a response is received. It is recommended that the Maintenance port be static at the time of the reset (in other words, do not send a read while attempting to cancel a response-less write). Once the requests have been dropped via maintr_rst, the stored TIDs are cleared for both the read and write paths. This means that if a response is received after the reset assertion for a previous request on either channel, it will be dropped silently.

Clocking

Core Clocks

The PHY operates on two clock domains: phy_clk, which is the primary core clock, and gt_pcs_clk, which is used for the Serial Transceiver interface. The gt_clk is not used by the PHY, but is used by the Serial Transceiver interface. The gt_pcs_clk is half the rate of gt_clk. As a general rule, phy_clk is equal to (gt_clk * operating link width)/4. So for a core operating at 2x, phy_clk is half the frequency of gt_clk. If the core trains down to 1x mode, phy_clk must switch to a quarter of the gt_clk rate. The Serial Transceivers also require a reference clock (refclk) that uses the dedicated clock pins of the transceiver. The reference clock frequency is selected when the core is generated (the available options depend on architecture and line rate).

Table 3-3: **Reference Clock Available Per Line Rate**

Line Rate (Gbps)	refclk 125 MHZ	refclk 156.25 MHz
1.25	x	
2.5	x	
3.125	x	x
5	x	
6.25		x

The LOG operates on the log_clk domain. For optimum throughput, log_clk should be at least as fast as phy_clk.

The BUF transfers packets between the log_clk and phy_clk domains. If the BUF was generated for unified clocks, log_clk and phy_clk must be synchronous. Otherwise, the clocks must match the rates of the interfacing sub-cores.

The cfg_clk domain on each sub-core's Configuration Register interface is independent of that sub-core's clock(s). However, in order to use the LOG's Maintenance Controller with the provided Configuration Fabric reference design, the cfg_clk for all these interfaces must be equivalent to log_clk.

Table 3-4 through Table 3-6 show the typical clock frequencies based on link width.

Table 3-4: **Typical Clock Rates for 4x Operation**

Line Rate (Gbps)	gt_clk (MHz)	gt_pcs_clk (MHz)	Full (4x) phy_clk (MHz)	Traindown (1x) phy_clk (MHz)	log_clk, cfg_clk (MHz)
1.25	62.5	31.25	62.5	15.63	62.5
2.5	125	62.5	125	31.25	125
3.125	156.25	78.13	156.25	39.06	156.25
5.0	250	125	250	62.5	250
6.25	312.5	156.25	312.5	78.13	312.5

Table 3-5: **Typical Clock Rates for 2x Operation**

Line Rate (Gbps)	gt_clk (MHz)	gt_pcs_clk (MHz)	Full (2x) phy_clk (MHz)	Traindown (1x) phy_clk (MHz)	log_clk, cfg_clk (MHz)
1.25	62.5	31.25	31.25	15.63	31.25
2.5	125	62.5	62.5	31.25	62.5
3.125	156.25	78.13	78.13	39.06	78.13
5.0	250	125	125	62.5	125
6.25	312.5	156.25	156.25	78.13	156.25

Table 3-6: **Typical Clock Rates for 1x Operation**

Line Rate (Gbps)	gt_clk (MHz)	gt_pcs_clk (MHz)	phy_clk (MHz)	log_clk, cfg_clk (MHz)
1.25	62.5	31.25	15.63	15.63
2.5	125	62.5	31.25	31.25
3.125	156.25	78.13	39.06	39.06
5.0	250	125	62.5	62.5
6.25	312.5	156.25	78.13	78.13

Clocking Reference Design

The example design takes a single differential input clock, `sys_clk`, and instantiates the clock buffers and clock modules needed to generate the clocks. The clocking scheme varies slightly between FPGA families to accommodate the specific architecture of each device.

7 Series Device Clocking

For the 7 series FPGAs, an MMCM is used to generate the clocks from the GT reference clock. A block diagram of the clocking scheme is shown in Figure 3-14.

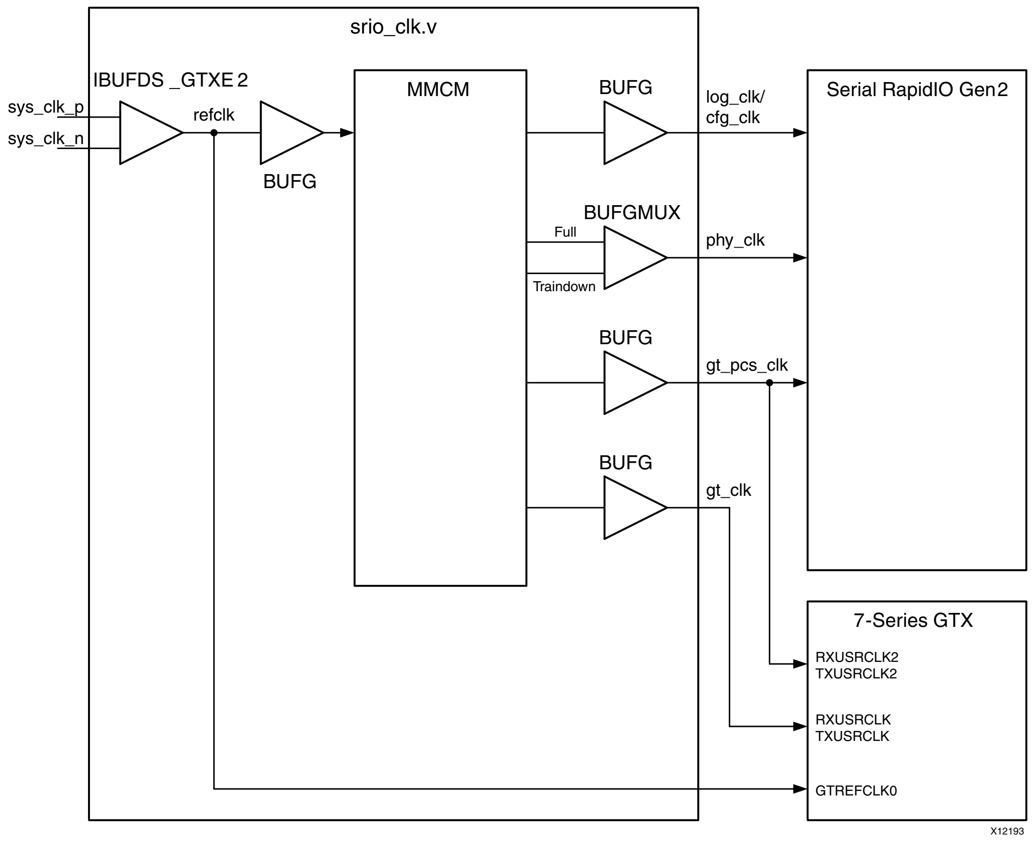


Figure 3-14: **Clocking for 7 Series FPGAs**

Note that MMCM multiplier and divider values are dependent on the reference clock frequency and the line rate. In 4x configurations, `log_clk` and `gt_clk` share a BUFG. In 1x configurations, `log_clk` and `phy_clk` share a BUFG (and no BUFGMUX is needed because there is only one possible `phy_clk` rate). In addition, if the Unified Clock option is selected in the GUI, `log_clk` and `phy_clk` are required to be the same rate. This means the BUFG for the `log_clk/cfg_clk` can be removed, and the `log_clk/cfg_clk` is tied to the `phy_clk`.

Virtex-6 Device Clocking

For Virtex-6 FPGAs, an MMCM is used to generate the clocks from the GT reference clock. A block diagram of the clocking scheme is shown in Figure 3-15.

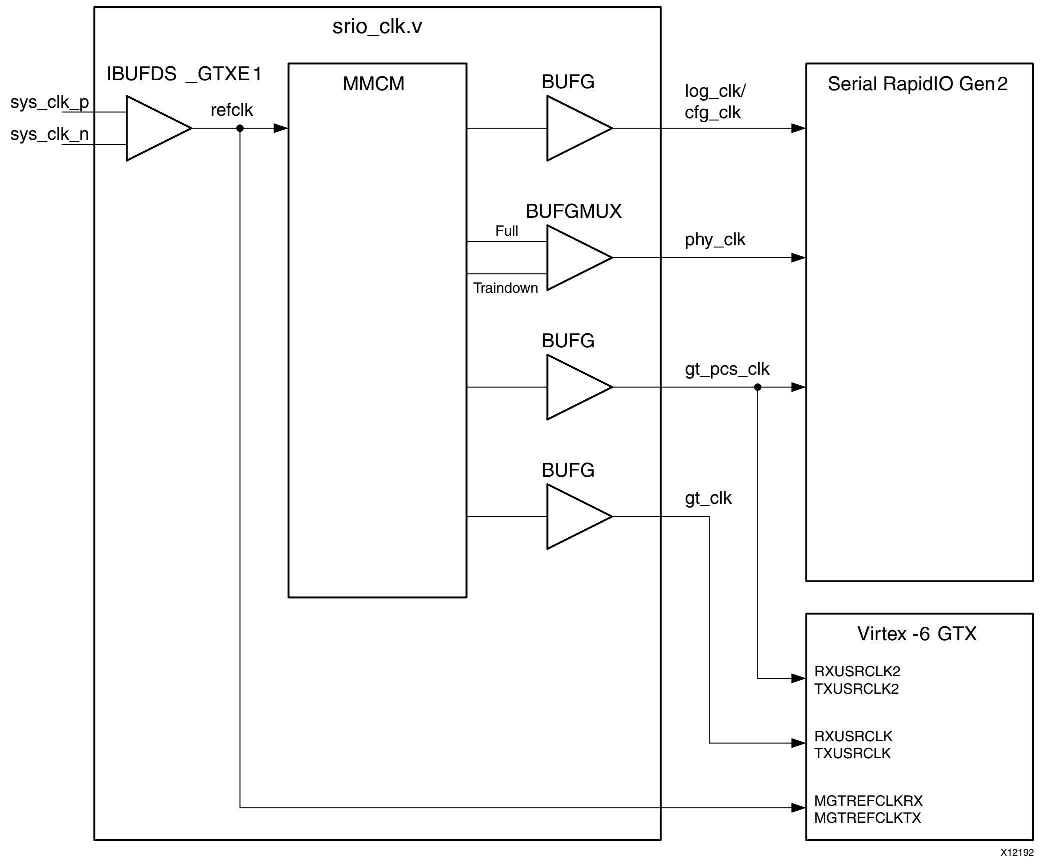


Figure 3-15: **Virtex-6 FPGA Clocking**

Note that MMCM multiplier and divider values are dependent on the reference clock frequency and the line rate. In 4x configurations, `log_clk` and `gt_clk` share a BUFG. In 1x configurations, `log_clk` and `phy_clk` share a BUFG (and no BUFGMUX is needed because there is only one possible `phy_clk` rate). In addition, if the Unified Clock option is selected in the GUI, the `log_clk` and the `phy_clk` are required to be the same rate. This means the BUFG for the `log_clk/cfg_clk` can be removed, and the `log_clk/cfg_clk` is tied to the `phy_clk`.

Potential Clock Resource Savings

There are multiple possibilities for reducing the number of clock resources and saving certain resources such as the BUFGs. The following sections describe potential solutions to clocking resource management.

Unified Clock

Unified clock settings indicate `log_clk/cfg_clk` and `phy_clk` all run at the same speed. This allows the BUFG for `log_clk/cfg_clk` to be removed and to tie `phy_clk` to `log_clk/cfg_clk`. If traindown is supported for the generated core and a traindown scenario occurs, the `log_clk/cfg_clk` changes as `phy_clk` changes.

Replace BUFG with BUFH

A BUFG can be replaced with a BUFH. The BUFH is similar to BUFG except that a BUFH can only clock a single clocking region. This requires certain parts of the core to fit within a single clock region.

One specific example is the BUFGs for the `gt_clk` and the `gt_pcs_clk` can be replaced with BUFHs. This requires the GTs and part of the PHY to fit within a single clock region, which it currently does. The LOC constraint may be required to place the BUFH, GTs, and PHY in the same region.

Another example is the BUFG for the `phy_clk` can be replaced with a BUFH if there is no requirement for the core to support traindown. This scenario also requires the BUFH and PHY to be in the same clock region.

Note that every BUFG replaced with a BUFH makes it more difficult to meet timing for the core because of the single clocking region restrictions.

Resource Sharing

Several aspects of resource sharing can occur when multiple instances of the SRIO Gen2 Endpoint are implemented, depending on the configuration and usage of the RapidIO system. If the same line rates, link widths, and user interface clocks are used for each instance (without traindown support), all of the clocking resources can be shared. In other words, if two or more clock resources operate with the same rate, they can be shared. Below are some examples of how resources can be shared.

If a design has different line rates, link widths, and may need to support traindown, but has the same user interface clocks for each instance of the RapidIO system, the BUFG for the `log_clk/cfg_clk` can be shared. Figure 3-16 shows an example of log_clk/cfg_clk resource sharing.

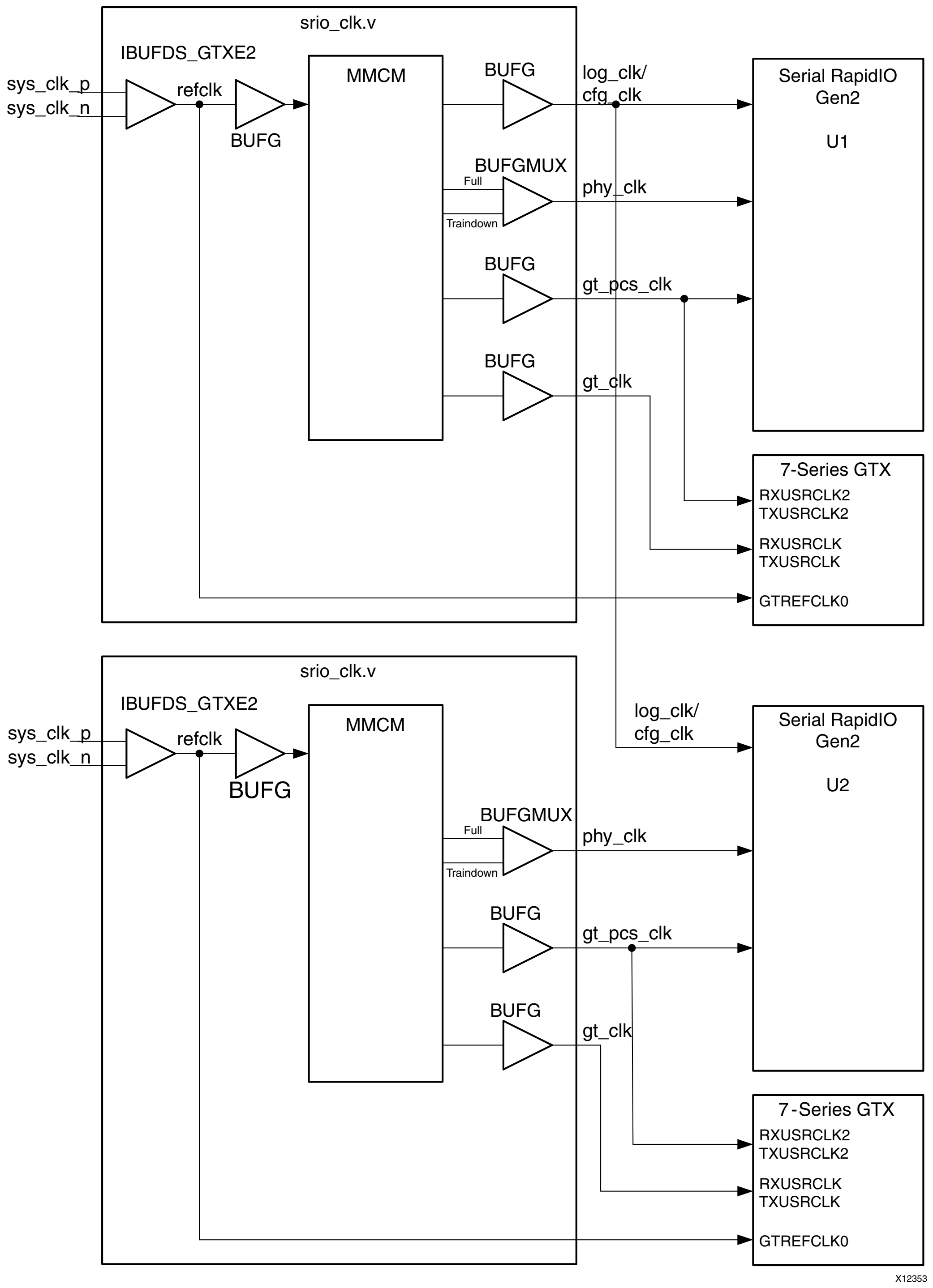


Figure 3-16: **log_clk/cfg_clk Resource Sharing**

If a design has the same line rates, but supports different link widths and traindown, and has a different user interface clocks for each instance of the RapidIO system, the BUFGs for the `gt_clk` and `gt_pcs_clk` can be shared. Figure 3-17 shows an example of `gt_clk` resource sharing.

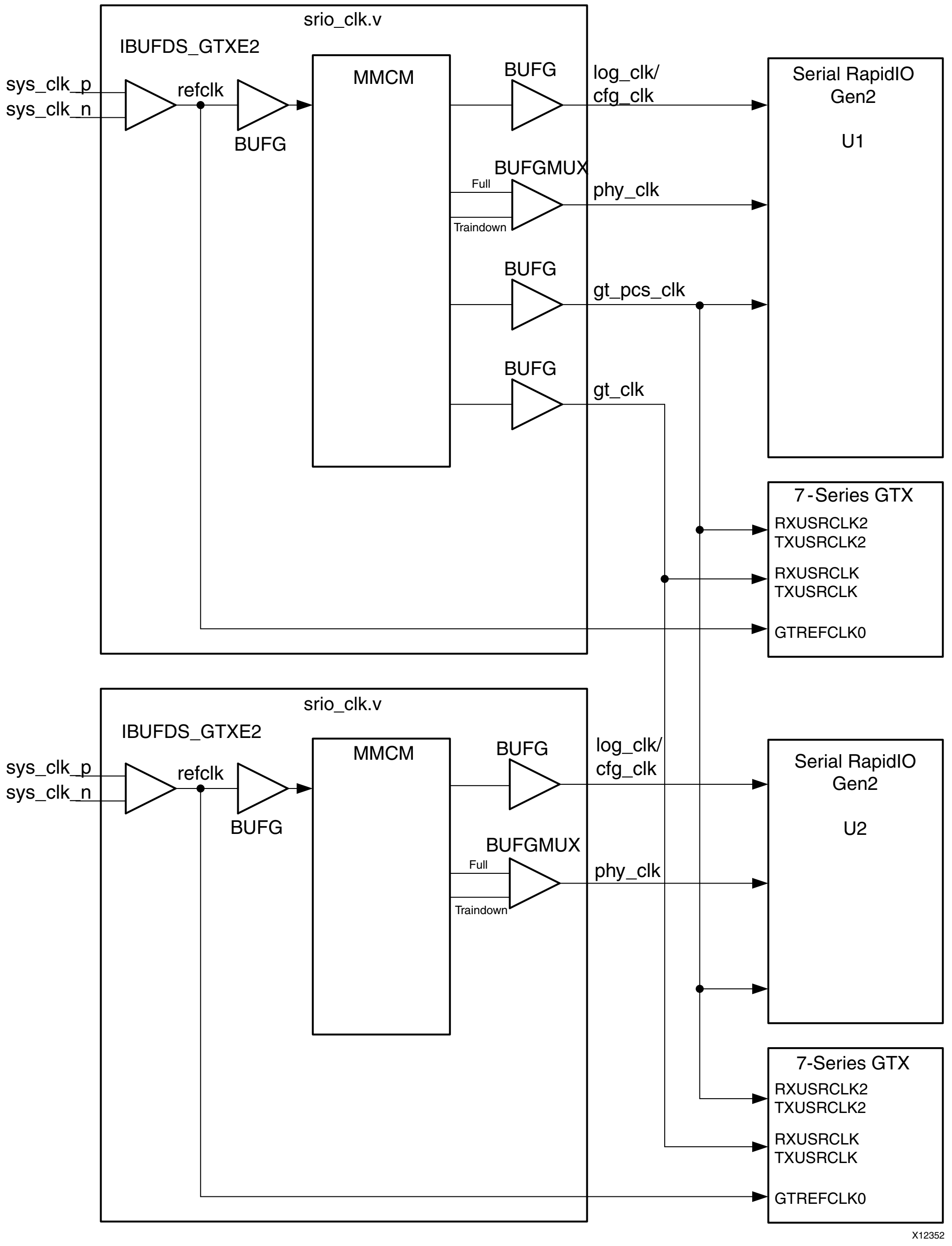


Figure 3-17: **gt_clk/gt_pcs_clk Resource Sharing**

Figure 3-18 shows all the resources being shared. This scenario assumes all the line rates, link widths, and user interfaces are the same, and that traindown is not supported. In this case, the MMCM and BUFGs are shared between two instances of the RapidIO system. In this scenario, another refclk may need to be brought in depending on place and route restrictions of the refclk and the core.

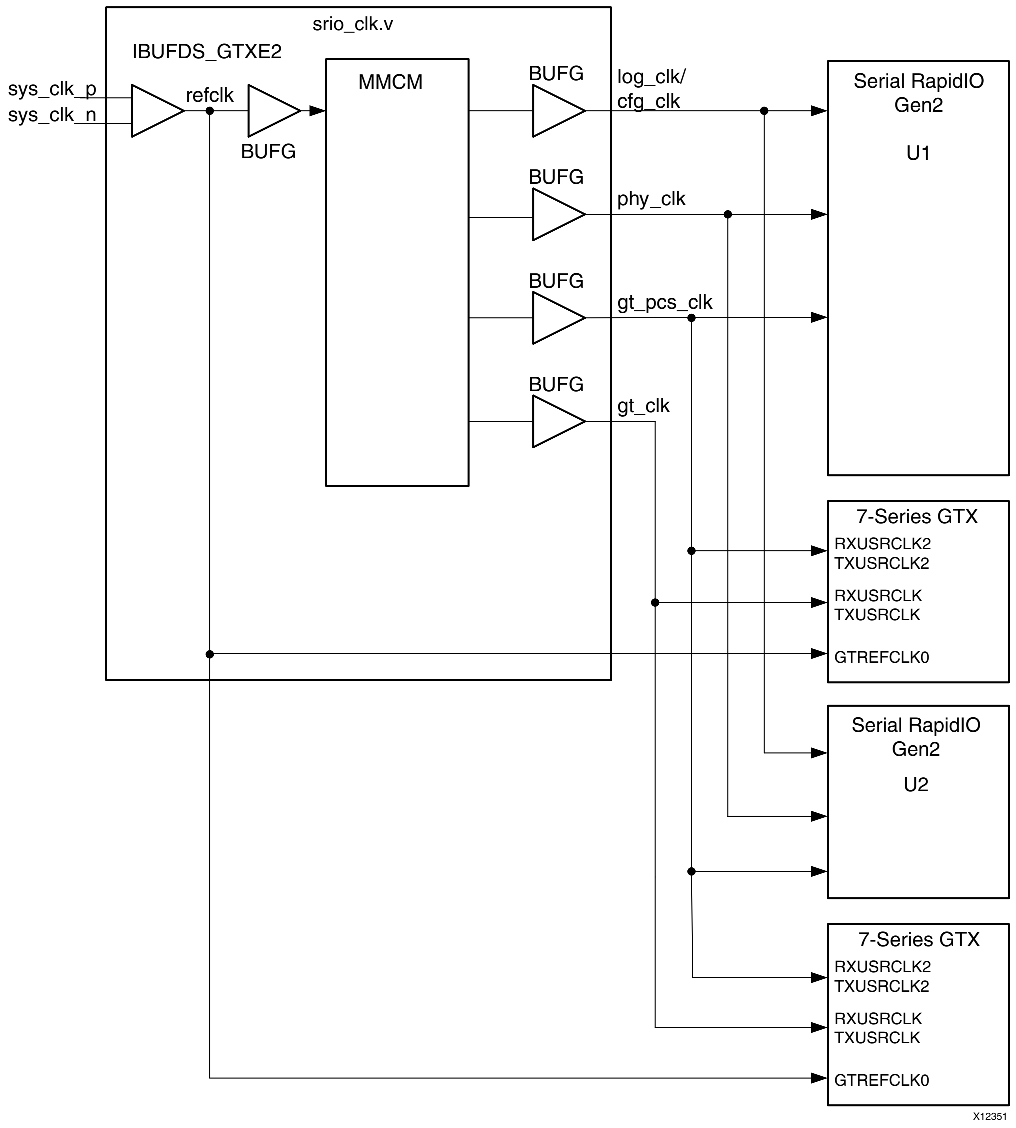


Figure 3-18: **All Clock Resource Sharing**

Resets

Each clock domain has an associated reset. The reset should be asserted for a minimum of four clock cycles of the respective clock domain and deassert synchronously (note that if the core is trained down, `phy_clk` will run slower than the original rate and the reset must still assert for four full cycles).

The included reset reference design module (`srio_rst.v`) has a single reset input, `sys_rst`. This signal is an asynchronous input. This module synchronizes the reset to each clock domain and extends the pulse to meet the minimum reset cycle requirements.

The initial hardware reset should be generated by the user. Resets may also be communicated in-band using the RapidIO protocol. Special care must be used when resetting the SRIO Gen2 Endpoint device. Both link partners should be reset together to guarantee ackID alignment. It is recommended that resets between link partners be overlapped to reduce the occurrence of lost packets and control symbols. One way to implement this is to handshake the core resets. Resets received from the link partner will be communicated to the user with the assertion of `phy_rcvd_link_reset` from the core. The `sys_rst` signal should be asserted upon receipt of a link reset. Depending on the implementation, a reset can also be signaled to the user application in response to the assertion of phy_rcvd_link_reset. To send a reset request to the link partner, assert the `phy_link_reset` signal until the port_initialized output goes low. At this time, `sys_rst` should be asserted to the reset reference design, completing the handshake.

Protocol Description

Transaction Classes

The SRIO Gen2 Endpoint supports the encoding and decoding of seven format types through the HELLO interface. Many format types also have transaction types associated with them. Some formats also have format-specific fields associated with them. Due to the variable nature of these packet headers, it is necessary to reuse fields within the encoded header on a format-by-format basis.

FTYPE2 - Request Class

Request packets allow the generating endpoint to read back data from another SRIO Gen2 Endpoint. The SRIO Gen2 Endpoint may be configured to support Request Class on a Common port, Initiator/Target port or Initiator/Target Read port. There are four Atomic transactions and a basic read associated with the Request Class. All Request transactions are a single beat on the AXI4-Stream interface and are only the encoded header.

- **NREAD (TTYPE = 4'b0100)**. Standard read transaction. Associated response are of the size provided in the size field.
- **ATOMIC Increment (TTYPE = 4'b1100)**. Post-read increment transaction. Associated response returns current value in the associated address space. Once the response is sent, the value in the associated location is incremented and updated. ATOMIC transactions can only be byte size (size-1 = 0), half-word size (size-1 = 1) or word size (size-1 = 3). Because of these size limitations, all responses are two beats: a header followed by a single DWORD of data.

- **ATOMIC Decrement (TTYPE = 4'b1101)**. Post-read decrement transaction. Associated response returns current value in the associated address space. Once the response is sent, the value in the associated location should be decremented and updated. ATOMIC transactions can only be byte size (size-1 = 0), half-word size (size-1 = 1) or word size (size-1 = 3). Because of these size limitations, all responses are two beats: a header followed by a single DWORD of data.
- **ATOMIC Set (TTYPE = 4'b1110)**. Post-read set transaction. Associated response returns the current value in the associated address space.

  Once the response is sent, the value in the associated location should be set to all 1s. ATOMIC transactions can only be byte size (size-1 = 0), half-word size (size-1 = 1) or word size (size-1 = 3). Because of these size limitations, all responses are two beats: a header followed by a single DWORD of data.
- **ATOMIC clear (TTYPE = 4'b1111)**. Post-read clear transaction. Associated response returns the current value in the associated address space. Once the response is sent, the value in the associated location should be set to all 0s. ATOMIC transactions can only be byte size (size-1 = 0), half-word size (size-1 = 1) or word size (size-1 = 3). Because of these size limitations, all responses are two beats: a header followed by a single DWORD of data.

FTYPE5 - Write Class

Write packets allow the end-point element to modify data within a link partner's memory space. The SRIO Gen2 Endpoint can be configured to support Write Class on a Common port, Initiator/Target port or Initiator/Target Write port. There are three Atomic transactions and two write transactions (one with a response, one without) associated with the Write Class. All write transactions are between 2 and 33 beats on the AXI4-Stream interface and are composed of the encoded header followed by a maximum of 32 dwords of data.

- **NWRITE (TTYPE = 4'b0100)**. Write transaction, no response. Standard write transaction allows for sub-dword or multi-dword writes. Receiving endpoint should perform write to specified address. Due to the responseless nature of this packet type, TID value is ignored/undefined.
- **NWRITE_R (TTYPE = 4'b0101)**. Write transaction, expect response. Standard write transaction allows for sub-dword or multi-dword writes. Receiving endpoint should perform write to specified address, and then send a single beat response with a status of OK or ERROR back to the initiating endpoint.
- **ATOMIC Swap (TTYPE = 4'b1100)**. Post-read write transaction. The associated response should return the current value in the targeted address space. Once the response is sent, the value in the targeted location should be replaced with the write data. ATOMIC transactions can only be byte size (size-1 = 0), half-word size (size-1 = 1) or word size (size-1 = 3). Because of these size limitations, all requests and responses are two beats: a header followed by a single DWORD of data.

- **ATOMIC Compare-and-Swap (TTYPE = 4'b1101)**. Post-read compare and swap transaction. The associated response should return the current value in the targeted address space. Once the response is sent, the value in the targeted location should be compared with the first eight bytes of payload. If the two values are equal, then the second DWORD of data is written to the identified memory location. The compare-and-swap transaction must send both a compare value as well as a write value. So, the request transactions are three beats on the user interface: header, compare value, and write value. All responses are two beats: a header followed by a single DWORD of data. ATOMIC transactions can only be byte size (size-1 = 0), half-word size (size-1 = 1) or word size (size-1 = 3).
- **ATOMIC Test-and-Swap (TTYPE = 4'b1110)**. Post-read test and swap transaction. The associated response should return the current value in the targeted address space. Once the response is sent, if the value at the identified address is zero, then the second DWORD of data must be written to the identified memory location. ATOMIC transactions can only be byte size (size-1 = 0), half-word size (size-1 = 1) or word size (size-1 = 3). Because of these size limitations, all requests and responses are two beats: a header followed by a single DWORD of data.

FTYPE6 - Streaming Write Class

Streaming write packets allow the endpoint element to modify large chunks of data within a link partner's memory space. The SRIO Gen2 Endpoint can be configured to support Streaming Write Class on a Common port, Initiator/Target port or Initiator/Target Write port. SWrite transactions are meant to be the most efficient of the SRIO transactions across the link interface and therefore lack many of the fields in other transactions. The SWrite Class uses only the ftype, priority and address fields. TID is not needed because no response is sent for this packet type. Size is not needed because all transactions are DWORD multiples. Streaming write transactions are between 2 and 33 beats on the AXI4-Stream interface and are composed of the encoded header followed by a maximum of 32 DWords of data.

FTYPE10 - Doorbell Class

Doorbell packets contain a 16-bit info field but no address or data. In order to make the HELLO interfaces as efficient as possible, part of the HELLO address field is used to pass Doorbell information. The SRIO Gen2 Endpoint can be configured to support Doorbell Class on a Common port or Initiator/Target port. Both Doorbell packets and their associated responses are a single beat (encoded header only) on the AXI user interface.

FTYPE11 - Message Class

Message packet header fields are very different than the other transaction classes. Because addressing for messages is handled by the receiving endpoint, messages reference a mailbox and letter for delivery with only one message sequence allowed to each mailbox/ letter combination at any one time. Each message packet is still restricted to the 256 byte data limit, but sequences of messages can be sent in order to build larger data messages.

The msglen field indicates how many message packets will make up the larger message sequence. Each packet within the message sequence is provided a sequential segment number so that the receiving end may properly identify which segments have been received. When splitting up a message sequence, it is expected that each packet, or segment, within that sequence (except the last packet) will be of the same size. It is that size minus 1, which is placed in the size-1 field of the HELLO header. Segment sizes are restricted to 8, 16, 32, 64, 128, or 256 bytes. The final segment size may be any dword multiple. It's size-1 field should still reflect the general segment size used for the rest of the packets in the sequence.

The SRIO Gen2 Endpoint can be configured to support Message Class on a Common port, Initiator/Target port, or separate Message-Only port. All messages contain data and so they will encompass multiple beats on the HELLO interface. Message responses do not contain data and will only span a single cycle on the HELLO interface.

FTYPE13 - Response Class

Response packets on the HELLO interface use three transaction types. Response size and, in one case, fields are dependent upon the ttype. A response transaction must populate the Destination ID (in the AXI4-Stream user field) with the Source ID from its corresponding request transaction. Additionally, each response must use the same TID, or target_info, value as the corresponding request so that each response can be correlated with its causal request packet.

- **No Data (TTYPE = 4'b0000)**. Response transaction without data. Response used when no data is returned. Typically, these responses are expected as a handshake to NWRITE_R and Doorbelll packets. Some endpoints may choose to use this response type for error responses (“E” field = 1'b1) as well because error responses are required to not carry data. These responses are always a single beat on the HELLO interface. When separate Read and Write Initiator ports are selected for the SRIO Gen2 Endpoint, No Data responses will be presented on the Write Response port. For other configurations, No Data responses are presented on either the Receive (I/O) or Initiator Response (Legacy) port. No Data responses may be transmitted over any of the egress ports.
- **Message (TTYPE = 4'b0001)**. Message response transaction. Response used for all Message Class transactions regardless of error status. These responses are always a single beat on the HELLO interface as Message responses do not contain data. The SRIO Gen2 Endpoint can be configured to support Messages and their corresponding responses on a Common port, Initiator/Target port or separate Message port.
- **With Data (TTYPE = 4'b1000)**. Response transaction with data payload. Response used when data is expected or returned. Typically these responses are expected as a handshake to Atomic and NRead packets. Endpoints may choose to use this response type for NRead and Atomic error responses (“E” field = 1'b1) even though error responses are required to not carry data. With Data Responses will generally span between 2 and 33 cycles on the HELLO interface dependent upon Request size. The exception is single-cycle error responses which are not allowed to carry data. When

separate Read and Write Initiator ports are selected for the SRIO Gen2 Endpoint, With Data responses will be presented on the Read Response port. For other configurations, With Data responses are presented on either the Receive (I/O) or Initiator Response (Legacy) port. With Data Responses can be transmitted over any of the egress ports.

Logical and Transport Layer Core

The Logical and Transport Layer core (LOG) is responsible for building (TX) RapidIO packets based on header inputs and collapsing (RX) RapidIO packets into their header and data components. Additionally, the LOG routes packets to the appropriate ports based on packet function.

Packets leaving the LOG and entering the BUF in the transmit direction are transferred in a single data stream after being built by the core. When the packet appears on the Transport or Link Interfaces, it will use SRIO Stream format.

Request Flows and Packet Arbitration

The Source and Destination ID fields control packet routing through a RapidIO system and, together with the priority, define the transaction request flow. A switch will look at the packet Destination ID and route that particular packet to the port for which that Destination ID is mapped. This mapping is one of the initial steps necessary in the bring up of a RapidIO system.

Many endpoint implementations mask packets based on the Destination ID. The SRIO Gen2 Endpoint does not perform that check; however, hooks are provided to the user to allow for implementation of this feature if desired. The `deviceid` output port on the LOG provides the user with the current value stored in the Base DeviceID CSR (Offset 0x60). Width of the output port is dependent on the selected system size. By comparing `tuser[15:0]` on the output port with the Device ID value, the user can create a transaction gate for the unlikely occurrence of an errant packet.

A received packet's Source ID is necessary in order to provide a response back to the proper endpoint. Most applications simply capture a packet's incoming Source ID and store it for use as the Destination ID for the generated response. This is the procedure used by the example design. The LOG uses the `tuser[31:16]` signal from the transmit port as the Source ID for the outgoing packet. By default, the `deviceid` output is tied back into the `tuser[15:0]` inputs in order to ease implementation. When the AXI4-Lite Maintenance interface is used, the LOG swaps the received request's Source ID and Destination ID for the response that is issued to the remote device. This functionality prevents a write to the Device ID CSR from creating a response transaction with a different Source/Destination ID pair than that of the corresponding request.

The LOG offering allows statically defined small system (8-bit Device ID) or large system (16-bit Device ID) support. The side effects of this CORE Generator software option are shown in Table 3-7.

Table 3-7: **LOG System Size Side Effects**

	Small System	Large System
tt Field for Outgoing Packets	2'b00	2'b01
deviceid Output Port	16 bits wide and right justified value from the Base_deviceID field of the Base Device ID CSR	16 bits wide and derived from the Large_base_deviceID field of the Base Device ID CSR

Request packets which share a Source ID/Destination ID pair and are of the same priority make up a transaction request flow. RapidIO compliancy dictates that requests of the same transaction request flow do not pass responses of that same flow. For that reason, it is important that when a response is generated, the priority field is bumped by one. Additionally, transactions that solicit responses, such as NWRITE_R transactions, may not be given a priority of 2'b11 in order to accommodate this priority bump.

Although a request of a priority flow may not pass a response, there is no rule requiring responses to be issued in order. It is entirely possible that read requests will be returned out of order. For this reason, the srcTID (and corresponding targetTID for Response type transactions) field is provided to allow tracking of outstanding request/response pairs. It is up to the user design to track the transaction IDs (TIDs) for each request and correlate them with their responses. It is intended that each Source ID/Destination ID pair has only one outstanding transaction representing each TID value at any one time. As responses are received, the TIDs are freed and the user may choose to reissue that TID.

Additionally, the PHY provides a `port_timeout` output port which reflects the value in the Port Response Time-out Control CSR of the PHY. This value can be used in implementation of a timer to determine if a request/response pair has been outstanding for longer than the specified time frame. By using the TID values, the customer may implement a timer to tag each transaction and take action on a long outstanding transaction. The action taken is beyond the scope of both this document and of the *RapidIO Specification*.

All transmit ports must share the same transmit path through the BUF and PHY. The LOG arbitrates between the interfaces on a first come, first served basis. If packets are available on multiple ports simultaneously, the LOG will select the port that has been inactive for the longest time. This prevents starvation of any given port. Additionally, the BUF has a `respone_only` input which will restrict the initiator ports from being arbitrated. This signal incorporates the `master_enable` signal from the PHY, and also asserts when the buffer is filled beyond a certain watermark. In asserting this signal, the BUF forces a path for response transactions in order to prevent deadlock scenarios.

The Port General Control CSR houses the PHY's Master Enable register. By allowing configuration access to this register, the *RapidIO Specification* allows a single endpoint to act as system host and turn on other endpoints as masters once the system is fully configured. The default value of the Master Enable bit is configurable through the SRIO Gen2 Endpoint CORE Generator software GUI.

Local Configuration Space Base Address Accesses

As per the *RapidIO Specification* requirements, the LOG provides a Local Configuration Space Base Address (LCSBA) register in the LOG configuration space. See Local Configuration Space Base Address 1 CSR in Chapter 2 for the register description. For the LOG, usage of this register is limited to bits 30:21. These 10 bits act as a comparison for bits 33:24 of the incoming 34-bit transaction address. If these two 10-bit vectors match for an incoming non-maintenance transaction, it will be routed to the Maintenance Port if it exists (otherwise the packet will be handled as a normal I/O transaction).

By default the LCSBA value is set in the GUI to 10'h3FF. If left this way, all NREAD and NWRITE transactions with addresses in the range 34’h3_FF00_0000 - 34’h3_FFFF_FFFF will be mapped to the maintenance port. It is up to the user to find a 16M block of address space outside of the endpoint-used address range to be used as configuration space offset if LCSBA support is enabled in the GUI. Note that the RapidIO Specification does not provide for disabling the LCSBA functionality, and it should only be disabled in a closed system.

Since the AXI4-Lite Maintenance interface is restricted to 32-bit transactions, any NREAD of a different size (smaller or larger) will result in an errored response. Writes of sizes other than 32 bits will be dropped and the registers will not be updated (an errored response will also be issued for NWRITE_R transactions). Since SWRITEs cannot be smaller than eight bytes, SWRITE transactions are not supported for LCSBA accesses, and the address of incoming SWRITEs will not be checked (in other words, all SWRITEs, including those within the LCSBA space, will be forwarded to the normal I/O port).

Buffer Design

The Serial RapidIO Buffer Design (BUF) design provides a configurable buffer solution to handle clock domain crossing, retransmission, flow control and response priority bumping. The TX and RX buffer sizes are separately configurable to hold 8, 16, or 32 packets. Customization provides the system designer with options when balancing system bandwidth and core size.

Serial RapidIO is a lossless protocol. The BUF uses retransmission to guarantee packet delivery. As packets are transmitted from the BUF to the PHY, they get tagged with an acknowledgement identifier (ackID). The PHY appends the ackID onto the packet prior to transmission for tracking by the link partner. If the link partner has space for the received packet and finds it to be error free, then it issues a packet accepted (PA) control symbol. Otherwise the link partner indicates the packet should be retried or is in error. The exact link protocol is discussed more in PHY, page 110.

The PHY provides `phy_next_fm` to the BUF indicating which ackID should be associated with the transmitting packet. As the PHY receives PA control symbols, the `phy_last_ack` bus is updated indicating that packet has successfully been transmitted and can be expunged from the transmit buffer. In the case where a packet is not accepted for whatever reason by the link partner, the PHY discontinues the current transfer and rewinds the packet counter. In order to signal the rewind event to the BUF, the PHY will assert `phy_rewind` while it updates the `phy_next_fm` and `phy_last_ack` pointers. Assertion of `phy_rewind` occurs regardless of whether a frame is being transferred or not. However, when in packet, the `phy_rewind` signal will remain asserted until the BUF completes the packet on the interface. When out of packet, `phy_rewind` may remain asserted for as little as two cycles.

Upon release of `phy_rewind`, the update `phy_last_ack` value will be used to determine which packets can be removed from the buffer, and packet transmittal/retransmittal begins.

When a rewind event occurs, there is no guarantee that the BUF will issue packets in the same order as it did prior to the rewind event. When packets accumulate within the BUF design, it arbitrates packets based on priority and transaction type as follows:

- Responses in order of receipt
- Priority 2 & 3 requests in order of receipt
- Priority 1 requests in order of receipt
- Priority 0 requests in order of receipt

Under normal operating conditions, the SRIO Gen2 Endpoint transmits packets at the same rate they are received so arbitration among packets in the buffer does not come into play. However, when errors occur or buffers begin to fill and flow control comes into play, the BUF will begin to back-up and multiple packets must be arbitrated.

The arbitration mechanism and its impact on the use model must be taken into account when considering system design issues, such as how to prioritize packets. If a system generates a steady traffic flow using the SRIO Gen2 Endpoint as initiator, then it is suggested to use the same priority for all request traffic so that no one flow gets starved. However, if a system uses bursts of data within 1 or 2 high priority flows, then a tiered priority methodology may be beneficial since the lower priority flows will have lulls of inactivity in which to complete.

Flow Control

The *RapidIO Specification* allows for two different forms of flow control, transmitter (TX) and receiver (RX) controlled. The BUF supports both forms. In addition, the user may choose to generate an RX-controlled-only buffer which will save on resources, possibly at the expense of bandwidth.

When using receiver controlled flow control, the BUF relies on PHY retry protocol to throttle link traffic. The *RapidIO Specification* requires that a receive buffer reserve one space per priority jump and 1 space for responses. As the receive buffer fills past these points, it should begin retrying lower priority packets shown in Table 3-8.

Table 3-8: **Receive BUF Priority Allowance**

Available Buffers Remaining	Lowest Priority Packet to Be Accepted
4 or more	All priorities
3	priorities 1, 2 and 3
2	priorities 2 and 3
1	priority 3 responses only
0	no space, nothing accepted

When a packet of too low of a priority gets retried, the BUF rewinds the packets it sent by re-queuing everything for arbitration. If the packet which was retried is a response packet, that same packet will be reissued; however, the packet priority will be incremented by one. This is allowed for endpoints by RapidIO protocol to prevent deadlock scenarios where response transactions back up and eventually block the receiver queue. The remaining packets will be arbitrated as shown in Table 3-8.

Figures 3-19 through 3-23 show examples of how retransmission effects packet queuing. Figure 3-19 shows a basic priority bump.

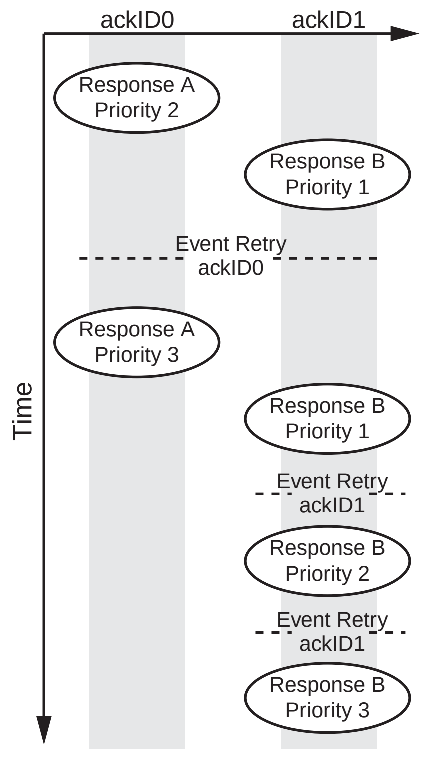


Figure 3-19: **Response Priority Bump**

The initial Response transaction, marked A. is sent as ackID 0 with an initial priority of 2. Another Response follows, Response B, as ackID 1, with an initial priority of 1. The link partner indicates it does not have space available for the priority 2 packet by issuing a retry for ackID 0. At this point, all packets issued after and including ackID 0 must be retransmitted. Additionally, the retried response packet, Response A, receives a priority bump in an attempt to push it through the receiving queue. After the retry event, Response A is reissued as a priority 3 packet using the rewound ackID of 0. Since all subsequent packets must be retransmitted, Response B is sent again using ackID 1. Response B does not receive a priority bump since it was not the retried response. After the responses are sent the second time, Response A is accepted and Response B is retried (RapidIO protocol prohibits out of order packet acknowledgements, so a retry of ackID 1 here must assume that ackID 0 has been accepted). The priority of Response B bumps to a 2. When the priority 2 response cannot be accepted by the link partner, the response is again retried and consequently bumped to a priority 3.

Figure 3-20 shows request packet reordering.

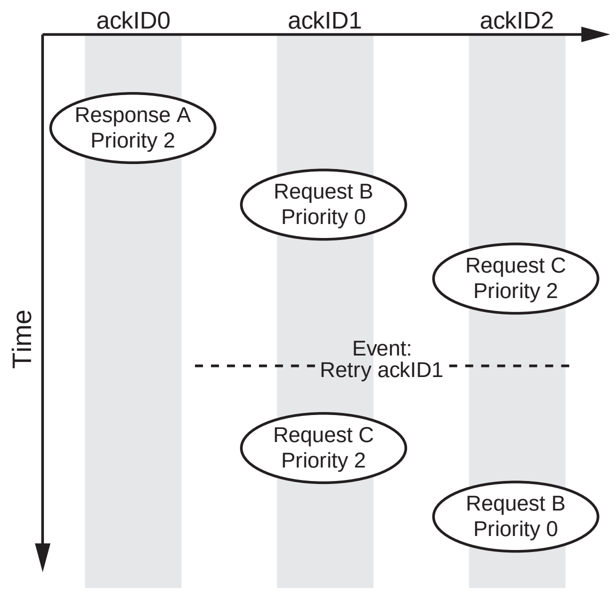


Figure 3-20: **Request Reordering**

Response A is sent (and accepted) with a priority of 2. As the next two Requests B and C enter the buffer, they are transmitted in the order they are received. However, when the link partner does not have space available for the Request B priority 0 packet, the packet is retried as Request C is being transmitted. When the core rewinds, it also reorders based on priority. This time the priority 2 Request C is transmitted first and associated with ackID 1. The priority 0 Request B is sent last and associated with ackID 2.

Figure 3-21 combines a priority bump with more complex reprioritization.

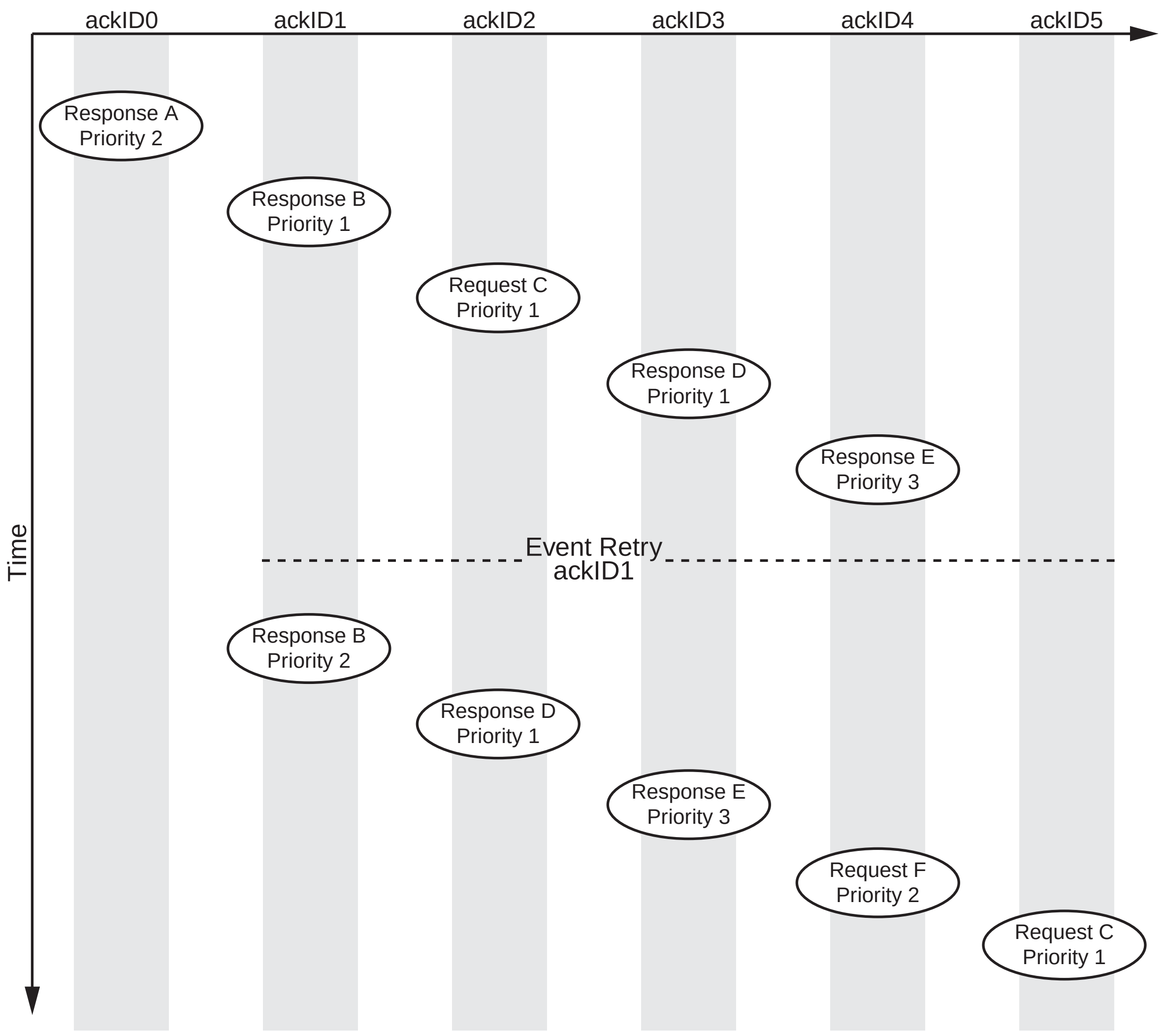


Figure 3-21: **Reprioritization and Reordering**

In this example, the packets are originally sent from the BUF in the order they are received since the buffer is clear and no queuing back-up exists. However, the retry event forces a back-up condition, and the BUF must reprioritize all packets in the queue. The retry is for ackID 1. In this example, the ackID of 1 implies that a packet control symbol for ackID 0 has already been accepted. Since ackID 1 was a Response type transaction, the priority was bumped due to the retry, and the initial response transaction was resent. At this point, the BUF begins sending transactions based on the priority rules described in Table 3-8. Responses issue in the order they were received. Response D jumps Request C, even though they are of the same priority because of D's response transaction type. Response E jumps Request C due to its transaction type, but it does not jump the lower priority Response D. This is because Response priorities do not result in reordering; they are issued in the order they were received. At this point Request F, which was put into the buffer after the Retry Event occurred and therefore was not part of the initial transaction sequence, gets reprioritized ahead of Request C due to its higher priority value. Request C issues last due to its low priority value and Request transaction type.

If a link partner has a small buffer, then that buffer will fill and empty faster. This means that the retry scenarios which govern RX flow control happen more frequently and those small buffers may run dry during retry recovery, particularly if packets sizes are small. Additionally, some major RapidIO vendors do not support transmitter controlled flow control. When interfacing to one of these vendors, generate a receiver controlled only core to optimize resource usage. It is also suggested to increase the buffer sizes to reduce the risk of dry and overrun buffers.

Transmitter controlled flow control is meant to keep link bandwidth up by limiting rewind scenarios. The BUF provides the user with a programmable watermark register (CSR 0x10004). This register carries each of the three watermark values used to determine when to block lower priority packets as the link partner's buffer space fills. It is recommended to write this register in its entirety in order to maintain valid watermark configurations (0 < WM2 < WM1 < WM0). The BUF bases packet transmittal decisions on a combination of status updates from the link partner (`phy_rcvd_buf_stat`), outstanding packet count (`phy_next_fm` - `phy_last_ack` - 1) and the internal watermark values. The BUF still sends packets in the order detailed in Table 3-8, but it will not send lower priority packets if the available buffer space falls below the watermark value for the packet. Additionally, response priorities will be automatically adjusted based on the calculated buffer space available (`phy_rcvd_buf_stat` - (`phy_next_fm` - `phy_last_ack` - 1)).

Figure 3-22 illustrates the valid priority packets to be sent when the buffer space available is within each watermark range.

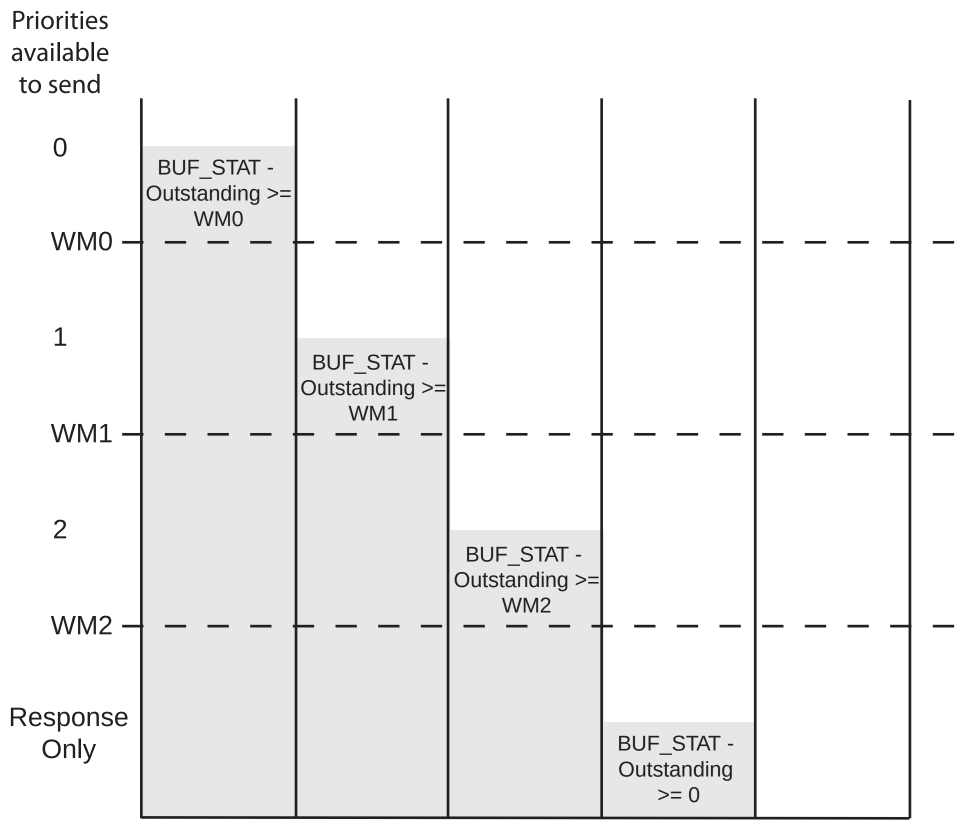
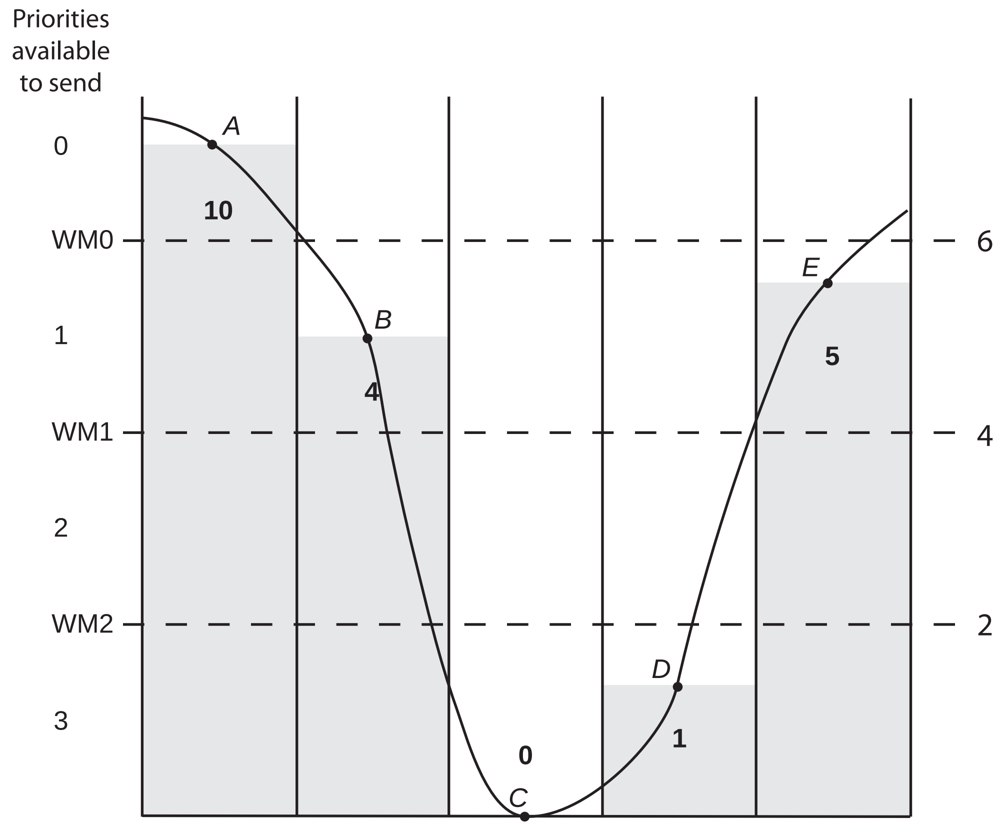

Priorities available to send
0
WM0
1
WM1
2
WM2
Response Only
BUF_STAT - Outstanding >= WM0
BUF_STAT - Outstanding >= WM1
BUF_STAT - Outstanding >= WM2
BUF_STAT - Outstanding >= 0

Figure 3-22: **Transmit Packet Priority Based on Watermarks**

For instance, if the space available is greater than or equal to the WM0 value, all packets may be sent and response priorities will *not* be bumped. On the other hand, if the buffer space available is less than WM2, but greater than 0, only response transactions will be sent and all response priorities will be bumped to 3. As an example, see Figure 3-23.



LNK_RX_BUF_STAT[0:4]	10	6	3	3	6
LNK_TNEXT_FM	0	8	15	15	0
LNK_TLAST_ACK	31	5	11	12	30

Figure 3-23: **TX Flow Control Example**

In this example, the user has set the watermarks to be:

- WM0 = 6
- WM1 = 4
- WM2 = 2

The curving line represents a constantly changing available buffer space value. The valid packets will be evaluated at several points along the slope.

- Point A is the starting point. The link partner has 10 available buffer slots for use. No packets have been sent. Available buffer space is 10. WM0 is 6. The BUF may send any packet it has available.
- Point B marks a point early in the transfer cycle. Eight packets have been sent across the link. Six have been accepted. The link partner indicates its buffers are beginning to

fill. At this point the BUF sees four available slots. Since that is equal to WM1, priority 0 packets will no longer be transmitted. All priority 0 requests will queue in the buffer.

- At point C, the link partner indicates it has only a few spaces left. The endpoint has not received acknowledgment for three of its outstanding packets. The BUF must stop transmission of packets until it receives indication that buffer spaces have become available.
- At point D, the BUF has received an additional acknowledgement lowering the outstanding packet count by one without a change in the buf_status. Since 1 space is available, the BUF is able to send a response. It will bump the response priority to 3 to ensure acceptance.
- At point E, the link partner buffer has become more available, reporting a buffer status of six. Since the BUF has only one outstanding packet, it is able to send any transaction except a priority 0 request.

In Figure 3-23, once the available buffer count drops below watermark 0, priority 0 request packets will begin to accumulate in the buffer. As the available buffer space continues to drop past point B, priority 1 requests will start to queue in the buffer. The buffer accumulates these packets and it is likely that eventually, it will assert the `response_only` output. At this point, only response type transactions will be accepted into the buffer until some of the lower priority packets get cleared out. Unless the system has constantly streaming responses, this will create windows of lower priority traffic. Smaller transmit buffers will force those windows on a more frequent basis; however, they are also more likely to incur bandwidth penalties since RX buffers could run dry in the buffer status turn-around time. This is particularly true when smaller packets are transferred.

The receive side of the buffer queues packets as first-in-first-out. As each packet comes in, the BUF checks the priority to see if space is available. Receive side availability is governed by hard watermarks ensuring that there is one space available for each allowed priority level, as shown in Table 3-8, page 104.

The configurable read buffer size gives system designers the ability to make trade-offs between bandwidth and resources. The smaller buffer provides a LUT savings at the expense of potentially reduced bandwidth. Small and medium size packets will quickly fill the 8 deep buffer and incur retries and TX flow control throttling. Asynchronous clocking implementations will exacerbate the situation due to latencies inherent to domain crossing logic.

PHY

This section describes the functionality within the Serial RapidIO PHY, including communication with the core through the Link Interface. This section also describes initializing the RapidIO link, managing control symbols, receiving and transmitting data packets, issuing and responding to discontinues, managing the acknowledgement identifiers, detecting special case error conditions, setting up time-out counters, and receiving and transmitting multicast event control symbols.

Initializing the RapidIO Link

The PHY must be initialized and aligned so that data packets and control symbols can be reliably received. The PHY goes through initialization and alignment after reset and recovers the incoming data and clock from the input serial stream.

Initialization and alignment occurs when coming out of reset or after loss of sync on the input port during system operation. Guidelines for initialization and alignment can be found in the *RapidIO Physical Layer 1x/4x LP-Serial Interconnect Specification*.

Monitor the following signals to monitor whether the link is initialized and aligned:

- **`port_initialized`**: Link has locked to receive stream.
- **`mode_1x`**: For a 2x or 4x core, signal indicates that the core has trained down to one lane. This is valid after `port_initialized` asserts.
- **`rx_lane_r`**: Indicates that the core has trained to the redundancy lane.
- **`idle2_selected:`**Indicates that the core is operating in IDLE2 mode.
- **`out_of_sync`**: Indicates that the scrambler is not synchronized (only valid when scrambling is enabled).
- **`link_initialized`**: Indicates seven consecutive error free control symbols have been received and 15 consecutive symbols have been sent. The core is fully trained and can now transmit data.

These signals can be used to determine when the PHY TX and RX serial links are operational. The PHY interface can then start receiving and transmitting packets.

Management of Control Symbols

Incoming control symbols are consumed by the PHY and are not passed on to the user application. The PHY processes control symbols, and if necessary, responds with an appropriate control symbol response. To illustrate, if the PHY receives an incoming link request control symbol, it will respond by automatically generating link response control symbols on the transmit port.

Table 3-9 contains the SRIO control symbols and the actions taken by the PHY. The transmit action describes the conditions when the PHY would transmit the control symbol. The receive action describes the action taken by the core when receiving the particular control symbol. For more information on SRIO defined control symbols, see the *RapidIO Physical Layer 1x/4x LP-Serial Interconnect Specification*.

Table 3-9: **SRIO Physical Layer Control Symbols**

Control Symbol	Transmit Action	Receive Action
Packet Accepted (PA)	Transmits PA control symbol when the BUF accepts a packet on the link receive interface.	Updates phy_last_ack output so that the BUF can release the acknowledged packet.
Packet Not Accepted (PNA)	Transmits PNA control symbol when the PHY detects an error condition on the incoming packet buffer. Discards all incoming data until a LR_IS control symbol is received.	Stops transmission of outbound data and transmits a LR_IS control symbol to the connected device. Asserts phy_rcvd_pna to the user for one clock cycle. Asserts phy_rewind output until the Link Response handshake is complete at which point phy_next_fm is updated with the next expected packet and phy_last_ack is updated to phy_next_fm-1.
Packet Retry (PR)	Transmits PR control symbol when the BUF is not accepting packets on the link receive interface and the internal buffer storage has been exhausted.	Updates phy_next_fm output to the BUF with ackID of retried packet and asserts phy_rewind, indicating a retransmission is necessary and transmits a RFR control symbol.
Stomp (STMP)	Inserts STMP control symbol when BUF terminates a packet by asserting a phyr_tlast and phyr_tuser[0].	Discontinues packet to the BUF by asserting phyr_tlast and phyr_tuser[0] to indicate a bad packet. Transmits PR to the sending device.
Status (STAT)	Receives STAT control symbols once every 1024 code groups.	Updates the phy_rcvdbuf_stat to the BUF with the buf_status value in the STAT control symbol.
Link Request - Input Status (LR_IS)	Transmits LR_IS when a PNA is received or other detected protocol violation. Monitors RX port for incoming LRESP control symbol.	Receives LRESP control symbol with the next expected ackID value and looks for next incoming packet ackID value to equal the expected ackID value.
Link Response (LRESP)	Transmits LRESP control symbols when receiving a LR_IS control symbol. Transmits the next expected ackID value by the BUF on the link receive interface.	Updates the `phy_last_ack` and/or `phy_next_fm` bus with the incoming ackID value in the LRESP. This signals the BUF to release and retry packets.
Link Request - Reset Device (LR_RD)	Transmits LR_RD control symbols when the phy_link_reset signal is asserted. It needs to be asserted until the port_initialized signal goes Low to successfully reset the opposite end of the link.	Asserts phy_rcvd_link_reset to indicate that the core has received four LR_RD control symbols. The PHY is also reset.

Table 3-9: **SRIO Physical Layer Control Symbols *(Cont'd)***

Control Symbol	Transmit Action	Receive Action
Start of Packet (SOP)	Inserts SOP on all outbound packets. When possible, core will transmit SOP control symbol to indicate a end of one packet and a start of another packet.	Strips incoming stream of SOP control symbol and uses SOP control symbol to mark the beginning of a packet.
End of Packet (EOP)	Inserts EOP at the end of an outbound packet if necessary.	Strips incoming stream of EOP control symbol and uses EOP control symbol to mark the end of a packet.
Restart from Retry (RFR)	Transmits RFR control symbol when PHY receives a PR control symbol. The `phy_next_fm` bus indicates that the BUF must retry a packet.	Strips incoming stream of RFR control symbol and looks for next packet ackID value to equal the expected ackID value.
Multi-cast Event (MCE)	If user asserts phy_mce, transmits MCE control symbol.	Asserts phy_rcvd_mce when the core receives an MCE control symbol.

Link Reset

The user should monitor the `phy_rcvd_link_reset` signal, which indicates that the PHY has received four Link Request Reset Device control symbols and is resetting the core. All states must return to their default value. Depending on the application implementation of `sys_rst`, `sys_rst` can be asserted in response to the assertion of `phy_rcvd_link_reset` to reset the entire application.

Driving the Receive BUF Status

The receive buffer status, **`phy_buf_stat[5:0]`**, is a six-bit signal that is buffer-driven to the PHY. The BUF typically stores the incoming packets in a Block RAM. This signal indicates the number of maximum-sized packets that the BUF can accommodate. The encoding value specifies the number of packet buffers currently available. This could represent anywhere from 0 (6'b00_0000) to 30 (6'b01_1110) available buffers. This information is encoded in the `buf_status` field of the outgoing acknowledgement, status and packet retry control symbols issued by the PHY.

Based on the previous information, the connected device decides whether it can transmit more packets to the PHY. If the BUF has no available buffers for packets (**`phy_buf_stat[5:0]`**= 6'b00_0000), it discontinues incoming packets. Doing so causes the PHY to send out a packet-retry control symbol to the sender and to discard all incoming packets.

If not using a standard RapidIO buffer, a minimum of one buffer must be implemented to store a full size packet. In this case, dynamically drive just the least significant bit of the **`phy_buf_stat[5:0]`** signal. All other unused bits of the `phy_buf_stat[5:0]` signal will be tied to ground. A value of 0 indicates no buffers available, and a 1 would mean that the device has one empty buffer.

If receiver based flow control is needed, tie **phy_buf_stat[5:0]**= 6’b11_1111.

Stripping the Final CRC in Received User-Defined Packets

RapidIO packets greater than 80 bytes contain both an intermediate and final CRC. RapidIO packets 80 bytes or less contain only the final CRC. When present, the intermediate CRC is stripped from the packet. The final CRC is also stripped in most cases. However, the RapidIO Specification allows for 16 bits of zero padding in addition to the final CRC. In the case of User-Defined packet types, the PHY will strip the last 16 bits of the packet but if the zero padding was used, the CRC will still be part of the packet passed to the BUF.

Interpreting the Transmit BUF Status

The transmit buffer status, phy_rcvd_buf_stat**[5:0]**, is a six-bit signal driven by the PHY to the BUF. It indicates the number of maximum-sized packets that the connected remote device can accommodate. The encoding value specifies the number of packet buffers the remote device has currently available, and could represent anywhere from 0 (6’b00_0000) to 30 (6’b1_1110) available buffers in IDLE1 operation, or from 0 (6'b00_0000) to 62 (6'b11_1110) in IDLE2 operation. This information is encoded in the buf_status field of incoming acknowledgement, status and packet retry control symbols issued by the remote PHY.

Based on the preceding information, the buffer application design may implement transmitter based flow control logic and decide whether it can transmit more packets to the connected devices.

If the phy_rcvd_buf_stat**[5:0]** = 6’b11_1111, the connected devices will want to use receiver based flow control.

Acknowledgement Identification

The following signals are provided by the PHY to the BUF in relation to the Acknowledgement Identifier, ackID.

- phy_last_ack[5:0]
- phy_next_fm[5:0]

The details of the preceding signals are provided in Managing the Transmit Acknowledgement Identification, page 116.

Discontinues in the Physical Layer

Discontinues are used if either the BUF or the PHY wants to discontinue data transfer due to unavailability of resources, FIFO overflows, internal conditions, or CRC errors.

This section describes PHY and BUF signals for indicating discontinues in the receive and transmit paths. The operation of the discontinues, their possible causes and effects are also provided.

Link Receive Source Discontinue

Operation

The PHY sets the discontinue bit in `m_axis_phyr_tuser` (bit 0) along with `m_axis_phyr_tlast` to indicate that it wants to discontinue data transfer to the BUF. Upon taking this action, the PHY stops the transfer of the packet.

- **Cause**: This discontinue signal is asserted by the PHY if a CRC error is detected, the packet is stomped, or other error conditions during the packet reception cause this signal to be asserted.
- **Effect**: When this signal is asserted, data transfer between the BUF and the PHY stops. The core stops the transfer of the packet and if necessary, transmits a packet-not-accepted control symbol or a packet retry to the sender of the packet. If the BUF receives any portion of the packet when the discontinue is asserted, the BUF is responsible for discarding the packet.

Link Receive Destination Discontinue

Operation

The BUF cannot discontinue a packet in the receive direction.

Link Transmit Source Discontinue (Rewind)

Operation

The BUF sets the discontinue bit in `m_axis_phyt_tuser` (bit 0) along with `m_axis_phyt_tlast` to indicate a desire to discontinue data transfer to the PHY.

- **Cause**: This discontinue signal is asserted when a rewind scenario occurs, as indicated by `phy_rewind`.
- **Effect**: When this signal is asserted, data transfer between the BUF and the PHY stops. The PHY will not increment the `phy_next_fm`, resulting in retransmission of the packet by the user. If a part of the packet has already been sent by the PHY to the RapidIO fabric when the discontinue was asserted, the PHY cancels the packet by issuing a stomp control symbol.

Link Transmit Destination Discontinue

Operation

The PHY asserts the `phy_rewind` signal to indicate that it wants to discontinue data transfer from the BUF. Upon assertion of this signal, the PHY will stop receiving the packet. The `phy_rewind` signal should remain asserted until the buffer signals the end of the packet by asserting `m_axis_phyt_tlast.`

- **Cause:** The PHY asserts this discontinue if it has entered the output port error recovery process because it has received a packet-not-accepted control symbol or has entered the output retry-stopped recovery process because it has received a packet retry control symbol.
- **Effect:** When this signal is asserted, data transfer between the BUF and the PHY stops. The core will change the `lnk_tnext_fm` to the packet that needs to be sent, resulting in retransmission of the packet by the BUF application.

Managing the Transmit Acknowledgement Identification

The acknowledge ID (ackID) is an identifier used for acknowledgement of packets transmitted across a link. In IDLE2 mode (when long control symbols are in use), the ackID has six significant bits and allows for a range of up to 63 outstanding unacknowledged packets. In IDLE1 mode (when short control symbols are in use), bit 0 is tied to zero, and allow for up to 31 outstanding unacknowledged packets.The ackIDs are assigned sequentially, in an increasing order, wrapping back to zero on overflow, to indicate the order of packet transmission. They are sent out to the RapidIO fabric as a Physical Layer header field in the transmit packet.

The following signals facilitate management of the ackIDs on the Link Transmit interface:

- `phy_last_ack[5:0]`: This signal indicates the last valid (expected) packet acknowledgement that was received by the PHY. The BUF may then release the frame whose acknowledgment was received for reuse. However, if there was an error with acknowledgement control symbol with the ackID, this signal remains unchanged.
- `phy_next_fm[5:0]`: This signal indicates the ackID of the next packet to be transmitted if the transmit link is idle. It may also denote the ackID of the packet currently being transmitted. However, if the interface gets a retry due to an error in the packet sent out, this signal remains unchanged. The BUF must also correlate this value to the packet that is currently being transmitted.

Normal Operation

During normal operation, the BUF must associate all transmit packets with the value being presented on **`phy_next_fm`**. This is the ackID value that will be transmitted in the physical layer header fields of that packet. Once the connected device acknowledges the packet, it will transmit a packet accepted control symbol with the ackID value equal to the ackID value found in the accepted packet. The PHY updates **`phy_last_ack`** with the ackID value found in the packet accepted control symbol. Based on this update, the BUF then releases the buffer location that is holding this packet.

Error Recovery Operation

This section describes scenarios that you must be aware of during error recovery. These cases illustrate the conditions causing the BUF to retransmit a packet or acknowledge one or several packets simultaneously.

Scenario 1: Packet Retry Due to Internal Condition

The connected device retries a packet due to some temporary internal condition.

Effect: Once a packet retry control symbol is received, the PHY discontinues the BUF on the link transmit interface. The core transmits a restart-from-retry control symbol. The core also updates the **`phy_next_fm`** with the ackID value received in the packet retry control symbol. This indicates that the BUF must start retransmitting packets from that ackID value, thereby reestablishing the proper ordering between the devices. If the retry control symbol is received with an ackID value equal to the current **`phy_next_fm`** (in other words, **`phy_next_fm`** does not increment) the BUF must retry the packet that was discontinued.

Scenario 2: Handling Out-of-Sequence ackIDs

The PHY transmits packets with ackIDs in a sequence, and it receives acknowledgements for all but an earlier ackID. This indicates a possible error with the packet of the ackID not received.

Effect: The PHY discontinues the BUF on the link transmit interface. It then transmits a link request input status control symbol. Once the connected device receives the link request input status control symbol, it will transmit a link response control symbol, indicating which packet ackID it expects to receive next. These actions may need to be taken based on the link response ackID value:

- The Link Response ackID field indicates that the connected device has acknowledged one or more ackIDs greater than what was currently being driven on **`phy_last_ack`**. The PHY will update **`phy_last_ack`** with the value in the link response control symbol. The BUF can release all buffer locations starting from the previous value of **`phy_last_ack`** to the updated value of **`phy_last_ack`**.
- The Link Response ackID field indicates that the connected device is expecting a packet with an ackID that was already transmitted. The PHY will update the **`phy_next_fm`** with the this value. The BUF must retransmit this packet to re-establish the proper ordering among devices.

Scenario 3: Device Receives Corrupt Control Symbol

The connected device receives a corrupt control symbol from the PHY.

Effect: The connected device will send a packet-not-accepted control symbol to the sender. The PHY will respond in a similar manner as described in Scenario 2: Handling Out-of-Sequence ackIDs.

Detecting Special Case Error Conditions

The PHY is designed to handle most error conditions as outlined by the *RapidIO 1x/4x LP-Serial Specification*. However, the following signals must be monitored to avoid system lockup:

- `phy_link_reset`: Send Link Reset
- `phy_rcvd_link_reset`: Link Issued Link Reset Control Symbol
- `rcvd_pna`: Link Packet Not Accepted Received
- `port_error`: Link Port Error
- `force_reinit`: Force Reinitialization

Scenario 1: Link Request Reset Device

The link received a Link Request Reset Device. The connected device may issue a link request reset device control symbol sequence. Upon receiving four consecutive link request reset device control symbols, the PHY resets itself and asserts `phy_rcvd_link_reset` to the user.

The behavior of the user application on the assertion of `phy_rcvd_link_reset` is implementation-specific and beyond the scope of this document. However, Xilinx recommends that the user application dissociate all ackIDs with the stored packets. After reset, the PHY will drive **`phy_next_fm`** to 6'b00_0000 and **`phy_last_ack`** to 6'b01_1111 in IDLE1 mode and 6'b11_1111 in IDLE2 mode.

Scenario 2: Fatal Error Conditions

There are some cases where the core will detect a fatal error as defined by the *RapidIO 1x/ 4x LP-Serial Specification*. This can be, but not necessarily limited to:

- **Link Response Timed Out**: The PHY transmitted a link request control symbol but no link response control symbol was received.
- **Link Response Misaligned AckID Value**: A link response ackID field contains a value that is not outstanding.

When these conditions are detected, the core will assert **`port_error`**. This signal indicates that the PHY has shut down transmission and reception of packets. The user must assert `sys_rst` to the core in order to re-initialize the link and reset the error recovery state machines.

Scenario 3: Repeated Reception of Packet-Not-Accepted without Progress

A core can have repeated reception of packet-not-accepted without forward progress. When an error is detected during link operation, the connected device issues a packet-not-accepted control symbol. The PHY issues a link request control symbol. The connected device responds with a link response control symbol containing the next expected ackID. The PHY retransmits that packet, and link operation is resumed. However, due to internal error conditions or unreliable link operation, its possible that the connected device continues to transmit a packet not accepted control symbol forcing the link to never make forward progress on packet transmission.

The PHY is designed to facilitate detection of this scenario and to allow you to react based on system and design requirements. Every time a packet not accepted control symbol is received, the `rcvd_pna` signal will assert for one clock cycle. This signal, combined with the **`phy_next_fm`** and **`phy_last_ack`**, allows detection of multiple packet-not-accepted control symbols being received without forward progress on packet transmission. This could indicate a serious physical link problem or another system problem. The behavior of the user application on detection of this scenario is beyond the scope of this document and is implementation-specific.

Possible options to implement include:

- Force re-initialization of link by asserting `force_reinit`.
- Reset the PHY by asserting `sys_rst`.
- Reset the connected device by asserting `phy_link_reset`.

Note: Asserting reset to the PHY or to the connected device will cause the two devices to either be misaligned on ackID tracking (a fatal error), or to retransmit previously transmitted data, which can result in greater system errors. It is best that if one device is reset, the other be reset too. For example, if a reset is issued to the connected device (using `phy_link_reset`), the link will be re-initialized, and the **`link_initialized`** signal will deassert until the link is reestablished. Xilinx recommends that the user application assert `sys_rst` to the PHY at that time, thereby forcing all ackID tracking to return to reset values. This prevents any further fatal or system error conditions.

Time-out Counters for Lost Packet Detection

Time-out counters expire when sufficient time has elapsed without receiving the expected response from the system. These are used to detect errors such as a lost response or request packets. The time-out counter value is defined in the Physical Layer Maintenance Command and Status Register (CSR) space as given in the following:

- **Port Link Time-out Control CSR (Block Offset 0X 20 Word 0)**: The port link time-out control register contains the time-out value for link events such as sending a packet to receiving the corresponding acknowledge, and sending a link-request to receiving the corresponding link-response. The reset value (the maximum time-out interval) is user-specified in the CORE Generator software. The max value represents approximately 3 seconds. The value scales linearly from 0 to max. The PHY uses this time-out value to control internal counters that detect error conditions and initiate error recovery or indicate fatal error on `port_error`.

- **Port Response Time-out Control CSR (Block Offset 0X 20 Word 1)**: The port response time-out control register contains the time-out value for link events such as sending a packet to receiving the corresponding response packet. The reset value (the maximum time-out interval) is user-specified in the CORE Generator. A value of all ones should correspond to a timeout of between 3 and 6 seconds. The user application is responsible for detection of lost response packets. This time-out value in the CSR is presented to the user through the output port `port_timeout`.

Tip: Setting the default value for the Port Response Time-out appropriately during core generation will allow the Port Link Time-out counter to attempt recovery prior to a Port Response time-out.

Transmitting and Receiving Multicast Event Control Symbols

This section describes how to transmit and receive multicast event control symbols.

Multicast Event Request

Operation

The `phy_mce` signal is used to request transmission of a multicast event control symbol by the user application.

- **Cause:** Allows the occurrence of a user-defined system event to be multicast throughout a system.
- **Effect**: When this signal is asserted, it results in the transmission of a multicast event control symbol by the PHY. This signal should be asserted for one clock cycle.

Multicast Event Decode

Operation

This multicast event decode signal (`phy_rcvd_mce`) is asserted by the PHY when it has decoded a multicast event control symbol received on the serial link.

- **Cause**: The PHY has decoded a multicast event control symbol it received.
- **Effect**: The PHY informs the user application that it has decoded the multicast event control symbol. As multicast events are system specific, the user must determine how to interpret the reception of a multicast event.

Hardware Bringup

The SRIO example design is intended to be easily placed on hardware. Running the implementation script, as-is, produces a usable bit file. However, before programming the device, or even running the script, some changes need to be made to some files.

Part Selection

If the wrong device is targeted during implementation, the bit file will not download to the device properly. For the Vivado Design Suite, the device is selected when creating the project. The part selected at the beginning of the project creation propagates all the way through to the bit file generation step.

For the ISE Design Suite, selecting the device in the ISE command-line flow takes one extra step. In the `implement.sh/bat` file, modify the ngdbuild command line to reflect the proper device:

```
ngdbuild -p xc7v585t-ffg1761-2 -uc srio_example_top.ucf srio_example_top.ngc
```

Pin Placement

See the applicable board schematic and supporting documents for proper pin placement. All pins, though there are few in the example design, need to be evaluated for placement.

Edit either the UCF or XCD and select the proper pins according to the board. Pay special attention to the clock pin, which must be placed on a MGTREFCLK site and must follow placement rules in relation to the transceivers.

The high-speed IO must also be placed with care. Select transceivers all within the same bank. For the higher line rates (5 Gbps, 6.25 Gbps), select transceivers in centralized banks on the right side of the chip for improved timing.

Do not modify the IOSTANDARD constraints or the timing constraints. If there are area groups, they should be removed or positioned to a natural location in relation to the new pin selections.

Clock Generation and Other Inputs

`Sys_clkp/n` is expected to be either a 125 MHz or 156.25 MHz free-running differential pair clock. The actual frequency depends on what was selected during core generation. See the appropriate transceiver user guide for jitter tolerances and other requirements.

The reset input is an active-high reset and can be connected to any general IO pin. Early on, connect the reset to a debounced push-button.

External Loopback

The SRIO design is immune to physical configurations and connections. One of the easiest ways to bring up a single board is to loop the high-speed IO back onto itself. It is useful for debugging as well as for users who only have one board available to them.

High-Speed IO Connections

Regardless of the physical configuration, the connections shown in Table 3-10 must be made.

Table 3-10: **High-Speed IO Connections**

Receive Side Pin	Transmit Side Pin
srio_rxn0	srio_txn0
srio_rxp0	srio_txp0
srio_rxn1	srio_txn1
srio_rxp1	srio_txp1
srio_rxn2	srio_txn2
srio_rxp2	srio_txp2
srio_rxn3	srio_txn3
srio_rxp3	srio_txp3

Programming

Program the device the same as any other Xilinx FPGA. Use the Analyzer tool to also use the provided ChipScope™ Analyzer VIO and Trigger mechanism.

Chipscope Analyzer

View core statistics, packet transfers, and kick off packet transfers with the ChipScope tool. See Hardware Features for more details.

Initial Signs of Success

After programming, the first indication of success is the bank of LEDs. These have been hand-selected as quick indicators of hardware viability. Table 3-11 details the LED lights and indications.

Table 3-11: **LED Indicators**

LED0[x]	Description	Indicates
0	Always 1'b1, Confirms orientation	If ever 0, orientation or pin placement is wrong.
1	Always 1'b1, Confirms orientation	If ever 0, orientation or pin placement is wrong.
2	Mode_Nx, 1'b1 when configured for only 1 lane	If ever 0, a multilane core has trained down to 1x mode. This may be intentional. If not, check high-speed IO connections. If connections are OK, check for indications of disparity errors on the link. Does not have to be 1 in order to achieve Port/Link Initialization.
3	Port is initialized	First sign of communication between endpoints. Must be 1 in order to gain Link Initialization
4	Link is initialized	When 1, the link is up and running correctly.
5	Clock lock	If ever 0, there is a problem with the clock source or connection.
6	Always 1'b0, Confirms orientation	If ever 1, orientation or pin placement is wrong, or using simulation features. "Autocheck error" in simulation.
7	Always 1'b1, Confirms orientation	If ever 1, orientation or pin placement is wrong, or using simulation features. "Exercise done" in simulation.

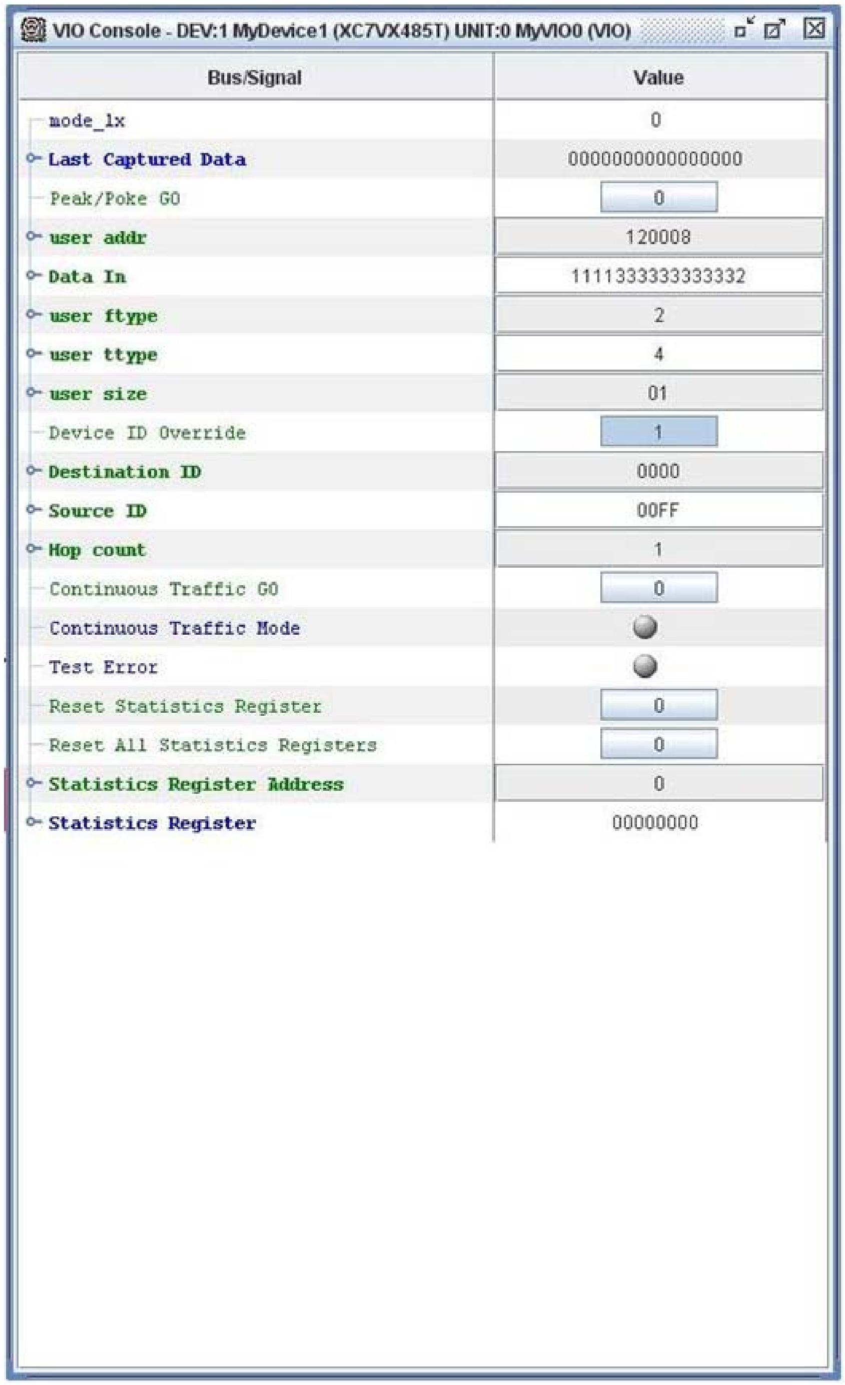
Hardware Features

The SRIO example project works both in simulation and hardware though some features play more prominently in one platform or the other. The following section describes some useful features when debugging or demoing hardware.

Chipscope Analyzer

A complete Chipscope project has been provided with the core. It is useful for quickly creating packets of all types, as well as observing behaviors on the user interface. The example design project comes with calls for the ICON, IL, and VIO features. To enable these features in hardware, be sure to set the USE_CHIPSCOPE parameter to 1 in of `srio_example_top.v`. It is not advised to use the ChipScope tool as part of a simulation and the tool should only be used in hardware. After programming, load the ChipScope project file provided under `example_design/chipscope`.

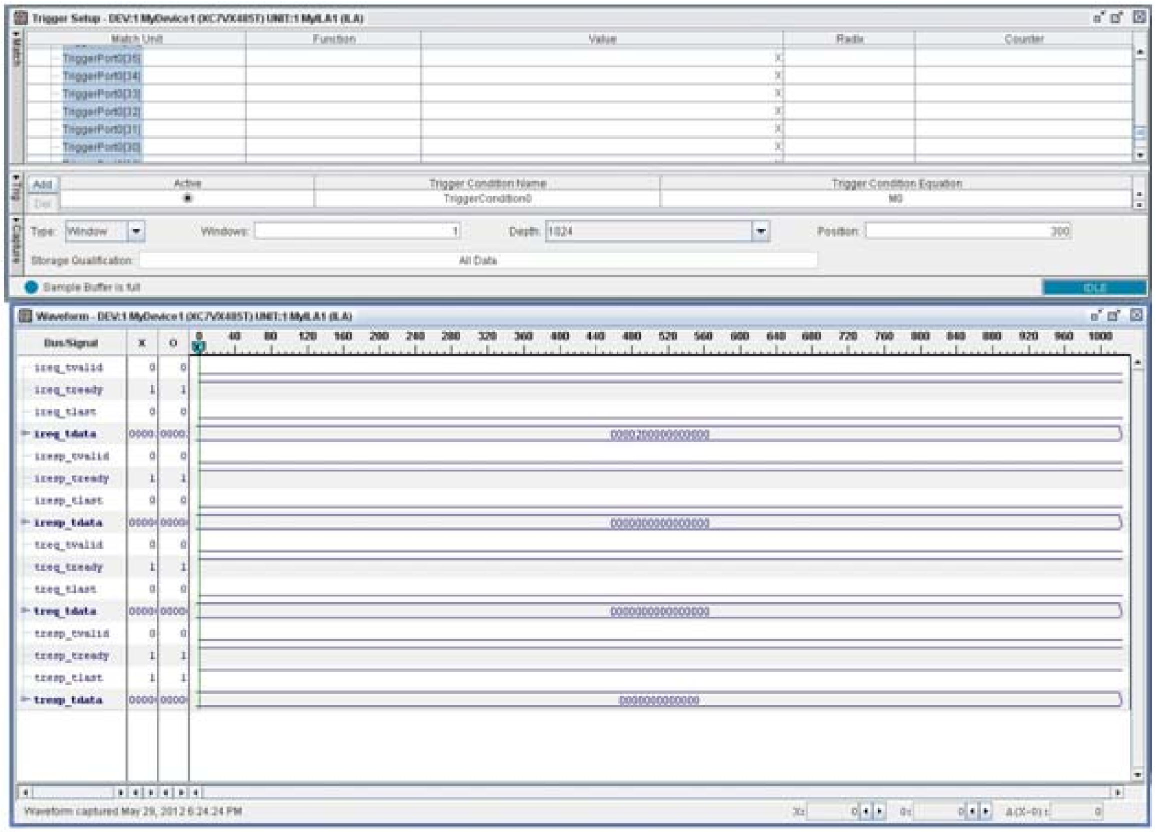
The VIO dashboard can send single packet of a specified FTYPE, TTYPE, and size. Click **GO** to issue the transaction. For write bursts, the outgoing data field is repeated for the appropriate size. A continuous traffic mode is also available through the VIO as well. It will run as long as Continuous Traffic Mode is set to 1. Figure 3-24 shows the VIO console.


VIO Console - DEV:1 MyDevice1 (XC7VX485T) UNIT:0 MyVIO0 (VIO)

Bus/Signal	Value
mode_1x	0
Last Captured Data	0000000000000000
Peak/Poke GO	0
user addr	120008
Data In	1111333333333332
user ftype	2
user ttype	4
user size	01
Device ID Override	1
Destination ID	0000
Source ID	00FF
Hop count	1
Continuous Traffic GO	0
Continuous Traffic Mode	●
Test Error	●
Reset Statistics Register	0
Reset All Statistics Registers	0
Statistics Register Address	0
Statistics Register	00000000

Figure 3-24: **VIO Console**

The IL feature of the ChipScope tool allows users to monitor traffic on the user interfaces by setting up triggers and viewing the resulting waveforms, as shown in Figure 3-25.



Trigger Setup - DEV:1 MyDevice1 (XC7VX485T) UNIT:1 MyILA1 (ILA)

Match Unit	Function	Value	Radix	Counter
TriggerPort0[35]		X		
TriggerPort0[34]		X		
TriggerPort0[33]		X		
TriggerPort0[32]		X		
TriggerPort0[31]		X		
TriggerPort0[30]		X		

Active	Trigger Condition Name	Trigger Condition Equation
●	TriggerCondition0	M0

Type: Window Windows: 1 Depth: 1024 Position: 300

Storage Qualification: All Data

Sample Buffer is full IDLE

Waveform - DEV:1 MyDevice1 (XC7VX485T) UNIT:1 MyILA1 (ILA)

Bus/Signal	X	O	
ireq_tvalid	0	0	
ireq_tready	1	1	
ireq_tlast	0	0	
ireq_tdata	0000	0000	0000200000000000
iresp_tvalid	0	0	
iresp_tready	1	1	
iresp_tlast	0	0	
iresp_tdata	0000	0000	0000000000000000
treq_tvalid	0	0	
treq_tready	1	1	
treq_tlast	0	0	
treq_tdata	0000	0000	0000000000000000
tresp_tvalid	0	0	
tresp_tready	1	1	
tresp_tlast	1	1	
tresp_tdata	0000	0000	0000000000000

Waveform captured May 29, 2012 6:24:24 PM

X: 0 O: 0 Δ(X-O): 0

Figure 3-25: **IL Feature of ChipScope Tool**

Self-Checking Example Design

The example design self checks incoming packets to ensure they are the expected type and ID. If an error does occur, the Error LED will turn red on the VIO.

Note: Do not attempt to issue single packets while in continuous traffic mode. This may cause the self-checker to detect an error.

Usable Address Space

When using the example design, reading from most memory locations results in the return of a repeatable data pattern. However, a dedicated memory space has been made for location 'h0012_xxxx. Any write to these locations will be stored in local memory, which may be retrieved at any time by issuing a follow-up read instruction. This feature can be found in `srio_response_gen.v`.

Statistics Gatherer

A set of statistics registers have been made available and can be accessed through the ChipScope tool or a user design. Use these registers for statistics, debugging, and measuring the status of the link. If using these registers through ChipScope VIO, set the desired address in the statistics address field. The desired register value will be returned real-time in the statistics data field. Reset all fields by clicking **Reset All Statistics Registers**. Reset just one field by setting the address to the appropriate register, then clicking **Reset Statistics Register.** This feature is implemented in `srio_statistics.v`.

Statistics register address mapping is shown in Table 3-12.

Table 3-12: **Statistics Register Addresses**

Address	Description
4'h0	Packets observed over the past 2^10 cycles
4'h1	Packets observed over the past 2^20 cycles
4'h2	Number of active cycles (tready & tvalid) on incoming port 1 over the past 2^10 cycles
4'h3	Number of active cycles (tready & tvalid) on incoming port 2 over the past 2^10 cycles
4'h4	Number of active cycles (tready & tvalid) on outgoing port 1 over the past 2^10 cycles
4'h5	Number of active cycles (tready & tvalid) on outgoing port 2 over the past 2^10 cycles
4'h6	Total PNAs sent
4'h7	Total PNAs received
4'h8	Total requests sent
4'h9	Total requests received
4'hA	Disparity error count
4'hB	Not-in-table error count
4'hC	Total packet retries sent
4'hD	Total packet retries received
4'hE	Returns 'h A_BAD_ADD1
4'hF	Returns 'h A_BAD_ADD1

Debug Bus

The debugging buss contains the signals shown in Table 3-13.

Table 3-13: **Debug Bus Signals**

Signal	Description
phy_debug[0]	IDLE2 selected
phy_debug[1]	Mode 1x
phy_debug[2]	IDLE1 selected
phy_debug[3]	Port initialized
phy_debug[4]	Rx lane r

Table 3-13: **Debug Bus Signals *(Cont'd)***

Signal	Description
phy_debug[5]	gtrx Channel Bond Enable
phy_debug[9:6]	0
phy_debug[13:10]	gtrx Channel is Aligned
phy_debug[29:14]	gtrx Character is Comma
phy_debug[33:30]	gttx Inhibit
phy_debug[34]	gtrx Reset Request
phy_debug[35]	Nx Lanes Aligned
phy_debug[36]	Nx Lanes Ready
phy_debug[37]	1x mode detected
phy_debug[41:38]	Init state
phy_debug[42]	Out of sync
phy_debug[95:43]	0
phy_debug[96]	Restart from Retry (RFR) sent
phy_debug[97]	Link Request (LREQ) sent
phy_debug[98]	Packet Not Accepted (PNA) sent
phy_debug[99]	Packet Retry (PR) sent
phy_debug[100]	Link Response (LRESP) sent
phy_debug[101]	Multicast Event (MCE) sent
phy_debug[102]	Packet Accepted (PA) sent
phy_debug[103]	Sent Initial Control Symbol
phy_debug[104]	Send Link Request (LREQ)
phy_debug[105]	Link reset
phy_debug[159:106]	0
phy_debug[160]	Send Packet Not Accepted (PNA)
phy_debug[161]	Packet Not Accepted (PNA) detect
phy_debug[162]	Send Packet Retry (PR)
phy_debug[163]	Packet Retry (PR) detect
phy_debug[164]	Output error stop
phy_debug[165]	Output retry stop
phy_debug[166]	Input error stop
phy_debug[167]	Input retry stop
phy_debug[191:168]	0
phy_debug[196:192]	Cause field for outgoing PNAs
phy_debug[197]	Output fatal error detect
phy_debug[198]	Output recoverable error detect

Table 3-13: **Debug Bus Signals *(Cont'd)***

Signal	Description
phy_debug[199]	Input retry detect
phy_debug[200]	Input recoverable error detect
phy_debug[201]	OLLM RX in packet
phy_debug[202]	OLLM RX framing sop
phy_debug[203]	OLLM RX framing eop
phy_debug[204]	OLLM RX framing discontinue
phy_debug[223:205]	0

SECTION II: VIVADO DESIGN SUITE

Customizing and Generating the Core

This chapter includes information on using the Xilinx Vivado tool to customize and generate the core.

The SRIO Gen2 Endpoint consists of the following components:

- Serial RapidIO Gen2 unified top-level containing the Logical Layer (LOG), Buffer Design (BUF), and Physical Layer (PHY).
- Endpoint example design

These components are generated through the Xilinx Vivado™ software using a graphical user interface (GUI).

The Serial RapidIO Gen2 top-level wrapper contains the unified core, consisting of the three subcores: LOG, BUF, and PHY. The top-level wrapper also calls the GT wrappers and the configuration fabric reference design.

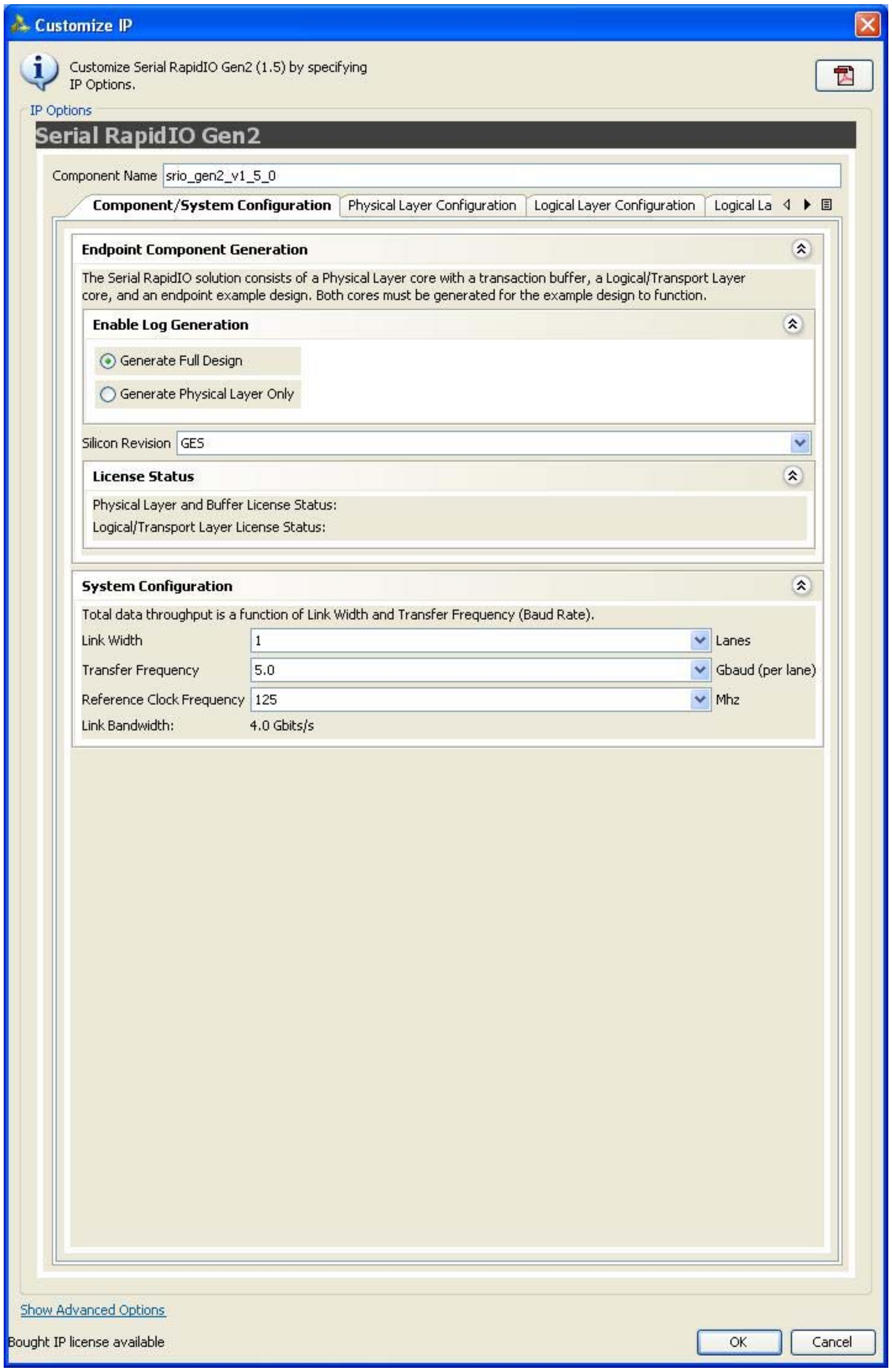
The endpoint example design sits above the core wrapper file. It contains synthesizable stimulus generators on each request interface as well as automatic response generators on all of the response ports.

GUI

This section describes the GUI pages and configuration options.

Component/System Configuration Page

Figure 4-1 shows the Serial RapidIO Gen2 Vivado software main configuration screen.



Customize IP

Customize Serial RapidIO Gen2 (1.5) by specifying IP Options.

IP Options

Serial RapidIO Gen2

Component Name: srio_gen2_v1_5_0

Component/System Configuration | Physical Layer Configuration | Logical Layer Configuration | Logical La

Endpoint Component Generation

The Serial RapidIO solution consists of a Physical Layer core with a transaction buffer, a Logical/Transport Layer core, and an endpoint example design. Both cores must be generated for the example design to function.

Enable Log Generation

[✓ Generate Full Design]

Generate Physical Layer Only

Silicon Revision: GES

License Status

Physical Layer and Buffer License Status:

Logical/Transport Layer License Status:

System Configuration

Total data throughput is a function of Link Width and Transfer Frequency (Baud Rate).

Link Width	1	Lanes
Transfer Frequency	5.0	Gbaud (per lane)
Reference Clock Frequency	125	Mhz
Link Bandwidth:	4.0 Gbits/s	

Show Advanced Options

Bought IP license available

Figure 4-1: **Vivado Main 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_gen2_v1_5.

Enable Log Generation

The LOG generation is optional. Enabling generation provides a customized netlist for the core and associated Vivado software files. The Logical Layer is not meant as a stand-alone core; a Physical Layer will be generated as well. Both layers must be generated for the example design to function.

Silicon Revision

The silicon revision is used to provide the correct components in the example design to match the silicon requirements.

Link Width

The link width represents the number of serial lanes per direction that will be generated. The Serial RapidIO Gen2 core can have one, two, or four lanes. The bandwidth of the system increases with the number of lanes.

Transfer Frequency

The transfer frequency represents the per-lane baud rate of the Serial RapidIO core. Each Serial Transceiver will be run at the selected line rate. The bandwidth of the system increases with transfer frequency.

Reference Clock Frequency

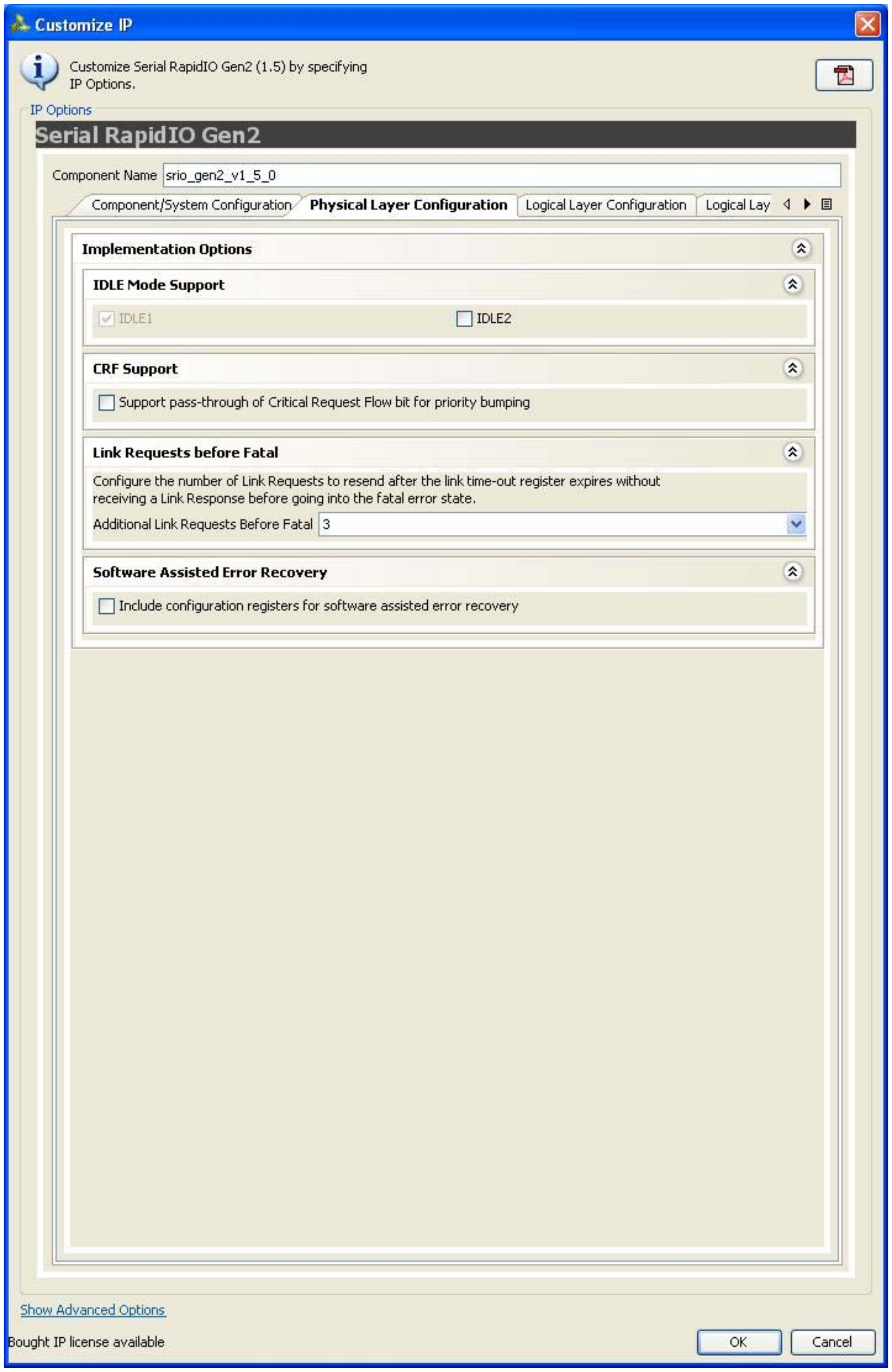
The reference clock frequency is the rate of the clock that will be brought into the FPGA on the dedicated transceiver reference clock pins.

Link Bandwidth

The maximum theoretical bandwidth of the link is calculated on this page of the GUI. This calculation accounts for the 8b/10b encoding overhead, but not the link protocol overhead (loss of bandwidth due to clock correction symbols, control symbols, etc.) or packet header overhead.

Physical Layer Configuration Page

Figure 4-2 shows the Serial RapidIO Gen2 Vivado Physical Layer configuration screen.


Customize IP
IP Options
Serial RapidIO Gen2
Component/System Configuration
Physical Layer Configuration
Logical Layer Configuration
Logical Lay
Show Advanced Options
OK
Cancel

Customize Serial RapidIO Gen2 (1.5) by specifying IP Options.

Component Name srio_gen2_v1_5_0

Implementation Options

IDLE Mode Support

[✓] IDLE1 ☐ IDLE2

CRF Support

☐ Support pass-through of Critical Request Flow bit for priority bumping

Link Requests before Fatal

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

Additional Link Requests Before Fatal 3

Software Assisted Error Recovery

☐ Include configuration registers for software assisted error recovery

Bought IP license available

Figure 4-2: **Vivado Physical Layer Configuration Screen**

IDLE Mode Support

The supported IDLE modes vary based on the transfer frequency chosen on the Component/System Generation Page. IDLE1 operation is only supported for line rates of 5.0

Gbaud or less per lane. Use of the IDLE1 sequence means that the short control symbol format will be used. The IDLE2 sequence was designed for links operating above 5.0 Gbaud per lane. Rates of 6.25 Gbaud only operate with IDLE2 sequence. The IDLE2 sequence provides additional functionality (link width and lane polarity information, plus randomized data for equalizer training for improved data recovery). If both the IDLE1 and IDLE2 sequences are enabled (only possible for 5.0 Gbaud or less), the core trains with the link partner as described in the RapidIO Specification to select an IDLE sequence.

CRF Support

Indicates whether the CRF bit is being used for extended priority mapping.

Link Requests Before Fatal

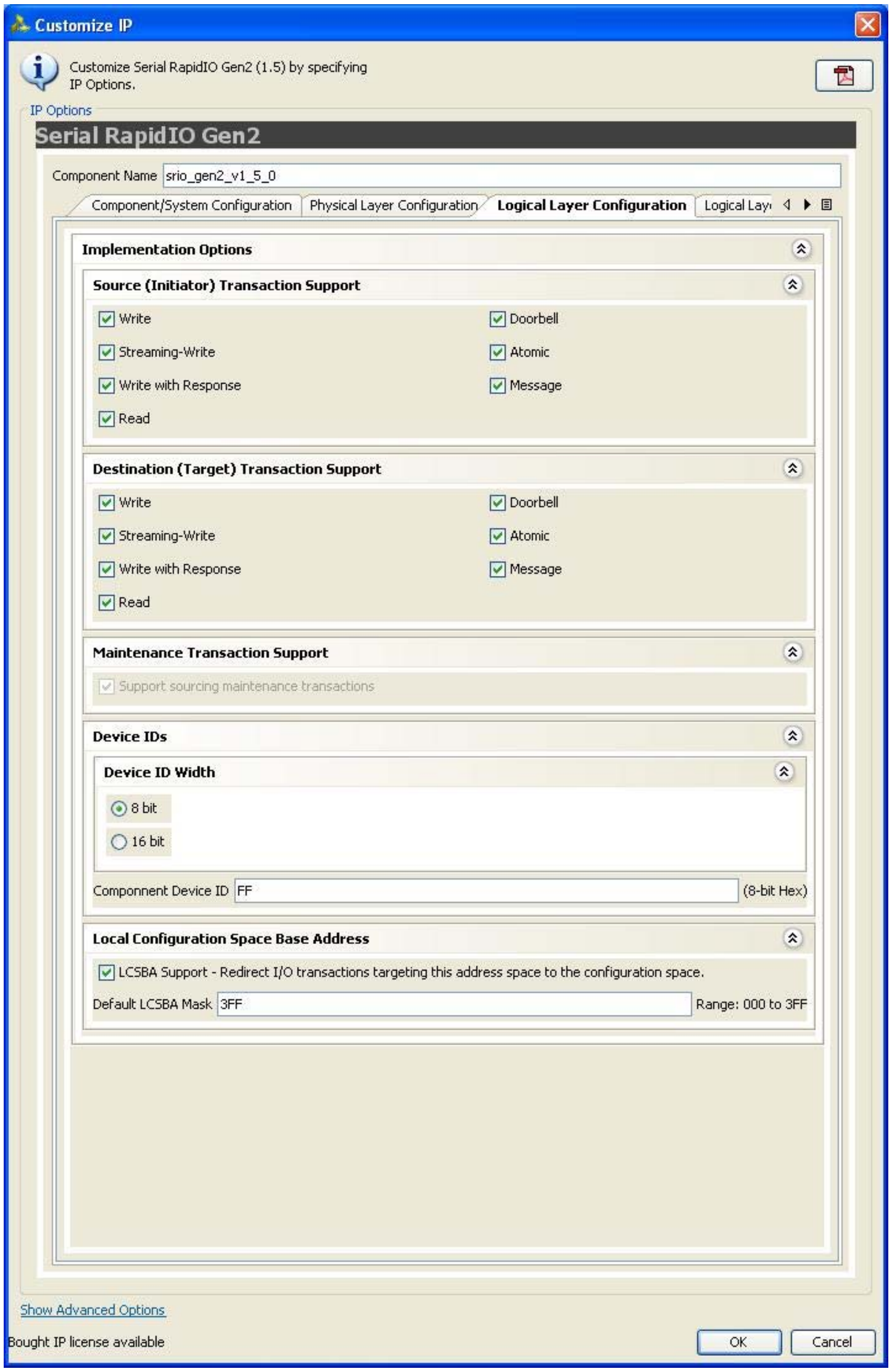
Use the drop-down list to select the number of link requests that are to be sent without receiving a link response prior to transiting to a fatal error state. Allowable options are 0 through 6 and Never Fatal. The default is zero. Xilinx recommends setting this value greater than zero for enhanced error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

Software Assisted Error Recovery

The RapidIO Specification defines three Command and Status Registers specifically for software-assisted error recovery. Specifically the Port n Link Maintenance Request CSR in Chapter 2, Port n Link Maintenance Response CSR in Chapter 2, and the Port n Local ackID CSR in Chapter 2.

Logical Layer Configuration Page

Figure 4-3shows the first page of the Serial RapidIO Gen2 Vivado Logical Layer configuration screen. This page does not exist if the Logical Layer was disabled on the first page of the GUI.



Customize IP

Customize Serial RapidIO Gen2 (1.5) by specifying IP Options.

IP Options

Serial RapidIO Gen2

Component Name: srio_gen2_v1_5_0

Component/System Configuration
Physical Layer Configuration
Logical Layer Configuration
Logical Lay

Implementation Options

Source (Initiator) Transaction Support

- [✓] Write
- [✓] Streaming-Write
- [✓] Write with Response
- [✓] Read
- [✓] Doorbell
- [✓] Atomic
- [✓] Message

Destination (Target) Transaction Support

- [✓] Write
- [✓] Streaming-Write
- [✓] Write with Response
- [✓] Read
- [✓] Doorbell
- [✓] Atomic
- [✓] Message

Maintenance Transaction Support

- [✓] Support sourcing maintenance transactions

Device IDs

Device ID Width

[✓ 8 bit] 16 bit

Componnent Device ID: FF (8-bit Hex)

Local Configuration Space Base Address

- [✓] LCSBA Support - Redirect I/O transactions targeting this address space to the configuration space.

Default LCSBA Mask: 3FF Range: 000 to 3FF

Figure 4-3: **Vivado Logical Layer Configuration Screen**

Source (Initiator) Transaction Support

Select the standard transmit operations supported by the processing element. These selections will affect the way that transactions are routed, and must be made carefully. If the

HELLO packet format is being used, the HELLO encoding and decoding logic is only generated for enabled packet types (and their corresponding responses). The SRIO Gen2 Endpoint does not support Port-Write transactions.

Destination (Target) Transaction Support

Select the standard receive operations supported by the processing element. These selections will affect the way that transactions are routed, and must be made carefully. If the HELLO format is being used, the HELLO encoding and decoding logic is only generated for enabled packet types (and their corresponding responses). The SRIO Gen2 Endpoint does not support Port-Write transactions.

Source Maintenance Support

This option determines whether the core should be capable of sourcing MAINTENANCE requests (including both local and remote requests). This is always enabled.

Device ID Width

The Device ID width of the core should match that of the link partner. Otherwise, transactions may be misinterpreted due to the header shift. Most systems use 8-bit Device IDs, but the SRIO Gen2 Endpoint also provides large system support through this option.

Component Device ID Reset Value

The Component Device ID determines the reset value for the Base Device ID CSR.

Local Configuration Space Base Address (LCSBA) Support

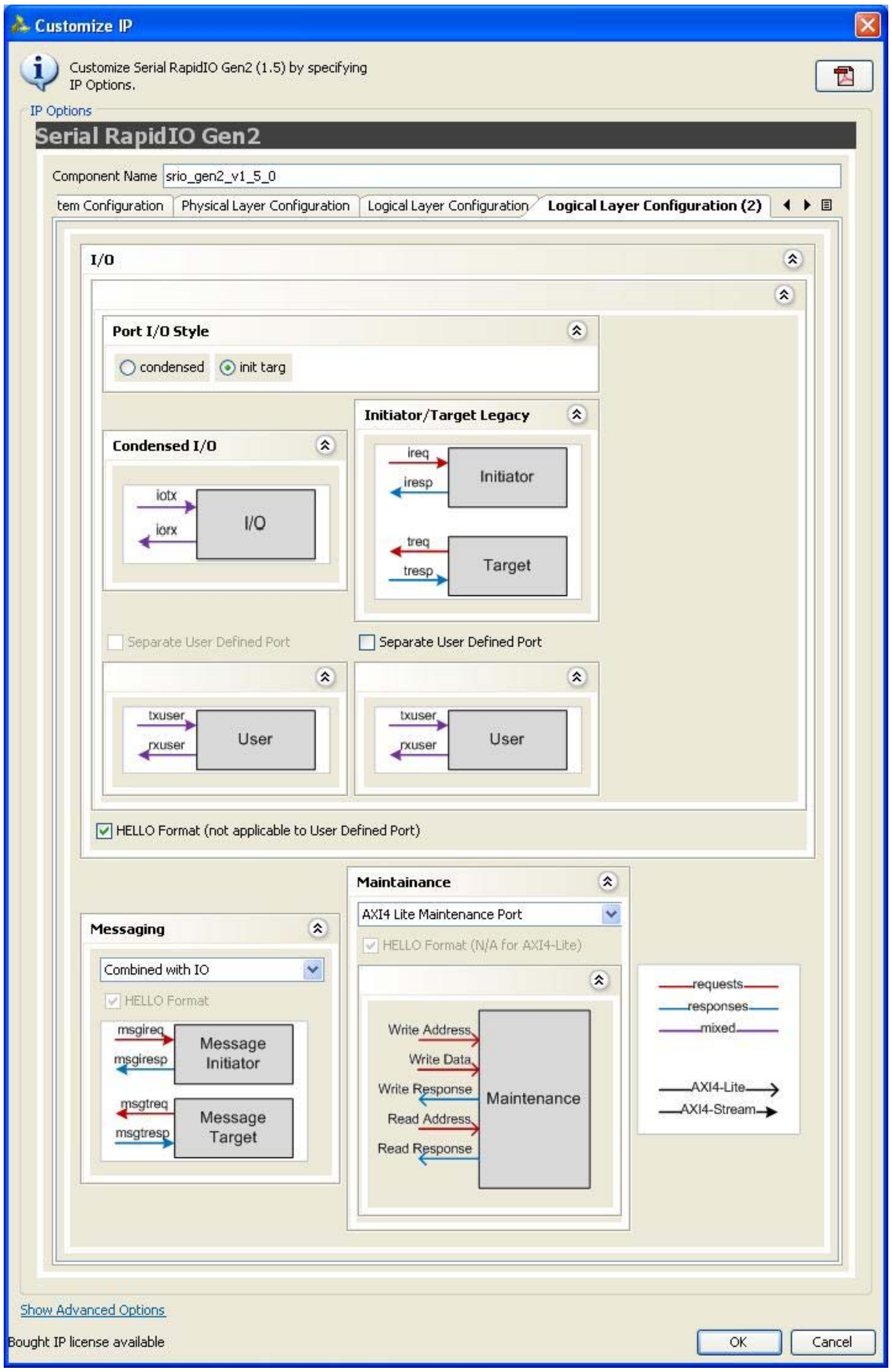
When enabled, the core checks the upper address bits of incoming I/O transactions and route the transactions to the Maintenance port if there is an address match. The specification does not provide a mechanism to disable LCSBA, so the system behavior in this case is undefined. LCSBA support should only be disabled in a closed system. Note also that LCSBA matching will only be performed by the core on HELLO formatted packets (so the I/O format must be set to HELLO and the transaction type must be supported in order for the packet to be rerouted by the core to the Maintenance port). If LCSBA support is required and these restrictions are not met, the transaction will come out the I/O or User-Defined port and the User Application will be responsible for routing it to the configuration space.

Local Configuration Space Base Address (LCSBA) Mask

Reset value for the LCSBA CSR. A 10-bit mask which is compared against the upper address bits of incoming I/O transactions, so that they may be rerouted to the configuration space when they target this address space. The core compares this value to bits [33:24] of the address, which includes the xamsbs field from the SRIO packet.

Logical Layer Configuration (2) Page

On this page, the User Interface can be customized to best fit the needs of the User Application. This page does not exist if the Logical Layer was disabled on the first page of the GUI. Figure 4-4 shows the Serial RapidIO Gen2 Vivado Logical Layer Interface Selection screen.


Customize IP

Customize Serial RapidIO Gen2 (1.5) by specifying IP Options.

IP Options

Serial RapidIO Gen2

Component Name srio_gen2_v1_5_0

tem Configuration | Physical Layer Configuration | Logical Layer Configuration | **Logical Layer Configuration (2)**

I/O

Port I/O Style: ( ) condensed [✓ init targ]

Separate User Defined Port (Condensed I/O): [ ]

Separate User Defined Port (Initiator/Target Legacy): [ ]

HELLO Format (not applicable to User Defined Port): [✓]

Messaging: Combined with IO

HELLO Format: [✓] (disabled)

Maintainance: AXI4 Lite Maintenance Port

HELLO Format (N/A for AXI4-Lite): [✓] (disabled)

Show Advanced Options

Bought IP license available

Figure 4-4: **Vivado Logical Layer Configuration Screen 2**

This page requires the user to choose a packet format for each port on the User Interface. See HELLO Packet Format in Chapter 3 and SRIO Stream Packet Format in Chapter 3 for more information.

I/O Style

The I/O interface can be configured to use one of two port styles.

- Condensed I/O - In the Condensed I/O style, there is a single transmit and a single receive AXI4-Stream channel for all I/O traffic. In this mode, the user can elect to also create a separate port for User-Defined transaction types.
- Initiator/Target - In the Initiator/Target style, transactions are sorted by the memory space that they are accessing. Initiator transactions consist of outgoing read and write requests (destined for memory at a remote device) and incoming responses. Target transactions are read and write requests which target memory in the local endpoint, plus the responses that the endpoint sends for those transactions. In this mode, the user can select a separate port for User-Defined transaction types.

I/O Format

The I/O port can be configured to use either HELLO or SRIO Stream formatted packets. For typical users, the HELLO format is strongly recommended. SRIO Stream format should only be used if full control is required. See Chapter 3, Designing with the Core for information on the two packet formats. Because User-Defined packets have user-defined header fields, the HELLO logic does not know how to construct/deconstruct them. Therefore, regardless of the I/O port style, packets on the User-Defined port use the SRIO Stream format.

Messaging Style

If MESSAGE transactions were enabled on the previous page, there are three options for the Messaging port interface style:

- Combined with IO - If MESSAGEs are combined with the I/O port, they will use the same interfaces that an I/O write transaction would use.
- Separate Messaging Port - In this style, MESSAGE transactions will have their own Initiator/Target style of port. This allows the User Application to process MESSAGE transactions independently from I/O operations.
- Combined with User-Defined Port - In this style, MESSAGE transactions are combined with User-Defined packets.

Messaging Format

The Messaging port (if a separate one exists) can be configured to use either HELLO or SRIO Stream formatted packets. For typical users, the HELLO format is strongly recommended. SRIO Stream format should only be used if full control is required. See Chapter 3, Designing with the Core for information on the two packet formats. If the port is combined with another port, MESSAGE transactions on the log interface must use the same format as that port.

Maintenance Style

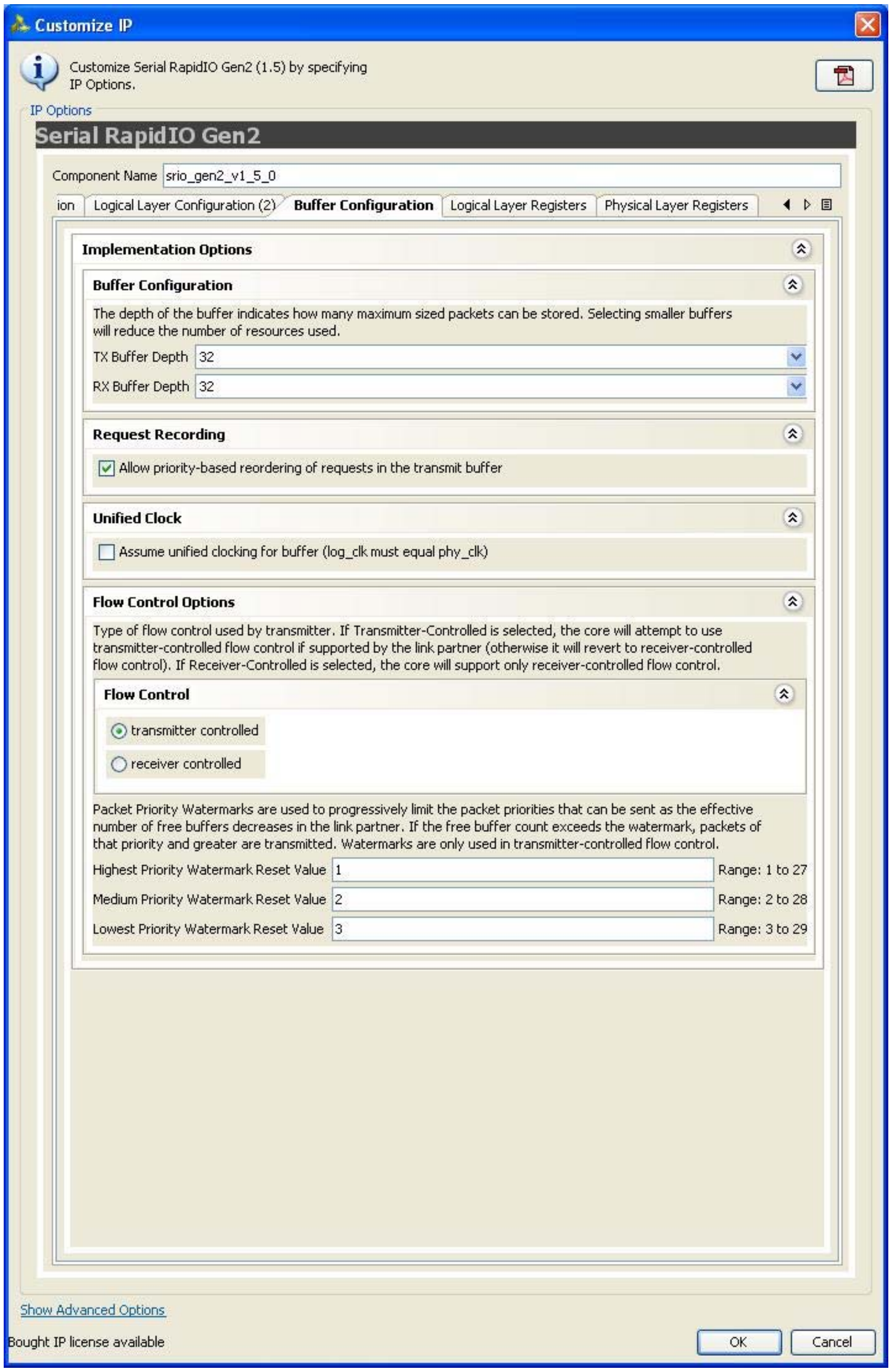
For complete handling of MAINTENANCE requests received from a remote device, the SRIO Gen2 Endpoint offers the option of an AXI4-Lite Maintenance port. If the user cannot source MAINTENANCE transactions (disabled on the previous page), there is no user Maintenance Interface, but the LOG will have an AXI4-Lite Master connection to the Configuration Fabric Reference design. If special functionality is needed, the Maintenance port can be configured in a similar way to the other User Ports with AXI4-Stream channels for direct access to MAINTENANCE packets. Maintenance traffic can also be routed to the User-Defined port if so required, by selecting Combined with User-Defined port from the drop-down menu. That option will only exist if the User-Defined port is enabled.

Maintenance Format

If the direct access to AXI4-Stream port style is chosen, the Maintenance port can be configured to use either HELLO or SRIO Stream formatted packets.

Buffer Configuration Page

Figure 4-5 shows the Serial RapidIO Gen2 Vivado Buffer Configuration screen.



Customize Serial RapidIO Gen2 (1.5) by specifying IP Options.

IP Options

Serial RapidIO Gen2

Component Name: srio_gen2_v1_5_0

Tabs: ion | Logical Layer Configuration (2) | **Buffer Configuration** | Logical Layer Registers | Physical Layer Registers

Implementation Options

Buffer Configuration

The depth of the buffer indicates how many maximum sized packets can be stored. Selecting smaller buffers will reduce the number of resources used.

TX Buffer Depth: 32

RX Buffer Depth: 32

Request Recording

[✓] Allow priority-based reordering of requests in the transmit buffer

Unified Clock

[ ] Assume unified clocking for buffer (log_clk must equal phy_clk)

Flow Control Options

Type of flow control used by transmitter. If Transmitter-Controlled is selected, the core will attempt to use transmitter-controlled flow control if supported by the link partner (otherwise it will revert to receiver-controlled flow control). If Receiver-Controlled is selected, the core will support only receiver-controlled flow control.

Flow Control

[✓ transmitter controlled] receiver controlled

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. Watermarks are only used in transmitter-controlled flow control.

Highest Priority Watermark Reset Value	1	Range: 1 to 27
Medium Priority Watermark Reset Value	2	Range: 2 to 28
Lowest Priority Watermark Reset Value	3	Range: 3 to 29

Bought IP license available

Figure 4-5: **Vivado Buffer Configuration Screen**

The depth of the transmit and receive buffers can be customized 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 (primarily Block RAMs and LUTs), whereas maximum buffer depth yields maximum throughput.

Request Reordering

Checking this allows the transmit buffer to reorder request packets. In this case, higher priority requests go out before lower priority requests. If unchecked, request packets exit in the order they were received. Regardless of this option, responses go out before requests.

Unified Clock

If the user design intends to use the same clock for `log_clk` (primary user clock) and `phy_clk`, check this box. Selecting this option significantly reduces both latency and resource utilization.

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. If the watermark value is too high for a smaller buffer, this could cause the core to lock up. If there is concern about the size of the buffer, a safe value for the watermarks would be WM0=3, WM1=2, WM2=1.

Priority 3 packets are 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:

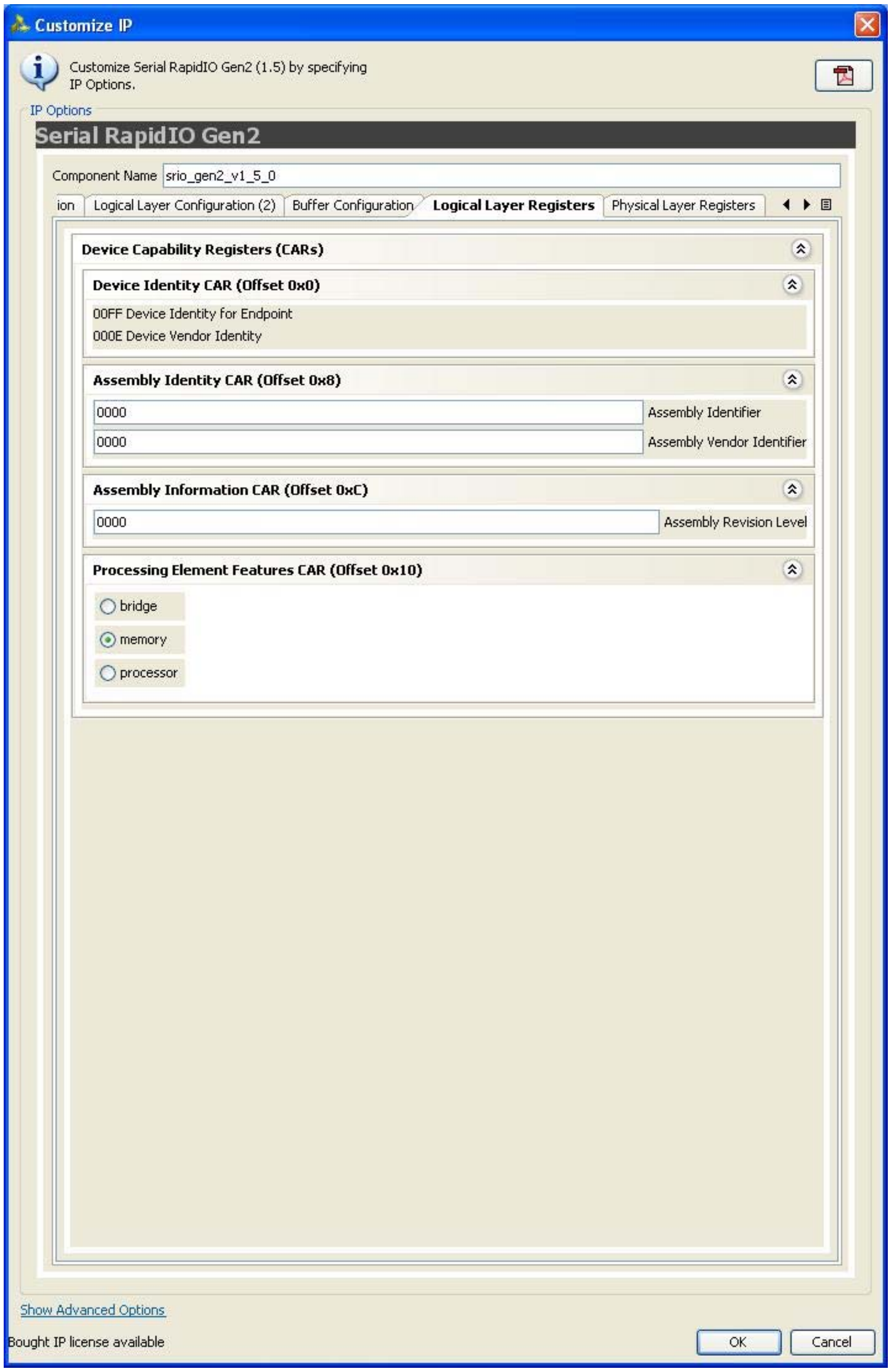
0 < WM2 < WM1 < WM0

- 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 Registers Page

Figure 4-6 shows the Serial RapidIO Gen2 Vivado Logical Layer Registers screen.



Customize IP

Customize Serial RapidIO Gen2 (1.5) by specifying IP Options.

IP Options

Serial RapidIO Gen2

Component Name: srio_gen2_v1_5_0

ion | Logical Layer Configuration (2) | Buffer Configuration | **Logical Layer Registers** | Physical Layer Registers

Device Capability Registers (CARs)

Device Identity CAR (Offset 0x0)

00FF Device Identity for Endpoint

000E Device Vendor Identity

Assembly Identity CAR (Offset 0x8)

0000	Assembly Identifier
0000	Assembly Vendor Identifier

Assembly Information CAR (Offset 0xC)

0000	Assembly Revision Level

Processing Element Features CAR (Offset 0x10)

bridge [✓ memory] processor

Show Advanced Options

Bought IP license available

OK Cancel

Figure 4-6: **Vivado Logical Layer Registers Screen**

The Logical Layer Configuration Registers hold capability and identity information for the device, as well as core status and control information.

Device Identity CAR

The Device Identity CAR stores information about a RapidIO device. This register holds Xilinx-assigned fields and is not modifiable.

Assembly Identity CAR

The Assembly Identity CAR stores information about a RapidIO device subsystem creator. The Assembly Identifier and Assembly Vendor Identifier fields of the register can be set by the user during core generation to uniquely identify the Endpoint. These values do not affect core functionality.

Assembly Information CAR

The Assembly Information CAR stores information about a RapidIO device subsystem revision. This is set during core generation and does not affect core functionality.

Processing Element Features CAR

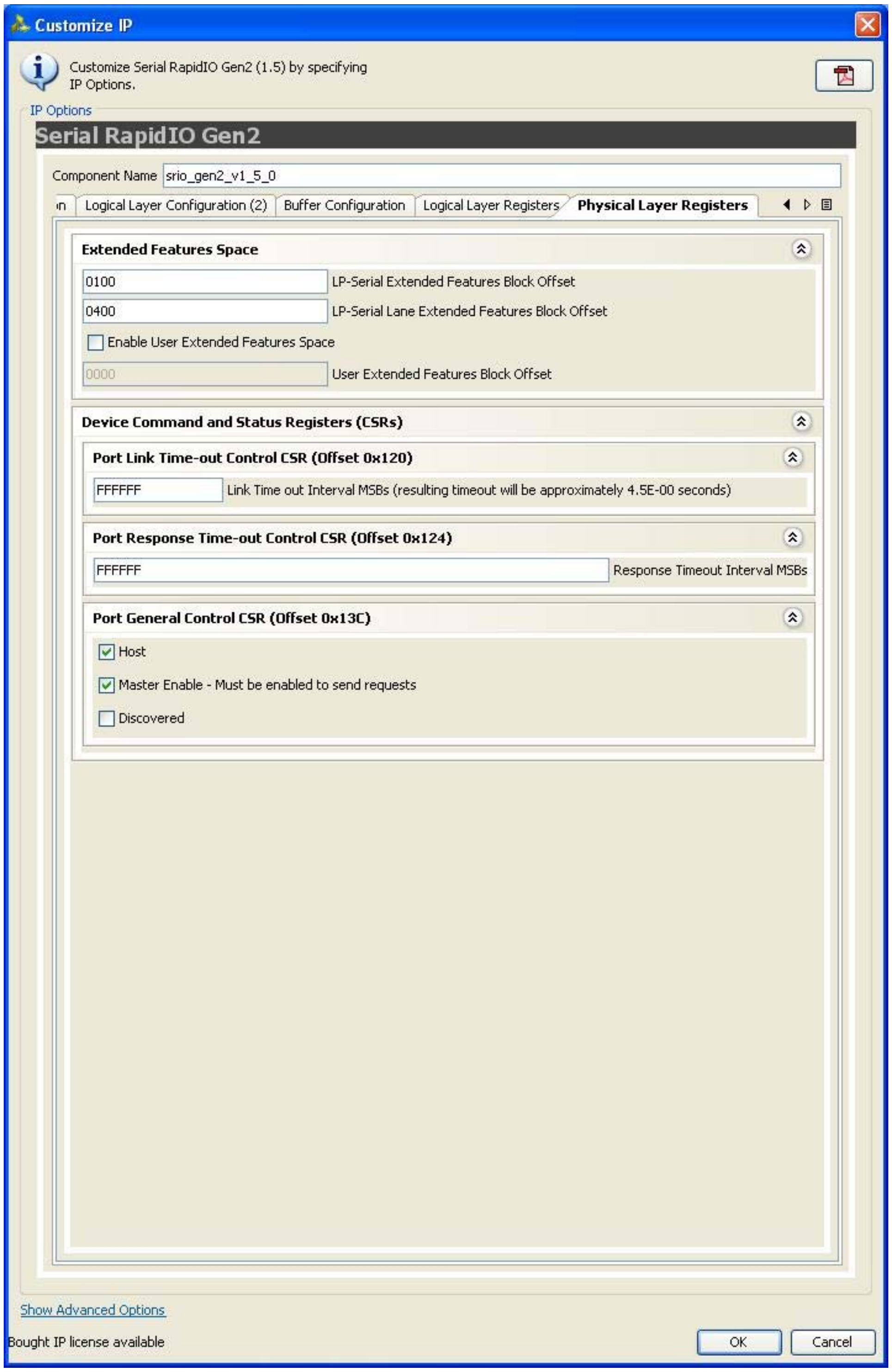
Selects the major functionality provided by the processing element. Allowable options are:

- Bridge
- Memory
- Processor

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

Physical Layer Registers Page

Figure 4-7 shows the Serial RapidIO Gen2 Vivado Physical Layer Registers screen.



Component Name: srio_gen2_v1_5_0

Extended Features Space

0100 LP-Serial Extended Features Block Offset

0400 LP-Serial Lane Extended Features Block Offset

☐ Enable User Extended Features Space

0000 User Extended Features Block Offset

Device Command and Status Registers (CSRs)

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

FFFFFF Link Time out Interval MSBs (resulting timeout will be approximately 4.5E-00 seconds)

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

FFFFFF Response Timeout Interval MSBs

Port General Control CSR (Offset 0x13C)

☑ Host

☑ Master Enable - Must be enabled to send requests

☐ Discovered

Figure 4-7: **Vivado Physical Layer Registers Screen**

The Physical Layer Configuration Registers hold core status and control information.

Extended Features Space

The Physical Layer Registers are stored in register blocks within the Extended Features (EF) address space. The Serial RapidIO Gen2 Physical Layer implements two Extended Features blocks - the LP-Serial EF block and the LP-Serial Lane EF block. The SRIO Gen2 Endpoint reserves 0x100 bytes off address space for the LP-Serial EF block, and 0x400 bytes of address space for the LP-Serial Lane EF block.

The block offset for the LP-Serial Features block must be smaller than the offset of the LP-Serial Lane EF block. Each block offset must be an integer multiple of the number of bytes reserved for the block.

The user may implement one or more EF blocks within the Extended Features address space as well. The User Application would have to create these registers somewhere and modify the Configuration Fabric reference design in order to accommodate the additional register block. If a User EF is being implemented, it should be enabled in the GUI and the block offset of the first EF block should be entered. That offset must be greater than the LP-Serial Lane EF block offset plus the 0x400 byte size that was reserved for that block. The User EF block offset must be a multiple of 0x100 and within the Extended Features address space.

See Extended Features Space in Chapter 2 for more information.

Port Link Timeout Control CSR Reset Value

Timeout 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 specification is followed. The max timeout scales linearly with the value in this CSR. A max time-out value of FF_FFFFh corresponds to a time-out length of approximately 4.5 seconds, but the max value is only accurate to within +/- 33%.

Port Response Timeout CSR Reset Value

Timeout 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 between 3 and 6 seconds. Implementation of the response timeout is left to the user.

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.

Note: If Master Enable is unchecked, the device will not be allowed to initiate requests until the bit is written with a 1.

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.

Output Generation

See Directory and File Contents in Chapter 6 for details about files and directories created when generating the core.

Chapter 5

Constraining the Core

This chapter defines the constraint requirements of the SRIO Gen2 Endpoint example design. An example constraints file (XCD) is provided with the example design, which implements the constraints defined in this chapter.

For each device family, the constraints file and implementation scripts will target one specific device (for example, when a Kintex 7 device is selected, an XC7K325T-FFG900 device will be targeted).

The example design and XCD can be used (retargeted) for other devices within the FPGA family. Care must be taken to select a proper pins and area group locations when a different part is selected through the GUI. Information is provided in this chapter to indicate which constraints to modify for those cases.

The constraints defined in this section are implemented in the `srio_example_top.xcd` file for the SRIO Gen2 Endpoint example design. See Chapter 6, Detailed Example Design for information about the location of the XCD.

Device, Package, and Speed Grade Selections

The SRIO Gen2 Endpoint can be implemented in Virtex-7, and Kintex™-7 devices with the following attributes:

- Large enough to accommodate the SRIO Gen2 Endpoint design
- Fast enough speed grade to meet the requirements listed in the Serial RapidIO Gen2 Product Specification
- Transceivers capable of running at the core's configured line rate

Clock Frequencies

The XCD applies the clock constraint on the input reference clock, which gets pushed through the clocking components and applied to each distinct clock domain. If a second source clock is used, a similar constraint must be applied to that clock as well.

The clock constraint specifies the frequency and duty-cycle of the source clock signal in the design. The following is the applied constraint for a 125 MHz reference frequency.

```
create_clock -period 8 -name sys_clkp -waveform {0 4} [get_ports sys_clkp]
```

For other configurations, the clock period is adjusted accordingly.

Clock Management

Advanced timing closure techniques may be needed in order to meet timing for 5.0 Gbaud or greater operation. For example, an area group may facilitate placement so that routing delays are minimized. Generally, the area group should be large enough to contain enough logic resources for the core, but small enough to have an impact on timing. Optimal size varies with core configuration.

Example syntax:

```
create_pblock pblock_phy
add_cells_to_pblock [get_pblocks pblock_phy] [get_cells -quiet -hierarchical
*ollm_tx_top_inst]
add_cells_to_pblock [get_pblocks pblock_phy] [get_cells -quiet -hierarchical
*ollm_rx_top_inst]
resize_pblock [get_pblocks pblock_phy] -add {SLICE_X122Y300:SLICE_X137Y349}
```

Currently, no configurations use area groups. The user will note that any area groups in the XCD are currently commented out. See UG612, Xilinx Timing Constraints User Guide for more information on area group constraints.

Additional steps that may be necessary to meet timing in certain devices:

- Location constraints for block RAMs
- Changing software options
- Reconfiguring the core with fewer features

Clock Placement

No clock placement constraints are required.

Banking

No banking constraints are required.

Transceiver Placement

The provided XCD uses placement constraints to specify the serial transceivers that will be used when the core is implemented. These can be moved around, but all lanes within a given core should be placed in adjacent transceivers for best timing results. The following constraints are provided by default with 4x Kintex-7 FPGA cores.

```
set_property LOC E3   [get_ports srio_rxn0]
set_property LOC E4   [get_ports srio_rxp0]
set_property LOC D1   [get_ports srio_txn0]
set_property LOC D2   [get_ports srio_txp0]
set_property LOC D5   [get_ports srio_rxn1]
set_property LOC D6   [get_ports srio_rxp1]
set_property LOC C3   [get_ports srio_txn1]
set_property LOC C4   [get_ports srio_txp1]
set_property LOC B5   [get_ports srio_rxn2]
set_property LOC B6   [get_ports srio_rxp2]
set_property LOC B1   [get_ports srio_txn2]
set_property LOC B2   [get_ports srio_txp2]
set_property LOC A7   [get_ports srio_rxn3]
set_property LOC A8   [get_ports srio_rxp3]
set_property LOC A3   [get_ports srio_txn3]
set_property LOC A4   [get_ports srio_txp3]
```

I/O Standard and Placement

The XCD constrains the I/O placement and I/O standard. These may be moved to any free location that also meets I/O placement and banking rules. The following is an example of the constraints applied to the Kintex-7 FPGA cores.

```
set_property LOC C8  [get_ports sys_clkp]
set_property LOC C7  [get_ports sys_clkn]
set_property LOC K18 [get_ports sys_rst]

set_property IOSTANDARD LVCMOS18 [get_ports sys_clkp]
set_property IOSTANDARD LVCMOS18 [get_ports sys_clkn]
set_property IOSTANDARD LVCMOS18 [get_ports sys_rst]
```

Detailed Example Design

This chapter introduces the SRIO Gen2 Endpoint reference design that is included with the SRIO Gen2 Endpoint. The reference design demonstrates how to generate the SRIO Gen2 Endpoint, including the LOG, BUF, and PHY provided through the SRIO Gen2 Endpoint top-level wrapper. In addition, the reference design illustrates using the default options in the SRIO Gen2 Endpoint example design.

Overview

The SRIO Gen2 Endpoint example design connects two instances of the core together. Each core instance can be configured to send supported packet types, check for receive packet mismatches, and report in the simulation transcript with details about the link traffic. Both the simulation host and the primary core being tested use the SRIO Gen2 Endpoint reference design (consisting of a LOG, BUF, and PHY provided through the SRIO Gen2 Endpoint top-level wrapper) and the Configuration Fabric reference design.

Generating the Core

Before using the SRIO Gen2 Endpoint, the PHY and LOG must be generated through the Vivado software. To generate a core, first create a Vivado project:

1. Start the Vivado software.
2. Choose **File** > **New Project**, click next.
3. Specify a project name and a project directory, click next. In this example, the directory is named <project_dir>.
4. Select **RTL Project**, click next.
5. If you have RTL sources already, add the files. Otherwise, do not add files. Click next.
6. Bypass adding existing IP, click next.
7. If you have an existing XDC file, add it now. Otherwise, skip this step. Click next.
8. Select either a **Kintex7** or **Virtex7** device from the device list. Click next.

9. Click finish.

Once the Vivado project is created, generate the SRIO Gen2 Endpoint:

1. In the left pane, click **IP Catalog**.
2. In the IP Catalog pane, locate **Serial RapidIO Gen2**, either by searching or by pushing through the hierarchy (Standard Bus Interface > RapidIO or Communications & Networking > Wireless).
3. Double-click the Serial RapidIO Gen2 core.
4. Specify a component name. In this example, the component is named <srio_component_name>.
5. Accept the default values on the screen, and click OK.

In order to obtain all of the files for the example design:

1. Locate the SRIO Gen2 core in the Hierarchy tab.
2. Right-click and select **Generate**. Select **All** and click OK.

Cores are generated into the <project_dir>/<project_dir>.src/sources_1/ip/<srio_component_name> directory. Also generated in the <srio_component_name> directory are the supporting files for the core, including the SRIO Gen2 Endpoint example design. Detailed information about the files and directories delivered with the SRIO Gen2 Endpoint core is contained in Directory and File Contents.

Directory and File Contents

The complete LogiCORE™ IP SRIO Gen2 Endpoint consists of the following components:

- SRIO Gen2 Unified top-level wrapper containing:
  - Serial RapidIO Gen2 Physical Layer (PHY)
  - Serial RapidIO Gen2 Buffer Design (BUF)
  - Serial RapidIO Gen2 Logical Layer (LOG)
- SRIO Gen2 Endpoint Example Design

The SRIO Gen2 Endpoint includes a configuration fabric reference design, example clock and reset modules, and an example user design.

The SRIO Gen2 Endpoint 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 three sub-cores (PHY, BUF, and LOG) must be generated in the same Vivado project directory.

As illustrated in the following directory structure, the Vivado project is <project_dir>; the component name for the SRIO Gen2 Endpoint top-level wrapper is <srio_component_name>.

SRIO Gen2 Endpoint

- <project_dir>/<project_dir>.src/sources_1/ip/<srio_component_name>
  Top-level project directory for the Vivado software project.
  - <srio_component name>/doc
    Contains the SRIO Gen2 Endpoint documentation and release notes text file and html file.
  - <srio_component_name>/example_design
    Contains the source files necessary to create the SRIO Gen2 Endpoint example design.
  - <srio_component_name>/example_design/cfg_fabric
    Contains the source files for the configuration fabric component.
  - <srio_component_name>/implement
    Contains the supporting files for synthesis and implementation of the SRIO Gen2 Endpoint example design.
  - <srio_component_name>/simulation
    Currently empty and used to maintain the hierarchy structure from the previous generation of the core.
  - <srio_component_name>/simulation/functional
    Contains the scripts and define files for simulating the SRIO Gen2 Endpoint example design in ModelSim.

SRIO Gen2 Endpoint

The following tables contain the files and their descriptions specific to the SRIO Gen2 Endpoint.

<project_dir>/<project_dir>.src/sources_1/ip/<srio_component_name>

This directory contains the directories for the SRIO Gen2 Endpoint example design and the implementation and simulation examples.

Table 6-1: **Project Directory**

Name	Description
<project_dir>/<project_dir>.src/sources_1/ip/<srio_component_name>	
<srio_component_name>.veo	The HDL template for the SRIO Gen2 Endpoint top-level wrapper.
<srio_component_name>.xml	Core file.
<srio_component_name>.xci	Core configuration options file.

<srio_component name>/doc

This directory contains the documentation that is included with the core.

Table 6-2: **Doc Directory**

Name	Description
<srio_component_name>/doc	
pg007_srio_gen2.pdf	LogiCORE IP Serial RapidIO Gen2 Endpoint Product Guide
srio_gen2_<version number>_vinfo.html	Readme HTML file for the current version of the core.
srio_gen2_<version number>_readme.txt	Readme TXT file for the current version of the core.

<srio_component_name>/example_design

This directory and its subdirectories contain all the source files, aside from the SRIO Gen2 core netlist, to create the SRIO Gen2 Endpoint example design. They include the reference design clocking module, reset module, example user design, and user constraints file.

Table 6-3: **Example Design Directory**

Name	Description
<srio_component_name>/example_design	
srio_example_top.xdc	Constraints file for the RapidIO Gen2 Endpoint example design.
srio_example_top.v	The top-level HDL file for the RapidIO Gen2 Endpoint example design
srio_clk.v	HDL clock module for generation of core clocks.
srio_wrapper.v	HDL wrapper file that instantiates the SRIO Gen2 Endpoint top-level wrapper, GT wrapper, and configuration fabric reference design.
srio_rst.v	Example reset module that generates and sequences the necessary resets from a sys_rst input.
srio_gt_wrapper_<fam>_<width>.v	Transceiver wrapper file.
gt_wrapper.v	GT wrapper from 7 series Transceiver Wizard for 7 series FPGA configurations.
gt_wrapper_gt.v	GT wrapper from 7 series Transceiver Wizard for 7 series FPGA configurations.

Table 6-3: **Example Design Directory *(Cont'd)***

Name	Description
gt_wrapper_ver.xco	XCO file used to generate GT wrapper files from the 7 Series Transceiver Wizard for 7 series FPGA configurations.
srio_sim.v	The srio_sim module instantiates two cores and connects them together for simulation. It also contains a timeout and mechanisms to report on whether the test passed or failed.
srio_dut.v	This module instantiates srio_wrapper and the clocking and reset modules.
srio_request_gen.v	This module generates request transactions to be sent to the link partner. It is parameterized with transaction types so that only transactions appropriate to the port and supported by the core will be sent.
instruction_list.v	This file is included into srio_request_gen and consists of a list of transactions that could be sent.
srio_response_gen.v	This file contains logic which creates responses to incoming transactions (if they are response-generating packet types).
srio_condensed_gen.v	This module only exists when the I/O port is configured in Condensed I/O mode. It allows the srio_request_gen module to issue transactions in Condensed I/O mode.
srio_quick_start.v	This module connects to the Maintenance port and issues transactions as part of the example design.
maintenance_list.v	Included into srio_quick_start, this file contains the maintenance instructions that will be issued by the example design.
srio_report.v	This file displays when packets are transmitted and/or received.
srio_statistics.v	This file collects statistics from the core and delivers information through a register interface, accessible via Chipscope or a user design.

<srio_component_name>/example_design/cfg_fabric

This directory contains the source files for the Configuration Fabric reference design. The configuration fabric serves as an AXI4-Lite interconnect that forwards packets to the appropriate core's configuration register port based upon the configuration offset of the transaction. The source code contains comments to help explain the functionality of the code. It is not required to use this configuration fabric in the design; however, some type of interconnect will be needed.

Table 6-4: **Configuration Fabric Directory**

Name	Description
<srio_component_name>/example_design/cfg_fabric	
cfg_fabric.v	File containing the configuration fabric reference design.

<srio_component_name>/implement

This directory contains the support files necessary for synthesis and implementation of the SRIO Gen2 Endpoint example design with the Xilinx tools.

Table 6-5: **Implementation Directory**

Name	Description
<srio_component_name>/implement	
planAhead_rdn.sh	Linux shell script that processes the example design through the Xilinx Vivado software flow.
planAhead_rdn.bat	Windows script that processes the example design through the Xilinx Vivado software flow.
planAhead_rdn.tcl	Underlying TCL script for use on the Vivado software TCL command line.

<srio_component_name>/simulation

This directory contains the functional subdirectory.

<srio_component_name>/simulation/functional

This directory contains the scripts and defines the files for simulating the SRIO Gen2 Endpoint example design in ModelSim.

Table 6-6: **Functional Directory**

Name	Description
<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.
wave.do	A ModelSim macro file that displays helpful waveform signals.

Example Design

The srio_example_top module instantiates all components of the core and example design that are needed to implement the design in hardware, as shown in Figure 6-1. This includes the clock and reset modules, the configuration fabric, and stimulus generators to create transactions.

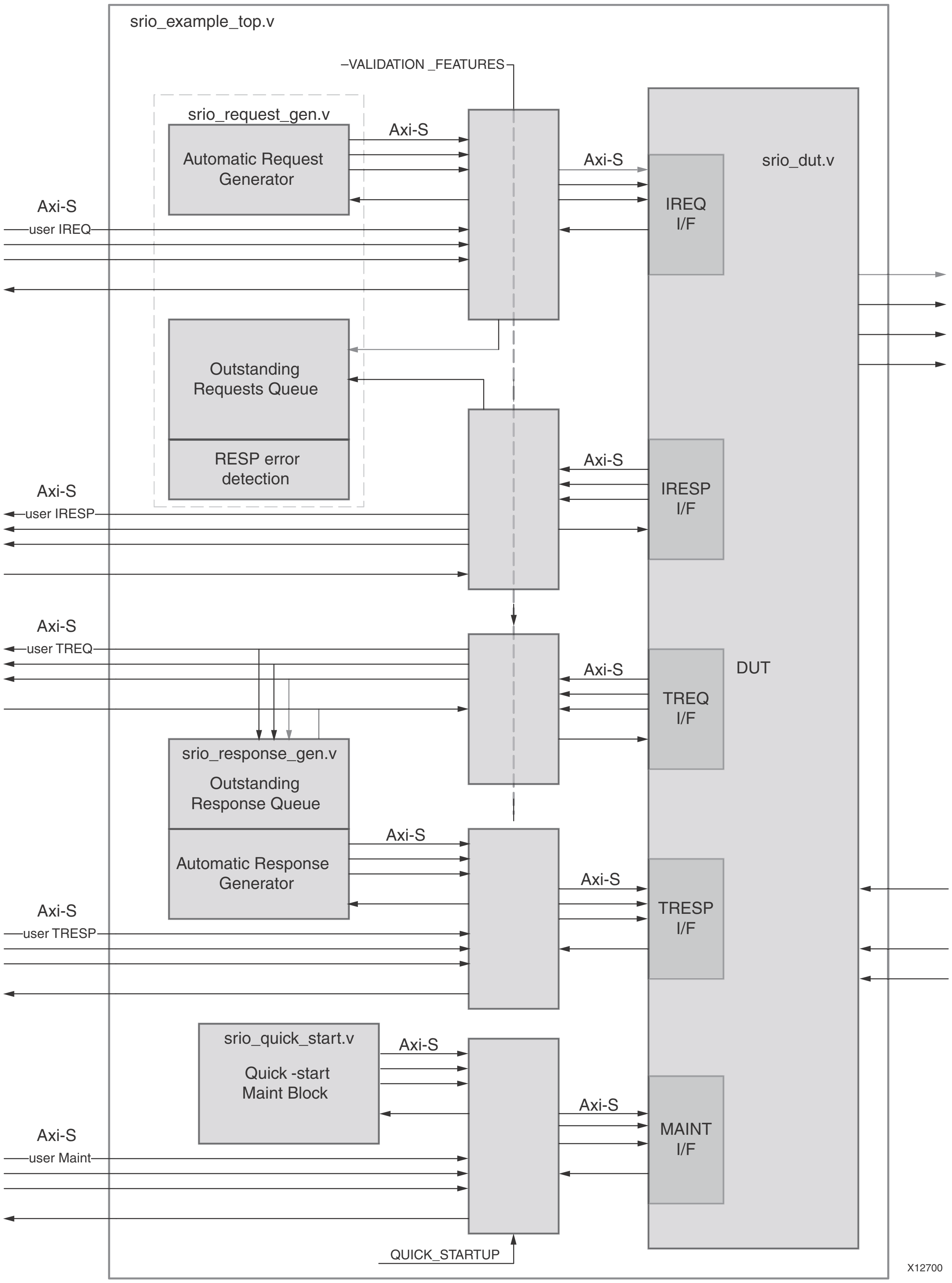


Figure 6-1: **Example Design Top Layer**

The srio_quick_start module, which is instantiated in srio_example_top, connects to the Maintenance port and generates maintenance transactions. The user can add, modify, or remove transactions by editing the maintenance_list file. The srio_request_gen module, which is instantiated within the srio_example_top, is where I/O (and message) transactions are generated. The transaction types to be generated are passed down through parameters so that only transactions supported on the connected port are produced. This file also stores the expected response for any response-generating-requests that it creates, and compares received responses to the expected value. The srio_response_gen module, also instantiated in srio_example_top, generates responses to requests received from the link partner.

User vs. Automatic Stimulus

There are two ways IO stimulus can be generated: automatically by the example design or externally by the user through the initiator and target request/response interfaces, as shown in Figure 6-1. There are also two ways the maintenance transactions can be generated: automatically by the example design or externally by the user through the maintenance interface, as shown in Figure 6-1. In Figure 6-2, the maintenance interface is configured as an AXI4-Stream interface, but this interface can also be configured as an AXI4-Lite interface if needed.

By default, the example design is configured to generate stimulus automatically (for both IO and maintenance transactions) and bypass the external interfaces. This default is achieved in the srio_example_top module by setting `VALIDATION_FEATURES=1` and `QUICK_STARTUP=1`. In addition, it is necessary to comment out the external interfaces to save pins and improve timing.

To use the external interfaces for IO generation, set the parameter VALIDATION_FEATURES=0 and uncomment the external interfaces (`axis_ireq_*`, `axis_tresp_*`, `axis_iresp_*`, `axis_treq_*`, `axis_iotx_*`, and `axis_iorx_*`) in srio_example_top. To use the external interfaces for maintenance transactions, set the parameter `QUICK_STARTUP=0` and uncomment the external interface (`axis_maintr_*`) in the srio_example_top module. srio_example_top contains comments to aid in this process, as shown in Figure 6-2.

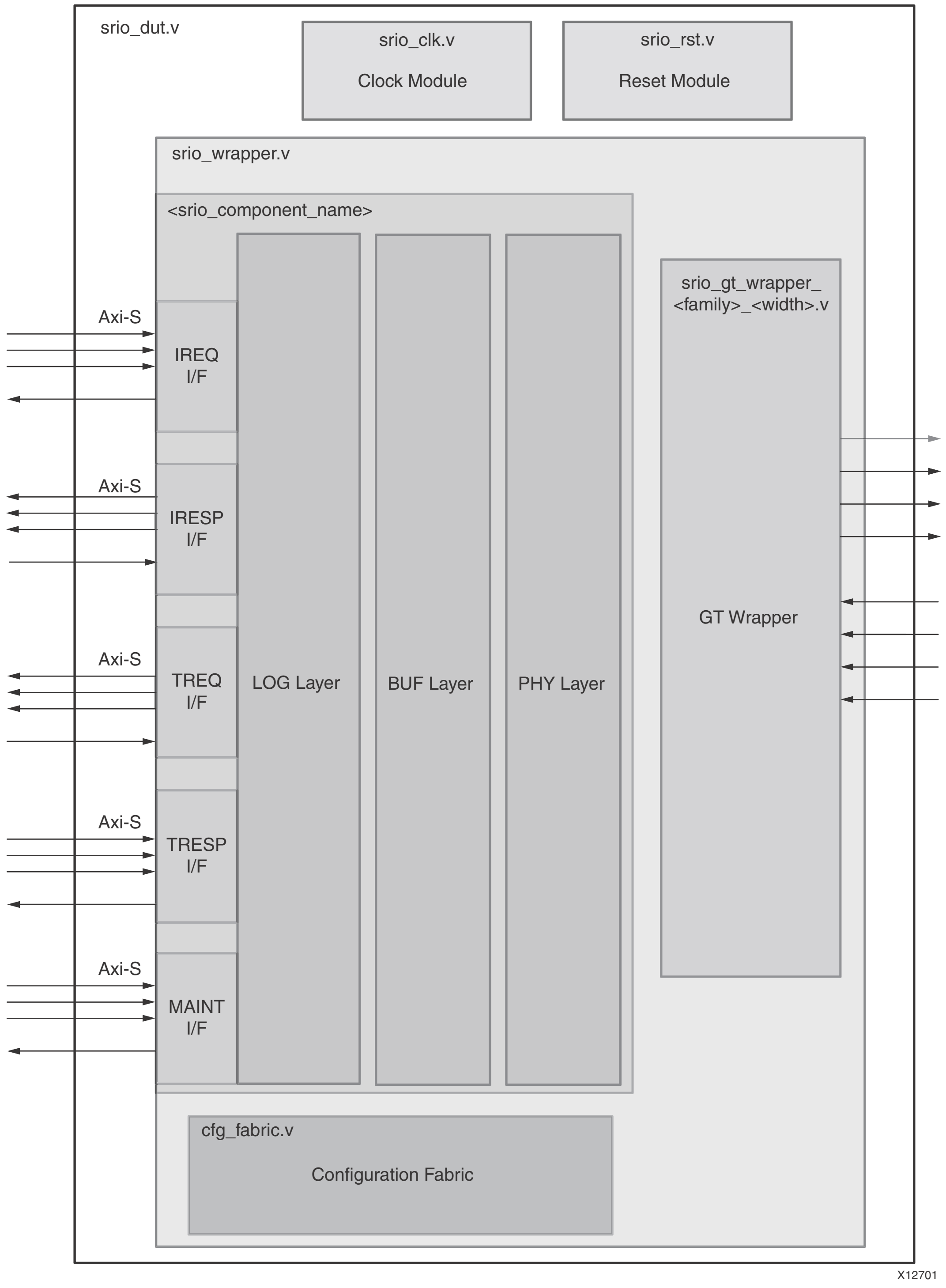


Figure 6-2: **RTL Layer Block Diagram**

Automatic Stimulus Generation

By default, the example design generates stimulus automatically for both the IO and maintenance transactions. The maintenance transactions are generated in the srio_quick_start module, and the transactions are specifically listed out in the `maintenance_list.vh` file.

These maintenance transactions are examples of a typical configuration of the core when coming out of reset. Transactions can also be added, modified, or removed in `maintenance_list` module. The `srio_quick_start.vh` is intended to be implemented in designs for managing common maintenance transactions if no processor is needed. This method is only good for setting up the core when coming out of reset.

The IO transactions are generated in the `srio_request_gen` and `srio_response_gen` modules. The `srio_request_gen` generates the initiator request IO transactions as specified in `instruction_list.vh` file.

Transactions can be added, modified, or removed in `instruction_list.vh`. The list contains random transactions, but is run in the same order each time. Only transactions supported by the connected port are produced. Figure 6-1 shows an example of a core configured with Initiator/Target Legacy ports. The `srio_request_gen` also keeps track of any expected responses for response-generating-requests, and compares the received response to the expected value. This comparison function is only available when using the automatic stimulus generation.

The `srio_response_gen` generates the target response IO transactions in response to requests received from the link partner. The data payload of write requests that fall within the scratch space of x0012_xxxx are stored in a local memory. All other write requests that do not require a response are dropped, and the `srio_response_gen` does not respond to SWRITE transaction types. Read requests that fall in the scratch space of x0012_xxxx get a response with data from that actual address. For instructions with an address offset outside of the scratch space, the response generator automatically fills the IO transaction data with an incrementing pattern. The responses are generated in the order that the requests are received; so no out-of-order transactions are generated. In all cases, response priorities are equal to the request priority plus 1.

Each instance of the `srio_request_gen` requires one block RAM. Each instance of `srio_response_gen` requires two block RAM resources.

Configuration Fabric

The Configuration Space is distributed across all the blocks of the SRIO Gen2 Endpoint. The Configuration Fabric reference design, located in the `cfg_fabric` module of the example design, manages the accesses to the configuration spaces of each of the blocks. This makes them appear unified to the user or processor element. The configuration modules in each of the blocks are slaves on the configuration bus (an AXI-Lite interface), and the Configuration Fabric module is the bus master. Reads and writes derived from Maintenance transactions that were issued either locally or from the link partner are presented to the Configuration Fabric via the Configuration Master port in the LOG block. The Configuration Fabric will route the reads and writes to the appropriate block, but will not route any transactions if the address is outside the valid range for the respective configuration registers. Writes to illegal space are dropped, and reads from illegal space are returned as zeros. See Accessing the Configuration Space in Chapter 3 for more detail.

Clock Module

The clock module in the example design is identical to the clock module delivered with the core, as described in Clocking in Chapter 3. The srio_clk module consists of one MMCM_ADV, one IBUFDS, and either three or four BUFGs depending on the selected family. Note that in 2x or 4x mode, one of the BUFGs is converted to a BUFGMUX. The srio_clk module generates the appropriate reference frequencies and user clocks, depending on the configuration.

Reset Module

The srio_rst module, as shown in Figure 6-2, takes in one asynchronous reset, and outputs a pulse-extended synchronized reset for each clock domain. The srio_rst module also contains a state machine to cause a forced re-initialization when called by the user. This module also resets the link when instructed to do so by the link partner.

Demonstration Test Bench

The srio_sim module, as shown in Figure 6-3, connects two instances of the example design (at the srio_example_top level) together. It generates the system clock and reset for simulation purposes, and monitors the simulation progress for status, including packet mis-compares, timeouts, and a tests complete flag. The two instances of the example design are srio_example_top_primary, which represents the user core, and srio_example_top_mirror, which acts as the link partner. The srio_example_top_mirror is configured to report received and transmitted transactions into the simulation transcript. When a simulation completes, either due to an error or when all requests have been sent and responses received, the simulation will stop and report into the transcript with the simulation status.

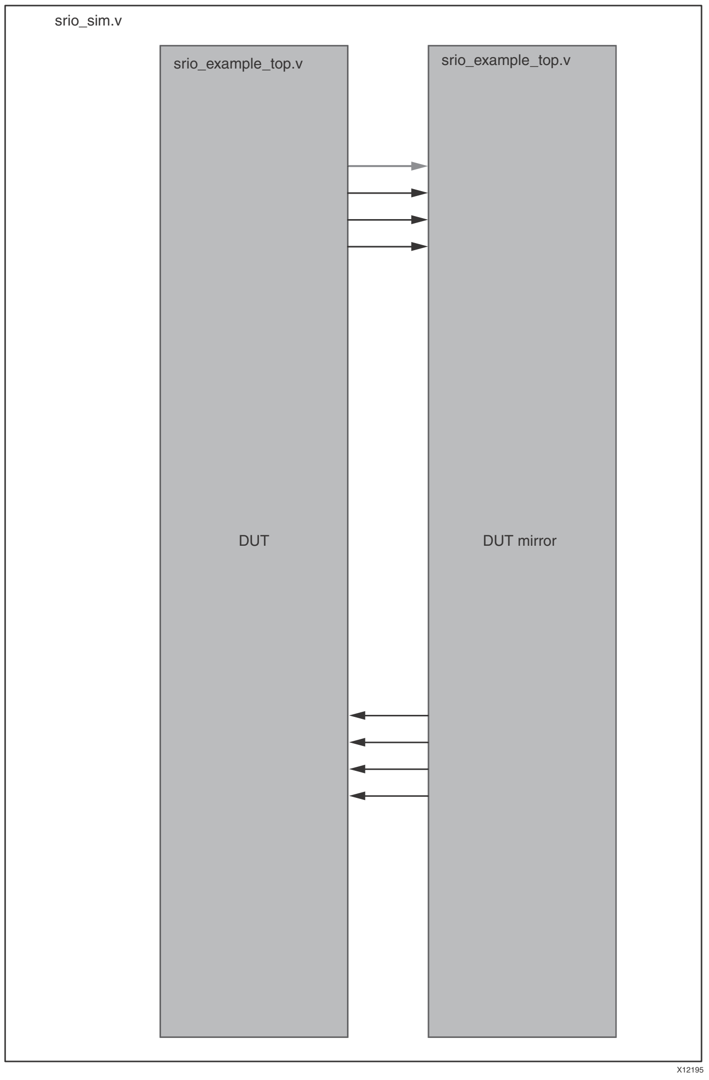


Figure 6-3: **Example Design Test Bench Layer**

Implementation

The recommended implementation flow through Vivado is through the GUI. After following the steps described in Generating the Core, follow these instructions:

1. Right-click on the core in the Hierarchy window, and select **Open IP Example Design...**
2. A new window will pop up, asking you to specify a directory for the example design. Select a new directory, or keep the default directory.
3. A new Vivado project will pop up. Click **Run Implementation** in the left side pane and follow the directions.

Simulation

Using the SRIO Gen2 Endpoint example design (delivered as part of the Serial SRIO Gen2 Endpoint), you can quickly simulate and observe the behavior of the SRIO Gen2 Endpoint.

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.

The Xilinx simulation libraries must be mapped into the simulator. If the libraries are not set for your environment, please refer to the *Synthesis and Simulation Design Guide* on the Xilinx software manuals Web page for assistance compiling Xilinx simulation models and setting up the simulator environment.

Simulations including the serial transceivers require a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator.

Functional Simulation

This section contains instructions for running a functional simulation of the SRIO Gen2 Endpoint using Verilog. A functional simulation model for the SRIO Gen2 Endpoint is provided when the core is 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_component_name> for the component name of the SRIO Gen2 Endpoint are used.

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

1. Launch the ModelSim simulator and set the current directory to

   ```
   <project_dir>/<project_dir>.src/sources_1/ip/<srio_component_name>/
   <srio_component_name>/simulation/functional
   ```

2. Launch the simulation script:

   ```
   modelsim> do simulate_mti.do
   ```

Self Checking Simulation

The simulation script compiles the SRIO Gen2 Endpoint functional simulation model, the SRIO Gen2 Endpoint example design, and supporting simulation files. It then runs the simulation and checks to ensure that it completed successfully. The simulation uses self-checking in the srio_request_gen module, as previously described. If any response does not match the expected type or ID, the test bench will report a mismatch:

```
*****************************************************************
******************** ERROR: TEST FAILED ************************
************ Packet Miscompare or Packet Corruption *************
*****************************************************************
```

If a response never returns, the simulation will eventually timeout:

```
#################################################################
################## ERROR: TEST FAILED !!!!! #####################
#################### Testbench Timed Out! #######################
######################### (<timestamp>) #########################
#################################################################
```

A simulation success will be indicated by this text:

```
*****************************************************************
************************ TEST PASSED ***************************
************* All Packets Checked and Accounted For *************
*****************************************************************
```

To observe more details of the operation of the core, inspect the simulation transcript and the waveform.

Messages and Warnings

SECTION III: ISE DESIGN SUITE

Customizing and Generating the Core

This chapter includes information on using the Xilinx CORE Generator™ tool to customize and generate the core.

The SRIO Gen2 Endpoint consists of the following components:

- Serial RapidIO Gen2 unified top-level wrapper containing Logical Layer (LOG), Buffer Design (BUF), and Physical Layer (PHY)
- Endpoint example design

These components are generated through the CORE Generator software using a graphical user interface (GUI).

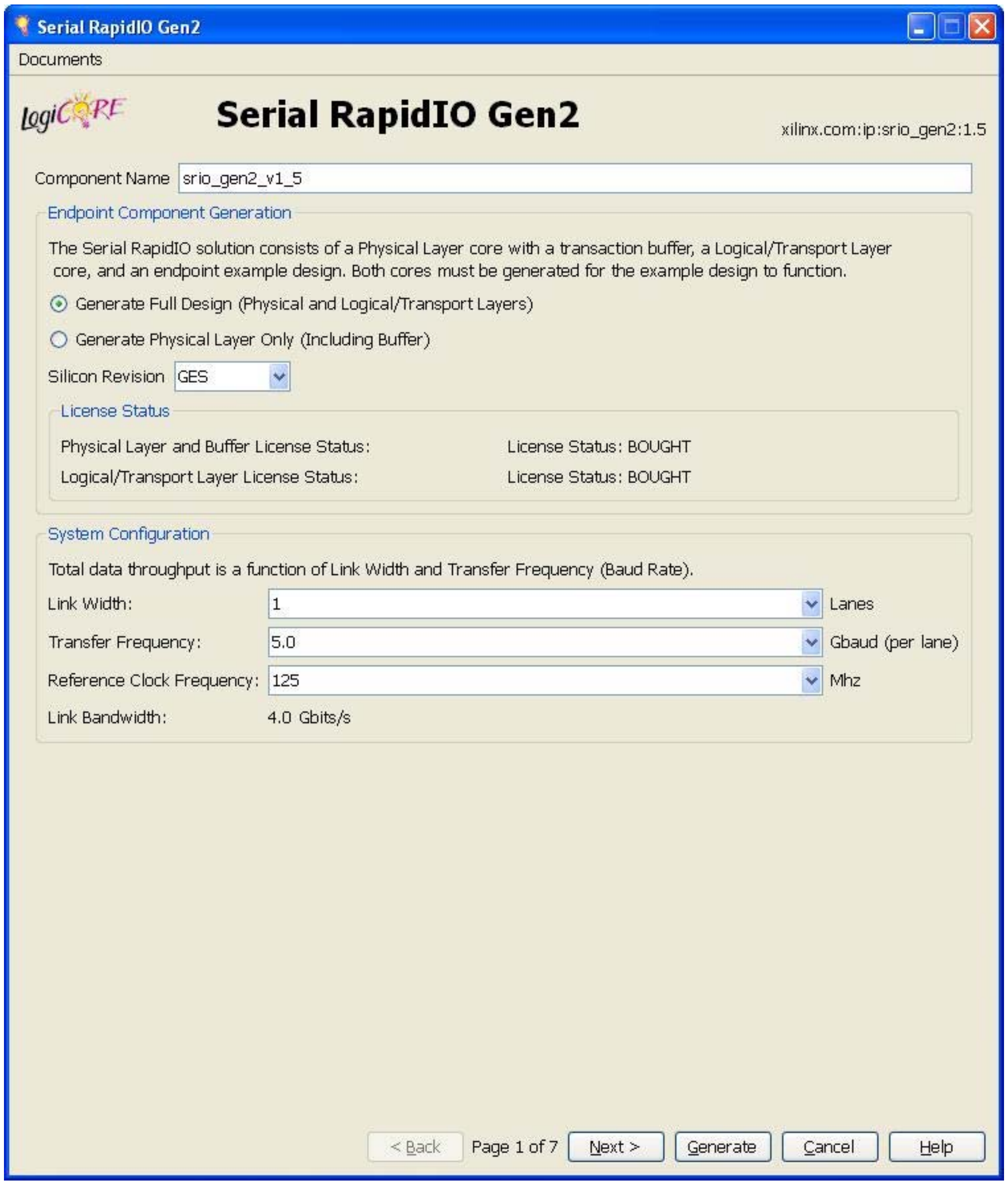
The Serial RapidIO Gen2 top-level wrapper contains the unified core, consisting of the three sub-cores: LOG, BUF, and PHY. The top-level wrapper also calls the GT wrappers and the configuration fabric reference design. The endpoint example design sits above the core wrapper file. It contains synthesizable stimulus generators on each request interface, as well as automatic response generators on all of the response ports.

This chapter describes the GUI options used to generate and customize the sub-cores. When you are generating the SRIO Gen2 Endpoint, they must all be created within the same CORE Generator software project directory to work with the Endpoint Example implementation and simulation scripts.

GUI

Main Configuration Page

Figure 7-1 shows the Serial RapidIO Gen2 CORE Generator software main configuration screen. Descriptions of the GUI options on this screen are provided in the following text.



Serial RapidIO Gen2

xilinx.com:ip:srio_gen2:1.5

Component Name: srio_gen2_v1_5

Endpoint Component Generation

The Serial RapidIO solution consists of a Physical Layer core with a transaction buffer, a Logical/Transport Layer core, and an endpoint example design. Both cores must be generated for the example design to function.

[✓ Generate Full Design (Physical and Logical/Transport Layers)]

Generate Physical Layer Only (Including Buffer)

Silicon Revision: GES

License Status

Physical Layer and Buffer License Status:	License Status: BOUGHT
Logical/Transport Layer License Status:	License Status: BOUGHT

System Configuration

Total data throughput is a function of Link Width and Transfer Frequency (Baud Rate).

Link Width:	1	Lanes
Transfer Frequency:	5.0	Gbaud (per lane)
Reference Clock Frequency:	125	Mhz
Link Bandwidth:	4.0 Gbits/s	

Figure 7-1: **Serial RapidIO Gen2 Main 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_gen2_v1_5.

Core Selection

The LOG generation is optional. Enabling generation provides a customized netlist for the core and associated CORE Generator software files. The Logical Layer is not meant as a stand-alone core; a Physical Layer will be generated as well. Both layers must be generated for the example design to function.

Silicon Revision

The silicon revision is used to provide the correct components in the example design to match the silicon requirements.

Link Width

The link width represents the number of serial lanes per direction that will be generated. The Serial RapidIO Gen2 core can have one, two, or four lanes. The bandwidth of the system increases with the number of lanes.

Transfer Frequency

The transfer frequency represents the per-lane baud rate of the Serial RapidIO core. Each Serial Transceiver will be run at the selected line rate. The bandwidth of the system increases with transfer frequency.

Reference Clock Frequency

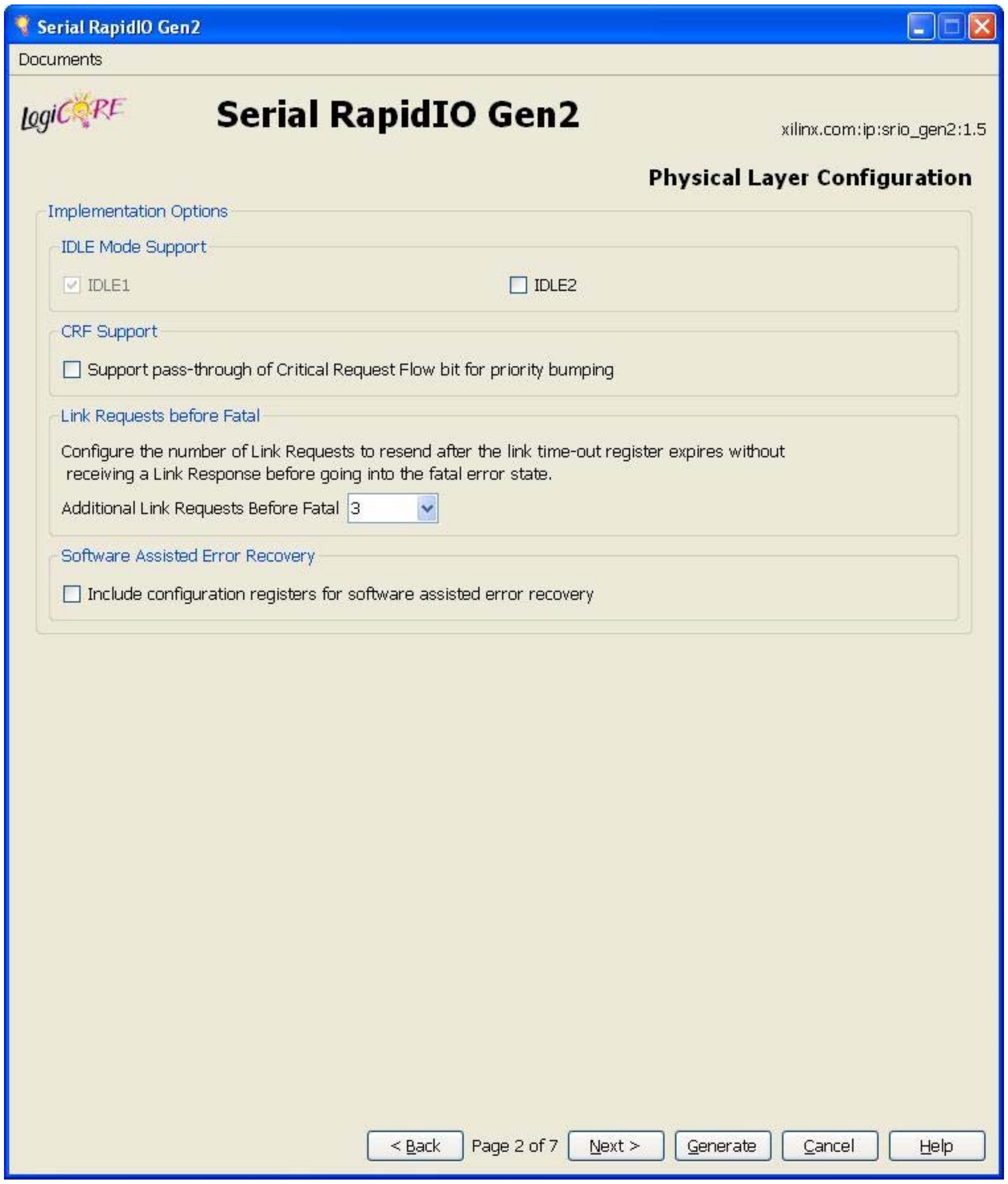
The reference clock frequency is the rate of the clock that will be brought into the FPGA on the dedicated transceiver reference clock pins.

Link Bandwidth

The maximum theoretical bandwidth of the link is calculated on this page of the GUI. This calculation accounts for the 8b/10b encoding overhead, but not the link protocol overhead (loss of bandwidth due to clock correction symbols, control symbols, etc.) or packet header overhead.

Physical Layer Configuration Page

Figure 7-2 shows the Serial RapidIO Gen2 CORE Generator software Physical Layer configuration screen.



Implementation Options

IDLE Mode Support

[✓] IDLE1 [ ] IDLE2

CRF Support

[ ] Support pass-through of Critical Request Flow bit for priority bumping

Link Requests before Fatal

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

Additional Link Requests Before Fatal: 3

Software Assisted Error Recovery

[ ] Include configuration registers for software assisted error recovery

Figure 7-2: **Physical Layer Configuration Screen**

IDLE Mode Support

The supported IDLE modes vary based on the transfer frequency chosen on page 1. IDLE1 operation is only supported for line rates of 5.0 Gbaud or less per lane. Use of the IDLE1 sequence means that the short control symbol format will be used. The IDLE2 sequence was designed for links operating above 5.0 Gbaud per lane. Rates of 6.25 Gbaud only operate with IDLE2 sequence. The IDLE2 sequence provides additional functionality (link width and lane polarity information, plus randomized data for equalizer training for improved data recovery). If both the IDLE1 and IDLE2 sequences are enabled (only possible for 5.0 Gbaud or less), the core will train with the link partner as described in the RapidIO Specification to select an IDLE sequence.

CRF Support

Indicates whether the CRF bit is being used for extended priority mapping.

Link Requests Before Fatal

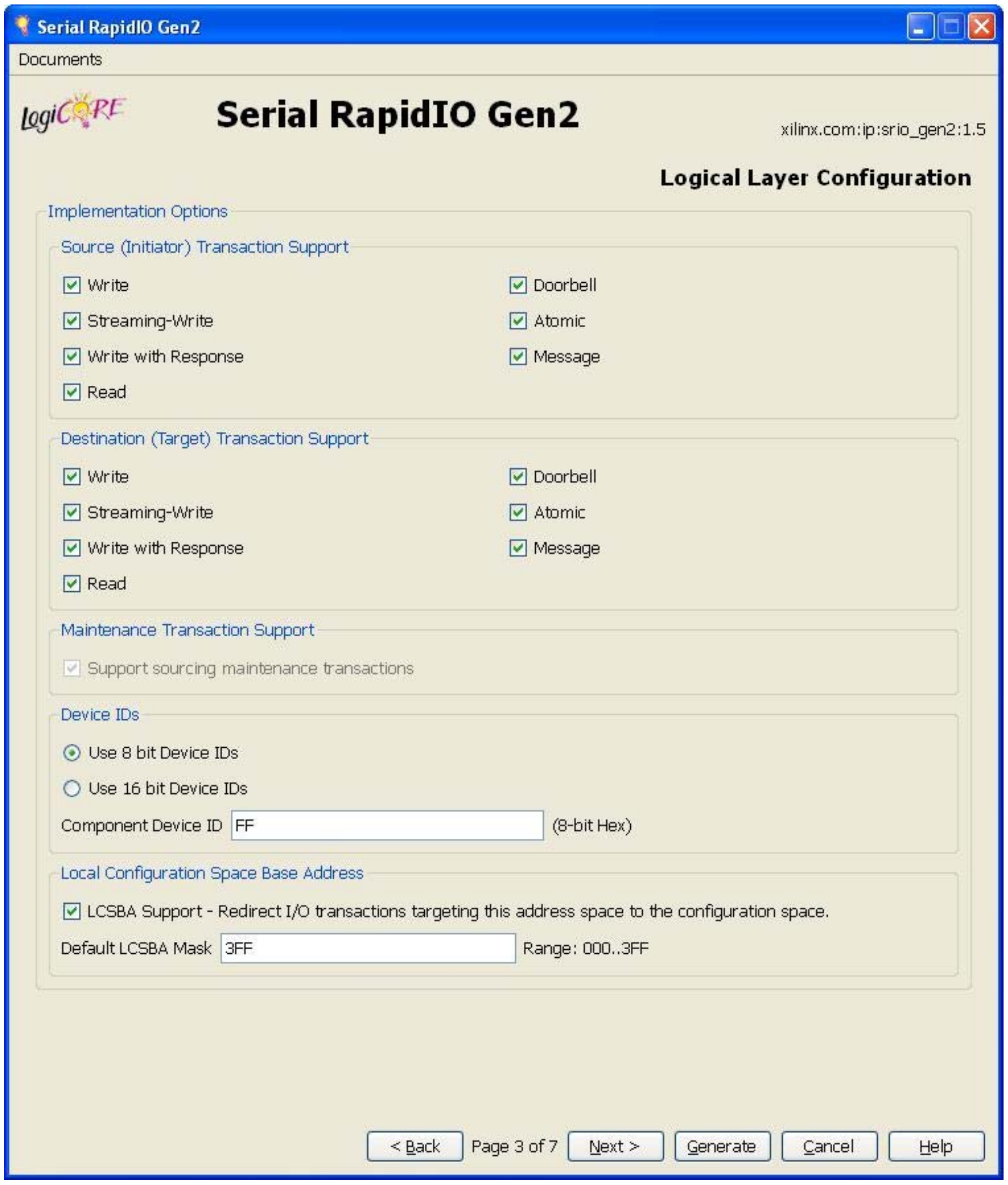
Use the drop-down list to select the number of link requests that are to be sent without receiving a link response prior to transiting to a fatal error state. Allowable options are 0 through 6 and Never Fatal. The default is zero. Xilinx recommends setting this value greater than zero for enhanced error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

Software Assisted Error Recovery

The RapidIO Specification defines three Command and Status Registers specifically for software-assisted error recovery. Specifically the Port n Link Maintenance Request CSR, page 59, Port n Link Maintenance Response CSR, page 59, and the Port n Local ackID CSR, page 60.

Logical Layer Configuration Page

Figure 7-3 shows the Serial RapidIO Gen2 CORE Generator software Logical Layer configuration screen. This page does not exist if the Logical Layer was disabled on the first page of the GUI.


Serial RapidIO Gen2

Documents

Serial RapidIO Gen2

xilinx.com:ip:srio_gen2:1.5

Logical Layer Configuration

Implementation Options

Source (Initiator) Transaction Support

- [✓] Write
- [✓] Streaming-Write
- [✓] Write with Response
- [✓] Read
- [✓] Doorbell
- [✓] Atomic
- [✓] Message

Destination (Target) Transaction Support

- [✓] Write
- [✓] Streaming-Write
- [✓] Write with Response
- [✓] Read
- [✓] Doorbell
- [✓] Atomic
- [✓] Message

Maintenance Transaction Support

- [✓] Support sourcing maintenance transactions

Device IDs

- [✓ Use 8 bit Device IDs]
- Use 16 bit Device IDs

Component Device ID: FF (8-bit Hex)

Local Configuration Space Base Address

- [✓] LCSBA Support - Redirect I/O transactions targeting this address space to the configuration space.

Default LCSBA Mask: 3FF Range: 000..3FF

< Back | Page 3 of 7 | Next > | Generate | Cancel | Help

Figure 7-3: **Logical Layer Configuration Screen**

Source (Initiator) Transaction Support

Select the standard transmit operations supported by the processing element. These selections will affect the way that transactions are routed, and must be made carefully. If the HELLO packet format is being used, the HELLO encoding and decoding logic will only be generated for enabled packet types (and their corresponding responses).

The SRIO Gen2 Endpoint does not support Port-Write transactions.

Destination (Target) Transaction Support

Select the standard receive operations supported by the processing element. These selections will affect the way that transactions are routed, and must be made carefully. If the HELLO format is being used, the HELLO encoding and decoding logic will only be generated for enabled packet types (and their corresponding responses).

The SRIO Gen2 Endpoint does not support Port-Write transactions.

Source Maintenance Support

This option determines whether the core should be capable of sourcing MAINTENANCE requests (including both local and remote requests). This is always enabled.

Device ID Width

The Device ID width of the core should match that of the link partner. Otherwise, transactions may be misinterpreted due to the header shift. Most systems use 8-bit Device IDs, but the SRIO Gen2 Endpoint also provides large system support through this option.

Component Device ID Reset Value

The Component Device ID determines the reset value for the Base Device ID CSR.

Local Configuration Space Base Address (LCSBA) Support

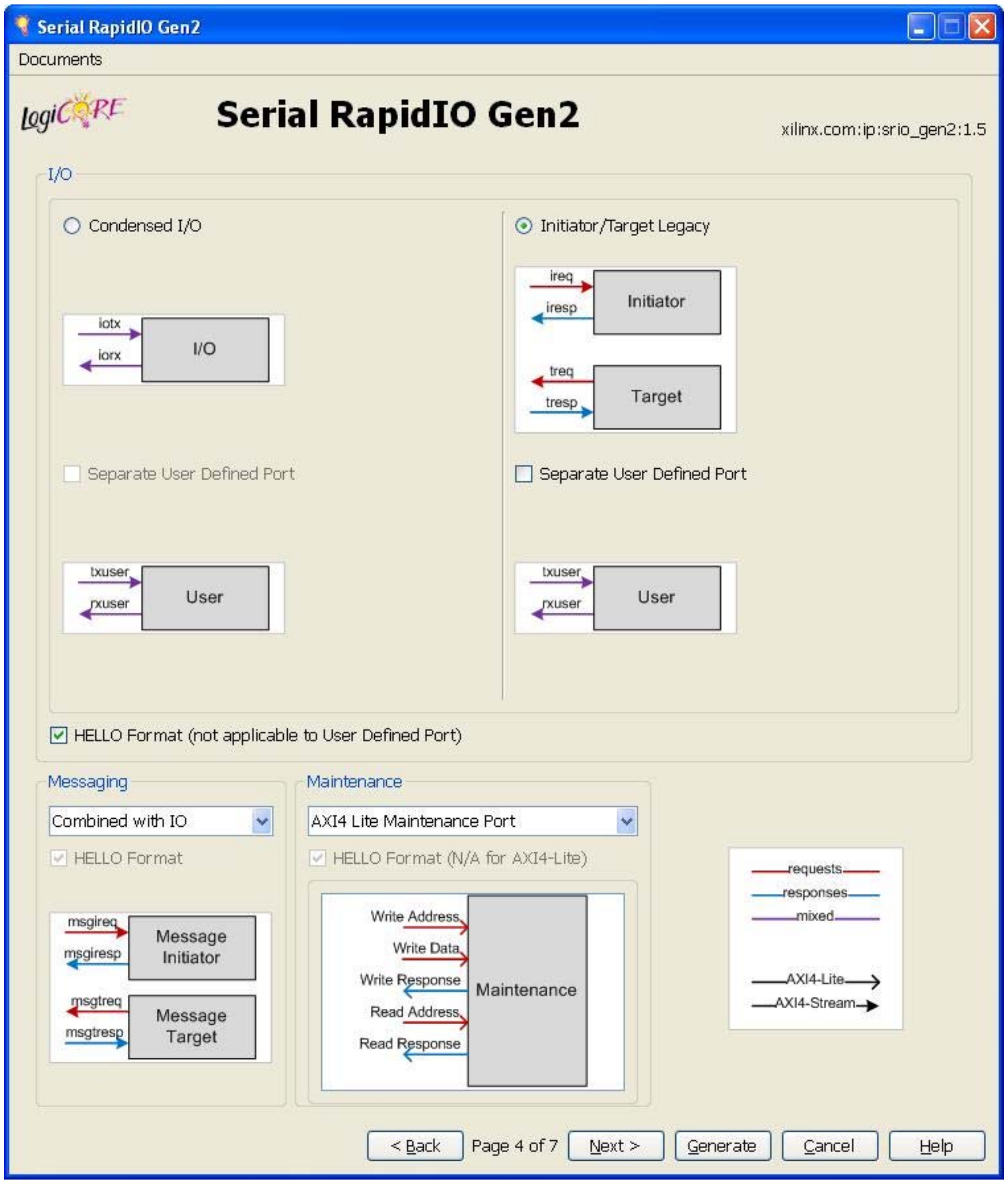
When enabled, the core will check the upper address bits of incoming I/O transactions and route the transactions to the Maintenance port if there is an address match. The specification does not provide a mechanism to disable LCSBA, so the system behavior in this case is undefined. LCSBA support should only be disabled in a closed system. Note also that LCSBA matching will only be performed by the core on HELLO formatted packets (so the I/O format must be set to HELLO and the transaction type must be supported in order for the packet to be rerouted by the core to the Maintenance port). If LCSBA support is required and these restrictions are not met, the transaction will come out the I/O or User-Defined port and the User Application will be responsible for routing it to the configuration space.

Local Configuration Space Base Address (LCSBA) Mask

Reset value for the LCSBA CSR. A 10-bit mask which is compared against the upper address bits of incoming I/O transactions, so that they may be rerouted to the configuration space when they target this address space. The core compares this value to bits [33:24] of the address, which includes the xamsbs field from the SRIO packet.

Logical Layer Interface Selection Page

On this page, the User Interface can be customized to best fit the needs of the User Application. This page does not exist if the Logical Layer was disabled on the first page of the GUI. Figure 7-4 shows the Serial RapidIO Gen2 CORE Generator software Logical Layer Interface Selection screen.



Condensed I/O [✓ Initiator/Target Legacy]

☐ Separate User Defined Port (Condensed I/O, disabled)

☐ Separate User Defined Port

☑ HELLO Format (not applicable to User Defined Port)

Messaging: Combined with IO

☑ HELLO Format (disabled)

Maintenance: AXI4 Lite Maintenance Port

☑ HELLO Format (N/A for AXI4-Lite) (disabled)

Figure 7-4: **Logical Layer Interface Selection Screen**

This page requires the user to choose a packet format for each port on the User Interface. See HELLO Packet Format, page 74 and SRIO Stream Packet Format, page 78 for more information.

I/O Style

The I/O interface can be configured to use one of three port styles.

- **Condensed I/O** - In the Condensed I/O style, there is a single transmit and a single receive AXI4-Stream channel for all I/O traffic. In this mode, the user can elect to also create a separate port for User-Defined transaction types.
- **Initiator/Target** - In the Initiator/Target style, transactions are sorted by the memory space that they are accessing. Initiator transactions consist of outgoing read and write requests (destined for memory at a remote device) and incoming responses. Target transactions are read and write requests which target memory in the local endpoint, plus the responses that the endpoint sends for those transactions. In this mode, the user can select a separate port for User-Defined transaction types.

I/O Format

The I/O port can be configured to use either HELLO or SRIO Stream formatted packets. For typical users, the HELLO format is strongly recommended. SRIO Stream format should only be used if full control is required. See Chapter 3, Designing with the Core for information on the two packet formats. Since User-Defined packets have user-defined header fields, the HELLO logic does not know how to construct/deconstruct them. Therefore, regardless of the I/O port style, packets on the User-Defined port use the SRIO Stream format.

Messaging Style

If MESSAGE transactions were enabled on the previous page, there are three options for the Messaging port interface style:

- **Combined with IO** - If MESSAGEs are combined with the I/O port, they will use the same interfaces that an I/O write transaction would use.
- **Separate Messaging Port** - In this style, MESSAGE transactions will have their own Initiator/Target style of port. This allows the User Application to process MESSAGE transactions independently from I/O operations.
- **Combined with User-Defined Port** - In this style, MESSAGE transactions are combined with User-Defined packets.

Messaging Format

The Messaging port (if a separate one exists) can be configured to use either HELLO or SRIO Stream formatted packets. For typical users, the HELLO format is strongly recommended. SRIO Stream format should only be used if full control is required. See Chapter 3, Designing with the Core for information on the two packet formats. If the port is combined with another port, MESSAGE transactions on the log interface must use the same format as that port.

Maintenance Style

For complete handling of MAINTENANCE requests received from a remote device, the SRIO Gen2 Endpoint offers the option of an AXI4-Lite Maintenance port. If the user cannot source MAINTENANCE transactions (disabled on the previous page), there will be no user Maintenance Interface, but the LOG will have an AXI4-Lite Master connection to the Configuration Fabric Reference design.

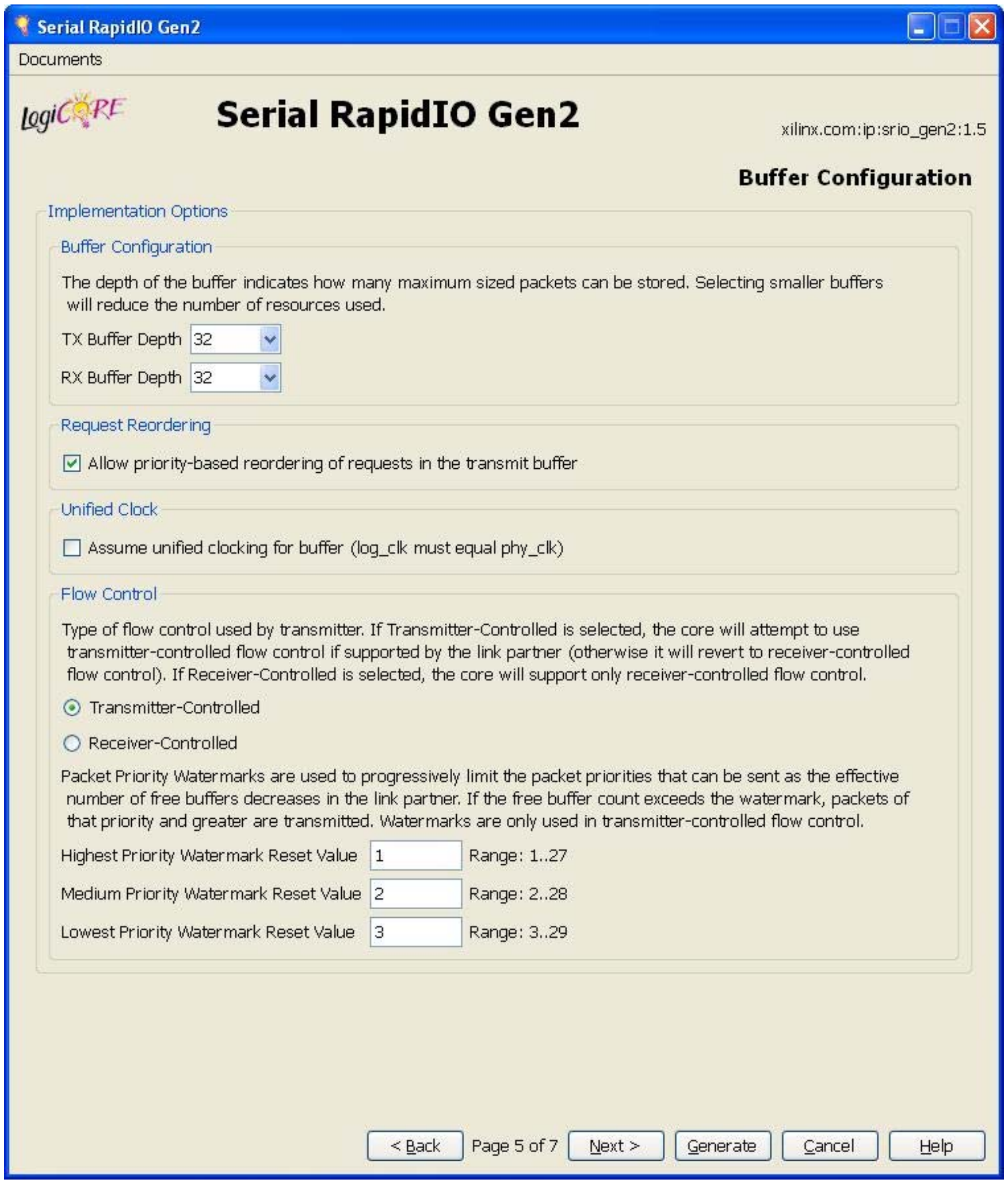
If special functionality is needed, the Maintenance port can be configured in a similar way to the other User Ports with AXI4-Stream channels for direct access to MAINTENANCE packets. Maintenance traffic can also be routed to the User-Defined port if so required, by selecting Combined with User-Defined port from the drop-down menu. That option will only exist if the User-Defined port is enabled.

Maintenance Format

If the direct access to AXI4-Stream port style is chosen, the Maintenance port can be configured to use either HELLO or SRIO Stream formatted packets.

Buffer Configuration Page

Figure 7-5 shows the Serial RapidIO Gen2 CORE Generator software Buffer Configuration screen.



The depth of the buffer indicates how many maximum sized packets can be stored. Selecting smaller buffers will reduce the number of resources used.

TX Buffer Depth 32

RX Buffer Depth 32

Request Reordering

[✓] Allow priority-based reordering of requests in the transmit buffer

Unified Clock

[ ] Assume unified clocking for buffer (log_clk must equal phy_clk)

Flow Control

Type of flow control used by transmitter. If Transmitter-Controlled is selected, the core will attempt to use transmitter-controlled flow control if supported by the link partner (otherwise it will revert to receiver-controlled flow control). If Receiver-Controlled is selected, the core will support only receiver-controlled flow control.

[✓ Transmitter-Controlled] Receiver-Controlled

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. Watermarks are only used in transmitter-controlled flow control.

Highest Priority Watermark Reset Value	1	Range: 1..27
Medium Priority Watermark Reset Value	2	Range: 2..28
Lowest Priority Watermark Reset Value	3	Range: 3..29

Figure 7-5: **Buffer Configuration Screen**

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 (primarily Block RAMs and LUTs), whereas maximum buffer depth yields maximum throughput.

Request Reordering

Selecting this option allows the transmit buffer to reorder request packets. In this case, higher priority requests go out before lower priority requests. If unchecked, request packets exit in the order they were received. Regardless of this option, responses go out before requests.

Unified Clock

If the user design uses the same clock for `log_clk` (primary user clock) and `phy_clk`, check this box. Selecting this option significantly reduces both latency and resource utilization.

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. If the watermark value is too high for a smaller buffer, this could cause the core to lock up. If there is concern about the size of the buffer, a safe value for the watermarks would be WM0=3, WM1=2, WM2=1.

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:

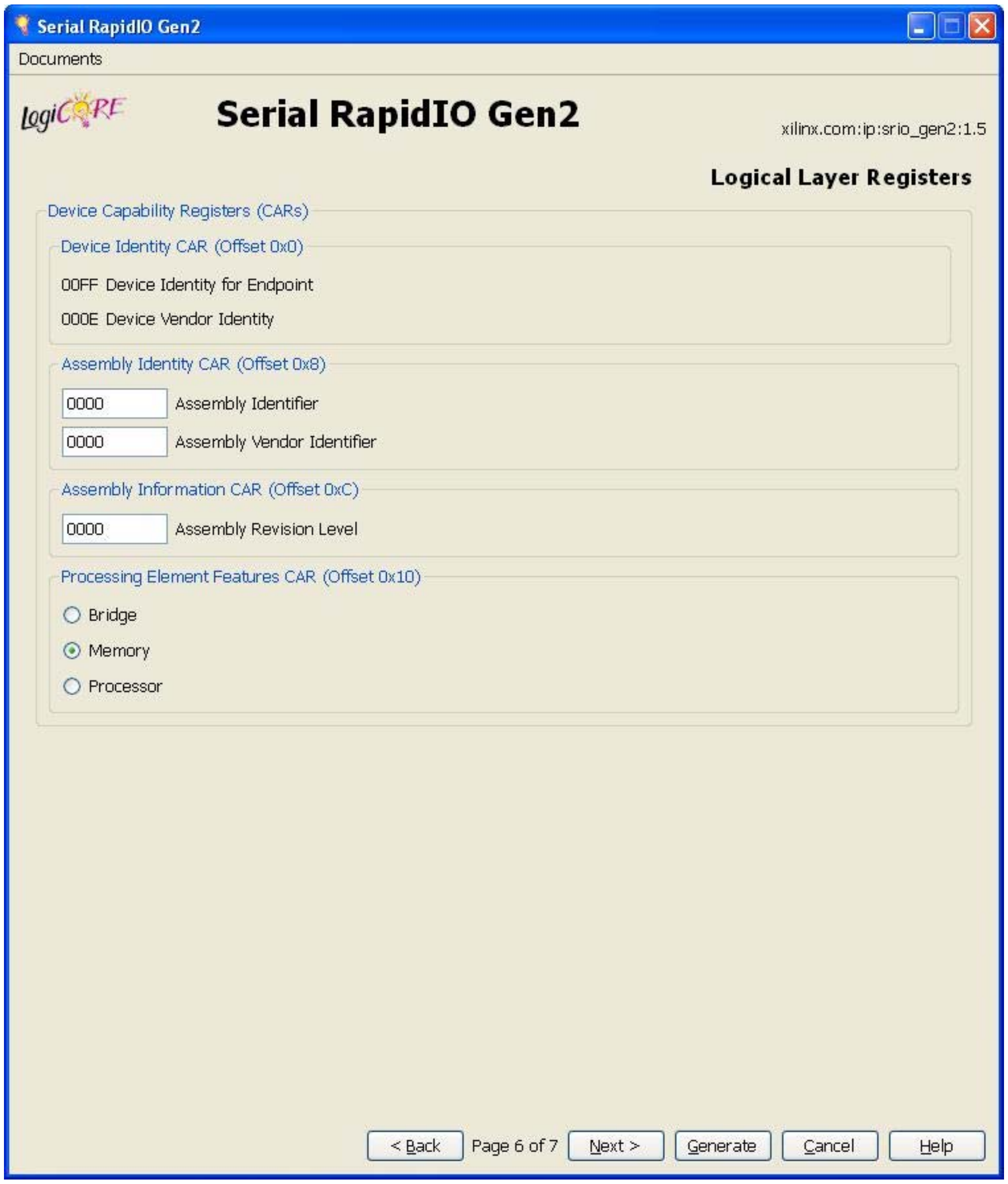
  $0 < WM2 < WM1 < WM0$

- **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 Registers Page

Figure 7-6 shows the Serial RapidIO Gen2 CORE Generator software Logical Layer Registers screen.



Device Capability Registers (CARs)

Device Identity CAR (Offset 0x0)

00FF Device Identity for Endpoint

000E Device Vendor Identity

Assembly Identity CAR (Offset 0x8)

0000	Assembly Identifier
0000	Assembly Vendor Identifier

Assembly Information CAR (Offset 0xC)

0000	Assembly Revision Level

Processing Element Features CAR (Offset 0x10)

Bridge [✓ Memory] Processor

Figure 7-6: **Logical Layer Registers Screen**

The Logical Layer Configuration Registers hold capability and identity information for the device, as well as core status and control information.

Device Identity CAR

The Device Identity CAR stores information about a RapidIO device. This register holds Xilinx-assigned fields and is not modifiable.

Assembly Identity CAR

The Assembly Identity CAR stores information about a RapidIO device subsystem creator. The Assembly Identifier and Assembly Vendor Identifier fields of the register can be set by the user during core generation to uniquely identify the Endpoint. These values do not affect core functionality.

Assembly Information CAR

The Assembly Information CAR stores information about a RapidIO device subsystem revision. This is set during core generation and does not affect core functionality.

Processing Element Features CAR

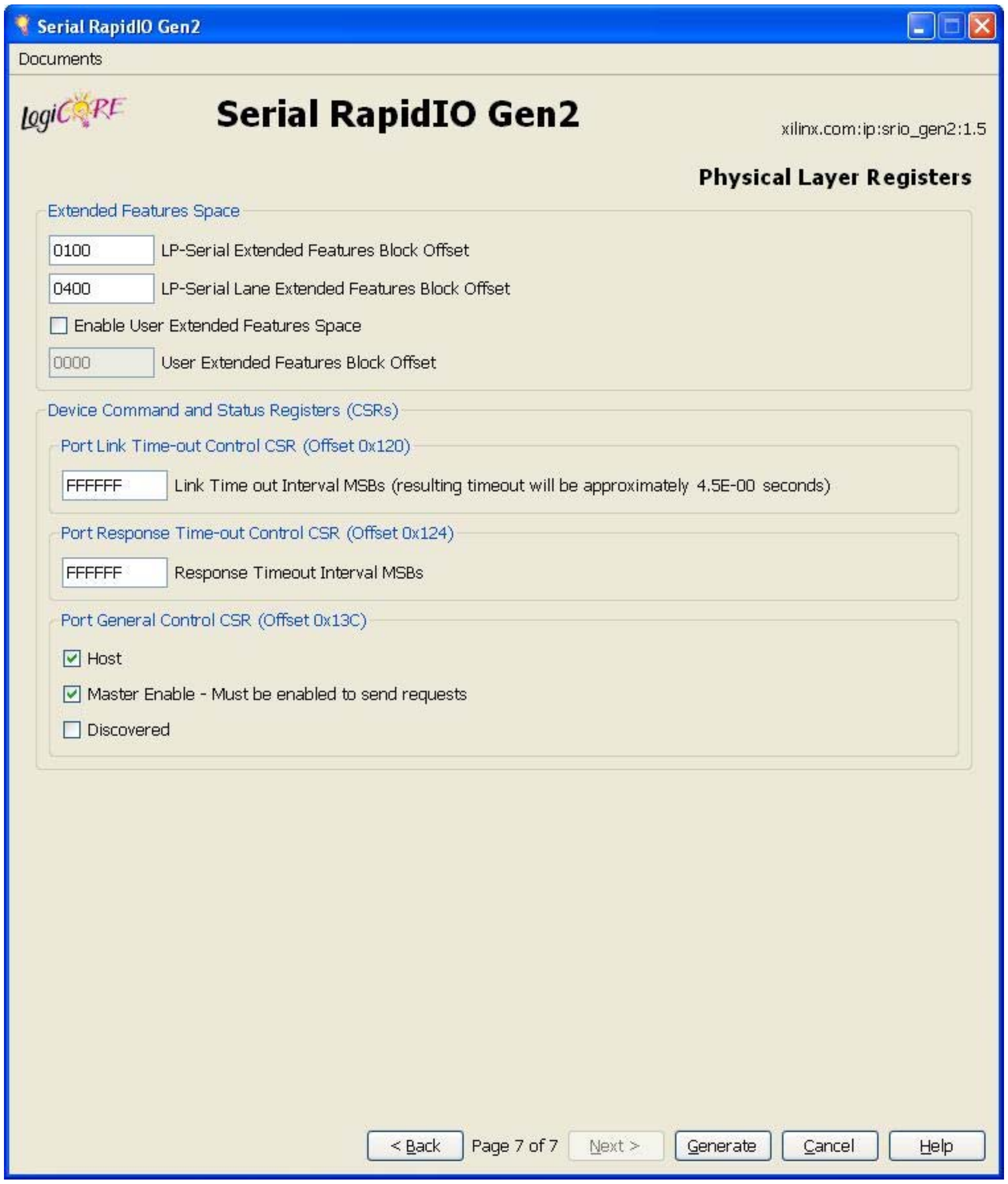
Selects the major functionality provided by the processing element. Allowable options are:

- Bridge
- Memory
- Processor

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

Physical Layer Registers Page

Figure 7-7 shows the Serial RapidIO Gen2 CORE Generator software Physical Layer Registers screen.



Extended Features Space

0100 LP-Serial Extended Features Block Offset

0400 LP-Serial Lane Extended Features Block Offset

☐ Enable User Extended Features Space

0000 User Extended Features Block Offset

Device Command and Status Registers (CSRs)

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

FFFFFF Link Time out Interval MSBs (resulting timeout will be approximately 4.5E-00 seconds)

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

FFFFFF Response Timeout Interval MSBs

Port General Control CSR (Offset 0x13C)

☑ Host

☑ Master Enable - Must be enabled to send requests

☐ Discovered

Figure 7-7: **Physical Layer Registers Screen**

The Physical Layer Configuration Registers hold core status and control information.

Extended Features Space

The Physical Layer Registers are stored in register blocks within the Extended Features (EF) address space. The Serial RapidIO Gen2 Physical Layer implements two Extended Features blocks - the LP-Serial EF block and the LP-Serial Lane EF block. The SRIO Gen2 Endpoint reserves 0x100 bytes off address space for the LP-Serial EF block, and 0x400 bytes of address space for the LP-Serial Lane EF block.

The block offset for the LP-Serial Features block must be smaller than the offset of the LP-Serial Lane EF block. Each block offset must be an integer multiple of the number of bytes reserved for the block.

The user may implement one or more EF blocks within the Extended Features address space as well. The User Application would have to create these registers somewhere and modify the Configuration Fabric reference design in order to accommodate the additional register block. If a User EF is being implemented, it should be enabled in the GUI and the block offset of the first EF block should be entered. That offset must be greater than the LP-Serial Lane EF block offset plus the 0x400 byte size that was reserved for that block. The User EF block offset must be a multiple of 0x100 and within the Extended Features address space.

See Extended Features Space, page 57 for more information.

Port Link Timeout Control CSR Reset Value

Timeout 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 specification is followed. The max timeout scales linearly with the value in this CSR. A max time-out value of FF_FFFFh corresponds to a time-out length of approximately 4.5 seconds, but the max value is only accurate to within +/- 33%.

Port Response Timeout CSR Reset Value

Timeout 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 between 3 and 6 seconds. Implementation of the response timeout is left to the user.

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.

Note: If Master Enable is unchecked, the device will not be allowed to initiate requests until the bit is written with a 1.

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.

Output Generation

See Directory and File Contents in Chapter 9 for details about files and directories created when generating the core.

Constraining the Core

This chapter defines the constraint requirements of the SRIO Gen2 Endpoint example design. An example user constraints file (UCF) is provided with the example design, which implements the constraints defined in this chapter.

For each device family, the constraints file and implementation scripts will target one specific device (for example, when a Virtex®-6 device is selected, an XC6V240T-FF1136 device will be targeted).

The example design and UCF can be used (retargeted) for other devices within the FPGA family. Information is provided in this chapter to indicate which constraints to modify for those cases.

The constraints defined in this section are implemented in the `srio_example_top.ucf` file for the SRIO Gen2 Endpoint example design. See Chapter 9, Detailed Example Design for information about the location of the UCF.

Device, Package, and Speed Grade Selections

The SRIO Gen2 Endpoint can be implemented in Virtex-7, Kintex™-7, and Virtex-6 devices with the following attributes:

- Large enough to accommodate the SRIO Gen2 Endpoint design
- Fast enough speed grade to meet the requirements listed in the *Serial RapidIO Gen2 Product Specification*
- Transceivers capable of running at the core's configured line rate

Clock Frequencies

These constraints specify the frequency, duty-cycle, input jitter, and priority of the clock signals in the design. If the reference clock module is not used, these may need to be adjusted. For example, if log_clk is from a different source in the design, the period should be updated accordingly. The following are example constraints for a 5.0 4x configuration.

```
NET “*gt_clk” TNM_NET = gt_clk;
TIMESPEC TS_gt_clk = PERIOD “gt_clk” 4 ns HIGH 50 %;

NET “*phy_clk” TNM_NET = phy_clk;
TIMESPEC TS_phy_clk = PERIOD “phy_clk” “TS_gt_clk” * 1 HIGH 50 %;

NET “*gt_pcs_clk” TNM_NET = gt_pcs_clk;
TIMESPEC TS_gt_pcs_clk = PERIOD “gt_pcs_clk” “TS_gt_clk” * 2 HIGH 50 %;

NET “*log_clk” TNM_NET = log_clk;
TIMESPEC TS_log_clk = PERIOD “log_clk” 4 ns HIGH 50 %;
```

For other configurations, the clock periods are adjusted accordingly. Note that the phy_clk constraint is derived from the gt_clk constraint. This is to reflect the phase relationship that must exist between these clocks.

Clock Management

Advanced timing closure techniques may be needed in order to meet timing for 5.0 Gbaud or greater operation.

For example, an area group may facilitate placement so that routing delays are minimized. Generally, the area group should be large enough to contain enough logic resources for the core, but small enough to have an impact on timing. Optimal size will vary with core configuration.

Example syntax:

```
INST '*' AREA_GROUP=AG_SRIO;
AREA_GROUP 'AG_SRIO' RANGE=SLICE_X92Y0:SLICE_X161Y79;
```

Generic area groups specific to each device will be provided by default in the UCF for 5.0 Gbaud or faster configurations. See UG612, *Xilinx Timing Constraints User Guide* for more information on area group constraints.

Additional steps that may be necessary:

- Location constraints for block RAMs.
- Using different cost tables (-t option for MAP and PAR) and/or SmartXplorer; see UG628, *Xilinx Command Line Tools User Guide* for more information.

Clock Placement

No clock placement constraints are required.

Banking

No banking constraints are required.

Transceiver Placement

The provided UCF uses placement constraints to specify the serial transceivers that will be used when the core is implemented. These can be moved around, but all lanes within a given core should be placed in adjacent transceivers for best timing results. The following constraints are provided by default with 4x Virtex-7 FPGA cores.

```
INST “*gtx_wrapper_i/gtx0_gtx_wrapper_i/gtxe1_i” LOC = GTXE1_X0Y11;
INST “*gtx_wrapper_i/gtx1_gtx_wrapper_i/gtxe1_i” LOC = GTXE1_X0Y10;
INST “*gtx_wrapper_i/gtx2_gtx_wrapper_i/gtxe1_i” LOC = GTXE1_X0Y9;
INST “*gtx_wrapper_i/gtx3_gtx_wrapper_i/gtxe1_i” LOC = GTXE1_X0Y8;
```

I/O Standard and Placement

The UCF constrains the I/O placement and I/O standard. These can be moved to any free location that also meets I/O placement and banking rules. The following is an example of the constraints applied to the Kintex-7 FPGA cores.

```
NET "sys_rst"  IOSTANDARD = LVCMOS18 | LOC = "K18" | PULLUP;
NET "sys_clkp" IOSTANDARD = LVCMOS18 | LOC = "G8";
NET "sys_clkn" IOSTANDARD = LVCMOS18 | LOC = "G7";
```

Simulation Constraints

The SRIO Gen2 Endpoint is provided with a simulation model, `<srio_component_name>.v` to allow using the design in a simulation environment. Note that the SRIO PHY core contains a `sim_train_en` port which should be asserted during simulation, allowing the SRIO link to initialize and synchronize in a reduced amount of time. This is achieved by decreasing the time for which the output transceiver must remain silent. Because of this, the silence timers for the core have been configured with smaller values to minimize simulation times.

If targeting Virtex-6 devices, the `SIM_GTXRESET_SPEEDUP` attribute of the Virtex-6 transceiver is set by default to shorten the time it takes to finish the `GTXRESET` sequence during simulation. The `GTXRESET` sequence is not shortened when implemented in hardware. If the full `GTXRESET` sequence is required (160 µs versus 300 ns), the `SIM_GTXRESET_SPEEDUP` attribute can be set to FALSE in the transceiver wrapper. The equivalent attribute for the 7 series devices is `WRAPPER_SIM_GTRESET_SPEEDUP`.

Detailed Example Design

This chapter introduces the SRIO Gen2 Endpoint reference design that is included with the SRIO Gen2 Endpoint. The reference design demonstrates how to generate the SRIO Gen2 Endpoint, including the LOG, BUF, and PHY provided through the SRIO Gen2 Endpoint top-level wrapper. In addition, the reference design illustrates using the default options in the SRIO Gen2 Endpoint example design.

Overview

The SRIO Gen2 Endpoint example design connects two instances of the core together. Each core instance can be configured to send supported packet types, check for receive packet mismatches, and report in the simulation transcript with details about the link traffic. Both the simulation host and the primary core being tested use the SRIO Gen2 Endpoint reference design (consisting of a LOG, BUF, and PHY provided through the SRIO Gen2 Endpoint top-level wrapper) and the Configuration Fabric reference design.

Generating the Cores

Before you can use the SRIO Gen2 Endpoint, the PHY and LOG 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 7 series and Virtex-6 devices.

Note: If an unsupported silicon family is selected, the SRIO Gen2 Endpoint 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.

Generation Options:

b. Select Verilog for the Design Entry.

Note: A VHDL reference design is not supported.

c. For Vendor, select Other (for XST).

Advanced Options:

Leave the advanced options at their default values.

To generate a Full SRIO Gen2 Endpoint with default values:

1. After creating the project, locate the directory containing SRIO Gen2 Endpoint in the list of available cores. It should be located under Standard Bus Interfaces > RapidIO > Serial RapidIO Gen2.
2. Double-click the Serial RapidIO Gen2 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 SRIO Gen2 Endpoint customization screen appears.
4. Accept the default values on the screen, and then click Finish.

By default, the cores 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 SRIO Gen2 Endpoint example design. Detailed information about the files and directories delivered with the SRIO Gen2 Endpoint core is provided in Directory and File Contents.

Directory and File Contents

The complete LogiCORE™ IP SRIO Gen2 Endpoint consists of the following components:

- SRIO Gen2 Unified top-level containing:
    - Serial RapidIO Gen2 Physical Layer (PHY)
    - Serial RapidIO Gen2 Buffer Design (BUF)
    - Serial RapidIO Gen2 Logical Layer (LOG)
- SRIO Gen2 Endpoint Example Design

The SRIO Gen2 Endpoint includes a configuration fabric reference design, example clock and reset modules, and an example user design.

The SRIO Gen2 Endpoint 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 three sub-cores (PHY, BUF, and LOG) 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 SRIO Gen2 Endpoint top-level wrapper is <srio_component_name>.

CORE Generator Software Project

<project_dir>
Top-level project directory for the CORE Generator software and CORE Generator output, such as the netlist and synthesis model.

SRIO Gen2 Endpoint

<project_dir>/<srio_component name>

<srio_component name>/doc
Contains the SRIO Gen2 Endpoint documentation and release notes text file and html file.

<srio_component_name>/example_design
Contains the source files necessary to create the SRIO Gen2 Endpoint example design.

<srio_component_name>/example_design/cfg_fabric
Contains the source files for the configuration fabric component.

<srio_component_name>/implement
Contains the supporting files for synthesis and implementation of the SRIO Gen2 Endpoint example design.

<srio_component_name>/implement/synthesis
Contains the files needed to synthesize the example design with XST.

<srio_component_name>/simulation
Currently empty and used to maintain the hierarchy structure from the previous generation of the core.

<srio_component_name>/simulation/functional
Contains the scripts and define files for simulating the SRIO Gen2 Endpoint example design in ModelSim.

SRIO Gen2 Endpoint

The following tables contain the files and their descriptions specific to the SRIO Gen2 Endpoint.

<project_dir>

The following table contains the project directory files for the SRIO Gen2 Endpoint top-level wrapper and the SRIO Gen2 Endpoint.

Table 9-1: **Project Directory**

Name	Description
<project_dir>	
<srio_component_name>.xco	Log file from CORE Generator software describing which options were used to generate the SRIO Gen2 Endpoint 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 SRIO Gen2 Endpoint were generated in the CORE Generator software.
<srio_component_name>.veo	The HDL template for the SRIO Gen2 Endpoint top-level wrapper.
<srio_component_name>.ngc	The netlist for the SRIO Gen2 Endpoint top-level wrapper.
<srio_component_name>.v	The structural simulation model for the SRIO Gen2 Endpoint top-level. It is used to support the functional simulation of the core in the RapidIO Gen2 Endpoint example design.
<srio_component_name>_synth.v	The synthesis model of the SRIO Gen2 Endpoint top-level wrapper.
<srio_component_name>.gise	Used by the software tools to track and manage the core in the context of ISE Design Suite. It is not necessary to interact with this file.
<srio_component_name>.xise	Used by the software tools to track and manage the core in the context of ISE Design Suite. It is not necessary to interact with this file.

Back To Top

<project_dir>/<srio_component name>

This directory contains the directories for the SRIO Gen2 Endpoint example design and the implementation and simulation examples.

<srio_component name>/doc

This directory contains the documentation that is included with the core.

Table 9-2: **Doc Directory**

Name	Description
<project_dir>/<srio_component_name>/doc	
pg007_srio_gen2.pdf	LogiCORE IP Serial RapidIO Gen2 Endpoint Product Guide
srio_gen2_<version number>_vinfo.html	Readme HTML file for the current version of the core.
srio_gen2_<version number>_readme.txt	Readme TXT file for the current version of the core.

Back To Top

<srio_component_name>/example_design

This directory and its subdirectories contain all the source files, aside from the SRIO Gen2 core netlist, to create the SRIO Gen2 Endpoint example design. They include the reference design clocking module, reset module, example user design, and user constraints file.

Table 9-3: **Example Design Directory**

Name	Description
<project_dir>/<srio_component_name>/example_design	
srio_example_top.ucf	The user constraints file (UCF) for the RapidIO Gen2 Endpoint example design.
srio_example_top.v	The top-level HDL file for the RapidIO Gen2 Endpoint example design.
srio_clk.v	HDL clock module for generation of core clocks.
srio_wrapper.v	HDL wrapper file that instantiates the SRIO Gen2 Endpoint top-level wrapper, GT wrapper, and configuration fabric reference design.
srio_rst.v	Example reset module that generates and sequences the necessary resets from a `sys_rst` input.
srio_gt_wrapper_<fam>_<width>.v	Transceiver wrapper file.
gt_wrapper.v	GT wrapper from 7 series Transceiver Wizard for 7 series FPGA configurations.
gt_wrapper_gt.v	GT wrapper from 7 series Transceiver Wizard for 7 series FPGA configurations.
gt_wrapper_ver.xco	XCO file used to generate GT wrapper files from the 7 Series Transceiver Wizard for 7 series FPGA configurations.

Table 9-3: **Example Design Directory *(Cont'd)***

Name	Description
gtx_wrapper.v	GTX wrapper from GTX Wizard for Virtex-6 FPGA configurations.
gtx_wrapper_gtx.v	GTX wrapper from GTX Wizard for Virtex-6 FPGA configurations.
gtx_wrapper_ver.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_sim.v	The srio_sim module instantiates two cores and connects them together for simulation. It also contains a timeout and mechanisms to report on whether the test passed or failed.
srio_dut.v	This module instantiates srio_wrapper and the clocking and reset modules.
srio_request_gen.v	This module generates request transactions to be sent to the link partner. It is parameterized with transaction types so that only transactions appropriate to the port and supported by the core will be sent.
instruction_list.v	This file is included into srio_request_gen and consists of a list of transactions that could be sent.
srio_response_gen.v	This file contains logic which creates responses to incoming transactions (if they are response-generating packet types).
srio_condensed_gen.v	This module only exists when the I/O port is configured in Condensed I/O mode. It allows the srio_request_gen module to issue transactions in Condensed I/O mode.
srio_quick_start.v	This module connects to the Maintenance port and issues transactions as part of the example design.
maintenance_list.v	Included into srio_quick_start, this file contains the maintenance instructions that will be issued by the example design.
srio_report.v	This file displays when packets are transmitted and/or received.
srio_statistics.v	This file collects statistics from the core and delivers information through a register interface, accessible through ChipScope analyzer or a user design.

Back To Top

<srio_component_name>/example_design/cfg_fabric

This directory contains the source files for the Configuration Fabric reference design. The configuration fabric serves as an AXI4-Lite interconnect that forwards packets to the appropriate core's configuration register port based upon the configuration offset of the transaction. The source code contains comments to help explain the functionality of the code. It is not required to use this configuration fabric in the design; however, some type of interconnect will be needed.

Table 9-4: **Configuration Fabric Directory**

Name	Description
`<project_dir>/<srio_component_name>/example_design/cfg_fabric`	
cfg_fabric.v	File containing the configuration fabric reference design.

Back To Top

<srio_component_name>/implement

This directory contains the support files necessary for synthesis and implementation of the SRIO Gen2 Endpoint example design with the Xilinx tools.

Table 9-5: **Implementation Directory**

Name	Description
<project_dir>/<srio_component_name>/implement	
implement.sh	A Linux shell script that processes the example design through the Xilinx tool flow.
implement.bat	A Windows script that process the example design through the Xilinx tool flow.

Back To Top

<srio_component_name>/implement/synthesis

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

Table 9-6: **Synthesis Directory**

Name	Description
<project_dir>/<srio_component_name>/implement/synthesis	
srio_example_top.prj	The XST project file for the example design; it lists all of the source files to be synthesized.
srio_example_top.scr	The XST script file for the example design that is used to synthesize the core, called from the implement script described previously.
srio_example_top.lso	The XST options file contains the options that will be used by XST to synthesize the core.

Back To Top

<srio_component_name>/simulation

This directory contains the functional subdirectory.

<srio_component_name>/simulation/functional

This directory contains the scripts and define files for simulating the SRIO Gen2 Endpoint example design in ModelSim.

Table 9-7: **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.
wave.do	A ModelSim macro file that displays helpful waveform signals.

Back To Top

Example Design

The srio_example_top module instantiates all components of the core and example design that are needed to implement the design in hardware, as shown in Figure 9-1. This includes the clock and reset modules, the configuration fabric, and stimulus generators to create transactions.

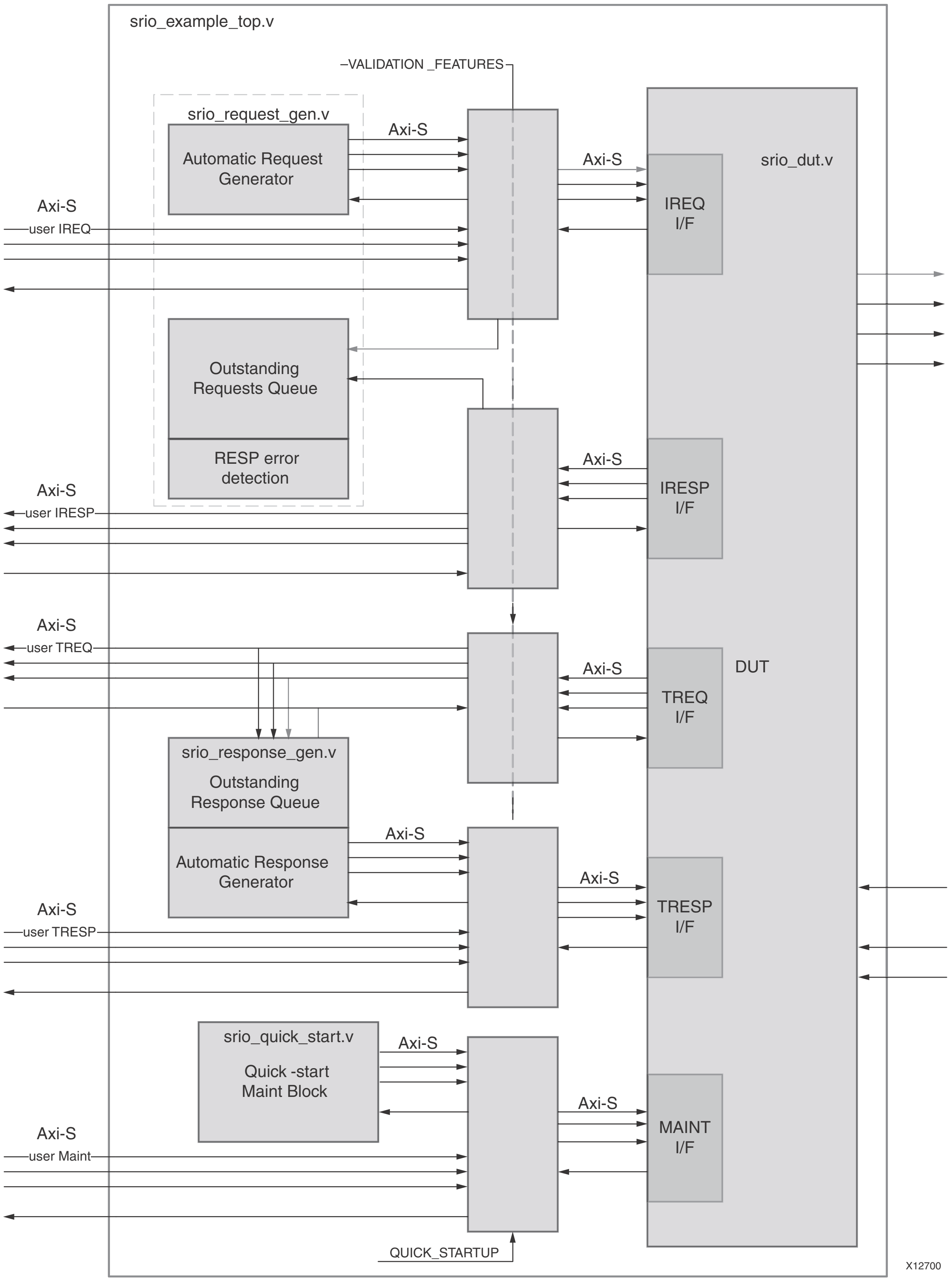


Figure 9-1: **Example Design Top Layer**

The srio_quick_start module, which is instantiated in srio_example_top, connects to the Maintenance port and generates maintenance transactions. The user can add, modify, or remove transactions by editing the maintenance_list file. The srio_request_gen module, which is instantiated within the srio_example_top, is where I/O (and message) transactions are generated. The transaction types to be generated are passed down through parameters so that only transactions supported on the connected port are produced. This file also stores the expected response for any response-generating-requests that it creates, and compares received responses to the expected value. The srio_response_gen module, also instantiated in srio_example_top, generates responses to requests received from the link partner.

User vs. Automatic Stimulus

There are two ways IO stimulus can be generated: automatically by the example design or externally by the user through the initiator and target request/response interfaces, as shown in Figure 9-1. There are also two ways the maintenance transactions can be generated: automatically by the example design or externally by the user through the maintenance interface, as shown in Figure 9-1. In Figure 9-2, the maintenance interface is configured as an AXI4-Stream interface, but this interface can also be configured as an AXI4-Lite interface if needed.

By default, the example design is configured to generate stimulus automatically (for both IO and maintenance transactions) and bypass the external interfaces. This default is achieved in the srio_example_top module by setting `VALIDATION_FEATURES=1` and `QUICK_STARTUP=1`. In addition, it is necessary to comment out the external interfaces to save pins and improve timing.

To use the external interfaces for IO generation, set the parameter VALIDATION_FEATURES=0 and uncomment the external interfaces (`axis_ireq_*`, `axis_tresp_*`, `axis_iresp_*`, `axis_treq_*`, `axis_iotx_*`, and `axis_iorx_*`) in srio_example_top. To use the external interfaces for maintenance transactions, set the parameter `QUICK_STARTUP=0` and uncomment the external interface (`axis_maintr_*`) in the srio_example_top module. srio_example_top contains comments to aid in this process, as shown in Figure 9-2.

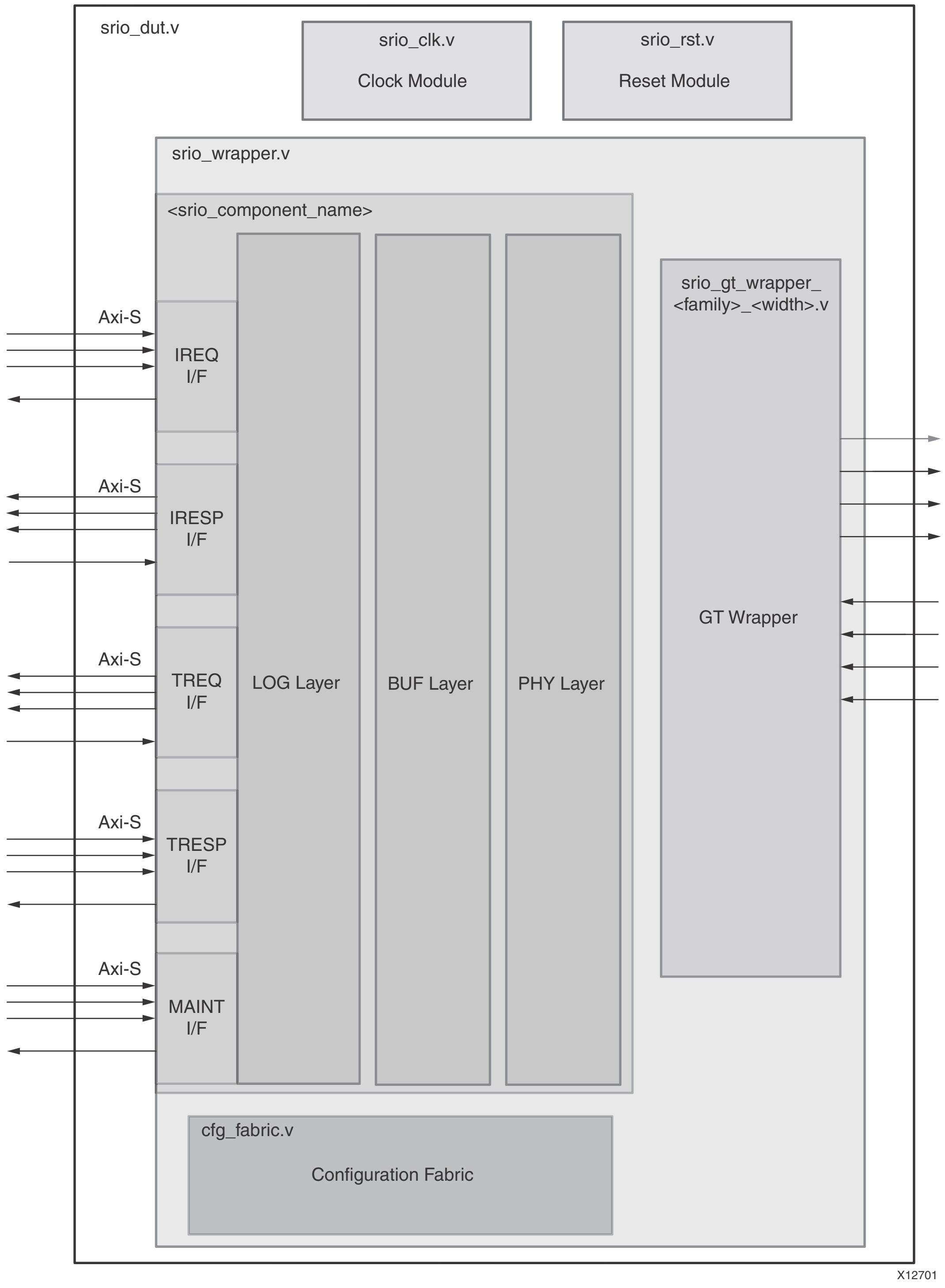


Figure 9-2: **RTL Layer Block Diagram**

Automatic Stimulus Generation

By default, the example design generates stimulus automatically for both the IO and maintenance transactions. The maintenance transactions are generated in the srio_quick_start module, and the transactions are specifically listed out in the `maintenance_list.v` file.

These maintenance transactions are examples of a typical configuration of the core when coming out of reset. Transactions can also be added, modified, or removed in `maintenance_list` module. The `srio_quick_start.v` is intended to be implemented in designs for managing common maintenance transactions if no processor is needed. This method is only good for setting up the core when coming out of reset.

The IO transactions are generated in the `srio_request_gen` and `srio_response_gen` modules. The `srio_request_gen` generates the initiator request IO transactions as specified in `instruction_list.v` file.

Transactions can be added, modified, or removed in `instruction_list.v`. The list contains random transactions, but is run in the same order each time. Only transactions supported by the connected port are produced. Figure 9-1 shows an example of a core configured with Initiator/Target Legacy ports. The `srio_request_gen` also keeps track of any expected responses for response-generating-requests, and compares the received response to the expected value. This comparison function is only available when using the automatic stimulus generation.

The `srio_response_gen` generates the target response IO transactions in response to requests received from the link partner. The data payload of write requests that fall within the scratch space of x0012_xxxx are stored in a local memory. All other write requests that do not require a response are dropped, and the `srio_response_gen` does not respond to SWRITE transaction types. Read requests that fall in the scratch space of x0012_xxxx get a response with data from that actual address. For instructions with an address offset outside of the scratch space, the response generator automatically fills the IO transaction data with an incrementing pattern. The responses are generated in the order that the requests are received; so no out-of-order transactions are generated. In all cases, response priorities are equal to the request priority plus 1.

Each instance of the `srio_request_gen` requires one block RAM. Each instance of `srio_response_gen` requires two block RAM resources.

Configuration Fabric

The Configuration Space is distributed across all the blocks of the SRIO Gen2 Endpoint. The Configuration Fabric reference design, located in the `cfg_fabric` module of the example design, manages the accesses to the configuration spaces of each of the blocks. This makes them appear unified to the user or processor element. The configuration modules in each of the blocks are slaves on the configuration bus (an AXI-Lite interface), and the Configuration Fabric module is the bus master. Reads and writes derived from Maintenance transactions that were issued either locally or from the link partner are presented to the Configuration Fabric via the Configuration Master port in the LOG block. The Configuration Fabric will route the reads and writes to the appropriate block, but will not route any transactions if the address is outside the valid range for the respective configuration registers. Writes to illegal space are dropped, and reads from illegal space are returned as zeros. See Accessing the Configuration Space in Chapter 3 for more detail.

Clock Module

The clock module in the example design is identical to the clock module delivered with the core, as described in Clocking in Chapter 3. The srio_clk module consists of one MMCM_ADV, one IBUFDS, and either three or four BUFGs depending on the selected family. Note that in 2x or 4x mode, one of the BUFGs is converted to a BUFGMUX. The srio_clk module generates the appropriate reference frequencies and user clocks, depending on the configuration.

Reset Module

The srio_rst module, as shown in Figure 9-2, takes in one asynchronous reset, and outputs a pulse-extended synchronized reset for each clock domain. The srio_rst module also contains a state machine to cause a forced re-initialization when called by the user. This module also resets the link when instructed to do so by the link partner.

Demonstration Test Bench

The srio_sim module, as shown in Figure 9-3, connects two instances of the example design (at the srio_example_top level) together. It generates the system clock and reset for simulation purposes, and monitors the simulation progress for status, including packet mis-compares, timeouts, and a tests complete flag. The two instances of the example design are srio_example_top_primary, which represents the user core, and srio_example_top_mirror, which acts as the link partner. The srio_example_top_mirror is configured to report received and transmitted transactions into the simulation transcript. When a simulation completes, either due to an error or when all requests have been sent and responses received, the simulation will stop and report into the transcript with the simulation status.

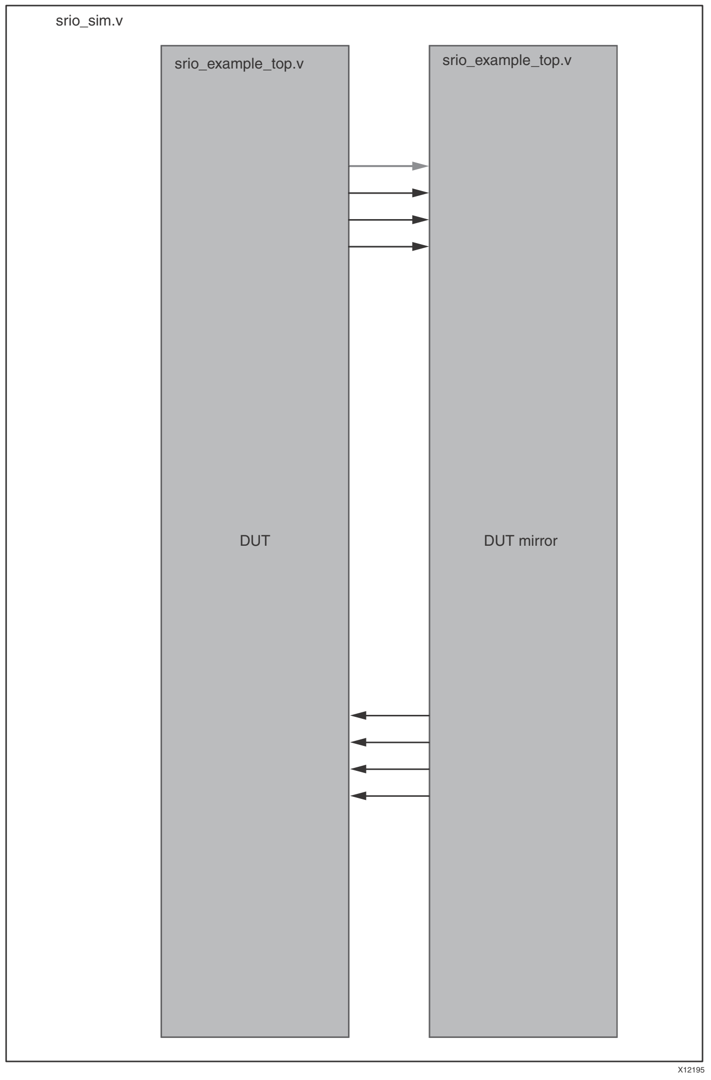


Figure 9-3: **Example Design Test Bench Layer**

Implementation

After the SRIO Gen2 Endpoint is generated, the SRIO Gen2 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 SRIO Gen2 Endpoint include scripts to assist you in running synthesis and implementation.

In the implementation example that follows, srio_gen2_v1_5 is the component name as generated by default from the SRIO Gen2 Endpoint configuration screen. If a core is generated with a different name, substitute that core name in the following commands.

- From a suitable DOS command window or Linux prompt, type the following to implement the SRIO Gen2 Endpoint example design.

  Windows

  ```
  > cd srio_gen2_v1_5\implement
  > implement.bat
  ```

  Linux

  ```
  % cd srio_gen2_v1_5/implement
  % ./implement.sh
  ```

These commands execute a script that synthesizes, builds, maps, and place-and-routes the SRIO Gen2 Endpoint example design. The resulting files are placed in the results directory of the SRIO Gen2 Endpoint, which is created by the implement script at runtime.

Simulation

Using the SRIO Gen2 Endpoint example design (delivered as part of the Serial SRIO Gen2 Endpoint), you can quickly simulate and observe the behavior of the SRIO Gen2 Endpoint.

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.

The Xilinx simulation libraries must be mapped into the simulator. If the libraries are not set for your environment, please refer to the *Synthesis and Simulation Design Guide* on the Xilinx software manuals Web page for assistance compiling Xilinx simulation models and setting up the simulator environment.

Simulations including the serial transceivers require a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator.

Functional Simulation

This section contains instructions for running a functional simulation of the SRIO Gen2 Endpoint using Verilog. A functional simulation model for the SRIO Gen2 Endpoint is provided when the core is 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_gen2_v1_5 for the component name of the SRIO Gen2

Endpoint 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_gen2_v1_5/simulation/functional`

2. Launch the simulation script:

   ```
   modelsim> do simulate_mti.do
   ```

Self Checking Simulation

The simulation script compiles the SRIO Gen2 Endpoint functional simulation model, the SRIO Gen2 Endpoint example design, and supporting simulation files. It then runs the simulation and checks to ensure that it completed successfully. The simulation uses self-checking in the srio_request_gen module, as previously described. If any response does not match the expected type or ID, the test bench will report a mismatch:

```
*****************************************************************
******************** ERROR: TEST FAILED *************************
************ Packet Miscompare or Packet Corruption *************
*****************************************************************
```

If a response never returns, the simulation will eventually timeout:

```
#################################################################
################## ERROR: TEST FAILED !!!!! #####################
#################### Testbench Timed Out! #######################
######################### (<timestamp>) #########################
#################################################################
```

A simulation success will be indicated by this text:

```
*****************************************************************
************************ TEST PASSED ***************************
************* All Packets Checked and Accounted For *************
*****************************************************************
```

To observe more details of 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 SRIO Gen2 Endpoint 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 Gen2 Endpoint. 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, PHY, LOG, and BUF).

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, an .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.

SECTION IV: APPENDICES

Packet and Control Symbol Formats

Scope

This appendix includes the block diagram for the RapidIO packet and control symbol format, which is based on the *RapidIO Interconnect Specification rev. 2.2*.

Figure A-1 shows the RapidIO packet format for Data packets. Figure A-2 presents the Control Symbol formats.

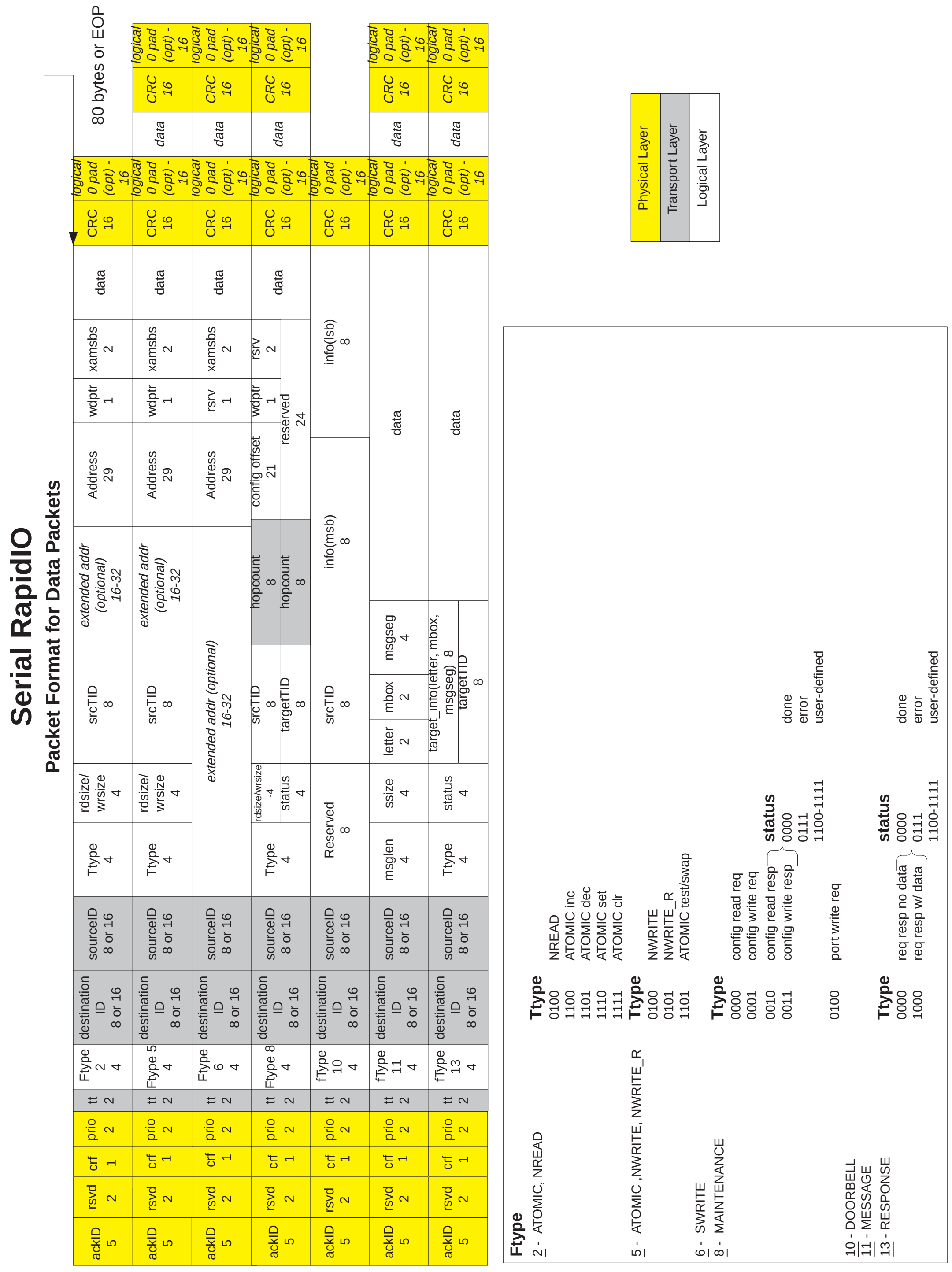
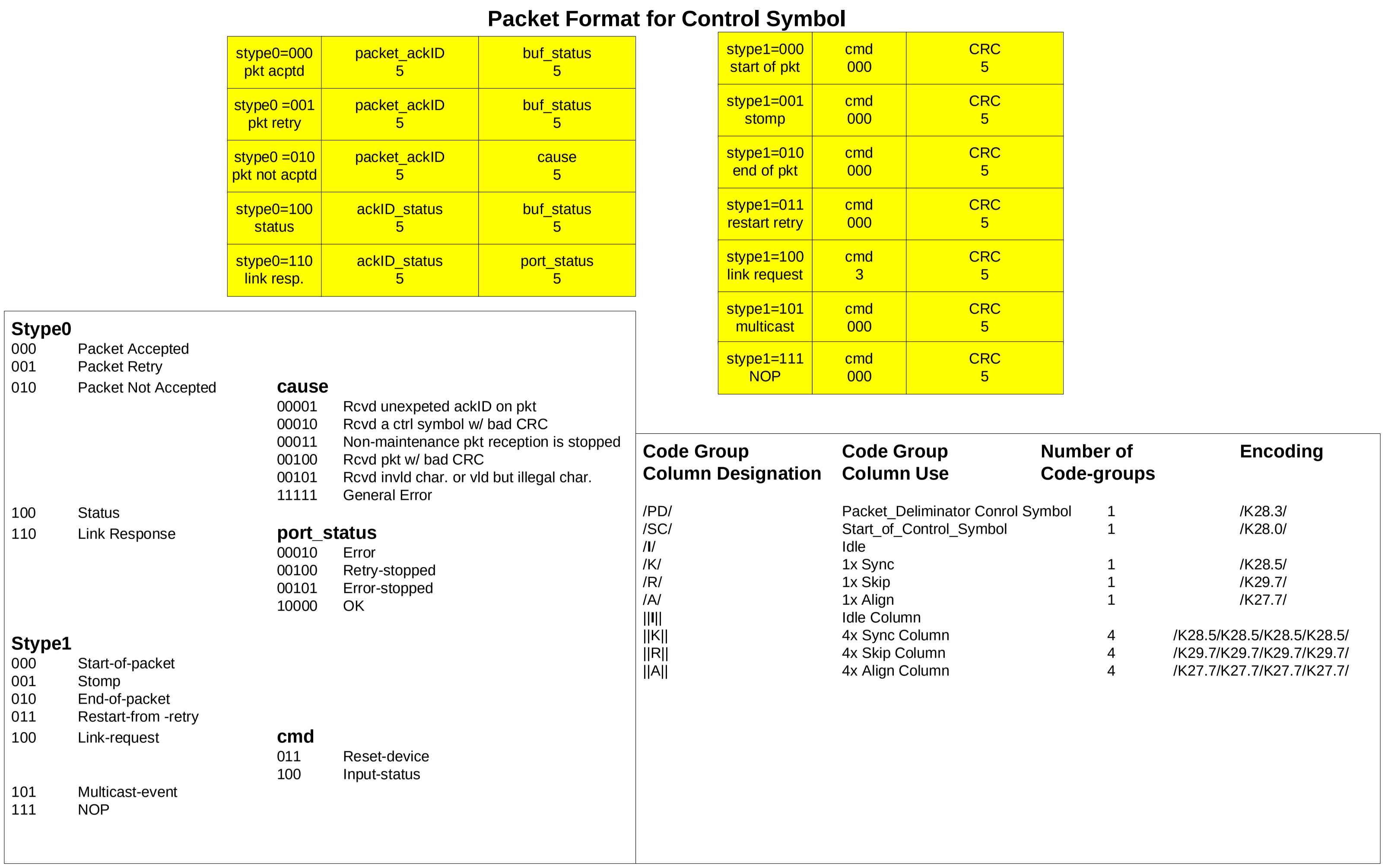


Figure A-1: **RapidIO Packet Format Block Diagram**



Packet Format for Control Symbol

stype0=000 pkt acptd	packet_ackID 5	buf_status 5
stype0 =001 pkt retry	packet_ackID 5	buf_status 5
stype0 =010 pkt not acptd	packet_ackID 5	cause 5
stype0=100 status	ackID_status 5	buf_status 5
stype0=110 link resp.	ackID_status 5	port_status 5

stype1=000 start of pkt	cmd 000	CRC 5
stype1=001 stomp	cmd 000	CRC 5
stype1=010 end of pkt	cmd 000	CRC 5
stype1=011 restart retry	cmd 000	CRC 5
stype1=100 link request	cmd 3	CRC 5
stype1=101 multicast	cmd 000	CRC 5
stype1=111 NOP	cmd 000	CRC 5

Stype0

000	Packet Accepted
001	Packet Retry
010	Packet Not Accepted
100	Status
110	Link Response

cause

00001	Rcvd unexpeted ackID on pkt
00010	Rcvd a ctrl symbol w/ bad CRC
00011	Non-maintenance pkt reception is stopped
00100	Rcvd pkt w/ bad CRC
00101	Rcvd invld char. or vld but illegal char.
11111	General Error

port_status

00010	Error
00100	Retry-stopped
00101	Error-stopped
10000	OK

Stype1

000	Start-of-packet
001	Stomp
010	End-of-packet
011	Restart-from -retry
100	Link-request
101	Multicast-event
111	NOP

cmd

011	Reset-device
100	Input-status

Code Group Column Designation	Code Group Column Use	Number of Code-groups	Encoding
/PD/	Packet_Deliminator Conrol Symbol	1	/K28.3/
/SC/	Start_of_Control_Symbol	1	/K28.0/
/I/	Idle		
/K/	1x Sync	1	/K28.5/
/R/	1x Skip	1	/K29.7/
/A/	1x Align	1	/K27.7/
||I||	Idle Column		
||K||	4x Sync Column	4	/K28.5/K28.5/K28.5/K28.5/
||R||	4x Skip Column	4	/K29.7/K29.7/K29.7/K29.7/
||A||	4x Align Column	4	/K27.7/K27.7/K27.7/K27.7/

Figure A-2: **RapidIO Packet Short Control Symbol Block Diagram**

Migrating

This appendix describes migrating from the Xilinx Serial RapidIO Gen1 core v5.6 or earlier to the SRIO Gen2 Endpoint v1.2 or later), this chapter provides key differences in behavior and options between the two cores.

For details on migrating from the ISE Design Suite to the Vivado Design Suite, see UG911, *Vivado Design Suite Migration Methodology Guide*.

Capability Differences between SRIO Gen1 and SRIO Gen2

Table B-1 highlights the differences between the SRIO Gen1 and SRIO Gen2 cores.

Table B-1: **Feature Differences between Gen1 and Gen2 Cores**

Features	SRIO Gen1	SRIO Gen2
Lane Width	x1, x4	x1, x2, x4
Supported Devices	Virtex-6/Spartan-6/Virtex-5/ Virtex-4	Virtex-6/Kintex-7/Virtex-7
Line Rate	1.25, 2.5, 3.125, 5.0	1.25, 2.5, 3.125, 5.0, 6.25
Doorbell and Message Support	Yes	Yes
VHDL Support	Yes	No(1)
AXI Support	No	Yes
RapidIO Specification	rev. 2.1(2)	rev. 2.2
User Interface		
Initiator/Target Style	Yes	Yes
Condensed I/O Port	No	Yes
User-defined Port	No(3)	Yes
Maintenance Port	Yes	Yes
Separate Messaging Port	No(3)	Yes
IDLE Support	IDLE1	IDLE1/IDLE2(4)

Table B-1: **Feature Differences between Gen1 and Gen2 Cores *(Cont'd)***

Features	SRIO Gen1	SRIO Gen2
Link-reset/mce	Yes	Yes(5)
Configuration Accesses	256-Byte Interface(6)	4-Byte Interface
Example Design	Contains ChipScope / Register Manager	No ChipScope / Configuration Fabric
Register Bit (Swap)	Big-Endian	Little Endian

1. No VHDL support in ISE Design Suite v13.3/13.4 for SRIO Gen2 Endpoint. This includes the example design, wrappers, or scripts.
2. Only the 5.0Gb portion of the *RapidIO Interconnect Specification* rev. 1.2 is supported.
3. This port is integrated into the Initiator/Target Port.
4. Check the *RapidIO Interconnect Specification* rev. 2.2 for more information on IDLE2.
5. Different mechanism of operation - more information provided below.
6. This is the max size, though only 4-byte accesses were used in the example design.

User Interface (AXI-4 Stream)

SRIO Gen2 uses AXI-4 Stream. There are two implementations of AXI4-Stream:

- HELLO Format
- SRIO Stream Format

The I/O port can be configured to use either HELLO Format or SRIO Stream Format packets. SRIO Stream Format is used between the LOG/BUF and BUF/PHY interfaces. It is also the format of the User-Defined user interface port of the LOG.

HELLO Format provides a standardization of the header field across packet types, and segments the header and data into separate transfers on the interface. Figure 3-2 shows packet data header in HELLO Format. Figure 3-4 shows Basic HELLO Packet Transfer on User Interface. Figure 3-5 shows Advanced HELLO Packet Transfer on User Interface.

In SRIO Stream Format, packets are presented fully formed as defined in the RapidIO Specification including all Logical/Transport layer fields (plus the priority which the specification defines as a Physical Layer field). Figure 3-6 shows Basic SRIO Stream Format on User Interface. Figure 3-7 shows Advanced SRIO Stream Format on User Interface.

At the user port in SRIO Gen1, the header information is provided through separate signals in the Initiator Port Signaling Interface described in UG503, *LogiCORE IP Serial RapidIO User Guide v5.6* or later.

Packet transfer between LOG and BUF and between BUF and PHY occurs through LocalLink interface as described in UG503, *LogiCORE IP Serial RapidIO v5.6 User Guide* or later.

Logical Layer Ports (User Interface)

In SRIO Gen1, there is only one user interface where the interface is divided into four separate interfaces from which a packet can be issued or consumed. The interfaces are Initiator Request, Initiator Response, Target Request and Target Response. In SRIO Gen2, the user interface is divided into four different ports:

- I/O port
- Messaging Port
- User-Defined Port
- Maintenance Port

There are two types of I/O port: Condensed I/O and Initiator/Target. The user can select either of them during CORE Generator tool configuration. The difference between the two ports are that the former one has only two channels (iotx and iorx) to transmit and receive I/O packets whereas the latter has four channels. They are ireq, iresp, treq and tresp, and are similar to the user interface in SRIO Gen1. All the channels in both ports are AXI4-Stream. Each channel could be configured with either HELLO format or SRIO Stream Format. See Chapter 2, Product Specification for a list of signals on these interface channels.

Unlike in SRIO Gen1 where both Message and other packet types are transmitted and received from the same channel, in SRIO Gen2 there is a separate Messaging port with four distinct channels: msgireq, msgiresp, msgtreq and msgtresp. However, it is not restricted to channel all the Message packets through this port, messages can also be combined onto the I/O ports, treated as write transactions. The Messaging port uses either HELLO or SRIO Stream Format.

Another port in the user interface is the User-Defined port with two AXI4-Stream interfaces (txuser and rxuser) that support only SRIO Stream Format. If a received transaction is not supported, it will be presented on the User-Defined port. This is an optional port and can be disabled in CORE Generator tool during core configuration.

The last port in the user interface is the Maintenance port. This is a separate port in SRIO Gen2 for only maintenance transactions. This is also an optional port and can be disabled in the CORE Generator GUI. This port can be configured both as AXI-4 Lite interface or as AXI4-Stream. The AXI4-Lite interface allows the user application to target either the local or remote configuration space, whereas the AXI4-Stream port only provides access to remote transactions. Table 2-7, page 26 lists the AXI-4 Lite Maintenance Port signals.

Configuration Access

This section describes the differences between the configuration access transactions.

Maintenance Transaction

There are two primary differences between SRIO Gen1 and SRIO Gen2 regarding maintenance transaction:

- SRIO Gen1 supports any maintenance transaction size supported by standard NREAD/ NWRITE packets. However, SRIO Gen2 only supports 4-byte accesses. The *RapidIO Specification* rev. 2.2 states that 4-byte accesses should be used. In SRIO Gen1, the max size was 256-byte accesses though only 4-byte accesses were used in the example design.
- While sending remote configuration accesses from SRIO Gen2, the necessary information to setup header of the outgoing transaction including Destination ID, TID, Priority and CRF are taken from the Maintenance Request Information Register. SRIO Gen2 is used as an endpoint core. Because of this, the core is not responsible for configuring the system so this information would not be used. However, if the core is not used as an endpoint, this register needs to be written to change these fields for each maintenance request. The one exception is TID. The TID will increment each subsequent request. When just issuing maintenance requests to the same Destination ID, the Maintenance Request Information Register has to be written only once. In SRIO Gen1, the above information is provided through the Maintenance Port Signaling interface as described in UG503, *LogiCORE IP Serial RapidIO v5.6 User Guide*.

Management Interface

SRIO Gen1 uses the register manager and the management interface to access the configuration registers of the core. In SRIO Gen2, the management interface has been replaced with AXI4-LITE interface which is referred as configuration interface. The Configuration Fabric, which is just an AXI4-LITE interconnect, is delivered as RTL reference design with the generation of the core (similar to the register manager in SRIO Gen1). The reference design manages the accesses to the configuration spaces of each of the cores (LOG, BUF, PHY). The Configuration Fabric module is the bus master on the configuration bus (AXI-LITE Interface) connecting with each of the cores whereas Configuration modules in each of these cores are slaves on the bus. All read and write Maintenance transactions arising locally or issued from a remote link partner are first routed to the Configuration Fabric via the Configuration Master port in the LOG.

The Management interface signals to the LOG in SRIO Gen1 core are described in UG503, *LogiCORE IP Serial RapidIO v5.6 User Guide*. Table 2-10, page 29 shows the corresponding interface in SRIO Gen 2 core.

Register Bit Swap

SRIO Gen1 is big-endian and local link based while SRIO Gen2 is AXI-based and is little-endian. It is a swap of the order, but it allows what was once bit 0 to still be on the left side since it will now be bit 31.

MCE Generation and Link-Reset Control Symbols

In SRIO Gen1, the MCE was a req/ack interface which required the user to hold the req signal (`lnk_mcast_req_n`) until an ACK (`lnk_mcast_ack_n`) is returned from the PHY indicating that an MCE was sent. In SRIO Gen2, the `phy_mce` signal needs to be asserted only for a single cycle, and an MCE will be sent.

There is a difference between the two cores regarding the link-reset control symbols. In SRIO Gen1, the signal must be asserted for at least four clock cycles to ensure that four link-reset control symbols were sent. In SRIO Gen2, the signal should be asserted until the `port_initialized` signal deasserts to successfully reset the link partner.

User Accessible Signals Comparison

Table B-2 lists the differences between user accessible signals.

Table B-2: **Signal Comparison**

SRIO Gen1	SRIO Gen2	Difference
lnk_mcast_req_n	phy_mce	See MCE Generation and Link-Reset Control Symbols
lnk_mcast_ack_n		No Equivalent
lnk_linkreset_n	phy_link_reset	Name Change
fast_train	sim_train_en	Name Change
port_initialized	port_initialized	Same
	idle_selected	New Signal
mode_sel	mode_1x	Name Change
lnk_trdy_n/lnk_rrdy_n	link_initialized	Name Change
lnk_porterr_n	port_error	Name Change
resp_time_out	port_timeout	Name Change

Clocking and Reset

This section describes the differences in the clock and reset functionality.

LOG Layer

Table B-3 details the differences in the LOG Layer.

Table B-3: **LOG Layer Comparison**

SRIO Gen1	SRIO Gen2	Difference
clk[(1)]	log_clk[(2)]	Name Change; Functional Difference
reset_n	log_rst	Name Change
mgt_clk	cfg_clk[(2)]	Name Change; Functional Difference
mgt_reset_n	cfg_rst	Name Change; Functional Difference

1. See UG503, *LogiCORE IP Serial RapidIO v5.6 User Guide*, for clock frequencies at different line rates and 1x/4x mode.
2. Check Table 3-4, page 89 through Table 3-6, page 89 for clock frequencies at different line rates and 1x/4x mode.

BUF Layer

Table B-4 details the differences in the LOG Layer.

Table B-4: **BUF Layer Comparison**

SRIO Gen1	SRIO Gen2	Difference
log_clk[(1)]	log_clk[(1)]	Same Name; Functional Difference
log_reset_n		No Equivalent
phy_clk[(1)]	phy_clk[(1)]	Same Name; Functional Difference
phy_reset_n		No Equivalent
	buf_rst	New Signal
mgt_clk	cfg_clk[(1)]	Name Change; Functional Difference
mgt_rst_n	cfg_rst	Name Change; Functional Difference

1. See UG503, *LogiCORE IP Serial RapidIO v5.6 User Guide*, for clock frequencies at different line rates and 1x/4x mode.
2. Check Table 3-4, page 89 through Table 3-6, page 89 for clock frequencies at different line rates and 1x/4x mode.

PHY Layer

Table B-4 details the differences in the PHY Layer.

Table B-5: **PHY Layer Comparison**

SRIO Gen1	SRIO Gen2	Difference
sys_reset_n		No Equivalent
lnk_reset_n		No Equivalent
uclk_dv4[(1)]		No Equivalent
uclk_lock		No Equivalent
uclk[(1)]		No Equivalent
	phy_clk[(2)]	New Signal

Table B-5: **PHY Layer Comparison** ***(Cont'd)***

SRIO Gen1	SRIO Gen2	Difference
	phy_rst	New Signal
	gt_pcs_clk[(2)]	New Signal
	gt_pcs_rst	New Signal
mgt_clk	cfg_clk[(2)]	Name Change; Functional Difference
mgt_reset_n	cfg_rst	Name Change; Functional Difference

1. See UG503, *LogiCORE IP Serial RapidIO v5.6 User Guide*, for clock frequencies at different line rates and 1x/4x mode.
2. Check Table 3-4, page 89 through Table 3-6, page 89 for clock frequencies at different line rates and 1x/4x mode.

SRIO Gen2 operates on the following clock domains: `phy_clk`, `gt_pcs_clk`, `gt_clk`, `log_clk` and `cfg_clk`. `phy_clk` and `gt_pcs_clk are` the PHY clock domains. gt_clk is not used by the PHY, but is used by the Serial Transceiver interface. Table 3-4, page 89 through Table 3-6, page 89 show typical clock rates at x1, x2 and x4 operations of SRIO Gen2 Endpoint.

Core Generation

The SRIO Gen2 CORE Generator GUI provides an option for selecting IDLE Mode Support. The IDLE2 sequence is supported only on SRIO Gen2 core. The IDLE2 sequence provides additional core functionality such as link width and lane polarity information, randomized data for improved data recovery etc.

As discussed in the User Interface (AXI-4 Stream), page 211, the CORE Generator GUI provides options to select the interface suitable for the user application. Through the GUI, select whether to use Hello Format or SRIO Stream Format. Figure 7-4, page 174 shows a screen shot of the CORE Generator GUI where this selection is done.

Example Design

The example design does the same thing in both SRIO Gen1 and SRIO Gen2 but is structured differently. They both send pre-constructed traffic to the core from block RAM. The example design in the SRIO Gen1 core has more logic for ChipScope. For the current Gen2 core, there is no ChipScope functionality. In addition, SRIO Gen1 provides a Register Manager; this is replaced in SRIO Gen2 by the Configuration Fabric.

Debugging

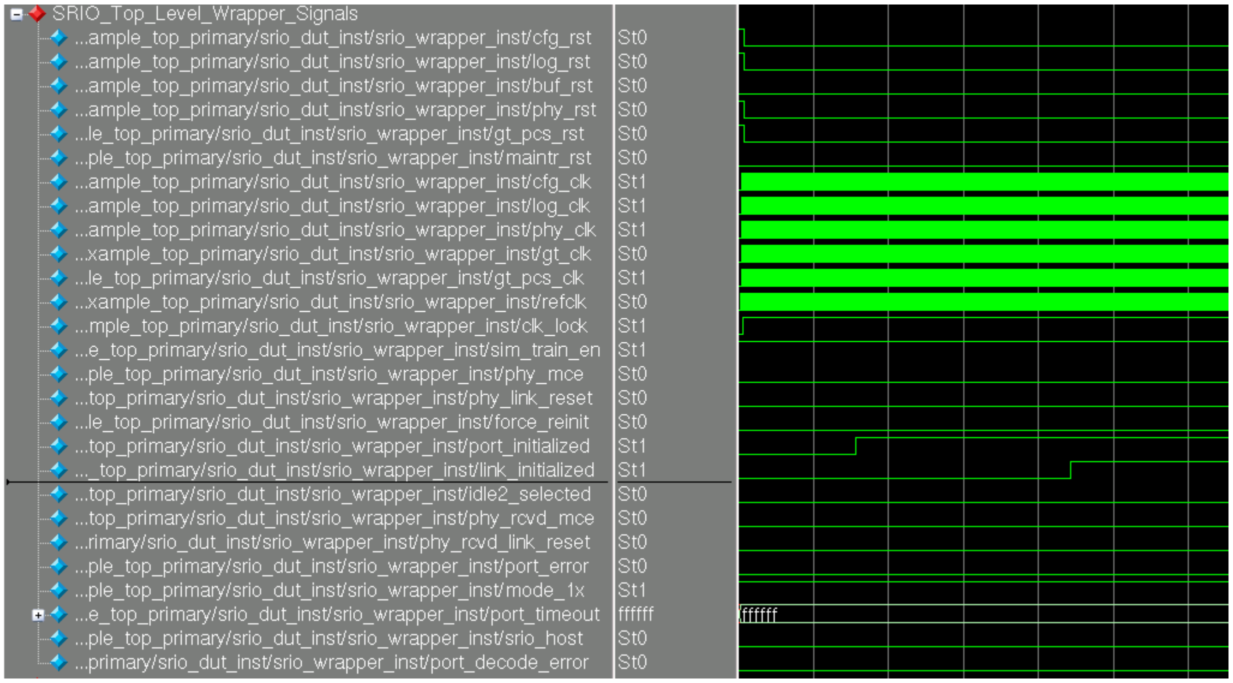
This chapter describes packet and signal analysis for debugging the SRIO Gen2 core. It provides a detailed description on tracking SRIO packets (Control Symbols and Data Characters) on the interfaces of the core and also tips to resolve link initialization issues. SWRITE Packet Analysis details a complete analysis of an SWRITE packet through different interfaces. For analysis of other packet types, see Xilinx Answer Record 50166.

Link Initialization

When debugging link issues, start by checking that the following SRIO Gen2 top-level wrapper signals are toggling properly. Including, but not limited to:

- `reset`
- `clk_lock`
- `port_initialized`
- `link_initialized`
- `port_error`
- `mode_1x`
- `port_decore_error`

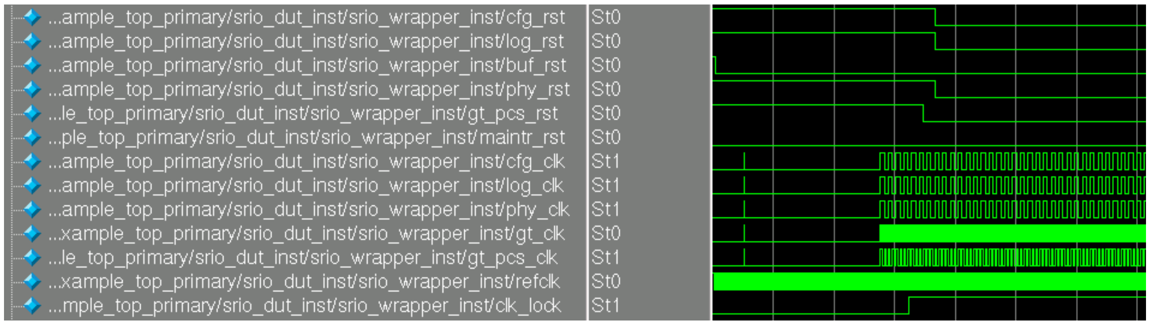
Figure C-1 shows a list of all the signals to check.



```
SRIO_Top_Level_Wrapper_Signals
...ample_top_primary/srio_dut_inst/srio_wrapper_inst/cfg_rst          St0
...ample_top_primary/srio_dut_inst/srio_wrapper_inst/log_rst          St0
...ample_top_primary/srio_dut_inst/srio_wrapper_inst/buf_rst          St0
...ample_top_primary/srio_dut_inst/srio_wrapper_inst/phy_rst          St0
...le_top_primary/srio_dut_inst/srio_wrapper_inst/gt_pcs_rst          St0
...ple_top_primary/srio_dut_inst/srio_wrapper_inst/maintr_rst         St0
...ample_top_primary/srio_dut_inst/srio_wrapper_inst/cfg_clk          St1
...ample_top_primary/srio_dut_inst/srio_wrapper_inst/log_clk          St1
...ample_top_primary/srio_dut_inst/srio_wrapper_inst/phy_clk          St1
...xample_top_primary/srio_dut_inst/srio_wrapper_inst/gt_clk          St0
...le_top_primary/srio_dut_inst/srio_wrapper_inst/gt_pcs_clk          St1
...xample_top_primary/srio_dut_inst/srio_wrapper_inst/refclk          St0
...mple_top_primary/srio_dut_inst/srio_wrapper_inst/clk_lock          St1
...e_top_primary/srio_dut_inst/srio_wrapper_inst/sim_train_en         St1
...ple_top_primary/srio_dut_inst/srio_wrapper_inst/phy_mce            St0
...top_primary/srio_dut_inst/srio_wrapper_inst/phy_link_reset         St0
...le_top_primary/srio_dut_inst/srio_wrapper_inst/force_reinit        St0
...top_primary/srio_dut_inst/srio_wrapper_inst/port_initialized       St1
..._top_primary/srio_dut_inst/srio_wrapper_inst/link_initialized      St1
...top_primary/srio_dut_inst/srio_wrapper_inst/idle2_selected         St0
...top_primary/srio_dut_inst/srio_wrapper_inst/phy_rcvd_mce           St0
...rimary/srio_dut_inst/srio_wrapper_inst/phy_rcvd_link_reset         St0
...ple_top_primary/srio_dut_inst/srio_wrapper_inst/port_error         St0
...ple_top_primary/srio_dut_inst/srio_wrapper_inst/mode_1x            St1
...e_top_primary/srio_dut_inst/srio_wrapper_inst/port_timeout         ffffff
...ple_top_primary/srio_dut_inst/srio_wrapper_inst/srio_host          St0
...primary/srio_dut_inst/srio_wrapper_inst/port_decode_error          St0
```

Figure C-1: **SRIO Top Level Wrapper Signals**

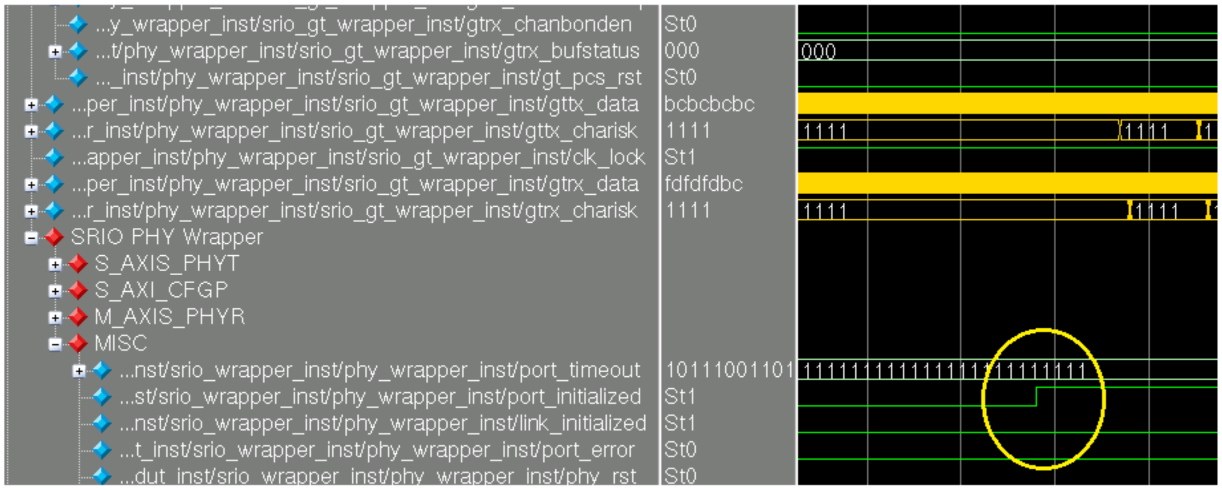
Figure C-2 shows a close up view of Figure C-1 to illustrate the correct toggling of the reset signals in the core.



```
...ample_top_primary/srio_dut_inst/srio_wrapper_inst/cfg_rst          St0
...ample_top_primary/srio_dut_inst/srio_wrapper_inst/log_rst          St0
...ample_top_primary/srio_dut_inst/srio_wrapper_inst/buf_rst          St0
...ample_top_primary/srio_dut_inst/srio_wrapper_inst/phy_rst          St0
...le_top_primary/srio_dut_inst/srio_wrapper_inst/gt_pcs_rst          St0
...ple_top_primary/srio_dut_inst/srio_wrapper_inst/maintr_rst         St0
...ample_top_primary/srio_dut_inst/srio_wrapper_inst/cfg_clk          St1
...ample_top_primary/srio_dut_inst/srio_wrapper_inst/log_clk          St1
...ample_top_primary/srio_dut_inst/srio_wrapper_inst/phy_clk          St1
...xample_top_primary/srio_dut_inst/srio_wrapper_inst/gt_clk          St0
...le_top_primary/srio_dut_inst/srio_wrapper_inst/gt_pcs_clk          St1
...xample_top_primary/srio_dut_inst/srio_wrapper_inst/refclk          St0
...mple_top_primary/srio_dut_inst/srio_wrapper_inst/clk_lock          St1
```

Figure C-2: **Core Reset Signals**

Figure C-3 shows a capture of `port_initalized` and `link_initalized` signals. When `port_initalized` and `link_initalized` are both valid, the user can send/receive packets. `link_initalized` indicates seven consecutive error free control symbols have been received, and 15 consecutive symbols have been sent. The red circles in Figure C-3 are control symbols.



```
...y_wrapper_inst/srio_gt_wrapper_inst/gtrx_chanbonden      St0
...t/phy_wrapper_inst/srio_gt_wrapper_inst/gtrx_bufstatus   000
..._inst/phy_wrapper_inst/srio_gt_wrapper_inst/gt_pcs_rst   St0
...per_inst/phy_wrapper_inst/srio_gt_wrapper_inst/gttx_data     bcbcbcbc
...r_inst/phy_wrapper_inst/srio_gt_wrapper_inst/gttx_charisk    1111
...apper_inst/phy_wrapper_inst/srio_gt_wrapper_inst/clk_lock    St1
...per_inst/phy_wrapper_inst/srio_gt_wrapper_inst/gtrx_data     fdfdfdbc
...r_inst/phy_wrapper_inst/srio_gt_wrapper_inst/gtrx_charisk    1111
SRIO PHY Wrapper
  S_AXIS_PHYT
  S_AXI_CFGP
  M_AXIS_PHYR
  MISC
    ...nst/srio_wrapper_inst/phy_wrapper_inst/port_timeout      10111001101
    ...st/srio_wrapper_inst/phy_wrapper_inst/port_initialized   St1
    ...nst/srio_wrapper_inst/phy_wrapper_inst/link_initialized  St1
    ...t_inst/srio_wrapper_inst/phy_wrapper_inst/port_error     St0
    ...dut_inst/srio_wrapper_inst/phy_wrapper_inst/phy_rst      St0
```

Figure C-3: **port_initialized and link_initialized Signals**

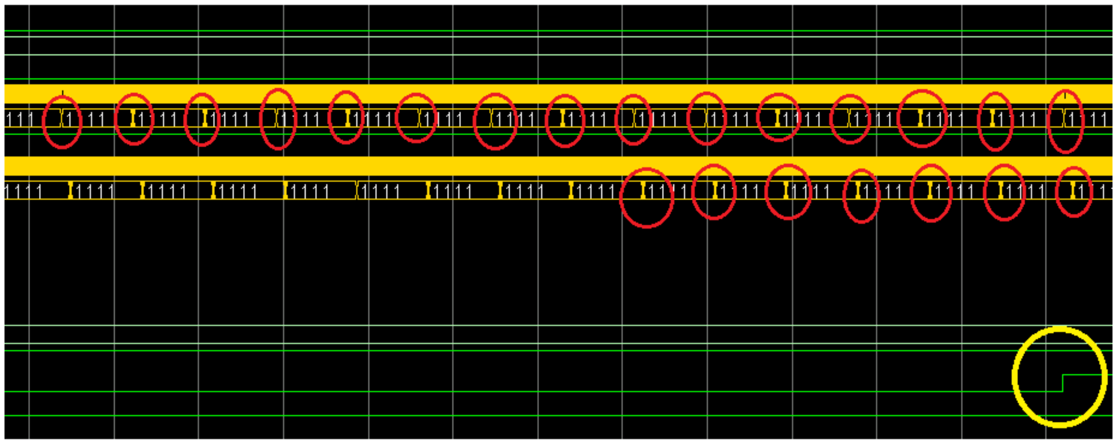
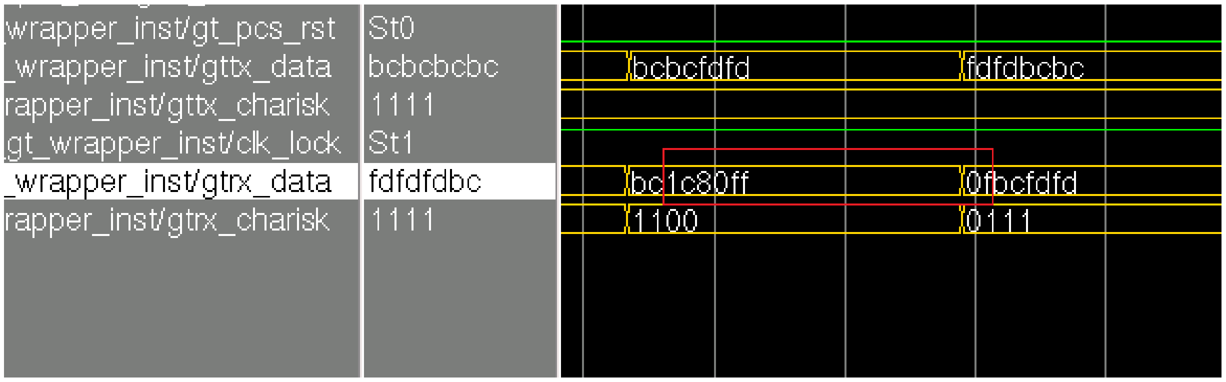


Figure C-4: **port_initialized and link_initialized Signals (Figure C-3 Enlarged)**

Figure C-5 shows a close up view of one of the red circles in Figure C-3 depicting the transmission of a control symbol. In the red box, "1c" is K28.0 which indicates the start of the control symbol. "bc" is K28.5, /K/, IDLE for sync. According to control symbol packet format shown in Figure A-2, the entire packet is decoded as follows:

- 80ff0f = 1000_0000_1111_1111_0000_1111
- stype0 = 100 (Status)
- ackID_status = 00000
- buf_status = 11111
- stype1 = 111 (NOP)
- cmd = 000

- CRC = 01111

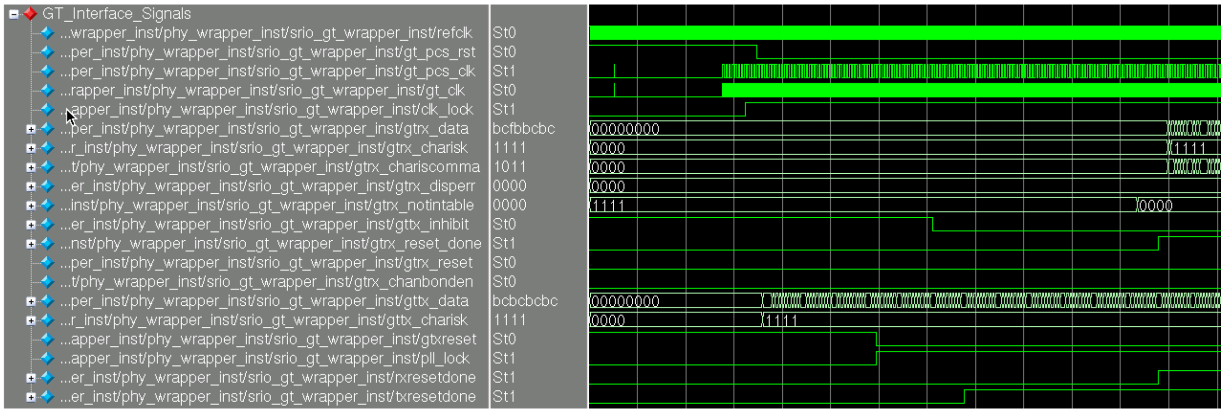


```
wrapper_inst/gt_pcs_rst      St0
_wrapper_inst/gttx_data      bcbcbcbc
rapper_inst/gttx_charisk     1111
gt_wrapper_inst/clk_lock     St1
_wrapper_inst/gtrx_data      fdfdfdbc
rapper_inst/gtrx_charisk     1111
```

Figure C-5: **Control Symbol Before Assertion of link_initialized Signal**

If the initialization doesn't complete (`port_initialized` and `link_initalized` signals are not asserted), check GT_RXDATA to confirm that IDLE (bcfd) ordered sets are being received and also check for any coding errors. This can be done by looking at RXNOTINTABLE which should always be '0'.

Ensure all the signals on the GT interface are toggling correctly. Figure C-6 shows those signals.

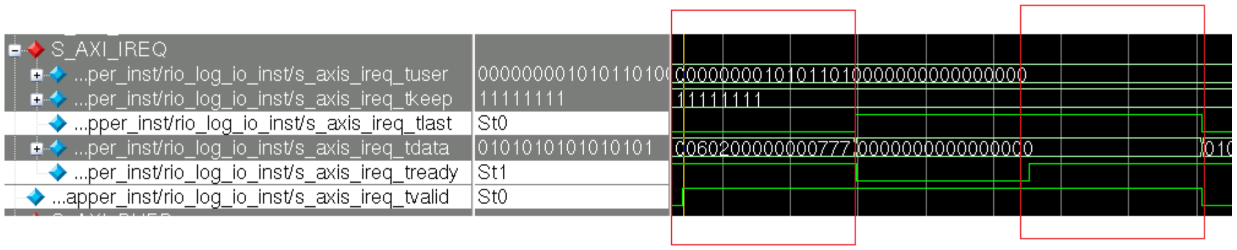


```
GT_Interface_Signals
...wrapper_inst/phy_wrapper_inst/srio_gt_wrapper_inst/refclk          St0
...per_inst/phy_wrapper_inst/srio_gt_wrapper_inst/gt_pcs_rst          St0
...per_inst/phy_wrapper_inst/srio_gt_wrapper_inst/gt_pcs_clk          St1
...rapper_inst/phy_wrapper_inst/srio_gt_wrapper_inst/gt_clk           St0
...apper_inst/phy_wrapper_inst/srio_gt_wrapper_inst/clk_lock          St1
...per_inst/phy_wrapper_inst/srio_gt_wrapper_inst/gtrx_data           bcfbbcbc
...r_inst/phy_wrapper_inst/srio_gt_wrapper_inst/gtrx_charisk          1111
...t/phy_wrapper_inst/srio_gt_wrapper_inst/gtrx_chariscomma           1011
...er_inst/phy_wrapper_inst/srio_gt_wrapper_inst/gtrx_disperr         0000
...inst/phy_wrapper_inst/srio_gt_wrapper_inst/gtrx_notintable         0000
...er_inst/phy_wrapper_inst/srio_gt_wrapper_inst/gttx_inhibit         St0
...nst/phy_wrapper_inst/srio_gt_wrapper_inst/gtrx_reset_done          St1
...per_inst/phy_wrapper_inst/srio_gt_wrapper_inst/gtrx_reset          St0
...t/phy_wrapper_inst/srio_gt_wrapper_inst/gtrx_chanbonden            St0
...per_inst/phy_wrapper_inst/srio_gt_wrapper_inst/gttx_data           bcbcbcbc
...r_inst/phy_wrapper_inst/srio_gt_wrapper_inst/gttx_charisk          1111
...apper_inst/phy_wrapper_inst/srio_gt_wrapper_inst/gtxreset          St0
...apper_inst/phy_wrapper_inst/srio_gt_wrapper_inst/pll_lock          St1
...er_inst/phy_wrapper_inst/srio_gt_wrapper_inst/rxresetdone          St1
...er_inst/phy_wrapper_inst/srio_gt_wrapper_inst/txresetdone          St1
```

Figure C-6: **GT Interface Signals**

SWRITE Packet Analysis

Figure C-7 shows a SWRITE Packet sent to the IREQ Interface, the data on `s_axis_ireq_tdata` is 0060200000000777_0000000000000000.



```
S_AXI_IREQ
...per_inst/rio_log_io_inst/s_axis_ireq_tuser    000000001010110100  00000000101011010000000000000000
...per_inst/rio_log_io_inst/s_axis_ireq_tkeep    11111111            11111111
...pper_inst/rio_log_io_inst/s_axis_ireq_tlast   St0
...per_inst/rio_log_io_inst/s_axis_ireq_tdata    0101010101010101    0060200000000777  0000000000000000  010
...per_inst/rio_log_io_inst/s_axis_ireq_tready   St1
...apper_inst/rio_log_io_inst/s_axis_ireq_tvalid St0
```

Figure C-7: **SWRITE Packet Transmission on IREQ Interface**

According to the HELLO FTYPE Packet Format in Figure 3-2, the corresponding field interpretation for the above data in Figure C-8 is as follows:

0060200000000777 =

0000_0000_0110_0000_0010_0000_0000_0000_0000_0000_0000_0000_0000_0111_0111_0111

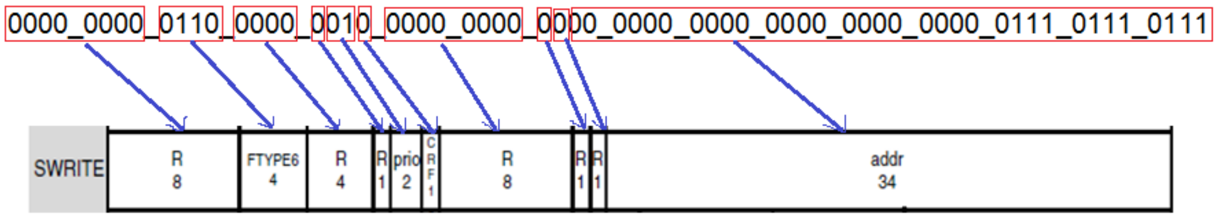


Figure C-8: **SWRITE HELLO FTYPE Packet Format**

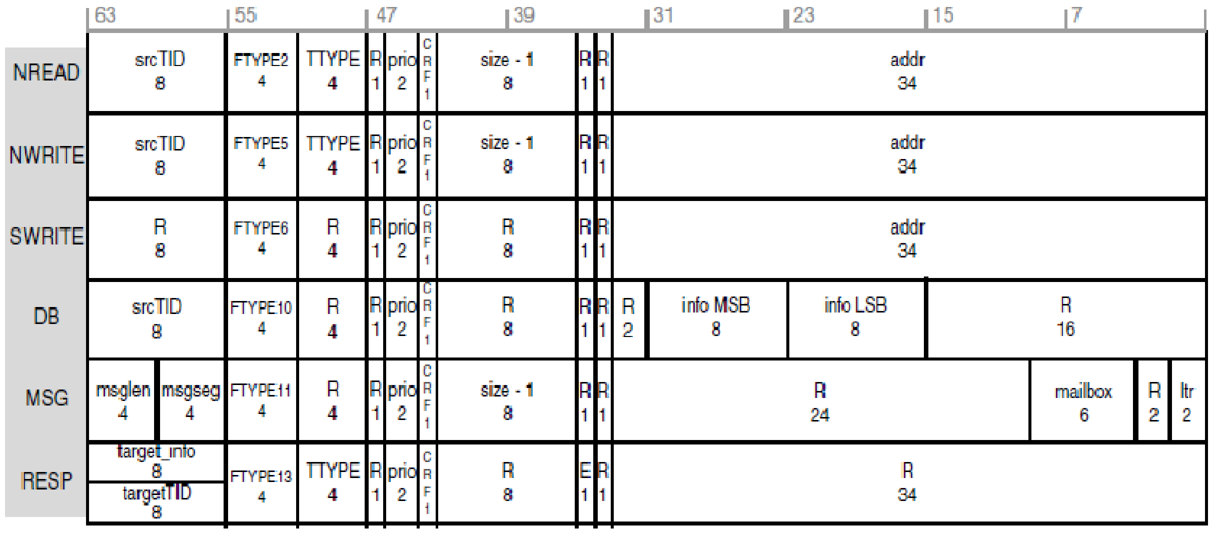
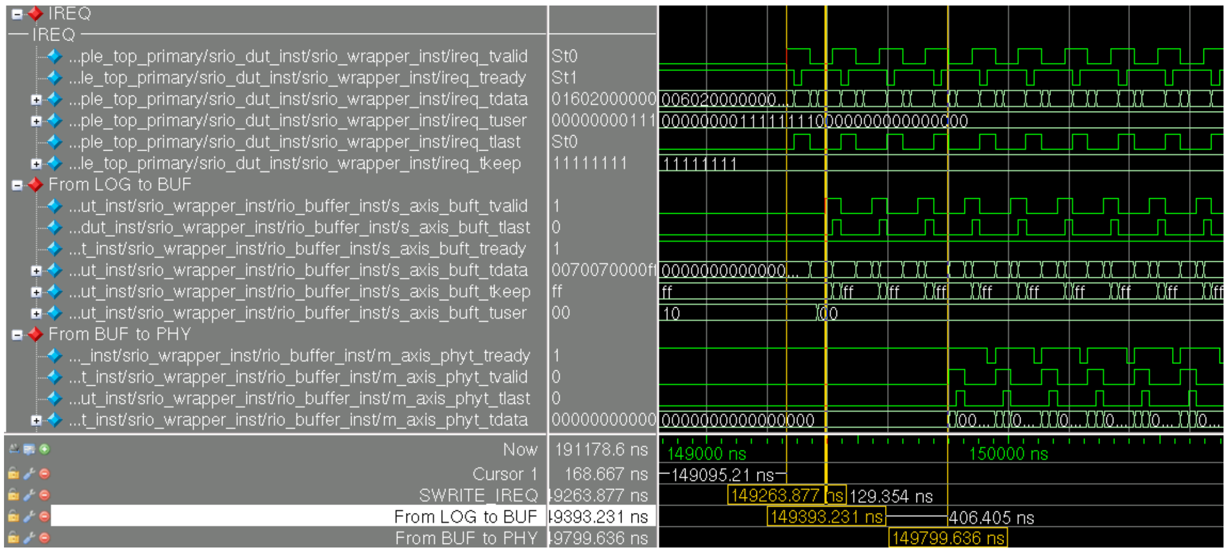


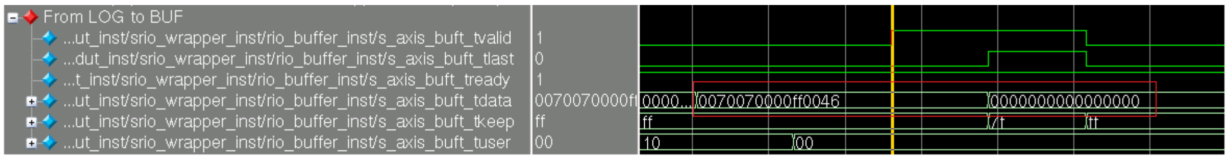
Figure C-9: **HELLO FTYPE Packet Format**

The data received on the IREQ interface of the SRIO Gen2 is captured by the Logical Layer (LOG). The LOG adds transport layer info into the packet, sends it to the buffer (BUF) and then BUF sends the same data to the Physical Layer (PHY). The naming of the signals is based on the block diagram in Figure 2-8.



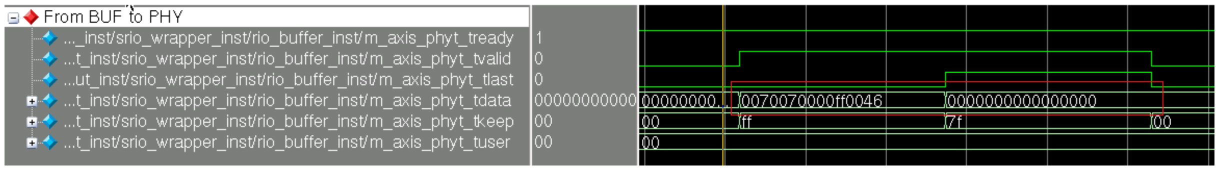
```
IREQ
 IREQ
   ...ple_top_primary/srio_dut_inst/srio_wrapper_inst/ireq_tvalid      St0
   ...le_top_primary/srio_dut_inst/srio_wrapper_inst/ireq_tready       St1
   ...ple_top_primary/srio_dut_inst/srio_wrapper_inst/ireq_tdata       01602000000
   ...ple_top_primary/srio_dut_inst/srio_wrapper_inst/ireq_tuser       00000000111
   ...ple_top_primary/srio_dut_inst/srio_wrapper_inst/ireq_tlast       St0
   ...le_top_primary/srio_dut_inst/srio_wrapper_inst/ireq_tkeep        11111111
From LOG to BUF
   ...ut_inst/srio_wrapper_inst/rio_buffer_inst/s_axis_buft_tvalid     1
   ...dut_inst/srio_wrapper_inst/rio_buffer_inst/s_axis_buft_tlast     0
   ...t_inst/srio_wrapper_inst/rio_buffer_inst/s_axis_buft_tready      1
   ...ut_inst/srio_wrapper_inst/rio_buffer_inst/s_axis_buft_tdata      0070070000ff
   ...ut_inst/srio_wrapper_inst/rio_buffer_inst/s_axis_buft_tkeep      ff
   ...ut_inst/srio_wrapper_inst/rio_buffer_inst/s_axis_buft_tuser      00
From BUF to PHY
   ..._inst/srio_wrapper_inst/rio_buffer_inst/m_axis_phyt_tready       1
   ...t_inst/srio_wrapper_inst/rio_buffer_inst/m_axis_phyt_tvalid      0
   ...ut_inst/srio_wrapper_inst/rio_buffer_inst/m_axis_phyt_tlast      0
   ...t_inst/srio_wrapper_inst/rio_buffer_inst/m_axis_phyt_tdata       00000000000

                  Now     191178.6 ns
             Cursor 1      168.667 ns
          SWRITE_IREQ    9263.877 ns
         From LOG to BUF  9393.231 ns
         From BUF to PHY  9799.636 ns
```

Figure C-10: **SWRITE Packet Transmission on IREQ, LOG to BUF and BUF to PHY**



```
From LOG to BUF
   ...ut_inst/srio_wrapper_inst/rio_buffer_inst/s_axis_buft_tvalid     1
   ...dut_inst/srio_wrapper_inst/rio_buffer_inst/s_axis_buft_tlast     0
   ...t_inst/srio_wrapper_inst/rio_buffer_inst/s_axis_buft_tready      1
   ...ut_inst/srio_wrapper_inst/rio_buffer_inst/s_axis_buft_tdata      0070070000ff
   ...ut_inst/srio_wrapper_inst/rio_buffer_inst/s_axis_buft_tkeep      ff
   ...ut_inst/srio_wrapper_inst/rio_buffer_inst/s_axis_buft_tuser      00
```

Figure C-11: **SWRITE Packet from LOG to BUF**

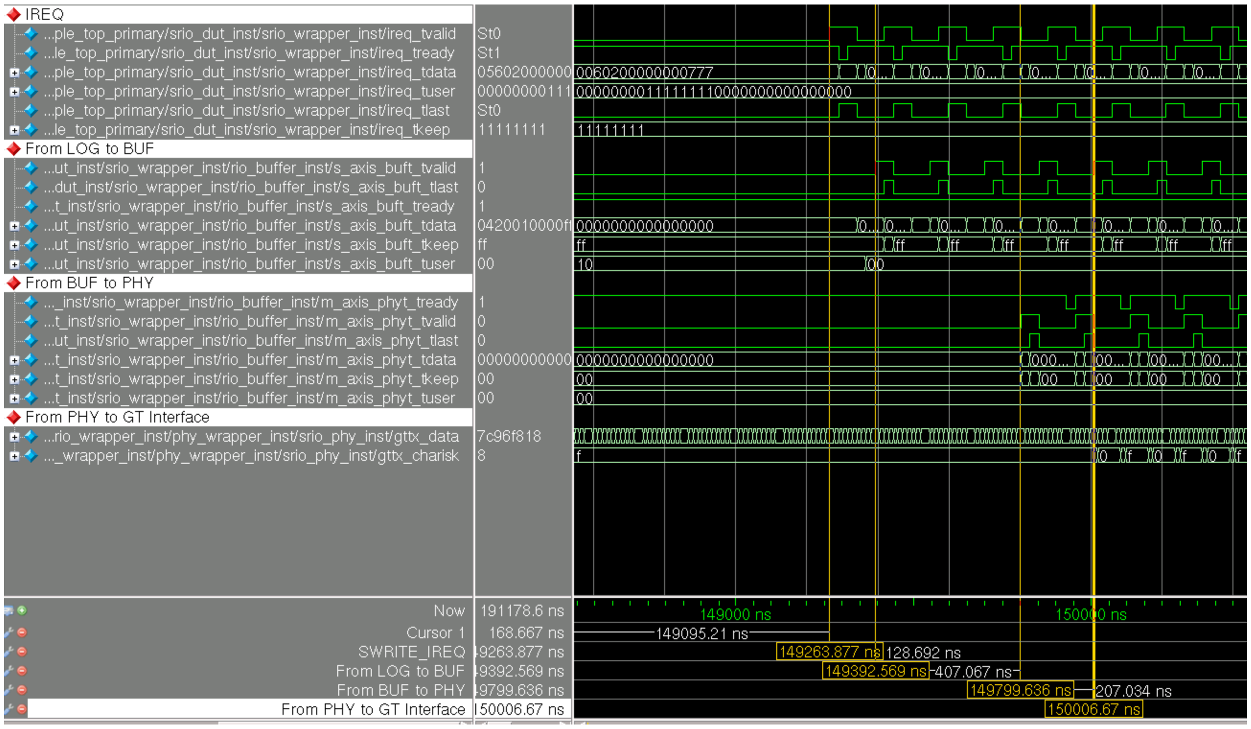


```
From BUF to PHY
   ..._inst/srio_wrapper_inst/rio_buffer_inst/m_axis_phyt_tready       1
   ...t_inst/srio_wrapper_inst/rio_buffer_inst/m_axis_phyt_tvalid      0
   ...ut_inst/srio_wrapper_inst/rio_buffer_inst/m_axis_phyt_tlast      0
   ...t_inst/srio_wrapper_inst/rio_buffer_inst/m_axis_phyt_tdata       00000000000
   ...t_inst/srio_wrapper_inst/rio_buffer_inst/m_axis_phyt_tkeep       00
   ...t_inst/srio_wrapper_inst/rio_buffer_inst/m_axis_phyt_tuser       00
```

Figure C-12: **SWRITE Packet from BUF to PHY**

Figure C-13 shows a simulation capture of all three interfaces discussed. Figure C-11 and Figure C-12 show the same data "0070070000ff0046_0000000000000000" being passed from the Transport interface to Link interface of the BUF.

The BUF transmits the packet coming from the LOG to the PHY. The PHY adds PHY layer info and sends the packet to the GT. Figure C-14 shows the SWRITE packet traveling from the IREQ to the GT interface. The four cursors in the waveform show the start of the packet in each interface.

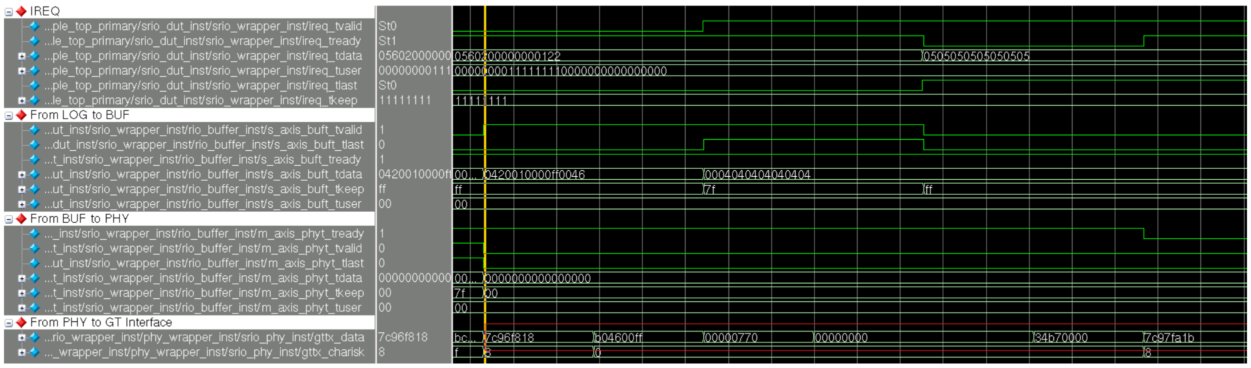


```
IREQ
  ...ple_top_primary/srio_dut_inst/srio_wrapper_inst/ireq_tvalid        St0
  ...le_top_primary/srio_dut_inst/srio_wrapper_inst/ireq_tready         St1
  ...ple_top_primary/srio_dut_inst/srio_wrapper_inst/ireq_tdata         05602000000
  ...ple_top_primary/srio_dut_inst/srio_wrapper_inst/ireq_tuser         00000000111
  ...ple_top_primary/srio_dut_inst/srio_wrapper_inst/ireq_tlast         St0
  ...le_top_primary/srio_dut_inst/srio_wrapper_inst/ireq_tkeep          11111111
From LOG to BUF
  ...ut_inst/srio_wrapper_inst/rio_buffer_inst/s_axis_buft_tvalid       1
  ...dut_inst/srio_wrapper_inst/rio_buffer_inst/s_axis_buft_tlast       0
  ...t_inst/srio_wrapper_inst/rio_buffer_inst/s_axis_buft_tready        1
  ...ut_inst/srio_wrapper_inst/rio_buffer_inst/s_axis_buft_tdata        0420010000ff
  ...ut_inst/srio_wrapper_inst/rio_buffer_inst/s_axis_buft_tkeep        ff
  ...ut_inst/srio_wrapper_inst/rio_buffer_inst/s_axis_buft_tuser        00
From BUF to PHY
  ..._inst/srio_wrapper_inst/rio_buffer_inst/m_axis_phyt_tready         1
  ...t_inst/srio_wrapper_inst/rio_buffer_inst/m_axis_phyt_tvalid        0
  ...ut_inst/srio_wrapper_inst/rio_buffer_inst/m_axis_phyt_tlast        0
  ...t_inst/srio_wrapper_inst/rio_buffer_inst/m_axis_phyt_tdata         00000000000
  ...t_inst/srio_wrapper_inst/rio_buffer_inst/m_axis_phyt_tkeep         00
  ...t_inst/srio_wrapper_inst/rio_buffer_inst/m_axis_phyt_tuser         00
From PHY to GT Interface
  ...rio_wrapper_inst/phy_wrapper_inst/srio_phy_inst/gttx_data          7c96f818
  ..._wrapper_inst/phy_wrapper_inst/srio_phy_inst/gttx_charisk          8

                    Now   191178.6 ns
               Cursor 1   168.667 ns
            SWRITE_IREQ   149263.877 ns
        From LOG to BUF   149392.569 ns
        From BUF to PHY   149799.636 ns
From PHY to GT Interface  150006.67 ns
```

Figure C-13: **SWRITE Packet from IREQ to GT Interface**

To view the packet on the GT interface, check `gttx_data`. As shown in Figure C-14, the entire packet transmitted from PHY to GT consists of:

7c96f818_ b04600ff_00000770_00000000_00000000_34b70000_7c97fa1b



```
IREQ
  ...ple_top_primary/srio_dut_inst/srio_wrapper_inst/ireq_tvalid        St0
  ...le_top_primary/srio_dut_inst/srio_wrapper_inst/ireq_tready         St1
  ...ple_top_primary/srio_dut_inst/srio_wrapper_inst/ireq_tdata         05602000000
  ...ple_top_primary/srio_dut_inst/srio_wrapper_inst/ireq_tuser         00000000111
  ...ple_top_primary/srio_dut_inst/srio_wrapper_inst/ireq_tlast         St0
  ...le_top_primary/srio_dut_inst/srio_wrapper_inst/ireq_tkeep          11111111
From LOG to BUF
  ...ut_inst/srio_wrapper_inst/rio_buffer_inst/s_axis_buft_tvalid       1
  ...dut_inst/srio_wrapper_inst/rio_buffer_inst/s_axis_buft_tlast       0
  ...t_inst/srio_wrapper_inst/rio_buffer_inst/s_axis_buft_tready        1
  ...ut_inst/srio_wrapper_inst/rio_buffer_inst/s_axis_buft_tdata        0420010000ff
  ...ut_inst/srio_wrapper_inst/rio_buffer_inst/s_axis_buft_tkeep        ff
  ...ut_inst/srio_wrapper_inst/rio_buffer_inst/s_axis_buft_tuser        00
From BUF to PHY
  ..._inst/srio_wrapper_inst/rio_buffer_inst/m_axis_phyt_tready         1
  ...t_inst/srio_wrapper_inst/rio_buffer_inst/m_axis_phyt_tvalid        0
  ...ut_inst/srio_wrapper_inst/rio_buffer_inst/m_axis_phyt_tlast        0
  ...t_inst/srio_wrapper_inst/rio_buffer_inst/m_axis_phyt_tdata         00000000000
  ...t_inst/srio_wrapper_inst/rio_buffer_inst/m_axis_phyt_tkeep         00
  ...t_inst/srio_wrapper_inst/rio_buffer_inst/m_axis_phyt_tuser         00
From PHY to GT Interface
  ...rio_wrapper_inst/phy_wrapper_inst/srio_phy_inst/gttx_data          7c96f818
  ..._wrapper_inst/phy_wrapper_inst/srio_phy_inst/gttx_charisk          8
```

Figure C-14: **SWRITE Packet Transmission on GT Interface**

The Start of Packet (SOP) is 7c96f818. 7c indicates the packet deliminator control symbol. The rest of the packet is decoded as follows (also illustrated in Figure C-15):

- 96f818 = 1001_0110_1111_1000_0001_1000
  - 100 = Status
  - 10110 = ackID 16

- 11111 = buf_status
- 000 = SOP
- 000 = cmd
- 11000 = CRC

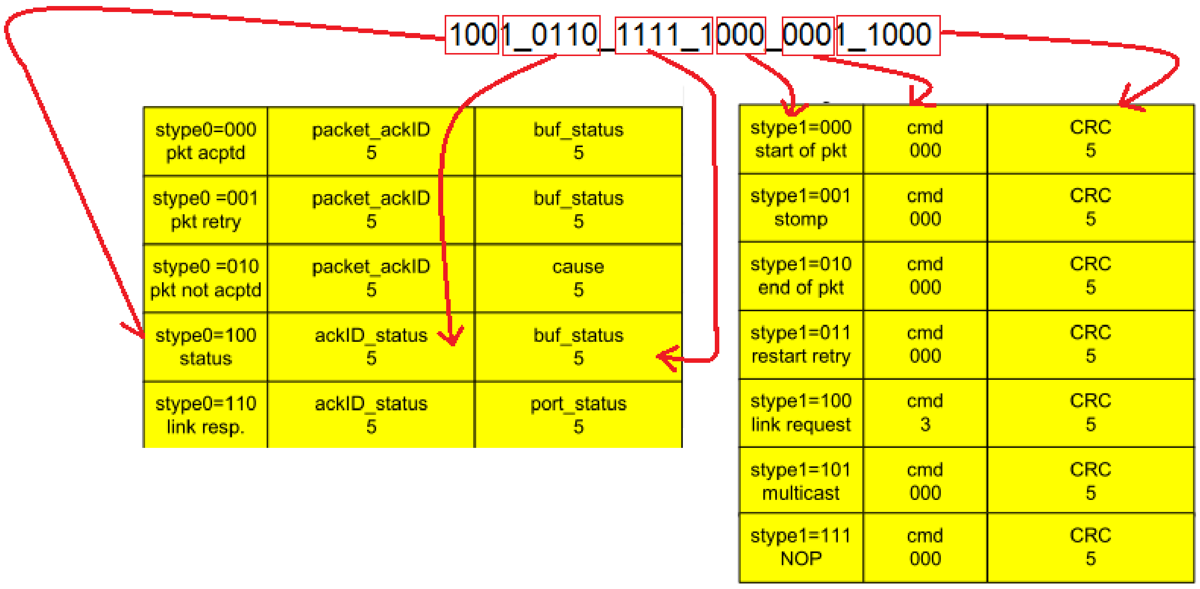


Figure C-15: **Packet Deliminator Control Symbol (SOP) Analysis**

The SWRITE packet = b04600ff 00000770 with write data of all 0s (64bits). This is decoded as follows (also illustrated in Figure C-16):

- b04600ff 00000770 = 1011_0000_0100_0110_0000_0000_1111_1111_0000_0000_0000_0000_0000_0111_0111 0000
    - ackID = 10110 (16)
    - rsvd = 00
    - crf = 0
    - prio = 01
    - tt = 00
    - FTYPE = 0110 (6, SWRITE)
    - Destination ID = 0000 0000
    - Source ID = 1111 1111

- Addr (29 bits) = 0000 0000 0000 0000 0000 0111 0111 0
- rsrv = 0
- xamsbs = 00

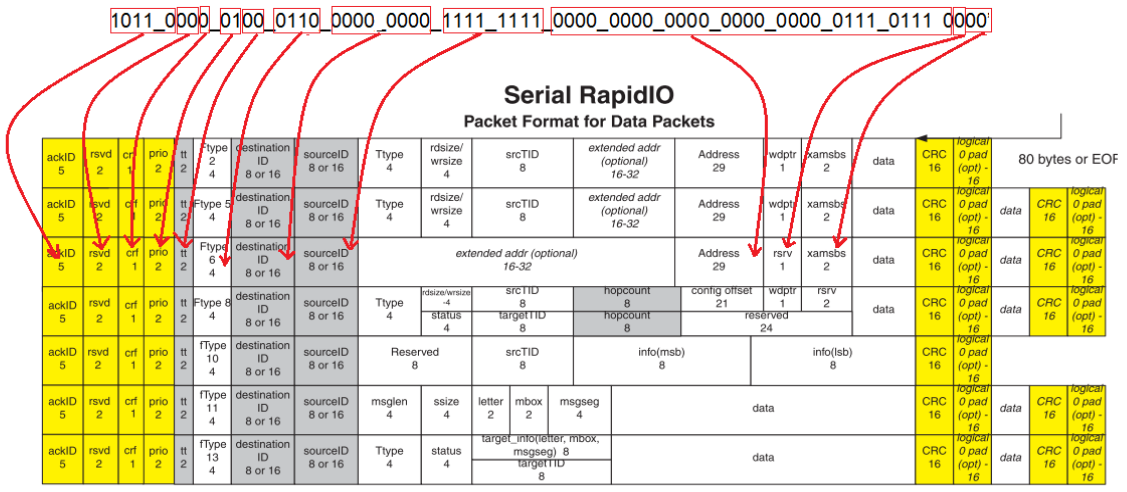
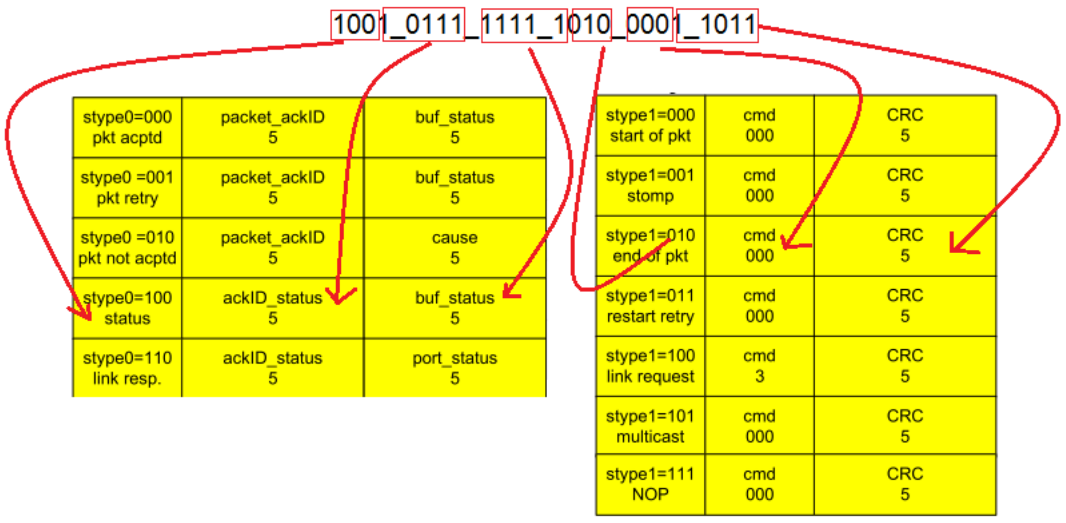


Figure C-16: **SWRITE Packet Header Field Analysis**

In this example, the End of Packet (EOP) = 7c97fa1b. 7c indicates the packet deliminator control symbol. This is decoded as follows (also illustrated in Figure C-17):

- 97f81c = 1001_0111_1111_1010_0001_1011
  - 100 = Status
  - 10111 = ackID 17
  - 11111 = buf_status
  - 010 = EOP
  - 000 = cmd
  - 11011 = CRC



stype0=000 pkt acptd	packet_ackID 5	buf_status 5
stype0 =001 pkt retry	packet_ackID 5	buf_status 5
stype0 =010 pkt not acptd	packet_ackID 5	cause 5
stype0=100 status	ackID_status 5	buf_status 5
stype0=110 link resp.	ackID_status 5	port_status 5

stype1=000 start of pkt	cmd 000	CRC 5
stype1=001 stomp	cmd 000	CRC 5
stype1=010 end of pkt	cmd 000	CRC 5
stype1=011 restart retry	cmd 000	CRC 5
stype1=100 link request	cmd 3	CRC 5
stype1=101 multicast	cmd 000	CRC 5
stype1=111 NOP	cmd 000	CRC 5

Figure C-17: **Packet Deliminator Control Symbol (EOP) Analysis**

Additional Resources

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see the Xilinx Support website at:

www.xilinx.com/support.

For a glossary of technical terms used in Xilinx documentation, see:

www.xilinx.com/support/documentation/sw_manuals/glossary.pdf.

References

These documents provide supplemental material:

1. UG761, *Xilinx AXI Reference Guide.*
2. UG911, *Vivado Design Suite Migration Methodology Guide*
3. Serial RapidIO Specifications, www.rapidio.org/specs/current.
4. *AMBA AXI4-Stream Protocol Specification*, infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0051a/index.html.
5. *AMBA AXI Protocol Specification*, infocenter.arm.com/help/topic/com.arm.doc.ihi0022c/index.html.

Technical Support

Xilinx provides technical support at www.xilinx.com/support for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation, or if changes are made to any section of the design labeled DO NOT MODIFY.

See the IP Release Notes Guide (XTP025) for more information on this core. For each core, there is a master Answer Record that contains the Release Notes and Known Issues list for the core being used. The following information is listed for each version of the core:

- New Features
- Resolved Issues
- Known Issues

Ordering Information

The Logical Layer (EF-DI-RIO-LOG) and the Physical Layer (EF-DI-RIO-PHY) IP cores are provided under the Xilinx Core License Agreement and can be generated using the Xilinx® CORE Generator™ system. The CORE Generator system is shipped with Xilinx ISE® Design Suite software.

A simulation evaluation license for the core is shipped with the CORE Generator system. To access the full functionality of the core, including FPGA bitstream generation, a full license must be obtained from Xilinx. For more information, visit the Serial RapidIO product page.

Contact your local Xilinx sales representative for pricing and availability of additional Xilinx LogiCORE IP modules and software. Information about additional Xilinx LogiCORE IP modules is available on the Xilinx IP Center.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/19/2011	1.0	Initial Xilinx release.
01/18/2012	2.0	Updated core to v1.3 and ISE Design Suite to v13.4. Added Appendix B, Migrating. Updated Table 2-1, Table 2-2, Software Assisted Error Recovery and other minor additions.
04/24/2012	3.0	Updated core to v1.4 and ISE Design Suite to v14.1. Combined PHY, BUF, and LOG modules into a single top-level wrapper as described in System Overview in Chapter 1.
07/25/2012	4.0	Added support for Vivado Design Suite. Added Link Initialization in Appendix C and SWRITE Packet Analysis in Appendix C.

Notice of Disclaimer