RapidIO MegaCore Function

User Guide

101 Innovation Drive
San Jose, CA 95134
www.altera.com

UG-MC_RIOPHY-2.16

Document last updated for Altera Complete Design Suite version: 12.0
Document publication date: June 2012

Feedback Subscribe

Contents

Chapter 4. Functional Description

Appendix D. Calculating Resource Utilization for Modular Configurations

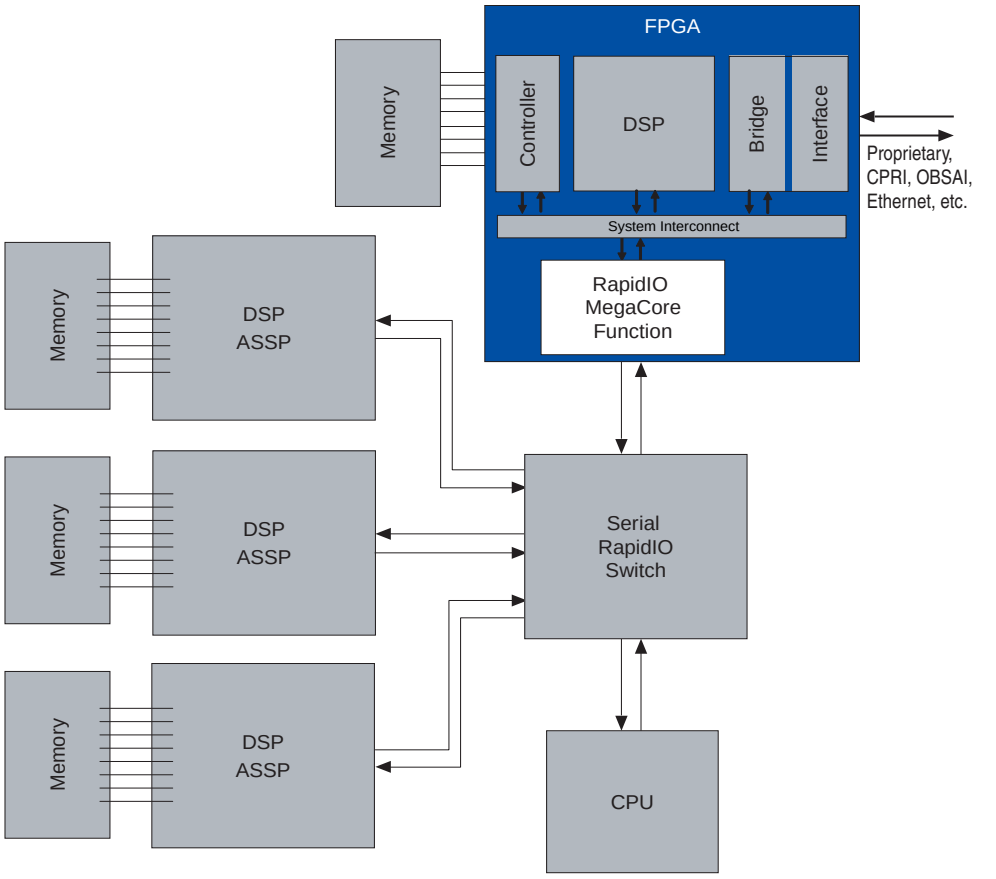
Additional Information

1. About This MegaCore Function

The RapidIO interconnect—an open standard developed by the RapidIO Trade Association—is a high-performance packet-switched interconnect technology designed to pass data and control information between microprocessors, digital signal processors (DSPs), communications and network processors, system memories, and peripheral devices.

The Altera® RapidIO MegaCore® function targets high-performance, multicomputing, high-bandwidth, and coprocessing I/O applications. Figure 1–1 shows an example system implementation.

Figure 1–1. Typical RapidIO Application



Features

This section outlines the features and supported transactions of the RapidIO IP core.

RapidIO IP Core Features

The RapidIO IP core has the following features:

- Compliant with RapidIO Trade Association, *RapidIO Interconnect Specification*, Revision 2.1, August 2009, available from the RapidIO Trade Association website at www.rapidio.org
- Successfully passed RIOLAB's Device Interoperability Level-3 (DIL-3) testing
- Supports 8-bit or 16-bit device IDs
- Supports incoming and outgoing multi-cast events
- **Physical layer features**
  - 1x/4x serial with integrated transceivers in selected device families and support for external transceivers in older device families
  - All four standard serial data rates supported: 1.25, 2.5, 3.125, and 5.0 gigabaud (Gbaud)
  - Receive/transmit packet buffering, flow control, error detection, packet assembly, and packet delineation
  - Automatic freeing of resources used by acknowledged packets
  - Automatic retransmission of retried packets
  - Scheduling of transmission, based on priority
  - Reset controller—fatal error does not require manual resetting
  - Optional automatic resetting of link partner after detection of fatal errors
  - Support for synchronizing with link partner's expected ackID after reset
  - Full control over integrated transceiver parameters
  - Configurable number of recovery attempts after link response time-out before declaring fatal error
- **Transport layer features**
  - Supports multiple Logical layer modules
  - A round-robin outgoing scheduler chooses packets to transmit from various Logical layer modules
- **Logical layer features**
  - Generation and management of transaction IDs
  - Automatic response generation and processing
  - Request to response time-out checking
  - Capability registers (CARs) and command and status registers (CSRs)
  - Direct register access, either remotely or locally
  - Maintenance master and slave Logical layer modules

- Input/Output Avalon® Memory-Mapped (Avalon-MM) master and slave Logical layer modules with burst support
- Avalon streaming (Avalon-ST) interface for custom implementation of message passing
- Doorbell module supporting 16 outstanding `DOORBELL` packets with time-out mechanism
- Support for preservation of transaction order between outgoing `DOORBELL` messages and I/O write requests
- New registers and interrupt indicate NWRITE_R transaction completion
- Support for preservation of transaction order between outgoing I/O read requests and I/O write requests from Avalon-MM interfaces

- SOPC Builder support
- IP functional simulation models for use in Altera-supported VHDL and Verilog HDL simulators
- Support for OpenCore Plus evaluation

Supported Transactions

The RapidIO IP core supports the following RapidIO transactions:

- `NREAD` request and response
- `NWRITE` request
- `NWRITE_R` request and response
- `SWRITE` request
- `MAINTENANCE` read request and response
- `MAINTENANCE` write request and response
- `MAINTENANCE` port-write request
- `DOORBELL` request and response

Device Family Support

Table 1–1 defines the device support levels for Altera IP cores.

Table 1–1. Altera IP Core Device Support Levels

FPGA Device Families	HardCopy Device Families
Preliminary support—The IP core is verified with preliminary timing models for this device family. The IP core meets all functional requirements, but might still be undergoing timing analysis for the device family. It can be used in production designs with caution.	**HardCopy Companion**—The IP core is verified with preliminary timing models for the HardCopy companion device. The IP core meets all functional requirements, but might still be undergoing timing analysis for the HardCopy device family. It can be used in production designs with caution.
Final support—The IP core is verified with final timing models for this device family. The IP core meets all functional and timing requirements for the device family and can be used in production designs.	**HardCopy Compilation**—The IP core is verified with final timing models for the HardCopy device family. The IP core meets all functional and timing requirements for the device family and can be used in production designs.

Table 1–2 shows the level of support offered by the Rapid IO IP core for each Altera device family.

Table 1–2. Device Family Support

Device Family	Support
Arria® GX	Final
Arria II GX	Final
Arria II GZ	Final
Arria V	Refer to the What's New in Altera IP page of the Altera website.
Cyclone® II	Final
Cyclone III	Final
Cyclone III LS	Final
Cyclone IV GX (1)	Final
Cyclone V	Refer to the What's New in Altera IP page of the Altera website.
HardCopy® II	HardCopy Compilation
HardCopy III	HardCopy Compilation
HardCopy IV E	HardCopy Compilation
HardCopy IV GX	HardCopy Compilation
Stratix® II	Final
Stratix II GX	Final
Stratix III	Final
Stratix IV	Final
Stratix IV GT	Final
Stratix V	Refer to the What's New in Altera IP page of the Altera website.
Other device families	No support

Note to Table 1–2:

(1) The RapidIO IP core supports only the EP4CGX50, EP4CGX75, EP4CGX110, and EP4CGX150 Cyclone IV GX devices.

IP Core Verification

Before releasing a version of the RapidIO IP core, Altera runs comprehensive regression tests in the current version of the Quartus® II software. These tests use the MegaWizard™ Plug-In Manager, SOPC Builder, and the Qsys system integration tool to create the instance files. These files are tested in simulation and hardware to confirm functionality.

Altera also performs interoperability testing to verify the performance of the IP core and to ensure compatibility with ASSP devices.

The RapidIO IP core v9.0 successfully passed RIOLAB's Device Interoperability Level-3 (DIL-3) testing in 2009.

Simulation Testing

Altera verifies the RapidIO IP core using the following industry-standard simulators:

- ModelSim® simulator
- VCS in combination with the Synopsys Native Testbench (NTB)

The test suite contains testbenches that use the RapidIO bus functional model (BFM) from the RapidIO Trade Association to verify the functionality of the IP core.

The regression suite tests various functions, including the following functionality:

- Link initialization
- Packet format
- Packet priority
- Error handling
- Throughput
- Flow control

Constrained random techniques generate appropriate stimulus for the functional verification of the IP core. Functional coverage metrics measure the quality of the random stimulus, and ensure that all important features are verified.

Hardware Testing

Altera tests and verifies the RapidIO IP core in hardware for different platforms and environments.

The hardware tests cover serial 1x and 4x variations running at 1.25, 2.5, 3.125, and 5.0 Gbaud, and processing the following traffic types:

- `NREAD`s of various size payloads—4 bytes to 256 bytes
- `NWRITE`s of various size payloads—4 bytes to 256 bytes
- `NWRITE_R`s of a few different size packets
- `SWRITE`s
- `Port-writes`
- `DOORBELL` messages

- MAINTENANCE reads and writes

The hardware tests also cover the following control symbol types:

- Status
- Packet-accepted
- Packet-retry
- Packet-not-accepted
- Start-of-packet
- End-of-packet
- Link-request, Link-response
- Stomp
- Restart-from-retry
- Multicast-event

Interoperability Testing

Altera performs interoperability tests on the RapidIO IP core, which certify that the RapidIO IP core is compatible with third-party RapidIO devices.

Altera performs interoperability testing with processors and switches from various manufacturers including:

- Texas Instruments Incorporated
- Integrated Device Technology, Inc. (IDT)

Testing of additional devices is an on-going process.

In addition, the RapidIO IP core v9.0 successfully passed RIOLAB's Device Interoperability Level-3 (DIL-3) testing in 2009.

Performance and Resource Utilization

This section contains tables showing IP core variation size and performance examples. Appendix D, Calculating Resource Utilization for Modular Configurations outlines ways you can reduce resource utilization to create smaller IP core variations.

Table 1–3 to Table 1–6 list the resources and expected performance for selected variations that use these modules:

- Physical layer with 8 KByte transmit buffers and 4 KByte receive buffers
- Transport layer
- Input/Output Avalon-MM master and slave

The numbers of LEs, combinational ALUTs, and logic registers in Table 1–3 to Table 1–6 are rounded up to the nearest 100.

Table 1–3 shows results obtained using the Quartus II software v11.1 for the following devices:

- Cyclone II (EP2C50F484C6)

- Cyclone III (EP3C55F780C6)
- Cyclone IV GX (EP4CGX50CF23C6)

Table 1–3. Serial RapidIO FPGA Resource Utilization I

Device	Parameters			LEs	Memory: M4K or M9K (1)
	Layers	Lane	Baud Rate (Gbaud)		
Cyclone II	Physical layer only	1×	3.125 with external SERDES	6,800	48
		4×	1.250 with external SERDES	10,000	55
	Physical and Transport layers, and I/O master and slave	1×	3.125 with external SERDES	11,800	91
		4×	1.250 with external SERDES	15,100	97
Cyclone III	Physical layer only	1×	3.125 with external SERDES	6,600	35
		4×	1.250 with external SERDES	9,700	41
	Physical and Transport layers, and I/O master and slave	1×	3.125 with external SERDES	11,300	70
		4×	1.250 with external SERDES	14,600	77
Cyclone IV GX	Physical layer only	1×	3.125	7,600	33
		4×	2.500	10,200	32
	Physical and Transport layers, and I/O master and slave	1×	3.125	13,300	63
		4×	2.500	16,500	63

Note to Table 1–3:

(1) M4K for Cyclone II devices and M9K for Cyclone III and Cyclone IV devices.

Table 1–4 shows results obtained using the Quartus II software v11.1 for a Cyclone V device (5CGXFC7C6F23C6).

Table 1–4. Serial RapidIO FPGA Resource Utilization II (Cyclone V Device)

Device	Parameters			Combinational ALUTs	Logic Registers	Memory	
	Layers	Mode	Baud Rate (Gbaud)			M10K	Memory ALUTs
Cyclone V	Physical layer only	1x	3.125	3,700	3,900	30	0
		4x	2.5	5,000	5,600	31	8
	Physical and Transport layers, and I/O master and slave	1x	3.125	7,000	7,400	60	0
		4x	2.5	8,300	9,600	61	0

Table 1–5 shows results obtained using the Quartus II software v11.1 for the following devices:

- Arria GX (EP1AGX60DF780C6)
- Arria II GX (EP2AGX65DF25C4)
- Arria II GZ (EP2AGZ225FF35C3)
- Arria V (5AGXBB1D4F31C4)

Table 1–5. Serial RapidIO FPGA Resource Utilization III

Device	Parameters			Combinational ALUTs	Logic Registers	Memory	
	Layers	Mode	Baud Rate (Gbaud)			M4K, M9K, or M10K (1)	M512 or Memory ALUT (2)
Arria GX	Physical layer only	1x	3.125	3,700	4,000	37	9
		4x	2.5	5,100	5,600	36	9
	Physical and Transport layers, and I/O master and slave	1x	3.125	6,300	7,100	78	11
		4x	2.5	7,600	9,200	74	13
Arria II GX	Physical layer only	1x	3.125	3,700	3,900	33	0
		4x	3.125	5,400	5,800	32	0
	Physical and Transport layers, and I/O master and slave	1x	3.125	7,100	7,400	63	0
		4x	3.125	8,200	9,600	63	0
Arria II GZ	Physical layer only	1x	5.00	3,700	4,000	29	20
		4x	3.125	5,500	6,000	29	38
	Physical and Transport layers, and I/O master and slave	1x	5.00	7,100	7,600	54	74
		4x	3.125	8,600	9,800	56	50
Arria V	Physical layer only	1x	5.00	3,600	4,000	29	0
		4x	3.125	5,000	6,300	31	0
	Physical and Transport layers, and I/O master and slave	1x	5.00	7,100	7,800	59	0
		4x	3.125	8,100	10,400	61	0

Note to Table 1–5:

(1) M4K for Arria GX devices; M9K for Arria II GX and Arria II GZ devices; and M10K for Arria V devices.

(2) M512 for Arria GX devices; Memory ALUT for Arria II GX, Arria II GZ, and Arria V devices.

Table 1–6 shows results obtained using the Quartus II software v11.1 for the following devices:

- Stratix II (EP2S30F672C3)
- Stratix II GX (EP2SGX30DF780C3)
- Stratix III (EP3SE260F1517C2)
- Stratix IV GX (EP4SGX230DF29C2)
- Stratix V (5SGXMA7H2F35C2)

Table 1–6. Serial RapidIO FPGA Resource Utilization IV (Part 1 of 2)

Device	Parameters			Combinational ALUTs	Logic Registers	Memory	
	Layers	Mode	Baud Rate (Gbaud)			M4K, M9K, or M20K (1)	M512 or Memory ALUT (2)
Stratix II	Physical layer only	1x	3.125 with external SERDES	3,700	3,900	38	10
		4x	3.125 with external SERDES	5,600	6,200	44	10
	Physical and Transport layers, and I/O master and slave	1x	3.125 with external SERDES	6,200	7,000	80	11
		4x	3.125 with external SERDES	8,100	9,600	82	14
Stratix II GX	Physical layer only	1x	3.125	3,700	4,000	37	9
		4x	3.125	5,300	5,700	35	10
	Physical and Transport layers, and I/O master and slave	1x	3.125	6,200	7,100	79	10
		4x	3.125	7,500	9,200	73	14
Stratix III	Physical layer only	1x	3.125 with external SERDES	3,800	4,000	32	42
		4x	3.125 with external SERDES	6,200	7,100	35	112
	Physical and Transport layers, and I/O master and slave	1x	3.125 with external SERDES	6,200	7,200	63	144
		4x	3.125 with external SERDES	8,800	11,200	62	403
Stratix IV GX	Physical layer only	1x	3.125	3,700	4,000	27	24
		4x	3.125	5,200	5,900	25	38
	Physical and Transport layers, and I/O master and slave	1x	3.125	7,100	7,600	51	87
		4x	3.125	9,000	10,300	51	83

Table 1–6. Serial RapidIO FPGA Resource Utilization IV (Part 2 of 2)

Device	Parameters			Combinational ALUTs	Logic Registers	Memory	
	Layers	Mode	Baud Rate (Gbaud)			M4K, M9K, or M20K (1)	M512 or Memory ALUT (2)
Stratix V GX	Physical layer only	1x	5.00	3,800	4,200	20	22
		4x	5.00	5,300	6,600	20	36
	Physical and Transport layers, and I/O master and slave	1x	5.00	7,000	7,800	41	96
		4x	5.00	8,600	10,700	45	90

Note to Table 1–5:

(1) M4K for Stratix II and Stratix II GX devices; M9K for Stratix III and Stratix IV devices; and M20K for Stratix V devices.

(2) M512 for Stratix II and Stratix II GX devices; Memory ALUT for Stratix III, Stratix IV GX, and Stratix V devices.

Table 1–7 shows the recommended device family speed grades for the supported link widths and internal clock frequencies. In all cases, Altera recommends that you set **Quartus II Analysis & Synthesis Optimization Technique** to **Speed**.

For information about how to apply the **Speed** setting, refer to volume 1 of the *Quartus II Handbook*.

Table 1–7. Recommended Device Family and Speed Grades (1) (Part 1 of 2)

Device Family	Mode	1x				4x			
	Rate	1.25 Gbaud	2.5 Gbaud	3.125 Gbaud	5.0 Gbaud	1.25 Gbaud	2.5 Gbaud	3.125 Gbaud	5.0 Gbaud
	f_{MAX}	31.25 MHz	62.50 MHz	78.125 MHz	125 MHz	62.5 MHz	125 MHz	156.25 MHz	250 MHz
Arria GX (2)		-6	-6	-6	(3)	-6	-6 (4)	(3)	(3)
Arria II GX		-4, -5, -6	-4, -5, -6	-4, -5, -6	(3)	-4, -5, -6	-4, -5	-4, -5	(3)
Arria II GZ		-3, -4	-3, -4	-3, -4	-3	-3, -4	-3, -4	-3, -4	(3)
Arria V		C4, -5, C6	C4, -5, C6	C4, -5, C6	C4 (5)	C4, -5, C6	C4, -5	C4 (5)	(3)
Stratix II, Stratix II GX		-3, -4, -5	-3, -4, -5	-3, -4, -5	(3)	-3, -4, -5	-3, -4	-3 (6)	(3)
Stratix III		-2, -3, -4	-2, -3, -4	-2, -3, -4	(3)	-2, -3, -4	-2, -3, -4	-2, -3	(3)
Stratix IV		-2, -3, -4	-2, -3, -4	-2, -3, -4	-2, -3, -4	-2, -3, -4	-2, -3, -4	-2, -3, -4	-2, -3 (7)
Stratix V		C1, -2, -3, -4	C1, -2, -3, -4	C1, -2, -3, -4	C1, -2, -3, -4	C1, -2, -3, -4	C1, -2, -3, -4	C1, -2, -3, -4	C1, -2, -3 (7)
Cyclone II, Cyclone III		-6, -7, -8	-6, -7, -8	-6, -7	(3)	-6, -7, -8	(3)	(3)	(3)
Cyclone IV GX (8)		-6, -7, -8	-6, -7, -8	-6, -7	(3)	-6, -7, -8	-6 (9)	(3)	(3)

Table 1–7. Recommended Device Family and Speed Grades (1) (Part 2 of 2)

Device Family	Mode	1x				4x			
	Rate	1.25 Gbaud	2.5 Gbaud	3.125 Gbaud	5.0 Gbaud	1.25 Gbaud	2.5 Gbaud	3.125 Gbaud	5.0 Gbaud
	f_{MAX}	31.25 MHz	62.50 MHz	78.125 MHz	125 MHz	62.5 MHz	125 MHz	156.25 MHz	250 MHz
Cyclone V (10)		C6, -7, C8	C6, -7, C8	C6, -7, C8	C6 (11)	C6, -7, C8	-6	(3)	(3)

Notes to Table 1–7:

(1) In this table, the entry -n indicates that both the industrial speed grade In and the commercial speed grade Cn are supported for this device family, RapidIO mode, and baud rate.

(2) Only the -6 speed grade is available for the Arria GX device family.

(3) Not supported for this device family.

(4) Altera does not recommend implementation of 4x RapidIO IP core variations with lane speeds of 2.5 Gbaud in the smallest member of the Arria GX device family (EP1AGX20). For other devices in the Arria GX family, you can use the Design Space Explorer in the Quartus II software to find the optimal Fitter settings for your design to meet the timing constraints. Following the Timing Advisor's recommendations, including optimizing for speed and using LogicLock regions may be necessary to meet timing, especially for more complex variations.

(5) Some simple Arria V 1× variations with lane speed of 5.0 Gbaud, and some simple Arria V 4× variations with lane speeds of 3.125 Gbaud, such as physical-layer-only variations, may meet timing in -5 speed grade devices, after following the Timing Advisor's recommendations.

(6) 4x 3.125 Gbaud is possible in a -4 speed grade Stratix II and Stratix II GX device only with the smallest Rx and Tx buffer sizes.

(7) Altera recommends that for designs that include a 4× 5.0 Gbaud RapidIO IP core variation and that target a -3 speed grade Stratix IV GX or Stratix V device, you use multiple seeds in the Quartus II Design Space Explorer to find the optimal Fitter settings to meet the timing constraints. Following the Timing Advisor's recommendations, including optimizing for speed and using LogicLock regions may be necessary to meet timing, especially for more complex variations implemented in the largest devices.

(8) The RapidIO IP core supports only the EP4CGX50, EP4CGX75, EP4CGX110, and EP4CGX150 Cyclone IV GX devices.

(9) Some simple Cyclone IV GX 4× variations, such as physical-layer-only variations, may meet timing at 2.5 Gbaud in -7 speed grade devices, after following the Timing Advisor's recommendations.

(10) Only the -6 and -7 speed grades are available for Cyclone V GT devices.

(11) The RapidIO IP core supports 1× 5.0 Gbaud variations that target the Cyclone V device family in Cyclone V GT devices only. The RapidIO parameter editor does not warn you of this fact. You can generate a 1× 5.0 Gbaud variation that targets a Cyclone V GX variation, for example, but when you attempt to add the extra constraints required for the RapidIO IP core, as discussed in "Specifying Constraints" on page 2–12, the Quartus II software Analysis and Synthesis tool fails.

Release Information

Table 1–8 provides information about this release of the RapidIO IP core.

Table 1–8. RapidIO Release Information

Item	Description
Version	12.0
Release Date	June 2012
Ordering Code	IP-RIOPHY
Product ID	0095
Vendor ID	6AF7

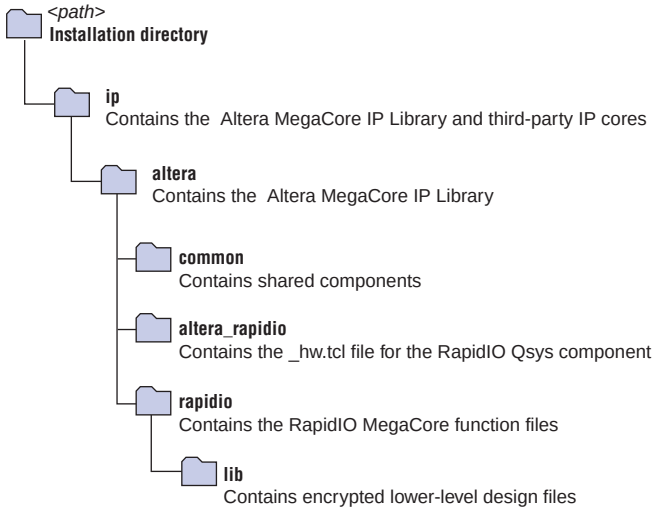
Altera verifies that the current version of the Quartus II software compiles the previous version of each IP core. Any exceptions to this verification are reported in the *MegaCore IP Library Release Notes and Errata*. Altera does not verify compilation with IP core versions older than the previous release.

Installation and Licensing

The RapidIO IP core is part of the Altera MegaCore IP Library, which is distributed with the Quartus II software and downloadable from the Altera website, www.altera.com.

Figure 1–2 shows the directory structure after you install the RapidIO IP core, where <*path*> is the installation directory. The default installation directory on Windows is **C:\altera**<*version number*>; on Linux it is **/opt/altera**<*version number*>**.**

Figure 1–2. Directory Structure



You can use Altera's free OpenCore Plus evaluation feature to evaluate the IP core in simulation and in hardware before you purchase a license. You must purchase a license for the IP core only when you are satisfied with its functionality and performance, and you want to take your design to production.

After you purchase a license for the RapidIO IP core, you can request a license file from the Altera website at **www.altera.com/licensing** and install it on your computer. When you request a license file, Altera emails you a **license.dat** file. If you do not have internet access, contact your local Altera representative.

OpenCore Plus Evaluation

With the Altera free OpenCore Plus evaluation feature, you can perform the following actions:

- Simulate the behavior of a megafunction (Altera IP core or AMPP^{SM} megafunction) in your system using the Quartus II software and Altera-supported VHDL and Verilog HDL simulators.
- Verify the functionality of your design and evaluate its size and speed quickly and easily.
- Generate time-limited device programming files for designs that include IP cores.
- Program a device and verify your design in hardware.

OpenCore Plus Time-Out Behavior

OpenCore Plus hardware evaluation supports the following two operation modes:

- *Untethered*—the design runs for a limited time.
- *Tethered*—requires a connection between your board and the host computer. If tethered mode is supported by all megafunctions in a design, the device can operate for a longer time or indefinitely.

All megafunctions in a device time out simultaneously when the most restrictive evaluation time is reached. If there is more than one megafunction in a design, a specific megafunction's time-out behavior may be masked by the time-out behavior of the other megafunctions.

For Altera IP cores, the untethered time-out is 1 hour; the tethered time-out value is indefinite.

Your design stops working after the hardware evaluation time expires.

The RapidIO IP core then behaves as if its Atlantic™ interface signals `atxena` and `arxena` are tied low. All packet transfers through the Physical layer are suppressed. As a result, the RapidIO IP core cannot transmit new packets (it only transmits the idle sequence and status control symbols), and cannot read packets from the Physical layer. If the remote link partner continues to transmit packets, the RapidIO IP core refuses new packets by sending `packet_retry` control symbols after its receiver buffer fills up beyond the corresponding threshold.

For Information About	Refer To
Installation and licensing	Altera Software Installation and Licensing
Open Core Plus	*AN 320: OpenCore Plus Evaluation of Megafunctions*

Design Flows

You can customize the RapidIO IP core to support a wide variety of applications. You can instantiate this IP core in the MegaWizard Plug-In Manager, in the Qsys system integration tool, or in SOPC Builder.

The MegaWizard Plug-In Manager flow offers the following advantages:

- Allows you to parameterize the IP core to create a variation that you can instantiate manually in your design.

The SOPC Builder flow offers the following advantages:

- Allows you to integrate other Altera-provided custom components such as DMA controllers, on-chip memories, and FIFOs in your design.
- Uses Avalon-MM interfaces.

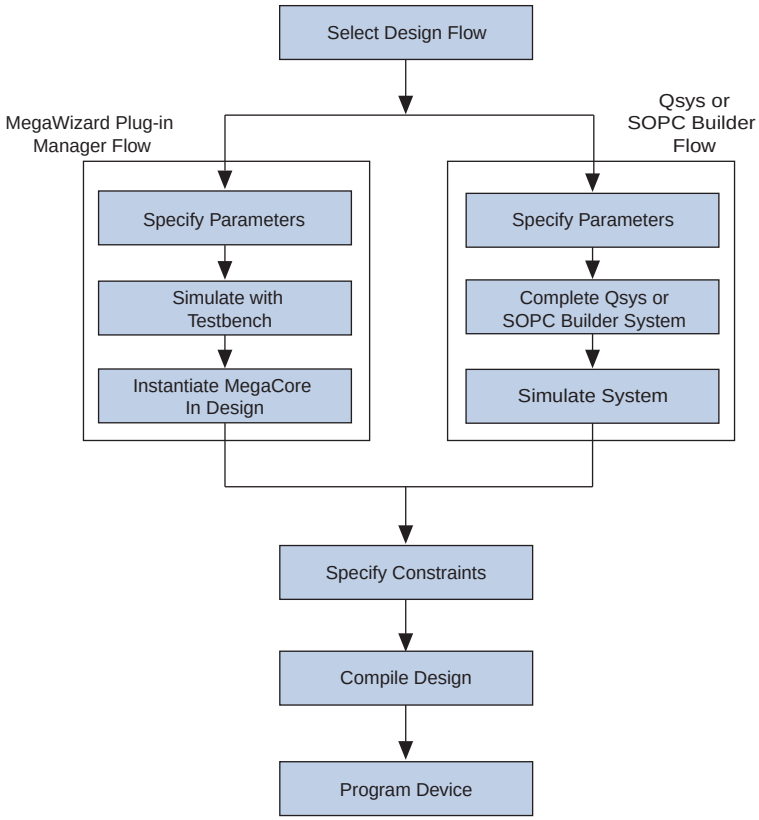
The Qsys flow offers the following additional advantages over SOPC Builder:

- Provides visualization of hierarchical designs.
- Allows greater customization of interconnect elements and pipelining.
- Provides closer integration with the Quartus II software.

RapidIO variations that target an Arria II GZ, Arria V, Cyclone V, or Stratix V device are supported in the MegaWizard Plug-In Manager and the Qsys flows only.

Figure 2–1 shows the stages for creating a system with the RapidIO IP core and the Quartus II software. Each stage is described in detail in subsequent sections.

Figure 2–1. RapidIO Design Flow



MegaWizard Plug-In Manager Design Flow

You can use the MegaWizard Plug-In Manager in the Quartus II software to parameterize a custom IP core variation. When you select the RapidIO IP core in the MegaWizard Plug-In Manager, the RapidIO parameter editor appears. The RapidIO parameter editor lets you interactively set parameter values and select optional ports. This flow is best for manual instantiation of an IP core in your design.

SOPC Builder Design Flow

The SOPC Builder design flow enables you to integrate a RapidIO endpoint in an SOPC Builder system. The SOPC Builder design flow automatically connects selected components with the system interconnect, eliminating the requirement to design low-level interfaces and significantly reducing design time. When you add a RapidIO IP core instance to your design, a RapidIO parameter editor guides you in selecting the properties of the RapidIO IP core instance.

Qsys Design Flow

The Qsys design flow enables you to integrate a RapidIO endpoint in a Qsys system. The Qsys design flow allows you to connect component interfaces with the system interconnect, eliminating the requirement to design low-level interfaces and significantly reducing design time. When you add a RapidIO IP core instance to your design, a RapidIO parameter editor guides you in selecting the properties of the RapidIO IP core instance.

MegaWizard Plug-In Manager Design Flow

The MegaWizard Plug-In Manager flow allows you to customize the RapidIO IP core, and manually integrate the function in your design.

Specifying Parameters

To specify RapidIO IP core parameters using the MegaWizard Plug-In Manager, follow these steps:

1. Create a Quartus II project using the **New Project Wizard** available from the File menu.
2. Launch the **MegaWizard Plug-In Manager** from the Tools menu, and follow the prompts in the MegaWizard Plug-In Manager interface to create a custom megafunction variation.

   To select the RapidIO IP core, click **Installed Plug-Ins > Interfaces > RapidIO**.

3. Specify the parameters on all pages in the **Parameter Settings** tab. For details about these parameters, refer to Chapter 3, Parameter Settings.
4. If you want to generate an IP functional simulation model for the IP core in the selected language, on the **EDA** tab, turn on **Generate simulation model**.

   The IP functional simulation model is a cycle-accurate VHDL or Verilog HDL model produced by the Quartus II software.

CAUTION: Use the simulation models only for simulation and not for synthesis or any other purposes. Using these models for synthesis creates a nonfunctional design.

5. Some third-party synthesis tools can use a netlist that contains the structure of this IP core but no detailed logic to optimize timing and performance of the design containing it.

   To use this feature if your synthesis tool supports it, turn on **Generate netlist**.

6. On the **Summary** tab, select the files you want to generate. A gray checkmark indicates a file that is automatically generated. All other files are optional.

   For more information about the files generated in your project directory, refer to the project files list in the HTML report file in your project directory.

7. Click **Finish** to generate the IP core and supporting files.

   You may have to wait several minutes for file generation to complete, especially if you are generating an IP functional simulation model.

8. If you generate the RapidIO IP core instance in a Quartus II project, you are prompted to add the Quartus II IP File (**.qip**) to the current Quartus II project. You can also turn on **Automatically add Quartus II IP Files to all projects**.

   The **.qip** is generated by the parameter editor, and contains information about the generated IP core. In most cases, the **.qip** contains all of the necessary assignments and information required to process the IP core or system in the Quartus II compiler. The MegaWizard Plug-In Manager generates a single **.qip** for each IP core.

9. After you review the generation report (**<***variation name***>.html**) in your project directory, click **Exit** to close the MegaWizard Plug-In Manager.

You can now integrate your custom IP core variation in your design, simulate, and compile.

When you integrate your RapidIO IP core variation in your design, note the connection and I/O assignment requirements described in "Completing the Qsys System" on page 2–10.

Simulating the Design

You can simulate your RapidIO IP core variation using the IP functional simulation model and the Verilog HDL demonstration testbench. The IP functional simulation model and testbench files are generated in your project directory. The directory also includes scripts to compile and run the demonstration testbench. The testbench demonstrates how to instantiate a model in a design and includes some simple stimulus to control the user interfaces of the RapidIO interface.

For information about the demonstration testbench, refer to Chapter 7, Testbenches.

If you specify VHDL for your RapidIO IP core, the IP functional simulation model is in VHDL, but the testbench is in Verilog HDL. Therefore, you must have a license to run mixed language simulations to run the testbench with the VHDL model. Alternatively, you can run simulation in a VHDL-only environment. In a VHDL-only environment, you must create your own test environment.

To simulate your MegaWizard Plug-In Manager flow generated RapidIO IP core variation with the IP functional simulation model and the Verilog HDL demonstration testbench, using the Mentor Graphics ModelSim simulator, perform the following steps:

1. Start the ModelSim simulator.
2. In ModelSim, change directory to your project directory.
3. Type the following command to set up the required libraries, compile the generated IP Functional simulation model, and exercise the simulation model with the provided testbench:

   ```
   do <variation>_run_modelsim.tcl ↵
   ```

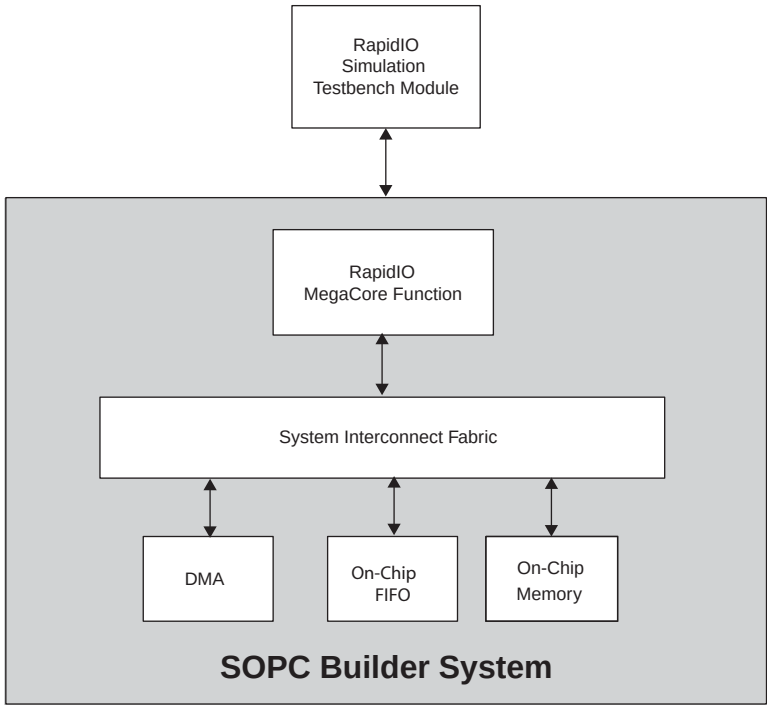
For Information About	Refer To
Quartus II software	See the Quartus II Help topics:
MegaWizard Plug-In Manager	"About the Quartus II Software" "About the MegaWizard Plug-In Manager"
A complete list of models or libraries required to simulate the RapidIO IP core	*<variation name>*_**run_modelsim.tcl** script provided with the demonstration testbench in Chapter 7, Testbenches
IP functional simulation models	*Simulating Altera Designs* chapter in volume 3 of the *Quartus II Handbook*

SOPC Builder Design Flow

You can use SOPC Builder to build a system that contains your customized RapidIO IP core. You can easily add other components and quickly create an SOPC Builder system. SOPC Builder automatically generates HDL files that include all of the specified components and interconnections. The HDL files are ready to be compiled by the Quartus II software to produce output files for programming an Altera device. SOPC Builder also generates a simulation testbench module that includes basic

transactions to validate the HDL files. Figure 2–2 shows a block diagram of an example SOPC Builder system.

Figure 2–2. SOPC Builder System



For Information About	Refer To
System interconnect fabric	*System Interconnect Fabric for Memory-Mapped Interfaces* and *System Interconnect Fabric for Streaming Interfaces* chapters in the *SOPC Builder User Guide* and the *Avalon Interface Specifications*
SOPC Builder	*SOPC Builder User Guide*
Quartus II software	Quartus II Help

Specifying Parameters

To specify RapidIO parameters using the SOPC Builder flow, follow these steps:

1. Create a new Quartus II project using the **New Project Wizard** available from the File menu.
2. On the Tools menu, click **SOPC Builder**.
3. For a new system, specify the system name and language.
4. On the **System Contents** tab, double-click **RapidIO** to add it to your system. The RapidIO parameter editor appears.

You can find **RapidIO** by expanding **Interface Protocols** >**High Speed** > **RapidIO**.

5. Specify the required parameters on all pages in the **Parameter Settings** tab of the RapidIO parameter editor in SOPC Builder. For detailed explanations of these parameters, refer to Chapter 3, Parameter Settings.
6. Click **Finish** to complete the RapidIO IP core and add it to the system.

Completing the SOPC Builder System

To complete the SOPC Builder system, follow these steps:

1. Add and parameterize any additional components. For a complete SOPC Builder system design example containing the RapidIO IP core, refer to Chapter 8, SOPC Builder Design Example.
2. Connect the components using the Connection panel on the **System Contents** tab.

   For Arria GX, Arria II GX, Cyclone IV GX, Stratix II GX, and Stratix IV GX designs, ensure that you connect the calibration clock (`cal_blk_clk`) to a clock signal with the appropriate frequency range of 10 to 125 MHz. The `cal_blk_clk` ports on other components that use transceivers must be connected to the same clock signal.

3. By default, SOPC Builder does not display clock names. To display clock names in the **Module Name** column and the clocks in the **Clock** column in the **System Contents** tab, click **Filters** to display the **Filters** dialog box. In the **Filter** list, click **All Interfaces**.

For Arria II GX, Cyclone IV GX, and Stratix IV GX designs with high-speed transceivers, you must add a dynamic reconfiguration block (`altgx_reconfig`) and connect it as specified in the *Arria II Device Handbook*, *Cyclone IV Device Handbook*, or *Stratix IV Device Handbook*. This block supports offset cancellation. The design compiles without the `altgx_reconfig` block, but it cannot function correctly in hardware.

4. If you intend to simulate your SOPC builder system, on the **System Generation** tab, turn on **Simulation** to generate a functional simulation model for your system.
5. Click **Generate** to generate the system.

Among the files generated by SOPC Builder is the **.qip** file. This file contains information about a generated IP core or system. In most cases, the **.qip** file contains all of the necessary assignments and information required to process the IP core or system in the Quartus II compiler. Generally, a single **.qip** file is generated for each SOPC Builder system. However, some more complex SOPC Builder components generate a separate **.qip** file. In that case, the system **.qip** file references the component **.qip** file.

6. For Arria II GX and Stratix IV GX designs, after you generate the system, you must create assignments for the high-speed transceiver VCCH settings by following these steps:
   a. In the Quartus II window, on the Assignments menu, click **Assignment Editor**.
   b. In the **<<new>>** cell in the **To** column, type the top-level signal name for your RapidIO IP core instance `td` signal. The default signal name that SOPC Builder generates is `td_rapidio`.
   c. Double-click in the **Assignment Name** column and click **I/O Standard**.
   d. Double-click in the **Value** column and click your standard (for example, **1.5-V PCML)**.
   e. In the new **<<new>>** row, repeat steps b to d for your RapidIO IP core instance `rd` signal.

Simulating the System

During system generation, SOPC Builder optionally generates an IP functional simulation model and testbench for the entire system which you can use to simulate your system easily in any Altera-supported simulation tool. SOPC Builder also generates a set of ModelSim Tcl scripts and macros that you can use to simulate the testbench, IP functional simulation models, and clear text RTL design files that describe your system in the ModelSim simulation software.

In the SOPC Builder design flow, if you specify Verilog HDL as the target HDL, a complete testbench is generated, but if you specify VHDL, only a link loopback module is generated.

By default, the testbench provided for Verilog HDL SOPC Builder systems initializes the link and reads some registers in the RapidIO IP core. The testbench provides tasks that you can use to create a test sequence specific to your SOPC Builder system. For an example SOPC Builder system with a more complete testbench, refer to Chapter 8, SOPC Builder Design Example.

To run the testbench SOPC Builder provides by default for a system in which the RapidIO IP core is generated in Verilog HDL, perform the following steps:

1. Start the ModelSim software.
2. On the File menu, change directory to the **<***SOPC Builder system***>_sim** subdirectory in your project directory.
3. Type the following command at the ModelSim command prompt:

   ```
   do setup_sim.do ↵
   ```

4. To compile all the files and load the design, type the following command at the ModelSim prompt:

   ```
   s ↵
   ```

5. To simulate the design, type the following command at the ModelSim prompt:

   ```
   run -all ↵
   ```

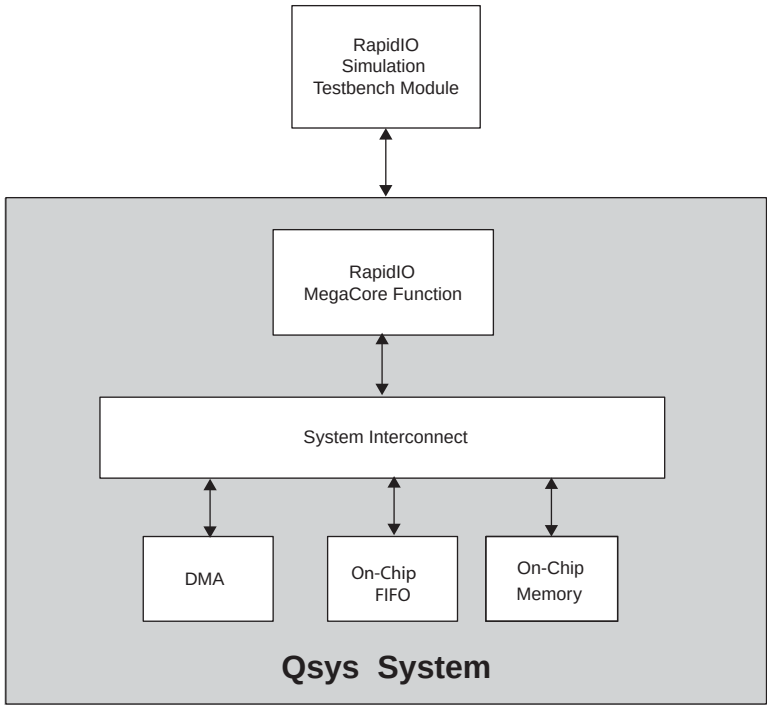
For more information about simulating SOPC Builder systems, refer to the *SOPC Builder User Guide* and *AN 351: Simulating Nios II Embedded Processor Designs*.

Qsys Design Flow

You can use Qsys to build a system that contains your customized RapidIO IP core. You can easily add other components and quickly create a Qsys system. Qsys can automatically generate HDL files that include all of the specified components and interconnections. The HDL files are ready to be compiled by the Quartus II software to produce output files for programming an Altera device.

Figure 2–2 shows a block diagram of an example Qsys system.

Figure 2–3. Qsys System



For Information About	Refer To
System interconnect	*Qsys Interconnect and System Design Components* chapter in volume 1 of the *Quartus II Handbook* and the *Avalon Interface Specifications*
Qsys	*System Design with Qsys* section in volume 1 of the *Quartus II Handbook*
Quartus II software	Quartus II Help

For information about converting an SOPC Builder system that includes a RapidIO IP core to a Qsys system, refer to Appendix C, Porting a RapidIO Design from the Previous Version of the Software.

Specifying Parameters

To specify RapidIO parameters using the Qsys flow, follow these steps:

1. Create a new Quartus II project using the **New Project Wizard** available from the File menu.
2. On the Tools menu, click **Qsys**.
3. On the **System Contents** tab, double-click **RapidIO** to add it to your system. The RapidIO parameter editor appears.

   You can find **RapidIO** by expanding **Interface Protocols** >**High Speed** > **RapidIO**.

4. Specify the required parameters on all tabs of the RapidIO parameter editor. For detailed explanations of these parameters, refer to Chapter 3, Parameter Settings.
5. Click **Finish** to complete the RapidIO IP core instance and add it to the Qsys system.

When you generate a RapidIO IP core in Qsys, you cannot modify the transceiver settings. After you generate the Qsys system you must make any modifications to the default values by editing the existing ALTGX megafunction in the MegaWizard Plug-In Manager.

Completing the Qsys System

To complete the Qsys system, follow these steps:

1. Add and parameterize any additional components.
2. Connect the components using the Connection panel on the **System Contents** tab.

   For Arria GX, Arria II GX, Arria II GZ, Cyclone IV GX, Stratix II GX, and Stratix IV GX designs, ensure that you connect the calibration clock (`cal_blk_clk`) to a clock signal with the appropriate frequency range of 10 to 125 MHz. The `cal_blk_clk` ports on other components that use transceivers must be connected to the same clock signal.

3. If some signals are not displayed, click the Filter icon to display the **Filters** dialog box. In the **Filter** list, click **All Interfaces**. Alternatively, if you right-click in the System Contents tab, a **Filter** menu option appears.

For Arria II GX, Arria II GZ, Cyclone IV GX, and Stratix IV GX designs with high-speed transceivers, you must add a dynamic reconfiguration block (`altgx_reconfig`) to your design using the MegaWizard Plug-In Manager. This block is not available in Qsys; you must generate it in the MegaWizard Plug-In Manager. You must connect it as specified in the *Arria II Device Handbook* or the *Stratix IV Device Handbook*. This block supports offset cancellation. The design compiles without the `altgx_reconfig` block, but it cannot function correctly in hardware.

For Arria V, Cyclone V, and Stratix V designs, you must add a dynamic reconfiguration block (Transceiver Reconfiguration Controller) to your design using the MegaWizard Plug-In Manager or Qsys, and connect it to the RapidIO IP core PHY IP reconfiguration signals. This block supports offset cancellation. The design compiles without the Transceiver Reconfiguration Controller, but it cannot function correctly in hardware.

For information about the Altera Transceiver Reconfiguration Controller, refer to the *Altera Transceiver PHY IP Core User Guide*.

For information about the number of reconfiguration interfaces you must configure in your Arria V, Cyclone V, or Stratix V dynamic reconfiguration block, refer to the descriptions of the `reconfig_togxb` and `reconfig_fromgxb` signals in Table 5–12 on page 5–8. An informational message in the RapidIO parameter editor tells you the required number of reconfiguration interfaces.

1. If you intend to simulate your Qsys system, on the **Generation** tab, turn on **Create simulation model** and select **Verilog** HDL or **VHDL** to generate a functional simulation model.
2. Click **Generate** to generate the system. Qsys generates the system and produces the <*system_name*>**.qip** file that contains the assignments and information required to process the IP core or system in the Quartus II Compiler.
3. In the Quartus II software, on the Project menu, click **Add/Remove Files in Project**.
4. In the **Settings** dialog box, under **Category**, highlight **Files**.
5. Browse to the **.qip** file and add it to your project.
6. For Arria II GX, Arria II GZ, and Stratix IV GX designs, after you generate the system, you must create assignments for the high-speed transceiver VCCH settings by following the instructions in step 6 on page 2–8.
7. If you want to modify the high-speed transceiver settings, you must edit the existing ALTGX megafunction in the MegaWizard Plug-In Manager.

Simulating the System

During system generation, Qsys optionally generates a RapidIO functional simulation model in the HDL you specify. In addition, you can simulate the static RapidIO link loopback module that is provided in the Verilog HDL. This static module is a design example located in <*Quartus II installation directory*>**\ip\altera\rapidio\lib\rio\qsys_cust_demo**. For information about this design example, refer to Chapter 9, Qsys Design Example.

The RapidIO IP core you generate in the Qsys flow is a hw.tcl-based IP core.

For information about simulating Qsys systems, refer to the *Creating a System with Qsys* chapter in volume 1 of the *Quartus II Handbook*.

Specifying Constraints

Altera provides constraint files in Tcl format that you must apply to ensure that the RapidIO IP core meets design timing requirements.

Constraints are not set automatically. You must run the Tcl constraint script to apply the constraints, in all three design flows.

To use the generated constraint files, follow these steps:

1. Open your Quartus II project in the Quartus II software.
2. On the View menu, point to **Utility Windows** and then click **Tcl Console**.
3. Source the generated constraint file by following one of these two steps:
   - In the MegaWizard Plug-In Manager flow or the SOPC Builder flow, type the following command at the Tcl console command prompt:

     `source <variation_name>_constraints.tcl ↵`

   - In the Qsys flow, type the following command at the Tcl console command prompt:

```
source \
<Qsys_system_name>/synthesis/submodules/<Qsys_system_name>_<instance_name>_constraints.tcl ↵
```

4. Add the Rapid IO constraints to your project by typing the following command at the Tcl console command prompt:

   `add_rio_constraints ↵`

   This command adds the necessary logic constraints to your Quartus II project.

   In the Qsys flow, you are likely to require the `-ref_clk_name`, `-sys_clk_name`, `-phy_mgmt_clk`, and `-patch_sdc` command-line options specified in Table 2–1.

The script automatically constrains the system clocks and the reference clock based on the data rate chosen. For supported transceivers, Altera recommends that you adjust the reference clock frequency in the **Physical Layer** tab of the RapidIO parameter editor only. However, you can adjust the system clock frequency in the Tcl constraints script or the generated Synopsys Design Constraint File (**.sdc**).

The Tcl script assumes that virtual pins and I/O standards are connected to Altera-provided pin names. For user-defined pin names, you must edit the script after generation to ensure that the assignments are made properly.

The `add_rio_constraints` command has the following additional options that you can use:

```
add_rio_constraints [-no_compile]
[-ref_clk_name <name>] [-sys_clk_name <name>] [-phy_mgmt_clk_name <name>]
[-patch_sdc] [-help]
```

Table 2–1 explains these options.

Table 2–1. add_rio_constraints Options

Constraint	Use
-no_compile	Use the -no_compile option to prevent analysis and synthesis. Use this option only if you performed analysis and synthesis or fully compiled your project prior to using this script. Using this option decreases turnaround time during development.
-ref_clk_name	The Rapid IO IP core has a top-level reference clock name (<*variation*>_clk in the Qsys flow by default, and ref_clk in the other flows). If, in your instantiation, you have connected the reference clock port of the IP core to a clock named something other than ref_clk (<*variation*>_clk in the Qsys flow), you must run the add_rio_constraints command with this option followed by the name of the clock connected to the reference clock port of the RapidIO IP core. The following example command illustrates the syntax: `add_rio_constraints -ref_clk_name CLK125`
-sys_clk_name	By default, the Avalon system clock name used for the RapidIO IP core is named clk_0. If you rename this clock in SOPC Builder, or you do not rename this clock to clk_0_clk_in in the Qsys system, or you connect the system clock to a clock named something other than clk_0 (or clk_0_clk_in_clk in the Qsys flow), you must run the add_rio_constraints command with this option followed by the updated clock name. The following example command illustrates the syntax: `add_rio_constraints -sys_clk_name CLK50`
-phy_mgmt_clk_name	This option is available only for RapidIO variations that target an Arria V, Cyclone V, or Stratix V device. By default, the PHY IP core management clock, which is present only in RapidIO variations that target an Arria V, Cyclone V, or Stratix V device, is named phy_mgmt_clk. If you rename this clock or you connect it to a clock named something other than <*variation*>_phy_mgmt_clk in the Qsys flow, you must run the add_rio_constraints command with this option followed by the updated clock name. The following example command illustrates the syntax: `add_rio_constraints -phy_mgmt_clk_name CLK_PHY_MGMT`
-patch_sdc	This option is only valid when used with the -ref_clk_name, -sys_clk_name, or -phy_mgmt_clk option. The -patch_sdc option patches the generated SDC script with the new clock names. A back-up copy of the SDC script is created before the patch is made, and any edits that were previously made to the SDC script are preserved.
-help	Use the -help option for information about the options used with the add_rio_constraints command.

For more information about timing analyzers, refer to the Quartus II Help and the *Timing Analysis* section in volume 3 of the *Quartus II Handbook*.

Compiling the Full Design and Programming the FPGA

You can use the **Start Compilation** command on the Processing menu in the Quartus II software to compile your design. After successfully compiling your design, program the targeted Altera device with the Programmer and verify the design in hardware.

Before compiling your design in the Quartus II software, you must apply the constraints as described in "Specifying Constraints" on page 2–12.

For Information About	Refer To
Compiling your design	*Quartus II Incremental Compilation for Hierarchical and Team-Based Design* chapter in volume 1 of the *Quartus II Handbook*
Programming the device	*Device Programming* section in *volume 3* of the *Quartus II Handbook*

Instantiating Multiple RapidIO IP Cores

If you want to instantiate multiple RapidIO IP cores, a few additional steps are required. The following sections outline these steps.

Clock and Signal Requirements for Arria V, Cyclone V, and Stratix V Devices

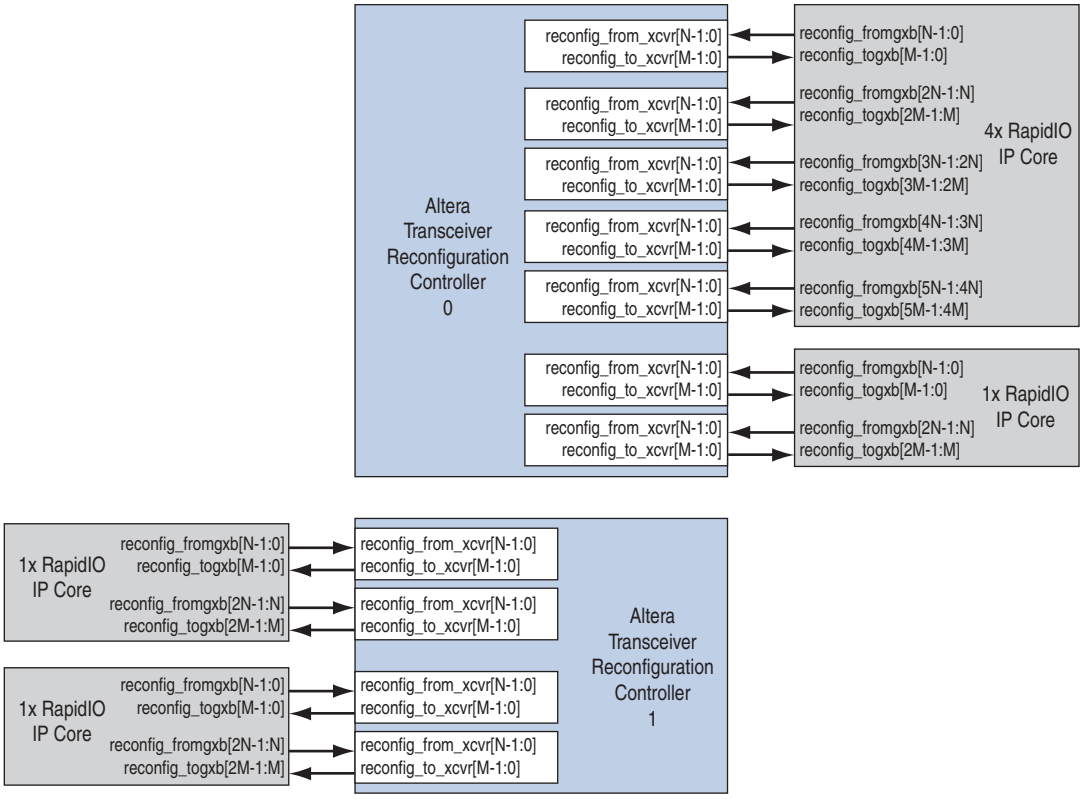
When your design targets an Arria V, Cyclone V, or Stratix V device, the transceivers are configured with the Altera Custom PHY IP core. When your design contains multiple RapidIO IP cores, the Quartus II Fitter handles the merge of multiple Custom PHY IP cores in the same transceiver block automatically. To merge multiple Custom PHY IP cores in the same transceiver block, the Fitter requires that the `phy_mgmt_clk_reset` input signal for all of the merged IP cores be driven by the same source.

If you have different RapidIO IP cores in different transceiver blocks on your device, you may choose to include multiple Transceiver Reconfiguration Controllers in your design. However, you must ensure that the Transceiver Reconfiguration Controllers that you add to your design have the correct number of interfaces to control dynamic reconfiguration of all your RapidIO IP core transceivers. The correct total number of reconfiguration interfaces is the sum of the reconfiguration interfaces for each RapidIO IP core; the number of reconfiguration interfaces for each RapidIO IP core is the number of channels plus one. You must ensure that the `reconfig_togxb` and `reconfig_fromgxb` signals of an individual RapidIO IP core connect to a single Transceiver Reconfiguration Controller.

For example, if your design includes one 4× RapidIO IP core and three 1× RapidIO IP cores, the Transceiver Reconfiguration Controllers in your design must include eleven dynamic reconfiguration interfaces: five for the 4× RapidIO IP core, and two for each of the 1× RapidIO IP cores. The dynamic reconfiguration interfaces connected to a single RapidIO IP core must belong to the same Transceiver Reconfiguration Controller. In most cases, your design has only a single Transceiver Reconfiguration Controller, which has eleven dynamic reconfiguration interfaces. If you choose to use two Transceiver Reconfiguration Controllers, for example, to accommodate placement and timing constraints for your design, each of the RapidIO IP cores must connect to a single Transceiver Reconfiguration Controller.

Figure 2–4 illustrates an example design with two Transceiver Reconfiguration Controllers and four RapidIO IP cores. In the example, Altera Transceiver Reconfiguration Controller 0 has seven reconfiguration interfaces, and Altera Transceiver Reconfiguration Controller 1 has four reconfiguration interfaces. Each sub-block shown in a Transceiver Reconfiguration Controller block represents a single reconfiguration interface. The example shows only one possible configuration for this combination of RapidIO IP cores; subject to the constraints described, you may choose a different configuration.

Figure 2–4. Example Connections Between Two Transceiver Reconfiguration Controllers and Four RapidIO IP Cores



Refer to Table 5–12 on page 5–8 for the values of N and M in Figure 2–4.

Refer to the "Transceiver Reconfiguration Controller" chapter of the *Altera Transceiver PHY IP Core User Guide* for more information about the Transceiver Reconfiguration Controller interfaces and how to control dynamic reconfiguration for multiple transceiver channels. Refer to Table 5–12 on page 5–8 for information about the `reconfig_fromgxb` and `reconfig_togxb` signals that connect a single RapidIO IP core to multiple Transceiver Reconfiguration Controller interfaces of the same Transceiver Reconfiguration Controller.

To enable the Quartus II software to place distinct RapidIO IP cores in the same Arria V, Cyclone V, or Stratix V transceiver block, you must ensure that the `phy_mgmt_clk` input to each RapidIO IP core is driven by the same programming interface clock.

Clock and Signal Requirements for Other Devices with Transceivers

When your design contains multiple IP cores that use the Arria GX or Stratix II GX transceiver (ALTGX or ALT2GXB) megafunction or the Arria II GX, Arria II GZ, Cyclone IV GX, or Stratix IV GX transceiver (ALTGX) megafunction, you must ensure that the `cal_blk_clk` and `gxb_powerdown` input signals are connected properly.

In any parameterization flow, you must ensure that the `cal_blk_clk` input to each RapidIO IP core (or any other megafunction or user logic that uses the ALTGX or ALT2GXB megafunction) is driven by the same calibration clock source.

When you use Qsys or SOPC Builder to create a system with multiple RapidIO IP core variations, use the **Connection** panel to ensure that `cal_blk_clk` and any other IP core variations in the system that use transceivers are connected to the `cal_blk_clk` port on the RapidIO IP core variation. In SOPC Builder, you must first filter the signals in the **System Contents** tab to display the clock connections, as described in steps 1 and 2 on page 8–8.

In any parameterization flow, when you merge multiple RapidIO IP cores in a single transceiver block, the same signal must drive `gxb_powerdown` to each of the RapidIO IP core variations and other megafunctions, IP cores, and user logic that use the ALTGX or ALT2GXB megafunction.

To successfully combine multiple high-speed transceiver channels in the same transceiver block, they must have the same dynamic reconfiguration setting. To use the dynamic reconfiguration capability for one transceiver instantiation but not another, in Stratix II GX devices, you must set `reconfig_clk` to `0` and `reconfig_togxb` to `3'b010` for all transceiver channels that do not use the dynamic reconfiguration capability. If two IP cores implement dynamic reconfiguration in the same transceiver block of an Arria II GX, Arria II GZ, Cyclone IV GX, Stratix II GX, or Stratix IV GX device, the parameters or characteristics that you want to control with the dynamic reconfiguration megafunction instance must be identical.

To support the dynamic reconfiguration block, turn on **Analog controls** on the **Reconfiguration Settings** tab in the transceiver parameter editor. Arria II GX, Arria II GZ, Cyclone IV GX, and Stratix IV GX device transceivers require a dynamic reconfiguration block, to support offset cancellation.

Arria GX devices do not support dynamic reconfiguration. However, the `reconfig_clk` and `reconfig_togxb` ports appear in variations targeted to Arria GX devices, so you must set `reconfig_clk` to `0` and `reconfig_togxb` to `3'b010`.

Sourcing Multiple Tcl Scripts

If you use Altera-provided Tcl scripts to specify constraints for IP cores, you must run the Tcl script associated with each generated RapidIO IP core. For example, if a system has `rio1` and `rio2` IP core variations, then you must source **rio1_constraints.tcl**, execute the `add_rio_constraints` command and then source **rio2_constraints.tcl** and run the `add_rio_constraints` command, sequentially, from the Tcl console after generation.

After you compile the design once, you can run the `add_rio_constraints` command with the `-no_compile` option to suppress analysis and synthesis, and decrease turnaround time during development. More specifically, after you run

```
source rio1_constraints.tcl; add_rio_constraints ↵
```

you can run

```
source rio2_constraints.tcl; add_rio_constraints -no_compile ↵
```

In the MegaWizard Plug-In Manager flow, the script contains virtual pins for most I/O ports on the RapidIO IP core to ensure that the I/O pin count for a device is not exceeded. These virtual pin assignments must reflect the names used to connect to each RapidIO instantiation.

3. Parameter Settings

You customize the RapidIO IP core by specifying parameters in the RapidIO parameter editor, which you access from the MegaWizard Plug-In Manager, SOPC Builder, or the Qsys system integration tool in the Quartus II software.

This chapter describes the parameters and how they affect the behavior of the IP core. Each section corresponds to a page in the **Parameter Settings**, **EDA**, or **Summary** tabs in the RapidIO parameter editor.

When parameterizing a IP core using the Qsys or SOPC Builder design flow, the **EDA** and **Summary** tabs are not visible. In these design flows, simulation model settings are inherited from options specified in the Qsys tool or SOPC Builder.

In the RapidIO parameter editor, you use the following pages from the **Parameter Settings** tab to parameterize the RapidIO IP core:

- **Physical Layer**
- **Transport and Maintenance**
- **I/O and Doorbell**
- **Capability Registers**

In the RapidIO parameter editor that appears in the Qsys flow, these four categories are in separate tabs. Subsequent sections describe each of these pages or tabs and their parameters.

For more information about setting simulation options in SOPC Builder refer to "About SOPC Builder" in Quartus II Help.

For more information about setting simulation options in the Qsys tool, refer to the *Creating a System with Qsys* chapter in volume 1 of the *Quartus II Handbook*.

Physical Layer Settings

The **Physical Layer** page defines the characteristics of the Physical layer based on these categories: **Device Options**, **Data Settings**, and **Receive Priority Retry Threshold**.

Device Options

Device Options comprise the following configuration options:

- **Mode Selection**
- **Transceiver Selection**
- **Configure Transceiver**
- **Automatically synchronize transmitted ackID**

- **Send link-request reset-device on fatal errors**
- **Link-request attempts**

Mode Selection

Mode Selection allows you to specify a **1x Serial** or **4x Serial** port consisting of one- or four-lane high-speed data serialization and deserialization.

The 4x variations do not support fallback to 1x mode. You must know whether the IP core has a 1x or 4x link partner and configure the FPGA accordingly. If fallback to 1x is required, the FPGA can be programmed with a 4x variation by default and then reprogrammed to a 1x configuration under system control after failure to synchronize at 4x.

Changing the **Mode Selection** resets the **Transceiver Selection** and transceiver configuration settings to their default values.

Transceiver Selection

You can select any one of the following transceiver options:

- **Stratix II GX PHY**
- **Stratix IV GX PHY**
- **Stratix V GX PHY**
- **Arria V GX PHY**
- **Cyclone V GX PHY**
- **Arria II GZ PHY**
- **HardCopy IV GX PHY**
- **Arria GX PHY**
- **Arria II GX PHY**
- **Cyclone IV GX PHY**
- **External Transceiver**

The **Stratix II GX PHY**, **Stratix IV GX PHY, Stratix V GX PHY, Arria V GX PHY, Cyclone V GX PHY, Arria II GZ PHY, HardCopy IV GX PHY, Arria GX PHY, Arria II GX PHY**, and **Cyclone IV GX PHY** options configure the serial RapidIO variations to use the built-in transceiver blocks of the respective device families.

Selecting **External Transceiver** allows you to use your RapidIO IP core with transceivers outside the device, using the XGMII interface. The RapidIO IP core does not support the XGMII interface for variations that target an Arria II GX, Arria II GZ, Arria V, Cyclone IV GX, Cyclone V, HardCopy IV GX, Stratix IV GX, or Stratix V device. In addition, the Qsys design flow does not support RapidIO IP core variations with external transceivers.

Transceiver Configuration

In the MegaWizard Plug-In Manager flow and the SOPC Builder flow, for the Arria GX, Arria II GX, Arria II GZ, Cyclone IV GX, HardCopy IV GX, Stratix II GX, and Stratix IV GX transceiver options, you can click **Configure Transceiver** to set the analog parameters for the transceiver block. When you click **Configure Transceiver**, the transceiver parameter editor appears. The transceiver parameter editor offers several configuration parameters that you can set, based on board-level conditions, design constraints, or other application-specific requirements, to ensure the proper operation of the serial link. Other transceiver parameters are preset to values compatible with the RapidIO IP core and the selected mode and device family, and cannot be modified. The default values in this interface specify a transceiver configuration that is compatible with the RapidIO IP core.

The **Arria V GX PHY**, **Cyclone V GX PHY**, and **Stratix V GX PHY** selections do not support a **Configure Transceiver** option, because the transceiver parameters are all set to required values automatically. The RapidIO IP core instantiates a Custom PHY IP core to configure the Arria V, Cyclone V, or Stratix V transceivers.

The Qsys flow does not support a **Configure Transceiver** option. To modify the default transceiver configuration in a RapidIO IP core you generate in the Qsys flow, you must edit the ALTGX megafunction instance in the MegaWizard Plug-In Manager. If your RapidIO IP core targets an Arria V, Cyclone V, or Stratix V device, Altera recommends you do not modify the default transceiver settings configured in the Custom PHY IP core instance generated with the RapidIO IP core.

When you regenerate your RapidIO IP core, the ALTGX megafunction instance reverts to the default settings determined by the RapidIO component. To modify the default transceiver configuration, you must re-edit the ALTGX megafunction manually following every regeneration of the RapidIO IP core.

For details of the transceiver block and the transceiver parameter editor, refer to the documents listed in the following table:

For information about	Refer to
Arria GX transceiver megafunction	Arria GX Transceiver User Guide section in volume 2 of the *Arria GX Device Handbook*
Arria II GX or Arria II GZ transceiver megafunction	*Transceiver Architecture in Arria II Devices* chapter in volume 2 of the *Arria II Device Handbook*
Arria V transceivers	*Volume 3: Transceivers* of the *Arria V Device Handbook*
Cyclone IV GX transceiver megafunction	*Cyclone IV Transceivers Architecture* chapter in volume 2 of the *Cyclone IV Device Handbook*
Cyclone V transceivers	*Volume 3: Transceivers* of the *Cyclone V Device Handbook*
HardCopy IV transceiver megafunction	*Volume 3: Transceivers* of the *HardCopy IV Device Handbook*
Stratix II GX transceiver megafunction	Stratix II GX Transceiver User Guide section in *volume 2* of the *Stratix II GX Device Handbook*
Stratix IV transceiver megafunction	*Transceiver Configuration Guide* section in *Volume 3* of the *Stratix IV Device Handbook*

Stratix V transceivers	*Volume 3: Transceivers* of the *Stratix V Device Handbook*
Custom PHY IP core	*Altera Transceiver PHY IP Core User Guide*

Transceiver Configuration Using the Configure Transceiver Button

When you click **Configure Transceiver** in the MegaWizard Plug-In Manager and SOPC Builder flows, in variations that target the relevant devices, the transceiver parameter editor appears. The following sections describe the transceiver parameter editor options that are available.

Parameter Settings

The transceiver parameter editor **Parameter Settings** tab has the following four pages:

- **General**
- **PLL/Ports**
- **Rx Analog**
- **Tx Analog**

On the **General** page, you set the device variation, if relevant.

Do not set the RapidIO transceiver input clock frequency in the transceiver parameter editor. The transceiver input clock is the RapidIO IP core reference clock, whose frequency you set in the RapidIO parameter editor. For information about how to set the transceiver input clock frequency, refer to "Reference Clock Frequency" on page 3–6. Changes you make to the input clock frequency setting in the transceiver parameter editor are ignored.

On the **PLL/Ports** page, you determine whether to use an auxiliary transmitter PLL, set the bandwidth for the transmitter PLL and the receiver CDR block, and specify the acceptable PPM threshold between the receiver CDR voltage-controlled oscillator and the input reference clock.

The transmitter or receiver PLL bandwidth is the measure of the PLL's ability to track the input clock and jitter, determined by the -3 dB frequency of the PLL's closed-loop gain. A higher bandwidth setting provides a faster lock time but tracks more jitter on the input clock source. A lower bandwidth setting filters out more high frequency input clock jitter, but increases lock time.

On the **Rx Analog** page, you select the DC gain and an equalizer control setting to boost the high frequency components of the incoming signal to compensate for losses in the channel. You also specify the receiver common-mode voltage and receiver termination resistance.

On the **Tx Analog** page, you specify the transmitter buffer power, transmitter common-mode voltage, transmitter termination resistance, and voltage output differential, and set the transmitter programmable pre-emphasis settings. Pre-emphasis boosts the high frequencies in the transmit data signal, which may be attenuated by the transmission medium. The pre-emphasis maximizes the data eye opening at the far end receiver, which is particularly useful in lossy transmission mediums.

Reconfiguration Settings

When you target an Arria II GX, Arria II GZ, Cyclone IV GX, HardCopy IV GX, Stratix II GX, or Stratix IV GX device, you can instantiate a transceiver reconfiguration block that dynamically changes the following physical media attachment (PMA) settings:

- Pre-emphasis
- Equalization
- Offset cancellation
- V_{OD} on a per channel basis

For these devices, the transceiver parameter editor has a **Reconfiguration Settings** tab. On this tab, preset options generate the required reconfiguration ports, which are required whether or not you instantiate a dynamic reconfiguration block to connect to this transceiver. On this tab you can select the starting channel number. For 4× variations, the starting channel number must be a multiple of four.

You must instantiate the transceiver reconfiguration block for a high-speed transceiver on an Arria II GX, Arria II GZ, Cyclone IV GX, HardCopy IV GX, or Stratix IV GX device, because the transceivers on these devices require offset cancellation. Your design can compile without this block, but the design cannot function correctly in hardware.

For more information about dynamic reconfiguration, refer to Table 5–12 on page 5–8 or to the relevant device handbook. For more information about offset cancellation, refer to the *Arria II Device Handbook*, *Stratix IV Device Handbook*, or *HardCopy IV Device Handbook*.

Protocol Settings

When you use an Arria GX or Stratix II GX device, the transceiver parameter editor has one additional tab, the **Protocol Settings** tab. All options on this tab are preset for compatibility with the RapidIO IP core, and should not be modified.

Synchronizing Transmitted ackID

The **Automatically synchronize transmitted ackID** option turns on support for using an initial `ackID` value specified by the RapidIO link partner. If the `ackID` value in the first seven status control symbols it receives on the link are identical, the RapidIO IP core uses this value as the starting `ackID` value for packets it transmits. If this option is turned off, the starting `ackID` value is 0.

Sending Link-Request Reset-Device on Fatal Errors

The **Send link-request reset-device on fatal errors** option specifies that if the RapidIO IP core identifies a fatal error, it transmits four `link-request` control symbols with `cmd` set to `reset-device` on the RapidIO link. By default, this option is turned off. The option is available for backward compatibility, because previous releases of the RapidIO IP core implement this behavior.

Number of Link-Request Attempts Before Declaring Fatal Error

The **Link-request attempts** parameter allows you to specify the number of times the RapidIO IP core sends a `link-request reset-device` control symbol following a `link-request` time-out, before declaring a fatal error. This parameter can have values 1 through 7. The default value in a new variation is 7.

Data Settings

Data Settings set the **Baud rate**, **Reference clock frequency**, **Receive buffer** size, and **Transmit buffer** size.

☞ Changing the **Baud rate** or the **Reference clock frequency** resets the **Transceiver Selection** and transceiver configuration settings to their default values.

Baud Rate

Baud rate defines the baud rate based on the value that you specify. Table 1–7 on page 1–10 shows the baud rates supported by the RapidIO IP core for each device family. A device family may include devices at speed grades that do not support all the indicated baud rates. Table 1–7 provides information about the speed grades supported for each device family, RapidIO mode, and baud rate combination.

Reference Clock Frequency

Reference clock frequency defines the frequency of the reference clock for your RapidIO IP core internal transceiver. This option is not available for variations with external transceivers. The RapidIO parameter editor allows you to select any frequency supported by the transceiver.

For more information about the reference clock in high-speed transceiver blocks, and the supported frequencies, refer to "Clocking and Reset Structure" on page 4–3.

Receive Buffer

Receive buffer defines the receive buffer size in KBytes based on the value that you specify. You can select a receive buffer size of **4**, **8**, **16**, or **32** KBytes.

Transmit Buffer

Transmit buffer defines the transmit buffer size in KBytes based on the value that you specify. You can select a transmit buffer size of **4**, **8**, **16**, or **32** KBytes.

☞ Buffers are implemented in embedded RAM blocks. Depending on the size of the device used, the maximum buffer size may be limited by the number of available RAM blocks.

Receive Priority Retry Thresholds

Retry thresholds can be set automatically by turning on **Auto-configured from receiver buffer size**, or manually by specifying the thresholds for **Priority 0**, **Priority 1**, and **Priority 2**. To specify valid values for these priority thresholds, follow these four guidelines:

- **Priority 2** Threshold > 9

- **Priority 1** Threshold > **Priority 2** Threshold + 4
- **Priority 0** Threshold > **Priority 1** Threshold + 4
- **Priority 0** Threshold < (receive buffer size x 1024/64)

Receive priority retry threshold values are numbers of 64-byte buffers. For more information about retry thresholds, refer to "Atlantic Interface Receive Buffer and Control Block" on page 4–18.

Transport and Maintenance Settings

The **Transport and Maintenance** page lets you enable and configure the Transport layer and Logical layer Input/Output Maintenance modules.

Transport Layer

The **Transport Layer** parameters determine whether a Transport layer is implemented, whether the RapidIO IP core uses 8-bit or 16-bit device IDs, and whether the Transport layer has an Avalon-ST pass-through interface.

Enable Transport Layer

Enable transport layer creates a Transport layer, which is required for the Maintenance, Input/Output, and Doorbell Logical layer modules, or to enable the Avalon-ST pass-through interface.

Turning off this option specifies no Transport layer is created. By turning off this option, you create a Physical-layer-only variation. When using the Qsys or SOPC Builder design flow, you cannot create a Physical-layer-only variation.

Device ID Width

The **Device ID Width** setting specifies a device ID width of **8-bit** or **16-bit**. RapidIO packets contain destination ID and source ID fields, which have the specified width. If this IP core uses 16-bit device IDs, it supports large common transport systems. In the Qsys design flow, this parameter is labeled **Enable 16-bit device ID width**.

This option requires a Transport layer. This option is available in the MegaWizard Plug-In Manager flow if you turn on **Enable transport layer**. This option is always available in the SOPC Builder and Qsys flows.

Avalon-ST Pass-Through Interface

Turn on **Enable Avalon-ST pass-through interface** to include the Avalon-ST pass-through interface in your RapidIO variation. This option requires a Transport layer. This option is available in the MegaWizard Plug-In Manager flow if you turn on **Enable transport layer**. This option is always available in the SOPC Builder and Qsys flows.

The Transport layer routes all unrecognized packets to the Avalon-ST pass-through interface. Unrecognized packets are those that contain Format Types (ftypes) for Logical layers not enabled in this IP core, or destination IDs not assigned to this endpoint. However, if you disable **Destination ID Checking**, the packet is a request packet with a supported ftype, and the Transport Type (tt) field of the packet matches the device ID width setting of this IP core, the packet is routed to the appropriate Logical layer.

The destination ID can match this endpoint only if the tt field in the packet matches the device ID width setting of the endpoint.

Request packets with a supported ftype and correct tt field, but an unsupported ttype, are routed to the Logical layer supporting the ftype, which allows the following tasks:

- An ERROR response can be sent to requests that require a response.
- An unsupported_transaction error can be recorded in the Error Management extension registers.

Response packets are routed to a Logical layer module or the Avalon-ST pass-through port based on the value of the target transaction ID field. For more information, refer to Table 4–7 on page 4–23, which defines the transaction ID ranges.

Destination ID Checking

Disable Destination ID checking by default lets you turn on or off the option to route a request packet with a supported ftype but a destination ID not assigned to this endpoint. The effect of this setting is detailed in the "Avalon-ST Pass-Through Interface" section.

This option requires a Transport layer. This option is available in the MegaWizard Plug-In Manager if you turn on **Enable transport layer**. This option is always available in the SOPC Builder and Qsys flows.

You specify the initial value for the option in the RapidIO parameter editor, and software can change it by modifying the value of the PROMISCUOUS_MODE bit in the Rx Transport Control register. Refer to Table 6–51 on page 6–24 for information about this register.

Input/Output Maintenance Logical Layer Module

The **Input/Output Maintenance Logical Layer Module** specifies the interface to the Maintenance Logical layer and the number of translation windows.

Maintenance Logical Layer

Maintenance logical layer interface(s) selects which parts of the Maintenance Logical layer to implement. You can specify any one of the following valid options:

- **Avalon-MM Master and Slave**
- **Avalon-MM Master**
- **Avalon-MM Slave**
- **None**

Transmit Address Translation Windows

Number of transmit address translation windows is applicable only if you select an **Avalon-MM Slave** as the **Maintenance logical layer interface(s)**. You can specify a value from **1** to **16** to define the number of transmit address translation windows supported.

Port Write

The **Port Write** options control whether the port-write requests are transmitted or received by the Maintenance Logical layer module. These options are supported only if the Maintenance Logical layer has an Avalon-MM slave port.

Port Write Tx Enable

Port Write Tx enable turns on or turns off the transmission of port-write requests by the Maintenance Logical layer module.

Port Write Rx Enable

Port Write Rx enable turns on or turns off the reception of port-write requests by the Maintenance Logical layer module.

I/O and Doorbell Settings

This page lets you enable and configure the Input/Output and Doorbell Logical layer modules.

I/O Logical Layer Interfaces

I/O logical layer Interfaces selects whether or not to add a master/slave Avalon-MM interface. You can specify one of the following options:

- **Avalon-MM Master and Slave**
- **Avalon-MM Master**
- **Avalon-MM Slave**
- **None**

I/O Slave Address Width

I/O slave address width specifies the Input/Output slave address width. The default width is 30 bits.

I/O Read and Write Order Preservation

I/O read and write order preservation controls support for order preservation between read and write operations (`NWRITE`, `NWRITE_R`, `SWRITE`, and `NREAD` requests) in the I/O Avalon-MM Logical layer slave module. By default this feature is turned off.

This option is available only if you set **I/O logical layer Interfaces** to **Avalon-MM Master and Slave** or **Avalon-MM Slave**.

Whether you turn on this feature or not, as required by the Avalon-MM specification, each individual Logical layer Avalon-MM slave module preserves response order. Even if the responses to two requests from the same Logical layer Avalon-MM slave module arrive in reverse order on the RapidIO link, the Logical layer module enforces the response order on the Avalon-MM interface. The slave module enforces the order by maintaining a queue of the Transaction IDs of transactions awaiting responses from the RapidIO link.

For more information about the I/O read and write order preservation feature, refer to "Input/Output Avalon-MM Slave Module" on page 4–44.

Avalon-MM Master

Number of Rx address translation windows is only applicable if you select an I/O Avalon-MM master as an I/O Logical layer interface. You can specify a value from **1** to **16**.

Avalon-MM Slave

Number of Tx address translation windows is only applicable if you select an I/O Avalon-MM slave as an I/O Logical layer interface. You can specify a value from **1** to **16**.

Doorbell Slave

Doorbell Tx enable controls support for the generation of outbound `DOORBELL` messages.

Doorbell Rx enable controls support for the processing of inbound `DOORBELL` messages. If not enabled, received `DOORBELL` messages are routed to the Avalon-ST pass-through interface if it is enabled, or are silently dropped if the pass-through interface is not enabled.

Prevent doorbell messages from passing write transactions controls support for preserving transaction order between `DOORBELL` messages and I/O write request transactions. This option is available only if you turn on **Doorbell Tx enable** and set **I/O logical layer Interfaces** to **Avalon-MM Master and Slave** or **Avalon-MM Slave**.

Capability Registers Settings

The **Capability Registers** page lets you set values for some of the capability registers (CARs), which exist in every RapidIO processing element and allow an external processing element to determine the endpoint's capabilities through `MAINTENANCE` read operations. All CARs are 32 bits wide.

The settings on the **Capability Registers** page do not cause any features to be enabled or disabled in the RapidIO IP core. Instead, they set the values of certain bit fields in some CARs.

Device Registers

The **Device Registers** options identify the device, vendor, and revision level and set values in the Device Identity (Table 6–12 on page 6–11) and Device Information (Table 6–13 on page 6–12) CARs.

Device ID

Device ID sets the DEVICE_ID field of the Device Identity register. This option uniquely identifies the type of device from the vendor specified in the Vendor Identity field of the Device Identity register.

This DEVICE_ID field of the Device Identity register (Table 6–12) should not be confused with the DEVICE_ID field in the Base Device ID CSR (Table 6–23 on page 6–16).

Vendor ID

Vendor ID uniquely identifies the vendor and sets the VENDOR_ID field in the Device Identity register. Set **Vendor ID** to the identifier value assigned by the RapidIO Trade Association to your company.

Revision ID

Revision ID identifies the revision level of the device. This value in the Device Information register (Table 6–13) is assigned and managed by the vendor specified in the VENDOR_ID field of the Device Identity register (Table 6–12).

Assembly Registers

The **Assembly Registers** options identify the vendor who manufactured the assembly or subsystem of the device. These registers include the Assembly Identity (Table 6–14 on page 6–12) and the Assembly Information (Table 6–15) CARs.

Assembly ID

Assembly ID corresponds to the ASSY_ID field of the Assembly Identity register (Table 6–14), which uniquely identifies the type of assembly. This field is assigned and managed by the vendor specified in the ASSY_VENDOR_ID field of the Assembly Identity register.

Vendor ID

Vendor ID uniquely identifies the vendor who manufactured the assembly. This value corresponds to the ASSY_VENDOR_ID field of the Assembly Identity register. In the Qsys design flow, this parameter is labeled **Assembly vendor ID**.

Revision ID

Revision ID indicates the revision level of the assembly and sets the ASSY_REV field of the Assembly Information CAR (Table 6–15). In the Qsys design flow, this parameter is labeled **Assembly revision ID**.

Extended Features Pointer

Extended features pointer points to the first entry in the extended feature list and corresponds to the EXT_FEATURE_PTR in the Assembly Information CAR.

Processing Element Features

The Processing Element Features CAR (Table 6–16 on page 6–12) identifies the major features of the processing element.

Bridge Support

Bridge Support, when turned on, sets the BRIDGE bit in the Processing Element Features CAR and indicates that this processing element can bridge to another interface such as PCI Express, a proprietary processor bus such as Avalon-MM, DRAM, or other interface.

Memory Access

Memory Access, when turned on, sets the MEMORY bit in the Processing Element Features CAR and indicates that the processing element has physically addressable local address space that can be accessed as an endpoint through non-maintenance operations. This local address space may be limited to local configuration registers, or can be on-chip SRAM, or another memory device.

Processor Present

Processor present, when turned on, sets the PROCESSOR bit in the Processing Element Features CAR and indicates that the processing element physically contains a local processor such as the Nios® II embedded processor or similar device that executes code. A device that bridges to an interface that connects to a processor should set the BRIDGE bit—as described in "Bridge Support"—instead of the PROCESSOR bit.

Switch Support

The **Switch Support** options define switch support characteristics.

Enable Switch Support

Enable switch support, when turned on, sets the SWITCH bit in the Processing Element Features CAR (Table 6–16 on page 6–12) and indicates that the processing element can bridge to another external RapidIO interface. A processing element that only bridges to a local endpoint is not considered a switch port.

Number of Ports

Number of ports specifies the total number of ports on the processing element. This value sets the PORT_TOTAL field of the Switch Port Information CAR (Table 6–17 on page 6–13).

Port Number

Port number sets the PORT_NUMBER field of the Switch Port Information CAR. This value is the number of the port from which the MAINTENANCE read operation accesses this register.

Data Messages

The **Data Messages** options indicate which, if any, data message operations are supported by user logic attached to the pass-through interface, which you must select on the **Transport and Maintenance** page.

Turning on one or both of **Source Operation** and **Destination Operation** causes additional input ports to be added to the RapidIO IP core to support reporting of data-message related errors through the standard Error Management Extension registers.

For more information, refer to Chapter 5, Signals and Chapter 6, Software Interface.

Source Operation

Source Operation, when turned on, sets the `Data Message` bit in the `Source Operations` CAR (Table 6–18 on page 6–14) and indicates that this endpoint can issue Data Message request packets.

Destination Operation

Destination Operation, when turned on, sets the `Data Message` bit in the `Destination Operations` CAR (Table 6–19 on page 6–14) and indicates that this endpoint can process received Data Message request packets.

EDA Settings

The **EDA** tab specifies the simulation libraries, and lets you turn on or off **Generate simulation model** and **Generate netlist**.

These options are not available when using the Qsys or SOPC Builder design flow. You can generate simulation models in the Qsys flow by indicating the appropriate HDL in which to generate a simulation model on the **Generation** tab in Qsys. You can generate simulation models in the SOPC Builder flow by turning on the **Simulation. Create project simulator files** option on the **System Generation** tab of SOPC Builder.

Simulation Libraries

Simulation Libraries displays a list of files that includes those needed to run simulation with the generated IP functional simulation model, and lets you turn on or off the generation of the simulation model.

File

File specifies one or more simulation library files.

Description

Description characterizes the file specified in the **File** field.

Generate Simulation Model

Generate simulation model turns on or off the generation of the IP functional simulation model.

Timing and Resource Estimation

The **Timing and Resource Estimation** option allows you to generate a netlist file that can be used by some third-party synthesis tools.

Generate netlist turns on or off the generation of netlist files used by some third-party synthesis tools to estimate timing and resource usage.

Summary

The **Summary** tab displays a list of files that are generated when you click **Finish**. Files automatically generated have a gray checkmark in the checkbox. To generate additional files, click in an empty checkbox. To prevent generation of files other than the automatically generated files, click the checkbox to remove a checkmark. The **Summary** tab does not appear in the SOPC Builder or Qsys design flow.

Interfaces

The Altera RapidIO IP core supports the following interfaces:

- RapidIO Interface
- Atlantic Interface
- Avalon Memory Mapped (Avalon-MM) Master and Slave Interfaces
- Avalon Streaming (Avalon-ST) Interface
- XGMII External Transceiver Interface

RapidIO Interface

The RapidIO interface complies with revision 2.1 of the RapidIO® serial interface standard described in the RapidIO Trade Association specifications. The protocol is divided into a three-layer hierarchy: Physical layer, Transport layer, and Logical layer.

More detailed information about the RapidIO interface specification is available from the RapidIO Trade Association website at www.rapidio.org.

Atlantic Interface

The Atlantic interface, an Altera protocol, provides access to the Physical layer. Physical-layer-only variations use the Atlantic interface to support user logic. For 1× variations, the Atlantic interface is 32 bits wide. For 4× variations, the Atlantic interface is 64 bits wide.

The Atlantic interface is a full-duplex synchronous protocol. The transmit Atlantic interface functions as a slave-sink interface. The receive Atlantic interface functions as a slave-source interface.

For more information about the Atlantic interface, refer to the *FS13: Atlantic Interface* specification.

Avalon Memory Mapped (Avalon-MM) Master and Slave Interfaces

The Avalon-MM master and slave interfaces execute transfers between the RapidIO IP core and the system interconnect. The system interconnect allows you to use the Qsys system integration tool or SOPC Builder to connect any master peripheral to any slave peripheral, without detailed knowledge of either the master or slave interface. The RapidIO IP core implements both Avalon-MM master and Avalon-MM slave interfaces.

For more information about the Avalon-MM interface, refer to *Avalon Interface Specifications*.

Avalon-MM Interface Byte Ordering

The RapidIO protocol uses big endian byte ordering, whereas Avalon-MM interfaces use little endian byte ordering. Table 4–1 shows the byte ordering for the Avalon-MM and RapidIO interfaces.

No byte- or bit-order swaps occur between the Avalon-MM protocol and RapidIO protocol, only byte- and bit-number changes. For example, RapidIO Byte0 is Avalon-MM Byte7, and for all values of i from 0 to 63, bit i of the RapidIO 64-bit double word[0:63] of payload is bit (63-i) of the Avalon-MM 64-bit double word[63:0].

Table 4–1. Byte Ordering

Byte Lane (Binary)	1000_0000	0100_0000	0010_0000	0001_0000	0000_1000	0000_0100	0000_0010	0000_0001
RapidIO Protocol (Big Endian)	Byte0[0:7]	Byte1[0:7]	Byte2[0:7]	Byte3[0:7]	Byte4[0:7]	Byte5[0:7]	Byte6[0:7]	Byte7[0:7]
	32-Bit Word[0:31] wdptr=0				32-Bit Word[0:31] wdptr=1			
	Double Word[0:63] RapidIO Address N = {29'hn, 3'b000}							
Avalon-MM Protocol (Little Endian)	Byte7[7:0]	Byte6[7:0]	Byte5[7:0]	Byte4[7:0]	Byte3[7:0]	Byte2[7:0]	Byte1[7:0]	Byte0[7:0]
	Address= N+7	Address= N+6	Address= N+5	Address= N+4	Address= N+3	Address= N+2	Address= N+1	Address= N
	32-Bit Word[31:0] Avalon-MM Address = N+4				32-Bit Word[31:0] Avalon-MM Address = N			
	64-bit Double Word0[63:0] Avalon-MM Address = N							

In variations of the RapidIO IP core that have 32-bit wide Avalon-MM interfaces, the order in which the two 32-bit words in a double word appear on the Avalon-MM interface in a burst transaction, is inverted from the order in which they appear inside a RapidIO packet. The RapidIO 32-bit word with `wdptr=0` is the most significant half of the double word at RapidIO address N, and the 32-bit word with `wdptr=1` is the least significant 32-bit word at RapidIO address N. Therefore, in a burst transaction on the Avalon-MM interface, the 32-bit word with `wdptr=0` corresponds to the Avalon-MM 32-bit word at address N+4 in the Avalon-MM address space, and must follow the 32-bit word with `wdptr=1` which corresponds to the Avalon-MM 32-bit word at address N in the Avalon-MM address space. Thus, when a burst of two or more 32-bit Avalon-MM words is transported in RapidIO packets, the order of the pair of 32-bit words is inverted so that the most significant word of each pair is transmitted or received first in the RapidIO packet.

Avalon Streaming (Avalon-ST) Interface

The Avalon-ST interface provides a standard, flexible, and modular protocol for data transfers from a source interface to a sink interface. The Avalon-ST interface protocol allows you to easily connect components together by supporting a direct connection to the Transport layer. The Avalon-ST interface is either 32 or 64 bits wide depending on the RapidIO lane width. This interface is available to create custom Logical layer functions like message passing.

For more information about how this interface functions with the RapidIO IP core, refer to the "Avalon-ST Pass-Through Interface" on page 4–59.

XGMII External Transceiver Interface

The XGMII interface is the external transceiver interface that connects the RapidIO IP core to an external transceiver.

The external transceiver interface provides 8-bit transmit and receive datapaths per serial lane, plus the necessary control and clocking signals to allow bidirectional data transfers. This interface is similar to the 10-Gigabit Media-Independent Interface (XGMII) using either HSTL Class 1 or SSTL Class 2 I/O drivers. The XGMII supports one control signal per 8 bits for the external transceiver encoder, and one control and one error signal per 8 bits from the external transceiver decoder.

On the transmit side, the 8-bit data (`td`) and 1-bit control (`tc`) signals per lane are transmitted on the rising and falling edges of a center aligned clock, `tclk`. The external transmitter should be disabled when the Initialization state machine (described in section 4.12 of *Part 6: LP-Serial Physical Layer Specification* of the *RapidIO Interconnect Specification, Revision 2.1*) is in the *SILENT* state and drives the `phy_dis` output signal high to request turning off the output driver.

On the receive side, the 8-bit data (`rd`), 1-bit control (`rc`), and error (`rerr`) signals per lane are received and sampled on the rising and falling edges of a center-aligned clock, `rclk`. Separate `rclk` signals are associated with each lane.

For further details, including timing requirements for the XGMII interface, refer to Appendix B, XGMII Interface Timing.

Clocking and Reset Structure

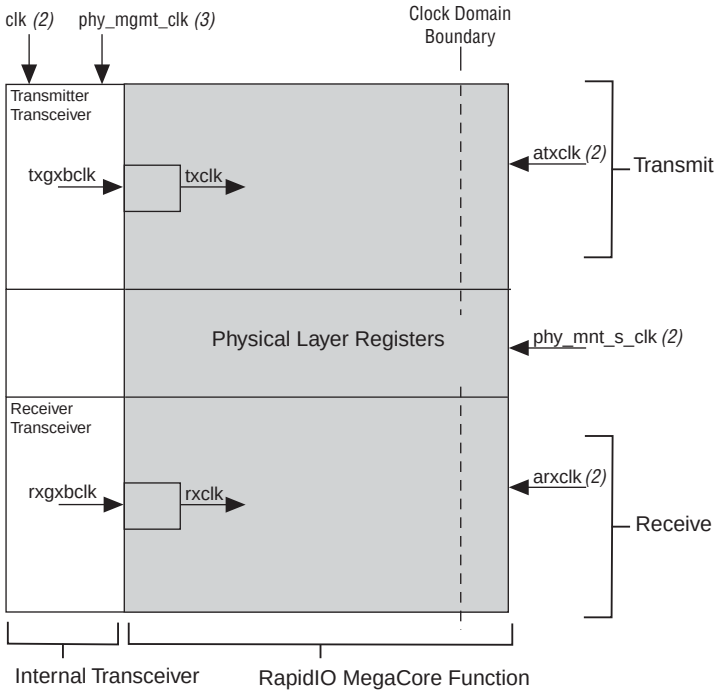
Clock domains in the RapidIO IP core depend on whether the IP core has only a Physical layer or has Physical, Transport, and Logical layers. The first part of this section describes the clock domains and reset structure for variations that have only a Physical layer. For information about variations that have Physical, Transport, and Logical layers, refer to "Clocking for RapidIO IP Cores with Physical, Transport, and Logical Layers" on page 4–7.

Clocking for RapidIO IP Cores with Only a Physical Layer

In addition to the high-speed clock domains inside the high-speed internal transceiver, the RapidIO IP core contains six clock domains: two transceiver clocks (`txgxbclk` and `rxgxbclk`), two internal global clocks (`txclk` and `rxclk`), and two Atlantic interface clocks (`atxclk` and `arxclk`). `txclk` is the main clock for the transmitter modules in the Physical layer, and `rxclk` is the recovered clock that drives the receiver modules in the Physical layer. An additional clock domain exists for the `phy_mnt_s` Avalon-MM interface.

Figure 4–1 shows the clock signals in a RapidIO IP core with internal transceivers. For information about the clock signals in the XGMII interface of a RapidIO IP core with external transceivers, refer to Appendix B, XGMII Interface Timing.

Figure 4–1. Clock Domains in a RapidIO IP Core with Internal Transceivers (1)



Notes to Figure 4–1:

(1) Clock descriptions:

clk	Reference clock
phy_mgmt_clk	Transceiver programming interface clock (Arria V, Cyclone V, and Stratix V devices only)
txgxbclk	Transmitter transceiver clock
rxgxbclk	Receiver transceiver clock
txclk	Transmitter internal global clock
rxclk	Receiver internal global clock (recovered clock)

(2) Input clocks (user supplied)

(3) The phy_mgmt_clk input clock is supported in Arria V, Cyclone V, and Stratix V devices only.

RapidIO IP core 4× variations using high-speed transceivers on Arria II GX, Arria II GZ, Arria V, Cyclone V, Stratix IV, and Stratix V devices are implemented in the transceiver TX bonded mode. All channels of a 4× variation that uses high-speed transceivers, on any supported device, must reside in a single transceiver block. To support this requirement, the starting channel number for a 4× variation must be a multiple of four, unless the variation targets a device whose transceiver is configured with a Custom PHY IP core.

For RapidIO IP core 4× variations using high-speed transceivers on Arria GX and Stratix II GX devices, you must ensure that the 0PPM clock group settings are set so that the IP core uses the internal phase compensation FIFOs within the transceiver block.

When you generate a custom IP core, the **<*variation name*>_constraints.tcl** script contains the required assignments. For Arria GX and Stratix II GX devices, the assignments in the generated Tcl script include the appropriate 0PPM clock group settings automatically. When you run the script, the constraints are applied to your project.

Reference Clock

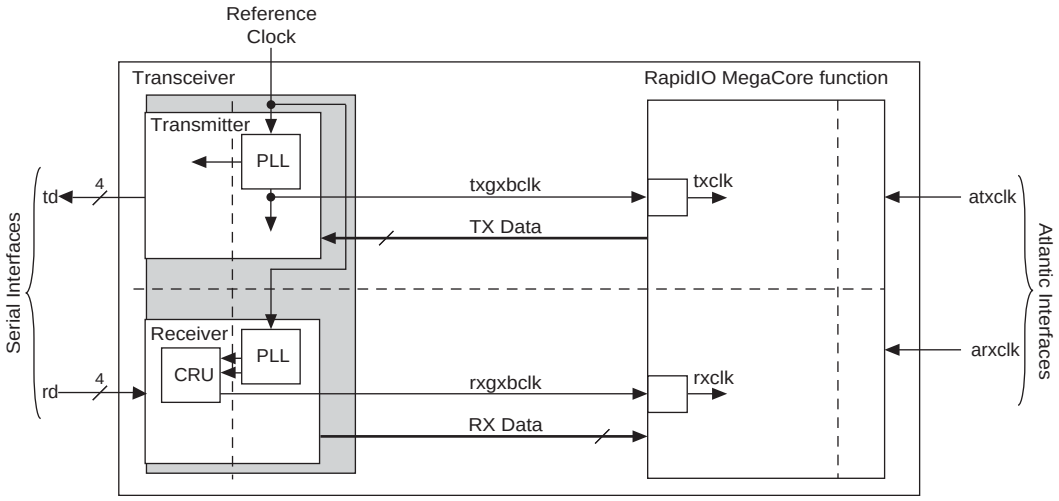
The main reference clock, `clk`, is the incoming reference clock for the transceiver's PLL. This reference clock can have any of a set of frequencies that the PLL in the transceiver can convert to the required internal clock speed for the RapidIO IP core baud rate. The RapidIO parameter editor lets you select one of the supported frequencies.

The ability to program the frequency of the input reference clock allows you to use an existing clock in your system as the reference clock for the RapidIO IP core.

For more information about the supported frequencies for the reference clock in your RapidIO variation, refer to the relevant device handbook.

Figure 4–2 shows the clock domain relationships and how the transceiver uses the reference clock.

Figure 4–2. Reference Clock and Clock Domains in a RapidIO IP Core with Internal Transceivers (1)



Note to Figure 4–2:

(1) The clock domain for the Physical layer's software interface, the Avalon-MM clock `phy_mnt_s_clk`, is not shown in this figure. The Custom PHY IP core clock `phy_mgmt_clk` is also not shown.

The PLL generates the high-speed transmit clock and the input clocks to the receiver high-speed deserializer clock and recovery unit (CRU). The CRU generates the recovered clock (`rxclk`) that drives the receiver logic.

The `txclk` clock is the main clock used in the transmitter modules of the Physical layer. If the RapidIO IP core uses an external transceiver, `txclk` is derived from the `clk` reference clock by dividing by one, two, or four, depending on the configuration of the IP core. The division is performed by a flip-flop-based circuit and does not require a PLL.

Baud Rates

The serial RapidIO specification specifies baud rates of 1.25, 2.5, 3.125, and 5.0 Gbaud. Table 4–2 shows the relationship between baud rates, transceiver clock rates, and internal clock rates. For information about device family support for different RapidIO variations, refer to Table 1–7 on page 1–10.

Table 4–2. Baud Rates and Clock Rates for Physical-Layer-Only RapidIO IP Core

Baud Rates (Gbaud)	Transceiver Clocks (MHz) (txgxbclk/rxgxbclk)	Internal Clocks (MHz) (txclk/rxclk)	
		1x mode	4x mode
5.0	250	125	250
3.125	156.25	78.125	156.25
2.5	125	62.5	125
1.25	62.5	31.25	62.5

For more information about using high-speed transceiver blocks, refer to the relevant device handbook.

Reset for RapidIO IP Cores with Only a Physical Layer

All reset signals can be asserted asynchronously to any clock. However, most reset signals must be deasserted synchronously to a specific clock. The Atlantic interface resets, for example, should be deasserted on the rising edge of the corresponding clock. Figure 4–4 on page 4–11 shows a circuit that ensures a reset signal lasts at least one clock period and is deasserted synchronously to the rising edge of the clock.

In Arria V, Cyclone V, and Stratix V devices, the internal transceiver has an additional reset signal, `phy_mgmt_clk_reset`, which resets the Custom PHY IP core. These variations have the following additional constraints:

- The Custom PHY IP core `phy_mgmt_clk_reset` signal and the RapidIO IP core `reset_n` signal must be driven from the same source, with the caveat that the `phy_mgmt_clk_reset` signal is active high and the `reset_n` signal is active low. The two reset signals must be asserted synchronously, but deasserted each according to its corresponding clock. Figure 4–5 on page 4–11 shows a circuit that ensures the requirements for these two reset signals are met.
- You must ensure that the system does not deassert `reset_n` and `phy_mgmt_clk_reset` when the Altera Transceiver Reconfiguration Controller `reconfig_busy` signal is asserted. The RapidIO IP core must remain in reset until the Transceiver Reconfiguration Controller is available.

For more information about the requirements for reset signals, refer to Chapter 5, Signals.

Variations of the serial RapidIO IP core that use the internal transceiver have a dedicated reset control module named `riophy_reset` to handle the specific requirements of the internal transceiver module. This reset control module is in the **riophy_reset.v** clear-text Verilog HDL source file, and is instantiated inside the top-level module found in the clear text **<***variation name***>_riophy_xcvr.v** Verilog HDL source file.

Variations of the serial RapidIO IP core that use an external transceiver do not require this special reset control module.

The `riophy_reset` module controls all of the RapidIO IP core's internal reset signals. In particular, it generates the recommended reset sequence for the transceiver. The reset sequence and requirements vary among device families. For details, refer to the relevant device handbook.

Consistent with normal operation, following the reset sequence, the Initialization state machine transitions to the *SILENT* state.

For details of the RapidIO Initialization state machine, refer to section 4.12 of *Part 6: LP-Serial Physical Layer Specification* of the *RapidIO Interconnect Specification, Revision 2.1*, available at www.rapidio.org.

If two communicating RapidIO IP cores are reset one after the other, one of the IP cores may enter the *Input Error Stopped* state because the other IP core is in the *SILENT* state while this one is already initialized. The initialized IP core enters the *Input Error Stopped* state and subsequently recovers.

Clocking for RapidIO IP Cores with Physical, Transport, and Logical Layers

Variations with Physical, Transport, and Logical layers have three clock inputs. The variations with internal transceivers have the reference clock, the Avalon system clock, and the internal transceiver's calibration-block clock. The variations with external transceivers have the reference clock, the Avalon system clock, and the external transceiver's input clock.

The reference clock signal drives the Physical layer. In systems created in SOPC Builder, it is called `clk_`<*variation name*>. In variations created in the MegaWizard Plug-In Manager or Qsys design flow, it is called `clk`. Qsys allows you to export the `clk` signal with a name of your choice.

For RapidIO IP cores with external transceivers, the reference clock frequency is determined by the baud rate you specify, the lane width, and the device family. For RapidIO IP cores with internal transceivers, you can specify the reference clock frequency when you create the RapidIO IP core instance. The choices available to you for this frequency are determined by the baud rate. For information about how the transceiver uses the reference clock, refer to "Reference Clock" on page 4–5.

The Avalon system clock drives the Transport and Logical layer modules; its frequency is nominally the same frequency as the Physical layer's internal clocks `txclk` and `rxclk`, but it can differ by up to ±50% provided the Avalon system clock meets f_{MAX} limitations. This clock is displayed as `clock` in the Qsys and SOPC Builder design flows, and is called `sysclk` in the MegaWizard Plug-In Manager design flow. Qsys allows you to export the `clock` signal with a name of your choice.

You must drive the Avalon system clock from a clock source that is running reliably when the RapidIO IP core comes out of reset.

In systems created in SOPC Builder, the external transceiver input clock is called `rclk_`<*variation name*>. In variations created in the MegaWizard Plug-In Manager design flow, it is called `rclk`. In systems created in the Qsys tool, you cannot configure the RapidIO IP core to use an external transceiver.

In variations that target a device for which the transceivers are configured with the ALTGX megafunction, and not with the Transceiver PHY IP core, the internal transceiver's calibration-block clock is called `cal_blk_clk` in the MegaWizard Plug-In Manager design flow and is also displayed as `cal_blk_clk` in the Qsys and SOPC Builder design flows. Refer to Table 4–5 and Table 4–6 for more information.

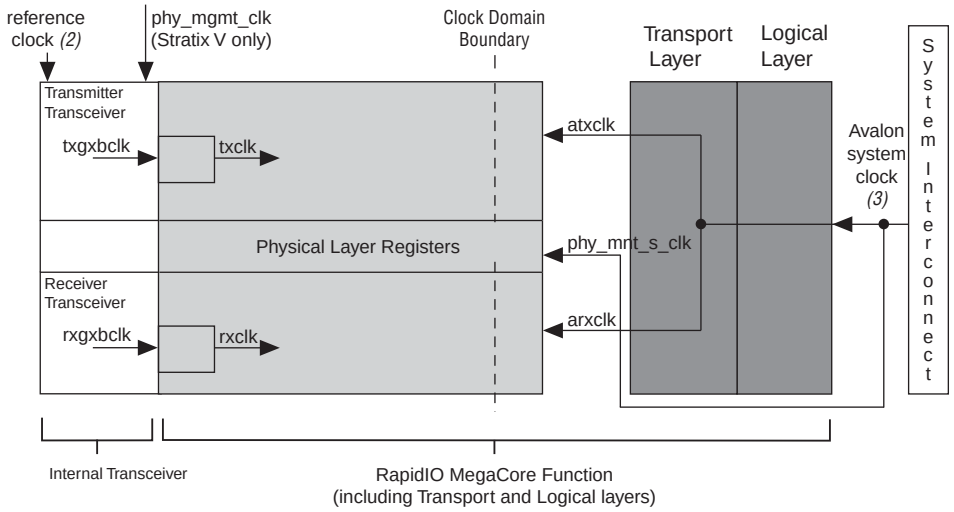
In Arria V, Cyclone V, and Stratix V devices, the internal transceiver has an additional clock, `phy_mgmt_clk`, which clocks the software interface to the transceiver.

The Physical layer's buffers implement clock domain crossing between the Avalon system clock domain and the Physical layer's clock domains.

In systems created with Qsys and SOPC Builder, the system interconnect manages clock domain crossing if some of the components of the system run on a different clock. For optimal throughput, run all the components in the datapath on the same clock.

All of the clock inputs for the Logical layer modules must be connected to the same clock source as the system clock. Figure 4–3 is a block diagram of the clock structure of variations with Physical, Transport, and Logical layers.

Figure 4–3. Clock Domains in RapidIO IP Core with Transport and Logical Layers (1)



Notes to Figure 4–3:

(1) Clock descriptions:

`phy_mgmt_clk`	PHY IP core management clock (Arria V, Cyclone V, Stratix V devices only)
`txgxbclk`	Transmitter transceiver clock
`rxgxbclk`	Receiver transceiver clock
`txclk`	Transmitter internal global clock
`rxclk`	Receiver internal global clock (recovered clock)
`atxclk, arxclk`	Atlantic interface clocks
`phy_mnt_s_clk`	Avalon-MM interface clock for register access

(2) The reference clock is called `clk` in variations generated with the MegaWizard Plug-In Manager or Qsys, and `clk_<variation_name>` in variations created with SOPC Builder.

(3) The Avalon system clock is called `sysclk` in variations generated with the MegaWizard Plug-In Manager and `clock` in variations created with SOPC Builder or Qsys.

Table 4–3 and Table 4–4 provide information about clock rates in the different RapidIO IP core variations with Physical, Transport, and Logical layers that use XGMII.

Table 4–3. Clock Frequencies for 1x Variations with all Three Layers Using XGMII

Variations Using XGMII					
Baud Rate (Gbaud)	reference clock (1), tclk, rclk (MHz)	txclk, rxclk (MHz)	Avalon system clock (2)		
			Minimum (MHz)	Typical (MHz)	Maximum (3) (MHz)
1.25	62.5	31.25	15.625	31.25	46.875
2.5	125	62.5	31.25	62.5	93.75
3.125	156.25	78.125	39.065	78.125	117.19

Notes to Table 4–3:

(1) The reference clock is called `clk` in variations generated with the MegaWizard Plug-In Manager or Qsys, and `clk_`<*variation_name*> in variations created with SOPC Builder.

(2) The Avalon system clock is called `sysclk` in variations generated with the MegaWizard Plug-In Manager and `clock` in variations created with SOPC Builder or Qsys.

(3) The maximum system clock frequency might be limited by the achievable f_{MAX} and can vary based on the family and speed grade.

Table 4–4. Clock Frequencies for 4x Variations with all Three Layers Using XGMII

Variations Using XGMII				
Baud Rate (Gbaud)	txclk, rxclk, reference clock (1), rclk, tclk (MHz)	Avalon System Clock (1), (2)		
		Minimum (MHz)	Typical (MHz)	Maximum (2) (MHz)
1.25	62.5	31.25	62.5	93.75
2.5	125	62.5	125	187.5
3.125	156.25	78.125	156.25	234.275

Notes to Table 4–4:

(1) Refer to Table 5–3 and Table 5–4 on page 5–2 for the reference and system clock signal names in the MegaWizard Plug-In Manager, Qsys, and SOPC Builder design flows.

(2) The maximum system clock frequency might be limited by the achievable f_{MAX} and can vary based on the family and speed grade.

Table 4–5 and Table 4–6 provide information about clock rates in the different RapidIO IP core variations with Physical, Transport, and Logical layers that use the internal high-speed transceivers.

Table 4–5. Clock Frequencies for 1x Variations with all Three Layers Using Internal Transceivers

Baud Rate (Gbaud)	Default reference clock frequency (1), (2) (MHz)	txclk, rxclk (MHz)	Avalon system clock (3)		
			Minimum (MHz)	Typical (MHz)	Maximum (4) (MHz)
1.25	62.5	31.25	15.625	31.25	46.875
2.5	125	62.5	31.25	62.5	93.75
3.125	156.25	78.125	39.065	78.125	117.19
5.0	250	125.0	62.50	125.0	187.50

Notes to Table 4–5:

(1) For information about the allowed reference clock frequencies in devices using internal transceivers, refer to "Reference Clock" on page 4–5.

(2) The reference clock is called `clk` in variations generated with the MegaWizard Plug-In Manager or Qsys, and `clk_<variation_name>` in variations created with SOPC Builder.

(3) The Avalon system clock is called `sysclk` in variations generated with the MegaWizard Plug-In Manager and `clock` in variations created with SOPC Builder or Qsys.

(4) The maximum system clock frequency might be limited by the achievable f_{MAX} and can vary based on the family and speed grade.

Table 4–6. Clock Frequencies for 4x Variations with all Three Layers Using Internal Transceivers

Baud Rate (Gbaud)	txclk, rxclk, and default reference clock frequency (1), (2) (MHz)	Avalon System Clock (1)		
		Minimum (MHz)	Typical (MHz)	Maximum (3) (MHz)
1.25	62.5	31.25	62.5	93.75
2.5	125	62.5	125	187.5
3.125	156.25	78.125	156.25	234.275
5.0	250	125.0	250	250

Notes to Table 4–6:

(1) Refer to Table 5–3 and Table 5–4 on page 5–2 for the reference and system clock signal names in the MegaWizard Plug-In Manager, Qsys, and SOPC Builder design flows.

(2) For information about the allowed reference clock frequencies in devices using internal transceivers, refer to "Reference Clock" on page 4–5.

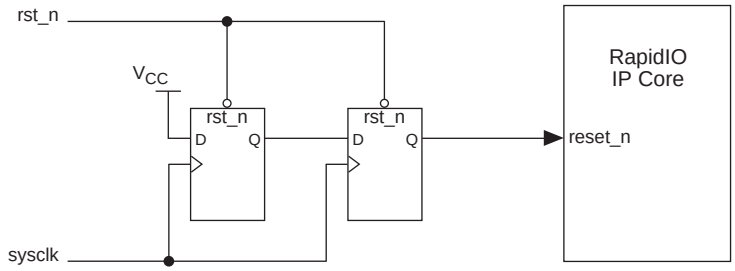
(3) The maximum system clock frequency might be limited by the achievable f_{MAX} and can vary based on the family and speed grade.

Reset for RapidIO IP Cores with Physical, Transport, and Logical Layers

RapidIO IP core variations with all three layers that do not target an Arria V, Cyclone V, or Stratix V device have a single main active-low reset input signal (`reset_n`). RapidIO IP core variations with all three layers that target an Arria V, Cyclone V, or Stratix V device have two main reset signals, `reset_n` and `phy_mgmt_clk_reset`. For constraints on the reset signals and detailed reset behavior, refer to "Reset for RapidIO IP Cores with Only a Physical Layer" on page 4–6.

The reset_n input signal can be asserted asynchronously, but must last at least one Avalon system clock period and be deasserted synchronously to the rising edge of the Avalon system clock. Figure 4–4 shows a circuit that ensures these conditions.

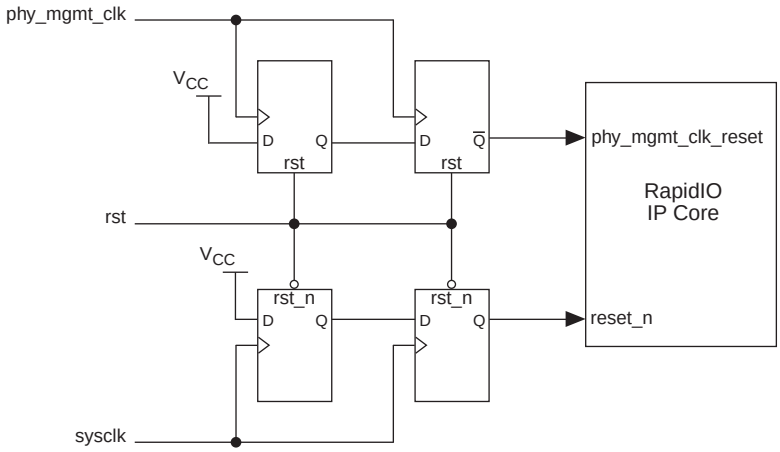
Figure 4–4. Circuit to Ensure Synchronous Deassertion of reset_n



In systems generated by Qsys and SOPC Builder, this circuit is generated automatically. However, if your RapidIO IP core variation is not generated by Qsys or SOPC Builder, you must implement logic to ensure the minimal hold time and synchronous deassertion of the reset_n input signal to the RapidIO IP core.

In Arria V, Cyclone V, and Stratix V devices, phy_mgmt_clk_reset must be asserted with reset_n. However, each signal is deasserted synchronously with its corresponding clock. Figure 4–5 shows a circuit that ensures these conditions.

Figure 4–5. Circuit to Also Ensure Synchronous Assertion of phy_mgmt_clk_reset with reset_n



In systems generated by Qsys, this circuit is generated automatically. However, if your Arria V, Cyclone V, or Stratix V RapidIO IP core variation is not generated by Qsys, you must implement logic to ensure that reset_n and phy_mgmt_clk_reset are driven from the same source, and that each meets the minimal hold time and synchronous deassertion requirements.

The assertion of reset_n causes the whole module to reset. In Arria V, Cyclone V, and Stratix V devices, the requirement that phy_mgmt_clk_reset be asserted with reset_n ensures that the PHY IP core resets with the RapidIO IP core. While the module is held in reset, the Avalon-MM waitrequest outputs are driven high and all other outputs are driven low. When the module comes out of the reset state, all buffers are empty. Refer to Chapter 6, Software Interface for the default value of registers after reset.

Physical Layer

This section describes features and blocks of the 1× or 4× serial Physical layer of the RapidIO IP core. Figure 4–6 on page 4–13 shows a high-level block diagram of the serial RapidIO IP core's Physical layer.

Features

The Physical layer has the following features:

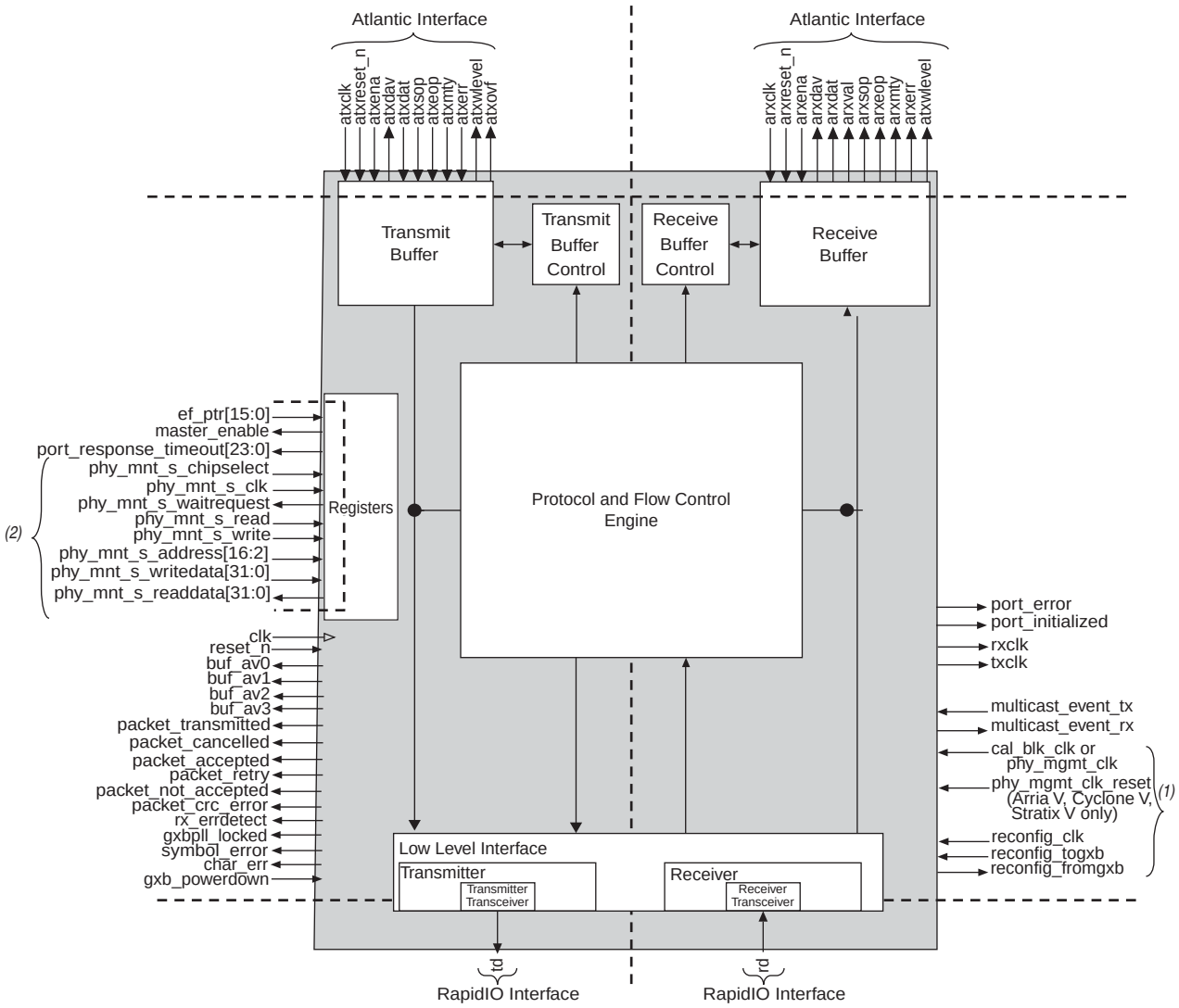
- Port initialization
- Transmitter and receiver with the following features:
  - One or four lane high-speed data serialization and deserialization (up to 5.0 Gbaud for 1× variations with 32-bit Atlantic interface; up to 5.0 Gbaud for 4× variations with 64-bit Atlantic interface)
  - Clock and data recovery (receiver)
  - 8B10B encoding and decoding
  - Lane synchronization (receiver)
  - Packet/control symbol assembly and delineation
  - Cyclic redundancy code (CRC) generation and checking on packets
  - Control symbol CRC-5 generation and checking
  - Error detection
  - Pseudo-random idle sequence generation
  - Idle sequence removal
- Software interface (status/control registers)
- Flow control (ackID tracking)
- Time-out on acknowledgements
- Order of retransmission maintenance and acknowledgements
- ackID assignment
- ackID synchronization after reset
- Error management
- Clock decoupling
- FIFO buffer with level output port
- Adjustable buffer sizes (4 KBytes to 32 KBytes)

- Four transmission queues and four retransmission queues to handle packet prioritization
- Can be configured to send `link-request` control symbols with `cmd` set to `reset-device` on fatal error
- Attempts `link-request` `link-response` control symbol pair a configurable number of times before declaring fatal error, when a `link-response` is not received

Physical Layer Architecture

Figure 4–6 shows the architecture of the Physical layer and illustrates the interfaces that it supports. Dotted lines indicate clock domain boundaries within the layer.

Figure 4–6. Physical Layer High Level Block Diagram



Notes to Figure 4–6:

(1) These signals exist for all device families with high-speed transceivers. Refer to Table 5–12 on page 5–8.

(2) These signals are only present in Physical-layer-only variations. In variations with a Transport layer, the registers are accessed through the system maintenance Avalon-MM slave interface.

SOPC Builder and Qsys do not support a Physical-layer-only variation.

Low-level Interface Receiver

The receiver in the low-level interface receives the input from the RapidIO interface, and performs the following tasks:

- Separates packets and control symbols
- Removes idle sequence characters
- Detects `multicast-event` and `stomp` control symbols
- Detects packet-size errors
- Checks the control symbol 5-bit CRC and asserts `symbol_error` if the CRC is incorrect

Receiver Transceiver

The receiver transceiver is an embedded megafunction in the Arria GX, Arria II GX, Arria II GZ, Stratix II GX, or Stratix IV GX device, or an embedded Custom PHY IP core in the Arria V, Cyclone V, or Stratix V device. Serial data from differential input pins is fed into the CRU to detect clock and data. Recovered data is deserialized into 10-bit code groups and sent to the pattern detector and word-aligner block to detect word boundaries. Properly aligned 10-bit code groups are then 8B10B decoded into 8-bit characters and converted to 16-bit data in the 8-to-16 demultiplexer.

CRC Checking and Removal

The RapidIO specification states that the Physical layer must add a 16-bit CRC to all packets. The size of the packet determines how many CRCs are required.

- For packets of 80 bytes or fewer—header and payload data included—a single 16-bit CRC is appended to the end of the packet.
- For packets longer than 80 bytes—header and payload data included—two 16-bit CRCs are inserted; one after the 80th transmitted byte and the other at the end of the packet.

Two null padding bytes are appended to the packet if the resulting packet size is not an integer multiple of four bytes.

In variations of the RapidIO IP core that include the Transport layer, the Transport layer removes the CRC after the 80^{th} byte (if present), but does not remove the final CRC nor the padding bytes. Therefore, a packet sent to the Avalon-ST pass-through receiver interface by the Transport layer is two or four bytes longer than the equivalent packet received by the Transport layer from the Avalon-ST pass-through interface. When processing the received packets, the Logical layer modules must ignore the final CRC and padding bytes (if present). In variations of the RapidIO IP core that include only the Physical layer, the 80^{th} byte CRC of a received packet is not removed.

The receiver uses the CCITT polynomial $x^{16} + x^{12} + x^5 + 1$ to check the 16-bit CRCs that cover all packet header bits (except the first 6 bits) and all data payload, and flags CRC and packet size errors.

Low-Level Interface Transmitter

The transmitter in the low-level interface transmits output to the serial RapidIO interface. This module performs the following tasks:

- Assembles packets and control symbols into a proper output format
- Generates the 5-bit CRC to cover the 19-bit symbol and appends the CRC at the end of the symbol
- Transmits an idle sequence during port initialization and when no packets or control symbols are available to transmit
- Transmits outgoing multicast-event control symbols in response to user requests
- Transmits status control symbols and the rate compensation sequence periodically as required by the RapidIO specification

The low-level transmitter block creates and transmits outgoing multicast-event control symbols. Each time the `multicast_event_tx` input signal changes value, this block inserts a multicast-event control symbol in the outgoing bit stream as soon as possible.

In 1.25, 2.5, and 3.125 Gbaud variations, the internal transmitters are not turned off while the initialization state machine is in the *SILENT* state. Instead, while in *SILENT* state, the transmitters send a continuous stream of K28.5 characters, all of the same disparity. This behavior causes the receiving end to declare numerous disparity errors and to detect a loss of `lane_sync` as intended by the specification.

In 5.0 Gbaud variations, the internal transmitters are turned off while the initialization state machine is in the *SILENT* state. This behavior also causes the link partner to detect the need to reinitialize the RapidIO link.

Transmitter Transceiver in Variations With an Internal Transceiver

The transmitter transceiver is an embedded megafunction in the Arria GX, Arria II GX, Arria II GZ, Cyclone IV GX, Stratix II GX, or Stratix IV GX device, or an embedded Custom PHY IP core in the Arria V, Cyclone V, or Stratix V device. The 16-bit parallel input data to the transmitter is internally multiplexed to 8-bit data and 8B10B encoded. The 10-bit encoded data is then serialized and sent to differential output pins.

Protocol and Flow Control Engine

The Physical layer protocol and flow control engine uses a sliding window protocol to handle incoming and outgoing packets.This block performs the following tasks:

- Monitors incoming and outgoing packet `ackID`s to maintain proper flow
- Processes incoming control symbols
- Creates and transmits outgoing control symbols

On the receiver side, this block keeps track of the sequence of `ackID`s and determines which packets are acknowledged and which packets to retry or drop. On the transmitter side, it keeps track of the sequence of `ackID`s, tells the transmit buffer control block which packet to send, and sets the outgoing packets' `ackID`. It also tells the transmit buffer control block when a packet has been acknowledged—and can therefore be discarded from the buffers.

The Physical layer protocol and flow control engine ensures that a maximum of 31 unacknowledged packets are transmitted, and that the ackIDs are used and acknowledged in sequential order.

If the receiver cannot accept a packet due to buffer congestion, a packet-retry control symbol with the packet's ackID is sent to the transmitter. The sender then sends a restart-from-retry control symbol and retransmits all packets starting from the specified ackID. The RapidIO IP core supports receiver-controlled flow control in both directions.

If the receiver or the protocol and flow control block detects that an incoming packet or control symbol is corrupted or a link protocol violation has occurred, the protocol and flow control block enters an error recovery process. Link protocol violations include acknowledgement time-outs based on the timers the protocol and flow control block sets for every outgoing packet. In the case of a corrupted incoming packet or control symbol, and some link protocol violations, the block instructs the transmitter to send a packet-not-accepted symbol to the sender. A link-request link-response control symbol pair is then exchanged between the link partners and the sender then retransmits all packets starting from the ackID specified in the link-response control symbol. The transmitter attempts the link-request link-response control symbol pair exchange as many times as specified by the value N that you provided for the **Link-request attempts** parameter in the RapidIO parameter editor. If the protocol and control block times out awaiting the response to the Nth link-request control symbol, it declares a fatal error.

The Physical layer can retransmit any unacknowledged packet because it keeps a copy of each transmitted packet until the packet is acknowledged with a packet-accepted control symbol.

When a time-out occurs for an outgoing packet, the protocol and flow control block treats it as an unexpected acknowledge control symbol, and starts the recovery process. If a packet is retransmitted, the time-out counter is reset.

Atlantic Interface

The Physical layer sends data to the Transport layer through a slave-source Atlantic interface, and accepts packet data from the Transport layer through a slave-sink Atlantic interface. The Atlantic interface data bus is 32 bits wide in 1× variations and 64 bits wide in 4× variations of the RapidIO IP core.

In variations with only the Physical layer, the Atlantic interface is the external datapath interface. In variations with a Transport layer, the Atlantic interface is used internally as the datapath interface between the Physical layer and the Transport layer, and is not visible.

For more information about the Atlantic interface specification, refer to *FS13: Atlantic Interface*.

The Physical layer Atlantic interface asserts the arxdav signal only when at least one full packet is available to be read from the receive buffer. However, the RapidIO IP core does not wait for a full packet to arrive on the RapidIO link before it begins sending the first receive buffer block to the Atlantic interface. If the arxena signal is asserted when the arxdav signal is not asserted, the first word becomes available on the Atlantic interface, and the arxval signal is asserted, as soon as the first 64-byte

block of a packet (or the full packet if it is smaller than 64 bytes) is ready to be read from the receive buffer. Refer to Figure 4–6 on page 4–13 for the Atlantic interface signals.

For variations that do not implement the Transport layer, to minimize latency, the RapidIO IP core can start transmitting a packet on the RapidIO link before it is completely received on the transmit Atlantic interface. The RapidIO IP core also can start outputting the packet on the receive Atlantic interface before the packet is completely received from the link partner on the RapidIO interface. In this case, if a packet error is detected after transmission starts from the Atlantic link but before the entire packet has been received, the receiver Atlantic interface block asserts the `arxerr` and `arxeop` signals and the packet is terminated. User logic should drop and ignore packets for which the `arxerr` signal is asserted, because the content of these packets is not reliable.

Similarly, if the user logic must abort the transmission of a packet that it has started to transfer to the RapidIO IP core through the transmit Atlantic interface, the user logic must assert the `atxerr` and `atxeop` signals. If the packet transmission has already started on the RapidIO port, the packet is aborted with a `stomp` control symbol.

The transmit Atlantic interface has an additional output signal, `atxovf`, which indicates a transmit buffer overflow condition. If, in an attempt to start transmitting a new packet, the user logic asserts `atxena` and `atxsop` three clock cycles or more after `atxdav` is deasserted, the Atlantic-interface transmit block drops the packet, asserts `atxovf`, and ignores further input until the next assertion of `atxsop` and `atxena`.

The Atlantic interface uses the `arxerr` signal to indicate the current packet being sent to the receive Atlantic interface is invalid. As an Atlantic signal, the `arxerr` signal is synchronous to `arxclk` and is only valid when `arxval` is asserted. When asserted, `arxerr` stays asserted until the end of the packet when `arxeop` is asserted. The `arxerr` signal can be asserted for the following reasons:

- CRC error—when a CRC error is detected, the `packet_crc_error` signal is asserted for one `rxclk` clock period. If the packet size is at least 64 bytes, the `arxerr` signal is asserted. If the packet size is less than 64 bytes, the errored packet does not reach the receive Atlantic interface.
- Stomp—the `arxerr` signal is asserted if a `stomp` control symbol is received in the midst of a packet, causing it to be prematurely terminated.
- Packet size—if a received packet exceeds the allowable size, it is cut short to the maximum allowable size (276 bytes total), and `arxerr` and `arxeop` are asserted on the last word.
- Outgoing symbol buffer full—under some congestion conditions, there may be no space in the outgoing symbol buffer for the `packet_accepted` control symbol. In this case, the packet cannot be acknowledged and must be retried. Thus, `arxerr` is asserted to indicate to the downstream circuit that the received packet should be ignored because it will be retried.
- Control symbol error —if an embedded or packet-delimiting control symbol is errored, `arxerr` is asserted and the packet in which it is embedded should be retransmitted by the far end as part of the error recovery process.

- Character error—if an errored character (an invalid 10-bit code, or a character of wrong disparity) or an illegal character (any control character other than the non-delimiting Start of Control (SC) character inside a packet) is received within a packet, the `arxerr` and `arxeop` signals are asserted and the rest of the packet is dropped.

Atlantic Interface Receive Buffer and Control Block

The Atlantic-interface receiver block accepts packet data from the low-level interface receiver module and stores the data in its receive buffer. The receiver buffer is partitioned into 64-byte blocks that are allocated from a free queue and returned to the free queue when no longer needed. As many as five 64-byte blocks may be required to store a packet.

Priority Threshold Values

The Atlantic-interface receiver block implements the RapidIO specification deadlock prevention rules by accepting or retrying packets based on three programmable threshold levels, called Priority Threshold values. The algorithm uses the packet's priority field value. The block determines whether to accept or retry a packet based on its priority, the threshold values, and the number of free blocks available in the receiver buffer, using the following rules:

- Packets of priority `0` (lowest priority) are retried if the number of available free 64-byte blocks is less than the Priority 0 Threshold.
- Packets of priority `1` are retried only if the number of available free 64-byte blocks is less than the Priority 1 Threshold.
- Packets of priority `2` are retried only if the number of available free 64-byte blocks is less than the Priority 2 Threshold.
- Packets of priority `3` (highest priority) are retried only if the receiver buffer is full.

The default threshold values are:

- **Priority 2** Threshold = 10
- **Priority 1** Threshold = 15
- **Priority 0** Threshold = 20

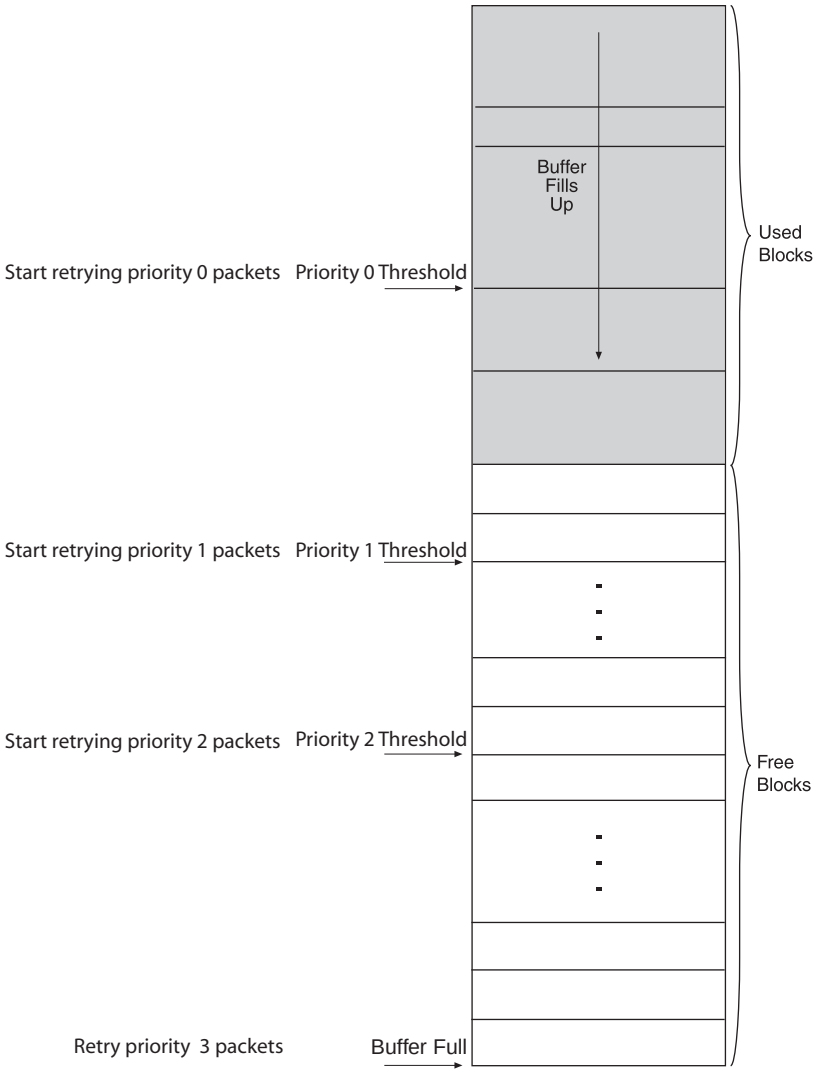
You can specify other threshold values by turning off **Auto-configured from receiver buffer size** on the **Physical Layer** page in the RapidIO parameter editor.

The RapidIO parameter editor enforces the following constraints to ensure the threshold values increase monotonically by at least the maximum size of a packet (five buffers), as required by the deadlock prevention rules:

- **Priority 2** Threshold > 9
- **Priority 1** Threshold > **Priority 2** Threshold + 4
- **Priority 0** Threshold > **Priority 1** Threshold + 4
- **Priority 0** Threshold < Number of available buffers

Figure 4–7 shows sample threshold values in context to illustrate how they work together to enforce the deadlock prevention rules.

Figure 4–7. Receiver Threshold Levels



Receive Buffer

The receive buffer provides clock decoupling between the two clock domains in the Atlantic-interface receiver block. You can specify a value of 4, 8, 16, or 32 KBytes to configure the receive buffer size.

The following fatal errors cause the receive buffer to be flushed and any stored packets to be lost:

- Receive a `port-response` control symbol with the `port_status` set to `Error`.
- Receive a `port-response` control symbol with the `port_status` set to `OK` but the `ackid_status` set to an `ackID` that is not pending (transmitted but not acknowledged yet).
- Transmitter times out while waiting for `link-response`.

- Receiver times out while waiting for link-request.

The following event also causes the receive buffer to be flushed, and any stored packets to be lost:

- Receive four consecutive link-request control symbols with the cmd set to reset-device.

Atlantic Interface Transmit Buffer and Control Block

The Atlantic-Interface transmitter block accepts packet data from the Atlantic interface and stores it in the transmit buffer for the RapidIO link low-level interface transmitter.

To meet the RapidIO specification requirements for packet priority handling and deadlock avoidance, the Atlantic-interface transmitter block includes four transmit queues and four retransmit queues, one for each priority level.

Transmit and Retransmit Queues

As packets are written to the transmitter's Atlantic interface, they are added to the end of the appropriate priority transmit queue. The transmitter always transmits the packet at the head of the highest priority nonempty queue. After being transmitted, the transmit buffer moves the packet to the corresponding priority retransmit queue.

When a packet-accepted control symbol is received for a non-acknowledged transmitted packet, the transmit buffer block removes the accepted packet from its retransmit queue.

If a packet-retry control symbol is received, all of the packets in the retransmit queues are returned to the head of the corresponding transmit queues. The transmitter sends a restart-from-retry symbol, and the transmission resumes with the highest priority packet available, possibly not the same packet that was originally transmitted and retried. If higher priority packets have been written to the Atlantic interface since the retried packet was originally transmitted, they are chosen automatically to be transmitted before lower priority packets are retransmitted.

The Physical layer protocol and flow control engine ensures that a maximum of 31 unacknowledged packets are transmitted, and that the ackIDs are used and acknowledged in incrementing order.

Transmit Buffer

The transmit buffer is the main memory in which the packets are stored before they are transmitted. The buffer is partitioned into 64-byte blocks to be used on a first-come, first-served basis by the transmit and retransmit queues.

The following fatal errors cause the transmit buffer to be flushed, and any stored packets to be lost:

- Receive a link-response control symbol with the port_status set to Error.
- Receive a link-response control symbol with the port_status set to OK but the ackid_status set to an ackID that is not pending (transmitted but not acknowledged yet).
- Transmitter times out while waiting for link-response.

- Receiver times out while waiting for `link-request`.

The following event also causes the transmit buffer to be flushed, and any stored packets to be lost:

- Receive four consecutive `link-request` control symbols with the `cmd` set to `reset-device`.

Forced Compensation Sequence Insertion

As packet data is written to the transmit Atlantic interface, it is stored in 64-byte blocks. To minimize the latency introduced by the RapidIO IP core, transmission of the packet starts as soon as the first 64-byte block is available (or the end of the packet is reached, for packets shorter than 64 bytes). Should the next 64-byte block not be available by the time the first one has been completely transmitted, `status` control symbols are inserted in the middle of the packet instead of idles as the true idle sequence can be inserted only between packets and cannot be embedded inside a packet. Embedding these status control symbols along with other symbols, such as `packet-accepted` symbols, causes the transmission of the packet to be stretched in time.

The RapidIO specification requires that compensation sequences be inserted every 5,000 code groups or columns, and that they be inserted only between packets. The RapidIO IP core checks whether the 5,000 code group quota is approaching before the transmission of every packet and inserts a compensation sequence when the number of code groups or columns remaining before the required compensation sequence insertion falls below a specified threshold.

The threshold is chosen to allow time for the transmission of a packet of maximum legal size—276 bytes—even if it is stretched by the insertion of a significant number of embedded symbols. The threshold assumes a maximum of 37 embedded symbols, or 148 bytes, which is the number of `status` control symbols that are theoretically embedded if the traffic in the other direction consists of minimum-sized packets.

Despite these precautions, in some cases—for example when using an extremely slow transmit Atlantic or Avalon clock—the transmission of a packet can be stretched beyond the point where a RapidIO link protocol compensation sequence must be inserted. In this case, the packet transmission is aborted with a `stomp` control symbol, the compensation sequence is inserted, and normal transmission resumes.

When the receive side receives the stomped packet, it marks it as errored by asserting `arxerr`. No traffic is lost and no protocol violation occurs, but an unexpected `arxerr` assertion occurs.

Transport Layer

The Transport layer is an optional module of the RapidIO IP core. It is intended for use in an endpoint processing element and must be used with at least one Logical layer module or the Avalon-ST pass-through interface.

When you create a RapidIO IP core variation using the parameter editor, you can turn on **Enable Transport Layer**, as described in "Transport Layer" on page 3–7.

If you do not turn on **Enable Transport Layer**, you define a Physical-layer-only variation. If you create a variation without a Transport layer, refer to"Physical Layer" on page 4–12 for more information.

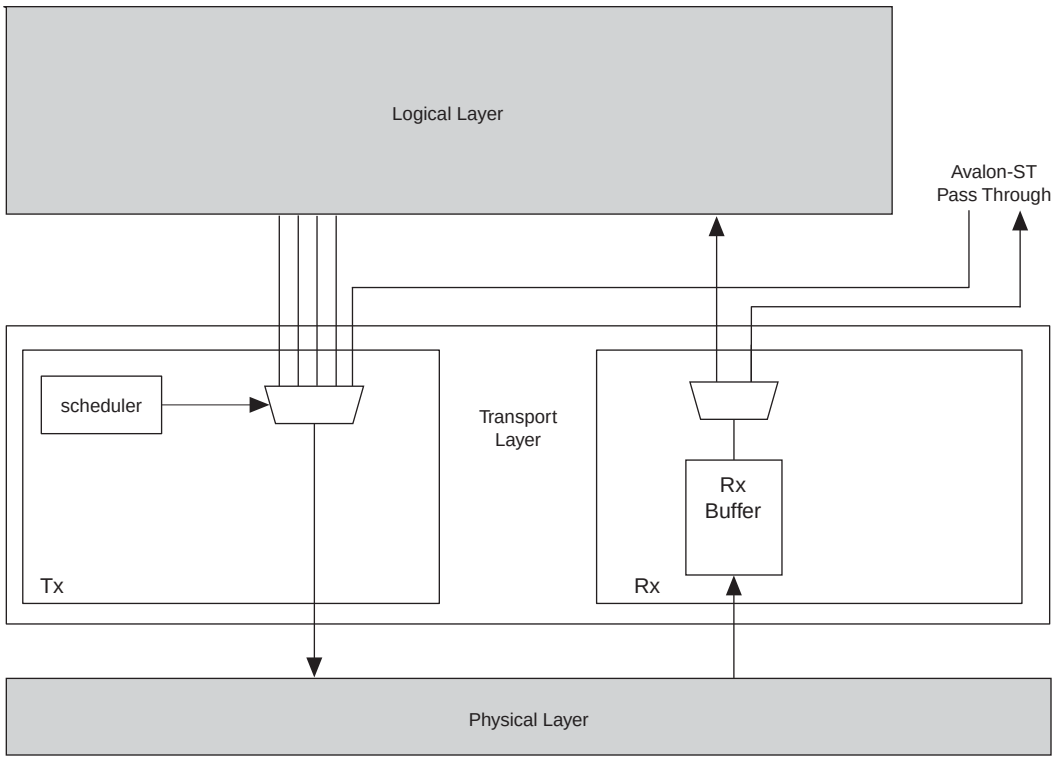
If you select **Transport Layer**, you can optionally turn on the following two additional parameters:

- **Enable Avalon-ST pass-through interface**—If you turn on this parameter, the Transport layer routes all unrecognized packets to the Avalon-ST pass-through interface.
- **Disable Destination ID checking by default**—If you turn on this parameter, request packets are considered recognized even if the destination ID does not match the value programmed in the Base Device ID CSR—Offset: 0x60. This feature enables the RapidIO IP core to process multi-cast transactions correctly.

In the Qsys and SOPC Builder design flows, **Transport Layer** is enabled automatically by default, and cannot be disabled.

The Transport layer module is divided into receiver and transmitter submodules. Figure 4–8 shows a block diagram of the Transport layer module.

Figure 4–8. Transport Layer Block Diagram



Receiver

On the receive side, the Transport layer module receives packets from the Physical layer. Packets travel through the Rx buffer, and any errored packet is eliminated. The Transport layer module routes the packets to one of the Logical layer modules or to the Avalon-ST pass-through interface based on the packet's destination ID, format type (`ftype`), and target transaction ID (`targetTID`) header fields. The destination ID matches only if the transport type (`tt`) field matches.

Packets with a destination ID different from the content of the relevant Base Device ID CSR ID field are routed to the Avalon-ST pass-through interface, unless you disable destination ID checking and the packet is a request packet with a `tt` field that matches the device ID width setting of the IP core. If you disable destination ID checking, the packet is a request packet with a supported `ftype`, and the `tt` field matches the device ID width setting of the current RapidIO IP core, the packet is routed to the appropriate Logical layer.

- Packets with unsupported `ftype` are routed to the Avalon-ST pass-through interface. Request packets with a supported `ftype` and a `tt` value that matches the RapidIO IP core's device ID width, but an unsupported `ttype` are routed to the Logical layer supporting the `ftype`. The Logical layer module then performs the following tasks:
  - Sends an `ERROR` response for request packets that require a response.
  - Records an `unsupported_transaction` error in the Error Management extension registers.
- Packets that would be routed to the Avalon-ST pass-through interface, in the case that the RapidIO IP core does not implement an Avalon-ST pass-through interface, are dropped. In this case, the Transport layer module asserts the `rx_packet_dropped` signal.
- `ftype=13` response packets are routed based on the value of their target transaction ID (`targetTID`) field. Each Logical layer module is assigned a range of transaction IDs (Table 4–7 specifies these ranges). If the transaction ID of a received response packet is not within one of the ranges assigned to one of the enabled Logical layer modules, the packet is routed to the pass-through interface.

Packets marked as errored by the Physical layer's assertion of `arxerr` (for example, packets with a CRC error or packets that were stomped) are filtered out and dropped from the stream of packets sent to the Logical layer modules or pass-through interface. In these cases, the `rx_packet_dropped` output signal is not asserted.

Transaction ID Ranges

To limit the required storage, a single pool of transaction IDs is shared between all destination IDs, although the RapidIO specification allows for independent pools for each Source-Destination pair. Further simplifying the routing of incoming `ftype=13` response packets to the appropriate Logical layer module, the Input-Output Avalon-MM slave module and the Doorbell Logical layer module are each assigned an exclusive range of transaction IDs that no other Logical layer module can use for transmitted request packets that expect an `ftype=13` response packet. Table 4–7 shows the transaction ID ranges assigned to various Logical layers.

Table 4–7. Transaction ID Ranges and Assignments (Part 1 of 2)

Range	Assignments
0–63	This range of Transaction IDs is used for `ftype=8` responses by the Maintenance Logical layer Avalon-MM slave module.
64–127	`ftype=13` responses in this range are reserved for exclusive use by the Input-Output Logical layer Avalon-MM slave module.

Table 4–7. Transaction ID Ranges and Assignments (Part 2 of 2)

Range	Assignments
128–143	`ftype=13` responses in this range are reserved for exclusive use by the Doorbell Logical layer module.
144–255	This range of Transaction IDs is currently unused and is available for use by Logical layer modules connected to the pass-through interface.

Response packets of `ftype=13` with transaction IDs outside the 64–143 range are routed to the Avalon-ST pass-through interface. Transaction IDs in the 0-63 range should not be used if the Maintenance Logical layer Avalon-MM slave module is instantiated because their use might cause the uniqueness of transaction ID rule to be violated.

If the Input-Output Avalon-MM slave module or the Doorbell Logical layer module is not instantiated, response packets in the corresponding Transaction IDs ranges for these layers are routed to the Avalon-ST pass-through interface.

Transmitter

On the transmit side, the Transport layer module uses a round-robin scheduler to select the Logical layer module to transmit packets. The Transport layer polls the various Logical layer modules to determine whether a packet is available. When a packet is available, the Transport layer transmits the whole packet, and then continues polling the next logical modules.

In a variation with a user-defined Logical layer connected to the Avalon-ST pass-through interface, you can abort the transmission of an errored packet by asserting the Avalon-ST pass-through interface `gen_tx_error` signal and `gen_tx_endofpacket`.

For more information about the Transport layer, refer to *Part 3: Common Transport Specification* of the *RapidIO Interconnect Specification, Revision 2.1*.

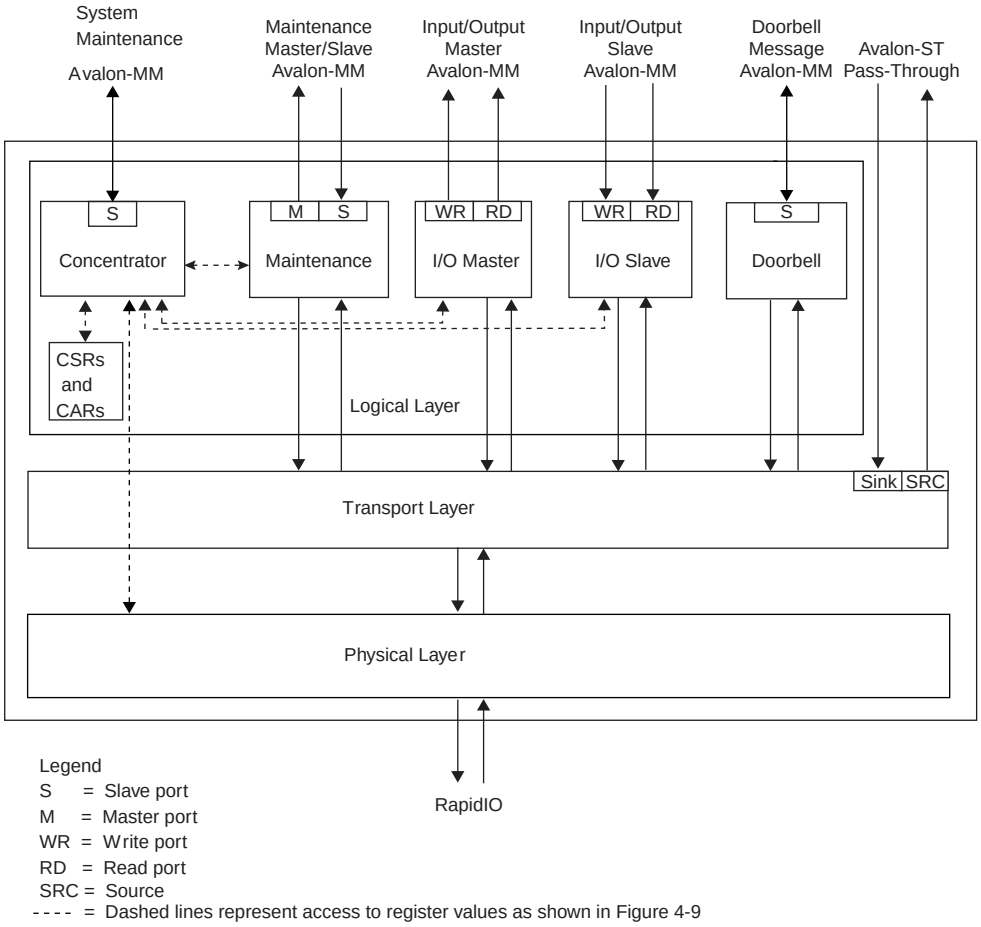
Logical Layer Modules

This section describes the features of the Logical layers, and how they integrate and interact with the Transport and Physical layers to create the three-layer RapidIO protocol. Figure 4–9 shows a high-level block diagram of the Logical layer, which consists of the following modules:

- Concentrator module that consolidates register access.
- Maintenance module that initiates and terminates `MAINTENANCE` transactions.
- I/O slave and master modules that initiate and terminate `NREAD`, `NWRITE`, `SWRITE`, and `NWRITE_R` transactions.

- Doorbell module that transacts RapidIO DOORBELL messages.

Figure 4–9. RapidIO IP Core Functional Block Diagram

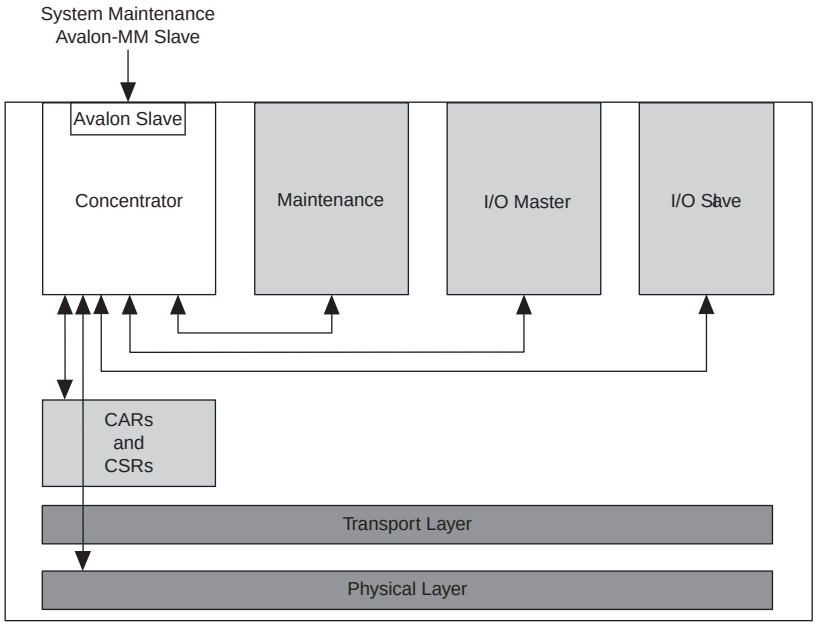


Concentrator Register Module

The Concentrator module provides an Avalon-MM slave interface that accesses all configuration registers in the RapidIO IP core, including the CARs and CSRs. The configuration registers are distributed among the implemented Logical layer modules and the Physical layer module. Figure 4–10 shows how the Concentrator module provides access to all the registers, which are implemented in different Logical layer modules. The Concentrator module is automatically included when you include the Transport layer.

Registers in the Doorbell Logical layer module are not accessed through the Concentrator. Instead, they are accessed directly through the Doorbell module's Avalon-MM slave interface.

Figure 4–10. Concentrator Module Provides Configuration Register Access



The Concentrator module provides access to the Avalon-MM slave interface and the RapidIO IP core register set. The interface supports simple reads and writes with variable latency. Accesses are to 32-bit words addressed by a 17-bit wide byte address. When accessed, the lower 2 bits of the address are ignored and assumed to be 0, which aligns the transactions to 4-byte words. The interface supports an interrupt line, `sys_mnt_s_irq`. When enabled, the following interrupts assert the `sys_mnt_s_irq` signal:

- Received port-write
- I/O read out of bounds
- I/O write out of bounds
- Invalid write
- Invalid write burstcount

For details on these and other interrupts, refer to Table 6–26 on page 6–16 and Table 6–27 on page 6–17.

Figure 4–11 and Figure 4–12 show different ways to access the RapidIO registers.

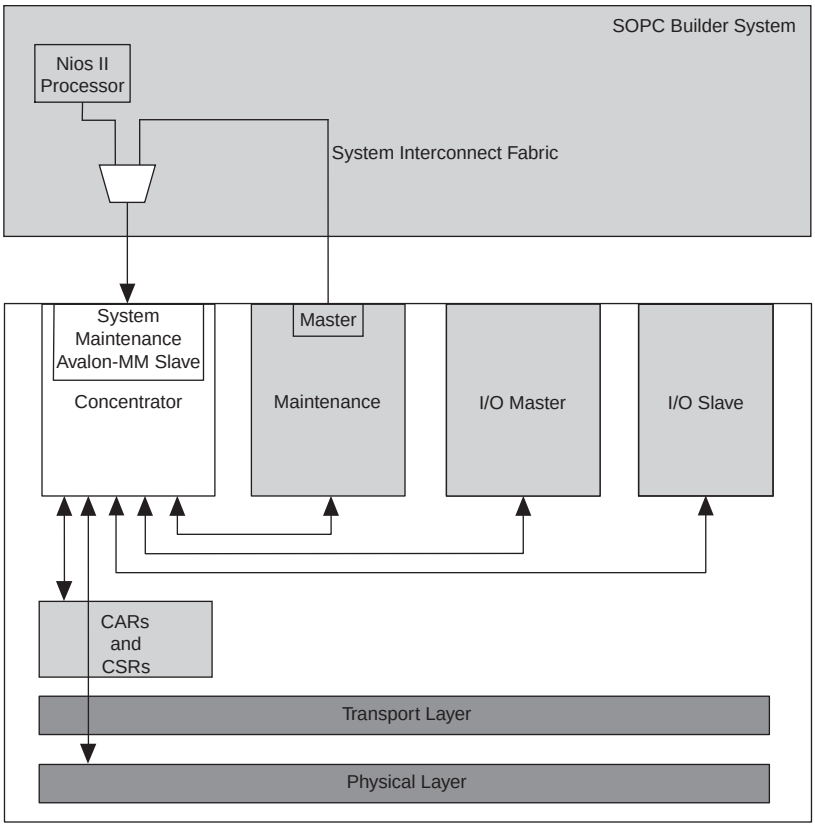
A local host can access these registers using one of the following methods:

- Qsys interconnect
- SOPC Builder system interconnect fabric
- Custom logic

A local host can access the RapidIO registers from a Qsys or SOPC Builder system as illustrated in Figure 4–11. In this figure, a Nios II processor is part of the SOPC Builder system and is configured as an Avalon-MM master that accesses the RapidIO IP core registers through the System Maintenance Avalon-MM slave. Alternatively, you can implement custom logic to access the RapidIO registers as shown in Figure 4–12.

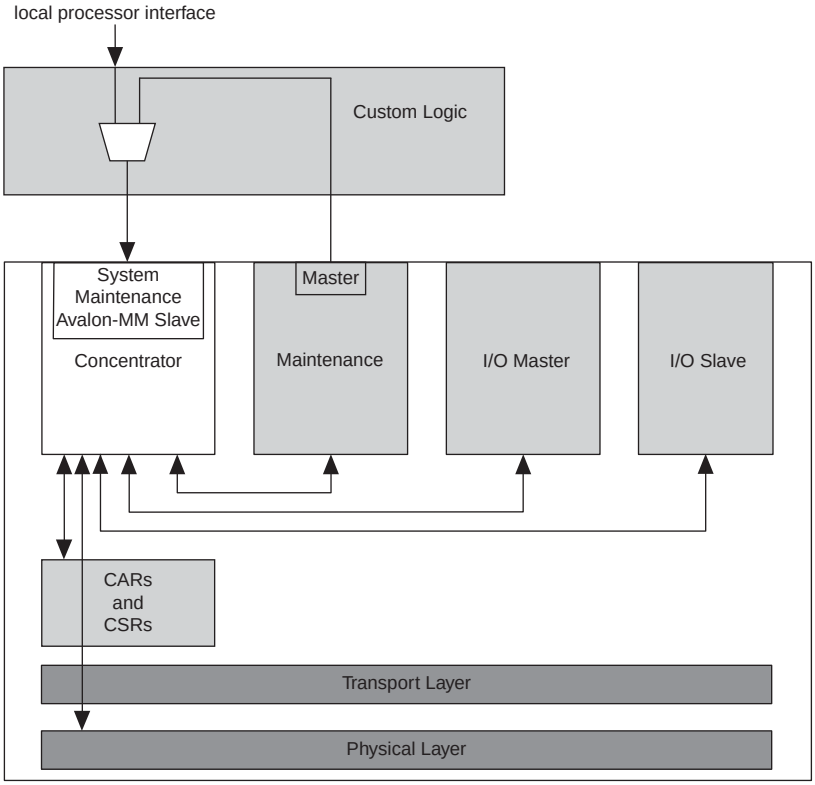
For implementation details, refer to the *SOPC Builder User Guide* or the *System Design with Qsys* section in *volume 1* of the *Quartus II Handbook*.

Figure 4–11. Local Host Accesses RapidIO Registers from an SOPC Builder System



A remote host can access the RapidIO registers by sending MAINTENANCE transactions targeted to this local RapidIO IP core. The Maintenance module processes MAINTENANCE transactions. If the transaction is a read or write, the operation is presented on the Maintenance Avalon-MM master interface. This interface must be routed to the System Maintenance Avalon-MM slave interface. This routing can be done with a Qsys or SOPC Builder system shown by the routing to the Concentrator's system Maintenance Avalon-MM slave in Figure 4–11. If you do not use a Qsys or SOPC Builder system, you can create custom logic as shown in Figure 4–12.

Figure 4–12. Custom Logic Accesses RapidIO IP core Registers



Maintenance Module

The Maintenance module is an optional component of the I/O Logical layer. The Maintenance module processes MAINTENANCE transactions, including the following transactions:

- Type 8 – MAINTENANCE reads and writes
- Type 8 – Port-write packets

When you create your custom RapidIO IP core variation in the parameter editor, you have the four choices for this module shown in Table 4–8.

Table 4–8. Maintenance Logical Layer Interface Options

Option	Use
Avalon-MM Master and Slave	Allows your IP core to initiate and terminate MAINTENANCE transactions
Avalon-MM Master	Restricts your IP core to terminating MAINTENANCE transactions
Avalon-MM Slave	Restricts your IP core to initiating MAINTENANCE transactions
None	Prevents your IP core from initiating or terminating MAINTENANCE transactions

If you add this module to your variation and select an **Avalon-MM Slave** interface, you must also select a **Number of Tx address translation windows**. A minimum of one window is required and a maximum of 16 windows are available.

For more information, refer to "Input/Output Maintenance Logical Layer Module" on page 3–8.

Figure 4–13 shows a high-level block diagram of the Maintenance module and the interfaces to other supporting modules. The Maintenance module can be segmented into the following four major submodules:

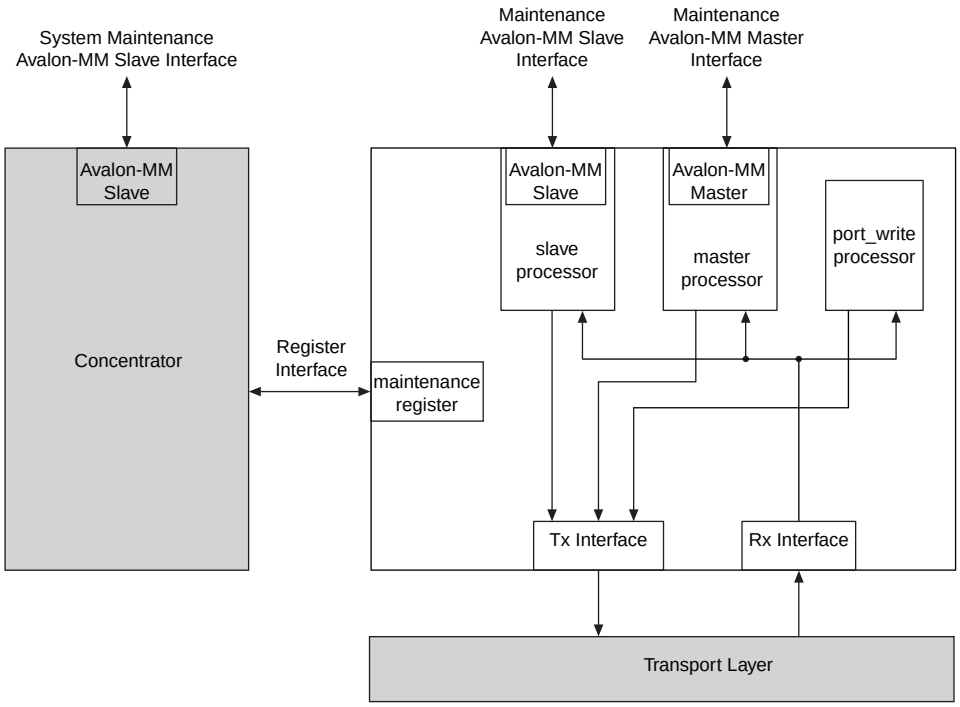
- Maintenance register
- Maintenance slave processor
- Maintenance master processor
- Port-write processor

The following interfaces are supported:

- Avalon-MM slave interface—User-exposed interface
- Avalon-MM master interface—User-exposed interface
- Tx interface—Internal interface used to communicate with the Transport layer
- Rx interface—Internal interface used to communicate with the Transport layer

- Register interface—Internal interface used to communicate with the Concentrator Module

Figure 4–13. Maintenance Module Block Diagram



Maintenance Register

The Maintenance Register module implements all of the control and status registers required by this module to perform its functions. These include registers described in Table 6–26 on page 6–16 through Table 6–32 on page 6–18. These registers are accessible through the System Maintenance Avalon-MM interface.

Maintenance Slave Processor

The Maintenance Slave Processor module performs the following tasks:

- For an Avalon read, composes the RapidIO logical header fields of a `MAINTENANCE` read request packet
- For an Avalon write, composes the RapidIO logical header fields of a `MAINTENANCE` write request packet
- Maintains status related to the composed `MAINTENANCE` packet
- Presents the composed `MAINTENANCE` packet to the Transport layer for transmission

The Avalon-MM slave interface allows you to initiate a `MAINTENANCE` read or write operation. The Avalon-MM slave interface supports the following Avalon transfers:

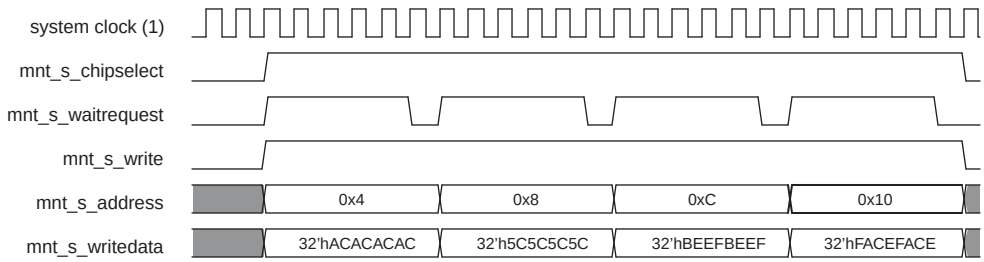
- Single slave write transfer with variable wait-states
- Pipelined read transfers with variable latency

At any time, there can be a maximum of 64 outstanding `MAINTENANCE` requests that can be `MAINTENANCE` reads, `MAINTENANCE` writes, or `port-write` requests.

Refer to the *Avalon Interface Specifications* for more details on the supported transfers.

Figure 4–14 shows the signal relationships for four write transfers on the Avalon-MM slave interface.

Figure 4–14. Write Transfers on the Avalon-MM Slave Interface

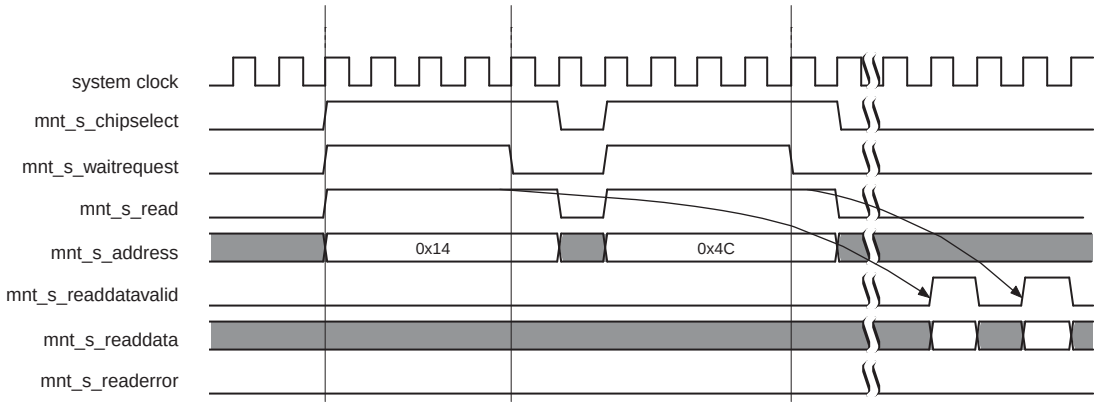


Note to Figure 4–14:

(1) Refer to Table 5–3 on page 5–2 for the system clock signal names in the MegaWizard Plug-In Manager, Qsys, and SOPC Builder design flows.

Figure 4–15 shows the signal relationships for two read transfers on the Avalon-MM interface.

Figure 4–15. Read Transfers on the Avalon-MM Slave Interface



Reads and writes on the Avalon-MM slave interface are converted to RapidIO maintenance reads and writes. The following fields of a `MAINTENANCE` type packet are assigned by the Maintenance module:

- `prio`
- `tt`
- `ftype` is assigned a value of `4'b1000`
- `dest_id`
- `src_id`

- ttype is assigned a value of 4'b0000 for reads and a value of 4'b0001 for writes
- rdsize/wrsize field is fixed at 4'b1000, because only 4-byte reads and writes are supported
- source_tid
- hop_count
- config_offset is generated by using the values programmed in the Tx Maintenance Address Translation Window registers, as described in Table 6–28 through Table 6–35.
- wdptr

Each window is enabled if the window enable (WEN) bit of the Tx Maintenance Window *n* Mask register (Table 6–30 on page 6–18) is set. Each window is defined by the following registers:

- A base register: Tx Maintenance Mapping Window *n* Base (Table 6–29 on page 6–18)
- A mask register: Tx Maintenance Mapping Window *n* Mask (Table 6–30)
- An offset register: Tx Maintenance Mapping Window *n* Offset (Table 6–31)
- A control register: Tx Maintenance Mapping Window *n* Control (Table 6–32)

For each defined and enabled window, the Avalon-MM address's least significant bits are masked out by the window mask and the resulting address is compared to the window base. If the addresses match, config_offset is created based on the following equation:

```
If (mnt_s_address & mask) == base
then config_offset = (offset[23:3] & mask[23:3])|
                (mnt_s_address[23:3] & ~mask[23:3])
```

where:

- mnt_s_address[31:0] is the Avalon-MM slave interface address
- config_offset[20:0] is the outgoing RapidIO register double-word offset
- base[31:0] is the base address register
- mask[31:0] is the mask register
- offset[23:0] is the window offset register

If the address matches multiple windows, the lowest number window register set is used.

The following fields are inserted from the control register of the mapping window that matches.

- prio
- dest_id
- hop_count

The tt value is determined by your selection of device ID width at the time you create this RapidIO IP core variation. The source_tid is generated internally and the wdptr is assigned the negation of mnt_s_address[2].

For a MAINTENANCE Avalon-MM slave write, the value on the mnt_s_writedata[31:0] bus is inserted in the payload field of the MAINTENANCE write packet.

Maintenance Master Processor

This module performs the following tasks:

- For a MAINTENANCE read, converts the received request packet to an Avalon read and presents it across the Maintenance Avalon-MM master interface.
- For a MAINTENANCE write, converts the received request packet to an Avalon write and presents it across the Maintenance Avalon-MM master interface.
- Performs accounting related to the received RapidIO MAINTENANCE read or write operation.
- For each MAINTENANCE request packet received from remote endpoints, generates a Type 8 Response packet and presents it to the Transport layer for transmission.

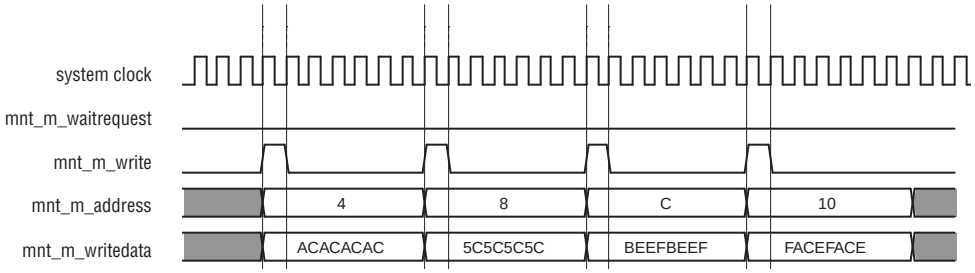
The Avalon-MM master interface supports the following Avalon transfers:

- Single master write transfer
- Pipelined master read transfers

Refer to *Avalon Interface Specifications* for details on the supported transfers.

Figure 4–16 shows the signal relationships for a sequence of four write transfers on the Maintenance Avalon-MM master interface.

Figure 4–16. Write Transfers on the Maintenance Avalon-MM Master Interface

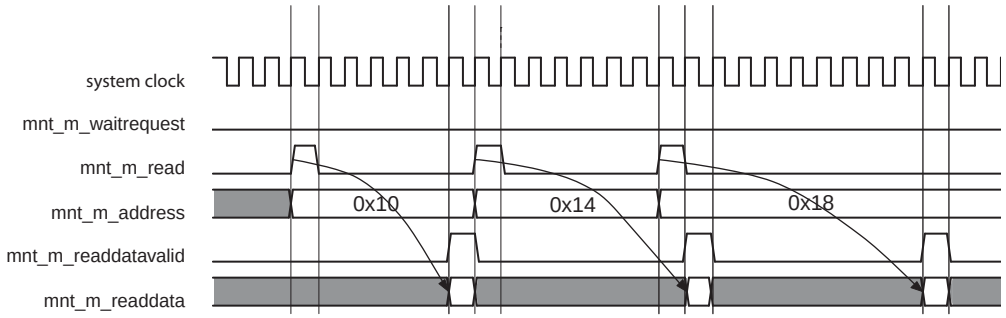


Note to Figure 4–16:

(1) Refer to Table 5–3 on page 5–2 for the system clock signal names in the MegaWizard Plug-In Manager, Qsys, and SOPC Builder design flows.

Figure 4–17 shows the signal relationships for a sequence of three read requests presented on the Maintenance Avalon-MM master interface.

Figure 4–17. Timing of a Read Request on the Maintenance Avalon-MM Master Interface



When a MAINTENANCE packet is received from a remote device, it is first processed by the Physical layer. After the Physical layer processes the packet, it is sent to the Transport layer. The Maintenance module receives the packet on the Rx interface. The Maintenance module extracts the fields of the packet header and uses them to compose the read or write transfer on the Maintenance Avalon-MM master interface. The following packet header fields are extracted:

- ttype
- rdsize/wrsize
- wdptr
- config_offset
- payload

The Maintenance module only supports single 32-bit word transfers, that is, rdsize and wrsize = 4'b1000; other values cause an error response packet to be sent.

The wdptr and config_offset values are used to generate the Avalon-MM address. The following expression is used to derive the address:

mnt_m_address = {rx_base, config_offset, wdptr, 2'b00}

where rx_base is the value programmed in the Rx Maintenance Mapping register at location 0x10088 (Table 6–28 on page 6–17).

The payload is presented on the mnt_m_writedata[31:0] bus.

Port-Write Processor

The port-write processor performs the following tasks:

- Composes the RapidIO logical header of a MAINTENANCE port-write request packet.
- Presents the port-write request packet to the Transport layer for transmission.
- Processes port-write request packets received from a remote device.
- Alerts the user of a received port-write using the sys_mnt_s_irq signal.

The port-write processor is controlled through the use of the registers that are described in the following sections:

- "Transmit Port-Write Registers" on page 6–18
- "Receive Port-Write Registers" on page 6–19

Port-Write Transmission

To send a port-write to a remote device, you must program the transmit port-write control and data registers. The `Tx Port Write Control` register is described in Table 6–33 on page 6–19 and the `Tx Port Write Buffer` is described in Table 6–35 on page 6–19. These registers are accessed using the System Maintenance Avalon-MM slave interface. The following header fields are supplied by the values stored at the `Tx Port Write Control` register:

- `DESTINATION_ID`
- `priority`
- `wrsize`

The other fields of the `MAINTENANCE port-write` packet are assigned as follows. The `ftype` is assigned a value of `4'b1000` and the `ttype` field is assigned a value of `4'b0100`. The `wdptr` and `wrsize` fields of the transmitted packet are calculated from the size of the `payload` to be sent as defined by the `size` field of the `Tx Port Write Control` register. The `source_tid` and `config_offset` are reserved and set to zero.

The `payload` is written to a `Tx Port Write Buffer` starting at address `0x10210`. This buffer can store a maximum of 64 bytes. The port-write processor starts the packet composition and transmission process after the `PACKET_READY` bit in the `Tx Port Write Control` register is set. The composed `Maintenance port-write` packet is sent to the Transport layer for transmission.

Port-Write Reception

The Maintenance module receives a `MAINTENANCE` packet on the Rx Atlantic interface from the Transport layer. The port-write processor handles `MAINTENANCE` packets with a `ttype` value set to `4'b0100`. The port-write processor extracts the following fields from the packet header and uses them to write the appropriate content to registers `Rx Port Write Control` (Table 6–36 on page 6–19) through `Rx Port Write Buffer` (Table 6–38 on page 6–20):

- `wrsize`
- `wdptr`
- `payload`

The `wrsize` and the `wdptr` determine the value of the `PAYLOAD_SIZE` field in the `Rx Port Write Status` register (Table 6–37 on page 6–20). The `payload` is written to the `Rx Port Write Buffer` starting at address `0x10260`. A maximum of 64 bytes can be written. While the `payload` is written to the buffer, the `PORT_WRITE_BUSY` bit of the `Rx Port Write Status` register remains asserted. After the `payload` is completely written to the buffer, the interrupt signal `sys_mnt_s_irq` is asserted by the Concentrator on behalf of the Port Write Processor. The interrupt is asserted only if the `RX_PACKET_STORED` bit of the `Maintenance Interrupt Enable` register (Table 6–27 on page 6–17) is set.

Maintenance Module Error Handling

The Maintenance Interrupt register (at 0x10080) and the Maintenance Interrupt Enable register (at 0x10084), described in Table 6–26 and Table 6–27, determine the error handling and reporting for MAINTENANCE packets.

The following errors can also occur for MAINTENANCE packets:

- A MAINTENANCE read or MAINTENANCE write request time-out occurs and a PKT_RSP_TIMEOUT interrupt (bit 24 of the Logical/Transport Layer Error Detect CSR, described in Table 6–52 on page 6–24) is generated if a response packet is not received within the time specified by the Port Response Time-Out Control register (Table 6–7 on page 6–6).
- The IO_ERROR_RSP (bit 31 of the Logical/Transport Layer Error Detect CSR) is set when an ERROR response is received for a transmitted MAINTENANCE packet.

For information about how the time-out value is calculated, refer to Table 6–7 on page 6–6.

For more information about the error management registers, refer to Table 6–52 on page 6–24.

Input/Output Logical Layer Modules

This section describes the following Input/Output Logical layer modules:

- "Input/Output Avalon-MM Master Module"
- "Input/Output Avalon-MM Slave Module" on page 4–44

Input/Output Avalon-MM Master Module

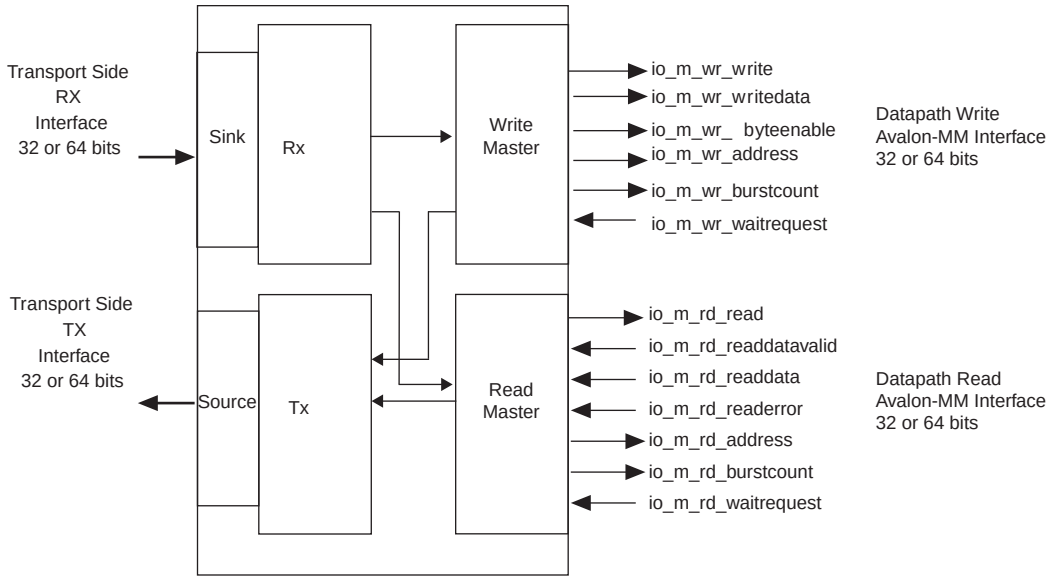
The Input/Output (I/O) Avalon-MM master Logical layer module receives RapidIO read and write request packets from a remote endpoint through the Transport layer module. The I/O Avalon-MM master module translates the request packets into Avalon-MM transactions, and creates and returns RapidIO response packets to the source of the request through the Transport layer. Figure 4–18 shows a block diagram of the I/O Avalon-MM master Logical module and its interfaces.

The I/O Avalon-MM master module is referred to as a master module because it is an Avalon-MM interface master.

To maintain full-duplex bandwidth, two independent Avalon-MM interfaces are used in the I/O master module—one for read transactions and one for write transactions.

The I/O Avalon-MM master module can process a mix of as many as seven NREAD or NWRITE_R requests simultaneously. If the Transport layer module receives an NREAD or NWRITE_R request packet while seven requests are already pending in the I/O Avalon-MM master module, the new packet remains in the Transport layer until one of the pending transactions completes.

Figure 4–18. I/O Master Block Diagram



Input/Output Avalon-MM Master Address Mapping Windows

Address mapping or translation windows are used to map windows of 34-bit RapidIO addresses into windows of 32-bit Avalon-MM addresses. Table 4–9 lists the registers used for address translation.

Table 4–9. Address Translation Registers

Registers	Location
Input/Output master base address	Table 6–39 on page 6–20
Input/Output master address mask	Table 6–40 on page 6–20
Input/Output master address offset	Table 6–41 on page 6–21

Your variation must have at least one translation window. You can change the values of the window defining registers at any time. You should disable a window before changing its window defining registers.

A window is enabled if the window enable (WEN) bit of the I/O Master Mapping Window *n* Mask register is set.

The number of mapping windows is defined by the **Number of receive address translation windows** parameter, which supports up to 16 sets of registers. Each set of registers supports one address mapping window.

For each window that is defined and enabled, the least significant bits of the incoming RapidIO address are masked out by the window mask and the resulting address is compared to the window base. If the addresses match, the Avalon-MM address is made of the least significant bits of the RapidIO address and the window offset using the following equation:

Let `rio_addr[33:0]` be the 34-bit RapidIO address, and `address[31:0]` the local Avalon-MM address.

Let `base[31:0]`, `mask[31:0]` and `offset[31:0]` be the three window-defining registers. The least significant three bits of these registers are always `3'b000`.

Starting from window `0`, for the first window in which
`((rio_addr & {xamm, mask}) == ({xamb, base} & {xamm, mask}),`

where `xamm` and `xamb` are the `Extended Address MSB` fields of the `I/O Master Mapping Window` *n* `Mask` and the `I/O Master Mapping Window` *n* `Base` registers, respectively,

let `address[31:3] = (offset[31:3] & mask[31:3]) |`
`(rio_addr[31:3] & ~mask[31:3])`

The value of `address[2]` is zero for variations with 64-bit wide datapath Avalon-MM interfaces.

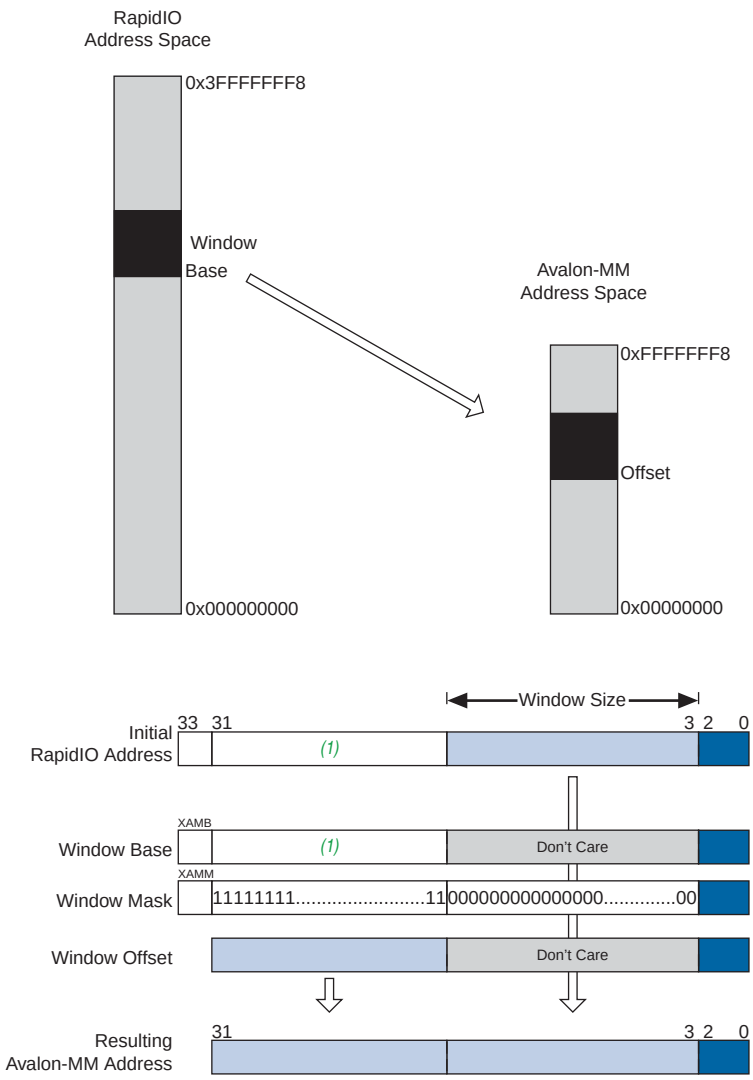
The value of `address[2]` is determined by the values of `wdptr` and `rdsize` or `wrsize` for variations with 32-bit wide datapath Avalon-MM interfaces.

The value of `address[1:0]` is always zero.

For each received `NREAD` or `NWRITE_R` request packet that does not match any enabled window, an `ERROR` response packet is returned.

Figure 4–19 shows a block diagram of the I/O master's window translation.

Figure 4–19. I/O Master Window Translation



Note to Figure 4–19:

(1) These bits must have the same value in the initial RapidIO address and in the window base.

RapidIO Packet Data wdptr and Data Size Encoding in Avalon-MM Transactions

The RapidIO IP core converts RapidIO packets to Avalon-MM transactions. The RapidIO packets' read size, write size, and word pointer fields are translated to the Avalon-MM burst count and byteenable values.

For information about the burst count values determined in the conversion process for read transactions, refer to Table 4–10. For information about the burst count and byteenable values determined in the conversion process for 32-bit datapath write transactions, by RapidIO IP core 1× variations, refer to Table 4–11. For information about the burst count and byteenable values determined in the conversion process for 64-bit datapath write transactions, by RapidIO IP core 4× variations, refer to Table 4–12.

Table 4–10. Avalon-MM I/O Master Read Transaction Burstcount (32-bit or 64-bit datapath)

RapidIO Values		Avalon-MM Burstcount Value	
rdsize (4'bxxxx)	wdptr (1'bx)	in 32-Bit Datapath	In 64-Bit Datapath
0000	0	1	1
	1	1	1
0001	0	1	1
	1	1	1
0010	0	1	1
	1	1	1
0011	0	1	1
	1	1	1
0100	0	1	1
	1	1	1
0101	0	1	1
	1	1	1
0110	0	1	1
	1	1	1
0111	0	2	1
	1	2	1
1000	0	1	1
	1	1	1
1001	0	2	1
	1	2	1
1010	0	2	1
	1	2	1
1011	0	2	1
	1	4	2
1100	0	8	4
	1	16	8
1101	0	24	12
	1	32	16
1110	0	40	20
	1	48	24
1111	0	56	28
	1	64	32

Table 4–11 lists the allowed write-request conversions for RapidIO IP core 1× variations.

Table 4–11. RapidIO Master Write Transaction Burstcount and Byteenable (32-Bit Datapath)

RapidIO Values		Avalon-MM Values		
wrsize (4'bxxxx)	wdptr (1'bx)	Maximum Burstcount (1)	Byteenable (8'b0000xxxx)	
			First Cycle or All Cycles	Second Cycle (If Different)
0000	0	1	1000	—
	1	1	1000	—
0001	0	1	0100	—
	1	1	0100	—
0010	0	1	0010	—
	1	1	0010	—
0011	0	1	0001	—
	1	1	0001	—
0100	0	1	1100	—
	1	1	1100	—
0101	0 (2)	1	1110	—
	1 (2)	1	0111	—
0110	0	1	0011	—
	1	1	0011	—
0111	0	2	1000	1111
	1	2	1111	0001
1000	0	1	1111	—
	1	1	1111	—
1001	0	2	1100	1111
	1	2	1111	0011
1010	0 (2)	2	1110	1111
	1 (2)	2	1111	0111
1011	0	2	1111	1111
	1	4	1111	—
1100	0	8	1111	—
	1	16	1111	—
1101	0 (3)	—	—	—
	1	32	1111	—
1110	0 (3)	—	—	—
	1 (3)	—	—	—
1111	0 (3)	—	—	—
	1	64	1111	—

Notes to Table 4–11:

(1) If the maximum burst count is larger than 2, the actual burst count depends on the size of the payload in the received request packet.

(2) This combination of `wdptr` and `wrsize` values should be avoided, because the resulting byteenable value is not allowed by the Avalon-MM specification.

(3) This combination of `wdptr` and `wrsize` values is reserved. If this combination is received, the RapidIO IP core declares an error.

Table 4–12 lists the allowed write-request conversions for RapidIO IP core 4× variations.

Table 4–12. RapidIO Master Write Transaction Burstcount and Byteenable (64-Bit Datapath)

RapidIO Values		Avalon-MM Values	
wrsize (4'bxxxx)	wdptr (1'bx)	Maximum Burstcount (1)	Byteenable (8'bxxxxxxxx)
0000	0	1	1000_0000
	1	1	0000_1000
0001	0	1	0100_0000
	1	1	0000_0100
0010	0	1	0010_0000
	1	1	0000_0010
0011	0	1	0001_0000
	1	1	0000_0001
0100	0	1	1100_0000
	1	1	0000_1100
0101	0 (2)	1	1110_0000
	1 (2)	1	0000_0111
0110	0	1	0011_0000
	1	1	0000_0011
0111	0 (2)	1	1111_1000
	1 (2)	1	0001_1111
1000	0	1	1111_0000
	1	1	0000_1111
1001	0 (2)	1	1111_1100
	1 (2)	1	0011_1111
1010	0 (2)	1	1111_1110
	1 (2)	1	0111_1111
1011	0	1	1111_1111
	1	2	1111_1111
1100	0	4	1111_1111
	1	8	1111_1111
1101	0 (3)	—	—
	1	16	1111_1111
1110	0 (3)	—	—
	1 (3)	—	—
1111	0 (3)	—	—
	1	32	1111_1111

Notes to Table 4–12:

(1) If the maximum burst count is larger than 2, the actual burst count depends on the size of the payload in the received request packet.

(2) This combination of `wdptr` and `wrsize` values should be avoided, because the resulting byteenable value is not allowed by the Avalon-MM specification.

(3) This combination of `wdptr` and `wrsize` values is reserved. If this combination is received, the RapidIO IP core declares an error.

Input/Output Avalon-MM Master Module Timing Diagrams

Figure 4–20 shows the timing dependencies on the Avalon-MM master interface for an incoming RapidIO NREAD transaction. Figure 4–21 shows the timing dependencies on the Avalon-MM master interface for an incoming RapidIO NWRITE transaction.

Both transaction requests are received on the RapidIO link and sent on to the Logical layer Avalon-MM master module. If the RapidIO link partner is also an Altera RapidIO IP core, the timing diagrams in "Input/Output Avalon-MM Slave Module Timing Diagrams" on page 4–55 show the same transactions as they originate on the Avalon-MM interface of the RapidIO link partner's Input/Output Avalon-MM slave module.

Figure 4–20. NREAD Transaction on the Input/Output Avalon-MM Master Interface

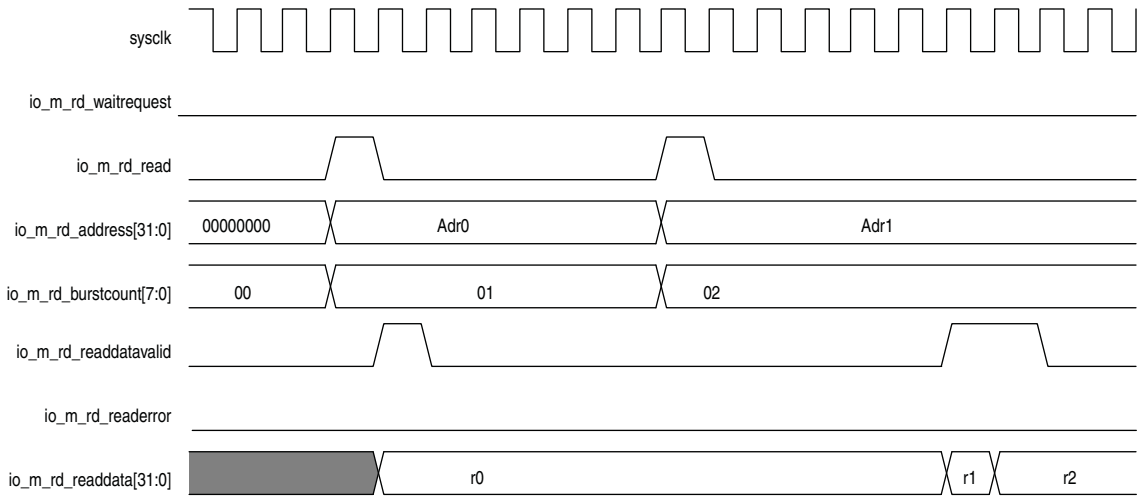
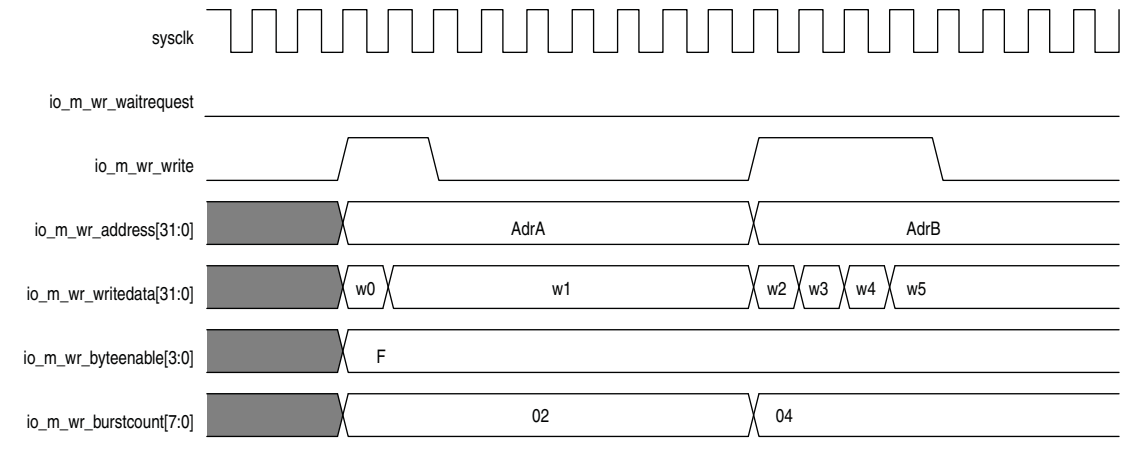


Figure 4–21. NWRITE Transaction on the Input/Output Avalon-MM Master Interface



Input/Output Avalon-MM Slave Module

The I/O Avalon-MM slave Logical layer module transforms Avalon-MM transactions to RapidIO read and write request packets that are sent through the Transport and Physical layer modules to a remote RapidIO processing element where the actual read or write transactions occur and from which response packets are sent back when required. Avalon-MM read transactions complete when the corresponding response packet is received. Figure 4–22 on page 4–45 shows a block diagram of the I/O Avalon-MM Logical layer slave module and its interfaces.

The I/O Avalon-MM slave module is referred to as a slave module because it is an Avalon-MM interface slave.

The maximum number of outstanding transactions (I/O Requests) supported is 26 (14 read requests + 12 write requests).

To maintain full-duplex bandwidth, two independent Avalon-MM interfaces are used in the I/O slave module—one for read transactions and one for write transactions.

When the read Avalon-MM slave creates a read request packet, the request is sent to both the Pending Reads buffer to wait for the corresponding response packet, and to the read request transmit buffer to be sent to the remote processing element through the Transport layer. When the read response is received, the packet's payload is used to complete the read transaction on the read Avalon-MM slave.

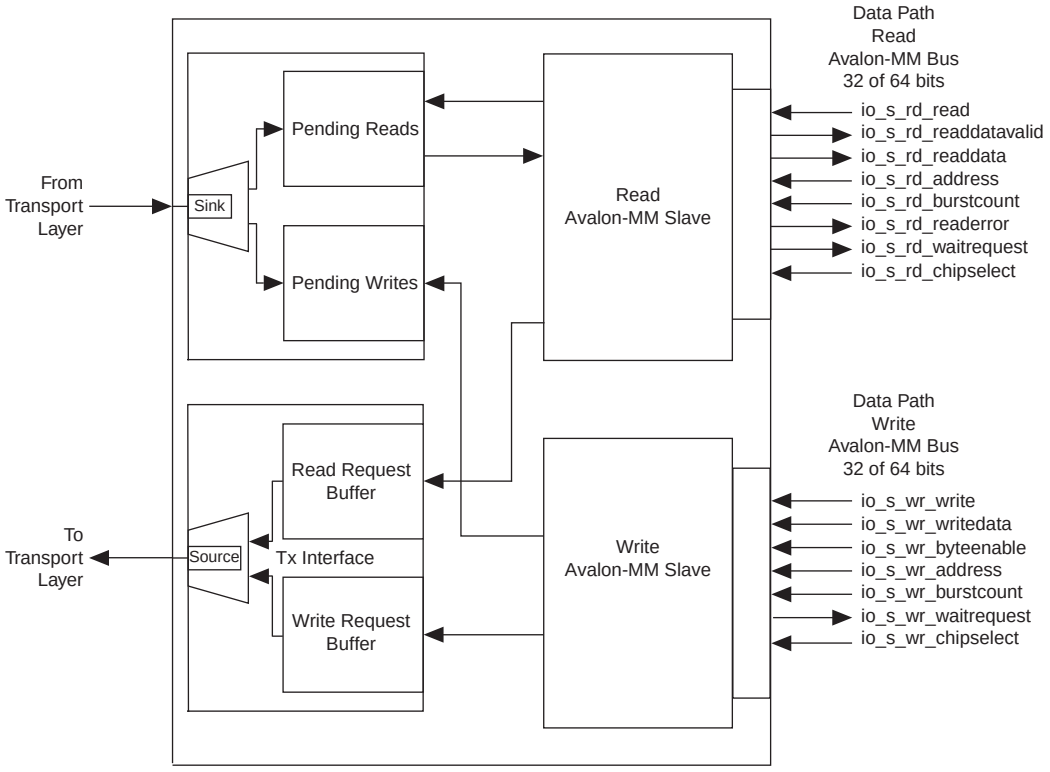
For a read operation, one of the following responses occurs:

- The read was successful. After a response packet is received, the read response and data are passed from the Pending Reads buffer back through the read Avalon-MM slave interface.
- The remote processing element is busy and the request packet is resent.
- An error or time-out occurs, which causes `io_s_rd_readerror` to be asserted on the read Avalon-MM slave interface and some information to be captured in the Error Management Extension registers.

How the write request is handled depends on the type of write request sent. For example, unlike a read request, not all write requests send tracking information to the Pending Writes buffer. `NWRITE` and `SWRITE` requests do not send write tracking information to the Pending Writes buffer. Only write requests such as `NWRITE_R`, that require a response, are sent to both the Pending Writes and Transmit buffers. Write requests are sent through the Transport and Physical layers to the remote processing element.

An outbound request that requires a response—an NWRITE_R or an NREAD transaction—is assigned a time-out value that is the sum of the VALUE field of the Port Response Time-Out Control register (Table 6–7 on page 6–6) and the current value of a free-running counter. When the counter reaches the time-out value, if the transaction has not yet received a response, the transaction times out. Refer to Table 6–7 for information about the duration of the time-out.

Figure 4–22. Input/Output Avalon-MM Slave Logical Layer Block Diagram



If you turn off the **I/O read and write order preservation** option in the RapidIO parameter editor, if a read and a write request arrive simultaneously or one clock cycle apart on the Avalon-MM interfaces, the order of transaction completion is undefined. However, if you turn on the **I/O read and write order preservation** option, the read requests buffer and the write requests buffer shown in Figure 4–22 are combined, to preserve the relative order of read and write requests that appear on the Avalon-MM interface.

Keeping Track of I/O Write Transactions

The following three registers are available to software to keep track of I/O write transactions:

- The Input/Output Slave Avalon-MM Write Transactions register described in Table 6–49 on page 6–23 holds a count of the write transactions that have been initiated on the write Avalon-MM slave interface.
- The Input/Output Slave RapidIO Write Requests register described in Table 6–50 on page 6–24 holds a count of the RapidIO write request packets that have been transferred to the Transport layer.

- The Input/Output Slave Pending NWRITE_R Transactions register described in Table 6–48 on page 6–23 holds a count of the NWRITE_R requests that have been issued but have not yet completed.

In addition, the NWRITE_RS_COMPLETED bit of the Input/Output Slave Interrupt Enable register described in Table 6–47 on page 6–23 controls a maskable interrupt in the Input/Output Slave Interrupt register described in Table 6–46 on page 6–22 that can be generated when the final pending NWRITE_R transaction completes.

You can use these registers to determine if a specific I/O write transaction has been issued or if a response has been received for any or all issued NWRITE_R requests.

Input/Output Avalon-MM Slave Address Mapping Windows

Address mapping or translation windows map windows of 32-bit Avalon-MM addresses to windows of 34-bit RapidIO addresses, and are defined by sets of the 32-bit registers in Table 4–13.

Table 4–13. Address Mapping and Translation Registers

Registers	Location
Input/Output slave base address	Table 6–42 on page 6–21
Input/Output slave address mask	Table 6–43 on page 6–21
Input/Output slave address offset	Table 6–44 on page 6–21
Input/Output slave packet control information (for packet header)	Table 6–45 on page 6–22

A base register, a mask register, and an offset register define a window. The control register stores information used to prepare the packet header on the RapidIO side of the transaction, including the target device's destination ID, the request packet's priority, and selects between the three available write request packet types: NWRITE, NWRITE_R and SWRITE. Figure 4–23 on page 4–48 illustrates this address mapping.

You can change the values of the window-defining registers at any time, even after sending a request packet and before receiving its response packet. However, you should disable a window before changing its window-defining registers. A window is enabled if the window enable (WEN) bit of the Input/Output Slave Mapping Window *n* Mask register is set, where *n* is the number of the transmit address translation window.

The number of mapping windows is defined by the parameter **Number of transmit address translation windows**; up to 16 windows are supported. Each set of registers supports one external host or entity at a time. Your variation must have at least one translation window.

For each window that is enabled, the least significant bits of the Avalon-MM address are masked out by the window mask and the resulting address is compared to the window base. If the addresses match, the RapidIO address in the outgoing request packet is made of the least significant bits of the Avalon-MM address and the window offset using the following equation:

Let avalon_address[31:0] be the 32-bit Avalon-MM address, and rio_addr[33:0] be the RapidIO address, in which rio_addr[33:32] is the 2-bit wide xamsbs field, rio_addr[31:3] is the 29-bit wide address field in the packet, and rio_addr[2:0] is implicitly defined by wdptr and rdsize or wrsize.

Let base[31:0], mask[31:0], and offset[31:0] be the values defined by the three corresponding window-defining registers. The least significant 3 bits of base, mask, and offset are fixed at 3'b000 regardless of the content of the window-defining registers.

Let xamo be the Extended Address MSBits Offset field in the Input/Output Slave Window *n* Offset register (the two least significant bits of the register).

Starting with window 0, find the first window for which

((address & mask) == (base & mask)).

Let

```
rio_addr [33:3] = {xamo, (offset [31:3] & mask [31:3]) |
 (avalon_address [31:3])}
```

If the address matches multiple windows, the lowest number window register set is used. The Avalon-MM slave interface's burstcount and byteenable signals determine the values of wdptr and rdsize or wrsize, as described in "Avalon-MM Burstcount and Byteenable Encoding in RapidIO Packets" on page 4–51.

The priority and DESTINATION_ID fields are inserted from the control register.

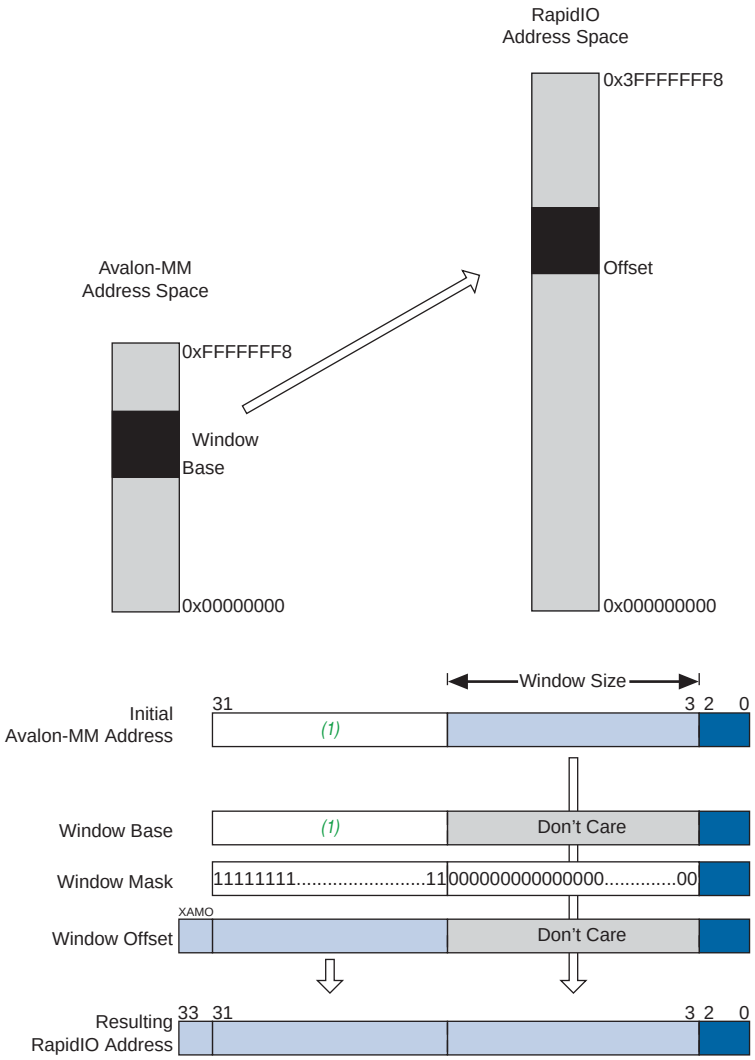
If the address does not match any window the following events occur:

- An interrupt bit, either WRITE_OUT_OF_BOUNDS or READ_OUT_OF_BOUNDS in the Input/Output Slave Interrupt register (Table 6–46 on page 6–22), is set.
- The interrupt signal sys_mnt_s_irq is asserted if enabled by the corresponding bit in the Input/Output Slave Interrupt Enable register (Table 6–47 on page 6–23).
- The COMPLETED_OR_CANCELLED_WRITES field of the Input/Output Slave RapidIO Write Requests register is incremented if the transaction is a write request.

An interrupt is cleared by writing 1 to the interrupt register's corresponding bit location.

Figure 4–23 shows the I/O slave Logical window translation process.

Figure 4–23. Input/Output Slave Window Translation



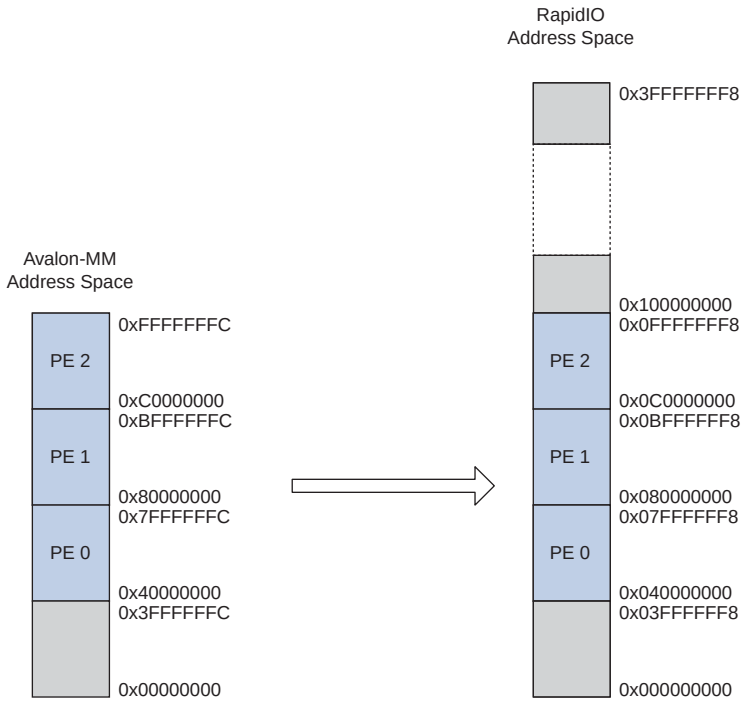
Note to Figure 4–23:

(1) These bits must have the same value in the initial Avalon-MM address and in the window base.

Input/Output Slave Translation Window Example

This section contains an example illustrating the use of I/O slave translation windows. In this example, a RapidIO IP core with 8-bit device ID communicates with three other processing endpoints through three I/O slave translation windows. For this example, the XAMO bits are set to 2'b00 for all three windows. The offset value differs for each window, which results in the segmentation of the RapidIO address space that is shown in Figure 4–24.

Figure 4–24. Input/Output Slave Translation Window Address Mapping



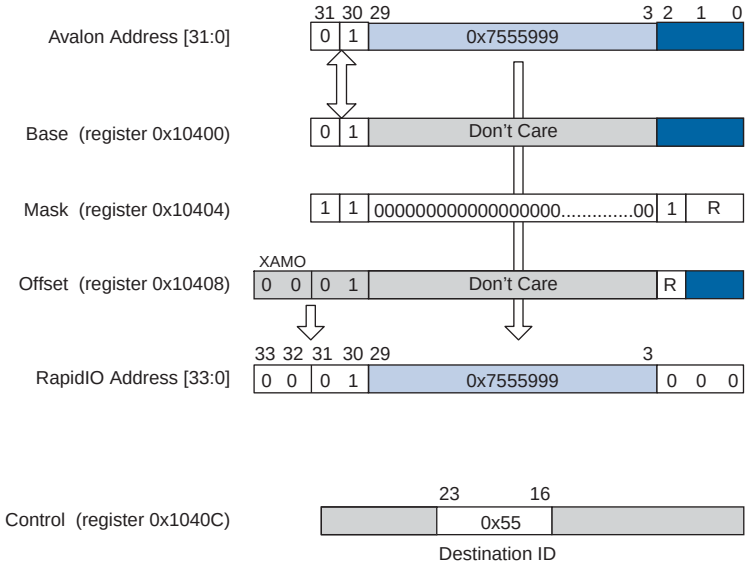
The two most significant bits of the Avalon-MM address are used to differentiate between the processing endpoints. Figure 4–26 through Figure 4–30 show the address translation implemented for each window. Each figure shows the value for the destination ID of the control register for one window.

Translation Window 0

An Avalon-MM address in which the two most significant bits have the value 2'b01 matches window 0. The RapidIO transaction corresponding to the Avalon-MM operation has a DESTINATION_ID value of 0x55. This value corresponds to processing endpoint 0.

Figure 4–25 shows address translation window 0.

Figure 4–25. Translation Window 0

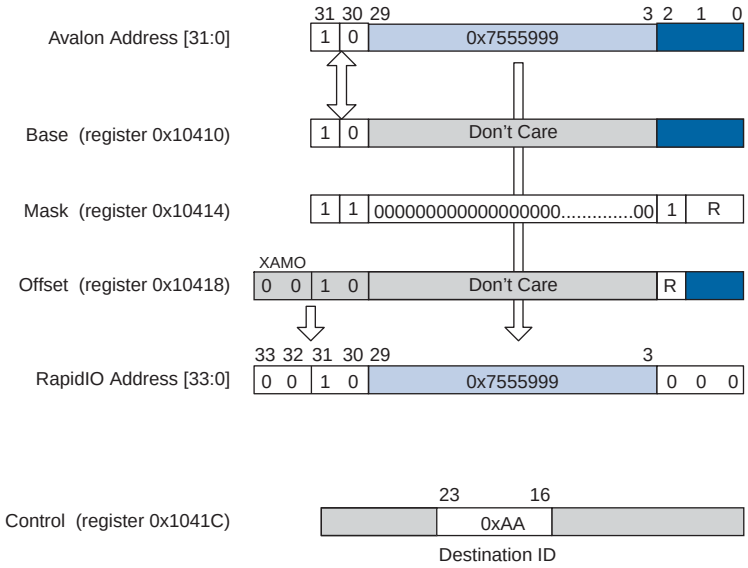


Translation Window 1

An Avalon-MM address in which the two most significant bits have a value of `2'b10` matches window 1. The RapidIO transaction corresponding to the Avalon-MM operation has a destination ID value of `0xAA`. This value corresponds to processing endpoint 1.

Figure 4–26 shows address translation window 1.

Figure 4–26. Translation Window 1

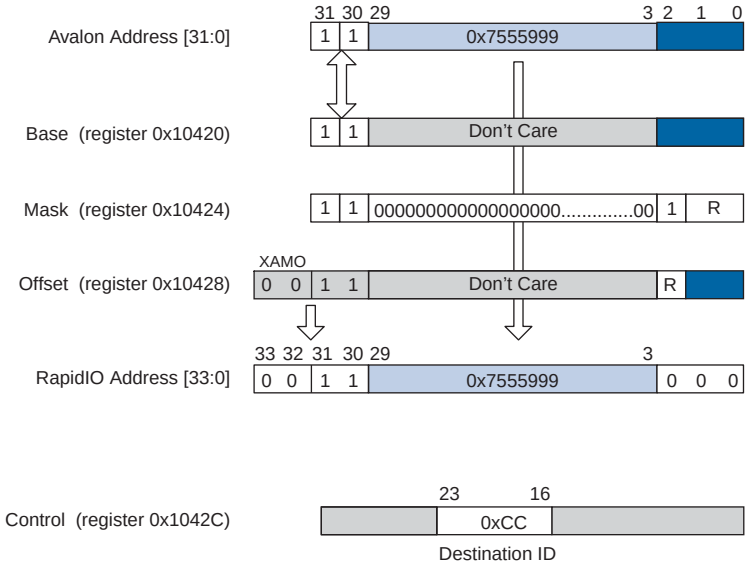


Translation Window 2

An Avalon-MM address in which the two most significant bits have a value of `2'b11` matches window 2. The RapidIO transaction corresponding to the Avalon-MM operation has a destination ID value of `0xCC`. This value corresponds to processing endpoint 2.

Figure 4–27 shows address translation window 2.

Figure 4–27. Translation Window 2



Avalon-MM Burstcount and Byteenable Encoding in RapidIO Packets

The RapidIO IP core converts Avalon-MM transactions to RapidIO packets. The Avalon-MM burst count, byteenable, and, in 32-bit variations, address bit 2 values are translated to the RapidIO packets' read size, write size, and word pointer fields.

For information about the packet size encoding used in the conversion process for 32–bit datapath read requests, refer to Table 4–14. For information about the encoding for 32-bit datapath write requests, refer to Table 4–15. For information about the encoding for 64-bit datapath conversion, refer to Table 4–16 and Table 4–17.

Table 4–14. Read Request Size Encoding (32-bit datapath) (Part 1 of 2)

Avalon-MM Values		RapidIO Values	
burstcount (1)	**address[2]** (2) **(1'bx)**	**wdptr (1'bx)**	**rdsize** (2) **(4'bxxxx)**
1	1	0	1000
1	0	1	1000
2	0	0	1011
3–4	0	1	1011
5–8	0	0	1100
9–16	0	1	1100
17–24	0	0	1101
25–32	0	1	1101
33–40	0	0	1110

Table 4–14. Read Request Size Encoding (32-bit datapath) (Part 2 of 2)

Avalon-MM Values		RapidIO Values	
burstcount (1)	address[2] (2) (1'bx)	wdptr (1'bx)	rdsize (2) (4'bxxxx)
41–48	0	1	1110
49–56	0	0	1111
57–64	0	1	1111

Notes to Table 4–14:

(1) For read transfers, the read size of the request packet is rounded up to the next supported size, but only the number of words corresponding to the requested read burst size is returned.

(2) Burst transfers of more than one Avalon-MM word must start on a double-word aligned Avalon-MM address. If the slave read burst count is larger than one and `io_s_rd_address[2]` is not zero, the transfer completes in the same manner as a failed mapping: the `READ_OUT_OF_BOUNDS` bit in the `Input/Output Slave Interrupt` register is set, `sys_mnt_s_irq` is asserted if enabled, and the transfer is marked as errored by asserting `io_s_rd_readerror` for the duration of the burst.

Table 4–15 lists the allowed burst count, byteenable, and address bit 2 value combinations for RapidIO IP core variations with a 32-bit Avalon-MM interface. Avalon-MM value combinations not listed in Table 4–15 flag interrupts in the RapidIO IP core. For more information about the relevant interrupts, refer to Table 6–46 on page 6–22.

Table 4–15. Write Request Size Encoding (32-bit datapath) (Part 1 of 2)

Avalon-MM Values			RapidIO Values	
burstcount (1)	byteenable (4'bxxxx)	address [2] (2) (1'bx)	wdptr (1'bx)	wrsize (4'bxxxx)
1	1000	1	0	0000
1	0100	1	0	0001
1	0010	1	0	0010
1	0001	1	0	0011
1	1000	0	1	0000
1	0100	0	1	0001
1	0010	0	1	0010
1	0001	0	1	0011
1	1100	1	0	0100
1	1110 (3)	1	0	0101
1	0011	1	0	0110
1	1100	0	1	0100
1	0111 (3)	0	1	0101
1	0011	0	1	0110
1	1111	1	0	1000
1	1111	0	1	1000

Table 4–15. Write Request Size Encoding (32-bit datapath) (Part 2 of 2)

Avalon-MM Values			RapidIO Values	
burstcount (1)	**byteenable (4'bxxxx)**	**address [2]** (2) **(1'bx)**	**wdptr (1'bx)**	**wrsize (4'bxxxx)**
2	1111 (4)	0	0	1011
4			1	1011
6 or 8			0	1100
10, 12, 14, 16			1	1100
18, 20, 22, 24			1	1101
26, 28, 30, 32			1	1101
34, 36, 38, 40			0	1110
42, 44, 46, 48			1	1110
50, 52, 54, 56			0	1111
58, 60, 62, 64			1	1111

Notes to Table 4–15:

(1) For write transfers in variations with 32-bit wide datapaths, odd burst sizes other than 1 are not supported. If one occurs, the `INVALID_WRITE_BURSTCOUNT` bit in the `Input/Output Slave Interrupt` register is set, causing `sys_mnt_s_irq` to be asserted if enabled.

(2) Burst transfers of more than one Avalon-MM word must start on a double-word aligned Avalon-MM address. If `io_s_wr_burstcount` is larger than one and `io_s_wr_address[2]` is not zero, the transfer completes in the same manner as a failed mapping: the `WRITE_OUT_OF_BOUNDS` bit in the `Input/Output Slave Interrupt` register is set and `sys_mnt_s_irq` is asserted if enabled.

(3) This is not a legal Avalon-MM byteenable pattern, but the RapidIO IP core supports it if user logic generates it.

(4) For all Avalon-MM write transfers with burstcount larger than 1, `io_s_wr_byteenable` must be set to `4'b1111`. If it is not, the transfer fails: the `INVALID_WRITE_BYTEENABLE` bit in the `Input/Output Slave Interrupt` register is set and `io_s_mnt_irq` is asserted if enabled.

Table 4–16 lists the allowed read-request size encodings for RapidIO IP core variations with a 64-bit Avalon-MM interface.

Table 4–16. Read Request Size Encoding (64-bit datapath)

Avalon-MM Values	RapidIO Values	
burstcount (1)	**wdptr (1'bx)**	**rdsize** (1) **(4'bxxxx)**
1	1'b0	4'b1011
2	1'b1	4'b1011
3–4	1'b0	4'b1100
5–8	1'b1	4'b1100
9–12	1'b0	4'b1101
13–16	1'b1	
17–20	1'b0	4'b1111
21–24	1'b1	
25–28	1'b0	
29–32	1'b1	

Note to Table 4–16:

(1) For read transfers, the read size of the request packet is rounded up to the next supported size, but only the number of words corresponding to the requested read burst size are returned.

Table 4–17 lists the allowed burst count and byteenable combinations for RapidIO IP core variations with a 64-bit Avalon-MM interface. Avalon-MM value combinations not listed in Table 4–17 flag interrupts in the RapidIO IP core. For more information about the relevant interrupts, refer to Table 6–46 on page 6–22.

Table 4–17. Write Request Size Encoding (64-bit datapath)

Avalon-MM Values		RapidIO Values	
burstcount	**byteenable (8'bxxxx_xxxx)**	**wdptr (1'bx)**	**wrsize (4'bx)**
1	1000_0000	0	0000
1	0100_0000	0	0001
1	0010_0000	0	0010
1	0001_0000	0	0011
1	0000_1000	1	0000
1	0000_0100	1	0001
1	0000_0010	1	0010
1	0000_0001	1	0011
1	1100_0000	0	0101
1	1110_0000 (1)	0	0110
1	0011_0000	0	0111
1	1111_1000 (1)	0	1000
1	0000_1100	1	1000
1	0000_0111 (1)	1	1001
1	0000_0011	1	1001
1	0001_1111 (1)	1	1010
1	1111_0000	0	1000
1	0000_1111	1	1000
1	1111_1100	0	1001
1	0011_1111	1	1001
1	1111_1110 (1)	0	1010
1	0111_1111 (1)	1	1010
1	1111_1111	0	1011
2	1111_1111 (2)	1	1011
3–4		0	1100
5–8		1	1100
9–12		1	1101
13–16			
17–20		1	1111
21–24			
25–28			
29–32			

Notes to Table 4–17:

(1) This is not a legal Avalon-MM byteenable pattern, but the RapidIO IP core supports it if user logic generates it.

(2) For all Avalon-MM write transfers with burstcount larger than 1, `io_s_wr_byteenable` must be set to `8'b1111_1111`. If it is not, the transfer fails: the `INVALID_WRITE_BYTEENABLE` bit in the `Input/Output Slave Interrupt` register is set and `io_s_mnt_irq` is asserted if enabled.

Input/Output Avalon-MM Slave Module Timing Diagrams

Figure 4–28 shows the timing dependencies on the Avalon-MM slave interface for an outgoing RapidIO NREAD request. Figure 4–29 shows the timing dependencies on the Avalon-MM slave interface for an outgoing NWRITE transaction. Both transaction requests originate on the Avalon-MM interface of the slave module. The timing diagrams in "Input/Output Avalon-MM Master Module Timing Diagrams" on page 4–43 show the same transactions after they are transmitted on the RapidIO link and received by an Altera RapidIO IP core link partner, when they are sent out as Avalon-MM requests by an Input/Output Avalon-MM master module in the partner RapidIO IP core.

Figure 4–28. NREAD Transaction on the Input/Output Avalon-MM Slave Interface

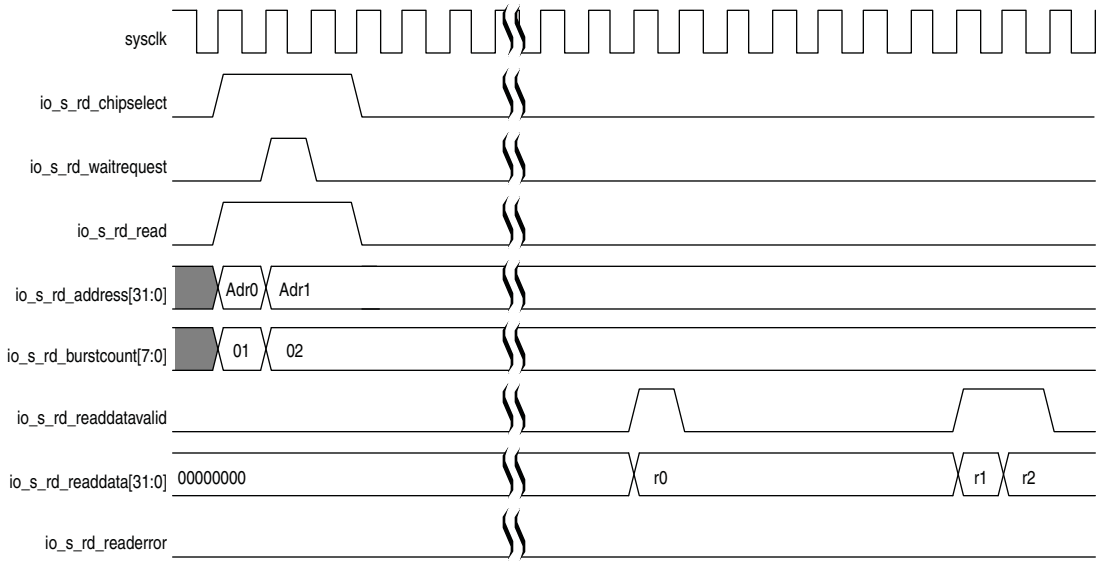
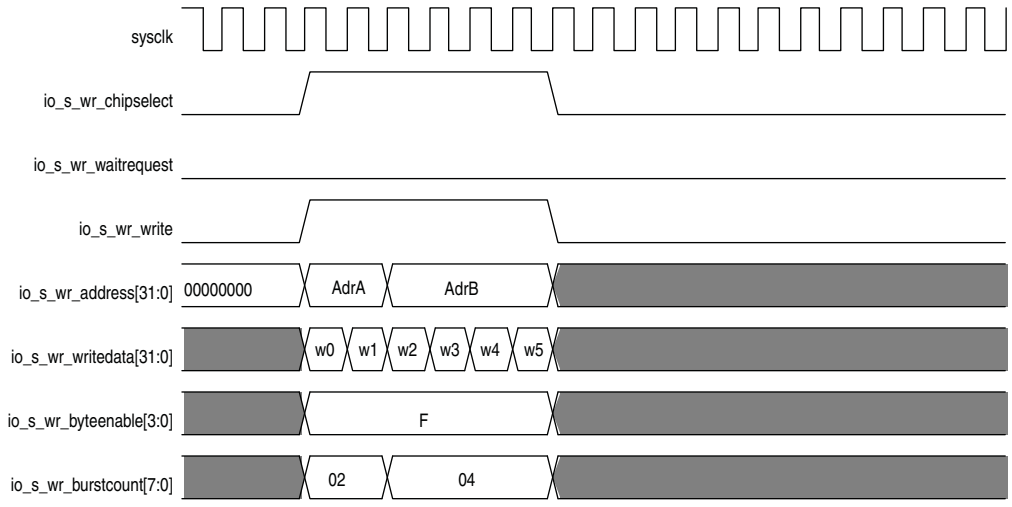


Figure 4–29. NWRITE Transaction on the Input/Output Avalon-MM Slave Interface



Doorbell Module

The Doorbell module provides support for Type 10 packet format (DOORBELL class) transactions, allowing users to send and receive short software-defined messages to and from other processing elements connected to the RapidIO interface.

Figure 4–9 on page 4–25 shows how the Doorbell module is connected to the Transport layer module. In a typical application the Doorbell module's Avalon-MM slave interface is connected to the system interconnect fabric, allowing an Avalon-MM master to communicate with RapidIO devices by sending and receiving DOORBELL messages.

When you configure the RapidIO IP core, you can enable or disable the DOORBELL operation feature, depending on your application requirements. If you do not need the DOORBELL feature, disabling it reduces device resource usage. If you enable the feature, a 32–bit Avalon-MM slave port is created that allows the RapidIO MegaCore to receive, generate, or both receive and generate RapidIO DOORBELL messages.

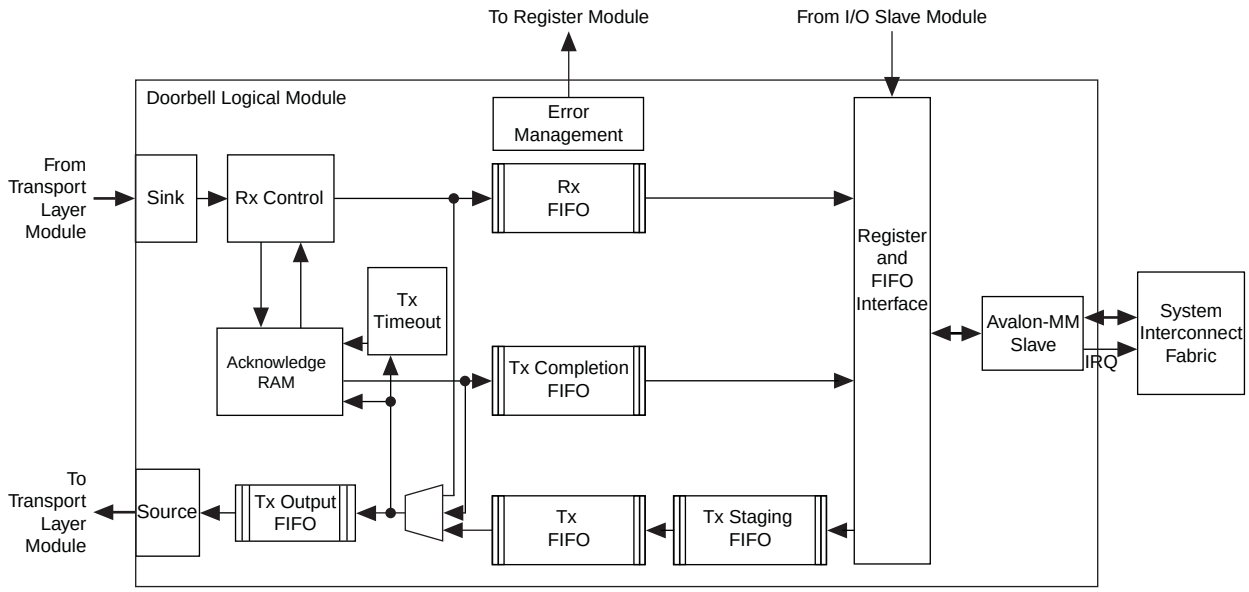
Doorbell Module Block Diagram

Figure 4–30 illustrates the Doorbell module. This module includes a 32–bit Avalon-MM slave interface to the user interface. The Doorbell module contains the following logic blocks:

- Register and FIFO interface that allows an external Avalon-MM master to access the Doorbell module's internal registers and FIFO buffers.
- Tx output FIFO that stores the outbound DOORBELL and response packets waiting for transmission to the Transport layer module.
- Acknowledge RAM that temporarily stores the transmitted DOORBELL packets pending responses to the packets from the target RapidIO device.
- Tx time-out logic that checks the expiration time for each outbound Tx DOORBELL packet that is sent.
- Rx control that processes DOORBELL packets received from the Transport layer module. Received packets include the following packet types:
  - Rx DOORBELL request.
  - Rx response DONE to a successfully transmitted DOORBELL packet.
  - Rx response RETRY to a transmitted DOORBELL message.
  - Rx response ERROR to a transmitted DOORBELL message.
- Rx FIFO that stores the received DOORBELL messages until they are read by an external Avalon-MM master device.
- Tx FIFO that stores DOORBELL messages that are waiting to be transmitted.
- Tx staging FIFO that stores DOORBELL messages until they can be passed to the Tx FIFO. The staging FIFO is present only if you select **Prevent doorbell messages from passing write transactions** in the RapidIO parameter editor.
- Tx completion FIFO that stores the transmitted DOORBELL messages that have received responses. This FIFO also stores timed out Tx DOORBELL requests.

- Error Management module that reports detected errors, including the following errors:
  - Unexpected response (a response packet was received, but its `TransactionID` does not match any pending request that is waiting for a response).
  - Request time-out (an outbound `DOORBELL` request did not receive a response from the target device).

Figure 4–30. Doorbell Module Block Diagram



Preserving Transaction Order

If you select **Prevent doorbell messages from passing write transactions** in the RapidIO parameter editor, each `DOORBELL` message from the Avalon-MM interface is kept in the Tx staging FIFO until all I/O write transactions that started on the write Avalon-MM slave interface before this `DOORBELL` message arrived on the Doorbell module Avalon-MM interface have been transmitted to the Transport layer. An I/O write transaction is considered to have started before a `DOORBELL` transaction if the `io_s_wr_write` and `io_s_wr_chipselect` signals are asserted while the `io_s_wr_waitrequest` signal is not asserted, on a cycle preceding the cycle on which the `drbell_s_write` and `drbell_s_chipselect` signals are asserted for writing to the `Tx Doorbell` register while the `drbell_s_waitrequest` signal is not asserted.

If you do not select **Prevent doorbell messages from passing write transactions** in the RapidIO parameter editor, the Doorbell Tx staging FIFO is not configured in the RapidIO IP core.

Doorbell Message Generation

To generate a `DOORBELL` request packet on the RapidIO serial interface, follow these steps, using the set of registers described in "Doorbell Message Registers" on page 6–26:

1. Optionally enable interrupts by writing the value 1 to the appropriate bit of the `Doorbell Interrupt Enable` register (Table 6–66).

2. Optionally enable confirmation of successful outbound messages by writing 1 to the COMPLETED bit of the Tx Doorbell Status Control register (Table 6–65).
3. Set up the priority field of the Tx Doorbell Control register (Table 6–60).
4. Write the Tx Doorbell register (Table 6–61) to set up the DESTINATION_ID and Information fields of the generated DOORBELL packet format.

Before writing to the Tx Doorbell register you must be certain that the Doorbell module has available space to accept the write data. Ensuring sufficient space exists avoids a waitrequest signal assertion due to a full FIFO. When the waitrequest signal is asserted, you cannot perform other transactions on the DOORBELL Avalon-MM slave port until the current transaction is completed. You can determine the combined fill level of the staging FIFO and the Tx FIFO by reading the Tx Doorbell Status register (Table 6–62). The total number of Doorbell messages stored in the staging FIFO and the Tx FIFO, together, is limited to 16 by the assertion of the drbell_s_waitrequest signal.

After a write to the Tx Doorbell register is detected, internal control logic generates and sends a Type 10 packet based on the information in the Tx Doorbell and Tx Doorbell Control registers. A copy of the outbound DOORBELL packet is stored in the Acknowledge RAM.

When the response to an outbound DOORBELL message is received, the corresponding copy of the outbound message is written to the Tx Doorbell Completion FIFO (if enabled), and an interrupt is generated (if enabled) on the Avalon-MM slave interface by asserting the drbell_s_irq signal of the Doorbell module. The ERROR_CODE field in the Tx Doorbell Completion Status register (Table 6–64) indicates successful or error completion.

The corresponding interrupt status bit is set each time a valid response packet is received, and resets itself when the Tx Completion FIFO is empty. Software optionally can clear the interrupt status bit by writing a 1 to this specific bit location of the Doorbell Interrupt Status register (Table 6–67).

Upon detecting the interrupt, software can fetch the completed message and determine its status by reading the Tx Doorbell Completion (Table 6–63) register and Tx Doorbell Completion Status register (Table 6–64), respectively.

An outbound DOORBELL message is assigned a time-out value based on the VALUE field of the Port Response Time-Out Control register (Table 6–7 on page 6–6) and a free-running counter. When the counter reaches the time-out value, if the DOORBELL transaction has not yet received a response, the transaction times out. Refer to Table 6–7 for information about how the time-out value is calculated.

An outbound message that times out before its response is received is treated in the same manner as an outbound message that receives an error response: if enabled, an interrupt is generated by the Error Management module by asserting the sys_mnt_s_irq signal, and the ERROR_CODE field in the Tx Doorbell Completion Status register (Table 6–64) is set to indicate the error.

If the interrupt is not enabled, the Avalon-MM master must periodically poll the Tx Doorbell Completion Status register to check for available completed messages before retrieving them from the Tx Completion FIFO.

DOORBELL request packets for which RETRY responses are received are resent by hardware automatically. No retry limit is imposed on outbound DOORBELL messages.

Doorbell Message Reception

DOORBELL request packets received from the Transport layer module are stored in an internal buffer, and an interrupt is generated on the DOORBELL Avalon-MM slave interface, if the interrupt is enabled.

The corresponding interrupt status bit is set every time a DOORBELL request packet is received and resets itself when the Rx FIFO is empty. Software can clear the interrupt status bit by writing a 1 to this specific bit location of the Doorbell Interrupt Status register (Table 6–67).

An interrupt is generated when a valid response packet is received and when a request packet is received. Therefore, when the interrupt is generated, you must check the Doorbell Interrupt Status register to determine the type of event that triggered the interrupt.

If the interrupt is not enabled, the external Avalon-MM master must periodically poll the Rx Doorbell Status register (Table 6–59) to check the number of available messages before retrieving them from the Rx doorbell buffer.

Appropriate Type 13 response packets are generated internally and sent for all the received DOORBELL messages. A response with DONE status is generated when the received DOORBELL packet can be processed immediately. A response with RETRY status is generated to defer processing the received message when the internal hardware is busy, for example when the Rx doorbell buffer is full.

Avalon-ST Pass-Through Interface

The Avalon-ST pass-through interface is an optional interface that is generated when you select the **Avalon-ST pass-through interface** in the **Transport and Maintenance** page of the RapidIO parameter editor (refer to "Avalon-ST Pass-Through Interface" on page 3–7). If destination ID checking is enabled, all packets received by the Transport layer whose destination ID does not match this RapidIO IP core's base device ID or whose ftype is not supported by this IP core's variation are routed to the Rx Avalon-ST pass-through interface. If you disable destination ID checking, request packets are instead routed to the Rx Avalon-ST pass-through interface only if they have ftypes that are not supported by this IP core's variation. After packets are routed to the Rx Avalon-ST pass-through interface, they can be further examined by a local processor or parsed and processed by a custom user function.

The following applications can use the Avalon-ST pass-through interface:

- User implementation of a RapidIO function not supported by this IP core (for example, data message passing)
- User implementation of a custom function not specified by the RapidIO protocol, but which may be useful for the system application

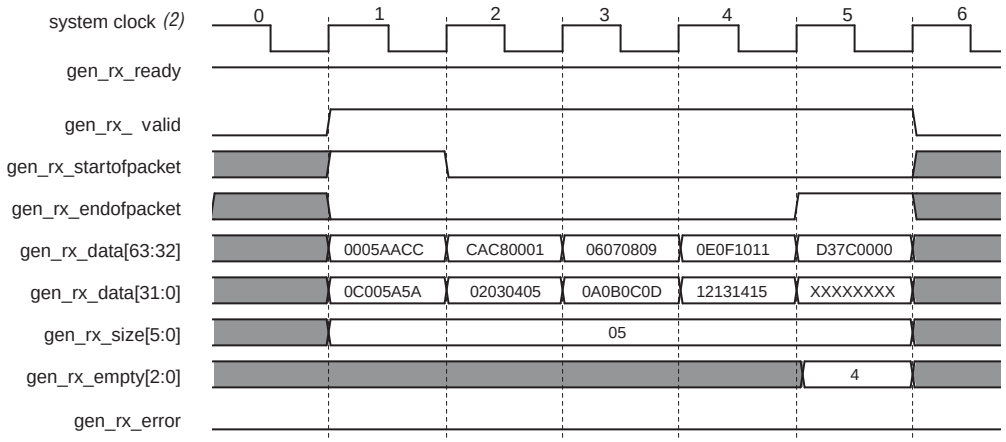
Pass-Through Interface Examples

This section contains two examples, one receiving and the other transmitting a packet through the Avalon-ST pass-through interface. The RapidIO IP core variation in the receiving example uses 8-bit device ID, and the variation in the transmitting example uses 16-bit device ID.

Packet Routed Through Rx Port on Avalon-ST Pass-Through Interface

The following example of a packet routed to the receiver Avalon-ST pass-through interface is for a variation that only has the Maintenance module and the Avalon-ST pass-through interface enabled. A packet received on the RapidIO interface with an `ftype` that does not indicate a `MAINTENANCE` transaction is routed to the receiver port of the Avalon-ST pass-through interface. The transaction diagram in Figure 4–31 shows a packet received on this interface.

Figure 4–31. Packet Received on the Avalon-ST Pass-Through Interface (1)



Notes to Figure 4–31:

(1) To improve readability of the figure, the data bus has been split in two and is displayed on two lines.

(2) Refer to Table 5–3 on page 5–2 for the system clock signal names in the MegaWizard Plug-In Manager, Qsys, and SOPC Builder design flows.

In cycle 0, the user logic indicates to the RapidIO IP core that it is ready to receive a packet transfer by asserting gen_rx_ready. In cycle 1, the IP core asserts gen_rx_valid and gen_rx_startofpacket. During this cycle, gen_rx_size is valid and indicates that five cycles are required to transfer the packet. Table 4–18 shows the RapidIO header fields and the payload carried on the gen_rx_data bus in each cycle.

Table 4–18. RapidIO Header Fields and gen_rx_data Bus Payload (Part 1 of 2)

Cycle	Field	gen_rx_data bus	Value	Comment
1	ackID	[63:59]	5'h00	
	rsvd	[58:57]	2'h0	
	CRF	[56]	1'b0	
	prio	[55:54]	2'h0	
	tt	[53:52]	2'h0	Indicates 8-bit device IDs.
	ftype	[51:48]	4'h5	A value of 5 indicates a Write Class packet.
	destinationID	[47:40]	8'haa	(1)
	sourceID	[39:32]	8'hcc	(1)
	ttype	[31:28]	4'h4	The value of 4 indicates a NWRITE transaction.
	wrsize	[27:24]	4'hc	The wrsize and wdptr values encode the maximum size of the payload field. In this example, they decode to a value of 32 bytes. For details, refer to Table 4-4 in *Part 1: Input/Output Logical Specification* of the *RapidIO Interconnect Specification, Revision 2.1*
	srcTID	[23:16]	8'h00	
	address[28:13]	[15:0]	16'h5a5a	The 29 bit address composed is 29'hb4b5959. This becomes 32'h5a5acac8, the double-word physical address.
2	address[12:0]	[63:51]	13'h1959	
	wdptr	[50]	1'b0	See description for the size field.
	xamsbs	[49:48]	2'h0	
	Payload Byte0,1	[47:32]	16'h0001	
	Payload Byte2,3	[31:16]	16'h0203	
	Payload Byte4,5	[15:0]	16'h0405	
3	Payload Byte6,7	[63:48]	16'h0607	
	Payload Byte8,9	[47:32]	16'h0809	
	Payload Byte10,11	[31:16]	16'h0a0b	
	Payload Byte12,13	[15:0]	16'h0c0d	

Table 4–18. RapidIO Header Fields and gen_rx_data Bus Payload (Part 2 of 2)

Cycle	Field	gen_rx_data bus	Value	Comment
4	Payload Byte14,15	[63:48]	16'h0e0f	
	Payload Byte16,17	[47:32]	16'h1011	
	Payload Byte18,19	[31:16]	16'h1213	
	Payload Byte20,21	[15:0]	16'h1415	
5	CRC[15:0]	[63:48]	16'hd37c	For packets with a payload greater than 80 bytes, the first CRC field is removed but the final CRC field is not removed. For packets smaller than 80 bytes, the CRC field is not removed.
	Pad bytes	[47:32]	16'h0000	The RapidIO requires that Pad bytes be added for the payload to adhere to 32-bit alignment.

Note to Table 4–18:

(1) In the case of a RapidIO IP core variation with 16-bit device ID, the destinationID and sourceID fields expand to a width of 16 bits each, and the fields described in the table rows following the destinationID field are shifted to the right and to the following clock cycles.

Bits [31:0] of the gen_rx_data bus are ignored in cycle 5 as the gen_rx_empty signals indicates that 4 bytes are not used in the end-of-packet word. In the case of a RapidIO IP core variation with 16-bit device ID, the value of gen_rx_empty would be 2, and only bits [15:0] of the gen_rx_data bus would be ignored in cycle 5.

NREAD Example Using Tx Port on Avalon-ST Pass-Through Interface

The next example shows the response to an NREAD transaction in a RapidIO IP core variation with 16-bit device ID. The response is presented on the Tx port of the Avalon-ST pass-through interface. The transaction diagram in Figure 4–32 shows the packet presented on this interface. The values captured on a rising clock edge are those shown in the previous clock cycle, because values change after the rising clock edge.

Figure 4–32. Packet Transmitted on the Avalon ST Pass-Through Interface

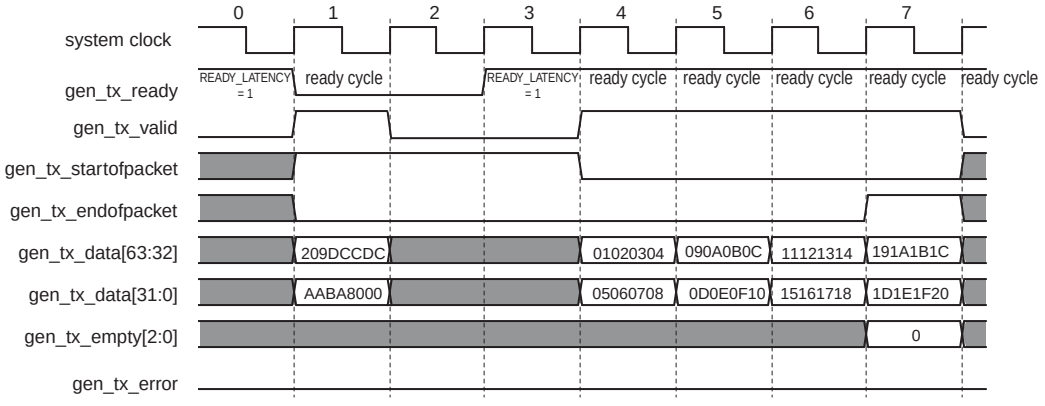


Figure 4–32 shows a response to a 32-byte NREAD request in a RapidIO IP core with 16-bit device ID. Table 4–19 shows the composition of the fields in the RapidIO packet header and the payload as they correspond to each clock cycle. The gen_tx_empty bits indicate a value of 0, because all bytes of the last word are read.

Table 4–19. RapidIO Header Fields on the gen_tx_data Bus (Part 1 of 2)

Cycle	Field	gen_tx_data bus	Value	Comment
1	ackID	[63:59]	5'h00	Value is a don't care, because it is overwritten by the Physical layer ackID value before the packet is transmitted on the RapidIO link.
	rsvd	[58:57]	2'h0	
	CRF	[56]	1'b0	
	prio	[55:54]	2'b10	Priority of the RESPONSE packet. Value must be incremented from the priority value of the REQUEST packet. For example, prio value 2'b10 indicates that the original request had a priority value of 2'b01.
	tt	[53:52]	2'h1	Indicates 16-bit device IDs
	ftype	[51:48]	4'hd	A value of 4'hd (13 decimal) indicates a Response Class packet.
	destinationId	[47:32]	16'hccdc	
	sourceId	[31:16]	16'haaba	
	ttype	[15:12]	4'h8	A value of 8 indicates a RESPONSE transaction with data payload.
	status	[11:8]	4'h0	A value of 0 indicates DONE. Requested transaction has been successfully completed.
	targetTID	[7:0]	8'h00	Value in the response packet matches the sourceTID of the corresponding request packet.
2	Payload Byte0,1	[63:48]	16'h0102	Payload double word 0
	Payload Byte2,3	[47:32]	16'h0304	
	Payload Byte4,5	[31:16]	16'h0506	
	Payload Byte6,7	[15:0]	16'h0708	
3	Payload Byte8,9	[63:48]	16'h090a	Payload double word 1
	Payload Byte10,11	[47:32]	16'h0b0c	
	Payload Byte12,13	[31:16]	16'h0d0e	
	Payload Byte14,15	[15:0]	16'h0f10	
4	Payload Byte16,17	[63:48]	16'h1112	Payload double word 2
	Payload Byte18,19	[47:32]	16'h1314	
	Payload Byte20,21	[31:16]	16'h1516	
	Payload Byte22,23	[15:0]	16'h1718	

Table 4–19. RapidIO Header Fields on the gen_tx_data Bus (Part 2 of 2)

Cycle	Field	gen_tx_data bus	Value	Comment
5	Payload Byte24,25	[63:48]	16'h191a	Payload double word 3
	Payload Byte26,27	[47:32]	16'h1b1c	
	Payload Byte28,29	[31:16]	16'h1d1e	
	Payload Byte30,31	[15:0]	16'h1f20	

Error Detection and Management

The error detection and management mechanisms in the RapidIO specification and those built into the RapidIO IP core provide a high degree of reliability. In addition to error detection, management, and recovery features, the RapidIO IP core also provides debugging and diagnostic aids.

This section describes the error detection and management features in the RapidIO IP core.

Physical Layer Error Management

Errors at the Physical layer are mainly of the following two types:

- Protocol violations
- Transmission errors

Protocol violations can be caused by a link partner that is not fully compliant to the specification, or can be a side effect of the link partner being reset.

Transmission errors can be caused by noise on the line and consist of one or more bit errors. The following mechanisms exist for checking and detecting errors:

- The receiver checks the validity of the received 8B10B encoded characters, including the running disparity.
- The receiver detects control characters changed into data characters or data characters changed into control characters, based on the context in which the character is received.
- The receiver checks the CRC of the received control symbols and packets.

The RapidIO IP core Physical layer transparently manages these errors for you. The RapidIO specification defines both input and output error detection and recovery state machines that include handshaking protocols in which the receiving end signals that an error is detected by sending a `packet-not-accepted` control symbol, the transmitter then sends an `input-status link-request` control symbol to which the receiver responds with a `link-response` control symbol to indicate which packet requires transmission. The input and output error detection and recovery state machines can be monitored by software that you create to read the status of the `Port 0 Error and Status` CSR (Table 6–10 on page 6–7).

In addition to the registers defined by the specification, the RapidIO IP core provides several output signals that enable user logic to monitor error detection and the recovery process. Refer to "Status Packet and Error Monitoring Signals" on page 5–6.

Protocol Violations

Some protocol violations, such as a packet with an unexpected `ackID` or a time-out on a packet acknowledgment, can use the same error recovery mechanisms as the transmission errors described in "Physical Layer Error Management" on page 4–64. Some protocol violations, such as a time-out on a link-request or the RapidIO IP core receiving a link-response with an `ackID` outside the range of transmitted `ackID`s, can lead to unrecoverable—or fatal—errors.

Fatal Errors

Fatal errors cause a soft reset of the Physical layer module, which clears all the transmit buffers and resets the transmission and expected `ackID` to zero. This effect also can be triggered by software by first writing a one and then a zero to the `PORT_DIS` bit of the `Port 0 Control` CSR (Table 6–11 on page 6–10).

If the link partner is reset when its expected `ackID` is not zero, a fatal error occurs when the link partner receives the next transmitted packet because the link partner's expected `ackID` is reset to zero, which causes a mismatch between the transmitted `ackID` and the expected `ackID`. The fatal error causes a soft reset of the IP core. After the soft reset completes, transmitted and expected `ackID`s are synchronized and normal operation resumes. Only the packets that were queued at the time of the fatal error are lost.

If **Send link-request reset-device on fatal errors** is turned on in the RapidIO parameter editor, fatal errors cause the transmitter to send `link-request` control symbols with `cmd` set to `reset-device` to the link partner.

Logical Layer Error Management

The Logical layer modules only need to process Logical layer errors because errors detected by the Physical layer are masked from the Logical layer module. Any packet that has the `arxerr` signal asserted is dropped in the Transport layer before it reaches the Logical layer modules.

The RapidIO specification defines the following common errors and the protocols for managing them:

- Malformed request or response packets
- Unexpected Transaction ID
- Missing response (time-out)
- Response with `ERROR` status

The RapidIO IP core implements part of the optional Error Management Extensions as defined in Part 8 of the *RapidIO Interconnect Specification Revision 2.1.* However, because the registers defined in the *Error Management Extension* specification are not all implemented in the RapidIO IP core, the error management registers are mapped in the Implementation Defined Space instead of being mapped in the Extended Features Space.

The following Error Management registers are implemented in the RapidIO IP core and provide the most useful information for error management:

- `Logical/Transport Layer Error Detect` CSR (Table 6–52)
- `Logical/Transport Layer Error Enable` CSR (Table 6–53)
- `Logical/Transport Layer Address Capture` CSR (Table 6–54)
- `Logical/Transport Layer Device ID Capture` CSR (Table 6–55)
- `Logical/Transport Layer Control Capture` CSR (Table 6–56)

For more information about these registers, refer to their descriptions in "Error Management Registers" on page 6–24.

When enabled, each error defined in the Error Management Extensions triggers the assertion of an interrupt on the `sys_mnt_s_irq` output signal of the System Maintenance Avalon-MM slave interface and causes the capture of various packet header fields in the appropriate capture CSRs.

In addition to the errors defined by the RapidIO specification, each Logical layer module has its own set of error conditions that can be detected and managed.

Maintenance Avalon-MM Slave

The Maintenance Avalon-MM slave module creates request packets for the Avalon-MM transaction on its slave interface and processes the response packets that it receives. Anomalies are reported through one or more of the following three channels:

- Standard error management registers
- Registers in the implementation defined space
- The Avalon-MM slave interface's error indication signal

The following sections describe these channels.

Standard Error Management Registers

The following standard defined error types can be declared by the I/O Avalon-MM slave module. The corresponding error bits are then set and the required packet information is captured in the appropriate error management registers.

- **IO Error Response** is declared when a response with `ERROR` status is received for a pending `MAINTENANCE` read or write request.
- **Unsolicited Response** is declared when a response is received that does not correspond to any pending `MAINTENANCE` read or write request.
- **Packet Response Timeout** is declared when a response is not received within the time specified by the `Port Response Time-Out` CSR (Table 6–7 on page 6–6) for a pending `MAINTENANCE` read or write request.

- **Illegal Transaction Decode** is declared for malformed received response packets occurring from any of the following events:
  - Response packet to pending MAINTENANCE read or write request with status not DONE nor ERROR.
  - Response packet with payload with a transaction type different from MAINTENANCE read response.
  - Response packet without payload, with a transaction type different from MAINTENANCE write response.
  - Response to a pending MAINTENANCE read request with more than 32 bits of payload. (The RapidIO IP core issues only 32-bit read requests.)

Registers in the Implementation Defined Space

The Maintenance register module defines the Maintenance Interrupt register (Table 6–26 on page 6–16) in which the following two bits report Maintenance Avalon-MM slave related error conditions:

- WRITE_OUT_OF_BOUNDS
- READ_OUT_OF_BOUNDS

These bits are set when the address of a write or read transfer on the Maintenance Avalon-MM slave interface falls outside of all the enabled address mapping windows. When these bits are set, the system interrupt signal sys_mnt_s_irq is also asserted if the corresponding bit in the Maintenance Interrupt Enable register (Table 6–27 on page 6–17) is set.

Maintenance Avalon-MM Slave Interface's Error Indication Signal

The mnt_s_readerror output is asserted when a response with ERROR status is received for a MAINTENANCE read request packet, when a MAINTENANCE read times out, or when the Avalon-MM read address falls outside of all the enabled address mapping windows.

Maintenance Avalon-MM Master

The Maintenance Avalon-MM master module processes the MAINTENANCE read and write request packets that it receives and generates response packets. Anomalies are reported by generating ERROR response packets. A response packet with ERROR status is generated in the following cases:

- Received a MAINTENANCE write request packet without payload or with more than 64 bytes of payload
- Received a MAINTENANCE read request packet of the wrong size (too large or too small)
- Received a MAINTENANCE read or write request packet with an invalid rdsize or wrsize value

These errors do not cause any of the standard-defined errors to be declared and recorded in the Error Management registers.

Port-Write Reception Module

The Port-Write reception module processes receive port-write request MAINTENANCE packets. The following bits in the Maintenance Interrupt register (Table 6–26) in the implementation-defined space report any detected anomaly. The System Maintenance Avalon-MM slave port interrupt signal sys_mnt_s_irq is asserted if the corresponding bit in the Maintenance Interrupt Enable register (Table 6–27) is set.

- The PORT_WRITE_ERROR bit is set when the packet is either too small (no payload) or too large (more than 64 bytes of payload), or if the actual size of the packet is larger than indicated by the wrsize field. These errors do not cause any of the standard defined errors to be declared and recorded in the error management registers.
- The PACKET_DROPPED bit is set when a port-write request packet is received but port-write reception is not enabled (by setting bit PORT_WRITE_ENA in the Rx Port Write Control register, described in Table 6–36 on page 6–19) or if a previously received port-write has not been read out from the Rx Port Write Buffer register (Table 6–38 on page 6–20).

Port-Write Transmission Module

Port-write requests do not cause response packets to be generated. Therefore, the port-write transmission module does not detect or report any errors.

Input/Output Avalon-MM Slave

The I/O Avalon-MM slave module creates request packets for the Avalon-MM transaction on its read and write slave interfaces and processes the response packets that it receives. Anomalies are reported through one or more of the following three channels:

- Standard error management registers
- Registers in the implementation defined space
- The Avalon-MM slave interface's error indication signal

Standard Error Management Registers

The following standard defined error types can be declared by the I/O Avalon-MM slave module. The corresponding error bits are then set and the required packet information is captured in the appropriate error management registers.

- **IO Error Response** is declared when a response with ERROR status is received for a pending NREAD or NWRITE_R request.
- **Unsolicited Response** is declared when a response is received that does not correspond to any pending NREAD or NWRITE_R request.
- **Packet Response Time-Out** is declared when a response is not received within the time specified by the Port Response Time-Out Response CSR (Table 6–7 on page 6–6) for an NREAD or NWRITE_R request.

- **Illegal Transaction Decode** is declared for malformed received response packets occurring from any of the following events:
  - NREAD or NWRITE_R response packet with status not DONE nor ERROR.
  - NWRITE_R response packet with payload or with a transaction type indicating the presence of a payload.
  - NREAD response packet without payload, with incorrect payload size, or with a transaction type indicating absence of payload.

Registers in the Implementation Defined Space

The I/O Avalon-MM slave module defines the Input/Output slave interrupt registers with the following bits. For details on when these bits are set, refer to their descriptions in Table 6–46 on page 6–22.

- INVALID_WRITE_BYTEENABLE
- INVALID_WRITE_BURSTCOUNT
- WRITE_OUT_OF_BOUNDS
- READ_OUT_OF_BOUNDS

When any of these bits are set, the system interrupt signal sys_mnt_s_irq is also asserted if the corresponding bit in the Input/Output Slave Interrupt Enable register (Table 6–47 on page 6–23) is set.

The Avalon-MM Slave Interface's Error Indication Signal

The io_s_rd_readerror output is asserted when a response with ERROR status is received for an NREAD request packet, when an NREAD request times out, or when the Avalon-MM address falls outside of the enabled address mapping window. As required by the Avalon-MM interface specification, a burst in which the io_s_rd_readerror signal is asserted completes despite the error signal assertion.

Input/Output Avalon-MM Master

The I/O Avalon-MM master module processes the request packets that it receives and generates response packets when required. Anomalies are reported through one or both of the following two channels:

- Standard error management registers
- Response packets with ERROR status

Standard Error Management Registers

The following two standard defined error types can be declared by the I/O Avalon-MM master module. The corresponding bits are then set and the required packet information is captured in the appropriate error management registers.

- **Unsupported Transaction** is declared when a request packet carries a transaction type that is not supported in the Destination Operations CAR (Table 6–19 on page 6–14), whether an ATOMIC transaction type, a reserved transaction type, or an implementation defined transaction type.

- **Illegal Transaction Decode** is declared when a request packet for a supported transaction is too short or if it contains illegal values in some of its fields such as in these examples:
  - Request packet with `priority = 3`.
  - `NWRITE` or `NWRITE_R` request packets without payload.
  - `NWRITE` or `NWRITE_R` request packets with reserved `wrsize` and `wdptr` combination.
  - `NWRITE`, `NWRITE_R`, `SWRITE`, or `NREAD` request packets for which the address does not match any enabled address mapping window.
  - `NREAD` request packet with `payload`.
  - `NREAD` request with `rdsize` that is not an integral number of transfers on all byte lanes. (The Avalon-MM interface specification requires that all byte lanes be enabled for read transfers. Therefore, Read Avalon-MM master modules do not have a byteenable signal).
  - Payload size does not match the size indicated by the `rdsize` or `wrsize` and `wdptr` fields.

Response Packets with ERROR Status

An `ERROR` response packet is sent for `NREAD` and `NWRITE_R` and Type 5 `ATOMIC` request packets that cause an `Illegal Transaction Decode` error to be declared. An `ERROR` response packet is also sent for `NREAD` requests if the `io_m_rd_readerror` input signal is asserted through the final cycle of the Avalon-MM read transfer.

Avalon-ST Pass-Through Interface

Packets with valid CRCs that are not recognized as being targeted to one of the implemented Logical layer modules are passed to the Avalon-ST pass-through interface for processing by user logic.

The RapidIO IP core also provides hooks for user logic to report any error detected by a user-implemented Logical layer module attached to the Avalon-ST pass-through interface.

The transmit side of the Avalon-ST pass-through interface provides the `gen_tx_error` input signal that behaves essentially the same way as the `atxerr` input signal described in "Atlantic Interface" on page 4–16.

If **Enable Avalon-ST pass-through interface** is enabled and at least one of the **Data Messages** options **Source Operation** and **Destination Operation** is turned on in the RapidIO parameter editor, the message passing error management input ports in Table 5–24 are added to the IP core to enable integrated error management.

5. Signals

This chapter lists the RapidIO IP core signals.

Qsys allows you to export signals with different names or prefixes. Refer to the Qsys **System Contents** tab for the signals that support this capability individually, and to the Qsys **HDL Example** tab for the list of signals that are bundled together as **exported_connections**. The signals bundled in **exported_connections** all take the prefix you specify in the Qsys **System Contents** tab.

A **yes** entry in the **Exported by SOPC Builder and by Qsys** column in the following tables indicates that the signal is exported by SOPC Builder and is included in the **exported_connections** conduit in Qsys. A **no** entry indicates that the signal is not exported by SOPC Builder and is not included in the **exported_connections** conduit in Qsys.

Physical Layer Signals

Table 5–1 through Table 5–13 list the pins used by the Physical layer of the serial RapidIO IP core. Refer to Figure 4–6 on page 4–13 for details on the I/O signals.

For signals and bus widths specific to your variation, refer to the HTML file (*<variation name>*.**html**) generated in your project directory by the RapidIO parameter editor. This file is not generated in the Qsys flow.

All signals except the reference clock and reset have a suffix (*<RapidIO variation name>* as defined in SOPC Builder) added to their signal names in the SOPC Builder design flow. For example, `rd` becomes `rd_rapidio`, if `rapidio` is the variation name.

Table 5–1. RapidIO Interface

Signal	Direction	Description	Exported by SOPC Builder and by Qsys
`rd`	Input	Receive data—a unidirectional data receiver. It is connected to the `td` bus of the transmitting device.	yes
`td`	Output	Transmit data—a unidirectional data driver. The `td` bus of one device is connected to the `rd` bus of the receiving device.	yes

Table 5–2. External Transceiver Interface (1)

Signal	Direction	Description	Exported by SOPC Builder
`td`	Output	Transmit data. 8-bit (1×) or 32-bit (4×) parallel data interface.	yes
`tc`	Output	Transmit control. 1 bit for 1×; 4 bits for 4×.	yes
`tclk`	Output	Transmit DDR center aligned clock.	yes
`phy_dis`	Output	External transmitter disable.	yes
`rd`	Input	Receive data. 8-bit (1×) or 32-bit (4×) parallel data interface.	yes
`rc`	Input	Receive control. 1 bit for 1×; 4 bits for 4×.	yes

Table 5–2. External Transceiver Interface (1)

Signal	Direction	Description	Exported by SOPC Builder
rclk	Input	Recovered DDR center aligned clock. 1 bit for 1×; 4 bits for 4×.	yes
rerr	Input	This input signal is used by external logic to indicate 8B10B decoding errors.	yes

Note to Table 5–2:

(1) Qsys does not support the RapidIO external transceiver interface.

Table 5–3. Avalon System Clock (1), (2)

Design Flow	Signal	Direction	Description	Exported by SOPC Builder and by Qsys
MegaWizard Plug-In Manager	sysclk	Input	Avalon system clock	—
Qsys or SOPC Builder	clock	Input	Avalon system clock	no

Note to Table 5–3:

(1) You connect this clock inside Qsys or SOPC Builder. If you connect it to an external clock, a port with the name of that external clock is added to the your Qsys or SOPC Builder system and this clock is connected to it. Qsys allows you to specify the external name for the clock signal.

(2) You must ensure that you drive this clock from a clock source that is running reliably when the RapidIO IP core comes out of reset.

Table 5–4. Reference Clock

Design Flow	Signal	Direction	Description	Exported by SOPC Builder and by Qsys
MegaWizard Plug-In Manager or Qsys	clk	Input	Physical layer reference clock	no (1)
SOPC Builder	clk_*<variation name>*	Input	Physical layer reference clock	yes

Note to Table 5–4:

(1) You connect this clock inside Qsys or export it. Qsys allows you to specify the external name for the clk signal.

Table 5–5. Global Signals

Signal	Direction	Description	Exported by SOPC Builder and by Qsys
reset_n	Input	Active-low system reset. In variations that implement only the Physical layer, this reset signal is associated with the reference clock. In variations with a Transport layer this reset is associated with the Avalon system clock. reset_n can be asserted asynchronously, but must stay asserted at least one clock cycle and must be de-asserted synchronously with the clock with which it is associated. Refer to Figure 4–4 on page 4–11 for a circuit that shows how to enforce synchronous deassertion of reset_n. Altera recommends that you apply an explicit 1 to 0 transition on the reset_n input port in simulation, to ensure that the simulation model is properly reset. In the Qsys flow, this signal is named clock_reset by default. In Arria V, Cyclone V, and Stratix V devices, the reset_n signal must be asserted synchronously with the embedded PHY IP core phy_mgmt_clk_reset signal described in Table 5–12 on page 5–8. Refer to Figure 4–5 on page 4–11 for a circuit that shows how to enforce all of the reset clocking requirements in Arria V, Cyclone V, and Stratix V devices. In addition, reset_n should not be deasserted when the Altera Transceiver Reconfiguration Controller reconfig_busy signal is high.	(1)
rxclk (2)	Output	Receive-side recovered clock. This signal is derived from the rxgxbclk clock—a clock driven by the transceiver—by division by 1 or 2, depending on the configuration of the IP core. For the frequency of this clock, refer to Table 4–2 on page 4–6	yes
txclk (3)	Output	The internal clock of the Physical layer. This signal is derived from the txgxbclk clock—a clock driven by the transceiver—by division by 1 or 2, depending on the configuration of the IP core. For the frequency of this clock, refer to Table 4–2 on page 4–6. This clock runs reliably only after the transceiver transmitter PLL is locked to the reference clock, which you can detect by monitoring the gxbpll_locked signal (refer to Table 5–12 on page 5–8). If you use this clock to drive the Avalon system clock, you must ensure you do not deassert reset_n before gxbpll_locked is asserted.	yes

Notes to Table 5–5:

(1) SOPC Builder exports this signal. In Qsys, you connect this reset signal inside the Qsys system or export it. Qsys allows you to specify the external name for any exported signal.

(2) In MegaCore variations generated using SOPC Builder, this signal is rxclk_<*variation name*>. In the Qsys design flow, this signal is exported as <*user_selected_prefix_for_exported_connections*>_rxclk.

(3) In MegaCore variations generated using SOPC Builder, this signal is txclk_<*variation name*>. In the Qsys design flow, this signal is exported as <*user_selected_prefix_for_exported_connections*>_txclk.

Table 5–6. Avalon-MM Slave Interface (1), (2) (Part 1 of 2)

Signal	Direction	Description	Exported by SOPC Builder and by Qsys
phy_mnt_s_clk	Input	Clock	—
phy_mnt_s_chipselect	Input	Slave chip select	—
phy_mnt_s_waitrequest	Output	Wait request	—
phy_mnt_s_read	Input	Read enable	—

Table 5–6. Avalon-MM Slave Interface (1), (2) (Part 2 of 2)

Signal	Direction	Description	Exported by SOPC Builder and by Qsys
phy_mnt_s_write	Input	Write enable	—
phy_mnt_s_address[16:0]	Input	Address bus	—
phy_mnt_s_writedata[31:0]	Input	Write data bus	—
phy_mnt_s_readdata[31:0]	Output	Read data bus	—

Notes to Table 5–6:

(1) All signals are in the phy_mnt_s_clk domain.

(2) This interface is not present in variations that implement the Transport layer. In those variations, the system maintenance Avalon-MM slave interface is used to access the Physical layer registers.

Atlantic Interface Signals

Table 5–7 and Table 5–8 list signals for the Atlantic receive and transmit interfaces. All Atlantic interface receive signals are in the arxclk clock domain, and all Atlantic interface transmit signals are in the atxclk clock domain. In Physical-layer-only variations of the RapidIO IP core, these two clocks are user-visible input clocks to the IP core. In variations with a Transport layer, these two clocks are connected to the Avalon system clock.

For information about the Atlantic interface signals and protocol, refer to the *FS13: Atlantic Interface* specification.

Table 5–7. Atlantic Receive Interface (1), (2) (Part 1 of 2)

Signal	Direction	Description	Exported by SOPC Builder and by Qsys
arxclk	Input	Atlantic receive interface clock.	—
arxreset_n	Input	Receive active-low reset. arxreset_n can be asserted asynchronously but should be deasserted synchronously to arxclk. This reset is connected internally to reset_n in variations that implement the Transport layer.	—
arxena	Input	Receive enable.	—
arxdav	Output	Receive data available. The arxdav signal is asserted when at least one complete packet is available to be read from the receive buffer. It is deasserted when the receive buffer does not have at least one complete packet available.	—
arxdat	Output	Receive data bus.	—
arxval	Output	Receive data valid.	—
arxsop	Output	Receive start of packet.	—

Table 5–7. Atlantic Receive Interface (1), (2) (Part 2 of 2)

Signal	Direction	Description	Exported by SOPC Builder and by Qsys
arxeop	Output	Receive end of packet.	—
arxmty	Output	Number of empty bytes on arxdat. During the final word of a packet, when arxeop is asserted, the arxmty signal indicates the number of bytes on the arxdat data bus that do not contain relevant data and should therefore be ignored. Relevant data may include packet data, CRC bytes, and null padding bytes received on the RapidIO link. Padding bytes ensure the width of the resulting RapidIO data payload is a multiple of 32 bits. In ×1 variations, the Atlantic interface width is 32 bits. Therefore, in ×1 variations, the value of arxmty is always 0. In ×4 variations, the Atlantic interface width is 64 bits, and therefore arxmty can have the values of 4 and 0.	—
arxerr	Output	Receive data error.	—
arxwlevel (3)	Output	Receive buffer write level (number of free 64-byte blocks in the receive buffer).	yes

Notes to Table 5–7:

(1) All of these signals are in the arxclk clock domain.

(2) This interface is not present in variations that include a Transport layer.

(3) The formula $\log_2$(size of the receive buffer in bytes/64)+1 determines the number of bits. For example, a receive buffer size of 16 KBytes would give: $\log_2$(16×1024/64)+1= 9 bits (for example, [8:0]).

Table 5–8. Atlantic Transmit Interface (1), (2) (Part 1 of 2)

Signal	Direction	Description	Exported by SOPC Builder and by Qsys
atxclk	Input	Atlantic transmit interface clock. This clock is connected internally to the Avalon system clock in variations that implement the Transport layer.	—
atxreset_n	Input	Transmit active-low reset. atxreset_n can be asserted asynchronously but should be deasserted on the rising edge of atxclk. This reset is connected internally to reset_n in variations that implement the Transport layer.	—
atxena	Input	Transmit enable.	—
atxdav	Output	Transmit data available. atxdav is asserted when the transmit buffer has space to accept at least one maximum size packet (for example, 276 bytes). It is deasserted when it does not have space to accept at least one maximum size packet.	—
atxdat	Input	Transmit data bus.	—
atxsop	Input	Transmit start of packet.	—
atxeop	Input	Transmit end of packet.	—
atxmty	Input	Number of empty bytes on atxdat. During the final word of a packet, when atxeop is asserted, the atxmty signal indicates the number of bytes on the atxdat data bus that do not contain relevant data and are therefore ignored.	—
atxerr	Input	Transmit data error.	—

Table 5–8. Atlantic Transmit Interface (1), (2) (Part 2 of 2)

Signal	Direction	Description	Exported by SOPC Builder and by Qsys
atxwlevel (3)	Output	Transmit buffer write level (number of free 64-byte blocks in the transmit buffer).	yes
atxovf	Output	Transmit buffer overflow. If a new packet is started by asserting atxena and atxsop three or more atxclk clock cycles after atxdav is deasserted, atxovf is asserted and the packet is ignored.	yes

Notes to Table 5–8:

(1) All of these signals are in the atxclk clock domain.

(2) This interface is not present in variations that include a Transport layer.

(3) The formula $\log_2$(size of the transmit buffer in bytes/64) determines the number of bits. For example, a transmit buffer size of 16 KBytes would give: $\log_2$(16×1024/64)= 8 bits (for example, [7:0]).

Status Packet and Error Monitoring Signals

Table 5–9 lists the status packet and error monitoring signals.

Table 5–9. Status Packet and Error Monitoring

Output Signal	Clock Domain	Description	Exported by SOPC Builder and by Qsys
packet_transmitted	txclk	Pulsed high for one clock cycle when a packet's transmission completes normally.	yes
packet_cancelled	txclk	Pulsed high for one clock cycle when a packet's transmission is cancelled by sending a stomp, a restart-from-retry, or a link-request control symbol.	yes
packet_accepted	rxclk	Pulsed high for one clock cycle when a packet-accepted control symbol is being transmitted.	yes
packet_retry	rxclk	Pulsed high for one clock cycle when a packet-retry control symbol is being transmitted.	yes
packet_not_accepted	rxclk	Pulsed high for one clock cycle when a packet-not-accepted control symbol is being transmitted.	yes
packet_crc_error	rxclk	Pulsed high for one clock cycle when a CRC error is detected in a received packet.	yes
symbol_error	rxclk	Pulsed high for one clock cycle when a corrupted symbol is received.	yes
port_initialized	txclk	This signal indicates that the serial RapidIO initialization sequence has completed successfully. This is a level signal asserted high while the initialization state machine is in the *1X_MODE* or *4X_MODE* state, as described in paragraph 4.6 of *Part VI of the RapidIO Specification*.	yes
port_error	txclk	This signal holds the value of the PORT_ERR bit of the Port 0 Error and Status CSR (offset 0x158) described in Table 6–10 on page 6–7.	yes
char_err	rxclk	Pulsed for one clock cycle when an invalid character or a valid but illegal character is detected.	yes

Multicast Event Signals

Table 5–10 lists the multicast event signals.

Table 5–10. Multicast Event Signals

Signal	Direction	Clock Domain	Description	Exported by SOPC Builder and by Qsys
multicast_event_tx	Input	txclk	Change the value of this signal to indicate the RapidIO IP core should transmit a multicast-event control symbol. This signal should remain stable for at least 10 txclk cycles.	yes
multicast_event_rx	Output	rxclk	Changes value when a multicast-event control symbol is received.	yes

Receive Priority Retry Threshold-Related Signals

Table 5–11 lists signals that are related to the **Receive Priority Retry Threshold** set in the RapidIO parameter editor.

Table 5–11. Priority Retry Threshold Signals (1)

Signal	Direction	Description	Exported by SOPC Builder and by Qsys
buf_av0	Output	Buffers available for priority 0 retry packets.	yes
buf_av1	Output	Buffers available for priority 1 retry packets.	yes
buf_av2	Output	Buffers available for priority 2 retry packets.	yes
buf_av3	Output	Buffers available for priority 3 retry packets.	yes

Note to Table 5–11:

(1) All of these signals are in the arxclk domain.

Transceiver Signals

Table 5–12 lists the transceiver signals in use for Arria GX, Arria II GX, Arria II GZ, Arria V, Cyclone IV GX, Cyclone V, Stratix II GX, Stratix IV GX, and Stratix V designs. These signals are connected directly to the transceiver block. In some cases these signals must be shared by multiple transceiver blocks that are implemented in the same device

Arria GX devices do not support dynamic reconfiguration, so the Quartus II software ties off the dynamic reconfiguration signals.

Table 5–12. Transceiver Signals (Part 1 of 3)

Signal	Direction	Description	Exported by SOPC Builder and by Qsys
cal_blk_clk (1)	Input	The Arria GX, Arria II GX, Arria II GZ, Cyclone IV GX, Stratix II GX, and Stratix IV GX transceiver's on-chip termination resistors are calibrated by a single calibration block. This circuitry requires a calibration clock. The frequency range of the cal_blk_clk is 10–125 MHz. For more information, refer to the *Arria GX Transceiver Architecture* chapter in volume 2 of the *Arria GX Device Handbook*, the *Transceiver Architecture for Arria II Devices* chapter in volume 2 of the *Arria II Device Handbook*, the *Cyclone IV Transceivers Architecture* chapter in volume 2 of the *Cyclone IV Device Handbook*, the *Stratix II GX Transceiver Architecture Overview* chapter in volume 2 of the *Stratix II GX Device Handbook*, or the *Stratix IV Transceiver Architecture* chapter in volume 2 of the *Stratix IV Device Handbook*. This signal is not present in Arria V, Cyclone V, or Stratix V variations.	no
phy_mgmt_clk (1)	Input	Clocks the Custom PHY IP core software interface. The expected maximum frequency of this clock is 250 MHz. This signal is present only in Arria V, Cyclone V, and Stratix V variations.	no
phy_mgmt_clk_reset	Input	Resets the Custom PHY IP core. This signal is present only in Arria V, Cyclone V, and Stratix V variations. phy_mgmt_clk_reset can be asserted asynchronously, but must stay asserted at least one clock cycle and must be de-asserted synchronously with phy_mgmt_clk. In addition, this signal must be driven by the same source as reset_n, to ensure that the two signals are asserted—but not deasserted—together. Refer to Figure 4–5 on page 4–11 for a circuit that shows how to enforce the synchronous assertion with reset_n and the minimal removal time and synchronous deassertion with phy_mgmt_clk. In addition, phy_mgmt_clk_reset should not be deasserted when the Altera Transceiver Reconfiguration Controller reconfig_busy signal is high.	no
rxgxbclk	Input	Transceiver receiver clock (recovered clock).	no
reconfig_clk (2)	Input	Reference clock for the dynamic reconfiguration controller. The frequency range for this clock is 2.5–50 MHz. If you use a dynamic reconfiguration block in your design to dynamically control the transceiver, then this clock is required by the dynamic reconfiguration block and the RapidIO IP core. If no external dynamic reconfiguration block is used, this input should be tied low. This signal is not present in Arria V, Cyclone V, or Stratix V variations.	yes

Table 5–12. Transceiver Signals (Part 2 of 3)

Signal	Direction	Description	Exported by SOPC Builder and by Qsys
reconfig_togxb [2]	Input	Driven from an external dynamic reconfiguration block. Supports the selection of multiple transceiver channels for dynamic reconfiguration. If no external dynamic reconfiguration block is used, then you must tie this bus to 3'b010 for Stratix II GX devices. Note that not using a dynamic reconfiguration block that enables offset cancellation results in a non-functional hardware design in Arria II GX, Arria II GZ, Arria V, Cyclone IV GX, Cyclone V, Stratix IV GX, and Stratix V devices. In Arria V, Cyclone V, and Stratix V devices, the width of this bus is (C + 1) × 70, where C is the number of channels, 1 or 4. This width supports communication from an Altera Reconfiguration Controller with C + 1 reconfiguration interfaces—one dedicated to each channel and another for the transceiver PLL—to the transceiver. If you omit the Altera Reconfiguration Controller from your simulation model, you must ensure all bits of this bus are tied to 0. For more information about the Altera Reconfiguration Controller component, refer to the *Altera Transceiver PHY IP Core User Guide*.	yes
reconfig_fromgxb [2]	Output	Driven to an external dynamic reconfiguration block. The bus identifies the transceiver channel whose settings are being transmitted to the dynamic reconfiguration block. If no external dynamic reconfiguration block is used, then this output bus can be left unconnected. In Arria V, Cyclone V, and Stratix V devices, the width of this bus is (C + 1) × 46, where C is the number of channels, 1 or 4. This width supports communication from the transceiver to C + 1 reconfiguration interfaces in an Altera Reconfiguration Controller, one interface dedicated to each channel and an additional interface for the transceiver PLL. For more information about the Altera Reconfiguration Controller component, refer to the *Altera Transceiver PHY IP Core User Guide*.	yes
gxbpll_locked	Output	Indicates the transceiver transmitter PLL is locked to the reference clock.	yes
gxb_powerdown	Input	Transceiver block reset and power down. This resets and powers down all circuits in the transceiver block. This signal does not affect the refclk buffers and reference clock lines. All the gxb_powerdown input signals of IP cores intended to be placed in the same quad should be tied together. The gxb_powerdown should be tied low or should remain asserted for at least 2 ms whenever it is asserted. This signal is not present in Arria V, Cyclone V, or Stratix V variations.	yes

Table 5–12. Transceiver Signals (Part 3 of 3)

Signal	Direction	Description	Exported by SOPC Builder and by Qsys
rx_errdetect	Output	Transceiver 8B10B code group violation signal bus. For details, refer to the relevant device handbook.	yes

Notes to Table 5–12:

(1) You connect this clock inside the Qsys tool or SOPC Builder. If you connect it to an external clock, a port with the name of that external clock is added to your Qsys or SOPC Builder system and this clock is connected to it.

(2) Refer to "Instantiating Multiple RapidIO IP Cores" on page 2–14 for information about how to successfully combine multiple high-speed transceiver channels—whether in two RapidIO IP core instances or in a RapidIO IP core and in another component—in the same transceiver block.

In addition to customization of the transceiver through the parameter editor (in variations that target a device for which the transceivers are configured with the ALTGX megafunction, and not with the Transceiver PHY IP core), you can use the transceiver reconfiguration block to dynamically modify the parameter interface. The dynamic reconfiguration block lets you reconfigure the following PMA settings:

- Pre-emphasis
- Equalization
- Offset cancellation
- V_{OD} on a per channel basis

The dynamic reconfiguration block is required for many device families, including Arria V, Cyclone V, and Stratix V devices. Refer to Chapter 2, Getting Started.

For more information, refer to "Device Options" on page 3–1 and the appropriate device handbook. For more information about offset cancellation, refer to the relevant device handbook.

Register-Related Signals

Table 5–13 lists the register-related signals.

Table 5–13. Register-Related Signals

Signal	Direction	Clock Domain	Description	Exported by SOPC Builder and by Qsys
ef_ptr[15:0]	Input	txclk	Most significant bits [31:16] of the PHEAD0 register.	yes
master_enable	Output	txclk	This output reflects the value of the Master Enable bit of the Port General Control CSR, which indicates whether this device is allowed to issue request packets. If the Master Enable bit is not set, the device may only respond to requests. User logic connected to the Avalon-ST pass-through interface should honor this value and not cause the Physical layer to issue request packets when it is not allowed.	yes
port_response_timeout [23:0]	Output	txclk	Most significant bits [31:8] of PRTCTRL register. User logic connected to the pass-through interface that results in request packets requiring a response can use this value to check for request to response time-out. This signal is present in Physical-layer-only variations and in variations that include the Avalon-ST pass-through interface.	yes

Transport and Logical Layer Signals

Table 5–14 through Table 5–25 list the signals used by the Transport layer and the Maintenance, Input/Output, and Doorbell Logical layer modules of the RapidIO IP core. For a list of descriptions of the pins and signals used and generated by the Physical layer, see "Physical Layer Signals" on page 5–1.

Clock and Reset Signals

Table 5–3 through Table 5–5 list the clock and reset signals used when the Transport layer and all Logical layer modules exist.

Avalon-MM Interface Signals

Table 5–14 through Table 5–21 list the standard signals for the Avalon-MM interfaces. Signals on Avalon-MM interfaces are in the Avalon system clock domain.

When you instantiate the IP core with the Qsys tool or SOPC Builder, these signals are automatically connected and are not visible as inputs or outputs of the system.

Refer to the *Avalon Interface Specifications* for details.

Table 5–14. System Maintenance Avalon-MM Slave Interface Signals

Signal	Direction	Description
sys_mnt_s_clk	Input	This signal is not used, therefore it can be left open. The Avalon clock is used internally to sample this interface.
sys_mnt_s_chipselect	Input	System maintenance slave chip select
sys_mnt_s_waitrequest	Output	System maintenance slave wait request
sys_mnt_s_read	Input	System maintenance slave read enable
sys_mnt_s_write	Input	System maintenance slave write enable
sys_mnt_s_address[16:0]	Input	System maintenance slave address bus
sys_mnt_s_writedata[31:0]	Input	System maintenance slave write data bus
sys_mnt_s_readdata[31:0]	Output	System maintenance slave read data bus
sys_mnt_s_irq	Output	System maintenance slave interrupt request

Table 5–15. Maintenance Avalon-MM Master Interface Signals

Signal	Direction	Description
mnt_m_clk	Input	This signal is not used, therefore it can be left open. The Avalon clock is used internally to sample this interface.
mnt_m_waitrequest	Input	Maintenance master wait request
mnt_m_read	Output	Maintenance master read enable
mnt_m_write	Output	Maintenance master write enable
mnt_m_address[31:0]	Output	Maintenance master address bus
mnt_m_writedata[31:0]	Output	Maintenance master write data bus
mnt_m_readdata[31:0]	Input	Maintenance master read data bus
mnt_m_readdatavalid	Input	Maintenance master read data valid

Table 5–16. Maintenance Avalon-MM Slave Interface Signals

Signal	Direction	Description
mnt_s_clk	Input	This signal is not used, therefore it can be left open. The Avalon clock is used internally as the clock reference for this interface.
mnt_s_chipselect	Input	Maintenance slave chip select.
mnt_s_waitrequest	Output	Maintenance slave wait request.
mnt_s_read	Input	Maintenance slave read enable.
mnt_s_write	Input	Maintenance slave write enable.
mnt_s_address[25:0]	Input	Maintenance slave address bus.
mnt_s_writedata[31:0]	Input	Maintenance slave write data bus.
mnt_s_readdata[31:0]	Output	Maintenance slave read data bus.
mnt_s_readdatavalid	Output	Maintenance slave read data valid.
mnt_s_readerror	Output	Maintenance slave read error, which indicates that the read transfer did not complete successfully. This signal is valid only when the mnt_s_readdatavalid signal is asserted.

The following parameters are used in some signal width definitions:

- n = (internal datapath width - 1)
- m = (internal datapath width/8) - 1
- k = 6 for 32-bit internal datapath width, and 5 for 64-bit internal datapath width
- j = (I/O slave address width - 1) — the **I/O slave address width** value is defined in the RapidIO parameter editor.

For signals and bus widths specific to your variation, refer to the HTML report file generated by the parameter editor.

Table 5–17. Input/Output Master Datapath Write Avalon-MM Interface Signals

Signal	Direction	Description
io_m_wr_clk	Input	This signal is not used, therefore it can be left open. The Avalon clock is used internally as the clock reference for this interface.
io_m_wr_waitrequest	Input	Input/Output master wait request
io_m_wr_write	Output	Input/Output master write enable
io_m_wr_address[31:0]	Output	Input/Output master address bus
io_m_wr_writedata[n:0]	Output	Input/Output master write data bus
io_m_wr_byteenable[m:0]	Output	Input/Output master byte enable
io_m_wr_burstcount[k:0]	Output	Input/Output master burst count

Table 5–18. Input/Output Master Datapath Read Avalon-MM Interface Signals

Signal	Direction	Description
io_m_rd_clk	Input	This signal is not used, therefore it can be left open. The Avalon clock is used internally as the clock reference for this interface.
io_m_rd_waitrequest	Input	Input/Output master wait request
io_m_rd_read	Output	Input/Output master read enable
io_m_rd_address[31:0]	Output	Input/Output master address bus
io_m_rd_readdata[n:0]	Input	Input/Output master read data bus
io_m_rd_readdatavalid	Input	Input/Output master read data valid
io_m_rd_burstcount[k:0]	Output	Input/Output master burst count
io_m_rd_readerror	Input	Input/Output master indicates that the burst read transfer did not complete successfully. This signal should be asserted through the final cycle of the read transfer.

Table 5–19. Input/Output Slave Datapath Write Avalon-MM Interface Signals (Part 1 of 2)

Signal	Direction	Description
io_s_wr_clk	Input	This signal is not used, therefore it can be left open. The Avalon clock is used internally as the clock reference for this interface.
io_s_wr_chipselect	Input	Input/Output slave chip select
io_s_wr_waitrequest	Output	Input/Output slave wait request
io_s_wr_write	Input	Input/Output slave write enable

Table 5–19. Input/Output Slave Datapath Write Avalon-MM Interface Signals (Part 2 of 2)

Signal	Direction	Description
io_s_wr_address[j:0]	Input	Input/Output slave address bus
io_s_wr_writedata[n:0]	Input	Input/Output slave write data bus
io_s_wr_byteenable[m:0]	Input	Input/Output slave byte enable
io_s_wr_burstcount[k:0]	Input	Input/Output slave burst count

Table 5–20. Input/Output Slave Datapath Read Avalon-MM Interface Signals

Signal	Direction	Description
io_s_rd_clk	Input	This signal is not used, therefore it can be left open. The Avalon clock is used internally as the clock reference for this interface.
io_s_rd_chipselect	Input	Input/Output slave chip select
io_s_rd_waitrequest	Output	Input/Output slave wait request
io_s_rd_read	Input	Input/Output slave read enable
io_s_rd_address[j:0]	Input	Input/Output slave address bus
io_s_rd_readdata[n:0]	Output	Input/Output slave read data bus
io_s_rd_readdatavalid	Output	Input/Output slave read data valid
io_s_rd_burstcount[k:0]	Input	Input/Output slave burst count
io_s_rd_readerror	Output	Input/Output slave read error indicates that the burst read transfer did not complete successfully. This signal is valid only when the io_s_rd_readdatavalid signal is asserted.

Table 5–21. Doorbell Message Avalon-MM Slave Interface Signals

Signal	Direction	Description
drbell_s_clk	Input	This signal is not used, therefore it can be left open. The Avalon clock is used internally as the clock reference for this interface.
drbell_s_chipselect	Input	Doorbell chip select
drbell_s_write	Input	Doorbell write enable
drbell_s_read	Input	Doorbell read enable
drbell_s_address[5:0]	Input	Doorbell address bus
drbell_s_writedata[31:0]	Input	Doorbell write data bus
drbell_s_readdata[31:0]	Output	Doorbell read data bus
drbell_s_waitrequest	Output	Doorbell wait request
drbell_s_irq	Output	Doorbell interrupt

Avalon-ST Pass-Through Interface Signals

Table 5–22 through Table 5–24 list the standard Avalon-ST pass-through interface signals.

When you instantiate the IP core with Qsys or SOPC Builder, these signals are automatically connected and are not visible as inputs or outputs of the system.

Table 5–22 describes the Avalon-ST pass-through interface transmission (Tx) signals.

Table 5–22. Avalon-ST Pass-Through Interface Transmission Signals

Signal	Type	Function
gen_tx_ready	Output	Indicates that the IP core is ready to receive data on the next clock cycle. Asserted by the Avalon-ST sink to mark *ready cycles*, which are the cycles in which transfers can take place. If ready is asserted on cycle N, the cycle (N+READY_LATENCY) is a ready cycle. In the RapidIO IP core, READY_LATENCY is equal to 1, so the cycle immediately following the rising clock edge on which gen_tx_ready is detected as asserted is the ready cycle. This signal may alternate between 0 and 1 when the Avalon-ST pass-through transmitter interface is idle. After gen_tx_valid is asserted, gen_tx_ready remains asserted for the duration of the packet transmission, unless the Physical layer transmit buffer fills.
gen_tx_valid	Input	Used to qualify all the other transmit side of the Avalon-ST pass-through interface input signals. On every ready cycle in which gen_tx_valid is high, data is sampled by the IP core. (1)
gen_tx_startofpacket	Input	Marks the active cycle containing the start of the packet. (1)
gen_tx_endofpacket	Input	Marks the active cycle containing the end of the packet. (1)
gen_tx_data	Input	A 32-bit or 64-bit wide data bus for 1x or 4x variations respectively. Carries the bulk of the information transferred from the source to the sink. (1)
gen_tx_empty	Input	This bus identifies the number of empty bytes on the last data transfer of the gen_tx_endofpacket. For a 32-bit wide data bus, this bus is 2 bits wide. For a 64-bit wide data bus, this bus is 3 bits wide. The least significant bit is ignored and assumed to be 0. The following values are supported: (1) 32-bit bus: 64-bit bus: 2'b0X none 3'b00X none 2'b1X [15:0] 3'b01X [15:0] 3'b10X [31:0] 3'b11X [47:0]
gen_tx_error	Input	If asserted any time during the packet transfer, this signal indicates the corresponding data has an error and causes the packet to be dropped by the IP core. A value of zero on any beat indicates the data on that beat is error-free. (1)

Note to Table 5–22:

(1) gen_tx_valid is used to qualify all the other input signals of the transmit side of the Avalon-ST pass-through interface.

Table 5–23 describes the Avalon-ST pass-through receiver (Rx) signals.

For more information about these signals, refer to the *Avalon Interface Specifications*.

Table 5–23. Avalon-ST Pass-Through Interface Receiver Signals

Signal	Type	Function
gen_rx_ready	Input	Indicates to the IP core that the user's custom logic is ready to receive data on the next clock cycle. Asserted by the sink to mark ready cycles, which are cycles in which transfers can occur. If ready is asserted on cycle N, the cycle (N+READY_LATENCY) is a ready cycle. The RapidIO IP core is designed for READY_LATENCY equal to1.
gen_rx_valid	Output	Used to qualify all the other output signals of the receive side pass-through interface. On every rising edge of the clock where gen_rx_valid is high, gen_rx_data can be sampled. (1)
gen_rx_startofpacket	Output	Marks the active cycle containing the start of the packet. (1)
gen_rx_endofpacket	Output	Marks the active cycle containing the end of the packet. (1)
gen_rx_data	Output	A 32-bit or 64-bit wide data bus for 1x or 4x mode respectively. (1)
gen_rx_empty	Output	This bus identifies the number of empty bytes on the last data transfer of the gen_rx_endofpacket. For a 32-bit wide data bus, this bus is 4 bits wide. For a 64-bit wide data bus, this bus is 8 bits wide. The least significant bit is ignored and assumed to be 0. The following values are supported: (1) 32-bit bus: 64-bit bus: 2'b0X none 3'b00X none 2'b1X [15:0] 3'b01X [15:0] 3'b10X [31:0] 3'b11X [47:0]
gen_rx_size (2)	Output	Identifies the number of cycles the current packet transfer requires. This signal is only valid on the start of packet cycle when gen_rx_startofpacket is asserted. (1)
gen_rx_error	Output	Indicates that the corresponding data has an error. This signal is never asserted by the RapidIO IP core. (1)

Notes to Table 5–23:

(1) gen_rx_valid is used to qualify all the other output signals of the receive side Avalon-ST pass-through interface.

(2) This is not an Avalon-ST signal. The gen_rx_size signal is exported when the RapidIO IP core is part of a Qsys or SOPC Builder system.

Error Management Extension Signals

Table 5–24 shows the signals that are added when the Avalon-ST pass-through interface is enabled and at least one of the **Data Messages** options (**Source Operation** or **Destination Operation)** is turned on in the RapidIO parameter editor.

Table 5–24. Message Passing Error Management Input Ports (1), (2) (Part 1 of 2)

Signal	Description
Message Passing Error Management Inputs	
error_detect_message_error_response	Sets the MESSAGE ERROR RESPONSE bit in the Logical/Transport Layer Error Detect CSR and triggers capture into the Error Management registers of the captured fields below.
error_detect_message_format_error	Sets the MESSAGE ERROR RESPONSE bit in the Logical/Transport Layer Error Detect CSR and triggers capture into the Error Management registers of the captured fields below.
error_detect_message_request_timeout	Sets the MESSAGE REQUEST TIME-OUT bit in the Logical/Transport Layer Error Detect CSR and triggers capture into the Error Management registers of the captured fields below.
error_capture_letter [1:0]	Field captured into the Logical/Transport Layer Control Capture CSR.
error_capture_mbox [1:0]	Field captured into the Logical/Transport Layer Control Capture CSR.
error_capture_msgseg_or_xmbox [3:0]	Field captured into the Logical/Transport Layer Control Capture CSR.
Common Error Management Inputs	
error_detect_illegal_transaction_decode	Sets the ILLEGAL TRANSACTION DECODE bit in the Logical/Transport Layer Error Detect CSR and triggers capture into the Error Management registers of the captured fields below.
error_detect_illegal_transaction_target	Sets the ILLEGAL TRANSACTION TARGET ERROR bit in the Logical/Transport Layer Error Detect CSR and triggers capture into the Error Management registers of the captured fields below.
error_detect_packet_response_timeout	Sets the PACKET RESPONSE TIME-OUT bit in the Logical/Transport Layer Error Detect CSR and triggers capture into the Error Management registers of the captured fields below.
error_detect_unsolicited_response	Sets the UNSOLICITED RESPONSE bit in the Logical/Transport Layer Error Detect CSR and triggers capture into the Error Management registers of the captured fields below.
error_detect_unsupported_transaction	Sets the UNSUPPORTED TRANSACTION bit in the Logical/Transport Layer Error Detect CSR and triggers capture into the Error Management registers of the captured fields below.

Table 5–24. Message Passing Error Management Input Ports (1), (2) (Part 2 of 2)

Signal	Description
error_capture_ftype [3:0]	Field captured into Logical/Transport Layer Control Capture CSR.
error_capture_ttype [3:0]	Field captured into Logical/Transport Layer Control Capture CSR.
error_capture_destination_id [15:0]	Field captured into Logical/Transport Layer Device ID Capture CSR.
error_capture_source_id [15:0]	Field captured into Logical/Transport Layer Device ID Capture CSR.

Notes to Table 5–24:

(1) All of these signals are exported by SOPC Builder and included in the **rio_data_messages** conduit bundle in Qsys. This conduit bundle is enabled only in RapidIO variations in which at least one of the **Data Messages Source Operation** or **Destination Operation** options is turned on.

(2) All these input signals are sampled in the Avalon system clock domain.

Packet and Error Monitoring Signal for the Transport Layer

Table 5–25 shows the packet and error monitoring signal for the Transport layer. For Physical layer packet and error monitoring signals, see Table 5–9 on page 5–6.

Table 5–25. Transport Layer Packet and Error Monitoring Signal

Signal	Clock Domain	Direction	Description	Exported by SOPC Builder and by Qsys
rx_packet_dropped	Avalon system clock	Output	Pulsed high one Avalon clock cycle when a received packet is dropped by the Transport layer. Received packets are only dropped if the Avalon-ST pass-through interface is not enabled in the variation. Examples of packets that are dropped include packets that have an incorrect destination ID, are of a type not supported by the selected Logical layers, or have a transaction ID outside the range used by the selected Logical layers.	yes

6. Software Interface

The RapidIO IP core supports the following sets of registers that control the RapidIO IP core or query its status:

- Standard RapidIO capability registers—CARs
- Standard RapidIO command and status registers—CSRs
- Extended features registers
- Implementation defined registers
- Doorbell specific registers

Some of these register sets are supported by specific RapidIO IP core layers only. This chapter organizes the registers by the layers they support. The Physical layer registers are described first, followed by the Transport and Logical layers registers.

All of the registers are 32 bits wide and are shown as hexadecimal values. The registers can be accessed only on a 32-bit (4-byte) basis. The addressing for the registers therefore increments by units of 4.

Reserved fields are labelled in the register tables. These fields are reserved for future use and your design should not write to or rely on a specific value being found in any reserved field or bit.

The following sets of registers are accessible through the System Maintenance Avalon-MM slave interface.

- CARs—Capability registers
- CSRs—Command and status registers
- Extended features registers
- Implementation defined registers

A remote device can access these registers only by issuing read/write `MAINTENANCE` operations destined for the local device. The local device must route these transactions, if they are addressing these registers, from the Maintenance master interface to the System Maintenance slave interface. Routing can be done by a Qsys or SOPC Builder system or by a user-provided design. Refer to "Maintenance Module" on page 4–28 for more details.

The `doorbell` registers can be accessed through the Doorbell Avalon-MM slave interface. These registers are implemented only if you turn on **Doorbell Tx enable** or **Doorbell Rx enable** in the RapidIO parameter editor. If you turn on only **Doorbell Rx enable**, only the Rx-related `doorbell` registers are implemented. If you turn on only **Doorbell Tx enable**, only the Tx-related `doorbell` registers are implemented.

Table 6–1 lists the access codes used to describe the type of register bits.

Table 6–1. Register Access Codes

Code	Description
RW	Read/write
RO	Read-only
RW1C	Read/write 1 to clear
RW0S	Read/write 0 to set
RTC	Read to clear
RTS	Read to set
RTCW	Read to clear/write
RTSW	Read to set/write
RWTC	Read/write any value to clear
RWTS	Read/write any value to set
RWSC	Read/write self-clearing
RWSS	Read/write self-setting
UR0	Unused bits/read as 0
UR1	Unused bits/read as 1

Table 6–2 lists the CAR, CSR and all the registers in the extended features implementation defined address spaces. The doorbell registers are listed in Table 6–57 on page 6–27.

Table 6–2. Memory Map (Part 1 of 3)

Address	Name	Used by
Capability Registers (CARs)		
0x0	Device Identity	These CARs are not used by any of the internal modules. They do not affect the functionality of the RapidIO IP core. These registers are all Read-Only. Their values are set using the RapidIO parameter editor when generating the IP core. These registers inform either a local processor or a processor on a remote end about the IP core's capabilities.
0x4	Device Information	
0x8	Assembly Identity	
0xC	Assembly Information	
0x10	Processing Element Features	
0x14	Switch Port Information	
0x18	Source Operations	
0x1C	Destination Operations	
Command and Status Registers (CSRs)		
0x4C	Processing Element Logical layer Control	Input/Output Slave Logical layer
0x58	Local Configuration Space Base Address 0	Input/Output Master Logical layer
0x5C	Local Configuration Space Base Address 1	Input/Output Master Logical layer
0x60	Base Device ID	Transport layer for routing or filtering. Input/Output Slave Logical layer

Table 6–2. Memory Map (Part 2 of 3)

Address	Name	Used by
0x68	Host Base Device ID Lock	Maintenance module
0x6C	Component Tag	Accessed via the Maintenance module
Extended Features Space		
0x100	Register Block Header	Physical layer
0x104–0x11C	Reserved	—
0x120	Port Link Time-out Control	Logical layer modules
0x124	Port Response Time-out Control	Logical layer modules
0x13C	Port General Control	Physical layer
0x148	Port 0 Local AckID	Physical layer
0x158	Port 0 Error and Status	Physical layer
0x15C	Port 0 Control	Physical layer
Implementation-Defined Space		
0x10000	Reserved	
0x10004		
0x10008		
0x1000C–0x1001C		
0x10020		
0x10024		
0x10028		
0x1002C-0x1007C		
0x10080	Maintenance Interrupt	Maintenance module
0x10084	Maintenance Interrupt Enable	Maintenance module
0x10088	Rx Maintenance Mapping	Maintenance module
0x1008C–0x100FC	Reserved	—
0x10100	Tx Maintenance Window 0 Base	Maintenance module
0x10104	Tx Maintenance Window 0 Mask	Maintenance module
0x10108	Tx Maintenance Window 0 Offset	Maintenance module
0x1010C	Tx Maintenance Window 0 Control	Maintenance module
0x10110–0x101FC	Tx Maintenance Windows 1-15	Maintenance module
0x10200	Tx Port Write Control	Maintenance module
0x10204	Tx Port Write Status	Maintenance module
0x10210–0x1024C	Tx Port Write Buffer	Maintenance module
0x10250	Rx Port Write Control	Maintenance module
0x10254	Rx Port Write Status	Maintenance module
0x10260–0x1029C	Rx Port Write Buffer	Maintenance module
0x102A0–0x102FC	Reserved	—
0x10300	I/O Master Window 0 Base	Input/Output Master Logical layer
0x10304	I/O Master Window 0 Mask	Input/Output Master Logical layer
0x10308	I/O Master Window 0 Offset	Input/Output Master Logical layer

Table 6–2. Memory Map (Part 3 of 3)

Address	Name	Used by
0x1030C	Reserved	—
0x10310–0x103FC	I/O Master Windows 1–15	Input/Output Master Logical layer
0x10400	I/O Slave Window 0 Base	Input/Output Slave Logical layer
0x10404	I/O Slave Window 0 Mask	Input/Output Slave Logical layer
0x10408	I/O Slave Window 0 Offset	Input/Output Slave Logical layer
0x1040C	I/O Slave Window 0 Control	Input/Output Slave Logical layer
0x10410-0x104FC	I/O Slave Windows 1–15	Input/Output Slave Logical layer
0x10500	I/O Slave Interrupt	Input/Output Slave Logical layer
0x10504	I/O Slave Interrupt Enable	Input/Output Slave Logical layer
0x10508	I/O Slave Pending NWRITE_R Transactions	Input/Output Slave Logical Layer
0x1050C	I/O Slave Avalon-MM Write Transactions	Input/Output Slave Logical layer and Doorbell module
0x10510	I/O Slave RapidIO Write Requests	Input/Output Slave Logical layer and Doorbell module
0x10514–0x105FC	Reserved	—
0x10600	Rx Transport Control	Transport layer
0x10604–0x107FC	Reserved	—
0x10800	Logical/Transport Layer Error Detect	Logical/Transport layer
0x10804	Logical/Transport Layer Error Enable	Logical/Transport layer
0x10808	Logical/Transport Layer Address	Logical/Transport layer
0x1080C	Logical/Transport Layer Device ID Capture	Logical/Transport layer
0x10810	Logical/Transport Layer Control Capture	Logical/Transport layer

Physical Layer Registers

Table 6–3 shows the memory map for the serial RapidIO Physical layer. Table 6–4 through Table 6–11 describe the registers for the Physical layer of the RapidIO IP core. The offset values are defined by the RapidIO standard.

Table 6–3. Physical Layer Register Map (Part 1 of 2)

Address	Name	Description
0x100	PHEAD0	1x/4x LP-Serial Register Block Header
0x104	PHEAD1	Reserved register
0x120	PLTCTRL	Port Link Time-out Control CSR
0x124	PRTCTRL	Port Response Time-out Control CSR
0x13C	PGCTRL	Port General Control CSR

Table 6–3. Physical Layer Register Map (Part 2 of 2)

Address	Name	Description
0x158	ERRSTAT	Port 0 Error and Status CSR
0x15C	PCTRL0	Port 0 Control CSR

Table 6–4. PHEAD0—1x/4x LP-Serial Register Block Header—0x100

Field	Bits	Access	Function	Default
EF_PTR	[31:16]	RO	Hard-wired pointer to the next block in the data structure, if one exists. The value is set from the ef_ptr input port.	ef_ptr
EF_ID	[15:0]	RO	Hard-wired extended features ID.	16'h0001

Table 6–5. PHEAD1—Reserved Register—0x104

Field	Bits	Access	Function	Default
RSRV	[31:0]	UR0	Reserved	32'h0

Table 6–6. PLTCTRL—Port Link Time-Out Control CSR—0x120

Field	Bits	Access	Function	Default
VALUE	[31:8]	RW	Time-out interval value for link-layer event pairs such as the time interval between sending a packet and receiving the corresponding acknowledge control symbol, or between sending a link-request and receiving the corresponding link-response. The duration of the link-response time-out is approximately equal to 4.5 seconds multiplied by the contents of this field, divided by (2^{24} - 1). Note: Avoid time-out values less than 0x000010 because they may not be reliable.	24'hFF_FFFF
RSRV	[7:0]	UR0	Reserved	8'h0

Table 6–7. PRTCTRL—Port Response Time-Out Control CSR—0x124

Field	Bits	Access	Function	Default
VALUE	[31:8]	RW	Time-out internal value. ■ Physical layer-only variations: This value is not used by the RapidIO IP core. The contents of this register drive the `port_response_timeout` output signal. ■ Variations using Logical layers: The duration of the port response time-out for all transactions that require a response—including `MAINTENANCE`, `DOORBELL`, `NWRITE_R`, and `NREAD` transactions—is approximately equal to 4.5 seconds multiplied by the contents of this field, divided by (2^{24} - 1). Note: Avoid time-out values less than 0x000010 because they may not be reliable. Note: A new value in this field might not propagate quickly enough to be applied to the next transaction. Any packet sent within 64 Avalon clock cycles of the value change in the register might be sent using the previous time-out value. Note: Avoid changing the value in this field when any packet is waiting to be transmitted or waiting for a response, to ensure that in each FIFO, the pending entries all have the same time-out value.	`24'hFF_FFFF`
RSRV	[7:0]	UR0	Reserved	`8'h0`

Table 6–8. Port General Control—Offset: 0x13C

Field	Bits	Access	Function	Default
`HOST`	[31]	RW	A host device is a device that is responsible for system exploration, initialization, and maintenance. Host devices typically initialize agent or slave devices. 'b0 - agent or slave device 'b1 - host device	`1'b0`
`ENA`	[30]	RW	The `Master Enable` bit controls whether or not a device is allowed to issue requests to the system. If `Master Enable` is not set, the device may only respond to requests. 'b0 - The processing element cannot issue requests 'b1 - The processing element can issue requests Variations that use only the Physical layer ignore this bit.	`1'b0`
`DISCOVER`	[29]	RW	This device has been located by the processing element responsible for system configuration. 'b0 - The device has not been previously discovered 'b1 - The device has been discovered by another processing element	`1'b0`
RSRV	[28:0]	RO	Reserved	`29'b0`

Table 6–9. Port 0 Local AckID CSR—Offset: 0x148

Field	Bits	Access	Function	Default
RSRV	[31:29]	RO	Reserved	3'b0
INBOUND_ACKID	[28:24]	RO	Next expected packet ackID.	5'b0
RSRV	[23:13]	RO	Reserved	11'b0
OUTSTANDING_ ACKID	[12:8]	RO	Next expected acknowledge control-symbol ackID.	5'b0
RSRV	[7:5]	RO	Reserved	3'b0
OUTBOUND_ACKID	[4:0]	RO	Next transmitted packet ackID.	5'b0

Table 6–10. Port 0 Error and Status CSR—Offset: 0x158 (1) (Part 1 of 3)

Field	Bits	Access	Function	Default
RSRV	[31:21]	RO	Reserved	11'b0
OUT_RTY_ENC	[20]	RW1C	Output port has encountered a retry condition. In all cases, this condition is caused by the port receiving a packet-retry control symbol. This bit is set if the OUT_RTY_STOP bit is set.	1'b0
OUT_RETRIED	[19]	RO	Output port has received a packet-retry control symbol and cannot make forward progress. This bit is cleared when a packet-accepted or packet-not-accepted control symbol is received.	1'b0
OUT_RTY_STOP	[18]	RO	Output port has been stopped due to a retry and is trying to recover. When a port receives a packet_retry control symbol, it enters the *Output Retry Stopped* state. In this state, the port transmits a restart-from-retry control symbol to its link partner. The link partner exits the *Input Retry Stopped* state and normal operation resumes. The port exits the *Output Retry Stopped* state.	1'b0
OUT_ERR_ENC	[17]	RW1C	Output port has encountered a transmission error and has possibly recovered from it. This bit is set when the OUT_ERR_STOP bit is set.	1'b0
OUT_ERR_STOP	[16]	RO	Output port has been stopped due to a transmission error and is trying to recover. The output port is in the *Output Error Stopped* state. The port enters into this state when it receives a packet-not-accepted control symbol. To exit from this state, the port issues an input-status link-request/input-status (restart-from-error) control symbol. The port waits for the link-response control symbol and exits the *Output Error Stopped* state.	1'b0
RSRV	[15:11]	RO	Reserved	5'b0
IN_RTY_STOP	[10]	RO	Input port is stopped due to a retry. When the receiver issues a packet-retry control symbol to its link partner, it enters the *Input Retry Stopped* state. The receiver issues a packet-retry when sufficient buffer space is not available to accept the packet for that specific priority. The receiver continues in the I*nput Retry Stopped* state until it receives a restart-from-retry control symbol.	1'b0
IN_ERR_ENC	[9]	RW1C	Input port has encountered a transmission error. This bit is set if the IN_ERR_STOP bit is set.	1'b0

Table 6–10. Port 0 Error and Status CSR—Offset: 0x158 (1) (Part 2 of 3)

Field	Bits	Access	Function	Default
IN_ERR_STOP	[8]	RO	Input port is stopped due to a transmission error. The port is in the *Input Error Stop* state. The following conditions cause the input port to transition to this state: ■ Cancellation of a packet by using the `restart-from-retry` control symbol. ■ Invalid character or valid character other than A, K, or R in an idle sequence. ■ Single bit transmission errors. ■ Any of the following link protocol violations: Unexpected packet accepted Unexpected packet-retry Unexpected packet-not-accepted packet Acknowledgment control symbol with an unexpected `packet_ackID` Link time-out while waiting for an acknowledgment control symbol ■ Corrupted control symbols, that is, CRC violations on the symbol. ■ Any of the following Packet Errors: Unexpected `ackID` value Incorrect CRC value Invalid characters or valid nondata characters Max data payload violations The recovery mechanism consists of these steps: 1. Issue a `packet-not-accepted` control symbol. 2. Wait for `link-request/input-status` control symbol. 3. Send `link-response` control symbol.	`1'b0`
RSRV	[7:5]	RO	Reserved	`3'h0`
PWRITE_PEND	[4]	RO	This register is not implemented and is reserved. It is always set to zero.	`1'b0`
RSRV	[3]	RO	Reserved	`1'b0`

Table 6–10. Port 0 Error and Status CSR—Offset: 0x158 (1) (Part 3 of 3)

Field	Bits	Access	Function	Default
PORT_ERR	[2]	RW1C	This bit is set if the input port error recovery state machine encounters an unrecoverable error or the output port error recovery state machine enters the *fatal_error* state. The input port error recovery state machine encounters an unrecoverable error if it times out while waiting for a link-request after sending a packet-not-accepted control symbol. The output port error recovery state machine enters the *fatal_error* state if the following sequence of events occurs: 1. The output port error recovery state machine enters the *stop_output* state when it receives a packet-not-accepted control symbol. In response, it sends the input-status link-request/input-status (restart-from-error) control symbol. 2. One of the following events occurs in response to the link-request control symbol: ■ If the link-response is received but the ackID is outside of the outstanding ackID set, or the port_status value is *Error*, then the output port error recovery state machine enters the *fatal_error* state. ■ If the port times out before receiving link-response, and the number of times this time-out event has occurred reaches the number you set in the RapidIO parameter editor as the value for **Link-request attempts**, then the output port error recovery state machine enters the *fatal_error* state. When the PORT_ERR bit is set, the RapidIO IP core performs an internal soft reset sequence, as described in "Fatal Errors" on page 4–65. The port_error output signal mirrors this register bit.	1'b0
PORT_OK	[1]	RO	Input and output ports are initialized and can communicate with the adjacent device. This bit is asserted when port_initialized is asserted and the following conditions exist: ■ The IP core has received at least 7 status control symbols. ■ The output port retry recovery state machine is not in the *stop_output* state. ■ The output port error recovery state machine is not in the *stop_output* state. ■ The input port retry recovery state machine is not in the *stop_input* state. ■ The input port error recovery state machine is not in the *stop_input* state.	1'b0
PORT_UNINIT	[0]	RO	Input and output ports are not initialized and are in training mode. This bit is the negation of the PORT_OK bit.	1'b1

Note to Table 6–10:

(1) Refer to "Error Detection and Management" on page 4–64 for details.

Table 6–11. Port 0 Control CSR—Offset: 0x15C (Part 1 of 2)

Field	Bits	Access	Function	Default
PORT_WIDTH	[31:30]	RO	Hardware width of the port: 'b00—Single-lane port. 'b01—Four-lane port. 'b10–'b11—Reserved.	2'b00 (for 1× variations), 2'b01 (for 4× variations) (1)
INIT_WIDTH	[29:27]	RO	Width of the ports after being initialized: 'b000—Single lane port, lane 0. 'b001—Single lane port, lane 2. 'b010—Four lane port. 'b011–'b111—Reserved.	3'b000 (for 1× variations), 3'b010 (for 4× variations)
PWIDTH_OVRIDE	[26:24]	UR0	Soft port configuration to override the hardware size: 'b000—No override. 'b001—Reserved. 'b010—Force single lane, lane 0. 'b011—Force single lane, lane 2. 'b100–'b111—Reserved.	3'b000
PORT_DIS	[23]	RW	Port disable: 'b0—Port receivers/drivers are enabled. 'b1—Port receivers are disabled, causing the drivers to send out idles. ■ When this bit transitions from 1 to 0, the initialization state machines' force_reinit signal is asserted. This assertion causes the port to enter the *SILENT* state and to attempt to reinitialize the link, as described in section 4.12 of *Part 6: LP-Serial Physical Layer Specification of the RapidIO Interconnect Specification, Revision 2.1*. ■ When reception is disabled, the input buffers are kept empty until this bit is cleared. ■ When PORT_DIS is asserted and the drivers are disabled, the transmit buffer are reset and kept empty until this bit is cleared, any previously stored packets are lost, and any attempt to write a packet to the atx Atlantic interface is ignored by the Physical layer. New packets are NOT stored for later transmission.	1'b0
OUT_PENA	[22]	RW	Output port transmit enable: 'b0—Port is stopped and not enabled to issue any packets except to route or respond to I/O logical MAINTENANCE packets, depending upon the functionality of the processing element. Control symbols are not affected and are sent normally. 'b1—Port is enabled to issue packets.	1'b1

Table 6–11. Port 0 Control CSR—Offset: 0x15C (Part 2 of 2)

Field	Bits	Access	Function	Default
IN_PENA	[21]	RW	Input port receive enable: 'b0—Port is stopped and only enabled to respond I/O Logical MAINTENANCE requests. Other requests return packet-not-accepted control symbols to force an error condition to be signaled by the sending device 'b1—Port is enabled to respond to any packet	1'b1
ERR_CHK_DIS	[20]	RW	This bit controls all RapidIO transmission error checking: 'b0—Error checking and recovery is enabled 'b1—Error checking and recovery is disabled Device behavior when error checking and recovery is disabled and an error condition occurs is undefined.	1'b0
Multicast-event Participant	[19]	RW	Send incoming Multicast-event control symbols to this port (multiple port devices only).	1'b0
RSRV	[18]	RO	Reserved	1'b0
Enumeration Boundary	[17]	RO	This feature is not supported.	1'b0
RSRV	[16:12]	RO	Reserved	5'b0
Re-transmit Suppression Mask	[11:4]	RO	This feature is not supported.	8'b0
RSRV	[3:1]	RO	Reserved	3'b0
PORT_TYPE	[0]	RO	This bit indicates the port type, parallel or serial. 'b0—Parallel port 'b1—Serial port	1'b1

Note to Table 6–11:

(1) Reflects the choice made in the RapidIO parameter editor.

Transport and Logical Layer Registers

This section lists the Transport and Logical layer registers. Table 6–2 provides a memory map of all accessible registers. This address space is accessible to the user through the System Maintenance Avalon-MM slave interface.

Capability Registers (CARs)

Table 6–12 through Table 6–19 describe the capability registers.

Table 6–12. Device Identity CAR—Offset: 0x00

Field	Bits	Access	Function	Default
DEVICE_ID	[31:16]	RO	Hard-wired device identifier	(1)
VENDOR_ID	[15:0]	RO	Hard-wired device vendor identifier	(1)

Note to Table 6–12:

(1) The default value is set in the RapidIO parameter editor.

Table 6–13. Device Information CAR—Offset: 0x04

Field	Bits	Access	Function	Default
DEVICE_REV	[31:0]	RO	Hard-wired device revision level	(1)

Note to Table 6–13:

(1) The default value is set in the RapidIO parameter editor.

Table 6–14. Assembly Identity CAR—Offset: 0x08

Field	Bits	Access	Function	Default
ASSY_ID	[31:16]	RO	Hard-wired assembly identifier	(1)
ASSY_VENDOR_ID	[15:0]	RO	Hard-wired assembly vendor identifier	(1)

Note to Table 6–14:

(1) The default value is set in the RapidIO parameter editor.

Table 6–15. Assembly Information CAR—Offset: 0x0C

Field	Bits	Access	Function	Default
ASSY_REV	[31:16]	RO	Hard-wired assembly revision level	(1)
EXT_FEATURE_PTR	[15:0]	RO	Hard-wired pointer to the first entry in the extended feature list. This pointer must be in the range of 16'h100 and 16'hFFFC.	(1)

Note to Table 6–15:

(1) The default value is set in the RapidIO parameter editor.

Table 6–16. Processing Element Features CAR—Offset: 0x10 (Part 1 of 2)

Field	Bits	Access	Function	Default
BRIDGE	[31]	RO	Processing element can bridge to another interface.	(1)
MEMORY	[30]	RO	Processing element has physically addressable local address space and can be accessed as an endpoint through nonmaintenance operations. This local address space may be limited to local configuration registers, on-chip SRAM, or other device.	(1)
PROCESSOR	[29]	RO	Processing element physically contains a local processor or similar device that executes code. A device that bridges to an interface that connects to a processor does not count.	(1)
SWITCH	[28]	RO	Processing element can bridge to another external RapidIO interface—an internal port to a local endpoint does not count as a switch port.	(1)
RSRV	[27:7]	RO	Reserved	21'h0
RE_TRAN_SUP	[6]	RO	Processing element supports suppression of error recovery on packet CRC errors: 1'b0—The error recovery suppression option is not supported 1'b1—The error recovery suppression option is supported	1'b0

Table 6–16. Processing Element Features CAR—Offset: 0x10 (Part 2 of 2)

Field	Bits	Access	Function	Default
CRF_SUPPORT	[5]	RO	Processing element supports the Critical Request Flow (CRF) indicator: 1'b0—Critical Request Flow is not supported 1'b1—Critical Request Flow is supported	1'b0
LARGE_TRANSPORT	[4]	RO	Processing element supports common transport large systems: 1'b0—Processing element does not support common transport large systems (device ID width is 8 bits). 1'b1—Processing element supports common transport large systems (device ID width is 16 bits). The value of this field is determined by the device ID width you select in the RapidIO parameter editor.	(1)
EXT_FEATURES	[3]	RO	Processing element has extended features list; the extended features pointer is valid.	1'b1
EXT_ADDR_SPRT	[2:0]	RO	Indicates the number of address bits supported by the processing element, both as a source and target of an operation. All processing elements support a minimum 34-bit addresses: 3'b111—Processing element supports 66, 50, and 34-bit addresses 3'b101—Processing element supports 66 and 34-bit addresses 3'b011—Processing element supports 50 and 34-bit addresses 3'b001—Processing element supports 34-bit addresses	3'b001

Note to Table 6–16:

(1) The default value is set in the RapidIO parameter editor.

Table 6–17. Switch Port Information CAR—Offset: 0x14

Field	Bits	Access	Function	Default
RSRV	[31:16]	RO	Reserved	16'h0
PORT_TOTAL	[15:8]	RO	The total number of RapidIO ports on the processing element: 8'h0—Reserved 8'h1—1 port 8'h2—2 ports ... 8'hFF—255 ports	(1)
PORT_NUMBER	[7:0]	RO	This is the port number from which the MAINTENANCE read operation accessed this register. Ports are numbered starting with 'h0.	(1)

Note to Table 6–17:

(1) The default value is set in the RapidIO parameter editor.

Table 6–18. Source Operations CAR—Offset: 0x18 (1)

Field	Bits	Access	Function	Default
RSRV	[31:16]	RO	Reserved	16'h0
READ	[15]	RO	Processing element can support a read operation	(2)
WRITE	[14]	RO	Processing element can support a write operation	(2)
SWRITE	[13]	RO	Processing element can support a streaming-write operation	(2)
NWRITE_R	[12]	RO	Processing element can support a write-with-response operation	(2)
Data Message	[11]	RO	Processing element can support data message operation	(3)
DOORBELL	[10]	RO	Processing element can support a DOORBELL operation	(4)
ATM_COMP_SWP	[9]	RO	Processing element can support an ATOMIC compare-and-swap operation	1'b0
ATM_TEST_SWP	[8]	RO	Processing element can support an ATOMIC test-and-swap operation	1'b0
ATM_INC	[7]	RO	Processing element can support an ATOMIC increment operation	1'b0
ATM_DEC	[6]	RO	Processing element can support an ATOMIC decrement operation	1'b0
ATM_SET	[5]	RO	Processing element can support an ATOMIC set operation	1'b0
ATM_CLEAR	[4]	RO	Processing element can support an ATOMIC clear operation	1'b0
ATM_SWAP	[3]	RO	Processing element can support an ATOMIC swap operation	1'b0
PORT_WRITE	[2]	RO	Processing element can support a port-write operation	(5)
Implementation Defined	[1:0]	RO	Reserved for this implementation	2'b00

Notes to Table 6–18:

(1) If one of the Logical layers supported by the RapidIO MegaCore is not selected in the MegaWizard Plug-In Manager, the corresponding bits in the Source and Destination Operations CARs are forced to zero. These bits cannot be set to one, even if the corresponding operations are supported by user logic attached to the Avalon-ST pass-through interface.

(2) The default value is 1'b1 if the I/O Logical layer interface **Avalon-MM Slave** was selected in the RapidIO parameter editor. The value is 1'b0 if the I/O Logical layer interface **Avalon-MM Slave** was not selected in the RapidIO parameter editor.

(3) The default value is set in the RapidIO parameter editor.

(4) The default value is 1'b1 if **Doorbell Tx enable** is turned on in the RapidIO parameter editor. If **Doorbell Tx enable** is turned off, the value is 1'b0.

(5) The default value is 1'b1 if **Port Write Tx enable** is turned on in the RapidIO parameter editor. If **Port Write Tx enable** is turned off, the value is 1'b0.

Table 6–19. Destination Operations CAR—Offset: 0x1C (1) **(Part 1 of 2)**

Field	Bits	Access	Comment	Default
RSRV	[31:16]	RO	Reserved	16'h0
READ	[15]	RO	Processing element can support a read operation	(2)
WRITE	[14]	RO	Processing element can support a write operation	(2)
SWRITE	[13]	RO	Processing element can support a streaming-write operation	(2)
NWRITE_R	[12]	RO	Processing element can support a write-with-response operation	(2)
Data Message	[11]	RO	Processing element can support data message operation	(3)
DOORBELL	[10]	RO	Processing element can support a DOORBELL operation	(4)
ATM_COMP_SWP	[9]	RO	Processing element can support an ATOMIC compare-and-swap operation	1'b0

Table 6–19. Destination Operations CAR—Offset: 0x1C (1) (Part 2 of 2)

Field	Bits	Access	Comment	Default
ATM_TEST_SWP	[8]	RO	Processing element can support an ATOMIC test-and-swap operation	1'b0
ATM_INC	[7]	RO	Processing element can support an ATOMIC increment operation	1'b0
ATM_DEC	[6]	RO	Processing element can support an ATOMIC decrement operation	1'b0
ATM_SET	[5]	RO	Processing element can support an ATOMIC set operation	1'b0
ATM_CLEAR	[4]	RO	Processing element can support an ATOMIC clear operation	1'b0
ATM_SWAP	[3]	RO	Processing element can support an ATOMIC swap operation	1'b0
PORT_WRITE	[2]	RO	Processing element can support a port-write operation	(5)
Implementation Defined	[1:0]	RO	Reserved for this implementation	2'b00

Notes to Table 6–19:

(1) If none of the Logical layers supported by the RapidIO MegaCore is selected, the corresponding bits in the Source and Destination Operations CAR are forced to zero. These bits cannot be set to one, even if the corresponding operations are supported by user logic attached to the Avalon-ST pass-through interface.

(2) The default value is 1'b1 if the **Avalon-MM Master** is selected as an Input/Output Logical layer interface in the RapidIO parameter editor. If the **Avalon-MM Master** is not selected, the value is 1'b0.

(3) The default value is set in the RapidIO parameter editor.

(4) The default value is 1'b1 if **Doorbell Rx enable** is turned on in the RapidIO parameter editor. If **Doorbell Rx enable** is turned off, the value is 1'b0.

(5) The default value element is 1'b1 if **Port Write Rx enable** is turned on in the RapidIO parameter editor. If **Port Write Rx enable** is turned off, the value is 1'b0.

Command and Status Registers (CSRs)

Table 6–20 through Table 6–25 describe the command and status registers.

Table 6–20. Processing Element Logical Layer Control CSR—Offset: 0x4C

Field	Bits	Access	Function	Default
RSRV	[31:3]	RO	Reserved	29'h0
EXT_ADDR_CTRL	[2:0]	RO	Controls the number of address bits generated by the Processing element as a source and processed by the Processing element as the target of an operation. 'b100 – Processing element supports 66 bit addresses 'b010 – Processing element supports 50 bit addresses 'b001 – Processing element supports 34 bit addresses All other encodings reserved	3'b001

Table 6–21. Local Configuration Space Base Address 0 CSR—Offset: 0x58

Field	Bits	Access	Function	Default
RSRV	[31]	RO	Reserved	1'b0
LCSBA	[30:15]	RO	Reserved for a 34-bit local physical address	16'h0
LCSBA	[14:0]	RO	Reserved for a 34-bit local physical address	15'h0

Table 6–22. Local Configuration Space Base Address 1 CSR—Offset: 0x5C (1)

Field	Bits	Access	Function	Default
LCSBA	[31]	RO	Reserved for a 34-bit local physical address	1'b0
LCSBA	[30:0]	RW	Bits 33:4 of a 34-bit physical address	31'h0

Note to Table 6–22:

(1) The Local Configuration Space Base Address registers are hard coded to zero. If the Input/Output Avalon-MM master interface is connected to the System Maintenance Avalon-MM slave interface, regular read and write operations rather than MAINTENANCE operations, can be used to access the processing element's registers for configuration and maintenance.

Table 6–23. Base Device ID CSR—Offset: 0x60

Field	Bits	Access	Function	Default
RSRV	[31:24]	RO	Reserved	8'h0
DEVICE_ID (1)	[23:16]	RW	This is the base ID of the device in a small common transport system.	8'hFF
		RO	Reserved if the system does not support 8-bit device ID.	
LARGE_DEVICE_ID (1)	[15:0]	RW	This is the base ID of the device in a large common transport system.	16'hFFFF
		RO	Reserved if the system does not support 16-bit device ID.	

Note to Table 6–23:

(1) In a small common transport system, the DEVICE_ID field is Read-Write and the LARGE_DEVICE_ID field is Read-only. In a large common transport system, the DEVICE_ID field is Read-only and the LARGE_DEVICE_ID field is Read-Write.

Table 6–24. Host Base Device ID Lock CSR—Offset: 0x68

Field	Bits	Access	Function	Default
RSRV	[31:16]	RO	Reserved	16'h0
HOST_BASE_DEVICE_ID	[15:0]	RW (1)	This is the base device ID for the processing element that is initializing this processing element.	16'hFFFF

Note to Table 6–24:

(1) Write once; can be reset. See Part 3 §3.5.2 of the *RapidIO Specification Rev 2.1* for more information.

Table 6–25. Component Tag CSR—Offset: 0x6C

Field	Bits	Access	Function	Default
COMPONENT_TAG	[31:0]	RW	This is a component tag for the processing element.	32'h0

Maintenance Interrupt Control Registers

Table 6–26 and Table 6–27 describe the registers that relate to the Maintenance module interrupts. If any of these error conditions are detected and if the corresponding Interrupt Enable bit is set, the sys_mnt_s_irq signal is asserted.

Table 6–26. Maintenance Interrupt—Offset: 0x10080 (Part 1 of 2)

Field	Bits	Access	Function	Default
RSRV	[31:7]	RO	Reserved	25'h0
PORT_WRITE_ERROR	[6]	RW1C	Port-write error	1'b0

Table 6–26. Maintenance Interrupt—Offset: 0x10080 (Part 2 of 2)

Field	Bits	Access	Function	Default
PACKET_DROPPED	[5]	RW1C	A received port-write packet was dropped. A port-write packet is dropped under the following conditions: ■ A port-write request packet is received but port-write reception has not been enabled by setting bit PORT_WRITE_ENABLE in the Rx Port Write Control register. ■ A previously received port-write has not been read out from the Rx Port Write register.	1'b0
PACKET_STORED	[4]	RW1C	Indicates that the IP core has received a port-write packet and that the payload can be retrieved using the System Maintenance Avalon-MM slave interface.	1'b0
RSRV	[3]	RO	Reserved	1'b0
RSRV	[2]	RO	Reserved	1'b0
WRITE_OUT_OF_BOUNDS	[1]	RW1C	If the address of an Avalon-MM write transfer presented at the Maintenance Avalon-MM slave interface does not fall within any of the enabled Tx Maintenance Address translation windows, then it is considered out of bounds and this bit is set.	1'b0
READ_OUT_OF_BOUNDS	[0]	RW1C	If the address of an Avalon-MM read transfer presented at the Maintenance Avalon-MM slave interface does not fall within any of the enabled Tx Maintenance Address translation windows, then it is considered out of bounds and this bit is set.	1'b0

Table 6–27. Maintenance Interrupt Enable—Offset: 0x10084

Field	Bit	Access	Function	Default
RSRV	[31:7]	RO	Reserved	25'h0
PORT_WRITE_ERROR	[6]	RW	Port-write error interrupt enable	1'b0
RX_PACKET_DROPPED	[5]	RW	Rx port-write packet dropped interrupt enable	1'b0
RX_PACKET_STORED	[4]	RW	Rx port-write packet stored in buffer interrupt enable	1'b0
RSRV	[3:2]	RO	Reserved	2'b00
WRITE_OUT_OF_BOUNDS	[1]	RW	Tx write request address out of bounds interrupt enable	1'b0
READ_OUT_OF_BOUNDS	[0]	RW	Tx read request address out of bounds interrupt enable	1'b0

Receive Maintenance Registers

Table 6–28 describes the receiver maintenance register.

Table 6–28. Rx Maintenance Mapping—Offset: 0x10088

Field	Bits	Access	Function	Default
RX_BASE	[31:24]	RW	Rx base address. The offset value carried in a received MAINTENANCE Type packet is concatenated with this RX_BASE to form a 32-bit Avalon Address as follows: Avalon_address = {rx_base, cfg_offset, word_addr, 2'b00}	8'h0
RSRV	[23:0]	RO	Reserved	24'h0

Transmit Maintenance Registers

Table 6–29 through Table 6–32 describe the transmitter maintenance registers. When transmitting a MAINTENANCE packet, an address translation process occurs, using a base, mask, offset, and control register. As many as sixteen groups of four registers can exist. The 16 register address offsets are shown in the table titles. For more details on how to use these windows, refer to "Maintenance Slave Processor" on page 4–30.

Table 6–29. Tx Maintenance Mapping Window n Base—Offset: 0x10100, 0x10110, 0x10120, 0x10130, 0x10140, 0x10150, 0x10160, 0x10170, 0x10180, 0x10190, 0x101A0, 0x101B0, 0x101C0, 0x101D0, 0x101E0, 0x101F0

Field	Bits	Access	Function	Default
BASE	[31:3]	RW	Start of the Avalon-MM address window to be mapped. The three least significant bits of the 32-bit base are assumed to be zero.	29'h0
RSRV	[2:0]	RO	Reserved	3'h0

Table 6–30. Tx Maintenance Mapping Window n Mask—Offset: 0x10104, 0x10114, 0x10124, 0x10134, 0x10144, 0x10154, 0x10164, 0x10174, 0x10184, 0x10194, 0x101A4, 0x101B4, 0x101C4, 0x101D4, 0x101E4, 0x101F4

Field	Bits	Access	Function	Default
MASK	[31:3]	RW	Mask for the address mapping window. The three least significant bits of the 32-bit mask are assumed to be zero.	29'h0
WEN	[2]	RW	Window enable. Set to one to enable the corresponding window.	1'b0
RSRV	[1:0]	RO	Reserved	2'h0

Table 6–31. Tx Maintenance Mapping Window n Offset—Offset: 0x10108, 0x10118, 0x10128, 0x10138, 0x10148, 0x10158, 0x10168, 0x10178, 0x10188, 0x10198, 0x101A8, 0x101B8, 0x101C8, '0x101D8, 0x101E8, 0x101F8

Field	Bits	Access	Function	Default
RSRV	[31:24]	RO	Reserved	8'h0
OFFSET	[23:0]	RW	Window offset	24'h0

Table 6–32. Tx Maintenance Mapping Window n Control—Offset: 0x1010C, 0x1011C, 0x1012C, 0x1013C, 0x1014C, 0x1015C, 0x1016C, 0x1017C, 0x1018C, 0x1019C, 0x101AC, 0x101BC, 0x101CC, 0x101DC, 0x101EC, 0x101FC

Field	Bits	Access	Function	Default
LARGE_DESTINATION_ID (MSB)	[31:24]	RO	Reserved if the system does not support 16-bit device ID.	8'h0
		RW	MSB of the Destination ID if the system supports 16-bit device ID.	
DESTINATION_ID	[23:16]	RW	Destination ID	8'h0
HOP_COUNT	[15:8]	RW	Hop count	8'hFF
PRIORITY	[7:6]	RW	Packet priority. 2'b11 is not a valid value for the PRIORITY field. Any attempt to write 2'b11 to this field is overwritten with 2'b10.	2'b00
RSRV	[5:0]	RO	Reserved	6'h0

Transmit Port-Write Registers

Table 6–33 through Table 6–35 describe the transmit port-write registers.

Refer to "Port-Write Processor" on page 4–34 for information about using these registers to transmit a port-write.

Table 6–33. Tx Port Write Control—Offset: 0x10200

Field	Bits	Access	Function	Default
LARGE_DESTINATION_ID (MSB)	[31:24]	RO	Reserved if the system does not support 16-bit device ID.	8'h0
		RW	MSB of the Destination ID if the system supports 16-bit device ID.	
DESTINATION_ID	[23:16]	RW	Destination ID	8'h0
RSRV	[15:8]	RO	Reserved	8'h00
PRIORITY	[7:6]	RW	Request packet's priority. 2'b11 is not a valid value for the priority field. An attempt to write 2'b11 to this field is overwritten as 2'b10.	2'b00
SIZE	[5:2]	RW	Packet payload size in number of double words. If set to 0, the payload size is single word. If size is set to a value larger than 8, the payload size is 8 double words (64 bytes).	4'h0
RSRV	[1]	RO	Reserved	1'b0
PACKET_READY	[0]	RW	Write 1 to start transmitting the port-write request. This bit is cleared internally after the packet has been transferred to the Transport layer to be forwarded to the Physical layer for transmission.	1'b0

Table 6–34. Tx Port Write Status—Offset: 0x10204

Field	Bits	Access	Function	Default
RSRV	[31:0]	RO	Reserved	31'h0

Table 6–35. Tx Port Write Buffer n—Offset: 0x10210 – 0x1024C

Field	Bits	Access	Function	Default
PORT_WRITE_DATA_n	[31:0]	RW	Port-write data. This buffer is implemented in memory and is not initialized at reset.	32'hx

Receive Port-Write Registers

Table 6–36 through Table 6–38 describe the receive port-write registers.

Refer to "Port-Write Reception Module" on page 4–68 for information about receiving port write MAINTENANCE packets.

Table 6–36. Rx Port Write Control—Offset: 0x10250

Field	Bits	Access	Function	Default
RSRV	[31:2]	RO	Reserved	30'h0
CLEAR_BUFFER	[1]	RW	Clear port-write buffer. Write 1 to activate. Always read 0.	1'b0
PORT_WRITE_ENA	[0]	RW	Port-write enable. If set to 1, port-write packets are accepted. If set to 0, port-write packets are dropped.	1'b1

Table 6–37. Rx Port Write Status—Offset: 0x10254

Field	Bits	Access	Function	Default
RSRV	[31:6]	RO	Reserved	26'h0
PAYLOAD_SIZE	[5:2]	RO	Packet payload size in number of double words. If the size is zero, the payload size is single word.	4'h0
RSRV	[1]	RO	Reserved	1'b0
PORT_WRITE_BUSY	[0]	RO	Port-write busy. Set if a packet is currently being stored in the buffer or if the packet is stored and has not been read.	1'b0

Table 6–38. Rx Port Write Buffer n—Offset: 0x10260 – 0x1029C

Field	Bits	Access	Function	Default
PORT_WRITE_DATA_n	[31:0]	RO	Port-write data. This buffer is implemented in memory and is not initialized at reset.	32'hx

Input/Output Master Address Mapping Registers

Table 6–39 through Table 6–41 describe the Input/Output master registers. When the IP core receives an NREAD, NWRITE, NWRITE_R, or SWRITE request packet, the RapidIO address has to be translated into a local Avalon-MM address. The translation involves the base, mask, and offset registers. There are up to 16 register sets, one for each address mapping window. The 16 possible register address offsets are shown in the table titles.

Refer to "Input/Output Avalon-MM Master Address Mapping Windows" on page 4–37 for more details.

Table 6–39. Input/Output Master Mapping Window n Base—Offset: 0x10300, 0x10310, 0x10320, 0x10330, 0x10340, 0x10350, 0x10360, 0x10370, 0x10380, 0x10390, 0x103A0, 0x103B0, 0x103C0, 0x103D0, 0x103E0, 0x103F0

Field	Bits	Access	Function	Default
BASE	[31:3]	RW	Start of the RapidIO address window to be mapped. The three least significant bits of the 34-bit base are assumed to be zeros.	29'h0
RSRV	[2]	RO	Reserved	1'b0
XAMB	[1:0]	RW	Extended Address: two most significant bits of the 34-bit base.	2'h0

Table 6–40. Input/Output Master Mapping Window n Mask—Offset: 0x10304, 0x10314, 0x10324, 0x10334, 0x10344, 0x10354, 0x10364, 0x10374, 0x10384, 0x10394, 0x103A4, 0x103B4, 0x103C4, 0x103D4, 0x103E4, 0x103F4

Field	Bits	Access	Function	Default
MASK	[31:3]	RW	Bits 31 to 3 of the mask for the address mapping window. The three least significant bits of the 34-bit mask are assumed to be zeros.	29'h0
WEN	[2]	RW	Window enable. Set to one to enable the corresponding window.	1'b0
XAMM	[1:0]	RW	Extended Address: two most significant bits of the 34-bit mask.	2'b0

Table 6–41. Input/Output Master Mapping Window n Offset—Offset: 0x10308, 0x10318, 0x10328, 0x10338, 0x10348, 0x10358, 0x10368, 0x10378, 0x10388, 0x10398, 0x103A8, 0x103B8, 0x103C8, 0x103D8, 0x103E8, 0x103F8

Field	Bits	Access	Function	Default
OFFSET	[31:3]	RW	Starting offset into the Avalon-MM address space. The three least significant bits of the 32-bit offset are assumed to be zero.	29'h0
RSRV	[2:0]	RO	Reserved	3'h0

Input/Output Slave Mapping Registers

Table 6–42 through Table 6–47 describe the Input/Output slave registers. The registers define windows in the Avalon-MM address space that are used to determine the outgoing request packet's ftype, DESTINATION_ID, priority, and address fields. There are up to 16 register sets, one for each possible address mapping window. The 16 possible register address offsets are shown in the table titles.

Refer to "Input/Output Avalon-MM Slave Address Mapping Windows" on page 4–46 for a description of how to use these registers.

Table 6–42. Input/Output Slave Mapping Window n Base—Offset: 0x10400, 0x10410, 0x10420, 0x10430, 0x10440, 0x10450, 0x10460, 0x10470, 0x10480, 0x10490, 0x104A0, 0x104B0, 0x104C0, 0x104D0, 0x104E0, 0x104F0

Field	Bits	Access	Function	Default
BASE	[31:3]	RW	Start of the Avalon-MM address window to be mapped. The three least significant bits of the 32-bit base are assumed to be all zeros.	29'h0
RSRV	[2:0]	RO	Reserved	3'h0

Table 6–43. Input/Output Slave Mapping Window n Mask—Offset: 0x10404, 0x10414, 0x10424, 0x10434, 0x10444, 0x10454, 0x10464, 0x10474, 0x10484, 0x10494, 0x104A4, 0x104B4, 0x104C4, 0x104D4, 0x104E4, 0x104F4

Field	Bits	Access	Function	Default
MASK	[31:3]	RW	29 most significant bits of the mask for the address mapping window. The three least significant bits of the 32-bit mask are assumed to be zeros.	29'h0
WEN	[2]	RW	Window enable. Set to one to enable the corresponding window.	1'b0
RSRV	[1:0]	RO	Reserved	2'h0

Table 6–44. Input/Output Slave Mapping Window n Offset—Offset: 0x10408, 0x10418, 0x10428, 0x10438, 0x10448, 0x10458, 0x10468, 0x10478, 0x10488, 0x10498, 0x104A8, 0x104B8, 0x104C8, 0x104D8, 0x104E8, 0x104F8

Field	Bits	Access	Function	Default
OFFSET	[31:3]	RW	Bits [31:3] of the starting offset into the RapidIO address space. The three least significant bits of the 34-bit offset are assumed to be zeros.	29'h0
RSRV	[2]	RO	Reserved	1'b0
XAMO	[1:0]	RW	Extended Address: two most significant bits of the 34-bit offset.	2'h0

Table 6–45. Input/Output Slave Mapping Window n Control—Offset: 0x1040C, 0x1041C, 0x1042C, 0x1043C, 0x1044C, 0x1045C, 0x1046C, 0x1047C, 0x1048C, 0x1049C, 0x104AC, 0x104BC, 0x104CC, 0x104DC, 0x104EC, 0x104FC

Field	Bits	Access	Function	Default
LARGE_DESTINATION_ID (MSB)	[31:24]	RO	Reserved if the system does not support 16-bit device ID.	8'h0
		RW	MSB of the Destination ID if the system supports 16-bit device ID.	
DESTINATION_ID	[23:16]	RW	Destination ID	8'h0
RSRV	[15:8]	RO	Reserved	8'h0
PRIORITY	[7:6]	RW	Request Packet's priority 2'b11 is not a valid value for the priority field. Any attempt to write 2'b11 to this field is overwritten with 2'b10.	2'h0
RSRV	[5:2]	RO	Reserved	4'h0
SWRITE_ENABLE	[1]	RW	SWRITE enable. Set to one to generate SWRITE request packets. (1)	1'b0
NWRITE_R_ENABLE	[0]	RW	NWRITE_R enable (1)	1'b0

Note to Table 6–45:

(1) Bits 0 and 1 (NWRITE_R_ENABLE and SWRITE_ENABLE) are mutually exclusive. An attempt to write ones to both of these fields at the same time is ignored, and that part of the register keeps its previous value.

Input/Output Slave Interrupts

Table 6–46 and Table 6–47 describe the available Input/Output slave interrupts and corresponding interrupt enable bits. These interrupt bits assert the sys_mnt_s_irq signal if the corresponding interrupt bit is enabled.

Table 6–46. Input/Output Slave Interrupt—Offset: 0x10500 (Part 1 of 2)

Field	Bits	Access	Function	Default
RSRV	[31:5]	RO	Reserved	27'h0
NWRITE_RS_COMPLETED	[4]	RW1C	Indicates no pending NWRITE_R transactions remain in the RapidIO IP core. Set when the PENDING_NWRITE_RS field of the Input/Output Slave Pending NWRITE_R Transactions register (offset 0x10508) is set to 0. Because of the inherent delay in incrementing the PENDING_NWRITE_RS field after the start of the corresponding write transaction on the Avalon-MM interface, you should wait at least 8 Avalon clock cycles after the start of the NWRITE_R transaction whose completion you wish to trigger an interrupt, before you clear this bit and enable this interrupt.	1'b0
INVALID_WRITE_BYTEENABLE	[3]	RW1C	Write byte enable invalid. Asserted when io_s_wr_byteenable is set to invalid values. For information about valid values see Table 4–15 and Table 4–17.	1'b0
INVALID_WRITE_BURSTCOUNT	[2]	RW1C	Write burst count invalid. Asserted when io_s_wr_burstcount is set to an odd number larger than one in variations with 32-bit wide datapath Avalon-MM write interfaces.	1'b0

Table 6–46. Input/Output Slave Interrupt—Offset: 0x10500 (Part 2 of 2)

Field	Bits	Access	Function	Default
WRITE_OUT_OF_BOUNDS	[1]	RW1C	Write request address out of bounds. Asserted when the Avalon-MM address does not fall within any enabled address mapping windows.	1'b0
READ_OUT_OF_BOUNDS	[0]	RW1C	Read request address out of bounds. Asserted when the Avalon-MM address does not fall within any enabled address mapping windows.	1'b0

Table 6–47. Input/Output Slave Interrupt Enable—Offset: 0x10504

Field	Bits	Access	Function	Default
RSRV	[31:5]	RO	Reserved	27'h0
NWRITE_RS_COMPLETED	[4]	RW	NWRITE_Rs-completed field enable.	1'b0
INVALID_WRITE_BYTEENABLE	[3]	RW	Write byte enable invalid interrupt enable	1'b0
INVALID_WRITE_BURSTCOUNT	[2]	RW	Write burst count invalid interrupt enable	1'b0
WRITE_OUT_OF_BOUNDS	[1]	RW	Write request address out of bounds interrupt enable	1'b0
READ_OUT_OF_BOUNDS	[0]	RW	Read request address out of bounds interrupt enable	1'b0

Table 6–48. Input/Output Slave Pending NWRITE_R Transactions—Offset: 0x10508

Field	Bits	Access	Function	Default
RSRV	[31:5]	RO	Reserved	27'h0
PENDING_NWRITE_RS	[4:0]	RO	Number of pending NWRITE_R write requests that have been initiated in the I/O Avalon-MM slave Logical layer module but have not yet completed. The value in this field might update only after a delay of 8 Avalon clock cycles after the start of the write burst on the Avalon-MM interface.	5'b0

Table 6–49. Input/Output Slave Avalon-MM Write Transactions—Offset: 0x1050C

Field	Bits	Access	Function	Default
RSRV	[31:16]	RO	Reserved	16'h0
STARTED_WRITES	[15:0]	RO	Number of write transfers initiated on Avalon-MM Input/Output Slave port so far. Count increments on first system clock cycle in which the io_s_wr_write and io_s_wr_chipselect signals are asserted and the io_s_wr_waitrequest signal is not asserted. This counter rolls over to 0 after its maximum value.	16'b0

Table 6–50. Input/Output Slave RapidIO Write Requests—Offset: 0x10510

Field	Bits	Access	Function	Default
RSRV	[31:16]	RO	Reserved	16'h0
COMPLETED_OR_CANCELLED_WRITES	[15:0]	RO	Number of write-request packets transferred from the Avalon-MM Input/Output Slave module to the Transport layer or cancelled. Count increments when the write-request packet is sent to the Transport layer, or when a write transaction is cancelled. This counter rolls over to 0 after its maximum value.	16'b0

Transport Layer Feature Register

Table 6–51 describes the Rx Transport Control register. This register controls the Transport layer mode.

Table 6–51. Rx Transport Control—Offset: 0x10600

Field	Bits	Access	Function	Default
RSRV	[31:1]	RO	Reserved	31'h0
PROMISCUOUS_MODE	[0]	RW	This bit determines whether the Transport layer checks destination IDs in incoming request packets, or promiscuously accepts all incoming request packets with a supported ftype. The reset value is set in the RapidIO parameter editor.	(1)

Note to Table 6–51:

(1) The default value is set in the RapidIO parameter editor.

Error Management Registers

Table 6–52 through Table 6–56 describe the error management registers. These registers can be used by software to diagnose problems with packets that are received by the local endpoint. If enabled, the detected error triggers the assertion of sys_mnt_s_irq. Information about the packet that caused the error is captured in the capture registers. After an error condition is detected, the information is captured and the capture registers are locked until the Error Detect CSR is cleared. Upon being cleared, the capture registers are ready to capture a new packet that exhibits an error condition.

Table 6–52. Logical/Transport Layer Error Detect CSR—Offset: 0x10800 (Part 1 of 2)

Field	Bits	Access	Function	Default
IO_ERROR_RSP	[31]	RW	Received a response of ERROR for an I/O Logical Layer Request.	1'b0
MSG_ERROR_RESPONSE	[30]	RW	Received a response of ERROR for a MSG Logical Layer Request.	1'b0
GSM error response	[29]	RO	This feature is not supported.	1'b0
MSG_FORMAT_ERROR	[28]	RW	Received MESSAGE packet data payload with an invalid size or segment.	1'b0
ILL_TRAN_DECODE	[27]	RW	Received illegal fields in the request/response packet for a supported transaction.	1'b0

Table 6–52. Logical/Transport Layer Error Detect CSR—Offset: 0x10800 (Part 2 of 2)

Field	Bits	Access	Function	Default
ILL_TRAN_TARGET	[26]	RW	Received a packet that contained a destination ID that is not defined for this end point.	1'b0
MSG_REQ_TIMEOUT	[25]	RW	A required message request has not been received within the specified time-out interval.	1'b0
PKT_RSP_TIMEOUT	[24]	RW	A required response has not been received within the specified time-out interval.	1'b0
UNSOLICIT_RSP	[23]	RW	An unsolicited/unexpected response packet was received.	1'b0
UNSUPPORT_TRAN	[22]	RW	A transaction is received that is not supported in the Destination Operations CAR.	1'b0
RSRV	[21:8]	RO	Reserved	22'h0
Implementation Specific error	[7:0]	RO	This feature is not supported.	8'b0

Table 6–53. Logical/Transport Layer Error Enable CSR—Offset: 0x10804 (Part 1 of 2)

Field	Bits	Access	Function	Default
IO_ERROR_RSP_EN	[31]	RW	Enable reporting of an I/O error response. Save and lock original request transaction information in all Logical/Transport Layer Capture CSRs.	1'b0
MSG_ERROR_RESPONSE_EN	[30]	RW	Enable reporting of a Message error response. Save and lock original request transaction information in all Logical/Transport Layer Capture CSRs.	1'b0
GSM error response enable	[29]	RO	This feature is not supported.	1'b0
MSG_FORMAT_ERROR_EN	[28]	RW	Enable reporting of a message format error. Save and lock original request transaction information in all Logical/Transport Layer Capture CSRs.	1'b0
ILL_TRAN_DECODE_EN	[27]	RW	Enable reporting of an illegal transaction decode error. Save and lock transaction capture information in Logical/Transport Layer Device ID and Control Capture CSRs.	1'b0
ILL_TRAN_TARGET_EN	[26]	RW	Enable reporting of an illegal transaction target error. Save and lock transaction capture information in Logical/Transport Layer Device ID and Control Capture CSRs.	1'b0
MSG_REQ_TIMEOUT_EN	[25]	RW	Enable reporting of a Message Request time-out error. Save and lock original request transaction information in Logical/Transport Layer Device ID and Control Capture CSRs for the last Message request segment packet received.	1'b0
PKT_RSP_TIMEOUT_EN	[24]	RW	Enable reporting of a packet response time-out error. Save and lock original request address in Logical/Transport Layer Address Capture CSRs. Save and lock original request destination ID in Logical/Transport Layer Device ID Capture CSR.	1'b0
UNSOLICIT_RSP_EN	[23]	RW	Enable reporting of an unsolicited response error. Save and lock transaction capture information in Logical/Transport Layer Device ID and Control Capture CSRs.	1'b0

Table 6–53. Logical/Transport Layer Error Enable CSR—Offset: 0x10804 (Part 2 of 2)

Field	Bits	Access	Function	Default
UNSUPPORT_TRAN_EN	[22]	RW	Enable report of an unsupported transaction error. Save and lock transaction capture information in Logical/Transport Layer Device ID and Control Capture CSRs.	1'b0
RSRV	[21-8]	RO	Reserved	14'h0
Implementation Specific error enable	[7-0]	RO	This feature is not supported.	8'b0

Table 6–54. Logical/Transport Layer Address Capture CSR—Offset: 0x10808

Field	Bits	Access	Function	Default
ADDRESS	[31:3]	RO	Bits 31 to 3 of the RapidIO address associated with the error.	29'h0
RSRV	[2]	RO	Reserved	1'b0
XAMSBS	[1:0]	RO	Extended address bits of the address associated with the error.	2'h0

Table 6–55. Logical/Transport Layer Device ID Capture CSR—Offset: 0x1080C

Field	Bits	Access	Function	Default
LARGE_DESTINATION_ID (MSB)	[31:24]	RO	Reserved if the system does not support 16-bit device ID.	8'h0
		RW	MSB of the Destination ID if the system supports 16-bit device ID.	
DESTINATION_ID	[23:16]	RO	The destination ID associated with the error.	8'h0
LARGE_SOURCE_ID (MSB)	[15:8]	RO	Reserved if the system does not support 16-bit device ID.	8'h0
		RW	MSB of the Source ID if the system supports 16-bit device ID.	
SOURCE_ID	[7:0]	RO	The source ID associated with the error.	8'h0

Table 6–56. Logical/Transport Layer Control Capture CSR—Offset: 0x10810

Field	Bits	Access	Function	Default
FTYPE	[31:28]	RO	Format type associated with the error.	4'h0
TTYPE	[27:24]	RO	Transaction type associated with the error.	4'h0
MSG_INFO	[23:16]	RO	Letter, mbox, and msgseg for the last message request received for the mailbox that had and error.	8'h0
Implementation Specific	[15:0]	RO	Reserved for this implementation.	16'h0

Doorbell Message Registers

The RapidIO IP core has registers accessible by the Avalon-MM slave port in the Doorbell module. These registers are described in the following sections.

Refer to "Doorbell Module" on page 4–56 for a detailed explanation of the DOORBELL messaging support.

Table 6–57. Doorbell Message Module Memory Map

Address	Name	Used by
Doorbell Message Space		
0x00	Rx Doorbell	External Avalon-MM master that generates or receives doorbell messages.
0x04	Rx Doorbell Status	
0x08	Tx Doorbell Control	
0x0C	Tx Doorbell	
0x10	Tx Doorbell Status	
0x14	Tx Doorbell Completion	
0x18	Tx Doorbell Completion Status	
0x1C	Tx Doorbell Status Control	
0x20	Doorbell Interrupt Enable	
0x24	Doorbell Interrupt Status	

Table 6–58. Rx Doorbell—Offset: 0x00

Field	Bits	Access	Function	Default
LARGE_SOURCE_ID (MSB)	[31:24]	RO	Reserved if the system does not support 16-bit device ID.	8'b0
			MSB of the DOORBELL message initiator device ID if the system supports 16-bit device ID.	
SOURCE_ID	[23:16]	RO	Device ID of the DOORBELL message initiator	8'b0
INFORMATION (MSB)	[15:8]	RO	Received DOORBELL message information field, MSB	8'b0
INFORMATION (LSB)	[7:0]	RO	Received DOORBELL message information field, LSB	8'b0

Table 6–59. Rx Doorbell Status—Offset: 0x04

Field	Bits	Access	Function	Default
RSRV	[31:8]	RO	Reserved	24'b0
FIFO_LEVEL	[7:0]	RO	Shows the number of available DOORBELL messages in the Rx FIFO. A maximum of 16 received messages is supported.	8'h0

Table 6–60. Tx Doorbell Control—Offset: 0x08

Field	Bits	Access	Function	Default
RSRV	[31:2]	RO	Reserved	30'h0
PRIORITY	[1:0]	RW	Request Packet's priority. 2'b11 is not a valid value for the priority field. An attempt to write 2'b11 to this field will be overwritten as 2'b10.	2'h0

Table 6–61. Tx Doorbell—Offset: 0x0C

Field	Bits	Access	Function	Default
LARGE_DESTINATION_ID (MSB)	[31:24]	RO	Reserved if the system does not support 16-bit device ID.	8'h0
		RW	MSB of the targeted RapidIO processing element device ID if the system supports 16-bit device ID.	
DESTINATION_ID	[23:16]	RW	Device ID of the targeted RapidIO processing element	8'h0
INFORMATION (MSB)	[15:8]	RW	MSB information field of the outbound DOORBELL message	8'h0
INFORMATION (LSB)	[7:0]	RW	LSB information field of the outbound DOORBELL message	8'h0

Table 6–62. Tx Doorbell Status—Offset: 0x10

Field	Bits	Access	Function	Default
RSRV	[31:24]	RO	Reserved	8'h0
PENDING	[23:16]	RO	Number of DOORBELL messages that have been transmitted, but for which a response has not been received. There can be a maximum of 16 pending DOORBELL messages.	8'h0
TX_FIFO_LEVEL	[15:8]	RO	The number of DOORBELL messages in the staging FIFO plus the number of DOORBELL messages in the Tx FIFO. The maximum value is 16.	8'h0
TXCPL_FIFO_LEVEL	[7:0]	RO	The number of available completed Tx DOORBELL messages in the Tx Completion FIFO. The FIFO can store a maximum of 16.	8'h0

Table 6–63. Tx Doorbell Completion—Offset: 0x14 (1)

Field	Bits	Access	Function	Default
LARGE_DESTINATION_ID	[31:24]	RO	Reserved if the system does not support 16-bit device ID.	8'h0
			MSB of the targeted RapidIO processing element device ID if the system supports 16-bit device ID.	
DESTINATION_ID	[23:16]	RO	The device ID of the targeted RapidIO processing element.	8'h0
INFORMATION	[15:8]	RO	MSB of the information field of an outbound DOORBELL message that has been confirmed as successful or unsuccessful.	8'h0
INFORMATION	[7:0]	RO	LSB of the information field of an outbound DOORBELL message that has been confirmed as successful or unsuccessful.	8'h0

Note to Table 6–63:

(1) The completed Tx DOORBELL message comes directly from the Tx Doorbell Completion FIFO.

Table 6–64. Tx Doorbell Completion Status—Offset: 0x18

Field	Bits	Access	Function	Default
RSRV	[31:2]	RO	Reserved	30'h0
ERROR_CODE	[1:0]	RO	This error code corresponds to the most recently read message from the Tx Doorbell Completion register. After software reads the Tx Doorbell Completion register, a read to this register should follow to determine the status of the message. 2'b00—Response DONE status 2'b01—Response with ERROR status 2'b10—Time-out error	2'h0

Table 6–65. Tx Doorbell Status Control—Offset: 0x1C

Field	Bits	Access	Function	Default
RSRV	[31:2]	RO	Reserved	30'h0
ERROR	[1]	RW	If set, outbound DOORBELL messages that received a response with ERROR status, or were timed out, are stored in the Tx Completion FIFO. Otherwise, no error reporting occurs.	1'h0
COMPLETED	[0]	RW	If set, responses to successful outbound DOORBELL messages are stored in the Tx Completion FIFO. Otherwise, these responses are discarded.18	1'h0

Table 6–66. Doorbell Interrupt Enable—Offset: 0x20

Field	Bits	Access	Function	Default
RSRV	[31:3]	RO	Reserved	29'b0
TX_CPL_OVERFLOW	[2]	RW	Tx Doorbell Completion Buffer Overflow Interrupt Enable	1'h0
TX_CPL	[1]	RW	Tx Doorbell Completion Interrupt Enable	1'h0
RX	[0]	RW	Doorbell Received Interrupt Enable	1'h0

Table 6–67. Doorbell Interrupt Status—Offset: 0x24

Field	Bits	Access	Function	Default
RSRV	[31:3]	RO	Reserved	29'h0
TX_CPL_OVERFLOW	[2]	RW1C	Interrupt asserted due to Tx Completion buffer overflow. This bit remains set until at least one entry is read from the Tx Completion FIFO. After reading at least one entry, software should clear this bit. It is not necessary to read all of the Tx Completion FIFO entries.	1'h0
TX_CPL	[1]	RW1C	Interrupt asserted due to Tx completion status	1'h0
RX	[0]	RW1C	Interrupt asserted due to received messages	1'h0

7. Testbenches

The RapidIO IP core includes two demonstration testbenches for your use. One tests a IP core variation that has only the Physical layer. The other testbench tests a IP core variation that has Physical, Transport, and Logical layers. The purpose of the supplied testbenches is to provide examples of how to parameterize the IP core and how to use the Atlantic interface in the Physical-layer-only IP core variations, and the Avalon Memory-Mapped (Avalon-MM) and Avalon Streaming (Avalon-ST) interfaces, to generate and process RapidIO transactions. The testbenches are not available for RapidIO IP cores generated in the Qsys flow.

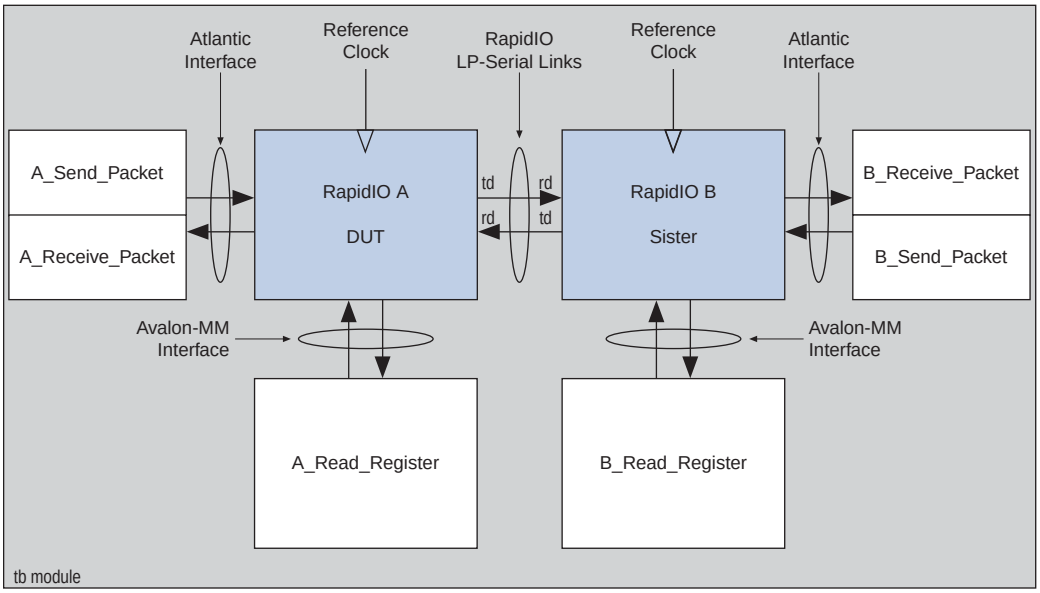
Testbench for Variations with Only a Physical Layer

The demonstration testbench that is generated for a Physical-layer-only variation demonstrates the following functions:

- Port initialization process
- Transmission, reception, and acknowledgment of packets with 8 to 256 bytes of data payload
- Support for 8-bit or 16-bit device ID fields
- Writing to and reading from the Atlantic interfaces
- Reading from the software interface registers
- Transmission and reception of multicast-event control symbols

The testbench consists of two RapidIO IP core instances interconnected through their high-speed serial interfaces, as shown in Figure 7–1. In the testbench, each IP core's `td` output is connected to the other IP core's `rd` input. The testbench module provides clocking and reset control, tasks to write to and read from the IP core's Atlantic interfaces, and a task to read from the command and status register (CSR) set. For variations with external transceivers, these IP cores are interconnected through their XGMII interfaces.

Figure 7–1. Serial RapidIO Physical Layer Demonstration Testbench (1)



Note to Figure 7–1:

(1) The external blocks, shown in white, are Verilog HDL tasks.

The testbench starts with the IP cores in a reset state. All clock inputs use a common reference clock. After coming out of the reset state, the IP cores start the port initialization process to detect the presence of a partner and establish bit synchronization and code group boundary alignment. After the IP cores assert their `port_initialized` output signals, the testbench checks that the port initialization process completed successfully by reading the `Error and Status CSR` to confirm the expected values of the `PORT_OK` and `PORT_UNINIT` register bits.

Packets with 8 to 256 bytes of data payload are then transmitted from one IP core to the other. The receiving IP core sends the proper acknowledgment symbols and the received packets are checked in the expected sequence for data integrity.

Table 7–1 describes the format of the transmitted packets.

Table 7–1. Serial Packets Format

Packet Byte	Format	Description
First Header word	`{ackID[4:0],Reserved[2:0], prio[1:0],tt[1:0], ftype[3:0]}`	`ackID` is set to zero and is replaced by the transmitting IP core. The `prio` field is used by the receiver to select the output queue. The `ttype` and `ftype` fields are used by the Transport and Logical layers and are ignored by the Physical layer IP cores, except I/O logical `MAINTENANCE` packet type.
DestinationID	`DestinationID[15:0]`	These fields are used by the Transport and Logical layers and are transferred unchanged by the Physical layer IP cores.
SourceID	`SourceID[15:0]`	
Last Header word	`{Transaction[3:0], Size[3:0],TID[7:0]}`	
Payload bytes	8–256 bytes	The payload bytes in the packet are set to an incrementing sequence starting at 0.

The received packet format is similar, but CRCs and padding (when required) are appended to the packet and an intermediate CRC is inserted in the packets after the first 80 bytes, when the packet size exceeds 80 bytes.

Table 7–2 lists the tasks used to write packets to a IP core for transmission, read and check a received packet, and read the value from a register and compare it to an expected value.

Table 7–2. Physical Layer Testbench Tasks

Function	Prototype	Comments

Write Packet to an Atlantic slave sink	`task send_packet;`
	`input [1:0] prio;`
	`input [1:0] tt;`
	`input [3:0] ftype;`
	`input [8:0] payload_sizes;`
		The `payload_size` should be an even number between 8 and 256 inclusive.
		
		The actual name of the task is prepended with A_ or B_ depending on which IP core it should act.

Read and check a packet from an Atlantic slave source	`task receive_packet;`
	`input [1:0] prio;`
	`input [1:0] tt;`
	`input [3:0] ftype;`
	`input [8:0] payload_size;`
		`prio`—packet priority
		`tt`—transport type
		`ftype`—packet format type
		`payload_size`—size of the packet payload

Read from Register	`task read_register;`
	`input [15:0]address;`
	`input [31:0]expected;`
		The read value is compared to the expected value, and any difference is flagged as an error. You can specify “`don't care`” values by putting ‘x’s in the corresponding bit position.

All of the packets are sent contiguously, in sequence. The testbench also changes the value of the `multicast_event_tx` input signal to the RapidIO IP core under test, multiple times during the test sequence, and the sister module checks that a multicast-event control symbol is sent for each transition. After all packets have been sent, the idle sequence is transmitted until the end of the simulation.

The testbench concludes by checking that all of the packets have been received. If no error is detected and all packets are received, the testbench issues a TESTBENCH PASSED message stating that the simulation was successful.

If an error is detected, a TESTBENCH FAILED message is issued to indicate that the testbench has failed. A TESTBENCH INCOMPLETE message is issued if the expected number of checks is not made. For example, this message is issued if not all packets are received before the testbench is terminated. The variable tb.exp_chk_cnt determines the number of checks done to ensure completeness of the testbench.

To generate a value change dump file called **dump.vcd** for all viewable signals, uncomment the line //`define MAKEDUMP in the <*variation name*>_**tb.v** file.

Testbench for a Variation with Physical, Transport, and Logical Layers

For a variation that includes Transport, Logical, and Physical layers, transactions are generated and monitored on the Avalon-MM interfaces and Avalon-ST interface. The Atlantic interfaces are not visible in variations with a Transport layer.

MAINTENANCE, Input/Output, or DOORBELL transactions are generated if you select the corresponding modules during parameterization of the IP core. Type 9 (Data Streaming) packets are transferred through the Avalon-ST pass-through interface, if present.

The testbench instantiates two symmetrical RapidIO IP core variations. One instance is the Device Under Test (DUT). The other instance acts as a RapidIO link partner for the RapidIO DUT module and is referred to as the sister_rio module. The sister_rio module responds to transactions initiated by the DUT and generates transactions to which the DUT responds. Bus functional models (BFM) are connected to the Avalon-MM and Avalon-ST interfaces of both the DUT and sister_rio modules, to generate transactions to which the link partner responds when appropriate, and to monitor the responses.

Figure 7–2 is a block diagram of the testbench in which all of the available Avalon-MM interfaces are enabled. The two MegaCore modules communicate with each other using the Serial RapidIO interface. The testbench initiates the following transactions at the DUT and targets them to the sister_rio module:

- SWRITE
- NWRITE_R
- NWRITE
- NREAD
- DOORBELL messages
- MAINTENANCE writes and reads
- MAINTENANCE port writes and reads
- Type 9 (Data Streaming) transactions (using the Avalon-ST interface)

Your specific variation may not have all of the interfaces enabled. If an interface is not enabled, the transactions supported by that interface are not exercised by the testbench.

In addition, the RapidIO IP core modules implement the following features:

- Multicast-event control symbol transmission and reception. The RapidIO IP core under test generates and transmits multicast-event control symbols in response to transitions on its `multicast_event_tx` input signal. The sister module checks that these control symbols arrive as expected.
- Disabled destination ID checking, or not, selected at configuration.
- NWRITE_R completion indication.
- Transaction order preservation between `DOORBELL` transactions and I/O write transactions, or not, selected at configuration. If this feature is selected, the RapidIO IP core under test generates and transmits `DOORBELL` and write transactions. The testbench checks that the transaction packets arrive on the link in the expected order.

Figure 7–2. Transport and IO Logical Layers Testbench

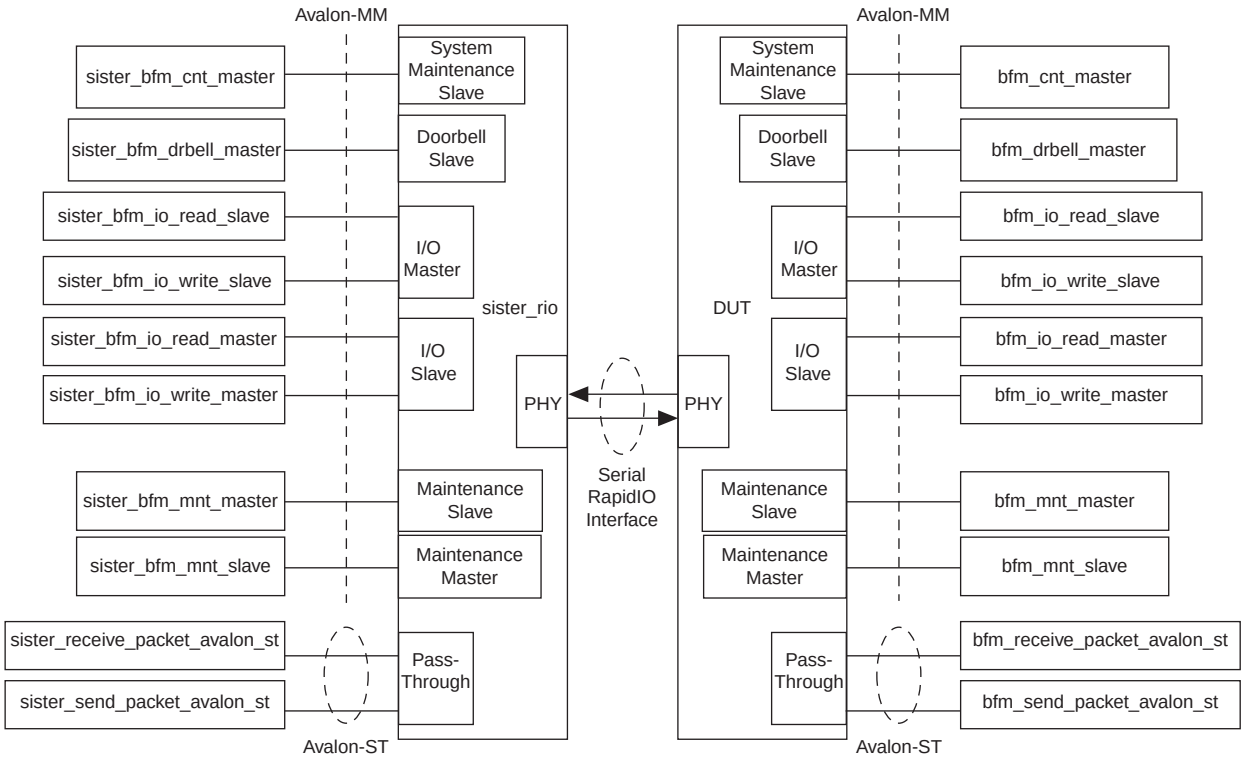


Figure 7–2 illustrates the system specified in Verilog HDL in the file <*design_name*>**_hookup.iv**. Activity across the Avalon-MM interfaces is generated and checked by running tasks that are defined in the bus functional models (BFMs). These models are implemented in the following files:

- <*design_name*>**_avalon_bfm_master.v**
- <*design_name*>**_avalon_bfm_slave.v**

The file <*design_name*>**_tb.v** implements the code that performs the test transactions. The code performs a reset and initialization sequence necessary for the DUT and sister_rio IP cores to establish a link and exchange packets.

Reset, Initialization, and Configuration

The clocks that drive the testbench are defined and generated in the <*design_name*>**_hookup.iv** file.

Refer to <*design_name*>**_hookup.iv** for the exact frequencies used for each of the clocks. The frequencies depend on the configuration of the variation.

The reset sequence is simple—the main reset signal for the DUT and the sister_rio IP core, `reset_n`, is driven low at the beginning of the simulation, is kept low for 200 ns, and is then deasserted.

After `reset_n` is deasserted, the testbench waits until both the DUT and the sister_rio modules have driven their `port_initialized` output signals high. These signal transitions indicate that both IP cores have completed their initialization sequence. The testbench then waits an additional 5000 ns, to allow time for a potential reset link-request control symbol exchange between the DUT and the sister_rio module. The testbench again waits until both the DUT and the sister_rio modules have driven their `port_initialized` output signals high. Following the 5000 ns wait, these signals indicate that the link is established and the Physical layer is ready to exchange traffic.

Next, basic programming of the internal registers is performed in the DUT and the sister_rio module. Table 7–3 shows the registers that are programmed in both the DUT and the sister_rio IP cores. For a full description of each register, refer to Chapter 6, Software Interface.

Table 7–3. Testbench Registers (Part 1 of 2)

Module	Register Address	Register Name	Description	Value
rio	0x00060	Base Device ID CSR	Program the DUT to have an 8-bit base device ID of 0xAA or a 16-bit device ID of 0xAAAA.	32'h00AA_FFFFor 32'h00FF_AAAA
rio	0x0013C	General Control CSR	Enable Request packet generation by the DUT.	32'h6000_0000
sister_rio	0x00060	Base Device ID CSR	Program the sister_rio module to have an 8-bit base device ID of 0x55 or a 16-bit device ID of 0x5555.	32'h0055_FFFF or 32'h00FF_5555
sister_rio	0x0013C	General Control CSR	Enable Request packet generation by the sister_rio module.	32'h6000_0000
rio	0x1040C	Input/Output Slave Window 0 Control	Set the DESTINATION_ID for outgoing transactions to a value 0x55 or 0x5555. The width of the DESTINATION_ID field depends on the sister_rio device ID width. This value matches the base device ID of the sister_rio module.	32'h0055_0000 or 32'h5555_0000
rio	0x10404	Input/Output Slave Window 0 Mask	Define the Input/Output Avalon-MM Slave Window 0 to cover the whole address space (mask set to all zeros) and enable it.	32'h0000_0004
sister_rio	0x10504	Input/Output Slave Interrupt Enable	Enable the I/O slave interrupts.	32'h0000_000F
sister_rio	0x10304	Input/Output Master Window 0 Mask	Enable the sister_rio I/O Master Window 0, which allows the sister_rio to receive I/O transactions.	32'h0000_0004

Table 7–3. Testbench Registers (Part 2 of 2)

Module	Register Address	Register Name	Description	Value
rio	0x1010C	TX Maintenance Window 0 Control	Set the DESTINATION_ID for outgoing MAINTENANCE packets to 0x55 or 0x5555, depending on the sister_rio device ID width. This value matches the base device ID of the sister_rio module. Set the hop count to 0xFF.	32'h0055_FF00 or 32'h5555_FF00
rio	0x10104	TX Maintenance Window 0 Mask	Enable the TX Maintenance window 0.	32'h0000_0004

Read and write tasks that are defined in the BFM instance, bfm_cnt_master, program the DUT's registers. Read and write tasks defined in the BFM instance sister_bfm_cnt_master program the sister_rio module's registers. For the exact parameters passed to these tasks, refer to the file <*design_name*>**_tb.v**. The tasks drive either a write or read transaction across the System Maintenance Avalon-MM slave interface.

In the configuration shown in Figure 7–2 on page 7–5, the IP cores can exchange basic packets across the serial link.

Maintenance Write and Read Transactions

If the Maintenance module is present, the testbench sends a few MAINTENANCE read and write request packets from the DUT to the sister_rio module. Transactions are initiated by Avalon-MM transactions on the DUT's Maintenance Avalon-MM slave interface, and are checked on the sister_rio's Maintenance Avalon-MM master interface.

The first set of tests performed are MAINTENANCE write and read requests. The DUT sends two MAINTENANCE write requests to the sister_rio module. The writes are performed by running the rw_addr_data task defined inside the BFM instance, bfm_mnt_master. The bfm_mnt_master is an instance of the module avalon_bfm_master, defined in the file <*design_name*>**_avalon_bfm_master.v**. The following parameters are passed to the task:

- 'WRITE —transaction type to be executed
- wr_address—address to be driven on the Avalon-MM address bus
- wr_data—write data to be driven on the Avalon-MM write data bus

The task performs the write transaction across the Maintenance Write Avalon-MM slave interface.

The DUT then sends two MAINTENANCE read requests to the sister_rio module. To perform the reads, run the rw_data task defined inside the BFM instance, bfm_mnt_master. The following parameters are passed to the task:

- 'READ— transaction type to be executed
- rd_address—address to be driven on the Avalon-MM address bus
- rd_data—parameter that stores the data read across the Avalon-MM read data bus

The rw_data task performs the read transaction across the Maintenance Read Avalon-MM slave interface.

The write transaction across the Avalon-MM interface is translated to a RapidIO MAINTENANCE write request packet. Similarly, the read transaction across the Avalon-MM interface is translated into a RapidIO MAINTENANCE read request packet.

The MAINTENANCE write and read request packets are received by the sister_rio module and translated to Avalon-MM transactions that are presented across the sister_rio module's Maintenance master Avalon-MM interface. An instance of avalon_bfm_slave, the BFM for an Avalon-MM slave, is driven by this interface. In the testbench, the write and read transactions are checked and data is returned for the read operation. The write operation is checked by invoking the read_writedata task of the BFM. The task returns the write address and the written data. This information is then checked for data integrity. The read operation is completed on the sister side by invoking the write_readdata task. This task returns the read address and drives the return data and read control signals on the Avalon-MM master read port of the sister_rio module. The read data is checked after it is received by the DUT.

SWRITE Transactions

The next set of operations performed are Streaming Writes (SWRITE). To perform SWRITE operations, one register in the IP core must be reconfigured as shown in Table 7–4.

Table 7–4. SWRITE Register

Module	Register Address	Name	Value	Description
rio	0x1040C	Input/Output Slave Mapping Window 0 Control	32'h0055_0002 or 32'h5555_0002	Sets the DESTINATION_ID for outgoing transactions to the value 0x55 or 0x5555, depending on the device ID width of the sister_rio. This value matches the base device ID of the sister_rio module. Enables SWRITE operations.

With the setting in Table 7–4, any write operation presented across the Input/Output Avalon-MM slave interface on the rio module is translated to a RapidIO Streaming Write transaction.

The Avalon-MM write address must map into Input/Output Slave Window 0. However, in this example the window is set to cover the entire Avalon-MM address space by setting the mask to all zeros.

The testbench generates a predetermined series of burst writes across the Avalon-MM slave I/O interface on the DUT. These write bursts are each converted to an SWRITE request packet sent on the RapidIO serial interface. Because Streaming Writes only support bursts that are multiples of a double word (multiple of 8 bytes), the testbench cycles from 8 to MAX_WRITTEN_BYTES in steps of 8 bytes. Two tasks carry out the burst writes, rw_addr_data and rw_data. The rw_addr_data task initiates the burst by providing the address, burstcount, and the content of the first data word, and the rw_data task completes the remainder of the burst.

At the sister_rio module, the SWRITE request packets are received and translated into Avalon-MM transactions that are presented across the Input/Output master Avalon-MM interface. The testbench calls the task read_writedata of the sister_bfm_io_write_slave. The task captures the written data.

The written data is then checked against the expected value by running an `expect` task. After completing the `SWRITE` tests, the testbench performs `NWRITE_R` operations.

NWRITE_R Transactions

To perform `NWRITE_R` operations, one register in the IP core must be reconfigured as shown in Table 7–5.

Table 7–5. NWRITE_R Transactions

Module	Register Address	Name	Value	Description
`rio`	`0x1040C`	`Input/Output Slave Mapping Window 0 Control`	`32'h0055_0001` or `32'h5555_0001`	Sets the `DESTINATION_ID` for outgoing transactions to the value `0x55` or `0x5555`, depending on the device ID width of the sister_rio. This value matches the base device ID of the sister_rio module. Enables `NWRITE_R` operations.

With the setting in Table 7–5, any write operation presented across the Input/Output Avalon-MM slave module's Avalon-MM write interface is translated to a RapidIO `NWRITE_R` transaction. The Avalon-MM write address must map to the range specified for the I/O Slave window 0.

To initialize testing of the new NWRITE_R completion indication feature, the test first checks that the `PENDING_NWRITE_RS` field of the `Input/Output Slave Pending NWRITE_R Transactions` register has value 0, that the `NWRITE_RS_COMPLETED` field of the `Input/Output Slave Interrupt Enable` register is set, and that the `NWRITE_RS_COMPLETED` field of the `Input/Output Slave Interrupt` register is clear, before setting the `Input/Output Slave Mapping Window 0 Control` register and starting the sequence of NWRITE_R transactions.

Initially, the testbench performs two single word transfers, writing to an even word address first and then to an odd word address. The testbench then generates a predetermined series of burst writes across the Input/Output Avalon-MM slave module's Avalon-MM write interface on the DUT. These write bursts are each converted into `NWRITE_R` request packets sent over the RapidIO Serial interface. The testbench cycles from 8 to `MAX_WRITTEN_BYTES` in steps of 8 bytes. Two tasks are invoked to carry out the burst writes, `rw_addr_data` and `rw_data`. The `rw_addr_data` task initiates the burst and the `rw_data` task completes the burst.

At the sister_rio module, the `NWRITE_R` request packets are received and presented across the I/O master Avalon-MM interface as write transactions. The testbench calls the `read_writedata` task of the sister_bfm_io_write_slave module. The task captures the written data. The written data is checked against the expected value.

In addition, the test checks that the `NWRITE_RS_COMPLETED` interrupt field of the `Input/Output Slave Interrupt` register is set, then clears the field, and checks again to confirm the field was cleared correctly.

NWRITE Transactions

To perform NWRITE operations, one register in the IP core must be reconfigured as shown in Table 7–6. With these settings, any write operation presented across the Input/Output Avalon-MM slave interface is translated into a RapidIO NWRITE transaction.

Table 7–6. NWRITE Transactions

Module	Register Address	Name	Value	Description
rio	0x1040C	Input/Output Slave Mapping Window 0 Control	32'h0055_0000or 32'h5555_0000	Sets the DESTINATION_ID for outgoing transactions to the value 0x55 or 0x5555, depending on the device ID width of the sister_rio. This value matches the base device ID of the sister_rio. Sets the write request type back to NWRITE.

Initially, the testbench performs two single word transfers, writing to an even word address first and then to an odd word address. The testbench then generates a predetermined series of burst writes across the Input/Output Avalon-MM slave module's Avalon-MM write interface on the DUT. These write bursts are each converted into an NWRITE request packet that is sent over the RapidIO serial interface. The testbench cycles from 8 to MAX_WRITTEN_BYTES in steps of 8 bytes. Two tasks are run to carry out the burst writes, rw_addr_data and rw_data. The rw_addr_data task initiates the burst and the rw_data task completes the remainder of the burst.

The sister_rio module receives the NWRITE request packets and presents them across the I/O master Avalon-MM slave interface as write transactions. The testbench calls the read_writedata task of the sister_bfm_io_write_slave module. The task captures the written data. The written data is checked against the expected value.

NREAD Transactions

The next set of transactions tested are NREADs. The DUT sends a group of NREAD transactions to the sister_rio module by cycling the read burst size from 8 to MAX_READ_BYTES in increments of 8 bytes. For each iteration, the rw_addr_data task is called. This task is defined in the bfm_io_read_master instance of the Avalon-MM master BFM. The task performs the read request packets across the I/O Avalon-MM Slave Read interface. The read transaction across the Avalon-MM interface is translated into a RapidIO NREAD request packets. The values of the rd_address, rd_byteenable, and rd_burstcount parameters determine the values for the rdsize, wdptr and xamsbs fields in the header of the RapidIO packet.

The NREAD request packets are received by the DUT and are translated into Avalon-MM read transactions that are presented across the sister_rio module's I/O master Avalon-MM interface. An instance of avalon_bfm_slave, the BFM for an Avalon-MM slave, is driven by this interface. The read operations are checked and data is returned by calling the task, write_readdata. This task drives the data and read datavalid control signals on the Avalon-MM master read port of the DUT.

The returned data is expected at the DUT's I/O Avalon-MM slave interface. The rw_data task is called and it captures the read data. This task is defined inside the instance of bfm_io_read_master. The read data and the expected value are then compared to ensure that they are equal.

Doorbell Transactions

To test DOORBELL messages, the doorbell interrupts must be enabled. To enable interrupts, the testbench sets the lower three bits in the Doorbell Interrupt Enable register located at address 0x0000_0020. The test also programs the DUT to store all of the successful and unsuccessful DOORBELL messages in the Tx Completion FIFO.

For more information, refer to Table 6–65 on page 6–29.

Next, the test pushes eight DOORBELL messages to the transmit DOORBELL Message FIFO of the DUT. The test increments the message payload for each transaction, which occurs when the rw_addr_data task (defined in the bfm_drbell_s_master instance) is invoked with a 'WRITE operation to the TX doorbell register at offset 0x0000_000C. This action programs the 16-bit message, an incrementing payload in this example, as well as the DESTINATION_ID—0x55 for an 8-bit device ID or 0x5555 for a 16-bit device ID—which is used in the DOORBELL transaction packet.

To verify that the DOORBELL request packets have been sent, the test waits for the drbell_s_irq signal to be asserted. The test then reads the Tx Doorbell Completion register (refer to Table 6–63 on page 6–28). This register provides the DOORBELL messages that have been added to the Tx Completion FIFO. Eight successfully completed DOORBELL messages should appear in that FIFO and each one should be accessible by reading the Tx Doorbell Completion register eight times in succession. To perform this verification, the test invokes the rw_data task defined in the instance bfm_drbell_s_master.

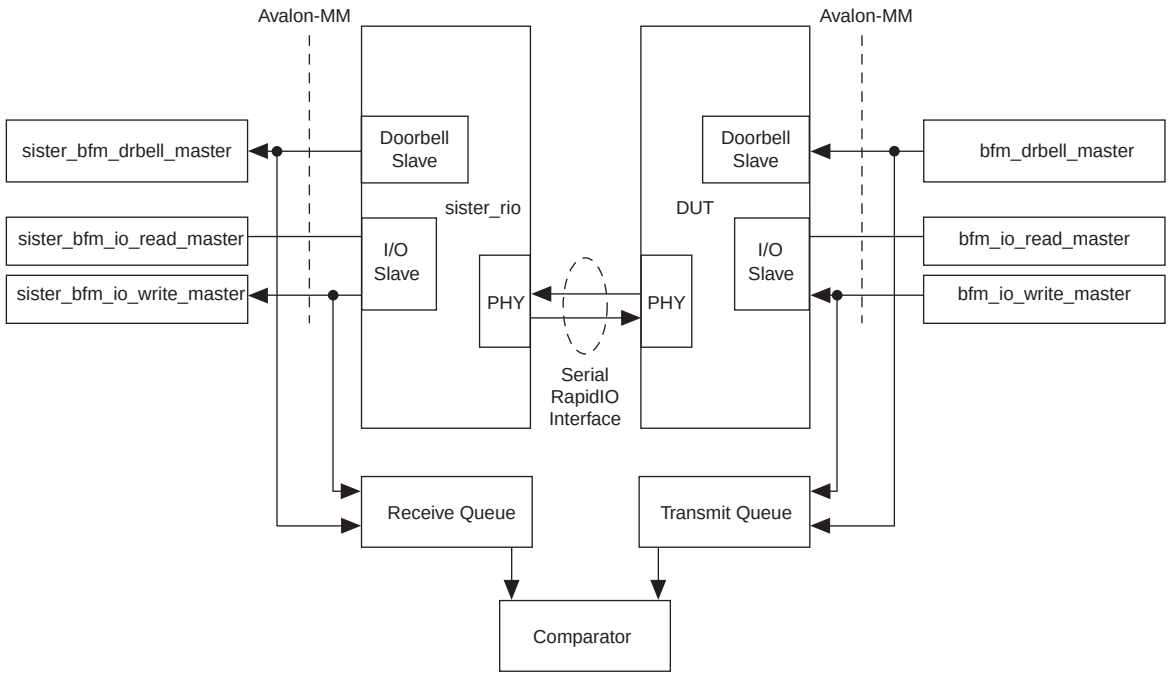
If you created the DUT with Doorbell Rx enable turned on and with Doorbell Tx enable turned off, the doorbell test programs the sister_rio module to send eight DOORBELL messages to the DUT. The test verifies that all eight DOORBELL messages were received by the DUT. The test calls the rw_addr_data task defined in the instance sister_bfm_drbell_s_master. The task performs a write to the Tx Doorbell register (refer to Table 6–61 on page 6–28). It programs the payload to be incrementing, starting at 0x0C01, and the DESTINATION_ID to have value 0xAA or 0xAAAA, matching the device ID of the DUT.

The test waits for the DUT to assert the drbell_s_irq signal, which indicates that a DOORBELL message has been received. The test then reads the eight received DOORBELL messages, by calling the rw_data task with a 'READ operation to the Rx DOORBELL register at offset 0x0000_0000. The task is called eight times, once for each message. It returns the received DOORBELL message and the message is checked for an incrementing payload starting at 0x0C01 and for the sourceId value 0x55 or 0x5555, the device ID of the sister RapidIO IP core variation.

Doorbell and Write Transactions With Transaction Order Preservation

Figure 7–3 shows the testbench for checking transaction order preservation. The test generates write transactions and Doorbell messages, and compares the transaction order before and after the transactions are transmitted on the RapidIO link. If a Doorbell module and I/O slave port are instantiated in the DUT, and **Prevent doorbell messages from passing write transactions** is turned on in the RapidIO parameter editor, the extra hardware is generated in the testbench.

Figure 7–3. Transaction Order Preservation Testbench



The transaction ordering test has two parts. The first part checks that transaction order is preserved among I/O write requests and Doorbell messages. The second part injects errors in the write transactions to force transaction cancellation, to test the integrity of the `COMPLETED_OR_CANCELLED_WRITES` field of the `Input/Output Slave RapidIO Write Requests` register. Because the behavior of the write transactions themselves is not under test, but only the preservation of transaction ordering between Doorbell messages and write requests, each part of the transaction ordering test generates only one type of write transaction.

In the first part of this test, the bfm_drbell_master sends a Doorbell message one clock cycle after the bfm_io_write_master sends a write request. Write requests are sent and checked according to the test sequence described in "SWRITE Transactions" on page 7–8, and Doorbell messages are sent and checked according to the test sequence described in "Doorbell Transactions" on page 7–11. The additional hardware shown in Figure 7–3 is used to compare the transaction order before and after transmission on the RapidIO link. Each queue has 40 bits of FIFO data. In each queue, the current entry is set to 0 for a write request and to 1 for a Doorbell message. The comparator compares bit by bit, checking for an exact match.

In the second part of this test, the DUT asserts an invalid byteenable value on the I/O slave port for a single NWRITE_R transaction, and then transmits 32 NWRITE_R transactions with a target address set out of bounds. After the bfm_io_write_master initiates the sequence of NWRITE_R transactions, the bfm_drbell_master generates transactions as in "Doorbell Transactions" on page 7–11. Each Doorbell transaction is sent to the DUT immediately following a different NWRITE_R transaction. In addition to checking for data integrity and for transaction order preservation despite the tracking complication of cancelled transactions, the testbench checks that the `I/O Slave Interrupt` register reflects each cancelled transaction correctly.

Port-Write Transactions

To test port-writes, the test performs some basic configuration of the port-write registers in the DUT and the sister_rio module. It then programs the DUT to transmit port-write request packets to the sister_rio module. The port-writes are received by the sister_rio module and retrieved by the test program.

The configuration enables the `Rx packet stored` interrupt in the sister_rio module. With the interrupt enabled, the sister_rio module asserts the `sister_sys_mnt_s_irq` signal, which indicates that an interrupt is set in either the `Maintenance Interrupt` register or the `Input/Output Slave Interrupt` register. Because this part of the testbench is testing port writes, the assertion of `sister_sys_mnt_s_irq` means that a Port-Write transaction has been received and that the payload can be retrieved. To enable the interrupt, call the task `rw_addr_data` defined in the sister_bfm_cnt_master module.

A write operation is performed by the task with the address `0x10084` and data `0x10` passed as parameters. In addition, the sister_rio module must be enabled to receive Port-Write transactions from the DUT. The task is called with the address `0x10250` and data `0x1`.

After the configuration is complete, the test performs the operations listed in Table 7–7.

Table 7–7. Port-Write Test

Operation	Action
Places data into the `TX_PORT_WRITE_BUFFER`	Write incrementing payload to registers at addresses `0x10210` to `0x1024C`
Indicates to the DUT that Port-Write data is ready	Write `DESTINATION_ID` = `0x55` or `0x5555`, depending on the device ID width setting, and `PACKET_READY` = `0x1` to `0x10200`
Waits for the sister_rio module to receive the port-write	Monitor `sister_sys_mnt_s_irq`
Verifies that the sister _rio module has the interrupt bit `PACKET_STORED` set	Read register at address `0x10080`
Retrieves the Port-Write payload from the sister_rio module and checks for data integrity	Read registers at addresses `0x10260–0x1029C`
Checks the sister_rio module `Rx Port Write Status` register for correct payload size	Read register at address `0x10254`
Clears the `PACKET_STORED` interrupt in the sister_rio module	Write `1` to bit 4 of register at address `0x10080`
Waits for the next interrupt at the sister _rio module	Monitor `sister_sys_mnt_s_irq`

The test iterates through these operations, each time incrementing the payload of the port write. The loop exits when the `max payload` for a port-write has been transmitted, 64 bytes.

All of the operations in the loop are executed by running the `rw_addr_data` task either in the bfm_cnt_master or the sister_bfm_cnt_master instances.

Transactions Across the Avalon-ST Pass-Through Interface

The demonstration testbench tests the Avalon-ST pass-through interface by exchanging Type 9 (Data Streaming) traffic between the DUT and the sister_rio module. The testbench tests this interface in a similar manner to variations that include only the Physical layer.

For details about testing the Physical interface, refer to"Testbench for Variations with Only a Physical Layer" on page 7–1.

8. SOPC Builder Design Example

The design example in this chapter shows you how to use SOPC Builder to build a system that combines a RapidIO IP core with other SOPC Builder components. SOPC Builder automatically generates HDL files that include all the specified components and interconnections. The resulting HDL files are ready to be compiled in the Quartus II software for programming an Altera device. SOPC Builder also generates a Verilog HDL simulation testbench module that performs basic transactions.

When you specify VHDL as your SOPC Builder language, only a link loopback module simulation testbench is generated for this MegaCore function.

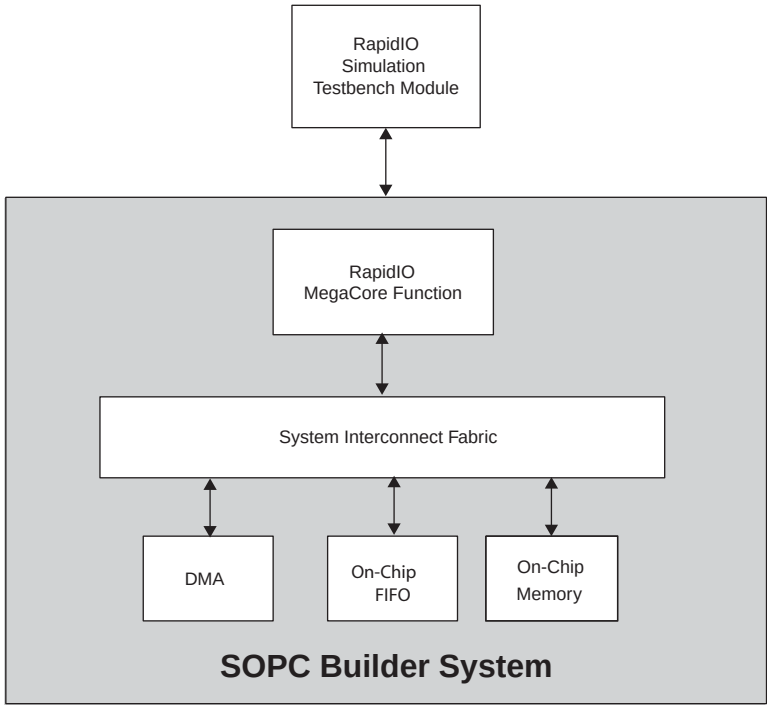
For more information about the system interconnect fabric, refer to the *System Interconnect Fabric for Memory-Mapped Interfaces* chapter in the *SOPC Builder User Guide*. For more information about SOPC Builder, refer to the *SOPC Builder Features* and *Building Systems with SOPC Builder* sections in the *SOPC Builder User Guide*.

The design example explains how to use SOPC Builder and the Quartus II software to generate a system containing the following components:

- RapidIO IP core
- DMA Controller
- On-Chip Memory
- On-Chip FIFO

Figure 8–1 shows a block diagram of the system you create in this chapter.

Figure 8–1. Example SOPC Builder System



In this chapter you create a design example by following these steps:

1. Creating a New Quartus II Project
2. Running SOPC Builder
3. Adding and Parameterizing the RapidIO Component
4. Adding and Connecting Other System Components
   a. Adding the DMA Controller
   b. Adding the On-Chip Memory
   c. Adding the On-Chip FIFO Memory
   d. Assigning Addresses and Setting the Clock Frequency
5. Generating the System
6. Simulating the System
7. Compiling and Programming the Device

After you compile your design, you can program your target Altera device and verify your design in hardware using the OpenCore Plus hardware evaluation feature or a full license.

This design example does not use all available parameters and options.

For more information about specific parameters used in this design example, refer to Chapter 3, Parameter Settings.

Creating a New Quartus II Project

You must create a new Quartus II project. You can create the project with the New Project Wizard, which helps you specify the working directory for the project, assign the project name, and designate the name of the top-level design entity. To create a new project, follow these steps:

1. On the Windows start menu, click **Programs > Altera > Quartus II** <*version*> **> Quartus II** <*version*> to run the Quartus II software.
2. On the File menu, click **New Project Wizard**. If you did not turn it off previously, the **New Project Wizard: Introduction** page appears.
3. On the **New Project Wizard: Introduction** page, click **Next**.
4. On the **New Project Wizard: Directory, Name, Top-Level Entity** page, enter the following information:
   a. Specify the working directory for your project. This design example uses the directory **C:\altera\project_rio\rapidio_sopc**.
   b. Specify the name of the project. This design example uses **rio_sys**. You must specify the same name for both the project and the top-level design entity.

   ☞ The Quartus II software specifies a top-level design entity that has the same name as the project automatically. Do not change this name.
5. Click **Next** to display the **Add Files** page.

   ☞ Click **Yes**, if prompted, to create a new directory.
6. Click **Next** to display the **Family and Device Settings** page.
7. On the **Family and Device Settings** page, select the following target device family and options:
   a. In the **Family** list, select **Stratix II GX**.

   ☞ This design example creates a design targeting the Stratix II GX device family. You can also use these procedures for other supported device families.

   b. In the **Target device** box, select **Specific device selected in 'Available devices' list**.
   c. In the **Available devices** list, select **EP2SGX90EF1152C3**.
8. Click **Next** to display the **EDA Tool Settings** page.
9. Click **Next** to display the **Summary** page.
10. Check the **Summary** page to ensure that you have entered all the information correctly.
11. Click **Finish** to complete the Quartus II project.

Running SOPC Builder

To run SOPC Builder to build your system, follow these steps:

1. On the Tools menu, click **SOPC Builder**.

For more information about how to use SOPC Builder, refer to Quartus II Help.

2. In the **System Name** box, type `rio_sys` as the project's top-level system name.

If you have an existing SOPC Builder system, on the File menu, click **New System** to display the **System Name** box.

3. Under **Target HDL**, select **Verilog**.
4. Click **OK.**

Although this design example requires the Verilog HDL target output, you can alternatively select VHDL for a project of your own.

Adding and Parameterizing the RapidIO Component

To instantiate and parameterize the RapidIO MegaCore component in your system, follow these steps:

1. In the Component Library, in the search box, type `RapidIO`.
2. Highlight **RapidIO** and click **Add**. The RapidIO component is added to the system, and the RapidIO parameter editor appears.
3. To parameterize your IP core, follow these steps:
   a. On the **Physical Layer** page, specify the parameters in Table 8–1.

Table 8–1. Set Physical Layer Options I

Option	Value	Comment
Mode Selection	4x Serial	
Transceiver Selection	Stratix II GX PHY	

b. Click **Configure Transceiver** to display the transceiver parameter editor.

c. On the **Tx Analog** page, for **What is the transmitter buffer power (VCCH)?**, select **1.5** V.

d. Click **Finish**.

e. Click **Finish**.

f. In the RapidIO parameter editor, on the Physical Layer page, specify the parameters in Table 8–2 and leave all other parameters at their default values.

Table 8–2. Set Physical Layer Options II

Option	Value	Comment
Baud rate	**2500** Mbaud	Default value.
Receive Priority Retry Threshold	Turn on **Auto-configured from receiver buffer size**	Default value. Receiver priority retry thresholds are expressed in terms of 64-byte buffers. Each maximum size packet requires five buffers.

g. Click **Next** to display the **Transport and Maintenance** page.

h. On the **Transport and Maintenance** page, set the parameters in Table 8–3.

Table 8–3. Set Transport Layer Options

Option	Value	Comment
Enable Avalon-ST pass-through interface	Turn on this option	For SOPC Builder, the Transport layer is always enabled; but you must turn on the **Enable Avalon-ST pass-through interface**.
Maintenance logical layer interface(s)	**Avalon-MM Master**	

i. Click **Next** to display the **I/O and Doorbell** page.

j. On the **I/O and Doorbell** page, set the parameters in Table 8–4.

Table 8–4. Set I/O and Doorbell Options

Option	Value	Comment
I/O logical layer interfaces	**Avalon-MM Master and Slave**	
I/O slave address width	**25**	The **Input/Output Slave address width** is set to **30** by default. However, to avoid over-allocating Avalon-MM memory space it must be set to a smaller value in this example design.
I/O read and write order preservation	**Off**	
Number of Rx address translation windows	**1**	
Number of Tx address translation windows	**1**	
Doorbell Tx enable and Doorbell Rx enable	Turn off these options.	Default value. When DOORBELL messaging is turned on, a 32-bit Avalon slave port enables DOORBELL messaging from the user application to the IP core. Turning off DOORBELL messaging reduces resource usage and may be desirable for some applications.

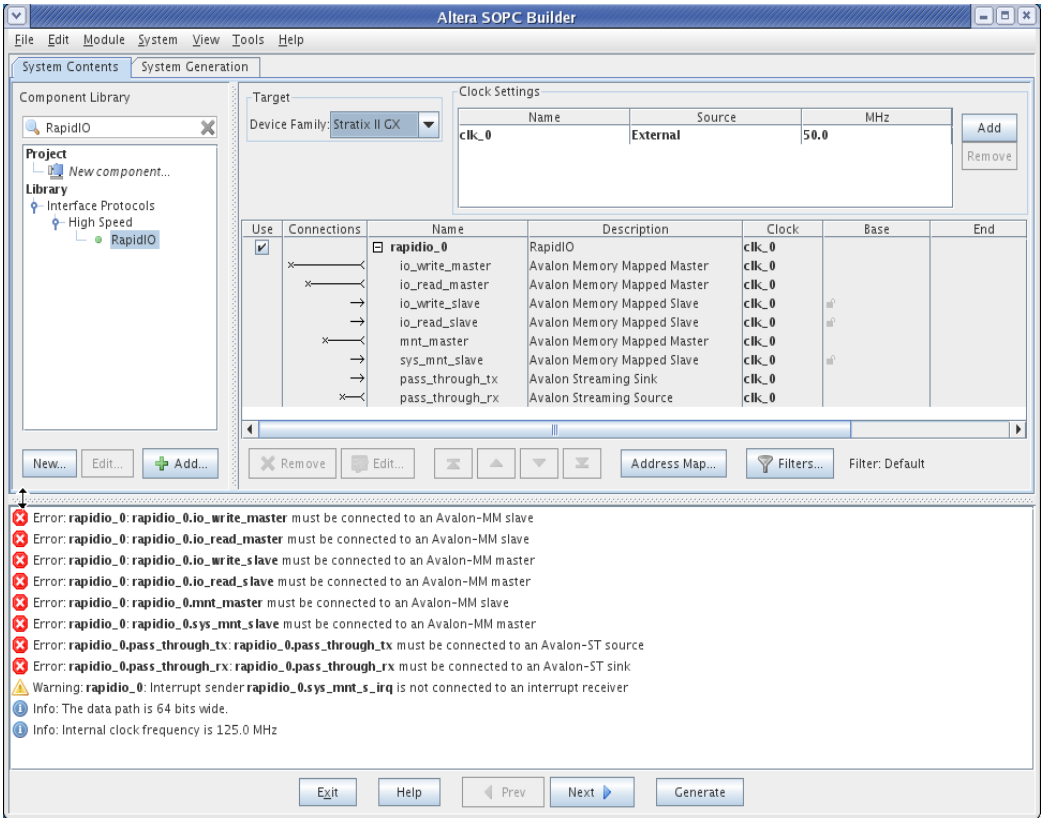
k. Click **Next** to display the **Capability Registers** page. You can set the Device Register IDs to match your system. Unless your design includes an additional extended feature block, keep the **Extended features pointer** default value of **0x0100**. You can keep the default values for all other parameters.

l. Under **Data Messages**, make sure both options are turned off.

m. Click **Finish** to complete parameterization and add the RapidIO IP core to the SOPC Builder system

After adding the RapidIO IP core component to your system, various Avalon-MM ports are created and shown as connection points in the **System Contents** tab. Error messages indicate that these ports are not connected, as shown in Figure 8–2.

Figure 8–2. RapidIO IP core Added and Avalon-MM Ports Created



These errors are resolved as you add the remaining components to your system and make all of the appropriate connections, as described in the following sections.

The default instance name of the RapidIO IP core component is **rapidio_0**. You can change the default name by right-clicking on the component name and then clicking **Rename**. The component name must be unique; it cannot be the same name as the system name.

Adding and Connecting Other System Components

To complete your system, you add and connect the following components, assign addresses, and set the clock frequency:

- DMA Controller
- On-Chip Memory
- On-Chip FIFO Memory

Adding the DMA Controller

To add a DMA controller to your system, follow these steps:

1. In the Component Library, in the search box, type `DMA Controller`.
2. Highlight **DMA Controller** and click **Add**. The DMA Controller component is added to the system, and the DMA Controller parameter editor appears.
3. On the **DMA Parameters** tab, turn on **Enable burst transfers** and select **64** as the **Maximum burst size**.
4. Click **Finish** to retain default settings for other parameters and add the DMA controller to your SOPC Builder system.

Adding the On-Chip Memory

To add on-chip memory to your system, follow these steps:

1. In the Component Library, in the search box, type `Memory (RAM or ROM)`.
2. Highlight **On-Chip Memory (RAM or ROM)** and click **Add**. The On-Chip Memory component is added to your system, and the On-Chip Memory parameter editor appears.
3. Select **64** as the **Data width**.
4. Click **Finish** to retain default settings for other parameters and add the On-Chip Memory to your SOPC Builder system.

Adding the On-Chip FIFO Memory

To add on-chip FIFO memory to your system, follow these steps:

1. In the Component Library, in the search box, type `FIFO Memory`.
2. Highlight **On-Chip FIFO Memory** and click **Add**. The On-Chip FIFO Memory component is added to your system, and the On-Chip FIFO Memory parameter editor appears.
3. Set the options in Table 8–5.

Table 8–5. Set Interface Parameter Options (Part 1 of 2)

Option	Value
Depth	**64**
Allow backpressure	On
Create status interface for input	Off
Create status interface for output	Off

Table 8–5. Set Interface Parameter Options (Part 2 of 2)

Option	Value
Input type	**AvalonST_SINK**
Output type	**AvalonST_SOURCE**
Avalon-ST port settings	
Bits per symbol	8 bits
Symbols per beat	8 symbols
Error width	1 bit
Channel width	0 bits
Enable packet data	On

4. Click **Finish** to retain default settings for other parameters and add the On-Chip FIFO Memory to your SOPC Builder system.

Connecting the Clocks and the System Components

You must now connect any unconnected clocks and other components in your system. For the external RapidIO processing elements to access the internal registers of the RapidIO variation, your system must meet the following criteria:

- The `Maintenance Master` port must be connected to the `System Maintenance Slave` port.
- The `System Maintenance Slave port Base address` must be assigned to address `0x0`.

Displaying Clock Information and Connecting Unconnected Clocks

By default, clock information is not displayed. The **Clock** column appears, but the clock input ports of the components are not displayed. To display the missing clock information, follow these steps:

1. On the **System Contents** tab, click **Filters**. The **Filters** dialog box appears.
2. In the **Filter** list, select **All Interfaces**.

   Information about the clocks in the system appears in the **Connections**, **Module Name**, **Description**, and **Clock** columns.
3. Close the **Filters** dialog box.
4. Connect all clocks designated as ***unconnected*** in the **Clock** column. Click ***unconnected*** in the **Clock** column to assign the clock to `clk_0`.

   For Arria GX, Arria II GX, Arria II GZ, Stratix II GX, and Stratix IV GX designs, you must ensure that you also connect the calibration clock (`cal_blk_clk`) to a clock with the appropriate frequency range 10–125 MHz. In this example, the default external clock, `clk_0`, is in this range.

Connecting System Components

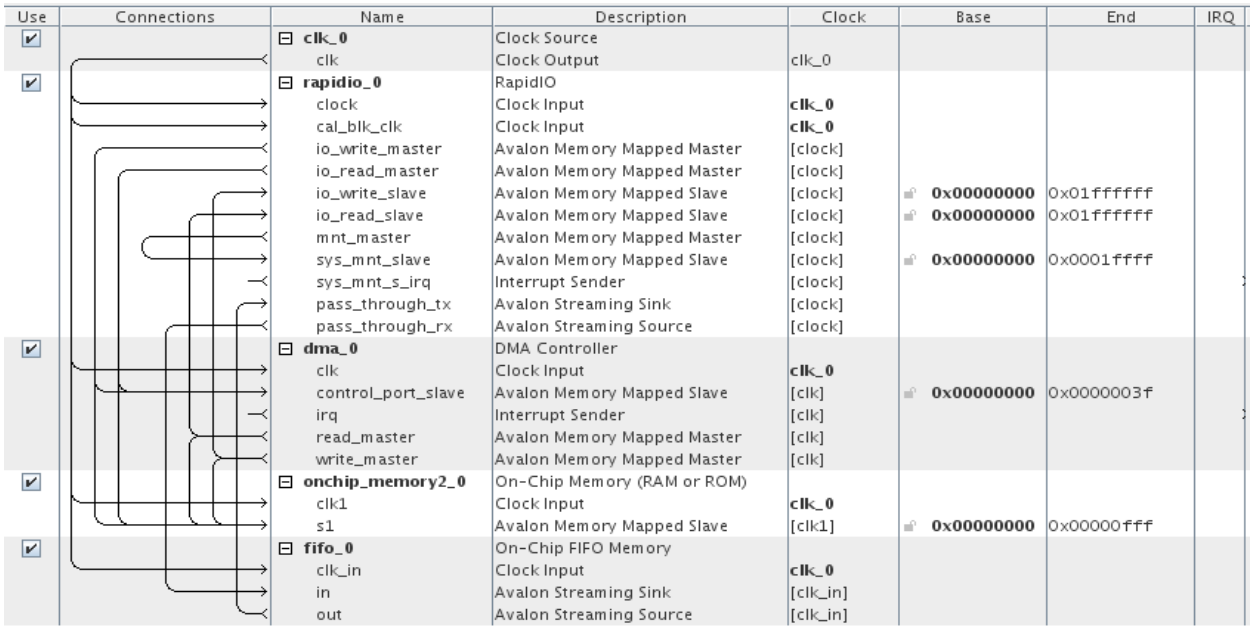
In SOPC Builder, clicking and hovering the mouse over the **Connections** column displays the potential connection points between components, represented as dots connecting wires. A filled dot shows that a connection is made; an open dot shows a potential connection point that is not currently connected. Clicking a dot toggles the connection status. To complete this design, create the connections listed in Table 8–6.

Table 8–6. Connect System Components

Make Connection From	To
rapidio_0 mnt_master	rapidio_0 sys_mnt_slave
rapidio_0 io_read_master	onchip_mem... s1
rapidio_0 io_write_master	onchip_mem... s1
rapidio_0 io_read_master	dma_0 control_port_slave
rapidio_0 io_write_master	dma_0 control_port_slave
dma_0 read_master	rapidio_0 io_read_slave
dma_0 write_master	rapidio_0 io_write_slave
dma_0 read_master	onchip_mem... s1
dma_0 write_master	onchip_mem... s1
rapidio_0 pass_through_tx	fifo_0 out
rapidio_0 pass_through_rx	fifo_0 in

Refer to Figure 8–3 to ensure that you connected the ports correctly.

Figure 8–3. Complete System Connections



As described in "Reset for RapidIO IP Cores with Physical, Transport, and Logical Layers" on page 4–10, the circuitry necessary to ensure the correct behavior of the `reset_n` input signal to the RapidIO IP core is created automatically by SOPC Builder. For this design example, you do not implement the logic described in Figure 4–4, because SOPC Builder implements it for you.

Assigning Addresses and Setting the Clock Frequency

To assign a specific address, follow these steps:

1. Click on the address that you want to change in the **Base** column, then type the address that you want to assign. Make the address assignments specified in Table 8–7.

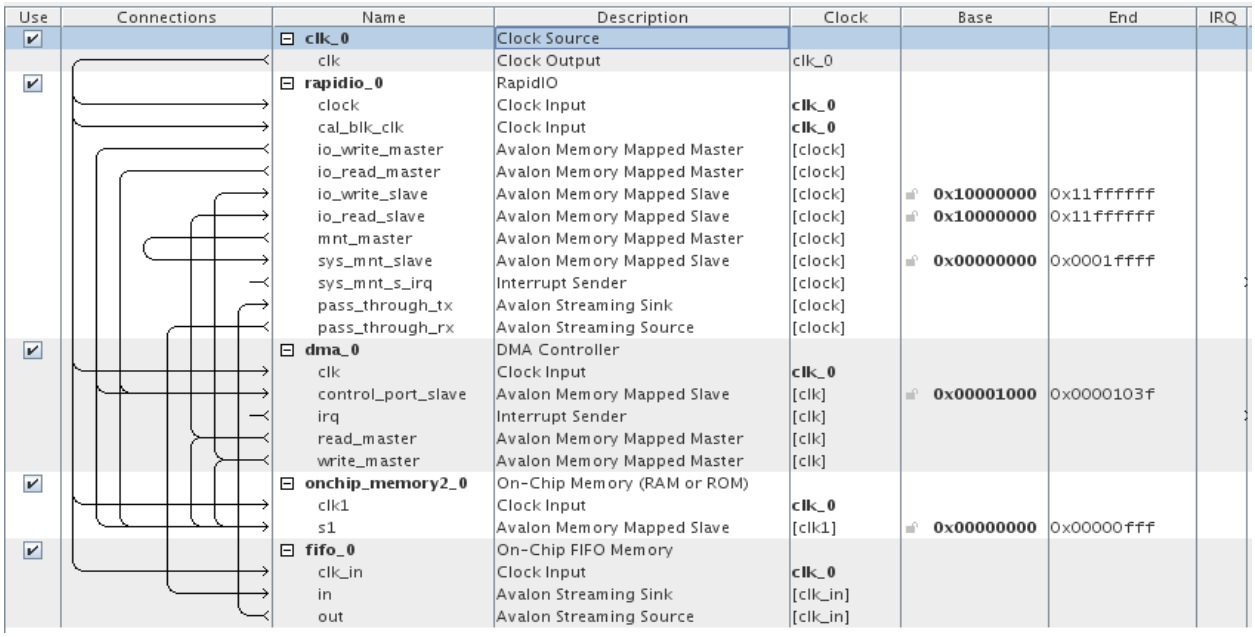
Table 8–7. Assign Addresses

Port Name	Base Address
rapidio_0 sys_mnt_slave	0x00000000
rapidio_0 io_read_slave	0x10000000
rapidio_0 io_write_slave	0x10000000
dma_0 control_port_slave	0x00001000
onchip_mem... s1	0x00000000

2. In the **Clock Settings** box, highlight **clk_0**, double-click **50.0** in the MHz column, type 125 for the external clock source `clk_0`, and press ↵ .
3. On the File menu, click **Save** and type `rio_sys` to save the SOPC Builder system in the **rio_sys.sopc** file.

Figure 8–4 shows the completed SOPC Builder system.

Figure 8–4. Complete SOPC Builder Example System



Generating the System

After you create your system with all the required components and connections and you have resolved any errors, generate the system by following these steps:

1. Click the **System Generation** tab.
2. Turn on **Simulation**. This setting enables the generation of the testbench and simulation model files for your SOPC Builder system.
3. Click **Generate** to start the generation process.

   If you are prompted to save your changes to **rio_sys.sopc**, click **Save**.

   Generating the system files, the simulation models, and the environment takes a few minutes.

   When the SOPC Builder system is generated successfully, the system HDL files are added to your project directory and are ready to be simulated and compiled with the Quartus II software.
4. After your SOPC Builder system generates successfully, click **Exit** to close SOPC Builder.

Simulating the System

The RapidIO IP core includes both a Verilog HDL testbench and a VHDL link loopback module. The steps in this section describe the Verilog HDL testbench.

The Verilog HDL testbench provided with the RapidIO IP core has two modes of operation:

- A generic mode that works with all RapidIO IP core variations and all SOPC Builder systems
- A special SOPC Builder design example mode that only works with the variation and SOPC Builder system described in this SOPC Builder design example, but which takes advantage of the various SOPC Builder components of the design example

To simulate your system with the sample Verilog HDL testbench, follow these steps:

1. The generic mode is enabled by default. To enable the special SOPC Builder design example mode, you must edit one file. In your project directory, open the **rapidio_0_sopc_tb.v** file in a text editor, search for the `SOPC_EXAMPLE_DESIGN` parameter, and change the value from `0` to `1`.
2. Start the ModelSim software. On the File menu, change directory to the **C:\altera\project_rio\rapidio_sopc\rio_sys_sim** directory.
3. Type the following command at the ModelSim command prompt:

   ```
   do setup_sim.do ↵
   ```
4. To compile all the files and load the design, type the following command at the ModelSim prompt:

   ```
   s ↵
   ```

5. To simulate the design, type the following command at the ModelSim prompt:

```
run -all ↵
```

When the special SOPC Builder Design Example mode is enabled, the RapidIO sample testbench performs the following transactions:

- Sends a sequence of read requests to the internal registers of the IP core
- Sets up the address translation register within the MegaCore for `MAINTENANCE` and `I/O` transactions
- Programs the DMA transfer data between the test module and on-chip memory
- Verifies data integrity

When simulation finishes, on the File menu, click **Quit** to close the ModelSim software and return to the Quartus II software to compile your system.

Compiling and Programming the Device

The SOPC Builder generated HDL system files are now ready for compilation in the Quartus II software, which generates an SRAM Object File (**.sof**) for device programming. To compile your system design in the Quartus II software, follow these steps:

1. Open the Quartus II project created in the "Creating a New Quartus II Project" on page 8–3.
2. On the View menu, point to **Utility Windows** and click **Tcl Console**.
3. Run the generated Tcl script at the Tcl command prompt, by typing the following command:

```
source rapidio_0_constraints.tcl ↵
```

4. Add the Rapid IO constraints to your project by typing the following command in the Quartus II Tcl Console window:

```
add_rio_constraints -ref_clk_name clk_rapidio_0 \
                    -sys_clk_name clk_0 ↵
```

The **rapidio_0_constraints.tcl** script file sets the required constraints for compilation. The default f_{MAX} constraint on the system clock domain is 125 MHz. Modify this constraint if the system clock domain operates at a different speed than the default setting.

5. After the script has finished Analysis and Synthesis, on the Processing menu, click **Start Compilation** to compile your system.
6. After you successfully compile your design, you can program your target Altera device and verify your design in hardware.

9. Qsys Design Example

The design example in this chapter shows you how to use Qsys to build a system that combines a RapidIO IP core with other Qsys components. Qsys automatically generates simulation models and HDL files that include all the specified components and interconnections. The design example includes a testbench to simulate the Qsys system you generate. However, this design example does not support programming your target Altera device and verifying your design in hardware.

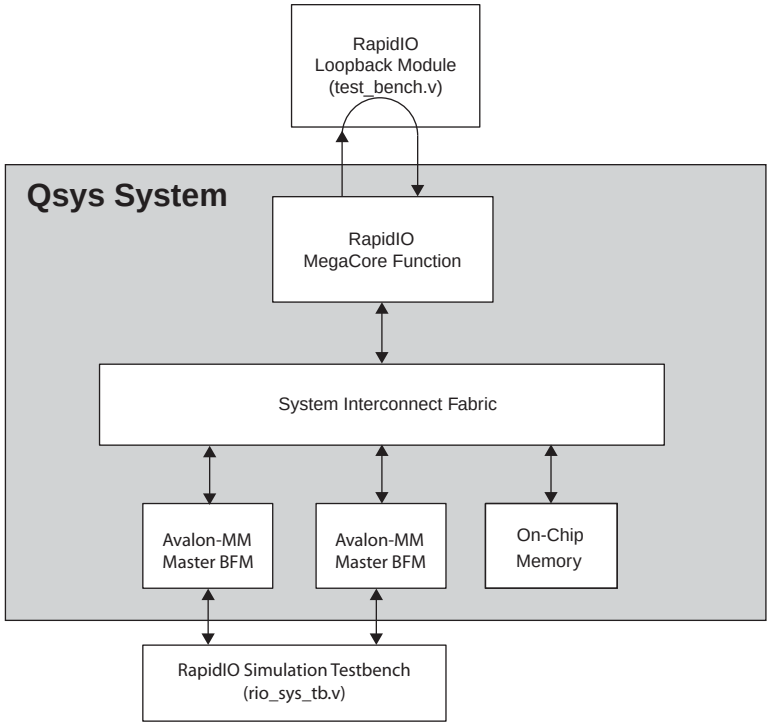
For more information about the system interconnect fabric, refer to the *Qsys Interconnect and System Design Components* chapter in volume 1 of the *Quartus II Handbook*. For more information about the Qsys integration tool, refer to the *System Design with Qsys* section in volume 1 of the *Quartus II Handbook*.

The design example explains how to use Qsys, an integral feature of the Quartus II software, to generate a system containing the following components:

- RapidIO IP core
- On-Chip Memory
- Two Master BFMs

Figure 9–1 shows a block diagram of the system you create in this chapter.

Figure 9–1. Simulation Example Qsys System



In this chapter you create a design example by following these steps:

1. Creating a New Quartus II Project
2. Running Qsys
3. Adding and Parameterizing the RapidIO Component
4. Adding and Connecting Other System Components
   a. Adding the Master Maintenance BFM
   b. Adding the Master I/O BFM
   c. Adding the On-Chip Memory
   d. Assigning Addresses and Setting the Clock Frequency
5. Generating the System
6. Simulating the System

The Qsys design example is a simulation example. It does not support programming your target Altera device and verifying your design in hardware.

This design example does not use all available parameters and options.

For more information about specific parameters used in this design example, refer to Chapter 3, Parameter Settings.

Creating a New Quartus II Project

You must create a new Quartus II project. You can create the project with the New Project Wizard, which helps you specify the working directory for the project, assign the project name, and designate the name of the top-level design entity. To create a new project, follow these steps:

1. On the Windows start menu, click **Programs > Altera > Quartus II** <*version*> **> Quartus II** <*version*> to run the Quartus II software.
2. On the File menu, click **New Project Wizard**. If you did not turn it off previously, the **New Project Wizard: Introduction** page appears.
3. On the **New Project Wizard: Introduction** page, click **Next**.
4. On the **New Project Wizard: Directory, Name, Top-Level Entity** page, enter the following information:
   a. Specify the working directory for your project. This directory is also called the Quartus II project directory. This directory can be any directory to which you have write permission, and the pathname should be free of spaces or special characters.
   b. Specify the name of the project. This design example uses **rio_sys**. You must specify this name for both the project and the Qsys system.

   ☞ The Quartus II software specifies a top-level design entity that has the same name as the project automatically. Do not change this name.

5. Click **Next** to display the **Add Files** page.

Click **Yes**, if prompted, to create a new directory.

6. Click **Next** to display the **Family and Device Settings** page.
7. On the **Family and Device Settings** page, select the following target device family and options:
   a. In the **Family** list, select **Stratix IV (GT/GX/E)**.

   This design example creates a design targeting the Stratix IV GX device family. You can also use these procedures for other supported device families, after modifying the design example **sim.do** file appropriately.

   b. In the **Target device** box, select **Auto device selected by the Fitter**.
8. Click **Next** to display the **EDA Tool Settings** page.
9. Click **Next** to display the **Summary** page.
10. Check the **Summary** page to ensure that you have entered all the information correctly.
11. Click **Finish** to complete the Quartus II project.

Running Qsys

This section provides instructions to create and generate your own Qsys system for the design example. The instructions teach you the process to create a Qsys system.

If you prefer to run the design example without performing the steps to create your own Qsys system, you can use the Qsys system file (**.qsys**) provided in the design example directory.

To run Qsys to generate your system, whether from your own **.qsys** file or the design example installation **.qsys** file, perform the following steps:

1. In the Quartus II software, on the Tools menu, click **Qsys**.
2. To use the Qsys system provided with your Quartus II installation in <*QII_install_dir*>, perform the following steps:
   a. Copy the file <*QII_install_dir*>**\ip\altera\rapidio\lib\rio\qsys_cust_demo\rio_sys.qsys** to your Quartus II project directory.
   b. In Qsys, on the File menu, click **Open**.
   c. Browse to your Quartus II project directory, if necessary, and click **rio_sys.qsys**.
   d. Click **Open**.
   e. Proceed directly to "Generating the System" on page 9–11 and follow the instructions.
3. To learn how to create a Qsys system by generating the design example Qsys system manually, proceed to "Adding and Parameterizing the RapidIO Component" and follow the instructions.

For more information about how to use Qsys, refer to Quartus II Help or to the *Creating a System with Qsys* chapter in volume 1 of the *Quartus II Handbook*.

Adding and Parameterizing the RapidIO Component

To instantiate and parameterize the RapidIO IP core in your system, perform the following steps:

1. In the Component Library, in the search box, type `RapidIO`.
2. Highlight the **RapidIO** MegaCore component and click **Add**.
3. To parameterize your IP core, follow these steps:
   a. On the **Physical Layer** page, specify the **Device Options** settings in Table 9–1.

Table 9–1. Set Physical Layer Device Options

Option	Value	Comment
Mode Selection	**4x Serial**	
Automatically synchronize transmitted ackID	Turn off	This value is the default value.
Send link-request reset-device on fatal errors	Turn off	This value is the default value.
Link request attempts	**7**	This value is the default value.

   b. Specify the Data Settings values in Table 9–2.

Table 9–2. Set Physical Layer Data Settings

Option	Value	Comment
Baud rate	**2500** Mbaud	This value is the default value.
Reference clock frequency	**125** MHz	This value is the default value.
Receiver buffer size	**4** Kbytes	This value is the default value.
Transmit buffer size	**8** Kbytes	This value is the default value.

   c. Specify the Receive Priority Retry Threshold values in Table 9–3.

Table 9–3. Set Physical Layer Receive Priority Retry Threshold

Option	Value	Comment
Receive Priority Retry Threshold	Turn on **Auto-configured from receiver buffer size**	This value is the default value.

d. Click the **Transport and Maintenance** tab.

e. Under **Transport Layer**, leave all three options turned off. For Qsys, the Transport layer is always enabled.

f. Under **I/O Maintenance Logical Layer Module**, set the parameters in Table 9–4.

Table 9–4. Set Transport Layer Options

Option	Value	Comment
Maintenance logical layer interface(s)	**Avalon-MM Master and Slave**	
Number of transmit address translation windows	**1**	This value is the default value.
Port write Tx enable	Turn off	This value is the default value.
Port write Rx enable	Turn off	This value is the default value.

g. Click the **I/O and Doorbell** tab.

On the **I/O and Doorbell** tab, leave all settings at their default values. To fully exercise the design example testbench, you must maintain the default I/O Logical layer Avalon-MM Master and Avalon-MM Slave ports. Turning off `DOORBELL` messaging, the default under **Doorbell Slave**, reduces resource usage and may be desirable for some applications.

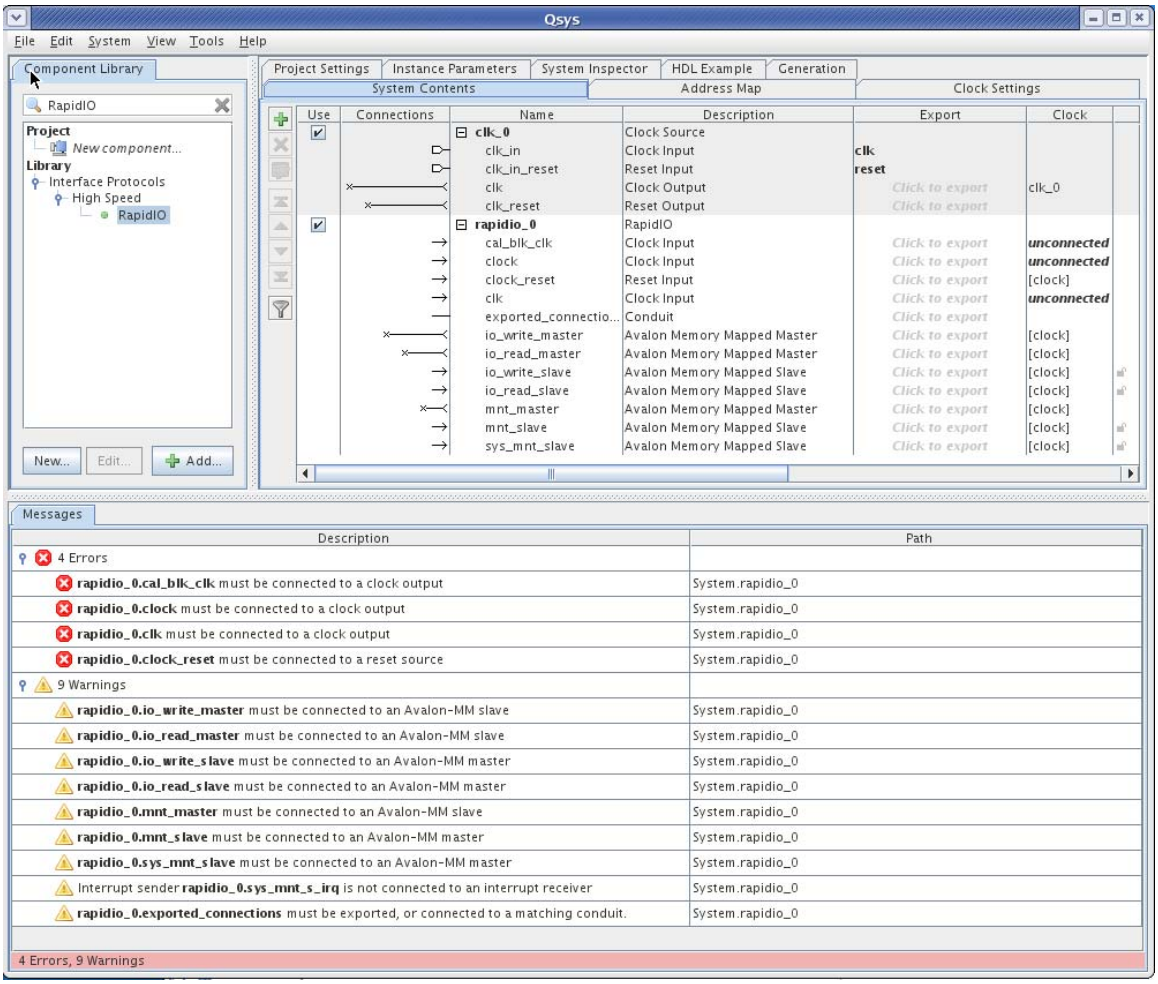
h. Click the **Capability Registers** tab. You can set the Device Register to match your system. Unless your design includes an additional extended feature block, keep the **Extended features pointer** default value of **0x0100**. You can keep the default values for all other parameters.

i. Under **Data Messages**, make sure both options are turned off.

j. Click **Finish** to complete parameterization and add the RapidIO IP core to the Qsys system.

After you add the RapidIO IP core component to your system, various Avalon-MM ports are created and shown as connection points in the **System Contents** tab. Error messages indicate that these ports are not connected, as shown in Figure 9–2.

Figure 9–2. RapidIO IP core Added and Avalon-MM Ports Created



These errors are resolved as you add the remaining components to your system and make all of the appropriate connections, as described in the following sections.

The default instance name of the RapidIO IP core component is **rapidio_0**. To run the design example, you must retain the default name. However, in your own system, you can change any default component instance name by right-clicking on the name and then clicking **Rename**. The component name must be unique; it cannot be the same name as the system name.

Adding and Connecting Other System Components

To complete your testbench system, you add and connect the following components, assign addresses, and set the clock frequency:

- Master Maintenance BFM

- Master I/O BFM
- On-Chip Memory

The BFM components are functional only for simulation; you cannot compile this design example system and program it on a device.

Adding the Master Maintenance BFM

To add the Master Maintenance BFM to your system, perform the following steps:

1. In the Component Library, in the search box, type `Altera Avalon MM Master BFM`.
2. Highlight **Altera Avalon MM Master BFM** and click **Add**. The Avalon-MM Master BFM component is added to the system, and the Avalon-MM Master BFM parameter editor appears.
3. Under **Port Widths** and **Parameters**, leave the default values.
4. Under **Port Enables**, turn on and turn off options to enable only the following options:
   - **Use the read signal**
   - **Use the write signal**
   - **Use the address signal**
   - **Use the readdata signal**
   - **Use the readdatavalid signal**
   - **Use the writedata signal**
   - **Use the waitrequest signal**
5. Click **Finish** to add the Avalon MM Master BFM to your Qsys system.
6. Right-click on the default name of the new component, **mm_master_bfm_0**, and click **Rename**.
7. Type the new name, `master_bfm`. The design example requires this name to run.

Adding the Master I/O BFM

To add the Master I/O BFM to your system, perform the following steps:

1. In the Component Library, in the search box, type `Altera Avalon MM Master BFM`.
2. Highlight **Altera Avalon MM Master BFM** and click **Add**. The Avalon-MM Master BFM component is added to the system, and the Avalon-MM Master BFM parameter editor appears.
3. Under **Port Widths**, leave the default values.
4. Under **Parameters**, set the options in Table 9–5.

Table 9–5. Set Parameter Options

Option	Value
Number of Symbols	**8**
Burstcount width	**6**

5. Under **Port Enables**, turn on and turn off options to enable only the following options:
   - **Use the read signal**
   - **Use the write signal**
   - **Use the address signal**
   - **Use the byteenable signal**
   - **Use the burstcount signal**
   - **Use the readdata signal**
   - **Use the readdatavalid signal**
   - **Use the writedata signal**
   - **Use the waitrequest signal**
6. Click **Finish** to add the second Avalon MM Master BFM to your Qsys system.
7. Right-click on the default name of the new component, **mm_master_bfm_0**, and click **Rename**.
8. Type the new name, `master_bfm_io`. The design example requires this name to run.

Adding the On-Chip Memory

To add on-chip memory to your system, perform the following steps:

1. In the Component Library, in the search box, type `On Chip Memory`.
2. Highlight **On-Chip Memory (RAM or ROM)** and click **Add**. The On-Chip Memory component is added to the system, and the On-Chip Memory parameter editor appears.
3. Select **64** as the **Data width**.
4. Click **Finish** to retain default settings for other parameters and add the On-Chip Memory to your Qsys system.

Connecting Clocks and the System Components

You must now connect any unconnected clocks and other components in your system.

To support external connections, you must export them. Click ***Click to export*** in the **Export** column for the **rapidio_0.clk** and **rapidio_0.exported_connections** ports. The **clk_0.clk_in** and **clk_0.clk_in_reset** signals are already exported.

For the external RapidIO processing elements to access the internal registers of the RapidIO variation, your system must meet the following criteria:

- The `Maintenance Master` port must be connected to the `System Maintenance Slave` port.
- The `System Maintenance Slave port Base address` must be assigned to address `0x0`.

The following sections show you how to make these connections and assignments, and others required for the design example.

Connecting Unconnected Clocks

Information about the clocks in the system appears in the **Connections**, **Name**, **Description**, and **Clock** columns.

Connect all clocks designated as ***unconnected*** in the **Clock** column. Click ***unconnected*** in the **Clock** column to assign the clock to `clk_0`.

This instruction does not affect the **rapidio_0.clk** port, which you exported previously. This port is designated ***exported*** in the Clock column.

You must ensure that you also connect the calibration clock (`cal_blk_clk`) to a clock with the appropriate frequency range 10–125 MHz. In this example, the default external clock, `clk_0`, is in this range.

Connecting System Components

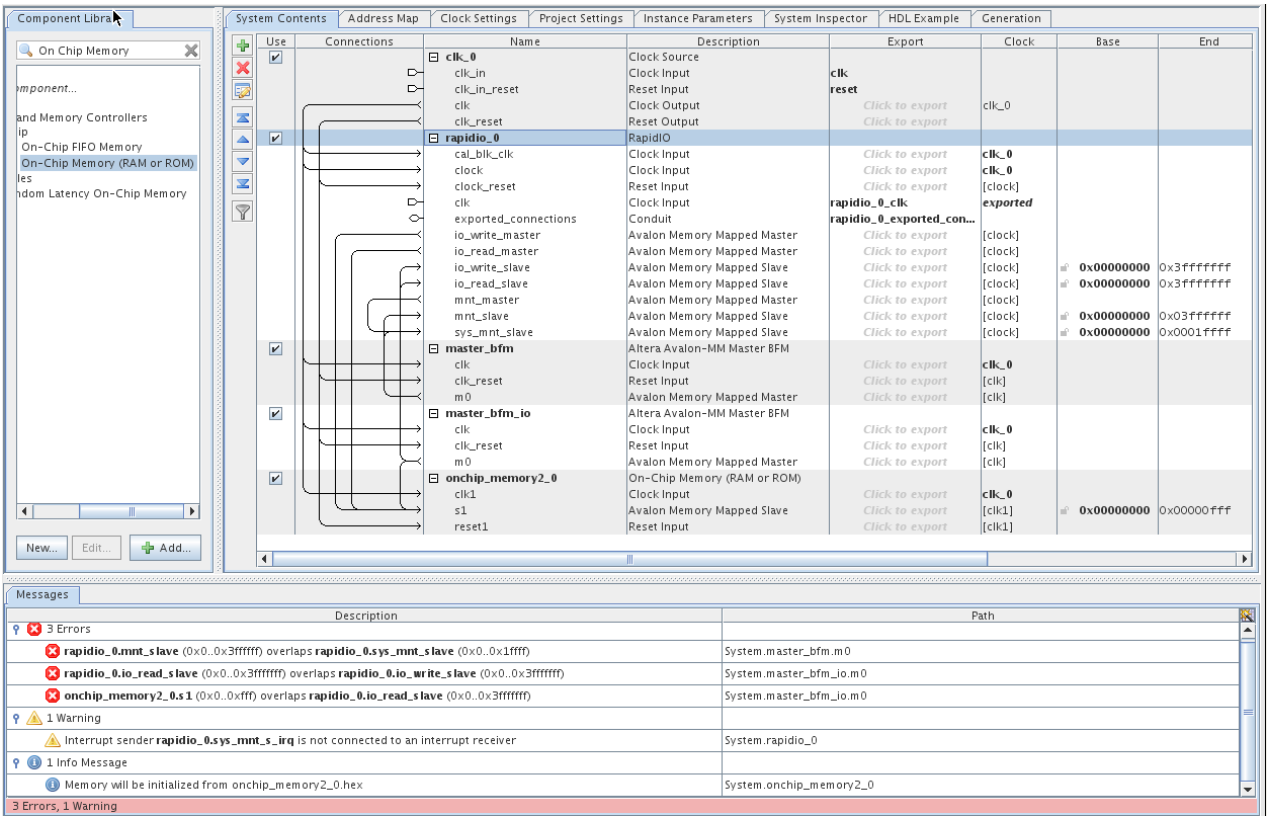
In Qsys, clicking and hovering the mouse over the **Connections** column displays the potential connection points between components, represented as dots connecting wires. A filled dot shows that a connection is made; an open dot shows a potential connection point that is not currently connected. Clicking a dot toggles the connection status. To complete this design, create the connections listed in Table 9–6.

Table 9–6. Connect System Components

Make Connection From	To
clk_0 `clk_reset`	rapidio_0 `clock_reset`
	master_bfm `clk_reset`
	master_bfm_io `clk_reset`
	onchip_mem... `reset1`
rapidio_0 `mnt_master`	rapidio_0 `sys_mnt_slave`
rapidio_0 `io_read_master`	onchip_mem... `s1`
rapidio_0 `io_write_master`	onchip_mem... `s1`
master_bfm `m0`	rapidio_0 `mnt_slave`
	rapidio_0 `sys_mnt_slave`
master_bfm_io `m0`	rapidio_0 `io_write_slave`
	rapidio_0 `io_read_slave`
	onchip_mem... `s1`

Refer to Figure 9–3 to ensure that you connected the ports correctly.

Figure 9–3. Complete System Connections



As described in "Reset for RapidIO IP Cores with Physical, Transport, and Logical Layers" on page 4–10, the circuitry necessary to ensure the correct behavior of the `reset_n` input signal to the RapidIO IP core is created automatically by Qsys. For this design example, you do not implement the logic described in Figure 4–4, because Qsys implements it for you.

The remaining errors are resolved as you modify the slave port base addresses, as described in the following section.

Assigning Addresses and Setting the Clock Frequency

To assign a specific address, follow these steps:

1. Click on the address that you want to change in the **Base** column, then type the address that you want to assign. Make the address assignments specified in Table 9–7.

Table 9–7. Assign Addresses (Part 1 of 2)

Port Name	Base Address
rapidio_0 io_write_slave	0x40000000
rapidio_0 io_read_slave	0x80000000
rapidio_0 mnt_slave	0x04000000

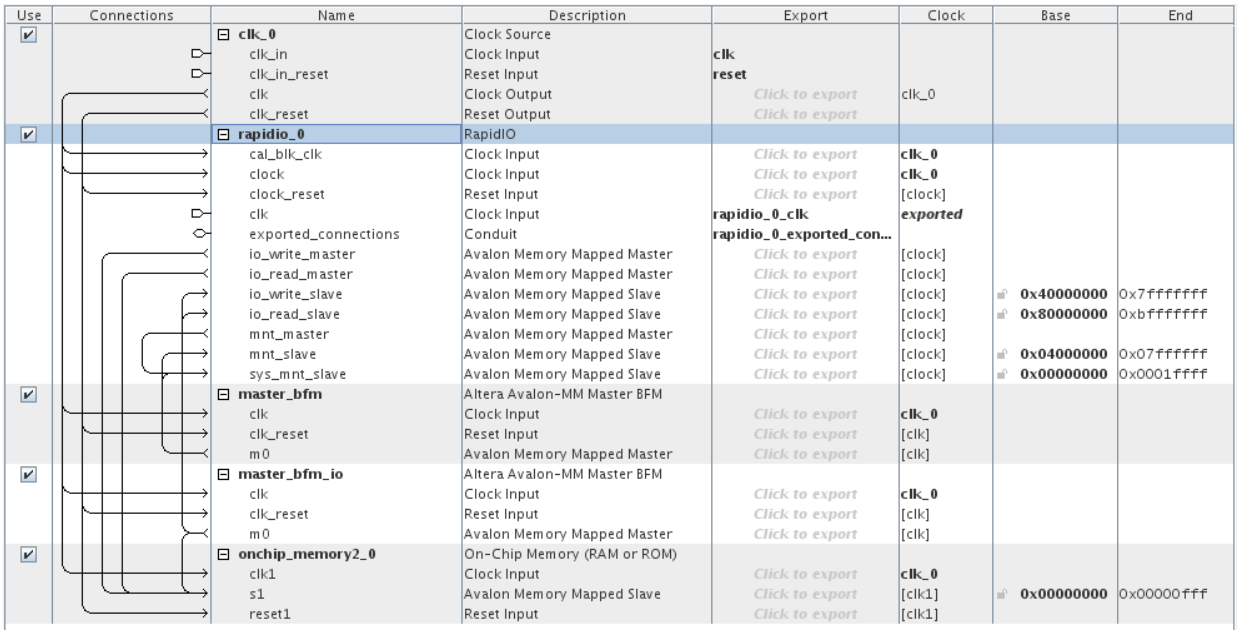
Table 9–7. Assign Addresses (Part 2 of 2)

Port Name	Base Address
rapidio_0 sys_mnt_slave	0x00000000
onchip_mem... s1	0x00000000

2. On the File menu, click **Save** and type `rio_sys` to save the Qsys system in the **rio_sys.qsys** file.

Figure 9–4 shows the completed Qsys system.

Figure 9–4. Complete Qsys Example System



Generating the System

After you create your system with all the required components and connections and you have resolved any errors, generate the system by following these steps:

1. Click the **Generation** tab.
2. For **Create simulation model**, select **Verilog**.
3. For **Create testbench Qsys system**, select **None**.
4. For **Create testbench simulation model**, select **None**.
5. Turn off **Create HDL design files for synthesis**. This Qsys system cannot run on hardware.
6. Turn off **Create block symbol file (.bsf)** to expedite the generation process.
7. Click **Generate** to start the generation process.

If you are prompted to save your changes to **rio_sys.qsys**, click **Save**.

Generating the system files, the simulation models, and the environment takes a few minutes.

When the Qsys system is generated successfully, the system HDL files are added to your project directory and are ready to be simulated with the Quartus II software.

8. After generation completes successfully, click **Exit** to close Qsys.

Although this design example requires the Verilog HDL target output, you can alternatively select VHDL for a project of your own.

Simulating the System

To simulate your system with the sample Verilog HDL testbench, follow these steps:

1. Copy the following files from the **\ip\altera\rapidio\lib\rio\qsys_cust_demo** subdirectory of your Quartus II installation directory to your Quartus II project directory:
   - **rio_sys_tb.v**
   - **sim.do**
   - **test_bench.v**
   - **test_input.v**
   - **test_result.v**
2. Start the ModelSim software. On the File menu, change directory to your Quartus II project directory.
3. Type the following command at the ModelSim command prompt:

```
do sim.do ↵
```

The RapidIO design example performs the following transactions in simulation:

- Sends a sequence of read requests to the internal registers of the IP core
- Sets up other internal registers of the IP core for `MAINTENANCE` and I/O transactions and reads the registers to ensure the write operations completed
- Writes data to the Maintenance slave, reads it back, and verifies data integrity
- Sends burst transfer write and read requests to the IP core to send out on the RapidIO link, and verifies data integrity

When simulation completes, on the File menu, click **Quit** to close the ModelSim software.

A. Initialization Sequence

This appendix describes the most basic initialization sequence for a RapidIO system that contains two RapidIO IP cores connected through their RapidIO interfaces.

To initialize the system, perform these steps:

1. Read the `Port 0 Error and Status (ERRSTAT) Command and Status` register (CSR) (`0x00158`) of the first RapidIO IP core to confirm port initialization.
2. Set the following registers in the first RapidIO IP core:
   a. To set the base ID of the device to `0x01`, set the `DEVICE_ID` field (bits 23:16) or the `LARGE_DEVICE_ID` field (bits 15:0) of the `Base Device ID` register (`0x00060`) to `0x1`.
   b. To allow request packets to be issued, write `1` to the `ENA` field (bit 30) of the `Port General Control CSR` (`0x13C`).
   c. To set the destination ID of outgoing maintenance request packets to `0x02`, set the `DESTINATION_ID` field (bits 23:16) or the combined {`LARGE_DESTINATION_ID (MSB)`, `DESTINATION_ID`} fields (bits 31:16) of the `Tx Maintenance Window 0 Control` register (`0x1010C`) to `0x02`.
   d. To enable an all-encompassing address mapping window for the maintenance module, write `1'b1` to the `WEN` field (bit 2) of the `Tx Maintenance Window 0 Mask` register (`0x10104`).
3. Set the following registers in the second RapidIO IP core:
   a. To set the base ID of the device to `0x02`, set the `DEVICE_ID` field (bits 23:16) or the `LARGE_DEVICE_ID` field (bits 15:0) of the `Base Device ID` register (`0x00060`) to `0x02`.
   b. To allow request packets to be issued, write `1'b1` to the `ENA` field (bit 30) of the `Port General Control CSR` (`0x13C`).
   c. To set the destination ID of outgoing maintenance packets to `0x0`, set the `DESTINATION_ID` field (bits 23:16) or the combined {`LARGE_DESTINATION_ID (MSB)`, `DESTINATION_ID`} fields (bits 31:16) of the `Tx Maintenance Window 0 Control` register (`0x1010C`) to `0x0`.
   d. To enable an all-encompassing address mapping window for the maintenance module, write `1'b1` to the `WEN` field (bit 2) of the `Tx Maintenance Window 0 Mask` register (`0x10104`).

These register settings allow one RapidIO IP core to remotely access the other RapidIO IP core.

To access the registers, the system requires an Avalon-MM master, for example a Nios II processor. The Avalon-MM master can program these registers.

You can use the Qsys system integration tool, available with the Quartus II software, to rapidly and easily build and evaluate your RapidIO system. For legacy systems, you can use SOPC Builder, a Quartus II software tool, to evaluate changes in your RapidIO system. For an example, refer to Chapter 8, SOPC Builder Design Example.

For more information about initializing a RapidIO system, refer to Fuller, Sam. 2005. *RapidIO: The Embedded System Interconnect*. John Wiley & Sons, Ltd., Chapter 10 *RapidIO Bringup and Initialization Programming*.

B. XGMII Interface Timing

This appendix describes the RapidIO XGMII interface required for the RapidIO IP core to communicate with an external transceiver. This appendix illustrates the clock layout and timing, provides insight into timing constraints and data alignment, and includes an example.

RapidIO XGMII Interface

The RapidIO IP core supports an XGMII-like interface that connects the RapidIO IP core to an external transceiver. The RapidIO XGMII interface is similar to the 10-Gigabit Media-Independent Interface (XGMII). The RapidIO XGMII interface is available for all RapidIO supported device families except Arria II GX, Arria II GZ, Arria V, Cyclone IV GX, Cyclone V, HardCopy IV GX, Stratix IV GX, and Stratix V devices. With these exceptions, the XGMII interface is available for devices that can also have internal transceivers, such as Arria GX and Stratix II GX devices.

The RapidIO XGMII interface connects to an external transceiver interface with these characteristics:

- 8-bit data transmit and receive datapaths per serial lane
- Control and clocking signals that allow bidirectional data transfers
- Supports `phy_dis`, an external transceiver transmitter disable signal

The RapidIO XGMII Transmit and Receive interfaces support bidirectional data transfer between the RapidIO IP core and an external transceiver. The Transmit interface allows the RapidIO IP core to transfer data to the external transceiver. The Receive interface allows the RapidIO IP core to process data received from the external transceiver.

The XGMII Receiver interface supports one control, one error, and one clock signal per 8 bits from the external transceiver decoder.

For maximum flexibility, the RapidIO XGMII-like interface features one clock signal per group of 8 bits of received data. The standard XGMII interface usually has only one receiver clock per interface.

The RapidIO specification requires that the link output drivers be disabled when the Initialization state machine is in the *SILENT* state to force the link partner to re-initialize. The `phy_dis` output signal is driven high by the RapidIO MegaCore when its initialization state machine is in the *SILENT* state so that this signal can be used to disable the link output drivers.

Figure B–1 through Figure B–3 illustrate the XGMII interface in 1x mode and 4x mode. Figure B–1 illustrates the 1x interface. Figure B–2 shows the 4x RapidIO XGMII Transmit interface allowing data from the RapidIO IP core to be transmitted to the external transceiver. Figure B–3 shows the 4x RapidIO XGMII Receiver interface which allows the RapidIO IP core to process data received from the external transceiver.

Figure B–1. 1x XGMII Clock Interface

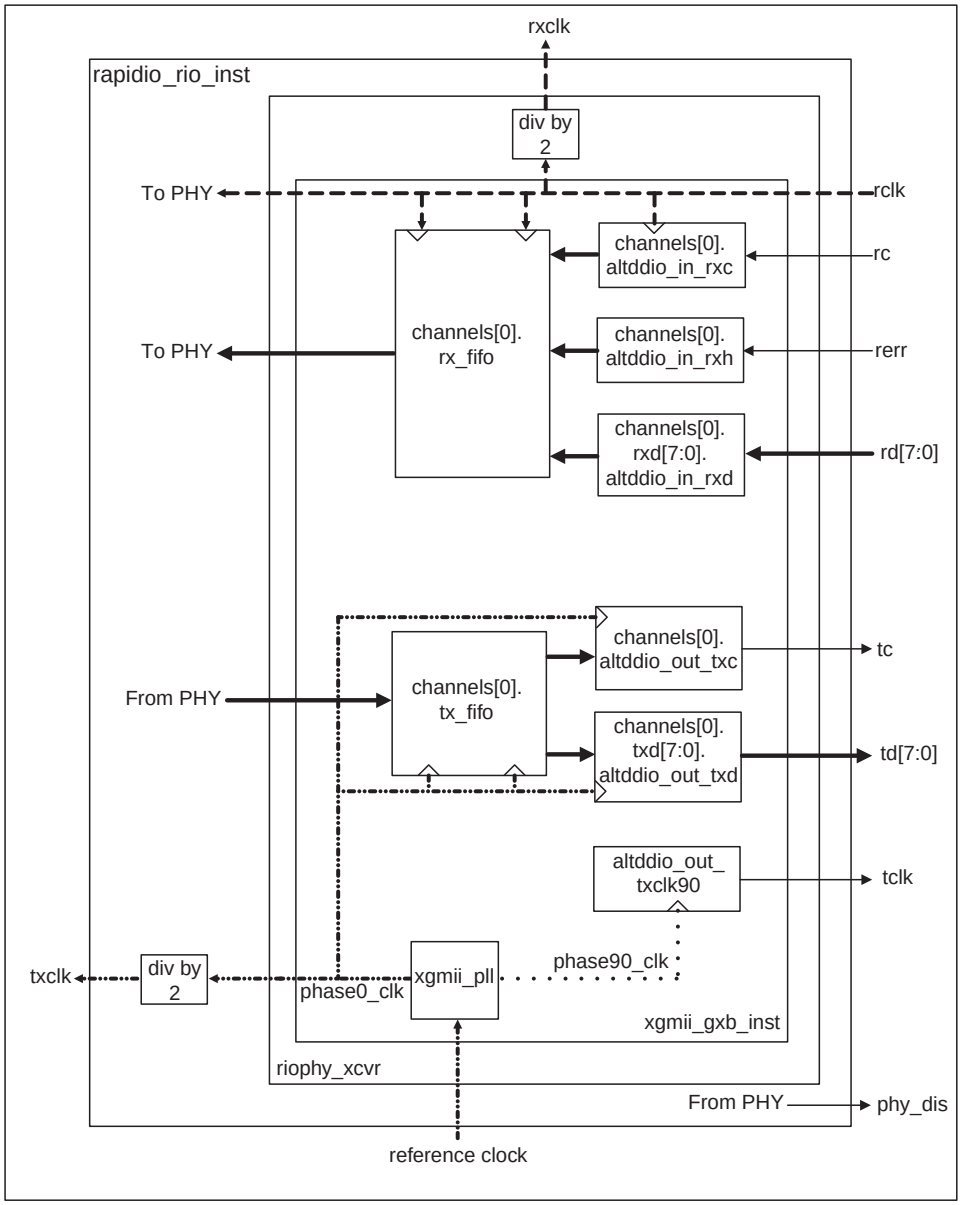


Figure B–2. 4x Tx XGMII Clock Interface

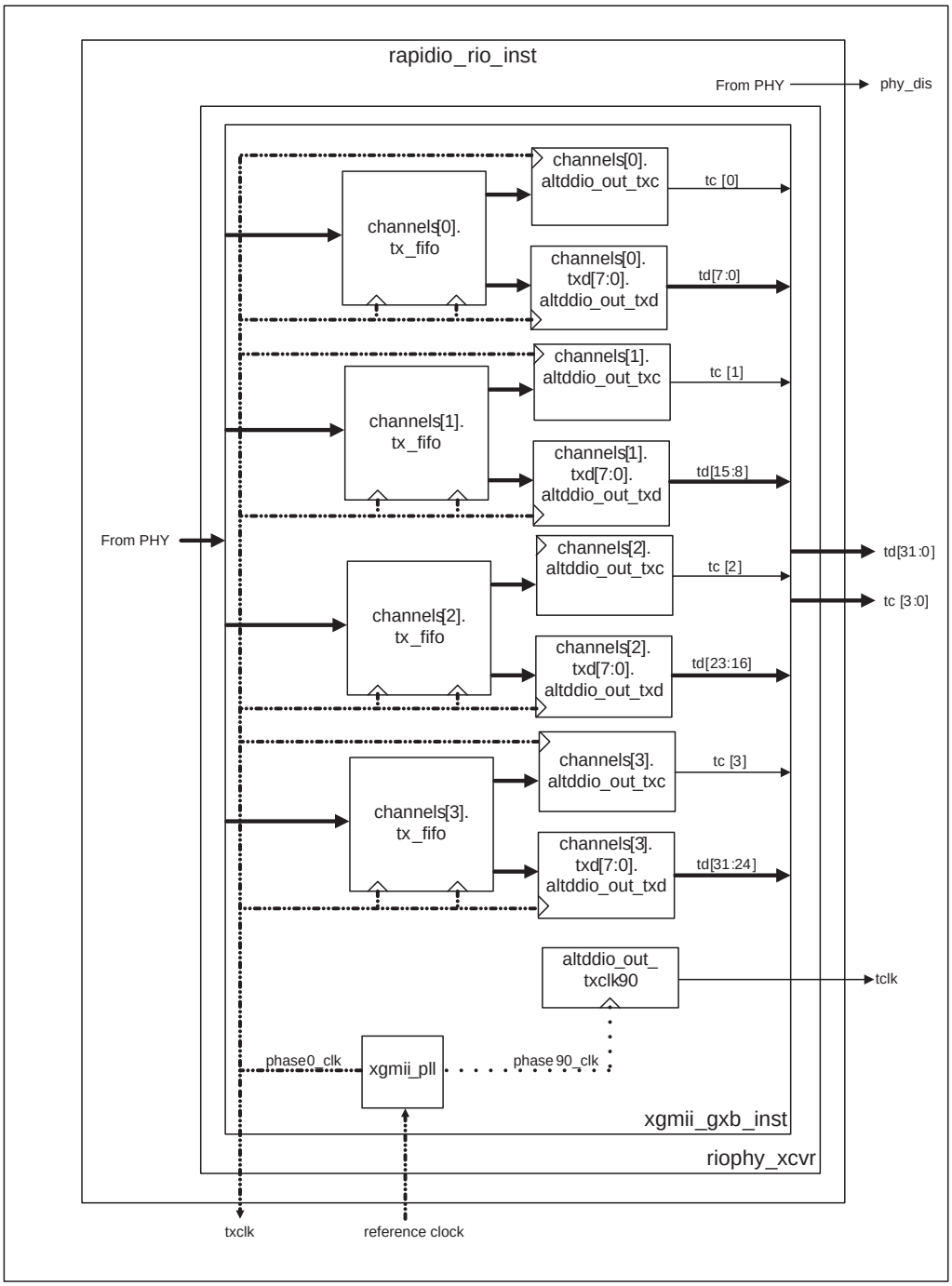
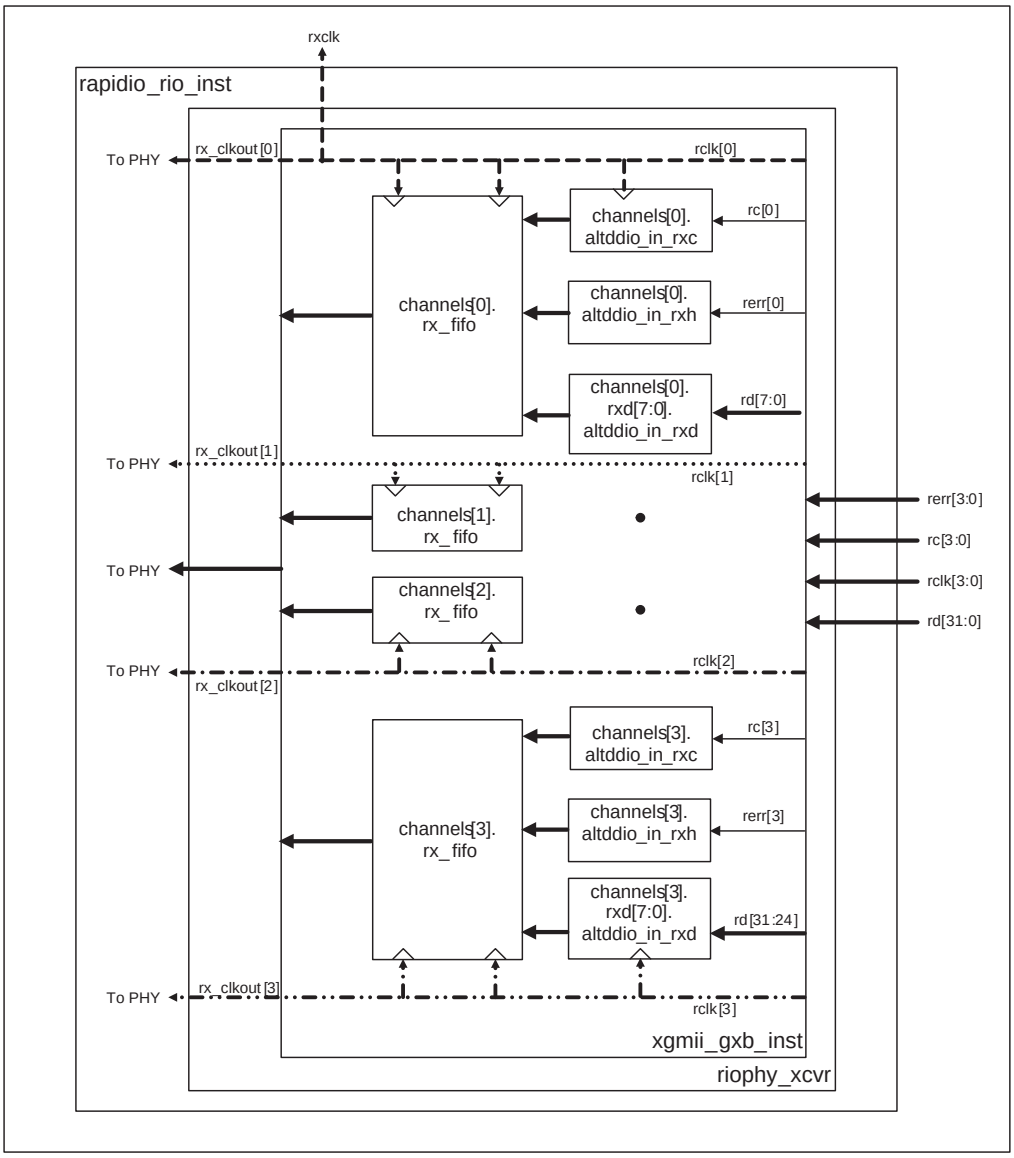


Figure B–3. 4x Rx XGMII Clock Interface



The `xgmii_pll` instantiated by the MegaCore generates two clocks:

- 90° phase shifted clock
- 0° phase shift clock

The 90° phase shifted clock ensures that the `td` and `tc` signals are transmitted on the rising and falling edges of the center aligned clock `tclk`. The `tclk` is not a separate clock domain because it is not an FPGA clock signal. Instead, alternating 1s and 0s are preloaded into the `altddio_out` serializer.

In some cases, due to timing or configuration settings, the external transceiver may require the data and control signals to be transmitted on the rising and falling edges of an edge aligned clock. If this change is required in the <*variation_name*>**_xgmii_gxb.v** file contains the `edge_aligned` parameter. By default, this is set to `0`. Setting `edge_aligned` to `1` clocks the `altddio_out` serializer with the `phase0_clk` instead, thus ensuring the data is transmitted on the edges of `tclk`.

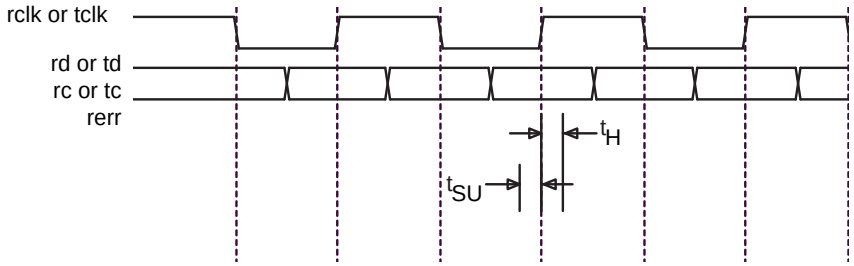
The RapidIO IP core outputs the Rx recovered MegaCore clock (`rxclk`). In 1x mode, `rxclk` is derived from the `rclk` input, and is divided by two internally by a flip-flop. In 4x mode, `rxclk` is directly driven by the `rclk` from channel 0.

If you have a 4x Rx XGMII RapidIO IP core with only one clock from the external transceiver, change the port mapping to ensure that the `rclk` input simultaneously drives the four `rclk` inputs to the IP core.

Timing Constraints

RapidIO transmits source-center aligned data using either HSTL Class 1 or SSTL Class 2 I/O drivers. The clock rate required is 156.25 MHz for 3.125 Gbaud, 125 MHz for 2.5 Gbaud, and 62.5 for 1.25 Gbaud. The timing diagram in Figure B–4 illustrates basic timing relationships.

Figure B–4. XGMII Timing



Notes to Figure B–4:

(1) A typical Transmitter t_{SU} and t_H at 3.125 Gbaud is 960 ps.

(2) An ideal Receiver t_{SU} and t_H at 3.125 Gbaud is 480 ps.

On the receive side, the 8-bit data (`rd`) and 1-bit control (`rc`) signals per lane are received and sampled on the rising and falling edges of a center aligned clock, `rclk`. Separate error (`rerr`) and `rclk` signals are associated with each lane.

On the transmit side, the 8-bit data (`td`) and 1-bit control (`tc`) signals per lane are transmitted on the rising and falling edges of a center aligned clock, `tclk`.

The RapidIO XGMII interface requires the following I/O timing relationships:

- Use Fast Inputs for `rd`, `rc` and other inputs.
- Use similar clock types (for example `rclk[0]` should not be a global clock and `rclk[1]` a regional clock).

Setting Quartus II t_{SU} and t_H Checks

You must specify t_{SU} and t_H timing requirements in the Quartus II software for the XGMII receive interface (rd, rc, rerr). The value to use for the t_{SU} and t_H is a function of the following:

- Effects of clock jitter and other signal integrity issues.
- Any clock phase offset on the output of the attached device.
- Skew over the traces.
- Adjustments for output clock phase. If not exactly center aligned, adjust the t_{SU} and t_H assignments accordingly.

Example

The following example describes how to calculate the appropriate timing constraints for an example design. If the external transceiver output clock (which is connected to rclk) is out 80 ps past the center point (including all other functions that adjust the clock phase), subtract 80 ps from the t_{sU} and subtract 80 ps from the t_H. Using the Rx ideal number above for 3.125 Gbaud and subtracting the trace skew, adjust for the clock phase as shown below for TimeQuest timing analyzer assignments:

- ```
  set_max_delay -from [get_pins -hierarchical *] \
  -to [get_ports {rd*}] 0.4
  ```
- ```
  set_max_delay -from [get_pins -hierarchical *] \
  -to [get_ports {rc}] 0.4
  ```
- ```
  set_max_delay -from [get_pins -hierarchical *] \
  -to [get_ports {rerr}] 0.4
  ```
- ```
  set_min_delay -from [get_pins -hierarchical *] \
  -to [get_ports {rd*}] -0.4
  ```
- ```
  set_min_delay -from [get_pins -hierarchical *] \
  -to [get_ports {rc}] -0.4
  ```
- ```
  set_min_delay -from [get_pins -hierarchical *] \
  -to [get_ports {rerr}] -0.4
  ```

C. Porting a RapidIO Design from the Previous Version of the Software

This appendix describes how to port your RapidIO design from the previous version of the RapidIO IP core and Quartus II software.

Upgrading a RapidIO Design Without Changing Tools

To upgrade your RapidIO design that you developed and generated using the RapidIO IP core v11.1, to the IP core v12.0, perform the following steps:

1. Open the Quartus II software v12.0.
2. On the File menu, click **Open Project**.
3. Navigate to the location of the **.qpf** file you generated with the Quartus II software v11.1.
4. Select the **.qpf** file and click **Open**.
5. If the project was generated by SOPC Builder originally, follow these steps:
   a. Open SOPC Builder.
   b. To edit the RapidIO IP core, double-click its name in SOPC Builder. The RapidIO parameter editor appears.
   c. Click **Finish**.
   d. In SOPC Builder, regenerate the project.
6. If the RapidIO IP core was generated using the MegaWizard Plug-In Manager originally, follow these steps:
   a. Open the existing IP core for editing in the MegaWizard Plug-In Manager.
   b. Click **Finish**.
7. If the RapidIO IP core was generated using the Qsys system integration tool originally, follow these steps:
   a. Open the Qsys system.
   b. To edit the RapidIO IP core, double-click its name in Qsys. The RapidIO parameter editor appears.
   c. Click **Finish**.
   d. In Qsys, regenerate the project.
8. Proceed with simulation, adding the RapidIO timing constraints, and compilation.

Before you add the RapidIO timing constraints, use the Assignment Editor to remove the old 0PPM assignments for this IP core. Otherwise, the new 0PPM settings are not written.

Upgrading an SOPC Builder Design with a RapidIO Component to a Qsys System

To upgrade an existing RapidIO IP core that you developed and generated using the MegaWizard Plug-In Manager, to a Qsys component, you must recreate the RapidIO component in Qsys.

To upgrade an existing RapidIO design that you developed and generated using SOPC Builder, to a Qsys design, you can open the SOPC Builder File (**.sopc**) for the design in Qsys, and regenerate the system in Qsys.

After you open the **.sopc** in Qsys, you must edit the RapidIO component to restore lost information. Altera recommends that you maintain a copy of your SOPC Builder system so that you can retrieve the information about the desired settings from the original RapidIO and ALTGX megafunctions.

In addition, Qsys prompts you to connect the RapidIO IP core input clock `clk` to an output clock. Connect it to the `clk_`<*variation*>`.clk` clock output signal.

The following changes and information losses occur if you upgrade an SOPC Builder system that includes a RapidIO IP core using this process:

- The transceiver settings of the RapidIO IP core are reset to default settings in the Qsys system. You must modify the transceiver settings by updating the ALTGX megafunction settings in the MegaWizard Plug-In Manager.
- Some signal names are modified. Signal names that have the _<*variation*> suffix in the SOPC Builder system, instead have the suffix `_from_the_`<*variation*> or `_to_the_`<*variation*> in the Qsys system. The suffix for each signal depends on the signal direction.

  You can view the signals that are affected on the **HDL Example** tab of your new Qsys system.
- Clock and reset signal renaming depends on the conduit names you provide in Qsys following the migration. You can view the current signal names on the **HDL Example** tab of your new Qsys system.
- Some RapidIO IP core parameters that exist in SOPC Builder systems but not in Qsys systems, disappear in migration. Their disappearance generates warning messages which you can safely ignore.

For general information about the SOPC Builder to Qsys migration path, refer to *AN632: SOPC Builder to Qsys Migration Guidelines*.

D. Calculating Resource Utilization for Modular Configurations

You can generate a RapidIO IP core variation optimized for your specific application. Applications that require only a Physical layer benefit from the small footprint of Altera's RapidIO Physical layer solution. You can support applications that require a full three-layer solution by adding the Transport layer and enabling only the Logical layer modules required to support the application. This flexibility lets you choose between functionality and resource usage.

Modular Configurations

Table D–1 contains a short list of typical variations showing the functionality supported and example resource consumption numbers.

Table D–1. Modular Configurations

Variation	Functionality	Total		Increment Over PHY	
		ALUTs	M4K	ALUTs	M4K
Physical layer (PHY)	Refer to the Physical layer features in "Features" on page 1–1	3,806	29	—	—
Physical and Transport Layer with:					
Maintenance slave module	Source MAINTENANCE transactions	4,869	33	1,063	4
Maintenance master module	Terminate MAINTENANCE transactions	4,496	32	690	3
I/O slave module	Source I/O write/read transactions	5,806	56	2,000	27
I/O master module	Terminate I/O write/read transactions	5,201	42	1,395	13
Doorbell receive module	Terminate DOORBELL messages	4,292	33	486	4
Doorbell transmit module	Source DOORBELL messages	4,738	37	932	8
Avalon-ST interface	Provide the user direct access to the Transport layer, allowing the user to implement custom Logical layer modules	4,202	31	396	2

A variation with only the Physical layer is smaller than a variation with additional modules. For this example, Table D–1 shows that the variation with only the Physical layer provides the smallest footprint with an ALUT count of 3,806. Three-layer variations optimized for minimal resource usage start at an ALUT count of 4,202. These variations include the Physical and Transport layers and the Avalon-ST interface.

Calculate Estimated ALUTs for Specific Modules

The example in this section illustrates how to estimate the number of ALUTs for a variation with a specific layer configuration. In this example, the application requires the I/O slave and I/O master Logical layer modules. The ALUT consumption is not calculated as the sum of 5,806 and 5,201 ALUTs because the ALUT usage for the Physical layer and Transport layer is already accounted for in both calculations and would be incorrectly counted twice. Instead, perform calculations as shown in the following example:

1. Using the ALUT count for the Transport layer with I/O slave module, subtract the ALUT count of the Physical layer module. The result is found in the Increment Over PHY column.

   5,806 - 3,806 = 2,000 ALUTs

2. Starting with the ALUT count for the Transport layer with an I/O master module, subtract the ALUT count of the Physical layer module.

   5,201 - 3,806 = 1,395 ALUTs

3. Add the Physical layer ALUT count to the layer results from the previous calculations to get the final ALUT count.

   3,806 + 2,000 + 1,395 = 7,201 ALUTs

The preceding example is an estimate of the ALUT count for a variation with the Physical layer, Transport layer, and a Logical layer containing both I/O master and slave modules. You can perform similar calculations for other combinations.

This chapter provides additional information about the document and Altera.

Document Revision History

The following table shows the revision history for this user guide.

Date	Version	Changes
May 2012	12.0	■ Added support for Cyclone V GT ×1 variation at 5.0 Gbaud. ■ Updated speed grade support for Arria V, Stratix IV GX, and Stratix V devices. ■ Moved Modular Configurations section from Chapter 1, About This MegaCore Function to new Appendix D, Calculating Resource Utilization for Modular Configurations. ■ Clarified additional constraints on deassertion of `reset_n` and `phy_mgmt_clk_reset` in "Clocking and Reset Structure" on page 4–3 and in Chapter 5, Signals.
November 2011	11.1	■ Added support for Arria V and Cyclone V devices. Variations that target one of these two device families configure the transceiver with the Custom PHY IP core. ■ Added Chapter 9, Qsys Design Example. ■ Enhanced description of `arxmty` signal in Table 5–7 on page 5–4. ■ Updated simulation sections in Chapter 2, Getting Started. ■ Refered to new What's New in Altera IP page for information about IP core support level for many device families.
May 2011	11.0	■ Upgraded to final support for Arria II GZ, Cyclone III LS, and Cyclone IV GX devices. ■ Upgraded to HardCopy Compilation support for HardCopy III, HardCopy IV E, and HardCopy IV GX devices. ■ Added preliminary support for Stratix V devices. ■ Added support for Custom PHY IP core in variations that target a Stratix V device.
December 2010	10.1	■ Added beta support for Qsys system integration tool. ■ Added read-only version of Port 0 Local AckID CSR.
July 2010	10.0	■ Added preliminary support for Cyclone IV GX devices. ■ Added support for configurable number of link-request attempts to be sent before fatal error, after time-out on link-response. ■ Added support for order preservation between read and write requests that come in on the Avalon-MM interface. ■ Removed support for Stratix GX devices.
November 2009	9.1	■ Added preliminary support for Cyclone III LS and HardCopy IV GX devices. ■ Added support for 5.0 Gbaud data rate. ■ Added support for order preservation between I/O write requests and `DOORBELL` requests. ■ Added NWRITE_R completion indication. ■ Added post-reset ackID synchronization. ■ Added transceiver configuration using full transceiver parameter editor.

Date	Version	Changes
March 2009	9.0	■ Corrected to preliminary support for HardCopy II devices. ■ Clarified the RapidIO IP core uses the transceiver bonded mode where relevant. ■ Updated Table 4–17.
February 2009	9.0	■ Added preliminary support for Arria II GX devices. ■ Added preliminary support for HardCopy III and HardCopy IV E devices. ■ Added support for outgoing multicast-event symbol generation. ■ Added support for 16-bit device ID. ■ Added Appendix C, Porting a RapidIO Design from the Previous Version of the Software.
November 2008	8.1	■ Added full support for Stratix III devices. ■ Added support for incoming multicast transactions. ■ Added GUI and register support to enable or disable destination ID checking. ■ Added GUI support to set transceiver starting channel number. ■ Added requirement to configure a dynamic reconfiguration block with Stratix IV transceivers, to enable offset cancellation. ■ Updated Figure 4–6 and Figure 7–2.
May 2008	8.0	■ Added Arria GX device support for 1x mode 3.125 GBaud variation. ■ Added Stratix IV device support. ■ Added GUI support to set VCCH and reference clock frequency. ■ Simplified Physical layer description in Functional Description chapter. ■ Updated the performance information.
October 2007	7.2	■ Added Avalon-ST pass-through interface to SOPC Builder flow. ■ Added support for EDA page and an option that creates a netlist for use by third-party synthesis tools. ■ Reorganized the user guide to make finding information easier and more efficient.

How to Contact Altera

To locate the most up-to-date information about Altera products, refer to the following table.

Contact (1)	Contact Method	Address
Technical support	Website	www.altera.com/support
Technical training	Website	www.altera.com/training
	Email	custrain@altera.com
Product literature	Website	www.altera.com/literature
Nontechnical support (general)	Email	nacomp@altera.com
(software licensing)	Email	authorization@altera.com

Note to Table:

(1) You can also contact your local Altera sales office or sales representative.

Typographic Conventions

The following table shows the typographic conventions this document uses.

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Indicate command names, dialog box titles, dialog box options, and other GUI labels. For example, **Save As** dialog box. For GUI elements, capitalization matches the GUI.
bold type	Indicates directory names, project names, disk drive names, file names, file name extensions, software utility names, and GUI labels. For example, **\qdesigns** directory, **D:** drive, and **chiptrip.gdf** file.
Italic Type with Initial Capital Letters	Indicate document titles. For example, *Stratix IV Design Guidelines*.
italic type	Indicates variables. For example, *n* + 1. Variable names are enclosed in angle brackets (< >). For example, *<file name>* and *<project name>***.pof** file.
Initial Capital Letters	Indicate keyboard keys and menu names. For example, the Delete key and the Options menu.
"Subheading Title"	Quotation marks indicate references to sections in a document and titles of Quartus II Help topics. For example, "Typographic Conventions."
`Courier type`	Indicates signal, port, register, bit, block, and primitive names. For example, `data1`, `tdi`, and `input`. The suffix n denotes an active-low signal. For example, `resetn`. Indicates command line commands and anything that must be typed exactly as it appears. For example, `c:\qdesigns\tutorial\chiptrip.gdf`. Also indicates sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword `SUBDESIGN`), and logic function names (for example, `TRI`).
↵	An angled arrow instructs you to press the Enter key.
1., 2., 3., and a., b., c., and so on	Numbered steps indicate a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■ ■ ■	Bullets indicate a list of items when the sequence of the items is not important.
	The hand points to information that requires special attention.
	The question mark directs you to a software help system with related information.
	The feet direct you to another document or website with related information.
	The multimedia icon directs you to a related multimedia presentation.
	A caution calls attention to a condition or possible situation that can damage or destroy the product or your work.
	A warning calls attention to a condition or possible situation that can cause you injury.
	The envelope links to the Email Subscription Management Center page of the Altera website, where you can sign up to receive update notifications for Altera documents.
	The feedback icon allows you to submit feedback to Altera about the document. Methods for collecting feedback vary as appropriate for each document.