RapidIO Intel FPGA IP User Guide

Updated for Intel® Quartus® Prime Design Suite: **20.3**

Attention: The RapidIO Intel FPGA IP is part of a product obsolescence and support discontinuation schedule.

For the schedule, refer to the Product Discontinuation Notice PDN2025.

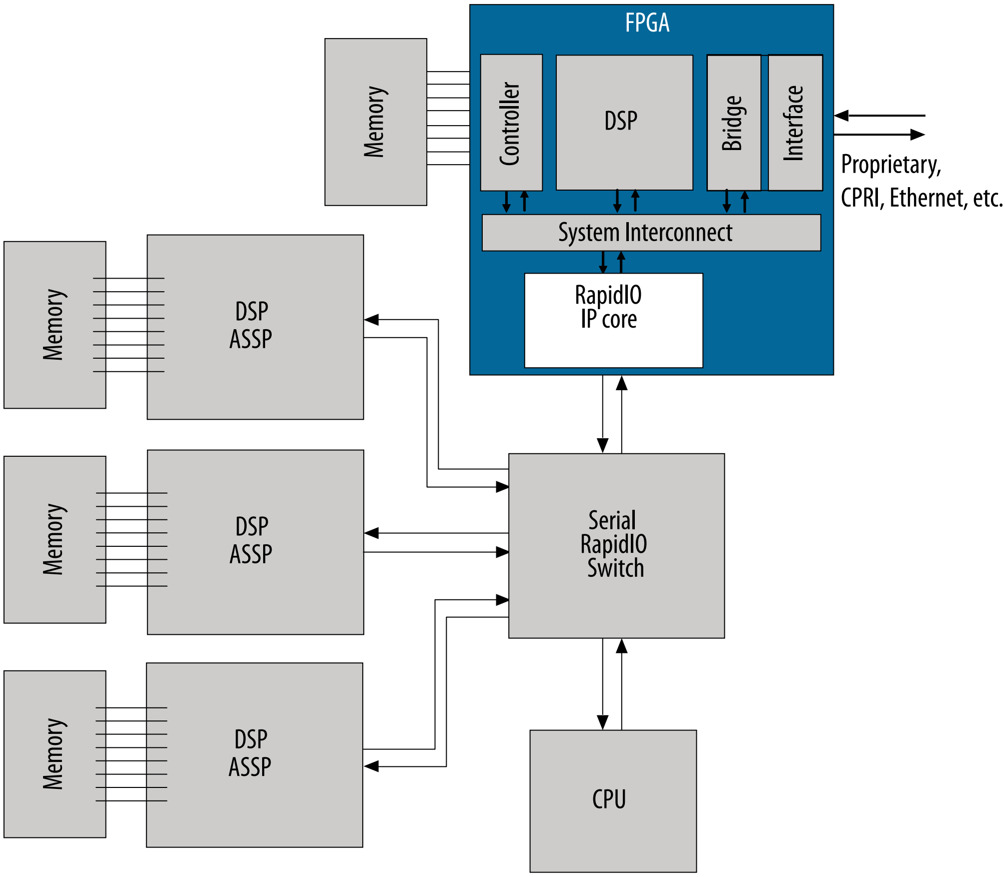
For new designs, Intel recommends that you use other IPs with equivalent functions. To see a list of available IPs, refer to the Intel® FPGA IP Portfolio web page.

Contents

1. About the RapidIO Intel FPGA IP Core

The RapidIO Intel FPGA IP core complies with the RapidIO v2.1 specification and targets high-performance, multicomputing, high-bandwidth, and coprocessing I/O applications. The RapidIO Interconnect is an open standard developed by the RapidIO Trade Association. It 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.

Figure 1. Typical RapidIO Application



1.1. Features

The RapidIO IP core has the following features:

- Compliant with *RapidIO Interconnect Specification, Revision 2.1, August 2009*, available from the RapidIO Trade Association website.
- 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.
- All RapidIO IP core variations include configuration of the high-speed transceivers on the device.
- All RapidIO IP core variations have a Transport layer.
- Physical layer features:
  - 1x/2x/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.
  - Automatic recovery from fatal errors.
  - 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.
- Platform Designer (Standard) support.
- IP functional simulation models for use in Intel®-supported VHDL and Verilog HDL simulators.
- Support for Intel FPGA IP Evaluation Mode.

Related Information

RapidIO Specification Webpage

1.1.1. 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

1.2. Device Family Support

The following are the device support level definitions for Intel FPGA IP cores:

Table 1. Intel FPGA IP Core Device Support Levels

FPGA 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.
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.

Table 2. Device Family Support

Device Family	Support Level
Arria® II GX	Final
Arria II GZ	Final
Arria V (GX, GT, GZ, SX, and ST)	Final
Intel Arria 10	Final
Cyclone® IV GX[1]	Final
Cyclone V (GX, GT, SX, and ST)	Final
Intel Cyclone 10 GX	Final
Stratix® IV	Final
Stratix IV GT	Final
Stratix V	Final
Other device families	No support

1.3. IP Core Verification

Before releasing a version of the RapidIO IP core, Intel runs comprehensive regression tests in the current version of the Intel Intel Quartus® Prime software. These tests use the parameter editor and the Platform Designer (Standard) system integration tool to create the instance files. These files are tested in simulation and hardware to confirm functionality.

Intel 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.

1.3.1. Simulation Testing

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

[1] The RapidIO IP core supports only the EP4CGX75, EP4CGX75, EP4CGX110, and EP4CGX150 Cyclone IV GX devices.

- ModelSim*
- VCS* in combination with the Synopsys Native Testbench (NTB)
- Xcelium*

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.

1.3.2. Hardware Testing

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

The hardware tests cover 1x, 2x, and 4x variations running at 1.25, 2.5, 3.125, and 5.0 Gbaud, and processing the following traffic types:

- NREADs of various size payloads—4 bytes to 256 bytes
- NWRITEs of various size payloads—4 bytes to 256 bytes
- NWRITE_Rs of several different size packets
- SWRITEs
- 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`

1.3.3. Interoperability Testing

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

Intel 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.

1.4. Performance and Resource Utilization

This section contains tables showing IP core variation size and performance examples.

The numbers of LEs, combinational ALUTs, ALMs, and primary logic registers are rounded up to the nearest 100.

Table 3. RapidIO IP Core Intel Arria 10 Resource Utilization

The listed results are obtained using the Intel Quartus Prime Standard Edition software v17.1 for an Intel Arria 10 (10AX115S1F45E1SG) device.

Device	Parameters			ALMs	Combinational ALUTs	Logic Registers	Memory Blocks (M20K) [2]
	Variation	Mode	Baud Rate (Gbaud)				
Intel Arria 10	Physical and Transport layers, I/O master and slave, and Maintenance master and slave	1x	5.00	12500	15000	17000	58
		2x	5.00	13300	15800	17700	52
		4x	3.125	12800	15400	17400	52

[2] M20K for Intel Arria 10, Intel Cyclone 10 GX, Stratix V, and Arria V GZ devices.

Table 4. RapidIO IP Core Intel Cyclone 10 GX Resource Utilization

The listed results are obtained using the Intel Quartus Prime Pro Edition software v18.0 for an Intel Cyclone 10 GX (10CX220YU484I5G) device.

Device	Parameters			ALMs	Combinational ALUTs	Logic Registers	Memory Blocks (M20K) [3][2]
	Variation	Mode	Baud Rate (Gbaud)				
Intel Cyclone 10 GX	Physical and Transport layers, I/O master and slave, and Maintenance master and slave	1x	5.00	12859	14855	17074	58
		2x	5.00	13272	15478	17660	52
		4x	3.125	13230	15478	17349	52

Table 5. RapidIO IP Core V-series FPGA Device Resource Utilization

The listed results are obtained using the Intel Quartus Prime Standard Edition software v17.1 for the following devices:

- Arria V GX (5AGXBB1D4F31C4)
- Arria V GZ (5AGZME1H2F35C3)
- Cyclone V (5CGXFC7C6F23C6)
- Stratix V (5SGXMA7H2F35C2)

Device	Parameters			ALMs	Combinational ALUTs	Logic Registers	Memory Blocks (M10K[4] or M20K[2])
	Variation	Mode	Baud Rate (Gbaud)				
Arria V GX	Physical and Transport layers, I/O master and slave, and Maintenance master and slave	1x	5.00	9700	13100	14600	113
		2x	3.125	11200	15500	17000	116
		4x		11000	15100	17700	116
Arria V GZ		1x	5.00	9700	13300	14700	63
		2x		11400	15100	17600	56
		4x		11300	15500	18000	63
Cyclone V GX		1x	3.125	9600	13700	14500	115
		2x		11200	15500	17000	116
		4x	2.5	11000	15500	17600	116
Stratix V GX		1x	5.00	9700	13300	14500	63
		2x		11300	15100	17500	57
		4x		11000	15000	17600	64

[3] M20K for Intel Arria 10, Intel Cyclone 10 GX, Stratix V, and Arria V GZ devices.

[4] M10K for Arria V, and Cyclone V devices.

Table 6. RapidIO IP Core Cyclone IV Resource Utilization

The listed results are obtained using the Intel Quartus Prime Standard Edition software v17.1 for a Cyclone IV GX (EP4CGX50CF23C6) device.

Device	Parameters			Combinational ALUTs	Logic Registers	Memory Blocks (M9K)[5]
	Variation	Mode	Baud Rate (Gbaud)			
Cyclone IV GX	Physical and Transport layers, I/O master and slave, and Maintenance master and slave	1×	3.125	22400	13600	123
		4×	1.25	25200	15600	123

Table 7. RapidIO IP Core Stratix IV and Legacy Arria Series Resource Utilization

The listed results are obtained using the Intel Quartus Prime Standard Edition software v17.1 for the following devices:

- Stratix IV GX (EP4SGX230DF29C2X)
- Arria II GX (EP2AGX5DF25C4)
- Arria II GZ (EP2AGZ225FF35C3)

Device	Parameters			ALMs	Combinational ALUTs	Logic Registers	Memory Blocks (M9K)[5]
	Variation	Mode	Baud Rate (Gbaud)				
Stratix IV GX	Physical and Transport layers, I/O master and slave, and Maintenance master and slave	1x	3.125	12000	13000	14000	44
		4x		13900	15100	16600	42
Arria II GX		1x	3.125	12700	12900	13800	115
		4x		14000	14200	16000	112
Arria II GZ		1x	5.00	11900	12700	13900	115
		4x	3.125	13700	15000	16400	115

1.5. Device Speed Grades

Following are the recommended device family speed grades for the supported link widths and internal clock frequencies. In all cases, Intel FPGA recommends that you set Intel Quartus Prime Analysis & Synthesis Optimization Technique to Speed.

[5] M9K for Cyclone IV GX, Stratix IV GX, Arria II GX, and Arria II GZ devices.

Table 8. Recommended Device Family Speed Grades for Newer Devices

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.

Device Family	Mode	Rate		1.25 Gbaud	2.5 Gbaud	3.125 Gbaud	5.0 Gbaud
		f_{MAX}	1x, 2x	31.25 MHz	62.50 MHz	78.125 MHz	125 MHz
			4x	62.5 MHz	125 MHz	156.25 MHz	250 MHz
Intel Arria 10	1x			-1, -2, -3	-1, -2, -3	-1, -2, -3	-1, -2
	2x			-1, -2, -3	-1, -2, -3	-1, -2, -3	-1, -2
	4x			-1, -2, -3	-1, -2, -3	-1, -2, -3	-1, -2
Arria V (GX, GT, SX, ST)	1x			C4, -5, C6	C4, -5, C6	C4, -5, C6	C4[6]
	2x			C4, -5, C6	C4, -5, C6	C4, -5, C6	C4, -5
	4x			C4, -5, C6	C4, -5	C4[6]	[7]
Arria V GZ	1x			-3, -4	-3, -4	-3, -4	-3
	2x			-3, -4	-3, -4	-3, -4	-3, -4
	4x			-3, -4	-3, -4	-3, -4	-3
Stratix V	1x			C1, -2, -3, -4	C1, -2, -3, -4	C1, -2, -3, -4	C1, -2, -3
	2x			C1, -2, -3, -4	C1, -2, -3, -4	C1, -2, -3, -4	C1, -2, -3, -4
	4x			C1, -2, -3, -4	C1, -2, -3, -4	C1, -2, -3, -4	C1, -2, -3[8]
Intel Cyclone 10 GX	1x			-5, -6	-5, -6	-5, -6	-5
	2x			-5, -6	-5, -6	-5, -6	-5
	4x			-5, -6	-5, -6	-5, -6	-5

continued...

[6] Some simple Arria V 1x variations with lane speed of 5.0 Gbaud, and some simple Arria V 4x 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.

[7] Not supported for this device family.

[8] Intel recommends that for designs that include a 4x 5.0 Gbaud RapidIO IP core variation and that target a -3 speed grade Stratix V device, you use multiple seeds in the Intel Quartus Prime 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 Logic Lock (Standard) regions may be necessary to meet timing, especially for more complex variations implemented in the largest devices.

Device Family	Mode	Rate		1.25 Gbaud	2.5 Gbaud	3.125 Gbaud	5.0 Gbaud
		f_{MAX}	1x, 2x	31.25 MHz	62.50 MHz	78.125 MHz	125 MHz
			4x	62.5 MHz	125 MHz	156.25 MHz	250 MHz
Cyclone V (GX, GT[9], SX, ST)	1x			C6, -7, C8	C6, -7, C8	C6, -7, C8	C7[10]
	2x			C6, -7	C6, -7	C6, -7	-7[11]
	4x			C6, -7, C8	C6, -7	[7]	[7]

Table 9. Recommended Device Family Speed Grades for Legacy Devices

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.

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 II GX		-4, -5, -6	-4, -5, -6	-4, -5, -6	[7]	-4, -5, -6	-4, -5	-4, -5	[7]
Arria II GZ		-3, -4	-3, -4	-3, -4	-3	-3, -4	-3, -4	-3, -4	[7]
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[12]
Cyclone IV GX[13]		-6, -7, -8	-6, -7, -8	-6, -7	[7]	-6, -7, -8	-6[14]	[7]	[7]

[9] Only the -7 speed grade is available for Cyclone V GT devices.

[10] The RapidIO IP core supports 1x 5.0 Gbaud variations that target the Cyclone V device family in speed grade C7 Cyclone V GT devices only. The RapidIO parameter editor does not warn you of this fact. You can generate a 1x 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, the Intel Quartus Prime software Analysis and Synthesis tool fails.

[11] The RapidIO IP core supports 2x 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 2x 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, the Intel Quartus Prime software Analysis and Synthesis tool fails.

[12] Intel recommends that for designs that include a 4x 5.0 Gbaud RapidIO IP core variation and that target a -3 speed grade Stratix IV GX device, you use multiple seeds in the Intel Quartus Prime 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 Logic Lock (Standard) regions may be necessary to meet timing, especially for more complex variations implemented in the largest devices.

[13] The RapidIO IP core supports only the EP4CGX50, EP4CGX75, EP4CGX110, and EP4CGX150 Cyclone IV GX devices.

[14] Some simple Cyclone IV GX 4x 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.

Related Information

Quartus Prime Standard Edition Handbook Volume 1: Design and Synthesis
For more information about how to apply the Speed settings.

1.6. Release Information

Intel FPGA IP versions match the Intel Quartus Prime Design Suite software versions until v19.1. Starting in Intel Quartus Prime Design Suite software version 19.2, Intel FPGA IP has a new versioning scheme.

The Intel FPGA IP version (X.Y.Z) number can change with each Intel Quartus Prime software version. A change in:

- X indicates a major revision of the IP. If you update the Intel Quartus Prime software, you must regenerate the IP.
- Y indicates the IP includes new features. Regenerate your IP to include these new features.
- Z indicates the IP includes minor changes. Regenerate your IP to include these changes.

Table 10. RapidIO Release Information

Item	Description
IP Version	19.2.0
Intel Quartus Prime Pro Edition	20.3
Intel Quartus Prime Standard Edition	18.1
Release Date	2020.09.28
Ordering Code	IP-RIOPHY

Intel verifies that the current version of the Intel Quartus Prime software compiles the previous version of each IP core. Any exceptions to this verification are reported in the release notes or errata. Intel does not verify compilation with IP core versions older than the previous release.

Related Information

- Intel FPGA IP Release Notes
- Knowledge Base Errata for RapidIO IP Core

2. Getting Started

You can customize the RapidIO IP core to support a wide variety of applications.

When you generate the IP core you can choose whether or not to generate a simulation model. If you generate a simulation model, Intel provides a Verilog testbench customized for your IP core variation. If you specify a VHDL simulation model, you must use a mixed-language simulator to run the testbench, or create your own VHDL-only simulation environment.

The following sections provide generic instructions and information for Intel FPGA IP cores. It explains how to install, parameterize, simulate, and initialize the RapidIO IP core.

Related Information

- Introduction to Intel FPGA IP Cores
  Provides general information about all Intel FPGA IP cores, including parameterizing, generating, upgrading, and simulating IP cores.
- Generating a Combined Simulator Setup Script
  Create simulation scripts that do not require manual updates for software or IP version upgrades.
- Project Management Best Practices
  Guidelines for efficient management and portability of your project and IP files.

2.1. Installing and Licensing Intel FPGA IP Cores

The Intel Quartus Prime software installation includes the Intel FPGA IP library. This library provides many useful IP cores for your production use without the need for an additional license. Some Intel FPGA IP cores require purchase of a separate license for production use. The Intel FPGA IP Evaluation Mode allows you to evaluate these licensed Intel FPGA IP cores in simulation and hardware, before deciding to purchase a full production IP core license. You only need to purchase a full production license for licensed Intel IP cores after you complete hardware testing and are ready to use the IP in production.

The Intel Quartus Prime software installs IP cores in the following locations by default:

Figure 2. IP Core Installation Path

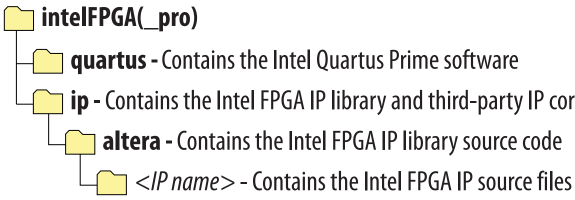

intelFPGA(_pro)
quartus - Contains the Intel Quartus Prime software
ip - Contains the Intel FPGA IP library and third-party IP cores
altera - Contains the Intel FPGA IP library source code
<IP name> - Contains the Intel FPGA IP source files

Table 11. IP Core Installation Locations

Location	Software	Platform
<*drive*>:\intelFPGA_pro\quartus\ip\altera	Intel Quartus Prime Pro Edition	Windows*
<*drive*>:\intelFPGA\quartus\ip\altera	Intel Quartus Prime Standard Edition	Windows
<*home directory*>:/intelFPGA_pro/quartus/ip/altera	Intel Quartus Prime Pro Edition	Linux*
<*home directory*>:/intelFPGA/quartus/ip/altera	Intel Quartus Prime Standard Edition	Linux

Note: The Intel Quartus Prime software does not support spaces in the installation path.

2.1.1. Intel FPGA IP Evaluation Mode

The free Intel FPGA IP Evaluation Mode allows you to evaluate licensed Intel FPGA IP cores in simulation and hardware before purchase. Intel FPGA IP Evaluation Mode supports the following evaluations without additional license:

- Simulate the behavior of a licensed Intel FPGA IP core in your system.
- Verify the functionality, size, and speed of the IP core quickly and easily.
- Generate time-limited device programming files for designs that include IP cores.
- Program a device with your IP core and verify your design in hardware.

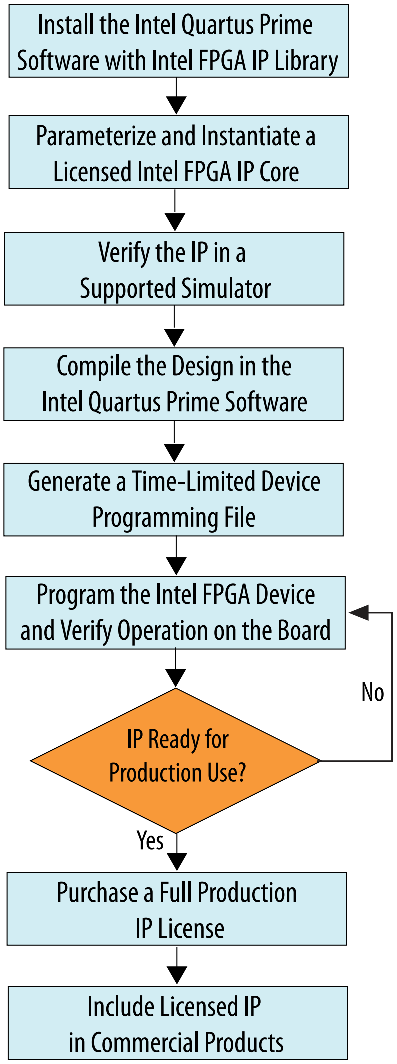
Intel FPGA IP Evaluation Mode supports the following operation modes:

- **Tethered**—Allows running the design containing the licensed Intel FPGA IP indefinitely with a connection between your board and the host computer. Tethered mode requires a serial joint test action group (JTAG) cable connected between the JTAG port on your board and the host computer, which is running the Intel Quartus Prime Programmer for the duration of the hardware evaluation period. The Programmer only requires a minimum installation of the Intel Quartus Prime software, and requires no Intel Quartus Prime license. The host computer controls the evaluation time by sending a periodic signal to the device via the JTAG port. If all licensed IP cores in the design support tethered mode, the evaluation time runs until any IP core evaluation expires. If all of the IP cores support unlimited evaluation time, the device does not time-out.
- **Untethered**—Allows running the design containing the licensed IP for a limited time. The IP core reverts to untethered mode if the device disconnects from the host computer running the Intel Quartus Prime software. The IP core also reverts to untethered mode if any other licensed IP core in the design does not support tethered mode.

When the evaluation time expires for any licensed Intel FPGA IP in the design, the design stops functioning. All IP cores that use the Intel FPGA IP Evaluation Mode time out simultaneously when any IP core in the design times out. When the evaluation time expires, you must reprogram the FPGA device before continuing hardware verification. To extend use of the IP core for production, purchase a full production license for the IP core.

You must purchase the license and generate a full production license key before you can generate an unrestricted device programming file. During Intel FPGA IP Evaluation Mode, the Compiler only generates a time-limited device programming file (<*project name*>_time_limited.sof) that expires at the time limit.

Figure 3. Intel FPGA IP Evaluation Mode Flow


Install the Intel Quartus Prime Software with Intel FPGA IP Library
Parameterize and Instantiate a Licensed Intel FPGA IP Core
Verify the IP in a Supported Simulator
Compile the Design in the Intel Quartus Prime Software
Generate a Time-Limited Device Programming File
Program the Intel FPGA Device and Verify Operation on the Board
IP Ready for Production Use?
No
Yes
Purchase a Full Production IP License
Include Licensed IP in Commercial Products

Note: Refer to each IP core's user guide for parameterization steps and implementation details.

Intel licenses IP cores on a per-seat, perpetual basis. The license fee includes first-year maintenance and support. You must renew the maintenance contract to receive updates, bug fixes, and technical support beyond the first year. You must purchase a full production license for Intel FPGA IP cores that require a production license, before generating programming files that you may use for an unlimited time. During Intel FPGA IP Evaluation Mode, the Compiler only generates a time-limited device programming file (<*project name*>`_time_limited.sof`) that expires at the time limit. To obtain your production license keys, visit the Self-Service Licensing Center.

The Intel FPGA Software License Agreements govern the installation and use of licensed IP cores, the Intel Quartus Prime design software, and all unlicensed IP cores.

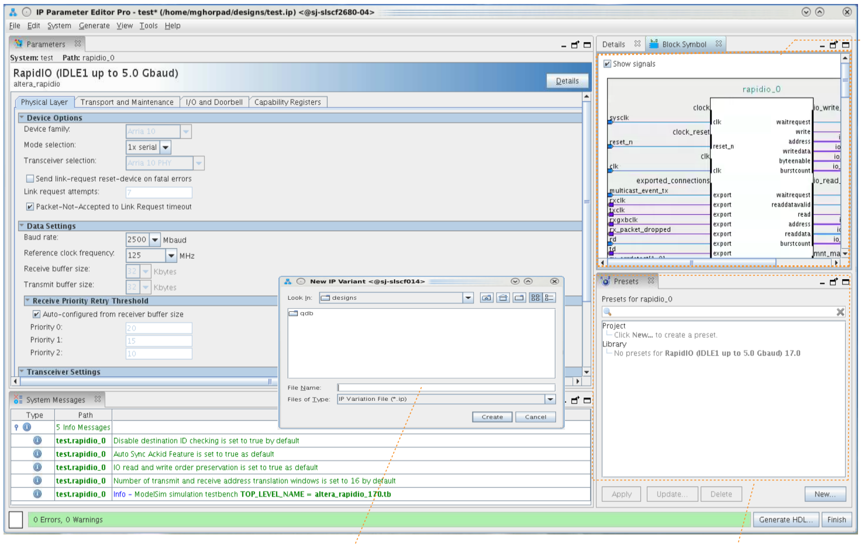
Related Information

- Intel FPGA Licensing Support Center
- Introduction to Intel FPGA Software Installation and Licensing

2.2. Generating IP Cores

You can quickly configure a custom IP variation in the parameter editor. Use the following steps to specify RapidIO IP core options and parameters in the Intel Quartus Prime software parameter editor:

Figure 4. IP Parameter Editor



IP Parameter Editor Pro - test* (/home/mghorpad/designs/test.ip) <@sj-slscf2680-04>

System: test Path: rapidio_0

RapidIO (IDLE1 up to 5.0 Gbaud)
altera_rapidio

Device Options
Device family: Arria 10
Mode selection: 1x serial
Transceiver selection: Arria 10 PHY
□ Send link-request reset-device on fatal errors
Link request attempts: 7
☑ Packet-Not-Accepted to Link Request timeout

Data Settings
Baud rate: 2500 Mbaud
Reference clock frequency: 125 MHz
Receive buffer size: 32 Kbytes
Transmit buffer size: 32 Kbytes

Receive Priority Retry Threshold
☑ Auto-configured from receiver buffer size
Priority 0: 20
Priority 1: 15
Priority 2: 10

New IP Variant <@sj-slscf014>
Look In: designs
File Name:
Files of Type: IP Variation File (*.ip)

Type	Path	
	5 Info Messages	
	test.rapidio_0	Disable destination ID checking is set to true by default
	test.rapidio_0	Auto Sync Ackid Feature is set to true as default
	test.rapidio_0	IO read and write order preservation is set to true as default
	test.rapidio_0	Number of transmit and receive address translation windows is set to 16 by default
	test.rapidio_0	Info - ModelSim simulation testbench TOP_LEVEL_NAME = altera_rapidio_170.tb

0 Errors, 0 Warnings

Presets for rapidio_0
Project
Click New... to create a preset.
Library
No presets for RapidIO (IDLE1 up to 5.0 Gbaud) 17.0

View IP Port and Parameter Details

Specify a Name for your IP Variation

Apply Preset Parameters for Specific Applications

1. In the Intel Quartus Prime Pro Edition software, click **File ➤ New Project Wizard** to create a new Intel Quartus Prime project, or **File ➤ Open Project** to open an existing Intel Quartus Prime project. The wizard prompts you to specify a device. In the Intel Quartus Prime Standard Edition software, this step is not required.
2. In the IP Catalog (**Tools ➤ IP Catalog**), locate and double-click **RapidIO (IDLE1 up to 5.0 Gbaud)** IP core to customize. The New IP Variation window appears.
3. Specify a top-level name for your custom IP variation. Do not include spaces in IP variation names or paths. The parameter editor saves the IP variation settings in a file with one of the following names:
   - `<your_ip>.ip` (When you generate Intel Arria 10 and Intel Cyclone 10 GX variations in the Intel Quartus Prime Pro Edition software.)
   - `<your_ip>.qsys` (When you generate any device variations in the Intel Quartus Prime Standard Edition software.)
4. Click **Create**/**OK**. The parameter editor appears.
5. Specify the parameters and options for your IP variation in the parameter editor, including one or more of the following:

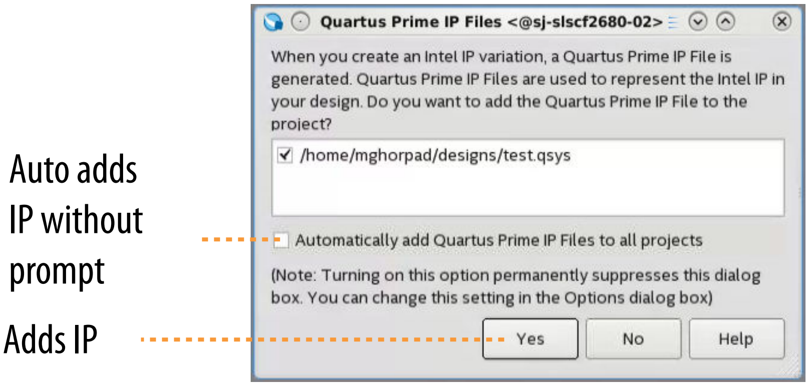
- Optionally select preset parameter values if provided for your IP core. Presets specify initial parameter values for specific applications.
- Specify parameters defining the IP core functionality, port configurations, and device-specific features.
- Specify options for processing the IP core files in other EDA tools.

6. Click **Generate HDL**. The **Generation** dialog box appears.
7. Specify output file generation options, and then click **Generate**. The software generates a testbench for your IP core when you create a simulation model. The synthesis and simulation files generate according to your specifications.

   Note: If you click **Generate ➤ Generate Testbench System**, and then click **Generate**, the Intel Quartus Prime software generates a testbench framework. However, the resulting testbench is composed of BFM stubs and does not exercise the RapidIO IP core in any meaningful way.

8. To generate an HDL instantiation template that you can copy and paste into your text editor, click **Generate ➤ Show Instantiation Template**.
9. Click **Finish**. Click **Yes** if prompted to add files representing the IP variation to your project. Optionally turn on the option to **Automatically add Intel Quartus Prime IP Files to All Projects**. Click **Project ➤ Add/Remove Files in Project** to add IP files at any time.

Figure 5. Adding IP Files to Project



Quartus Prime IP Files <@sj-slscf2680-02>

When you create an Intel IP variation, a Quartus Prime IP File is generated. Quartus Prime IP Files are used to represent the Intel IP in your design. Do you want to add the Quartus Prime IP File to the project?

☑ /home/mghorpad/designs/test.qsys

□ Automatically add Quartus Prime IP Files to all projects

(Note: Turning on this option permanently suppresses this dialog box. You can change this setting in the Options dialog box)

Note: For Intel Arria 10 and Intel Cyclone 10 GX devices, the generated `.qsys` file must be added to your project to represent IP and Platform Designer systems. For devices released prior to Intel Arria 10 and Intel Cyclone 10 GX devices, the generated `.qip` and `.sip` files must be added to your project for IP and Platform Designer systems.

10. After generating and instantiating your IP variation, make appropriate pin assignments to connect ports.

    Note: Some IP cores generate different HDL implementations according to the IP core parameters. The underlying RTL of these IP cores contains a unique hash code that prevents module name collisions between different variations of the IP core. This unique code remains consistent, given the same IP settings and software version during IP generation. This unique code can change if you edit the IP core's parameters or upgrade the IP core version. To avoid dependency on these unique codes in your simulation environment, refer to *Generating a Combined Simulator Setup Script.*

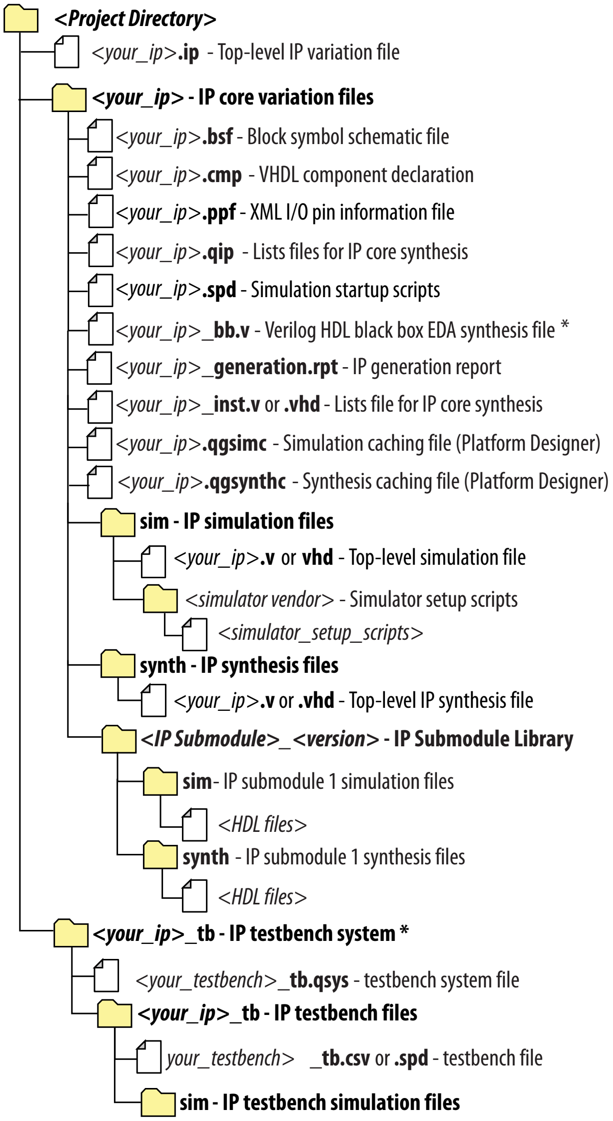
Related Information

- Intel FPGA IP Release Notes
- Generating a Combined Simulator Setup Script

2.2.1. IP Core Generation Output (Intel Quartus Prime Pro Edition)

The Intel Quartus Prime software generates the following output file structure for individual IP cores that are not part of a Platform Designer system.

Figure 6. Individual IP Core Generation Output (Intel Quartus Prime Pro Edition)


<Project Directory>
<your_ip>.ip - Top-level IP variation file
<your_ip> - IP core variation files
<your_ip>.bsf - Block symbol schematic file
<your_ip>.cmp - VHDL component declaration
<your_ip>.ppf - XML I/O pin information file
<your_ip>.qip - Lists files for IP core synthesis
<your_ip>.spd - Simulation startup scripts
<your_ip>_bb.v - Verilog HDL black box EDA synthesis file *
<your_ip>_generation.rpt - IP generation report
<your_ip>_inst.v or .vhd - Lists file for IP core synthesis
<your_ip>.qgsimc - Simulation caching file (Platform Designer)
<your_ip>.qgsynthc - Synthesis caching file (Platform Designer)
sim - IP simulation files
<your_ip>.v or vhd - Top-level simulation file
<simulator vendor> - Simulator setup scripts
<simulator_setup_scripts>
synth - IP synthesis files
<your_ip>.v or .vhd - Top-level IP synthesis file
<IP Submodule>_<version> - IP Submodule Library
sim- IP submodule 1 simulation files
<HDL files>
synth - IP submodule 1 synthesis files
<HDL files>
<your_ip>_tb - IP testbench system *
<your_testbench>_tb.qsys - testbench system file
<your_ip>_tb - IP testbench files
your_testbench> _tb.csv or .spd - testbench file
sim - IP testbench simulation files

* If supported and enabled for your IP core variation.

Table 12. Output Files of Intel FPGA IP Generation

File Name	Description
`<your_ip>.ip`	Top-level IP variation file that contains the parameterization of an IP core in your project. If the IP variation is part of a Platform Designer system, the parameter editor also generates a `.qsys` file.
`<your_ip>.cmp`	The VHDL Component Declaration (`.cmp`) file is a text file that contains local generic and port definitions that you use in VHDL design files.
`<your_ip>_generation.rpt`	IP or Platform Designer generation log file. Displays a summary of the messages during IP generation.
`<your_ip>.qgsimc` (Platform Designer systems only)	Simulation caching file that compares the `.qsys` and `.ip` files with the current parameterization of the Platform Designer system and IP core. This comparison determines if Platform Designer can skip regeneration of the HDL.
`<your_ip>.qgsynth` (Platform Designer systems only)	Synthesis caching file that compares the `.qsys` and `.ip` files with the current parameterization of the Platform Designer system and IP core. This comparison determines if Platform Designer can skip regeneration of the HDL.
`<your_ip>.qip`	Contains all information to integrate and compile the IP component.
`<your_ip>.csv`	Contains information about the upgrade status of the IP component.
`<your_ip>.bsf`	A symbol representation of the IP variation for use in Block Diagram Files (`.bdf`).
`<your_ip>.spd`	Input file that `ip-make-simscript` requires to generate simulation scripts. The `.spd` file contains a list of files you generate for simulation, along with information about memories that you initialize.
`<your_ip>.ppf`	The Pin Planner File (`.ppf`) stores the port and node assignments for IP components you create for use with the Pin Planner.
`<your_ip>_bb.v`	Use the Verilog blackbox (`_bb.v`) file as an empty module declaration for use as a blackbox.
`<your_ip>_inst.v` or `_inst.vhd`	HDL example instantiation template. Copy and paste the contents of this file into your HDL file to instantiate the IP variation.
`<your_ip>.regmap`	If the IP contains register information, the Intel Quartus Prime software generates the `.regmap` file. The `.regmap` file describes the register map information of master and slave interfaces. This file complements the `.sopcinfo` file by providing more detailed register information about the system. This file enables register display views and user customizable statistics in System Console.
`<your_ip>.svd`	Allows HPS System Debug tools to view the register maps of peripherals that connect to HPS within a Platform Designer system. During synthesis, the Intel Quartus Prime software stores the `.svd` files for slave interface visible to the System Console masters in the `.sof` file in the debug session. System Console reads this section, which Platform Designer queries for register map information. For system slaves, Platform Designer accesses the registers by name.

`<your_ip>.v`
`<your_ip>.vhd`
	HDL files that instantiate each submodule or child IP core for synthesis or simulation.

`mentor/`	Contains a `msim_setup.tcl` script to set up and run a ModelSim simulation.
`aldec/`	Contains a Riviera-PRO* script `rivierapro_setup.tcl` to setup and run a simulation.
`/synopsys/vcs` `/synopsys/vcsmx`	Contains a shell script `vcs_setup.sh` to set up and run a VCS simulation. Contains a shell script `vcsmx_setup.sh` and `synopsys_sim.setup` file to set up and run a VCS MX simulation.

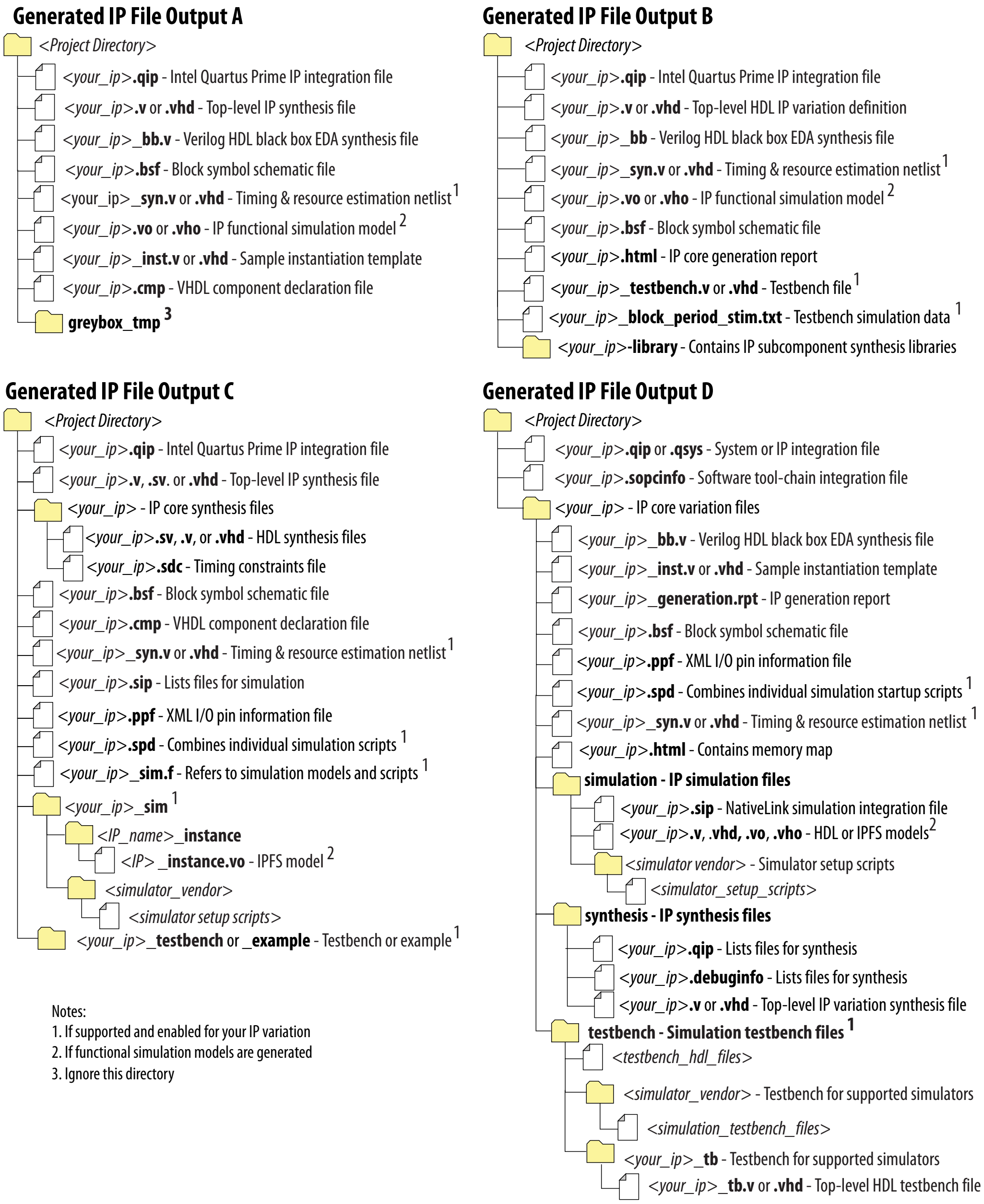
continued...

File Name	Description
/cadence	Contains a shell script ncsim_setup.sh and other setup files to set up and run an NCSim simulation.
/xcelium	Contains an Xcelium Parallel simulator shell script xcelium_setup.sh and other setup files to set up and run a simulation.
/submodules	Contains HDL files for the IP core submodule.
<IP submodule>/	Platform Designer generates /synth and /sim sub-directories for each IP submodule directory that Platform Designer generates.

2.3. IP Core Generation Output (Intel Quartus Prime Standard Edition)

The Intel Quartus Prime Standard Edition legacy parameter editors generate one of the following output file structures for individual IP cores:

Figure 7. IP Core Generated Files (Legacy Parameter Editors)


Generated IP File Output A
<Project Directory>
<your_ip>.qip - Intel Quartus Prime IP integration file
<your_ip>.v or .vhd - Top-level IP synthesis file
<your_ip>_bb.v - Verilog HDL black box EDA synthesis file
<your_ip>.bsf - Block symbol schematic file
<your_ip>_syn.v or .vhd - Timing & resource estimation netlist 1
<your_ip>.vo or .vho - IP functional simulation model 2
<your_ip>_inst.v or .vhd - Sample instantiation template
<your_ip>.cmp - VHDL component declaration file
greybox_tmp 3
Generated IP File Output B
<Project Directory>
<your_ip>.qip - Intel Quartus Prime IP integration file
<your_ip>.v or .vhd - Top-level HDL IP variation definition
<your_ip>_bb - Verilog HDL black box EDA synthesis file
<your_ip>_syn.v or .vhd - Timing & resource estimation netlist 1
<your_ip>.vo or .vho - IP functional simulation model 2
<your_ip>.bsf - Block symbol schematic file
<your_ip>.html - IP core generation report
<your_ip>_testbench.v or .vhd - Testbench file 1
<your_ip>_block_period_stim.txt - Testbench simulation data 1
<your_ip>-library - Contains IP subcomponent synthesis libraries
Generated IP File Output C
<Project Directory>
<your_ip>.qip - Intel Quartus Prime IP integration file
<your_ip>.v, .sv. or .vhd - Top-level IP synthesis file
<your_ip> - IP core synthesis files
<your_ip>.sv, .v, or .vhd - HDL synthesis files
<your_ip>.sdc - Timing constraints file
<your_ip>.bsf - Block symbol schematic file
<your_ip>.cmp - VHDL component declaration file
<your_ip>_syn.v or .vhd - Timing & resource estimation netlist 1
<your_ip>.sip - Lists files for simulation
<your_ip>.ppf - XML I/O pin information file
<your_ip>.spd - Combines individual simulation scripts 1
<your_ip>_sim.f - Refers to simulation models and scripts 1
<your_ip>_sim 1
<IP_name>_instance
<IP>_instance.vo - IPFS model 2
<simulator_vendor>
<simulator setup scripts>
<your_ip>_testbench or _example - Testbench or example 1
Generated IP File Output D
<Project Directory>
<your_ip>.qip or .qsys - System or IP integration file
<your_ip>.sopcinfo - Software tool-chain integration file
<your_ip> - IP core variation files
<your_ip>_bb.v - Verilog HDL black box EDA synthesis file
<your_ip>_inst.v or .vhd - Sample instantiation template
<your_ip>_generation.rpt - IP generation report
<your_ip>.bsf - Block symbol schematic file
<your_ip>.ppf - XML I/O pin information file
<your_ip>.spd - Combines individual simulation startup scripts 1
<your_ip>_syn.v or .vhd - Timing & resource estimation netlist 1
<your_ip>.html - Contains memory map
simulation - IP simulation files
<your_ip>.sip - NativeLink simulation integration file
<your_ip>.v, .vhd, .vo, .vho - HDL or IPFS models 2
<simulator vendor> - Simulator setup scripts
<simulator_setup_scripts>
synthesis - IP synthesis files
<your_ip>.qip - Lists files for synthesis
<your_ip>.debuginfo - Lists files for synthesis
<your_ip>.v or .vhd - Top-level IP variation synthesis file
testbench - Simulation testbench files 1
<testbench_hdl_files>
<simulator_vendor> - Testbench for supported simulators
<simulation_testbench_files>
<your_ip>_tb - Testbench for supported simulators
<your_ip>_tb.v or .vhd - Top-level HDL testbench file
Notes:
1. If supported and enabled for your IP variation
2. If functional simulation models are generated
3. Ignore this directory

2.4. RapidIO IP Core Testbench Files

The RapidIO IP core testbench is generated when you create a simulation model of the IP core.

For Intel Arria 10 and Intel Cyclone 10 GX variations:

- The testbench script appears in `<your_ip>/sim/<vendor>`.
- The testbench and simulation files appear in `<your_ip>/altera_rapidio_<version>/sim/tb`.

For variations other than Intel Arria 10 and Intel Cyclone 10 GX:

- The testbench script appears in `<your_ip>/simulation/<vendor>`.
- The testbench and simulation files appear in `<your_ip>/simulation/submodules`.
- The main testbench file is `<your_ip>/simulation/submodules/<your_ip>_rapidio_0_tb.v`

The RapidIO IP core does not generate an example design. The static design example included in the RapidIO installation directory does not function correctly with Intel Arria 10 and Intel Cyclone 10 GX IP core variations.

2.5. Simulating IP Cores

The Intel Quartus Prime software supports RTL- and gate-level design simulation of Intel FPGA IP cores in supported EDA simulators[15]. Simulation involves setting up your simulator working environment, compiling simulation model libraries, and running your simulation.

You can use the functional simulation model and the testbench or example design generated with your IP core for simulation. The functional simulation model and testbench files are generated in a project subdirectory. This directory may also include scripts to compile and run the testbench. For a complete list of models or libraries required to simulate your IP core, refer to the scripts generated with the testbench. You can use the Intel Quartus Prime NativeLink feature to automatically generate simulation files and scripts. NativeLink launches your preferred simulator from within the Intel Quartus Prime software.

Note: The Intel Quartus Prime Pro Edition software does not support NativeLink RTL simulation.

Related Information

Simulating Intel FPGA IP Cores

2.5.1. Simulating the Testbench with the ModelSim Simulator

To simulate the RapidIO IP core testbench using the Mentor Graphics ModelSim simulator, perform the following steps:

1. Start the ModelSim simulator.
2. In ModelSim, change directory to the directory where the testbench simulation script is located:

[15] Aldec Riviera simulator is not supported for this IP core.

- For Intel Arria 10 and Intel Cyclone 10 GX variations, change directory to `<`*`your_ip`*`>/sim/mentor`.
- For variations other than Intel Arria 10 and Intel Cyclone 10 GX, change directory to `<`*`your_ip`*`>/simulation/mentor`.

3. To set up the required libraries, compile the generated simulation model, and exercise the simulation model with the provided testbench, type one of the following sets of commands:

   a. For Intel Arria 10 variations using the Intel Quartus Prime Standard Edition software, type the following commands:

   ```
   do msim_setup.tcl
   set TOP_LEVEL_NAME <your_ip>_altera_rapidio_<version>.tb
   ld
   run -all
   ```

   For example:

   ```
   set TOP_LEVEL_NAME my_srio_altera_rapidio_171.tb
   ```

   where "my_srio" is the IP variation.

   b. For Intel Arria 10 and Intel Cyclone 10 GX variations using the Intel Quartus Prime Pro Edition software, type the following commands:

   ```
   do msim_setup.tcl
   set TOP_LEVEL_NAME altera_rapidio_<version>.tb
   ld
   run -all
   ```

   For example:

   ```
   set TOP_LEVEL_NAME altera_rapidio_171.tb
   ```

   c. For variations other than Intel Arria 10 and Intel Cyclone 10 GX, type the following commands:

   ```
   do msim_setup.tcl
   set TOP_LEVEL_NAME rapidio_0.tb
   ld
   run -all
   ```

   Simulation of the testbench might take few minutes. The testbench displays a TESTBENCH_PASSED message after completion.

2.5.2. Simulating the Testbench with the VCS Simulator

To simulate the RapidIO IP core testbench using the Synopsys VCS simulator, perform the following steps:

1. Change directory to the directory where the testbench simulation script is located:

   - For Intel Arria 10 and Intel Cyclone 10 GX variations, change directory to <*your_ip*>/sim/synopsys.
   - For variations other than Intel Arria 10 and Intel Cyclone 10 GX, change directory to <*your_ip*>/simulation/synopsys/vcs.

2. Type the following command to set up the required libraries, compile the generated IP functional model, and exercise the simulation model with the provided testbench:

```
sh vcs_setup.sh TOP_LEVEL_NAME="tb"
./simv
```

2.5.3. Simulating the Testbench with the Xcelium Simulator

This simulator is only available in Intel Quartus Prime Pro Edition. To simulate the RapidIO IP core testbench using the Cadence Xcelium simulator, perform the following steps:

1. Change directory to the directory where the testbench simulation script is located:
   - For Intel Arria 10 and Intel Cyclone 10 GX variations, change directory to <*your_ip*>/sim/xcelium.
2. Type the following command to set up the required libraries, compile the generated IP functional model, and exercise the simulation model with the provided testbench:

```
sh xcelium_setup.sh TOP_LEVEL_NAME="altera_rapidio_<version>.tb"
USER_DEFINED_SIM_OPTIONS="-input\\"@run\2ms\;\exit\""
```

2.6. Integrating Your IP Core in Your Design

When you integrate your IP core instance in your design, you must pay attention to some additional requirements. If you generate your IP core from the Platform Designer (Standard) IP catalog and build your design in Platform Designer (Standard), you can perform these steps in Platform Designer (Standard). If you generate your IP core directly from the Intel Quartus Prime IP catalog, you must implement these steps manually in your design.

2.6.1. Calibration Clock

For Arria II GX, Arria II GZ, Cyclone IV 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.

2.6.2. Dynamic Transceiver Reconfiguration Controller

RapidIO IP core variations that target an Intel Arria 10 and Intel Cyclone 10 GX devices include a reconfiguration controller block and do not require an external reconfiguration controller. All other RapidIO IP core variations require an external reconfiguration controller to function correctly in hardware.

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. You must connect it as specified in 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, and connect it to the RapidIO IP core dynamic reconfiguration signals `reconfig_fromgxb` and `reconfig_togxb`. This block supports offset cancellation. The design compiles without the Transceiver Reconfiguration Controller, but it cannot function correctly in hardware.

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. An informational message in the RapidIO parameter editor tells you the required number of reconfiguration interfaces.

Related Information

- Arria II Device Handbook
- Cyclone IV Device Handbook
- Stratix IV Device Handbook
- V-Series Transceiver PHY IP Core User Guide
  For information about the Transceiver Reconfiguration Controller.
- Transceiver Signals on page 124
  For descriptions of the reconfig_togxb and reconfig_fromgxb signals.

2.6.3. Transceiver Settings

If you want to modify the high-speed transceiver settings in an Arria II GX, Arria II GZ, Cyclone IV GX, or Stratix IV GX variation, you must first generate the IP core and then edit the existing ALTGX megafunction in the Intel Quartus Prime software. Regenerating overwrites the changes.

The ALTGX megafunction that is generated in your RapidIO IP core is not accessible through Platform Designer (Standard). You must edit this megafunction using the Intel Quartus Prime software.

If your RapidIO IP core targets an Arria V, Cyclone V, or Stratix V device, Intel recommends you do not modify the default transceiver settings configured in the Custom PHY IP core instance generated with the RapidIO IP core.

If your RapidIO IP core targets Intel Arria 10 and Intel Cyclone 10 GX devices, Intel recommends you do not modify the default transceiver settings configured in the Intel Arria 10 Native PHY and the Intel Cyclone 10 GX Transceiver Native PHY IP cores.

2.6.4. Adding Transceiver Analog Settings for Arria II GX, Arria II GZ, and Stratix IV GX Variations

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 these instructions:

1. In the Intel Quartus Prime window, on the Assignments menu, click **Assignment Editor**.
2. In the **<<new>>** cell in the **To** column, type the top-level signal name for your RapidIO IP core instance `td` signal.
3. Double-click in the **Assignment Name** column and click **I/O Standard**.
4. Double-click in the **Value** column and click your standard (for example, **1.5-V PCML)**.
5. In the **<<new>>** row, repeat steps **2** to **4** for your RapidIO IP core instance `rd` signal.

2.6.5. External Transceiver PLL

RapidIO IP cores that target Intel Arria 10 and Intel Cyclone 10 GX devices require an external TX transceiver PLL to compile and to function correctly in hardware. You must instantiate and connect this IP core to the RapidIO IP core.

You can create an external transceiver PLL from the IP Catalog. Select the ATX PLL IP core or the fPLL IP core. In the PLL parameter editor, set the following parameter values:

- Set **PLL output frequency** to one half the value you select for the **Baud rate** parameter in the RapidIO parameter editor. The transceiver performs dual edge clocking, using both the rising and falling edges of the input clock from the PLL. Therefore, this PLL output frequency setting supports the customer-selected maximum data rate on the RapidIO link.
- Set PLL reference clock frequency to the value you select for the **Reference clock frequency** parameter in the RapidIO parameter editor.
- Turn on **Include Master Clock Generation Block.**
- Turn on **Enable bonding clock output ports.**
- Set **PMA interface width** to **20**.

When you generate a RapidIO IP core, the Quartus Prime software also generates the HDL code for an ATX PLL, in the following file: <your_ip>/ `altera_rapidio_<version>/synth/` `<your_ip>_altera_rapidio_<version>_<random_string>.v/.vhd.`[16]

However, the HDL code for the RapidIO IP core does not instantiate the ATX PLL. If you choose to use the ATX PLL provided with the RapidIO IP core, you must instantiate and connect the ATX PLL instance with the RapidIO IP core in user logic.

[16] For Intel Arria 10 and Intel Cyclone 10 GX devices, please refer to `<`*`your_ip`*`>_generation.rpt` file to get the filename for ATX PLL HDL code, listed in the line: `ATX_PLL_wrapper_name:` *`<ATX PLL name>`*.

You must connect the TX PLL IP core to the RapidIO IP core according to the following rules.

Table 13. External Transceiver TX PLL Connections to RapidIO IP Core

Signal	Direction	Connection Requirements
pll_refclk0	Input	Drive the PLL pll_refclk0 input port and the RapidIO IP core reference clock clk signal from the same clock source. The minimum allowed frequency for the pll_refclk0 clock in Intel Arria 10 and Intel Cyclone 10 GX ATX PLL is 100 MHz.
tx_bonding_clocks[(6 x <number of lanes>)-1:0]	Output	Connect tx_bonding_clocks[6n+5:6n] to the tx_bonding_clocks_chN input bus of transceiver channel N, for each transceiver channel N that connects to the RapidIO link. The transceiver channel input ports are RapidIO IP core input ports.

To see how to configure and connect a TX PLL IP core to the other system components, such as the external reset controller, refer to the cleartext testbench files.

Related Information

- Testbench on page 167
  For more information on how to configure and connect a TX PLL IP Core to other system components, such as the external reset controller.
- Intel Arria 10 Transceiver PHY User Guide
  For information about the connection requirements and flexibility.
- Intel Cyclone 10 GX Transceiver PHY User Guide
  For information about the connection requirements and flexibility.

2.6.6. Transceiver PHY Reset Controller for Intel Arria 10 and Intel Cyclone 10 GX Variations

You must add a Transceiver PHY Reset Controller IP core to your design, and connect it to the RapidIO IP core reset signals. This block implements a reset sequence that resets the device transceivers correctly.

In the Transceiver PHY Reset Controller parameter editor, you must perform the following for compatibility with the RapidIO IP core:

- Leave unchecked the **Use separate RX reset per channel** option.

When you generate a RapidIO IP core that target Intel Arria 10 and Intel Cyclone 10 GX devices, the Intel Quartus Prime software generates the HDL code for the Transceiver PHY Reset Controller in the following file: <your_ip>/altera_rapidio_<version>/synth/<your_ip>_altera_rapidio_<version>_<random_string>.v/.vhd[17]

Related Information

- Intel Arria 10 Transceiver PHY User Guide

[17] For Intel Arria 10 and Intel Cyclone 10 GX devices, please refer to *<your_ip>*_generation.rpt file to get the filename for Transceiver PHY Reset Controller HDL code, listed in the line: PHY_Reset_Controller_wrapper_name: *<PHY Reset Controller name>*

- Intel Cyclone 10 GX Transceiver PHY User Guide

2.7. Specifying Timing Constraints

For variations other than Intel Arria 10 and Intel Cyclone 10 GX, Intel provides constraint files in Tcl format that you must apply to ensure that the RapidIO IP core meets design timing requirements.

Note: Constraints are not set automatically. You must run the Tcl constraint script to apply the constraints.

To use the generated constraint files, follow these steps:

1. Open your project in the Intel Quartus Prime software.
2. On the View menu, point to **Utility Windows** and then click **Tcl Console**.
3. Source the generated constraint file by typing the following command at the Tcl console command prompt:

   ```
   source <variation_name>/synthesis/submodules/
   <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 Intel Quartus Prime project.

   If you rename any clocks in Platform Designer (Standard), you require the -ref_clk_name, -sys_clk_name, -phy_mgmt_clk, and -patch_sdc command-line options specified.

The script automatically constrains the system clocks and the reference clock based on the data rate chosen. For supported transceivers, Intel 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 Intel-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 14. 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 `clk`. If, in Platform Designer (Standard), you rename this clock or you connect the reference clock port of the IP core to a clock named something other than `clk`, 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 `sysclk`. If, in Platform Designer (Standard), you rename this clock or connect it to a clock named something other than `sysclk`, 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, in Platform Designer (Standard), you rename this clock or you connect it to a clock named something other than <variation>`_phy_mgmt_clk`, 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.

Related Information

- Timing Analysis Overview
  For more information about timing analysis.
- Quartus Prime Help
  For more information about timing analyzers.

2.8. Compiling the Full Design and Programming the FPGA

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

Note: Before compiling your design in the Intel Quartus Prime software, you must apply constraints.

Related Information

- Specifying Timing Constraints on page 32
  More information on specifying constraints in the Quartus Prime software.
- Quartus Prime Incremental Compilation for Hierarchical and Team-Based Design
  More information about compiling your design.

- Quartus Prime Standard Edition Handbook Volume 3: Verification
  More information about programming the device.

2.9. 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.

2.9.1. Clock and Signal Requirements for Arria V, Cyclone V, and Stratix V Variations

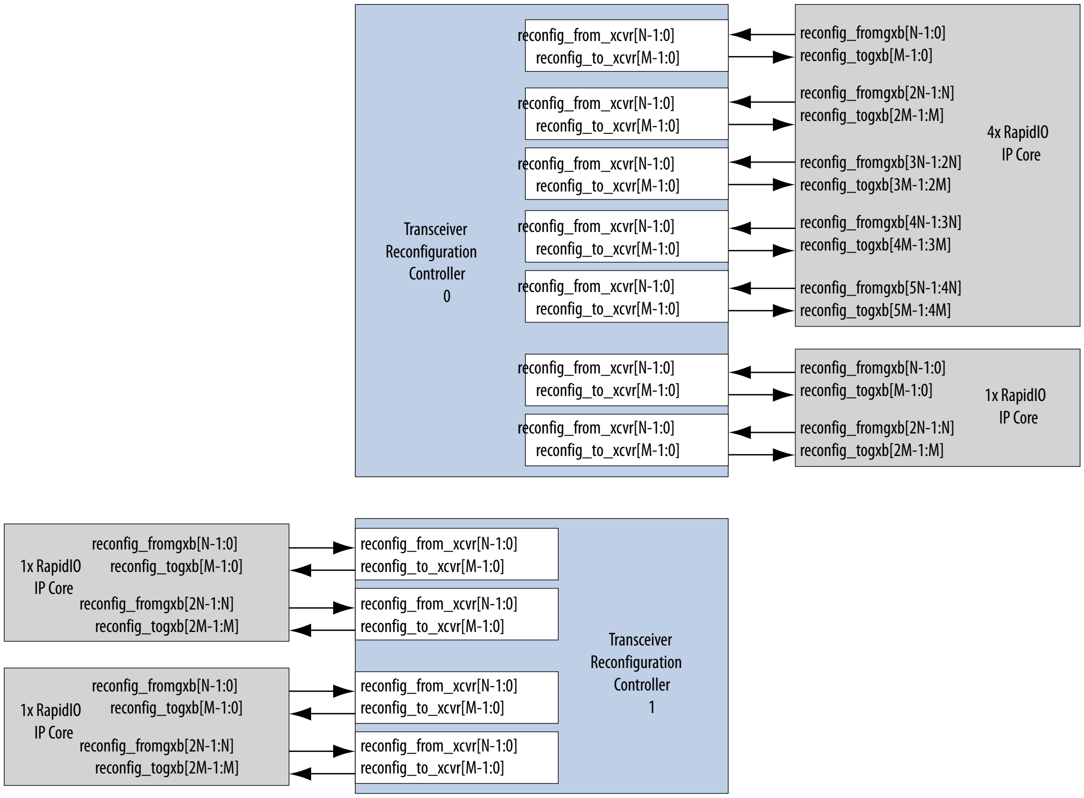
When your design targets an Arria V, Cyclone V, or Stratix V device, the transceivers are configured with the Custom PHY IP core. When your design contains multiple RapidIO IP cores, the Intel Quartus Prime Fitter handles the merge of multiple Custom PHY IP cores in the same transceiver block automatically. 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.

In the following example, the Transceiver Reconfiguration Controller 0 has seven reconfiguration interfaces, and 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 8. Example Connections Between Two Transceiver Reconfiguration Controllers and Four RapidIO IP Cores



To enable the Intel Quartus Prime 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.

Related Information

- Transceiver Reconfiguration Controller IP Core Overview
  More information about the Transceiver Reconfiguration Controller.
- Transceiver Signals on page 124
  For information about the reconfig_fromgxb and reconfig_togxb signals that connect a single RapidIO IP core.

2.9.2. Clock and Signal Requirements for Arria II GX, Arria II GZ, Cyclone IV GX, and Stratix IV GX Variations

RapidIO IP cores that target an Arria II GX, Arria II GZ, Cyclone IV GX, or Stratix IV GX device all instantiate an ALTGX transceiver megafunction to configure the device transceivers. When your design contains multiple IP cores that use the ALTGX megafunction, you must ensure that the `cal_blk_clk` and `gxb_powerdown` input signals are connected properly.

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 megafunction) is driven by the same calibration clock source.

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 megafunction.

To successfully combine multiple high-speed transceiver channels in the same transceiver block, they must have the same dynamic reconfiguration setting. If two IP cores implement dynamic reconfiguration in the same transceiver block of an Arria II GX, Arria II GZ, Cyclone IV 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.

2.9.3. Correcting the Synopsys Design Constraints File to Distinguish RapidIO IP Core Instances

When you instantiate multiple RapidIO IP core instances in your design, you must modify the Synopsys Design Constraints File (`.sdc`) to repeat the `create_generated_clock` statements for each IP core instance. The statements must include the full name of the variation in the clock names.

If you do not do this, the source and destination clocks each have multiple matches; the `rxclk` and `clk_div_by_2` filters match the relevant clocks in all of the IP core instances.

2.9.4. Sourcing Multiple Tcl Scripts for Variations other than Intel Arria 10 and Intel Cyclone 10 GX

If you use Intel-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

Note: 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`

3. Parameter Settings

You customize the RapidIO IP core by specifying parameters in the RapidIO parameter editor, which you access from the IP Catalog.

This chapter describes the parameters and how they affect the behavior of the IP core.

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**

3.1. Physical Layer Settings

The **Physical Layer** tab defines the characteristics of the Physical layer based on these categories: **Device Options**, **Data Settings**, **Receive Priority Retry Threshold**, and **Transceiver Settings**[18].

3.1.1. Device Options

Device Options comprise the following configuration options:

- Mode selection
- Transceiver selection
- Automatically synchronize transmitted ackID
- Send link-request reset-device on fatal errors
- Link-request attempts
- Packet-Not-Accepted to Link Request timeout

Mode Selection

Mode selection allows you to specify a **1x serial, 2x serial,** or **4x serial** port consisting of one-, two-, or four-lane high-speed data serialization and deserialization.

The 2x mode is available only in variations that target an Intel Arria 10, Arria V, Intel Cyclone 10 GX, Cyclone V, or Stratix V device.

[18] Only available in IP core variations that target Intel Arria 10 and Intel Cyclone 10 GX devices.

The 2x and 4x variations do not support fallback to 1x or 2x mode. You must know whether the IP core has a 1x, 2x, or 4x link partner and configure the FPGA accordingly. If fallback is required, the FPGA can be programmed with a 2x or 4x variation by default and then reprogrammed to a 1x (or 2x) configuration under system control after failure to synchronize in the original mode.

Transceiver Selection

The **Transceiver selection** parameter value is determined by the device family your IP core targets.Although this parameter appears in the parameter editor, you cannot modify its value.

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.

Note: This parameter is not available for variations that target Intel Arria 10 and Intel Cyclone 10 GX devices. In Intel Arria 10 and Intel Cyclone 10 GX variations, this option is turned on internally and cannot be modified.

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.

Note: This parameter is not available for variations that target Intel Arria 10 and Intel Cyclone 10 GX devices. In Intel Arria 10 and Intel Cyclone 10 GX variations, this parameter is set internally to the value 7 and cannot be modified.

Packet-Not-Accepted to Link Request timeout

This parameter specifies whether or not the IP core enters a Fatal Error state if it sends a `packet-not-accepted` control symbol and then does not receive any `link-request` control symbol from the RapidIO link partner within the period of time indicated in the VALUE field of the PLTCTRL register at offset 0x120. By default, for backward compatibility, this parameter is turned on.

3.1.2. Data Settings

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

Baud Rate

Baud rate defines the baud rate based on the value that you specify. A device family may include devices at speed grades that do not support all the indicated baud rates. The speed grade tables in this user guide provide 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. The RapidIO parameter editor allows you to select any frequency supported by the transceiver.

Receive Buffer Size

Receive buffer size 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.

This parameter is not available for variations that target an Intel Arria 10 and Intel Cyclone 10 GX devices. RapidIO IP core Intel Arria 10 and Intel Cyclone 10 GX variations have a Physical layer receive buffer size of 32 KBytes.

Transmit Buffer Size

Transmit buffer size 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.

This parameter is not available for variations that target an Intel Arria 10 and Intel Cyclone 10 GX devices. RapidIO IP core Intel Arria 10 and Intel Cyclone 10 GX variations have a Physical layer transmit buffer size of 32 KBytes.

Note: 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.

Related Information

- Device Speed Grades on page 13
  For information about the speed grades supported for each device family, RapidIO mode, and baud rate combination.
- Clocking and Reset Structure on page 50
  For more information about the reference clock in high-speed transceiver blocks, and the supported frequencies.

3.1.3. 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.

Related Information

Physical Layer Receive Buffer on page 62
For more information about retry thresholds.

3.1.4. Transceiver Settings

Transceiver Settings options comprise the following configuration options:

Enable transceiver dynamic reconfiguration

Enable transceiver dynamic reconfiguration parameter allows you to specify whether or not the Intel Arria 10 or Intel Cyclone 10 GX Native PHY IP core dynamic reconfiguration interface is available in the visible signals of the RapidIO IP core. If you do not expect to use this interface, you can turn off this parameter to lower the number of IP core signals to route.

Note: This parameter is available only in IP core variations that target Intel Arria 10 and Intel Cyclone 10 GX devices.

All the parameters listed below are only available when you turn on the **Enable transceiver dynamic reconfiguration** parameter.

- **Enable transceiver capability registers**
- **Set user-defined IP identifier**
- **Enable transceiver control and status register**
- **Enable transceiver PRBS soft accumulators**

3.2. Transport and Maintenance Settings

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

3.2.1. Transport Layer

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

Enable 16-Bit Device ID Width

The **Enable 16-bit device ID width** setting specifies whether the IP core supports an 8-bit device ID width or a 16-bit device ID width. 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.

Enable 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.

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.

Note: 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.

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 settings is detailed in the above section.

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.

This parameter is not available for variations that target Intel Arria 10 and Intel Cyclone 10 GX devices. RapidIO IP core Intel Arria 10 and Intel Cyclone 10 GX variations do not check destination IDs by default. However, you can modify the PROMISCUOUS_MODE setting during normal operation.

Related Information

- Transaction ID Ranges on page 69
  For more information on transaction ID ranges.
- Error Management Registers on page 161

3.2.2. Input/Output Maintenance Logical Layer Module

The **I/O Maintenance Logical Layer Module** parameters specify 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 (this option is not valid in Intel Arria 10 and Intel Cyclone 10 GX variations)
- Avalon-MM Slave (this option is not valid in Intel Arria 10 and Intel Cyclone 10 GX variations)
- None

For variations that target Intel Arria 10 and Intel Cyclone 10 GX devices. RapidIO IP core, only two of the values are valid. Intel Arria 10 and Intel Cyclone 10 GX variations either include a Maintenance Logical layer module (Avalon-MM Master and Slave) or do not include a Maintenance Logical layer module (None).

Transmit Address Translation Windows

Number of transmit address translation windows is applicable only if you select **Avalon-MM Slave** or **Avalon-MM Master and 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.

This parameter is not available for variations that target Intel Arria 10 and Intel Cyclone 10 GX devices. RapidIO IP core Intel Arria 10 and Intel Cyclone 10 GX variations that include a Maintenance Logical layer module have 16 Maintenance transmit address translation windows.

3.2.3. Port Write

The **Port Write** options control whether the Maintenance Logical layer module can transmit or receive `port-write` requests. These options are available only if the Maintenance Logical layer has an Avalon-MM slave port.

These options are not available independently for variations that target Intel Arria 10 and Intel Cyclone 10 GX devices. RapidIO IP core Intel Arria 10 and Intel Cyclone 10 GX variations that include a Maintenance Logical layer module either support both reception and transmission of `port-write` requests, or do not support `port-write` requests at all (neither reception nor transmission).

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.

3.3. I/O and Doorbell Settings

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

3.3.1. I/O Logical Layer Interfaces

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

- Avalon-MM Master and Slave
- Avalon-MM Master (this option is not valid in Intel Arria 10 and Intel Cyclone 10 GX variations)
- Avalon-MM Slave (this option is not valid in Intel Arria 10 and Intel Cyclone 10 GX variations)
- None

3.3.2. I/O Slave Address Width

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

However, because the I/O Logical layer slave module addresses all hold word address values in 1x variations or double-word address values in 2x and 4x variations, the width of the external I/O Logical layer slave module address buses is the value you specify, minus 2 in 1x variations, or the value you specify, minus 3 in 2x and 4x variations.

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

This parameter is not available for variations that target Intel Arria 10 and Intel Cyclone 10 GX devices. RapidIO IP core Intel Arria 10 and Intel Cyclone 10 GX variations that include an I/O Logical layer Avalon-MM slave module preserve transaction ordering between read and write operations in the I/O Avalon-MM Logical layer slave module.

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.

Related Information

Input/Output Avalon-MM Slave Module on page 90
For more information about the I/O read and write order preservation feature.

3.3.4. 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**.

This parameter is not available for variations that target Intel Arria 10 and Intel Cyclone 10 GX devices. RapidIO IP core Intel Arria 10 and Intel Cyclone 10 GX variations that include I/O Logical layer master module have 16 Rx address translation windows.

3.3.5. 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**.

This parameter is not available for variations that target Intel Arria 10 and Intel Cyclone 10 GX devices. RapidIO IP core Intel Arria 10 and Intel Cyclone 10 GX variations that include I/O Logical layer slave module have 16 Tx address translation windows.

3.3.6. 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.

These parameters are linked for variations that target Intel Arria 10 and Intel Cyclone 10 GX devices. RapidIO IP core Intel Arria 10 and Intel Cyclone 10 GX variations either support outbound and inbound DOORBELL messages, or do not support DOORBELL messages. If you turn on one of these options, you must turn on both.

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**.

This parameter is not available for variations that target Intel Arria 10 and Intel Cyclone 10 GX devices. RapidIO IP core Intel Arria 10 and Intel Cyclone 10 GX variations that support DOORBELL messages preserve transaction order between DOORBELL messages and I/O write request transactions.

3.4. Capability Registers Settings

The **Capability Registers** tab 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.

Note: 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.

3.4.1. Device Registers

The **Device Registers** options identify the device, vendor, and revision level and set values in the `Device Identity` and `Device Information` 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.

Note: This DEVICE_ID field of the `Device Identity` register should not be confused with the DEVICE_ID field in the `Base Device ID` CSR.

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 is assigned and managed by the vendor specified in the VENDOR_ID field of the `Device Identity` register.

Related Information

- Capability Registers (CARs) on page 148
- Command and Status Registers (CSRs) on page 152

3.4.2. Assembly Registers

The **Assembly Registers** options identify the vendor who manufactured the assembly or subsystem of the device. These registers include the `Assembly Identity` and the `Assembly Information` CARs.

Assembly ID

Assembly ID corresponds to the ASSY_ID field of the `Assembly Identity` register, 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.

Assembly Vendor ID

Assembly vendor ID uniquely identifies the vendor who manufactured the assembly. This value corresponds to the ASSY_VENDOR_ID field of the `Assembly Identity` register.

Assembly Revision ID

Assembly revision ID indicates the revision level of the assembly and sets the ASSY_REV field of the `Assembly Information` CAR.

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.

Related Information

Capability Registers (CARs) on page 148

3.4.3. Processing Element Features

The `Processing Element Features` CAR 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 instead of the PROCESSOR bit.

Related Information

Capability Registers (CARs) on page 148

3.4.4. 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 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.

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.

Related Information

Capability Registers (CARs) on page 148

3.4.5. 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.

Note: 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.

Source Operation

Source operation, when turned on, sets the Data Message bit in the Source Operations CAR 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 and indicates that this endpoint can process received Data Message request packets.

Related Information

- Capability Registers (CARs) on page 148
- Signals on page 120
- Software Interface on page 138

4. Functional Description

4.1. Interfaces

The Intel RapidIO IP core supports the following interfaces:

- RapidIO Interface
- Avalon Memory Mapped (Avalon-MM) Master and Slave Interfaces
- Avalon Streaming (Avalon-ST) Interface

4.1.1. 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.

Related Information

RapidIO Interconnect Specification Webpage
More detailed information about the RapidIO interface specifications.

4.1.2. 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 Platform Designer (Standard)Platform Designer (Standard) system integration tool 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.

Related Information

Avalon Interface Specifications
For more information about the Avalon-MM interface.

4.1.2.1. Avalon-MM Interface Widths in the RapidIO IP Core

The RapidIO IP core has multiple Avalon-MM interfaces. The width of the data bus varies with the interface and with the RapidIO IP core mode.

- I/O Logical layer master and slave interfaces have a databus width of 32 bits in 1x variations and a databus width of 64 bits in 2x and 4x variations.
- Maintenance module has a databus width of 32 bits.
- Doorbell module has a databus width of 32 bits.

4.1.2.2. Avalon-MM Interface Byte Ordering

The RapidIO protocol uses big endian byte ordering, whereas Avalon-MM interfaces use little endian byte ordering.

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 15. Byte Ordering for the Avalon-MM and RapidIO interface

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]				32-Bit Word[0:31]			
	wdptr=0				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]				32-Bit Word[31:0]			
	Avalon-MM Address = N+4				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.

4.1.3. 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.

Related Information

Avalon-ST Pass-Through Interface on page 119
For more information about how Avalon-ST interface functions with the RapidIO IP core.

4.2. Clocking and Reset Structure

4.2.1. RapidIO IP Core Clocking

The RapidIO IP core has the following clock inputs:

- `sysclk:` Avalon system clock.
- `clk:` : reference clock for the transceiver Tx PLL and Rx PLL. In Intel Arria 10 and Intel Cyclone 10 GX variations, this clock port drives only the Rx PLL.
- `cal_blk_clk`: transceiver calibration-block clock (Arria II GX, Arria II GZ, Cyclone IV GX, and Stratix IV GX variations only).
- `reconfig_clk`: transceiver reconfiguration interface clock (Arria II GX, Arria II GZ, Cyclone IV GX, and Stratix IV GX variations only).
- `phy_mgmt_clk`: transceiver software interface clock (Arria V, Arria V GZ, Cyclone V, and Stratix V variations only).
- `tx_bonding_clocks_chN`: Intel Arria 10 and Intel Cyclone 10 GX devices transceiver channel clocks for the transceiver channel that corresponds to RapidIO lane N (Intel Arria 10 and Intel Cyclone 10 GX variations only)

In addition, if you turn on **Enable transceiver dynamic reconfiguration** for your RapidIO Intel Arria 10 and Intel Cyclone 10 GX variations, the IP core includes a reconfig_clk_chN input clock for each RapidIO lane `N. Each reconfig_clk_chN clocks the` Intel Arria 10 and Intel Cyclone 10 GX Native PHY dynamic reconfiguration interface for RapidIO lane N.

The RapidIO IP core supports ±100 ppm reference clock difference and can handle a maximum difference of ±200 ppm between the `rxclk` and `txclk` clocks.

The RapidIO IP core provides the following clock outputs from the transceiver:

- Transceiver receiver clock (recovered clock) (`rxgxbclk`)
- Recovered data clock (`rxclk`). Recovered clock that drives the receiver modules in the Physical layer.
- Transceiver transmit-side clock (`txclk`). Main clock for the transmitter modules in the Physical layer.

RapidIO IP core 2x and 4x variations are implemented in the transceiver TX bonded mode. All channels of a 2x or 4x variation, on any supported device, must reside in a single transceiver block.

To support this requirement in Arria II GX, Arria II GZ, Cyclone IV GX, and Stratix IV GX variations, the starting channel number for a 4x variation must be a multiple of four.

When you generate a custom IP core for variations other than Intel Arria 10 and Intel Cyclone 10 GX, the **<*variation name*>_constraints.tcl** script contains the required assignments. When you run the script, the constraints are applied to your project.

4.2.1.1. Avalon System Clock

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 called `sysclk`. Platform Designer (Standard) allows you to export the `clock` signal with a name of your choice.

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

4.2.1.2. Reference Clock

The reference clock signal drives the transceiver and the Physical layer. By default, this clock is called `clk` in the generated IP core. Platform Designer (Standard) allows you to export the `clk` signal with a name of your choice.

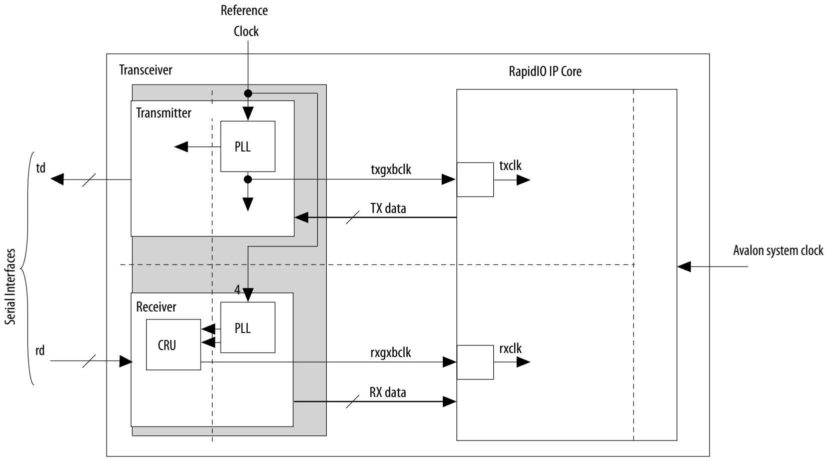
The reference clock, `clk`, is the incoming reference clock for the transceiver's PLL. The frequency of the input clock must match the value you specify for the **Reference clock frequency** parameter. The transceiver PLL converts the reference clock frequency to the internal clock speed that supports the RapidIO IP core baud rate.

The RapidIO parameter editor lets you select one of the supported frequencies. The selection allows you to use an existing clock in your system as the reference clock for the RapidIO IP core.

Note: You must drive the external transceiver TX PLL `pll_refclk0` input clock from the same source from which you drive the RapidIO IP core `clk` input clock.

This source must be within ±100 ppm of its nominal value, to ensure the difference between any two devices in the RapidIO system is within ±200 ppm.

Figure 9. Reference Clock in a variation other than Intel Arria 10 and Intel Cyclone 10 GX RapidIO IP Core



Note:
This figure does not show the Custom PHY IP core clock phy_mgmt_clk.

The above figure shows how the transceiver uses the reference clock in variations other than Intel Arria 10 and Intel Cyclone 10 GX. In Intel Arria 10 and Intel Cyclone 10 GX variations, `clk` is the reference clock only for the RX PLL. The reference clock for the TX PLL is an input signal to the TX PLL IP core that you connect to the RapidIO IP core.

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.

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

Related Information

- Intel Arria 10 Transceiver PHY User Guide
- Intel Cyclone 10 GX Transceiver PHY User Guide

4.2.1.3. Other Input Clocks

In variations that target a device for which the transceivers are configured with the ALTGX megafunction, and not with a Transceiver PHY IP core, the transceiver's calibration-block clock is called `cal_blk_clk`.

In Arria V, Cyclone V, and Stratix V devices, the transceiver has an additional clock, `phy_mgmt_clk`, which clocks the software interface to the transceiver. In Intel Arria 10 and Intel Cyclone 10 GX devices, the transceiver has an input clock bus `tx_bonding_clocks_chN`. These clocks should be driven by the external TX transceiver PLL. Intel Arria 10 and Intel Cyclone 10 GX variations also have an option interface, the Intel Arria 10 and Intel Cyclone 10 GX Native PHY dynamic reconfiguration interface, which includes a clock signal for each transceiver channel.

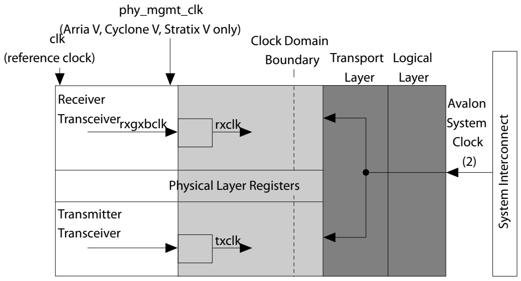
4.2.1.4. Clock Domains

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 Platform Designer (Standard), 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.

Note: All of the clock inputs for the Logical layer modules must be connected to the same clock source as the Avalon system clock.

Figure 10. Clock Domains in RapidIO IP Core



Notes:
(1) Clock descriptions:
phy_mgmt_clk- PHY IP core management clock (Arria V, Cyclone V,Stratix V devices only)
reconfig_clk_chN- Intel Arria 10 and Intel Cyclone 10 GX transceiver dynamic reconfiguration interface clock (Intel Arria 10 and Intel Cyclone 10 GX devices only)
rxclk- Receiver internal global clock (recovered clock)
rxgxbclk- Receiver transceiver clock
txclk- Transmitter internal global clock
tx_bonding_clocks_chN- Transmitter transceiver clock
(2) The Avalon system clock is called sysclk.

4.2.1.5. Baud Rates and Clock Frequencies

The RapidIO specification specifies baud rates of 1.25, 2.5, 3.125, and 5.0 Gbaud.

Table 16. Clock Frequencies for 1x and 2x RapidIO IP Core Variations

Baud Rate (Gbaud)	rxgxbclk (MHz)		txclk, rxclk (MHz)	Avalon system clock (sysclk)		
	1x	2x		Minimum (MHz)	Typical (MHz)	Maximum (MHz)[19]
1.25	62.5	31.25	31.25	15.625	31.25	46.875
2.5	125	62.5	62.5	31.25	62.5	93.75
3.125	156.25	78.125	78.125	39.065	78.125	117.19
5.0	250[20]	125	125	62.50	125	187.50

[19] The maximum system clock frequency might be limited by the achievable fMAX and can vary based on the family and speed grade.

[20] For Arria V and Cyclone V device variations, the rxgxbclk frequency in RapidIO IP core 1x variations at 5 GBaud, is 125 MHz.

Table 17. Clock Frequencies for 4x RapidIO IP Core Variations

Baud Rate (Gbaud)	rxgxbclk (MHz)	Avalon System Clock (sysclk)		
		Minimum (MHz)	Typical (MHz)	Maximum (MHz)[19]
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	250	250

Note: The default reference clock frequency is 125 MHz across all device variations. You can select different reference clock frequencies from the set of allowed reference clock frequencies.

The reference clock is called `clk`.

Related Information

- Device Speed Grades on page 13
  For more information about device family speed grades for different RapidIO variations.
- Reference Clock on page 51
  For more information about the allowed reference clock frequencies.

4.2.2. Reset for RapidIO IP Cores

The RapidIO IP core has the following reset input signals:

- `reset_n`: main active-low reset signal
- `phy_mgmt_clk_reset`: transceiver software management interface signal to reset the Custom PHY IP core included in the RapidIO Arria V, Cyclone V, or Stratix V variation (Arria V, Cyclone V, and Stratix V variations only)
- `tx_analogreset`, `rx_analogreset`, `tx_digitalreset`, `rx_digitalreset`: transceiver reset signals (Intel Arria 10 and Intel Cyclone 10 GX variations only)

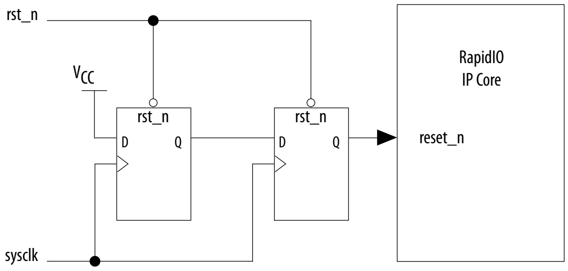
In addition, if you turn on **Enable transceiver dynamic reconfiguration** for your RapidIO Intel Arria 10 and Intel Cyclone 10 GX variations, the IP core includes `reconfig_reset_chN` input clock to reset the Intel Arria 10 or Intel Cyclone 10 GX Native PHY dynamic reconfiguration interface for each RapidIO lane N.

4.2.2.1. General RapidIO Reset Signal Requirements

All reset signals can be asserted asynchronously to any clock. However, most reset signals must be deasserted synchronously to a specific clock.

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 11. Circuit to Ensure Synchronous Deassertion of reset_n



In systems generated by Platform Designer (Standard), this circuit is generated automatically. However, if your RapidIO IP core variation is not generated by Platform Designer (Standard), you must implement logic to ensure the minimal hold time and synchronous deassertion of the `reset_n` input signal to the RapidIO IP core.

4.2.2.2. Reset Controller

All RapidIO IP core variations other than Intel Arria 10 and Intel Cyclone 10 GX include a dedicated reset control module to handle the specific requirements of the internal transceiver module. Intel Arria 10 and Intel Cyclone 10 GX RapidIO IP core variations do not include a reset controller.

The reset control module is named `riophy_reset`. This `riophy_reset` module is defined 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.

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.

4.2.2.3. Reset Requirements for Arria V, Cyclone V, and Stratix V Variations

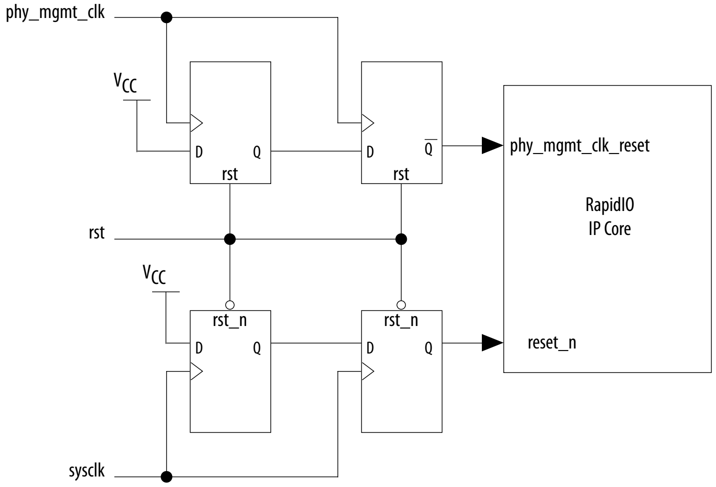
Arria V, Cyclone V, and Stratix V variations have the following additional constraints:

- 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. The following figure 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 Transceiver Reconfiguration Controller `reconfig_busy` signal is asserted. The RapidIO IP core must remain in reset until the Transceiver Reconfiguration Controller is available.

The assertion of `reset_n` causes the whole IP core 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.

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 12. Circuit to Also Ensure Synchronous Assertion of phy_mgmt_clk_reset with reset_n



In systems generated by Platform Designer (Standard), this circuit is generated automatically. However, if your Arria V, Cyclone V, or Stratix V RapidIO IP core variation is not generated by Platform Designer (Standard), 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.

Related Information

- Software Interface on page 138
  For information about the default value of registers after reset.
- Signals on page 120
  For more information about the requirements for reset signals.

4.2.2.4. Reset Requirements for Intel Arria 10 and Intel Cyclone 10 GX Variations

To implement the reset sequence correctly for your RapidIO IP core configured on Intel Arria 10 and Intel Cyclone 10 GX devices, you must connect the `tx_analogreset`, `tx_digitalreset`, `rx_analogreset`, and `rx_digitalreset` signals to Transceiver PHY Reset Controller IP core. User logic must drive the following signals from a single reset source:

- RapidIO IP core `reset_n` (active low) input signal.
- Transceiver PHY Reset Controller `reset` (active high) input signal.
- `TX PLL mcgb_rst` (active high) input signal. However, Intel Arria 10 and Intel Cyclone 10 GX device requirements take precedence. Depending on the TX PLL configuration, your design might need to drive `TX PLL mcgb_rst` with different constraints.

User logic must connect the remaining input reset signals of the RapidIO IP core to the corresponding output signals of the Transceiver PHY Reset Controller IP core.

Note: In the Transceiver PHY Reset Controller parameter editor, set the value of RX_PER_CHANNEL to 0 for Intel Arria 10 and Intel Cyclone 10 GX variations.

Related Information

- Intel Arria 10 Transceiver PHY User Guide
  For information about the Intel Arria 10 Transceiver PHY Reset Controller IP core.
- Intel Cyclone 10 GX Transceiver PHY User Guide
  For information about the Intel Cyclone 10 GX Transceiver PHY Reset Controller IP core.

4.2.2.5. RapidIO IP Core Reset Behavior

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

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.

4.3. Physical Layer

4.3.1. Features

The Physical layer has the following features:

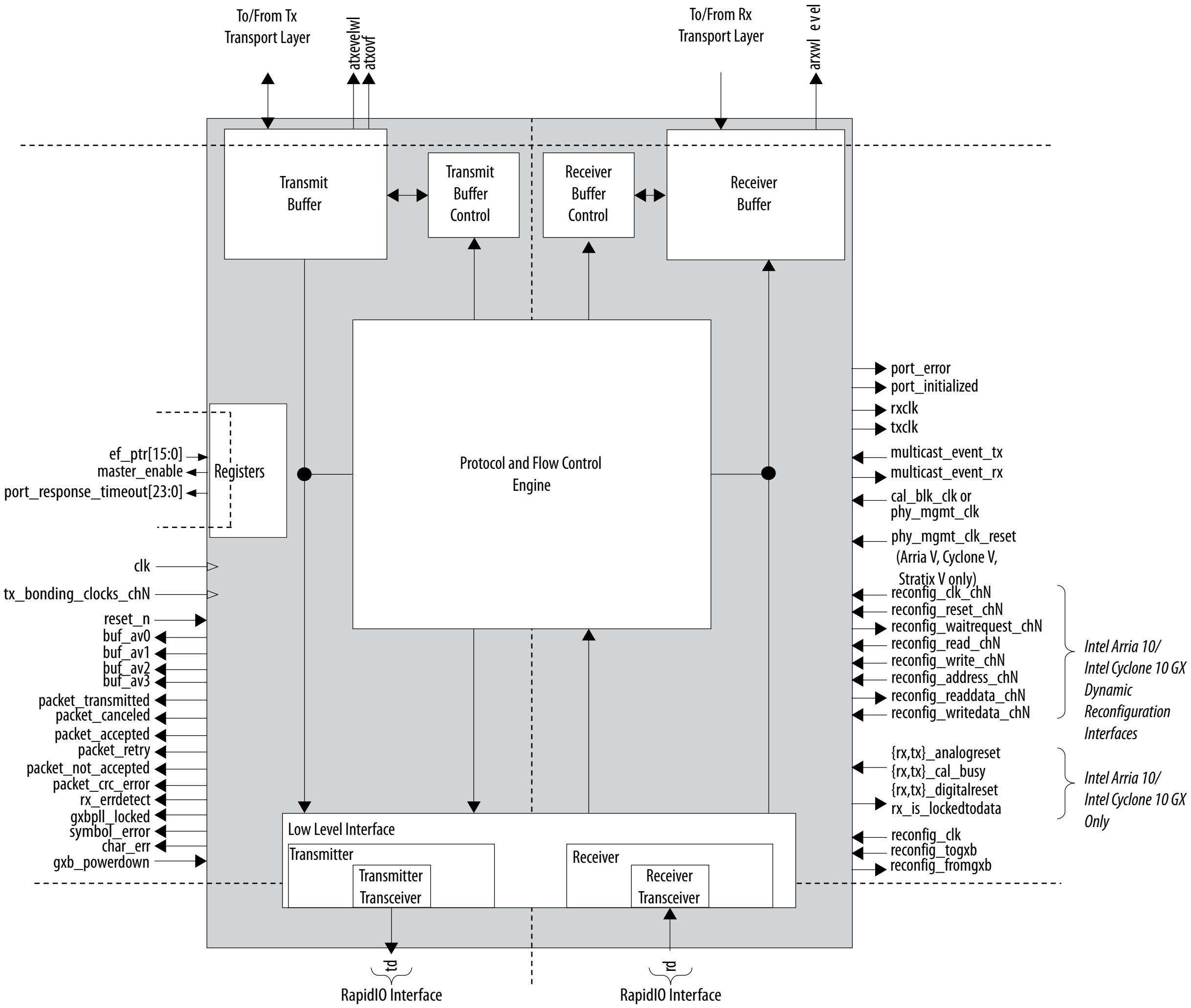
- Port initialization
- Transmitter and receiver with the following features:
  - One, two, or four lane high-speed data serialization and deserialization (up to 5.0 Gbaud for 1x variations with 32-bit Atlantic interface; up to 5.0 Gbaud for 2x and 4x variations with 64-bit Atlantic interface)
  - Clock 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 acknowledgments

- Order of retransmission maintenance and acknowledgments
- 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

4.3.2. Physical Layer Architecture

Figure 13. Physical Layer High Level Block Diagram

The following figure shows the architecture of the Physical layer and illustrates the interfaces that it supports. Dotted lines indicate clock domain boundaries within the layer.



4.3.3. 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

4.3.3.1. Receiver Transceiver

The receiver transceiver is an embedded megafunction in the Arria II GX, Arria II GZ, Cyclone IV GX, or Stratix IV GX device, or an embedded Custom PHY IP core in the Arria V, Cyclone V, or Stratix V device, or an embedded Intel Arria 10 or Intel Cyclone 10 GX Native PHY IP core in the Intel Arria 10 or Intel Cyclone 10 GX devices. The receiver transceiver implements the following process:

1. Feeds serial data from differential input pins to the CRU to detect clock and data.
2. Deserializes recovered data into 10-bit code groups.
3. Sends the code groups to the pattern detector and word-aligner block to detect word boundaries.
4. Performs 8B10B decoding on properly aligned 10-bit code groups to convert them to 8-bit characters.
5. Converts 8-bit characters to 16-bit or 32-bit data in the 8-to-16 or 8-to-32 demultiplexer.

4.3.3.2. 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.[21]

In variations of the RapidIO IP core that include the Transport layer, the Transport layer removes the CRC after the 80th 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 80th 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.

4.3.4. Low-Level Interface Transmitter

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

[21] The RapidIO IP core removes the intermediate CRC from the RapidIO packet before it presents it at the Rx passthrough interface. The CRC and the padding at the end of the packet are not removed.

- 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.

4.3.4.1. Transmitter Transceiver

The transmitter transceiver is an embedded megafunction in the Arria II GX, Arria II GZ, Cyclone IV GX, or Stratix IV GX device, or an embedded Custom PHY IP core in the Arria V, Cyclone V, or Stratix V device, or an embedded Intel Arria 10 or Intel Cyclone 10 GX Native PHY IP core in the Intel Arria 10 or Intel Cyclone 10 GX devices.

The transmitter transceiver implements the following process:

1. Multiplexes the 16-bit or 32-bit parallel input data to the transmitter to 8-bit data.
2. Performs 8B10B encoding on the 8-bit data to convert it to 10-bit code groups.
3. Serializes the 10-bit encoded data and sends it to differential output pins.

4.3.5. 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.

4.3.6. Physical Layer Receive Buffer

The Physical layer passes data to the Transport layer through a Physical layer receive buffer.. The data passes between the buffer and the Transport layer on a bus that is 32 bits wide in 1x variations and 64 bits wide in 2x and 4x variations.

The Physical layer receiver block accepts packet data from the low-level interface receiver module and stores the data in its receive buffer. The receive buffer provides clock decoupling between the Physical layer rxclk clock domain and the Transport layer sysclk clock domain.

You can specify a value of 4, 8, 16, or 32 KBytes to configure the receive buffer size in devices other than Intel Arria 10 and Intel Cyclone 10 GX. RapidIO Intel Arria 10 and Intel Cyclone 10 GX variations have a receive buffer size of 32 KBytes. 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. The IP core provides the current number of 64-byte blocks in the free queue in the arxwlevel output signal.

As many as five 64-byte blocks may be required to store a packet.

4.3.6.1. Error Conditions that Flush the Receive Buffer

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`.

4.3.6.2. Error Conditions Flagged for the Transport Layer

The Physical layer passes data from the receive buffer to the Transport layer in 64-Kbyte blocks. The Physical layer might identify an error condition after it begins passing a packet from the receive buffer to the Transport layer. In that case, the Physical layer flags an Errored packet indication to the Transport layer. The Physical layer flags an Errored packet in the following cases:

- 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 Physical layer flags the error. If the packet size is less than 64 bytes, the Physical layer identifies and drops the errored packet before it begins sending the packet to the Transport layer.
- Stomp—the Physical layer flags an error if it receives a `stomp` control symbol in the midst of a packet, causing the packet to be prematurely terminated.
- Packet size—if a received packet exceeds the allowable size, the Physical layer cuts it short to the maximum allowable size (276 bytes total), and flags the error.
- Outgoing symbol buffer full—under some congestion conditions, the outgoing symbol buffer has no space available for the `packet_accepted` control symbol. In this case, the RapidIO IP core cannot acknowledge the packet, and the link partner must retry transmission. The Physical layer flags an error to indicate to the Transport layer that it should ignore the received packet because it will be retried.
- Control symbol error —if an embedded or packet-delimiting control symbol is errored, the Physical layer flags the error. The packet in which the errored control symbol is embedded should be retransmitted by the link partner as part of the error recovery process.
- Character error—if the Physical layer receives 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) within a packet. In this case the Physical layer flags the error and drops the rest of the packet.

4.3.6.3. Receive Priority Threshold Values

The Physical layer 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

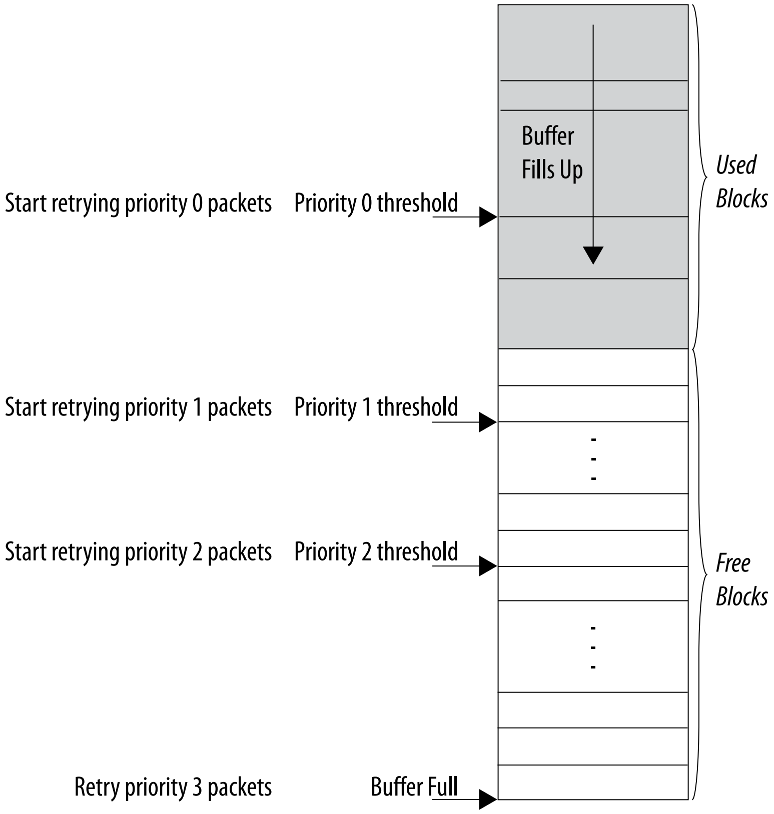
You can specify other threshold values by turning off **Auto-configured from receiver buffer size** on the **Physical Layer** page in the RapidIO parameter editor.

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

The following figure shows sample threshold values in context to illustrate how they work together to enforce the deadlock prevention rules.

Figure 14. Receiver Threshold Levels



4.3.7. Physical Layer Transmit Buffer

The Physical layer accepts packet data from the Transport layer and stores it in the transmit buffer for the RapidIO link low-level interface transmitter. The data passes from the Transport layer to the Physical layer on a bus that is 32 bits wide in 1x variations and 64 bits wide in 2x and 4x variations.

The transmit buffer implements the following features:

- Provides clock decoupling between the Transport layer `sysclk` clock domain and the Physical layer `txclk` clock domain.
- Implements the RapidIO specification requirements for packet priority handling and deadlock avoidance, by configuring individual priority transmit and retransmit queues.

The transmit buffer is the main memory in which the packets are stored before they are transmitted. You can specify a value of 4, 8, 16, or 32 KBytes to configure the total memory space available for the transmit buffer in devices other than Intel Arria 10 and Intel Cyclone 10 GX. RapidIO Intel Arria 10 and Intel Cyclone 10 GX devices have a total transmit buffer size of 32 KBytes.

The transmit buffer space is partitioned into 64-byte blocks that are allocated from a free queue and returned to the free queue when no longer needed. The 64-byte blocks are used on a first-come, first-served basis by the individual transmit and retransmit queues.

The IP core provides the current number of 64-byte blocks in the free queue in the atxwlevel output signal. The transmit buffer also has an output signal, atxovf, which indicates a transmit buffer overflow condition.

4.3.7.1. Transmit and Retransmit Queues

To meet the RapidIO specification requirements for packet priority handling and deadlock avoidance, the Physical layer transmit buffer implements four transmit queues and four retransmit queues, one for each priority level.

As the Transport layer writes packets to the Physical layer, the Physical layer adds them to the end of the appropriate priority transmit queue. The transmitter always transmits the packet at the head of the highest priority non-empty queue. After the packet is transmitted, the Physical layer moves the packet from the transmit queue 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. The Transport layer might have written higher priority packets to the Physical layer since the retried packet was originally transmitted. In that case, the higher priority packets 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 ascending order.

4.3.7.2. Error Conditions that Flush the Transmit Buffer

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.

4.3.7.3. Forced Compensation Sequence Insertion

As packet data is written to the transmit buffer, 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 Avalon system 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 a `stomp` control symbol in the midst of a packet, it provides an error indication to the Transport layer. Because the packet was prematurely terminated at transmission, no traffic is lost and no protocol violation occurs.

4.4. Transport Layer

The Transport layer is a required module of the RapidIO IP core. The Transport layer 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.

You can optionally turn on the following two Transport layer parameters:

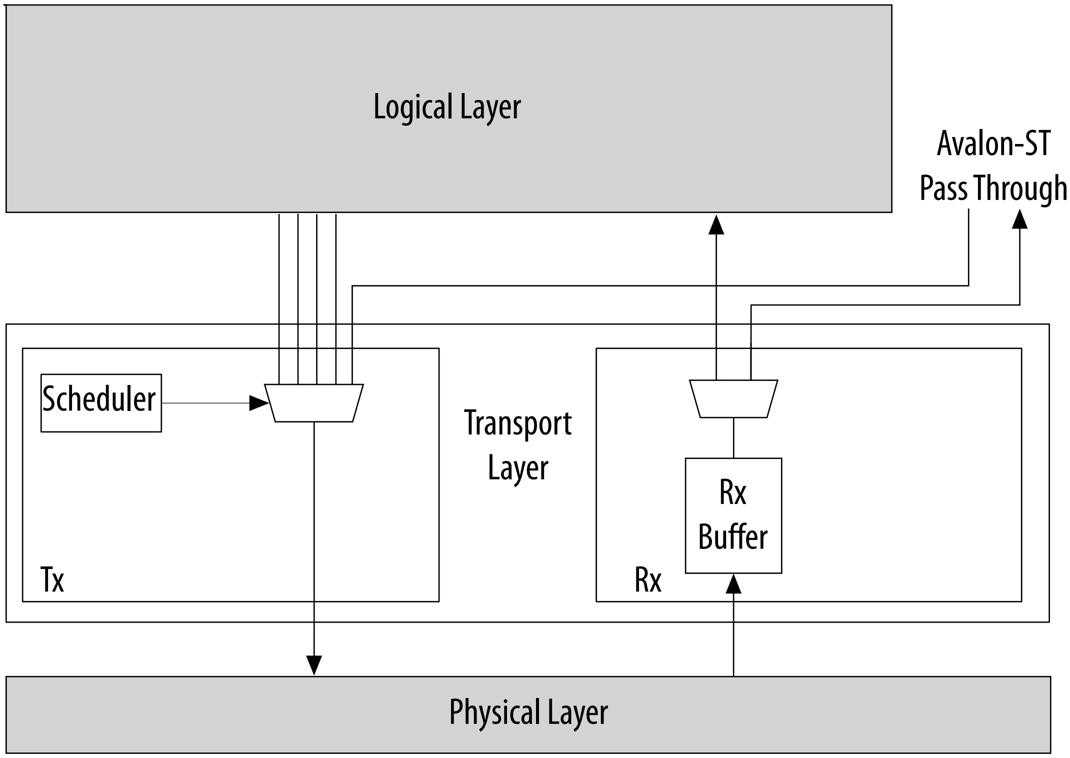
- **Enable Avalon-ST pass-through interface**—If you turn on this parameter, the Transport layer routes all unrecognized packets to the Avalon-ST pass-through interface.
- **Disable Destination ID checking by default**—If you turn on this parameter, request packets are considered recognized even if the 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. This parameter is turned on in RapidIO Intel Arria 10 and Intel Cyclone 10 GX variations.

  You can also turn on and turn off destination ID checking in the PROMISCUOUS_MODE field of the Rx `Transport Control` register at offset 0x10600.

Note: The Transport layer is enabled automatically by default, and cannot be disabled. Beginning with the RapidIO IP core v14.0 release, the RapidIO IP core no longer supports Physical-layer only instances.

The Transport layer module is divided into receiver and transmitter submodules.

Figure 15. Transport Layer Block Diagram



Related Information

- Command and Status Registers (CSRs) on page 152
  For more information on Base Device ID CSR—Offset: 0x60 register.

- Transport Layer Feature Register on page 160
  For more information on Rx Transport Control register.

4.4.1. 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. 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 (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.

Related Information

Transaction ID Ranges on page 69
For more information on transaction IDs ranges.

4.4.2. 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 18. Transaction ID Ranges and Assignments

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.
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.

4.4.3. 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.

Related Information

RapidIO Interconnect Specification Webpage

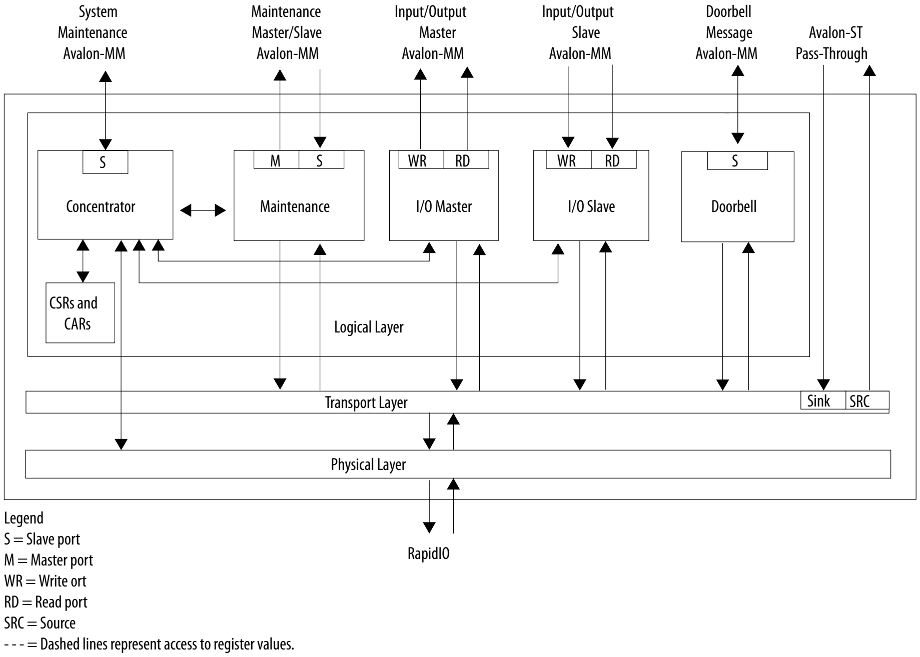
For information about the Transport layer refer to Part 3: Common Transport Specification.

4.5. 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. The following figure 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 16. RapidIO IP Core Functional Block Diagram

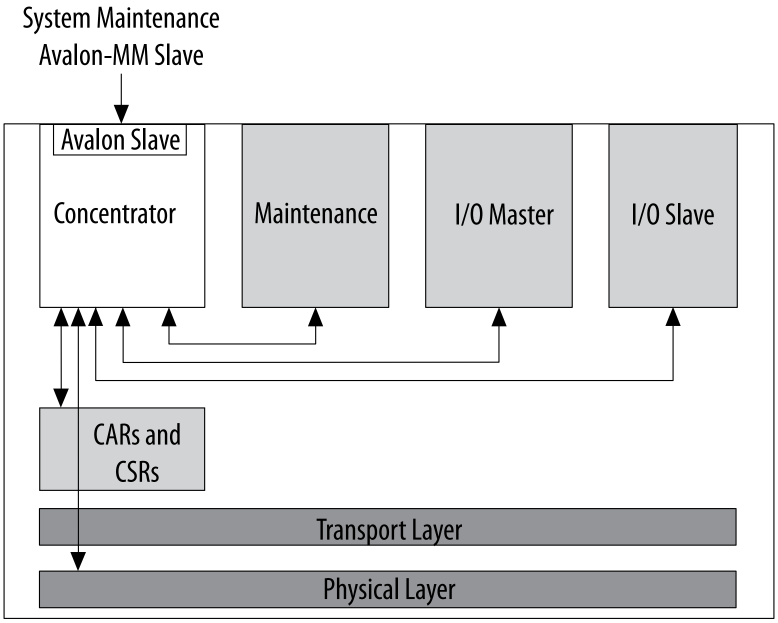


4.5.1. Concentrator Register Module

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. The following figure 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.

Note: 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 17. 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:

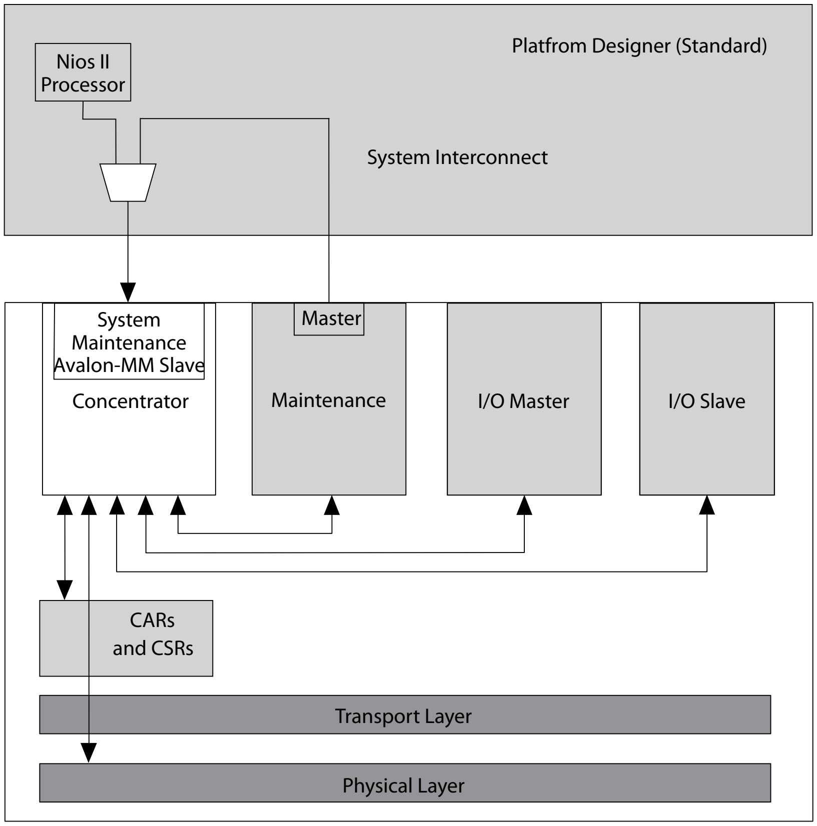
- Received port-write
- I/O read out of bounds
- I/O write out of bounds
- Invalid write
- Invalid write burstcount

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

- Platform Designer (Standard) interconnect
- Custom logic

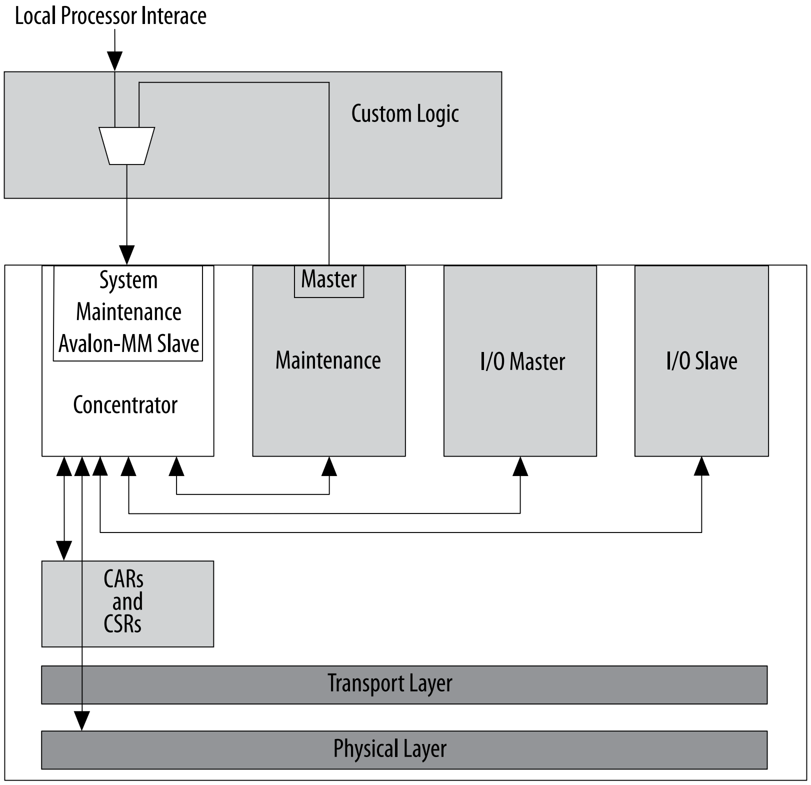
A local host can access the RapidIO registers from a Platform Designer (Standard) system. In this figure, a Nios II processor is part of the Platform Designer (Standard) system and is configured as an Avalon-MM master that accesses the RapidIO IP core registers through the System Maintenance Avalon-MM slave. Alternatively, you can implement custom logic to access the RapidIO registers.

Figure 18. Local Host Accesses RapidIO Registers from a Platform Designer (Standard) 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 Platform Designer (Standard) system shown by the routing to the Concentrator's system Maintenance Avalon-MM slave in the previous figure. If you do not use a Platform Designer (Standard) system, you can create custom logic as shown below.

Figure 19. Custom Logic Accesses RapidIO IP core Registers



Related Information

- Maintenance Interrupt Control Registers on page 153
  For details on interrupts.
- Creating a System With Platform Designer (Standard)
  For information on implementation details.

4.5.2. 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 two or four choices for this module.

Table 19. 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. This option is not available for Intel Arria 10 and Intel Cyclone 10 GX variations.
Avalon-MM Slave	Restricts your IP core to initiating MAINTENANCE transactions. This option is not available for Intel Arria 10 and Intel Cyclone 10 GX variations.
None	Prevents your IP core from initiating or terminating MAINTENANCE transactions.

Note: If you add this module to your variation other than Intel Arria 10 and Intel Cyclone 10 GX 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. Intel Arria 10 and Intel Cyclone 10 GX variations have 16 Maintenance transmit address translation windows.

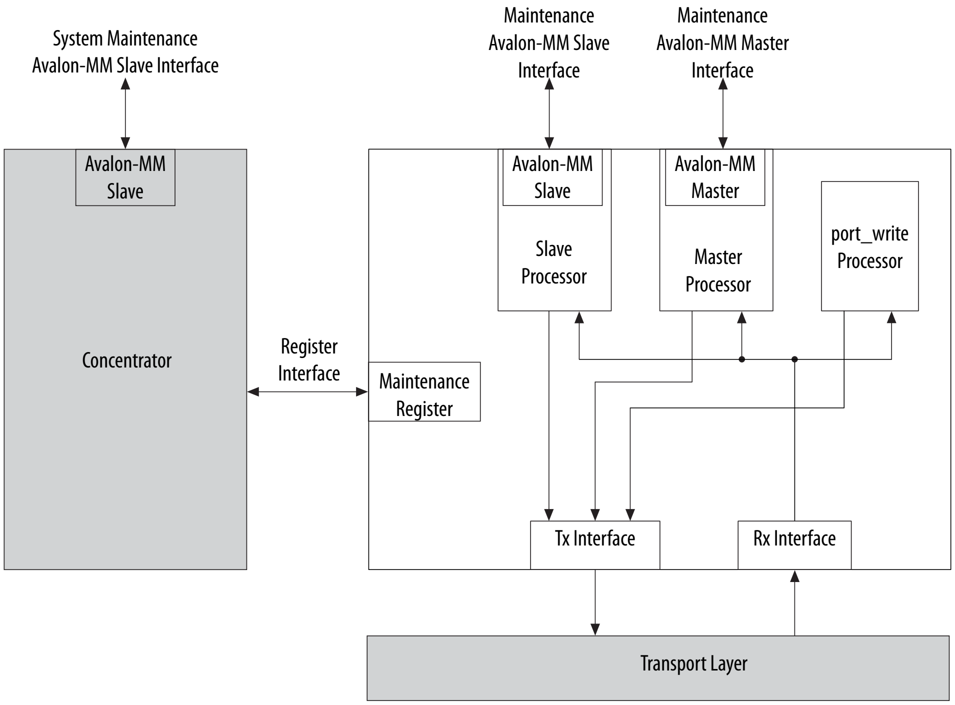
The Maintenance module can be segmented into the following four major submodules:

- 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 20. Maintenance Module Block Diagram



Related Information

Input/Output Maintenance Logical Layer Module on page 41

4.5.2.1. Maintenance Register

The Maintenance Register module implements all of the control and status registers required by this module to perform its functions. These registers are accessible through the System Maintenance Avalon-MM interface.

Related Information

- Maintenance Interrupt Control Registers on page 153
- Receive Maintenance Registers on page 154
- Transmit Maintenance Registers on page 154

4.5.2.2. 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

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

Figure 21. Signal Relationships for Four Write Transfers on the Avalon-MM Slave Interface

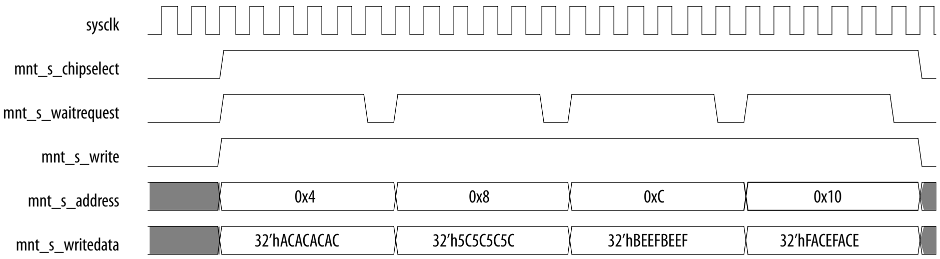
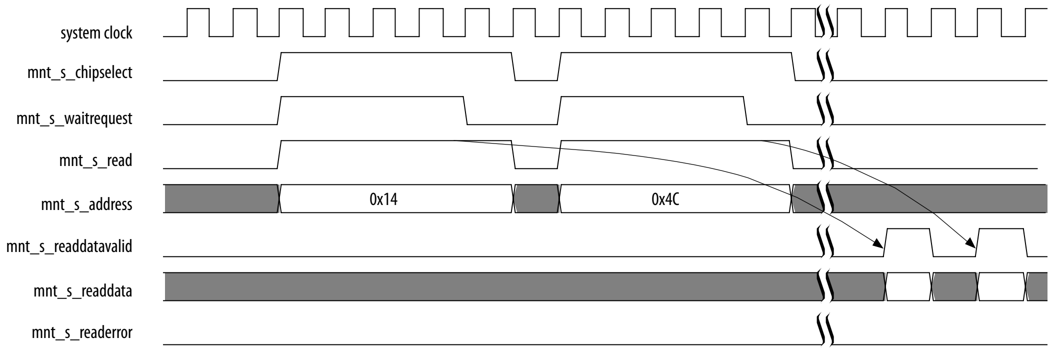


Figure 22. Signal Relationships for Two Read Transfers on the Avalon-MM Slave Interface



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.
- wdptr

Each window is enabled if the window enable (WEN) bit of the Tx Maintenance Window n Mask register is set. Each window is defined by the following registers:

- A base register: Tx Maintenance Mapping Window n Base
- A mask register: Tx Maintenance Mapping Window n Mask
- An offset register: Tx Maintenance Mapping Window n Offset
- A control register: Tx Maintenance Mapping Window n Control

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[23:1] & mask[25:3]) == base[25:3]
then config_offset = (offset[23:3] & mask[23:3])|
(mnt_s_address[21:1] & ~mask[23:3])
```

where:

- mnt_s_address[23:0] 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[0].

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.

Related Information

- Avalon Interface Specifications
  For more details on the supported transfers.
- Receive Maintenance Registers on page 154
- Transmit Maintenance Registers on page 154
- Transmit Port-Write Registers on page 155

4.5.2.3. 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

Figure 23. Signal Relationships for Four Write Transfers on the Maintenance Avalon-MM Master Interface

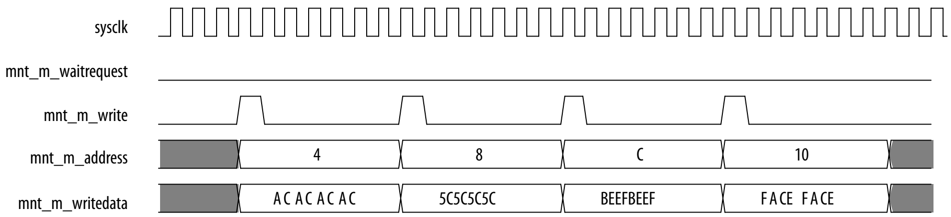
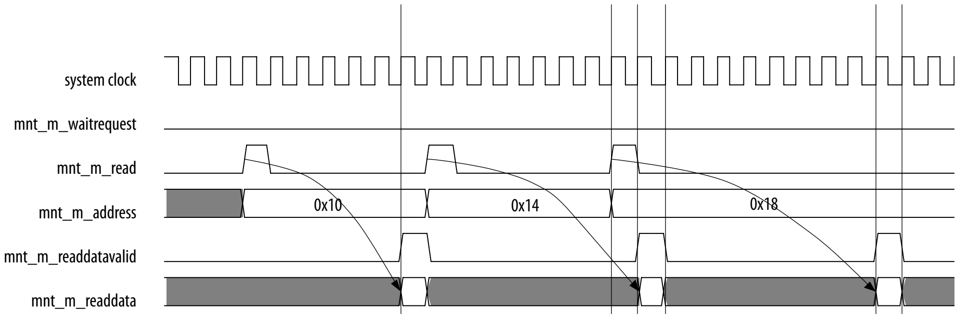


Figure 24. Timing of a Read Request on the Maintenance Avalon-MM Master Interface



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.

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

Related Information

- Avalon Interface Specifications
  For more details on the supported transfers.
- Receive Maintenance Registers on page 154

4.5.2.4. 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 receive and transmit port-write registers.

Related Information

- Transmit Port-Write Registers on page 155
- Receive Port-Write Registers on page 156

4.5.2.4.1. Port-Write Transmission

To send a port-write to a remote device, you must program the transmit port-write control and data registers. 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.

Related Information

- Transmit Port-Write Registers on page 155
- Receive Port-Write Registers on page 156

4.5.2.4.2. 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 through Rx Port Write Buffer :

- wrsize
- wdptr
- payload

The wrsize and the wdptr determine the value of the PAYLOAD_SIZE field in the Rx Port Write Status register. 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 is set.

Related Information

- Receive Port-Write Registers on page 156
- Maintenance Interrupt Control Registers on page 153

4.5.2.5. Maintenance Module Error Handling

The Maintenance Interrupt register (at 0x10080) and the Maintenance Interrupt Enable register (at 0x10084), 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) is generated if a response packet is not received within the time specified by the Port Response Time-Out Control register.
- 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.

Related Information

- Maintenance Interrupt Control Registers on page 153
- Error Management Registers on page 161
  For more information about the error management registers.
- Physical Layer Registers on page 142
  For information about how the time-out value is calculated.

4.5.3. 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

4.5.3.1. Input/Output Avalon-MM Master Module

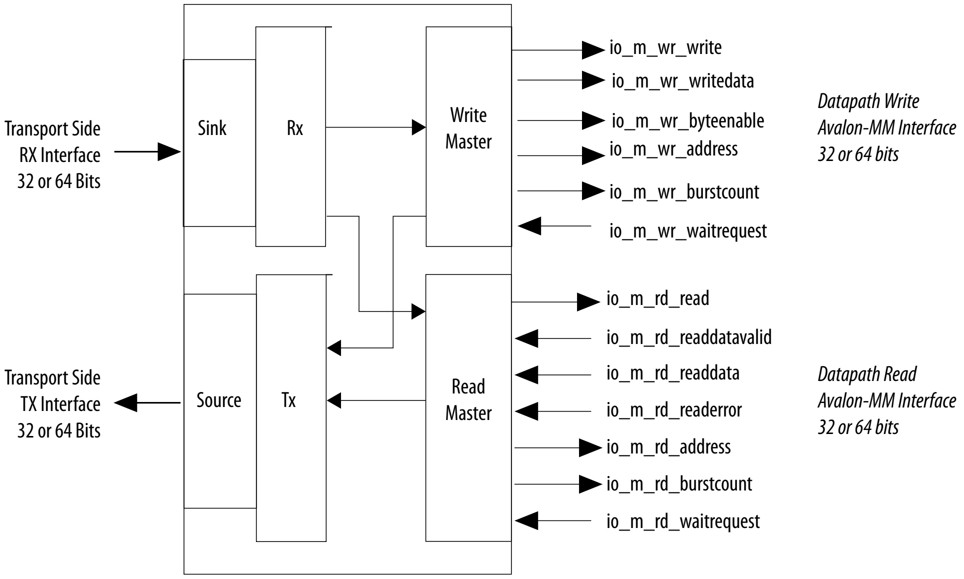
The Input/Output (I/O) Avalon-MM master Logical layer module receives RapidIO read and write request packets from a remote endpoint through the Transport layer module. The I/O Avalon-MM master module translates the request packets into Avalon-MM transactions, and creates and returns RapidIO response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

Note: 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 25. I/O Avalon-MM Master Logical Module Block Diagram



4.5.3.1.1. 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 20. Address Translation Registers

Registers	Location
Input/Output master base address	0x10300, 0x10310, 0x10320, 0x10330, 0x10340,0x10350, 0x10360, 0x10370, 0x10380, 0x10390, 0x103A0, 0x103B0, 0x103C0, 0x103D0, 0x103E0, 0x103F0
Input/Output master address mask	0x10304, 0x10314, 0x10324, 0x10334, 0x10344,0x10354, 0x10364, 0x10374, 0x10384, 0x10394, 0x103A4, 0x103B4, 0x103C4, 0x103D4, 0x103E4, 0x103F4
Input/Output master address offset	0x10308, 0x10318, 0x10328, 0x10338, 0x103480x10358, 0x10368, 0x10378, 0x10388, 0x10398, 0x103A8, 0x103B8, 0x103C8, 0x103D8, 0x103E8, 0x103F8

Your variation must have at least one translation window. Intel Arria 10 and Intel Cyclone 10 GX variations have 16 address translation windows. 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])

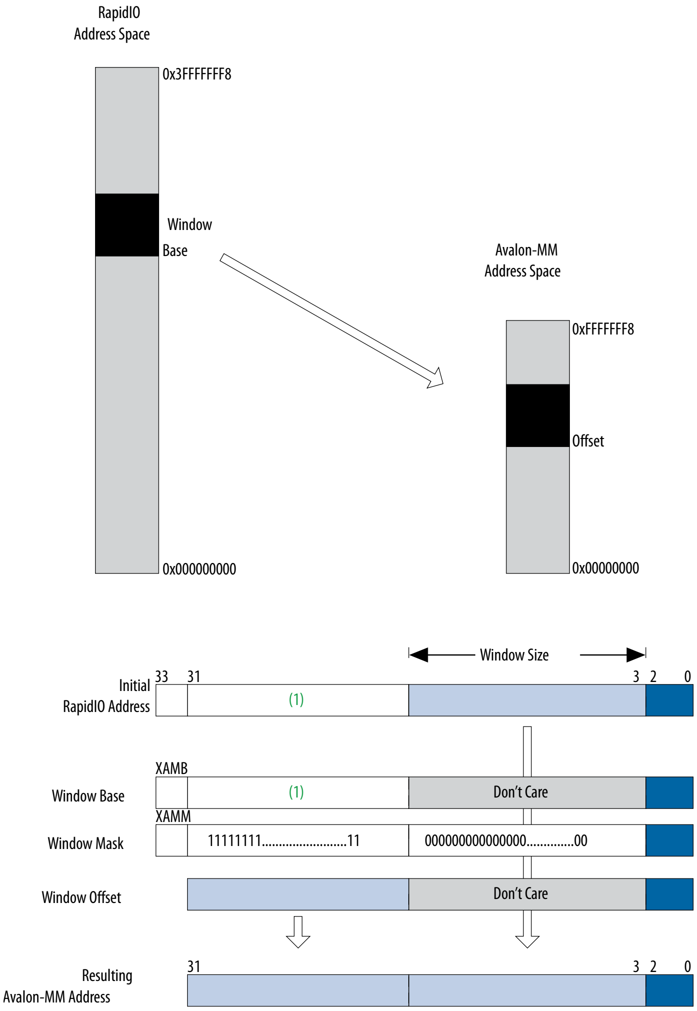
The value of address[2] is zero for variations with 64-bit wide datapath Avalon-MM interfaces.

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 26. I/O Master Window Translation



Note:
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.

Table 21. 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 22. RapidIO Master Write Transaction Burstcount and Byteenable (32-Bit Datapath)

RapidIO Values		Avalon-MM Values		
wrsize (4'bxxxx)	wdptr (1'bx)	Maximum Burstcount	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(22)	1	1110	—
	1(22)	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(22)	2	1110	1111
	1(22)	2	1111	0111
1011	0	2	1111	1111
	1	4	1111	—
1100	0	8	1111	—
	1	16	1111	—
1101	0(23)	—	—	—

continued...

[22] This combination of wdptr and wrsize values should be avoided, because the resulting byteenable value is not allowed by the Avalon-MM specification.

[23] This combination of wdptr and wrsize value is reserved. If this combination is received, the RapidIO IP core declares an error.

RapidIO Values		Avalon-MM Values		
wrsize (4'bxxxx)	wdptr (1'bx)	Maximum Burstcount	Byteenable (8’b0000xxxx)	
			First Cycle or All Cycles	Second Cycle (If Different)
	1	32	1111	—
1110	0[23]	—	—	—
	1[23]	—	—	—
1111	0[23]	—	—	—
	1	64	1111	—

Table 23. RapidIO Master Write Transaction Burstcount and Byteenable (64-Bit Datapath) for 2x and 4x Variations

RapidIO Values		Avalon-MM Values	
wrsize (4'bxxxx)	wdptr (1'bx)	Maximum Burstcount	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[22]	1	1110_0000
	1[22]	1	0000_0111
0110	0	1	0011_0000
	1	1	0000_0011
0111	0[22]	1	1111_1000
	1[22]	1	0001_1111
1000	0	1	1111_0000
	1	1	0000_1111
1001	0[22]	1	1111_1100
	1[22]	1	0011_1111
1010	0[22]	1	1111_1110
	1[22]	1	0111_1111
1011	0	1	1111_1111

continued...

RapidIO Values		Avalon-MM Values	
wrsize (4'bxxxx)	wdptr (1'bx)	Maximum Burstcount	Byteenable (8'bxxxxxxxx)
	1	2	1111_1111
1100	0	4	1111_1111
	1	8	1111_1111
1101	0[23]	—	—
	1	16	1111_1111
1110	0[23]	—	—
	1[23]	—	—
1111	0[23]	—	—
	1	32	1111_1111

Related Information

Input/Output Master Address Mapping Registers on page 156

4.5.3.2. Input/Output Avalon-MM Master Module Timing Diagrams

Below figures shows the timing dependencies. 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 Intel RapidIO IP core, the input/output avalon-MMslave module timing diagrams 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 27. NREAD Transaction on the Input/Output Avalon-MM Master Interface

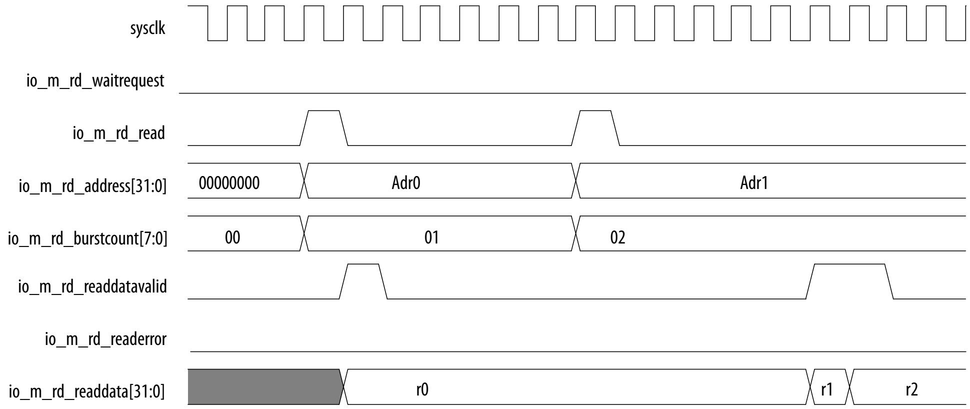
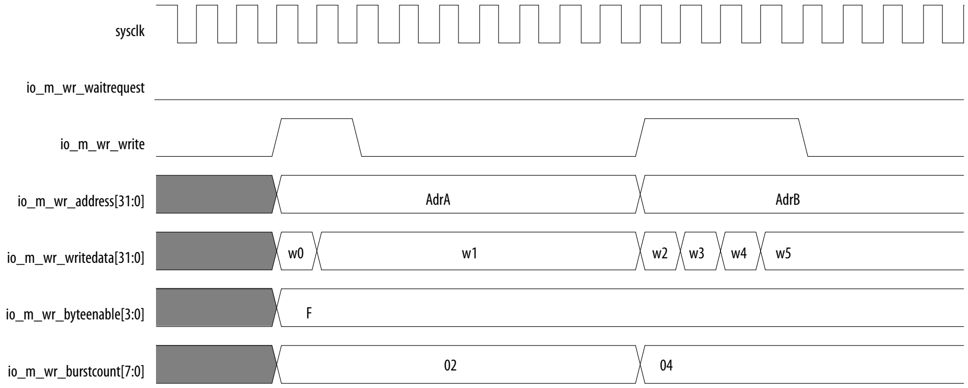


Figure 28. NWRITE Transaction on the Input/Output Avalon-MM Master Interface



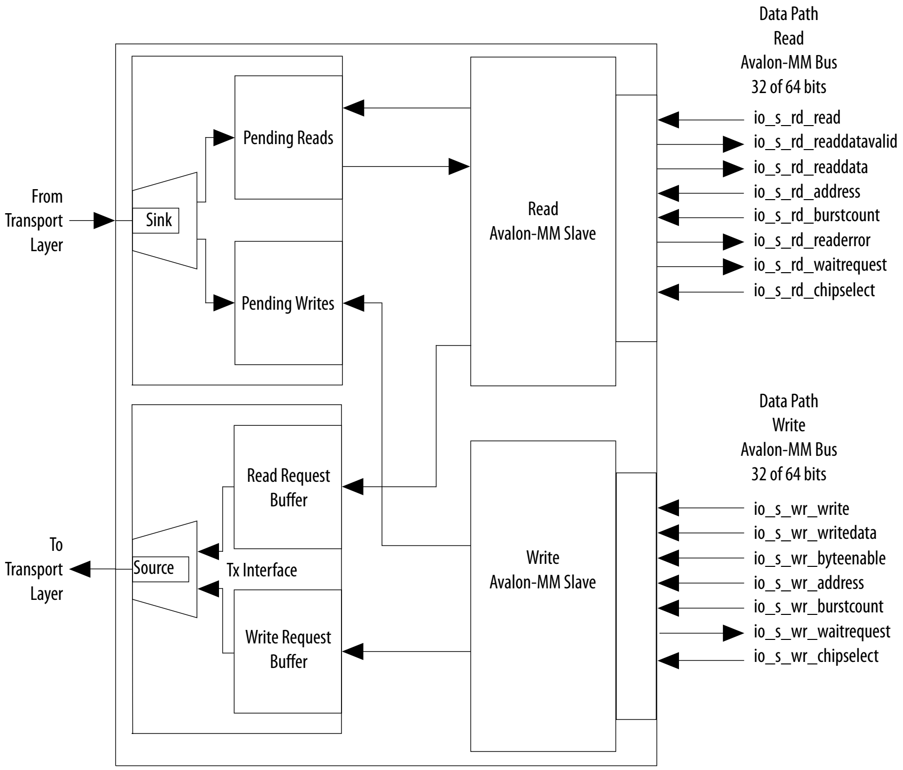
Related Information

Input/Output Avalon-MM Slave Module Timing Diagrams on page 102

4.5.3.3. 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 29. Input/Output Avalon-MM Slave Logical Layer Block Diagram



Note:

- 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 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.

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 the figure are combined, to preserve the relative order of read and write requests that appear on the Avalon-MM interface. In Intel Arria 10 and Intel Cyclone 10 GX variations, the read and write request buffers are combined.

Related Information

Physical Layer Registers on page 142
For more information about the duration of the time-out.

4.5.3.3.1. 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 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 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 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 controls a maskable interrupt in the Input/Output Slave Interrupt register 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.

Related Information

Input/Output Slave Interrupts on page 159

4.5.3.3.2. 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.

Table 24. Address Mapping and Translation Registers

Registers	Location
Input/Output slave base address	0x10400, 0x10410, 0x10420, 0x10430, 0x10440, 0x10450, 0x10460, 0x10470, 0x10480, 0x10490, 0x104A0, 0x104B0, 0x104C0, 0x104D0, 0x104E0, 0x104F0
Input/Output slave address mask	0x10404, 0x10414, 0x10424, 0x10434, 0x10444, 0x10454, 0x10464, 0x10474, 0x10484, 0x10494, 0x104A4, 0x104B4, 0x104C4, 0x104D4, 0x104E4, 0x104F4
Input/Output slave address offset	0x10408, 0x10418, 0x10428, 0x10438, 0x10448, 0x10458, 0x10468, 0x10478, 0x10488, 0x10498, 0x104A8, 0x104B8, 0x104C8, 0x104D8, 0x104E8, 0x104F8
Input/Output slave packet control information (for packet header)	0x1040C, 0x1041C, 0x1042C, 0x1043C, 0x1044C, 0x1045C, 0x1046C, 0x1047C, 0x1048C, 0x1049C, 0x104AC, 0x104BC, 0x104CC, 0x104DC, 0x104EC, 0x104FC

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.

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. Intel Arria 10 and Intel Cyclone 10 GX variations have 16 transmit address translation windows.

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,Nb'0} & mask) == (base & mask))

where N is 2 in 1x variations and 3 in 2x and 4x variations.

Let
rio_addr [33:3] = {xamo, (offset [31:3] & mask [31:3]) | ({avalon_address,Nb'0} [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.

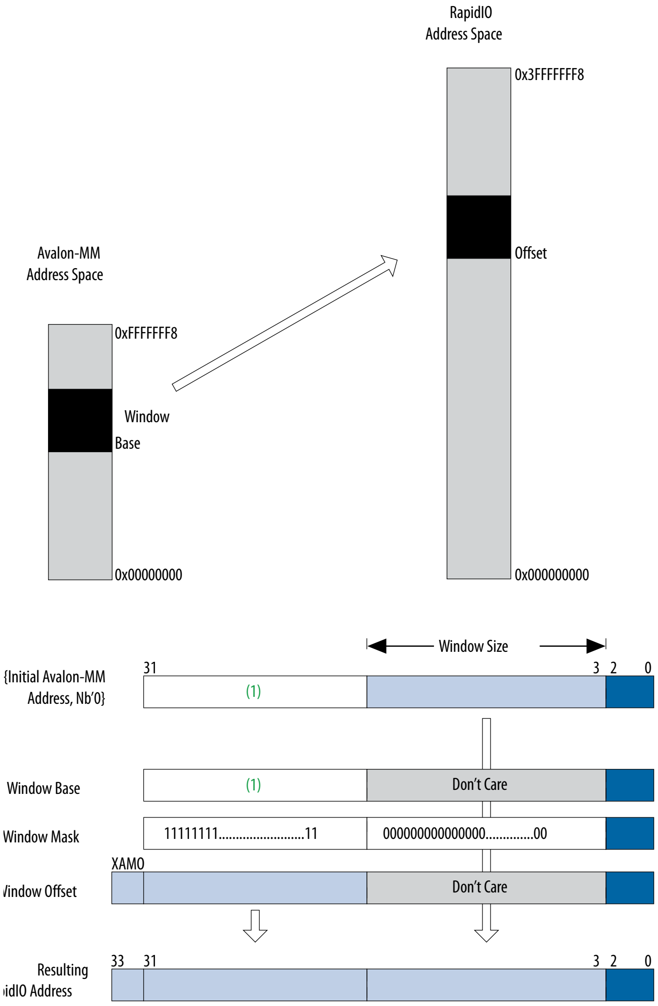
The priority and DESTINATION_ID fields are inserted from the control register.

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, 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.
- 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 30. Input/Output Slave Window Translation



Note:
These bits must have the same value in the initial Avalon-MM address and in the window

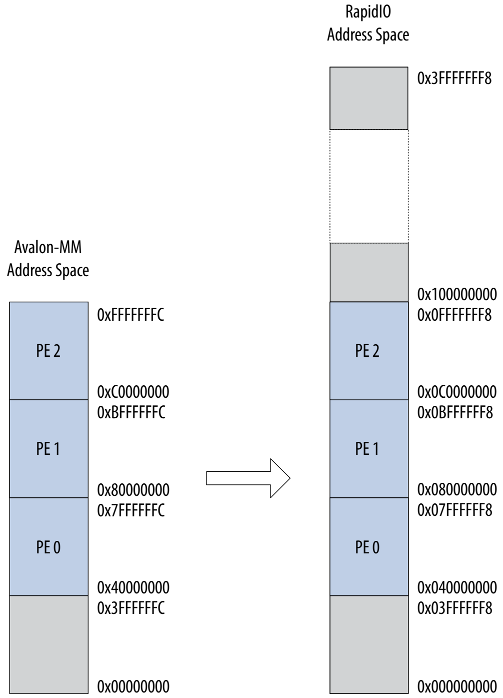
Related Information

- Input/Output Slave Mapping Registers on page 157
- Avalon-MM Burstcount and Byteenable Encoding in RapidIO Packets on page 98
- Input/Output Slave Interrupts on page 159

4.5.3.3.3. 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 below.

Figure 31. Input/Output Slave Translation Window Address Mapping

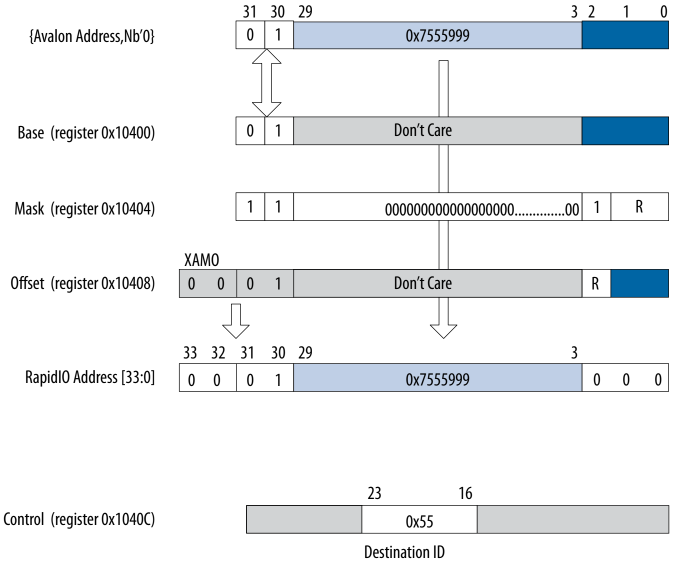


The two most significant bits of the Avalon-MM address are used to differentiate between the processing endpoints. Figures in the following sections show the address translation implemented for each window. Each figure shows the value for the destination ID of the control register for one window.

4.5.3.3.4. 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.

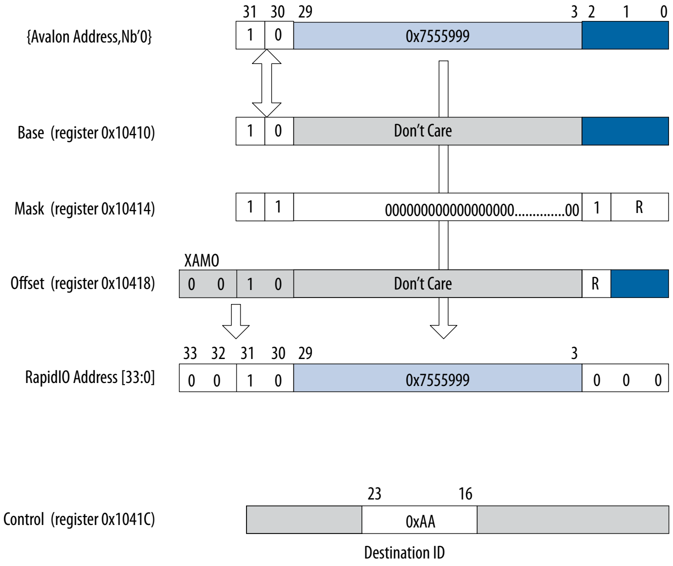
Figure 32. Translation Window 0



4.5.3.3.5. 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.

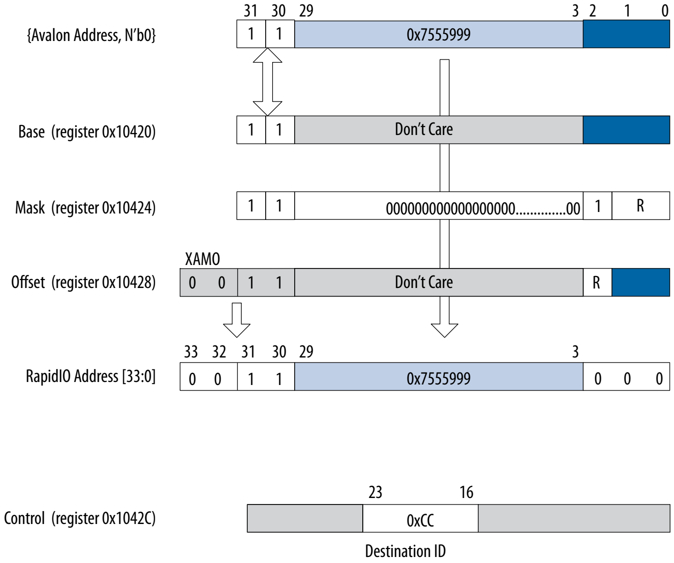
Figure 33. Translation Window 1



4.5.3.3.6. 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 34. Translation Window 2



4.5.3.4. 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.

Table 25. Read Request Size Encoding (32-bit datapath)

Avalon-MM Values		RapidIO Values	
burstcount[24]	address[0] (1'bx)[25]	wdptr (1'bx)	rdsize (4'bxxxx)[25]
1	1	0	1000
1	0	1	1000

continued...

[24] 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.

[25] 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[0] 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.

Avalon-MM Values		RapidIO Values	
burstcount[24]	address[0] (1'bx)[25]	wdptr (1'bx)	rdsize (4'bxxxx)[25]
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
41–48	0	1	1110
49–56	0	0	1111
57–64	0	1	1111

Table 26. Write Request Size Encoding (32-bit datapath)

Avalon-MM Values			RapidIO Values	
burstcount[26]	byteenable (4'bxxxx)	address [0] (1'bx)[27]	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

continued...

[24] 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.

[25] 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[0]` 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.

[26] 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.

[27] 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[0]` 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.

Avalon-MM Values			RapidIO Values	
burstcount[26]	byteenable (4'bxxxx)	address [0] (1'bx)[27]	wdptr (1'bx)	wrsize (4'bxxxx)
1	0010	0	1	0010
1	0001	0	1	0011
1	1100	1	0	0100
1	1110[28]	1	0	0101
1	0011	1	0	0110
1	1100	0	1	0100
1	0111[28]	0	1	0101
1	0011	0	1	0110
1	1111	1	0	1000
1	1111	0	1	1000
2	1111[29]	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, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64			1	1111

[26] 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.

[27] 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[0]` 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.

[28] This is not a legal Avalon-MM byteenable pattern, but the RapidIO IP core supports it if user logic generates it.

[29] 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 27. Allowed Read Request Size Encoding (64-bit datapath variations)

Avalon-MM Values	RapidIO Values	
burstcount[24]	wdptr (1'bx)	rdsize (4'bxxxx)[24]
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'b1110
21–24	1'b1	
25–28	1'b0	4'b1111
29–32	1'b1	

The following table 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 flag interrupts in the RapidIO IP core.

Table 28. 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[28]	0	0110
1	0011_0000	0	0111
1	1111_1000[28]	0	1000
1	0000_1100	1	1000
1	0000_0111[28]	1	1001
1	0000_0011	1	1001

continued...

Avalon-MM Values		**RapidIO Values**	
burstcount	**byteenable (8'bxxxx_xxxx)**	**wdptr (1'bx)**	**wrsize (4'bx)**
1	0001_1111[28]	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[28]	0	1010
1	0111_1111[28]	1	1010
1	1111_1111	0	1011
2	1111_1111[30]	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			

Related Information

- Input/Output Slave Interrupts on page 159
- Input/Output Slave Interrupts on page 159
  For more information about the relevant interrupts.

4.5.3.5. Input/Output Avalon-MM Slave Module Timing Diagrams

The following figures show the timing dependencies on the Avalon-MM slave interface for an outgoing RapidIO NREAD request and the timing dependencies on the Avalon-MM slave interface for an outgoing NWRITE transaction, respectively. 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" show the same transactions after they are transmitted on the RapidIO link and received by an Intel 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.

[30] 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.

Figure 35. NREAD Transaction on the Input/Output Avalon-MM Slave Interface

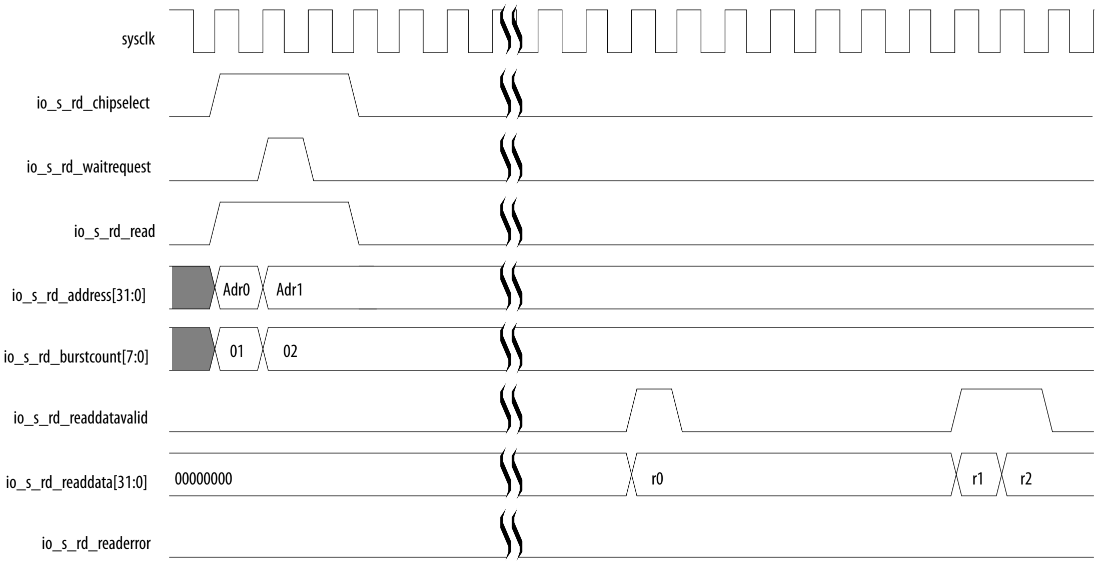
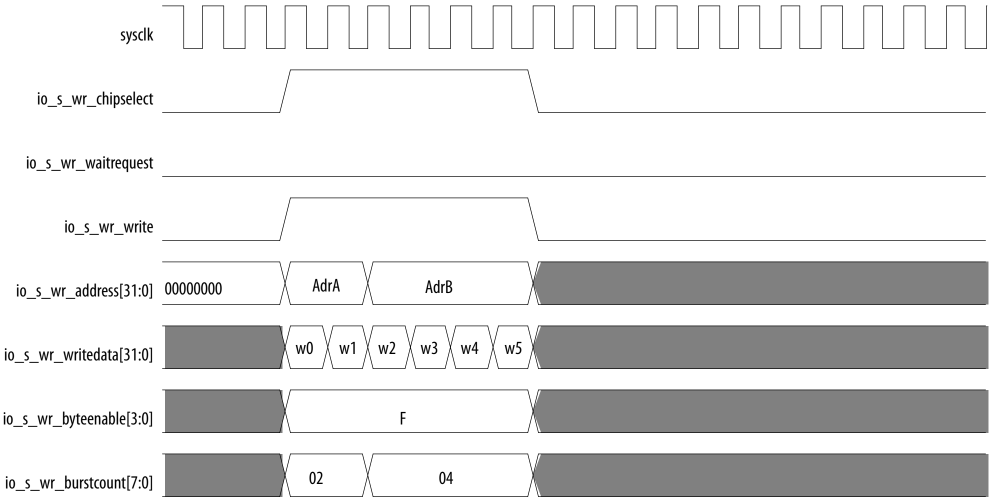


Figure 36. NWRITE Transaction on the Input/Output Avalon-MM Slave Interface



Related Information

Input/Output Avalon-MM Master Module Timing Diagrams on page 89

4.5.4. 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.

The RapidIO block diagram 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 IP core to receive, generate, or both receive and generate RapidIO DOORBELL messages.

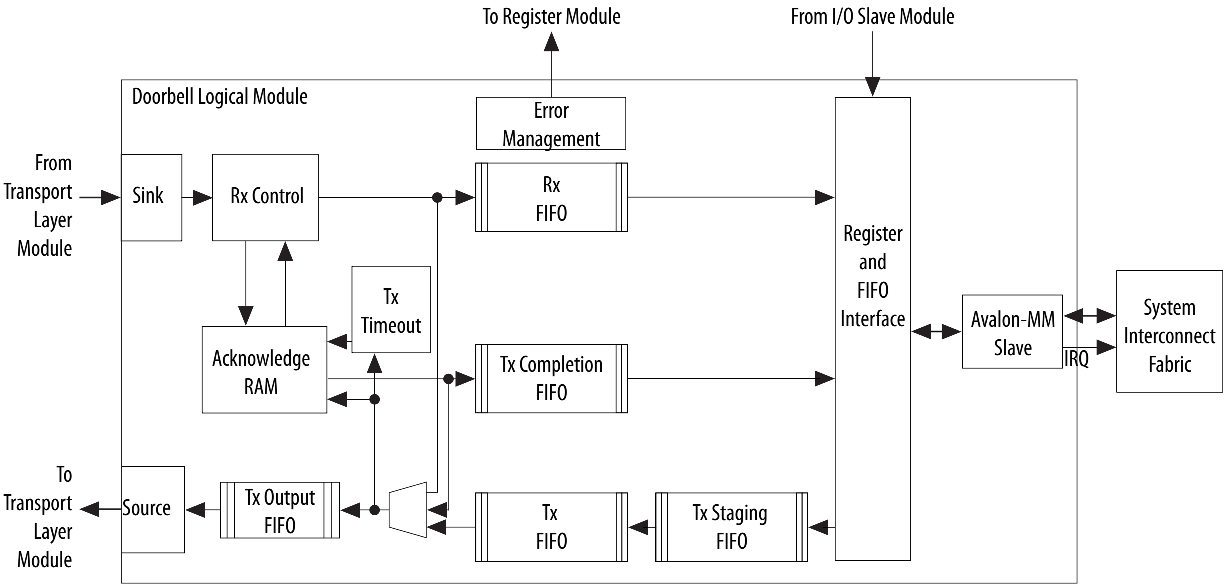
Related Information

Logical Layer Modules on page 70

4.5.4.1. Doorbell Module Block Diagram

This module includes a 32–bit Avalon-MM slave interface to the user interface.

Figure 37. Doorbell Module Block Diagram



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. Intel Arria 10 and Intel Cyclone 10 GX variations have a staging FIFO and prevent DOORBELL messages from passing write transactions.
- 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).

4.5.4.2. Preserving Transaction Order

Your RapidIO IP core Doorbell module has a Tx staging FIFO in any of the following situations:

- You select **Prevent doorbell messages from passing write transactions** in the RapidIO parameter editor.
- Your RapidIO IP core targets Intel Arria 10 and Intel Cyclone 10 GX devices.

If the module has a Tx staging FIFO, 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.

4.5.4.3. Doorbell Message Generation

To generate a DOORBELL request packet on the RapidIO serial interface, follow these steps, using the set of registers:

1. Optionally enable interrupts by writing the value 1 to the appropriate bit of the Doorbell Interrupt Enable register.
2. Optionally enable confirmation of successful outbound messages by writing 1 to the COMPLETED bit of the Tx Doorbell Status Control register.
3. Set up the priority field of the Tx Doorbell Control register.
4. Write the Tx Doorbell register to set up the DESTINATION_ID and Information fields of the generated DOORBELL packet format.

Note: 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. 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 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.

Upon detecting the interrupt, software can fetch the completed message and determine its status by reading the Tx Doorbell Completion register and Tx Doorbell Completion Status register, respectively.

An outbound DOORBELL message is assigned a time-out value based on the VALUE field of the Port Response Time-Out Control register 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.

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 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.

Related Information

- Doorbell Message Registers on page 163
- Physical Layer Registers on page 142
  For information about how the time-out value is calculated.

4.5.4.4. 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.

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 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.

Related Information

Doorbell Message Registers on page 163

4.5.5. 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. 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

Related Information

Transport Layer on page 40

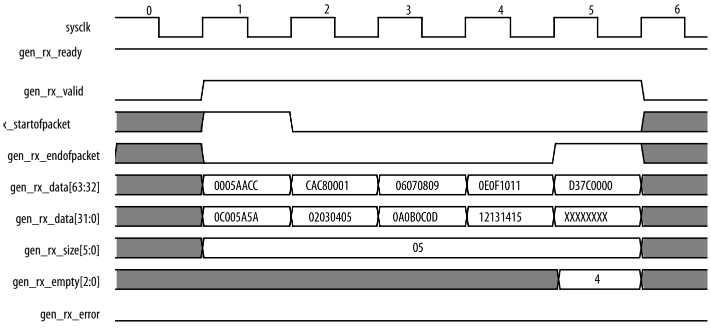
4.5.5.1. Pass-Through Interface Examples

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.

Figure 38. Packet Received on the Avalon-ST Pass-Through Interface



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

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 29. RapidIO Header Fields and gen_rx_data Bus Payload

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	[31]
	sourceID	[39:32]	8'hcc	[31]
	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	
4	Payload Byte14,15	[63:48]	16'h0e0f	
	Payload Byte16,17	[47:32]	16'h1011	

continued...

[31] In the case of 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.

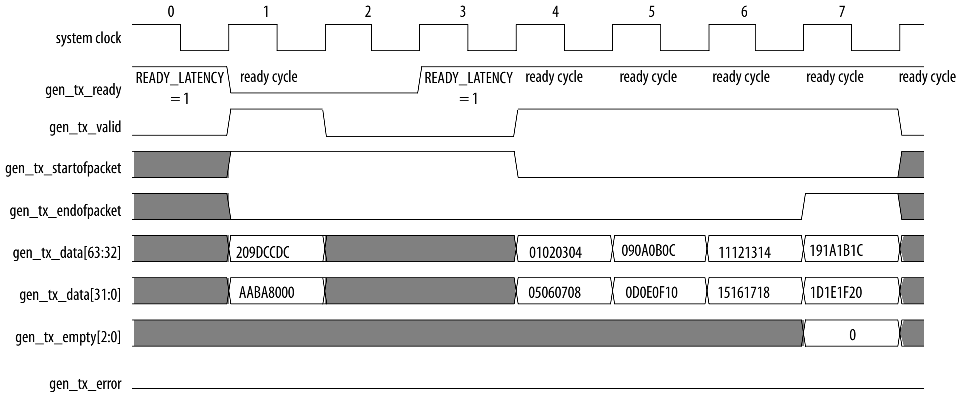
Cycle	Field	gen_rx_data bus	Value	Comment
	Payload Byte18,19	[31:16]	16'h1213	
	Payload Byte20,21	[15:0]	16'h1415	
5	CRC[15:0]	[63:48]	16'hd37c	For packets with a payload greater than 80 bytes, the first CRC field is removed but the final CRC field is not removed. For packets smaller than 80 bytes, the 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.

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 below 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 39. Packet Transmitted on the Avalon ST Pass-Through Interface



The figure shows a response to a 32-byte NREAD request in a RapidIO IP core with 16-bit device ID. The following table 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 30. RapidIO Header Fields on the gen_tx_data Bus

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	In the case of a RapidIO IP core variation with a 8-bit device ID width, the destinationID and sourceID fields shrink to a width of 8 bits each, and the fields described in the following table rows shift to the left and to an earlier clock cycle if appropriate.
	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	
5	Payload	[63:48]	16'h191a	Payload double word 3

continued...

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

4.6. 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.

4.6.1. 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.

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.

Related Information

- Physical Layer Registers on page 142
- Physical Layer Signals on page 120
  Refer to "Status Packet and Error Monitoring Signals"

4.6.1.1. 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. 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 ackIDs, can lead to unrecoverable—or fatal—errors.

Related Information

Physical Layer Error Management on page 112

4.6.1.2. Fatal Errors

Fatal errors cause a soft reset of the Physical layer module, which clears all the transmit buffers and resets the inbound and outbound 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. The following are considered fatal errors.

- 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.
- Transmitter times out while waiting for link-response control symbol.

  Note: An output port enters the Output Error Stop state when it receives a packet-not-accepted control Symbol. To exit from this state, the port issues link-request/input-status (restart-from-error) control symbol. The port waits in the Output Error Stop State for a link-response control symbol.
- Receiver times out while waiting for a link-request/input-status control symbol

  Note: Upon detection of an error, the input port enters the Input Error Stop state. To recover, the input port performs the following:

  1. Issues a packet-not-acceptedcontrol symbol
  2. Waits for link-request/Input-Status control symbol
  3. Sends link-response control symbol

  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.

Related Information

Physical Layer Registers on page 142

4.6.2. 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 with an error detected in the Physical layer is dropped in the Physical layer or 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
- `Logical/Transport Layer Error Enable` CSR
- `Logical/Transport Layer Address Capture` CSR
- `Logical/Transport Layer Device ID Capture` CSR
- `Logical/Transport Layer Control Capture` CSR

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.

Related Information

Error Management Registers on page 161

4.6.2.1. 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.

4.6.2.1.1. 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 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.)

Related Information

Physical Layer Registers on page 142

4.6.2.1.2. Registers in the Implementation Defined Space

The Maintenance register module defines the `Maintenance Interrupt` register 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 is set.

Related Information

Maintenance Interrupt Control Registers on page 153

4.6.2.1.3. 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.

4.6.2.2. 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

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

4.6.2.3. Port-Write Reception Module

The Port-Write reception module processes receive port-write request MAINTENANCE packets. The following bits in the `Maintenance Interrupt` register 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 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 or if a previously received port-write has not been read out from the `Rx Port Write Buffer` register.

Related Information

- Receive Port-Write Registers on page 156
- Maintenance Interrupt Control Registers on page 153

4.6.2.4. 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.

4.6.2.5. 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

4.6.2.5.1. 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 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.

Related Information

Physical Layer Registers on page 142

4.6.2.5.2. 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.

- 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 is set.

Related Information

Input/Output Slave Interrupts on page 159

4.6.2.5.3. 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.

4.6.2.6. 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

4.6.2.6.1. 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, 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.

Related Information

Capability Registers (CARs) on page 148

4.6.2.6.2. 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.

4.6.3. 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.

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 are added to the IP core to enable integrated error management.

Related Information

- Physical Layer Receive Buffer on page 62
- Error Management Extension Signals on page 135

5. Signals

Platform Designer (Standard) allows you to export signals with different names or prefixes. Refer to the Platform Designer (Standard) **System Contents** tab for the signals that support this capability individually, and to the Platform Designer (Standard) **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 Platform Designer (Standard) **System Contents** tab.

A **yes** entry in the **Exported by Platform Designer (Standard)** column in the following tables indicates that the signal is included in the **exported_connections** conduit in Platform Designer (Standard). A **no** entry indicates that the signal is not included in the **exported_connections** conduit in Platform Designer (Standard).

5.1. Physical Layer Signals

Below tables lists the pins used by the Physical layer of the RapidIO IP core.

Table 31. RapidIO Interface

Signal	Direction	Description	Exported by Platform Designer (Standard)
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 32. Main Clock Signals

Signal	Direction	Description
sysclk[32]	Input	Avalon system clock
clk	Input	Physical layer reference clock. In Intel Arria 10 and Intel Cyclone 10 GX variations, this clock is the reference clock for the RX CDR block in the transceiver. In other variations, this clock is also the reference clock for the TX PLL in the transceiver.

[32] 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 33. Global Signals

Signal	Direction	Description
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. Intel 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 Platform Designer (Standard) 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. In addition, reset_n should not be deasserted when the Transceiver Reconfiguration Controller reconfig_busy signal is high.
rxclk	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 for each baud rate and mode.
txclk	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 for each baud rate and mode. 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. 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.

Table 34. Status Packet and Error Monitoring

Output Signal	Clock Domain	Description	Exported by Platform Designer (Standard)
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 canceled 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 RapidIO initialization sequence has completed successfully.	yes

continued...

Output Signal	Clock Domain	Description	Exported by Platform Designer (Standard)
		This is a level signal asserted high while the initialization state machine is in the 1X_MODE, 2X_MODE, or 4X_MODE state, as described in paragraph 4.6 of *Part VI of the RapidIO Specification*.	
`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

Related Information

- Physical Layer Architecture on page 59
  For details of the I/O signals.
- General RapidIO Reset Signal Requirements on page 54
- Transceiver Signals on page 124
- Reset Requirements for Arria V, Cyclone V, and Stratix V Variations on page 55
- Baud Rates and Clock Frequencies on page 53

5.1.1. Status Packet and Error Monitoring Signals

Below table lists the status packet and error monitoring signals.

Table 35. Status Packet and Error Monitoring

Output Signal	Clock Domain	Description	Exported by Platform Designer (Standard)
`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 RapidIO initialization sequence has completed successfully. This is a level signal asserted high while the initialization state machine is in the 1X_MODE, 2X_MODE, or 4X_MODE state, as described in paragraph 4.6 of *Part VI of the RapidIO Specification*.	yes

continued...

Output Signal	Clock Domain	Description	Exported by Platform Designer (Standard)
port_error	txclk	This signal holds the value of the PORT_ERR bit of the Port 0 Error and Status CSR (offset 0x158)	yes
char_err	rxclk	Pulsed for one clock cycle when an invalid character or a valid but illegal character is detected.	yes
no_sync_indicator	rxclk	Deasserted to indicate that at least one lane is not synchronized.	yes

Related Information

Physical Layer Registers on page 142

5.1.2. Multicast Event Signals

Table 36. Multicast Event Signals

Signal	Direction	Clock Domain	Description	Exported by Platform Designer (Standard)
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

5.1.3. Receive Priority Retry Threshold-Related Signals

These signals are related to the **Receive Priority Retry Threshold** set in the RapidIO parameter editor.

Table 37. Priority Retry Threshold Signals

Signal[33]	Direction	Description	Exported by Platform Designer (Standard)
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

[33] All of these signals are in the sysclk domain.

5.1.4. Physical Layer Buffer Status Signals

Table 38. Physical Layer Buffer Status Signals

Signal[34]	Direction	Description
atxwlevel[35]	Output	Transmit buffer write level (number of free 64-byte blocks in the transmit buffer).
atxovf	Output	Transmit buffer overflow.status.
arxwlevel[35]	Output	Receive buffer write level (number of free 64-byte blocks in the receive buffer).

5.1.5. Transceiver Signals

Transceiver 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.

Table 39. Transceiver Signals

Signal	Direction	Description
cal_blk_clk[36]	Input	The Arria II GX, Arria II GZ, Cyclone IV 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. This signal is not present in Arria V, Intel Arria 10, Cyclone V, Intel Cyclone 10 GX, or Stratix V variations.
phy_mgmt_clk[36]	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.
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. Fig: *Circuit to Also Ensure Synchronous Assertion of phy_mgmt_clk_reset with reset_n* 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 Transceiver Reconfiguration Controller reconfig_busy signal is high.
rxgxbclk	Output	Transceiver receiver clock (recovered clock).

continued...

[34] All of these signals are in the sysclk domain.

[35] The formula log2(size of the transmit/receive buffer in bytes/64)+1 determines the width of this signal in bits. For example, a transmit or receive buffer size of 16 KBytes would give: log2(16×1024/64)+1= 9 bits (for example, [8:0]).

[36] You connect this clock inside the Platform Designer (Standard) tool. If you connect it to an external clock, a port with the name of that external clock is added to your Platform Designer (Standard) system and this clock is connected to it.

Signal	Direction	Description
reconfig_clk	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, Intel Arria 10, Cyclone V, Intel Cyclone 10 GX, or Stratix V variations.
reconfig_togxb	Input	Driven from an external dynamic reconfiguration block. Supports the selection of multiple transceiver channels for dynamic reconfiguration. Note that not using a dynamic reconfiguration block that enables offset cancellation results in a non-functional hardware design. 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, 2, or 4. This width supports communication from 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 Reconfiguration Controller from your simulation model, you must ensure all bits of this bus are tied to 0. This signal is not present in Intel Arria 10 and Intel Cyclone 10 GX variations.
reconfig_fromgxb	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. However, not using a dynamic reconfiguration block that enables offset cancellation results in a non-functional hardware design. 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, 2, or 4. This width supports communication from the transceiver to C + 1 reconfiguration interfaces in Reconfiguration Controller, one interface dedicated to each channel and an additional interface for the transceiver PLL. This signal is not present in Intel Arria 10 and Intel Cyclone 10 GX variations.
gxbpll_locked	Output / Input	Indicates the transceiver transmitter PLL is locked to the reference clock. In Intel Arria 10 and Intel Cyclone 10 GX variations, this is an input signal to the RapidIO IP core that is intended to be connected to the external PLL; in all other variations, this is an output signal from the transceiver PLL in the RapidIO IP core.
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, Intel Arria 10, Cyclone V, Intel Cyclone 10 GX, or Stratix V variations.
rx_errdetect	Output	Transceiver 8B10B code group violation signal bus. The signal width depends on the IP core mode.
tx_bonding_clocks_ch0[5:0]	Input	Transceiver channel TX input clocks for RapidIO lane 0. This signal is available only in Intel Arria 10 and Intel Cyclone 10 GX variations. Each transceiver channel that corresponds to a RapidIO lane has six input clock bits. The bits are expected to be driven from a TX PLL.
tx_bonding_clocks_ch1[5:0]	Input	Transceiver channel TX input clocks for RapidIO lane 1. This signal is available only in Intel Arria 10 and Intel Cyclone 10 GX 2x and 4x variations. Each transceiver channel that corresponds to a RapidIO lane has six input clock bits. The bits are expected to be driven from a TX PLL.

continued...

Signal	Direction	Description
tx_bonding_clocks_ch2[5:0]	Input	Transceiver channel TX input clocks for RapidIO lane 2. This signal is available only in Intel Arria 10 and Intel Cyclone 10 GX 4x variations. Each transceiver channel that corresponds to a RapidIO lane has six input clock bits. The bits are expected to be driven from a TX PLL.
tx_bonding_clocks_ch3[5:0]	Input	Transceiver channel TX input clocks for RapidIO lane 3. This signal is available only in Intel Arria 10 and Intel Cyclone 10 GX 4x variations. Each transceiver channel that corresponds to a RapidIO lane has six input clock bits. The bits are expected to be driven from a TX PLL.
tx_analogreset [<number of lanes>-1:0]	Input	You must connect these signals to Transceiver PHY Reset Controller IP core, which implements the appropriate reset sequence for the device. Connect each signal to the corresponding signal in the Transceiver PHY Reset Controller IP core. These signals are available only in Intel Arria 10 and Intel Cyclone 10 GX IP core variations.
rx_analogreset [<number of lanes>-1:0]	Input	
tx_digitalreset [<number of lanes>-1:0]	Input	
rx_digitalreset [<number of lanes>-1:0]	Input	
rx_is_lockedtodata [<number of lanes>-1:0]	Output	
tx_cal_busy [<number of lanes>-1:0]	Output	
rx_cal_busy [<number of lanes>-1:0]	Output	

Each of these individual interfaces is an Avalon-MM interface you use to access the hard registers for the corresponding transceiver channel on the Intel Arria 10 and Intel Cyclone 10 GX devices. These signals are available if you turn on **Enable transceiver dynamic reconfiguration** in the RapidIO parameter editor.

Table 40. Intel Arria 10 and Intel Cyclone 10 GX Transceiver Dynamic Reconfiguration Avalon-MM Interface Signals

Signal	Direction	Description
reconfig_clk_ch0	Input	Intel Arria 10/Intel Cyclone 10 GX dynamic reconfiguration interface clock for the transceiver channel configured for RapidIO lane 0.
reconfig_reset_ch0	Input	Intel Arria 10/Intel Cyclone 10 GX dynamic reconfiguration interface reset for the transceiver channel configured for RapidIO lane 0.
reconfig_waitrequest_ch0	Output	Intel Arria 10/Intel Cyclone 10 GX dynamic reconfiguration slave wait request for the transceiver channel configured for RapidIO lane 0. The RapidIO IP core uses this signal to stall the requestor on the interconnect.
reconfig_read_ch0	Input	Intel Arria 10/Intel Cyclone 10 GX dynamic reconfiguration slave read request for the transceiver channel configured for RapidIO lane 0.
reconfig_write_ch0	Input	Intel Arria 10/Intel Cyclone 10 GX dynamic reconfiguration slave write request for the transceiver channel configured for RapidIO lane 0.
reconfig_address_ch0[9:0]	Input	Intel Arria 10/Intel Cyclone 10 GX dynamic reconfiguration slave address bus for the transceiver channel configured for RapidIO lane 0. The address is a word address, not a byte address.

continued...

Signal	Direction	Description
`reconfig_writedata_ch0[31:0]`	Input	Intel Arria 10/Intel Cyclone 10 GX dynamic reconfiguration slave write data bus for the transceiver channel configured for RapidIO lane 0.
`reconfig_readdata_ch0[31:0]`	Output	Intel Arria 10/Intel Cyclone 10 GX dynamic reconfiguration slave read data bus for the transceiver channel configured for RapidIO lane 0.
`reconfig_clk_ch1`	Input	Intel Arria 10/Intel Cyclone 10 GX dynamic reconfiguration interface clock for the transceiver channel configured for RapidIO lane 1. This signal is available only in 2x and 4x variations.
`reconfig_reset_ch1`	Input	Intel Arria 10/Intel Cyclone 10 GX dynamic reconfiguration interface reset for the transceiver channel configured for RapidIO lane 1. This signal is available only in 2x and 4x variations.
`reconfig_waitrequest_ch1`	Output	Intel Arria 10/Intel Cyclone 10 GX dynamic reconfiguration slave wait request for the transceiver channel configured for RapidIO lane 1. The RapidIO IP core uses this signal to stall the requestor on the interconnect. This signal is available only in 2x and 4x variations.
`reconfig_read_ch1`	Input	Intel Arria 10/Intel Cyclone 10 GX dynamic reconfiguration slave read request for the transceiver channel configured for RapidIO lane 1. This signal is available only in 2x and 4x variations.
`reconfig_write_ch1`	Input	Intel Arria 10 dynamic reconfiguration slave write request for the transceiver channel configured for RapidIO lane 1. This signal is available only in 2x and 4x variations.
`reconfig_address_ch1[9:0]`	Input	Intel Arria 10/Intel Cyclone 10 GX dynamic reconfiguration slave address bus for the transceiver channel configured for RapidIO lane 1. The address is a word address, not a byte address. This signal is available only in 2x and 4x variations.
`reconfig_writedata_ch1[31:0]`	Input	Intel Arria 10/Intel Cyclone 10 GX dynamic reconfiguration slave write data bus for the transceiver channel configured for RapidIO lane 1. This signal is available only in 2x and 4x variations.
`reconfig_readdata_ch1[31:0]`	Output	Intel Arria 10/Intel Cyclone 10 GX dynamic reconfiguration slave read data bus for the transceiver channel configured for RapidIO lane 1. This signal is available only in 2x and 4x variations.
`reconfig_clk_ch2`	Input	Intel Arria 10/Intel Cyclone 10 GX dynamic reconfiguration interface clock for the transceiver channel configured for RapidIO lane 2. This signal is available only in 4x variations.
`reconfig_reset_ch2`	Input	Intel Arria 10 dynamic reconfiguration interface reset for the transceiver channel configured for RapidIO lane 2. This signal is available only in 4x variations.
`reconfig_waitrequest_ch2`	Output	Intel Arria 10/Intel Cyclone 10 GX dynamic reconfiguration slave wait request for the transceiver channel configured for RapidIO lane 2. The RapidIO IP core uses this signal to stall the requestor on the interconnect. This signal is available only in 4x variations.
`reconfig_read_ch2`	Input	Intel Arria 10/Intel Cyclone 10 GX dynamic reconfiguration slave read request for the transceiver channel configured for RapidIO lane 2. This signal is available only in 4x variations.
`reconfig_write_ch2`	Input	Intel Arria 10/Intel Cyclone 10 GX dynamic reconfiguration slave write request for the transceiver channel configured for RapidIO lane 2. This signal is available only in 4x variations.
`reconfig_address_ch2[9:0]`	Input	Intel Arria 10/Intel Cyclone 10 GX dynamic reconfiguration slave address bus for the transceiver channel configured for RapidIO lane 2. The address is a word address, not a byte address.

continued...

Signal	Direction	Description
		This signal is available only in 4x variations.
`reconfig_writedata_ch2[31:0]`	Input	Intel Arria 10/Intel Cyclone 10 GX dynamic reconfiguration slave write data bus for the transceiver channel configured for RapidIO lane 2. This signal is available only in 4x variations.
`reconfig_readdata_ch2[31:0]`	Output	Intel Arria 10/Intel Cyclone 10 GX dynamic reconfiguration slave read data bus for the transceiver channel configured for RapidIO lane 2. This signal is available only in 4x variations.
`reconfig_clk_ch3`	Input	Intel Arria 10/Intel Cyclone 10 GX dynamic reconfiguration interface clock for the transceiver channel configured for RapidIO lane 3. This signal is available only in 4x variations.
`reconfig_reset_ch3`	Input	Intel Arria 10 dynamic reconfiguration interface reset for the transceiver channel configured for RapidIO lane 3. This signal is available only in 4x variations.
`reconfig_waitrequest_ch3`	Output	Intel Arria 10/Intel Cyclone 10 GX dynamic reconfiguration slave wait request for the transceiver channel configured for RapidIO lane 3. The RapidIO IP core uses this signal to stall the requestor on the interconnect. This signal is available only in 4x variations.
`reconfig_read_ch3`	Input	Intel Arria 10/Intel Cyclone 10 GX dynamic reconfiguration slave read request for the transceiver channel configured for RapidIO lane 3. This signal is available only in 4x variations.
`reconfig_write_ch3`	Input	Intel Arria 10/Intel Cyclone 10 GX dynamic reconfiguration slave write request for the transceiver channel configured for RapidIO lane 3. This signal is available only in 4x variations.
`reconfig_address_ch3[9:0]`	Input	Intel Arria 10/Intel Cyclone 10 GX dynamic reconfiguration slave address bus for the transceiver channel configured for RapidIO lane 3. The address is a word address, not a byte address. This signal is available only in 4x variations.
`reconfig_writedata_ch3[31:0]`	Input	Intel Arria 10/Intel Cyclone 10 GX dynamic reconfiguration slave write data bus for the transceiver channel configured for RapidIO lane 3. This signal is available only in 4x variations.
`reconfig_readdata_ch3[31:0]`	Output	Intel Arria 10/Intel Cyclone 10 GX dynamic reconfiguration slave read data bus for the transceiver channel configured for RapidIO lane 3. This signal is available only in 4x variations.

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. For details, go to the appropriate device handbook. For more information about offset cancellation, refer to the relevant device handbook.

Related Information

- Transceiver Architecture in Arria II Devices

- Cyclone IV Transceivers Architecture
- Transceiver Architecture in Stratix IV Devices
- Intel Cyclone 10 GX Transceiver PHY User Guide
- Reset Requirements for Arria V, Cyclone V, and Stratix V Variations on page 55
- V-Series Transceiver PHY IP Core User Guide
  For more information about the Reconfiguration Controller component.
- Intel Arria 10 Transceiver PHY User Guide
- External Transceiver PLL on page 30
- Instantiating Multiple RapidIO IP Cores on page 34
  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.
- Getting Started on page 17
- Device Options on page 37

5.1.6. Register-Related Signals

Table 41. Register-Related Signals

Signal	Direction	Clock Domain	Description
`ef_ptr[15:0]`	Input	`txclk`	Most significant bits [31:16] of the PHEAD0 register.
`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.
`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 variations that include the Avalon-ST pass-through interface.
`input_enable`[37]	input	txclk	Enables the port to respond to any RapidIO packets. There are two ways the user can enable the port: • Driving this signal high- 1'b1 • Writing 1'b1 to bit 21 of `Port 0 Control CSR` register.[38]
`output_enable`[37]	input	txclk	Enables the port to issue RapidIO packets. There are two ways the user can enable the port: • Driving this signal high- 1'b1 • Writing 1'b1 to bit 22 of `Port 0 Control CSR` register.[39]

[37] This signal is available only in Intel Arria 10 and Intel Cyclone 10 GX IP core variations.

[38] The IP core performs an OR operation between `input_enable` and bit 21 of `Port 0 Control CSR` register.

Related Information

Physical Layer Registers on page 142
For information about Port 0 Control CSR register.

5.2. Transport and Logical Layer Signals

Below are the signals used by the Transport layer and the Maintenance, Input/Output, and Doorbell Logical layer modules of the RapidIO IP core.

Related Information

Physical Layer Signals on page 120
For a list of descriptions of the pins and signals used and generated by the Physical layer, including the RapidIO IP core clock signals.

5.2.1. Avalon-MM Interface Signals

Signals on Avalon-MM interfaces are in the Avalon system clock domain.

When you instantiate the IP core in Platform Designer (Standard), these signals are automatically connected and are not visible as inputs or outputs of the system.

Table 42. 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. This address is a word address (addresses a 4-byte (32-bit) word), not a byte address.
`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 43. 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

continued...

[39] The IP core performs an OR operation between `output_enable` and bit 22 of `Port 0 Control` CSR register.

Signal	Direction	Description
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 44. 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[23:0]	Input	Maintenance slave address bus. This address is a word address (addresses a 4-byte (32-bit) word), not a byte address.
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 minus N) - 1) — the **I/O slave address width** value is defined in the RapidIO parameter editor. N is 2 for 1x variations and 3 for 2x and 4x variations.

The internal datapath width is 32 bits in RapidIO 1x variations, and 64 bits in RapidIO 2x and 4x variations.

Table 45. 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.

continued...

Signal	Direction	Description
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 46. 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 47. Input/Output Slave Datapath Write Avalon-MM Interface Signals

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.
io_s_wr_address[j:0]	Input	Input/Output slave address bus. In 1x variations, this address is a word address (addresses a 4-byte (32-bit) word), not a byte address. In 2x and 4x variations, this address is a double-word address (addresses an 8-byte (64-bit) word).
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 48. 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.

continued...

Signal	Direction	Description
`io_s_rd_address[j:0]`	Input	Input/Output slave address bus. In 1x variations, this address is a word address (addresses a 4-byte (32-bit) word), not a byte address. In 2x and 4x variations, this address is a double-word address (addresses an 8-byte (64-bit) word).
`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 49. 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[3:0]`	Input	Doorbell address bus. This address is a word address (addresses a 4-byte (32-bit) word), not a byte address.
`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

Related Information

Avalon Interface Specifications
For more information about these signals.

5.2.2. Avalon-ST Pass-Through Interface Signals

When you instantiate the IP core with Platform Designer (Standard), these signals are automatically connected and are not visible as inputs or outputs of the system.

Table 50. Avalon-ST Pass-Through Tx 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.

continued...

<table>
<tr><th>Signal</th><th>Type</th><th colspan="2">Function</th></tr>
<tr><td></td><td></td><td colspan="2">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.</td></tr>
<tr><td>gen_tx_valid[40]</td><td>Input</td><td colspan="2">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.</td></tr>
<tr><td>gen_tx_startofpacket</td><td>Input</td><td colspan="2">Marks the active cycle containing the start of the packet[40]</td></tr>
<tr><td>gen_tx_endofpacket</td><td>Input</td><td colspan="2">Marks the active cycle containing the end of the packet.[40]</td></tr>
<tr><td>gen_tx_data</td><td>Input</td><td colspan="2">A 32-bit wide data bus for 1x variations, or a 64-bit wide data bus for 2x or 4x variations. Carries the bulk of the information transferred from the source to the sink.[40]</td></tr>
<tr><td rowspan="2">gen_tx_empty</td><td rowspan="2">Input</td><td colspan="2">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:</td></tr>
<tr><td>32-bit bus:
2'b0X none
2'b1X [15:0]</td><td>64-bit bus:
3'b00X none
3'b01X [15:0]
3'b10X [31:0]
3'b11X [47:0]</td></tr>
<tr><td>gen_tx_error</td><td>Input</td><td colspan="2">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.[40]</td></tr>
</table>

Table 51. Avalon-ST Pass-Through Rx 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[41]	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.
gen_rx_startofpacket	Output	Marks the active cycle containing the start of the packet.[41]
gen_rx_endofpacket	Output	Marks the active cycle containing the end of the packet.[41]
gen_rx_data	Output	A 32-bit wide data bus for 1x mode, or a 64-bit wide data bus for 2x or 4x mode.[41]

continued...

[40] This signal is used to qualify all the other input signals of the transmit side of the Avalon-ST pass-through interface.

[41] This signal is used to qualify all the other output signals of the receive side Avalon-ST pass-through interface.

<table>
<tr><th>Signal</th><th>Type</th><th colspan="2">Function</th></tr>
<tr><td rowspan="3">gen_rx_empty</td><td rowspan="3">Output</td><td colspan="2">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 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:</td></tr>
<tr><td>32-bit bus:
2'b0X none
2'b1X [15:0]</td><td>64-bit bus:
3'b00X none
3'b01X [15:0]
3'b10X [31:0]
3'b11X [47:0]</td></tr>
<tr><td colspan="2">If the received number of bytes, including padding and CRC, is a multiple of four (for a 32-bit wide data bus) or a multiple of eight (for a 64-bit wide data bus), the value of gen_rx_empty is zero.
The value of gen_rx_empty does not tell you whether the final 16 bits of the data transfer are padding or CRC bits; your custom logical layer application must decode the header fields to determine how to interpret the received bits.</td></tr>
<tr><td>gen_rx_size[42]</td><td>Output</td><td colspan="2">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.[41]</td></tr>
<tr><td>gen_rx_error</td><td>Output</td><td colspan="2">Indicates that the corresponding data has an error. This signal is never asserted by the RapidIO IP core.[41]</td></tr>
</table>

Related Information

- Avalon Interface Specifications
  For more information about these signals.
- CRC Checking and Removal on page 60

5.2.3. Error Management Extension Signals

These signals 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 52. Message Passing Error Management Input Ports

Signal[43], [44]	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.

continued...

[42] This is not an Avalon-ST signal. The gen_rx_size signal is exported when the RapidIO IP core is part of a Platform Designer (Standard) system.

[43] All of these signals are included in the rio_data_messages conduit bundle in Platform Designer (Standard). 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.

[44] All of these input signals are sampled in the Avalon system clock domain.

Signal[43], [44]	Description
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.
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.

[43] All of these signals are included in the rio_data_messages conduit bundle in Platform Designer (Standard). 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.

[44] All of these input signals are sampled in the Avalon system clock domain.

5.2.4. Packet and Error Monitoring Signal for the Transport Layer

Table 53. Transport Layer Packet and Error Monitoring Signal

Signal	Clock Domain	Direction	Description	Exported by Platform Designer (Standard)
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

Related Information

Status Packet and Error Monitoring Signals on page 122
For information on Physical layer packet and error monitoring signals.

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.

Note: Reserved fields are labeled 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 Platform Designer (Standard) system or by a user-provided design.

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 54. 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 55. Memory Map

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
0x68	Host Base Device ID Lock	Maintenance module

continued...

Address	Name	Used by
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

continued...

Address	Name	Used by
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
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

Related Information

- Doorbell Message Registers on page 163
- Maintenance Module on page 74

6.1. Physical Layer Registers

The offset values are defined by the RapidIO standard.

Table 56. Physical Layer Register Map

Address	Name	Description
0x100	PHEAD0	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
0x158	ERRSTAT	Port 0 Error and Status CSR
0x15C	PCTRL0	Port 0 Control CSR

Table 57. PHEAD0—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 58. PHEAD1—Reserved Register—0x104

Field	Bits	Access	Function	Default
RSRV	[31:0]	UR0	Reserved	32'h0

Table 59. 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 60. PRTCTRL—Port Response Time-Out Control CSR—0x124

Field	Bits	Access	Function	Default
VALUE	[31:8]	RW	Time-out internal value.	24'hFF_FFFF

continued...

Field	Bits	Access	Function	Default
			• 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.	
RSRV	[7:0]	UR0	Reserved	8'h0

Table 61. 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 62. 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 63. Port 0 Error and Status CSR—Offset: 0x158

Field[45]	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 Input 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
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.	1'b0

continued...

[45] Refer to *Error Detection and Management* for details

Field[45]	Bits	Access	Function	Default
			• 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: Issue a `packet-not-accepted` control symbol. Wait for `link-request/input-status` control symbol. Send `link-response` control symbol.	
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
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: 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. 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". 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:	1'b0

continued...

[45] Refer to *Error Detection and Management* for details

Field[45]	Bits	Access	Function	Default
			• 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.	
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

Table 64. Port 0 Control CSR—Offset: 0x15C

Field	Bits	Access	Function	Default
PORT_WIDTH	[31:30]	RO	Hardware width of the port: 'b00—Single-lane port. 'b10—Two-lane port. 'b01—Four-lane port. ''b11—Reserved.	2'b00 (for 1x variations), 2'b10 (for 2x variations), 2'b01 (for 4x variations)[46]
INIT_WIDTH	[29:27]	RO	Width of the ports after being initialized: 'b000—Single lane port, lane 0. 'b001—Single lane port, redundancy lane (lane 1 for 2x variations and lane 2 for 4x variations). 'b010—Four-lane port. 'b011—Two-lane port. `b100 – 'b111—Reserved.	3'b000 (for 1x variations), 3'b011 (for 2x variations), 3'b010 (for 4x variations)
PWIDTH_OVRIDE	[26:24]	URO	Soft port configuration to override the hardware size: 'b000—No override. 'b001—Reserved. 'b010—Force single lane, lane 0. 'b011—Force single lane, redundancy lane. 'b100–'b111—Reserved.	3'b000
PORT_DIS	[23]	RW	Port disable: 'b0—Port receivers/drivers are enabled.	1'b0

continued...

[45] Refer to *Error Detection and Management* for details

[46] Reflects the choice made in the RapidIO parameter editor.

Field	Bits	Access	Function	Default
			'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.	
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
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

Related Information

- Error Detection and Management on page 112

- Fatal Errors on page 113

6.2. Transport and Logical Layer Registers

This section lists the Transport and Logical layer registers. This address space is accessible to the user through the System Maintenance Avalon-MM slave interface.

6.2.1. Capability Registers (CARs)

Table 65. Device Identity CAR—Offset: 0x00

Field	Bits	Access	Function	Default
DEVICE_ID	[31:16]	RO	Hard-wired device identifier	[47]
VENDOR_ID	[15:0]	RO	Hard-wired device vendor identifier	[47]

Table 66. Device Information CAR—Offset: 0x04

Field	Bits	Access	Function	Default
DEVICE_REV	[31:0]	RO	Hard-wired device revision level	[47]

Table 67. Assembly Identity CAR—Offset: 0x08

Field	Bits	Access	Function	Default
ASSY_ID	[31:16]	RO	Hard-wired assembly identifier	[47]
ASSY_VENDOR_ID	[15:0]	RO	Hard-wired assembly vendor identifier	[47]

Table 68. Assembly Information CAR—Offset: 0x0C

Field	Bits	Access	Function	Default
ASSY_REV	[31:16]	RO	Hard-wired assembly revision level	[47]
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.	[47]

Table 69. Processing Element Features CAR—Offset: 0x10

Field	Bits	Access	Function	Default
BRIDGE	[31]	RO	Processing element can bridge to another interface.	[47]
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.	[47]
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.	[47]
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.	[47]

continued...

[47] The default value is set in the RapidIO parameter editor.

Field	Bits	Access	Function	Default
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
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.	(47)
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

Table 70. 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	(47)
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.	(47)

Table 71. Source Operations CAR—Offset: 0x18

Field[48]	Bits	Access	Function	Default
RSRV	[31:16]	RO	Reserved	16'h0
READ	[15]	RO	Processing element can support a read operation	[49]
WRITE	[14]	RO	Processing element can support a write operation	[49]
SWRITE	[13]	RO	Processing element can support a streaming-write operation	[49]
NWRITE_R	[12]	RO	Processing element can support a write-with-response operation	[49]
Data Message	[11]	RO	Processing element can support data message operation	[47]
DOORBELL	[10]	RO	Processing element can support a DOORBELL operation	[50]
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	[51]
Implementation Defined	[1:0]	RO	Reserved for this implementation	2'b00

[48] If one of the Logical layers supported by the RapidIO IP core 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.

[49] 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.

[50] 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.

[51] 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 72. Destination Operations CAR—Offset: 0x1C

Field[52]	Bits	Access	Comment	Default
RSRV	[31:16]	RO	Reserved	16'h0
READ	[15]	RO	Processing element can support a read operation	[53]
WRITE	[14]	RO	Processing element can support a write operation	[53]
SWRITE	[13]	RO	Processing element can support a streaming-write operation	[53]
NWRITE_R	[12]	RO	Processing element can support a write-with-response operation	[53]
Data Message	[11]	RO	Processing element can support data message operation	[47]
DOORBELL	[10]	RO	Processing element can support a DOORBELL operation	[54]
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	[55]
Implementation Defined	[1:0]	RO	Reserved for this implementation	2'b00

[52] If none of the Logical layers supported by the RapidIO IP core 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.

[53] 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.

[54] 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.

[55] 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.

6.2.2. Command and Status Registers (CSRs)

Table 73. 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 74. 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 75. Local Configuration Space Base Address 1 CSR—Offset: 0x5C

Field[56]	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

Table 76. Base Device ID CSR—Offset: 0x60

Field	Bits	Access	Function	Default
RSRV	[31:24]	RO	Reserved	8'h0
DEVICE_ID[57]	[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 [57]	[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.	

[56] 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.

[57] 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 77. 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[58]	This is the base device ID for the processing element that is initializing this processing element.	16'hFFFF

Table 78. 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

6.2.3. Maintenance Interrupt Control Registers

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 79. Maintenance Interrupt—Offset: 0x10080

Field	Bits	Access	Function	Default
RSRV	[31:7]	RO	Reserved	25'h0
PORT_WRITE_ERROR	[6]	RW1C	Port-write error	1'b0
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

[58] Write once; can be reset. See Part 3 of the *RapidIO Specification Rev 2.1* for more information.

Table 80. 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

6.2.4. Receive Maintenance Registers

Table 81. 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

6.2.5. Transmit 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 below the table titles.

Table 82. 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 83. 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 84. 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 85. 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

Related Information

Maintenance Slave Processor on page 77
For more details on how to use these windows.

6.2.6. Transmit Port-Write Registers

Table 86. 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.	8'h0
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 87. Tx Port Write Status—Offset: 0x10204

Field	Bits	Access	Function	Default
RSRV	[31:0]	RO	Reserved	31'h0

Table 88. 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

Related Information

Port-Write Processor on page 81

6.2.7. Receive Port-Write Registers

Table 89. 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 90. 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 91. 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

Related Information

Port-Write Reception Module on page 116
For information about receiving port write MAINTENANCE packets.

6.2.8. Input/Output Master Address Mapping 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 below the table titles.

Table 92. 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 93. 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 94. 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

Related Information

Input/Output Avalon-MM Master Address Mapping Windows on page 83

6.2.9. Input/Output Slave Mapping 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 below the table titles.

Table 95. 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 96. 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 97. 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 98. 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.(59)	1'b0
NWRITE_R_ENABLE	[0]	RW	NWRITE_R enable. (59)	1'b0

Related Information

- Input/Output Avalon-MM Slave Address Mapping Windows on page 93
- Avalon-MM Burstcount and Byteenable Encoding in RapidIO Packets on page 98
- Input/Output Slave Interrupts on page 159

6.2.10. Input/Output Slave Interrupts

These interrupt bits assert the `sys_mnt_s_irq` signal if the corresponding interrupt bit is enabled.

Table 99. Input/Output Slave Interrupt—Offset: 0x10500

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.	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
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 100. 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

continued...

[59] 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.

Field	Bits	Access	Function	Default
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 101. 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 102. 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 103. 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 canceled. Count increments when the write-request packet is sent to the Transport layer, or when a write transaction is canceled. This counter rolls over to 0 after its maximum value.	16'b0

6.2.11. Transport Layer Feature Register

This register controls the Transport layer mode.

Table 104. 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.	[60]

6.2.12. 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 105. Logical/Transport Layer Error Detect CSR—Offset: 0x10800

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
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

[60] The default (reset) value is set in the RapidIO parameter editor for variations other than Intel Arria 10 and Intel Cyclone 10 GX variations. The default (reset) value for Intel Arria 10 and Intel Cyclone 10 GX variations is 1.

Table 106. Logical/Transport Layer Error Enable CSR—Offset: 0x10804

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
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 107. 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 108. 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.	

continued...

Field	Bits	Access	Function	Default
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 109. 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

6.2.13. 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.

Table 110. 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 111. 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.	

continued...

Field	Bits	Access	Function	Default
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 112. 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 113. 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 114. 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 115. 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 116. Tx Doorbell Completion—Offset: 0x14

Field[61]	Bits	Access	Function	Default
LARGE_DESTINATION_ID	[31:24]	RO	Reserved if the system does not support 16-bit device ID. MSB of the targeted RapidIO processing element device ID if the system supports 16-bit device ID.	8'h0
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

Table 117. 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 118. 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 119. 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

[61] The completed Tx DOORBELL message comes directly from the Tx Doorbell Completion FIFO.

Table 120. 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

Related Information

Doorbell Module on page 103

7. Testbench

The RapidIO IP core includes a demonstration testbench for your use when you create a simulation model. The purpose of the testbench is to provide examples of how to parameterize the IP core and how to use the Avalon Memory-Mapped (Avalon-MM) and Avalon Streaming (Avalon-ST) interfaces, to generate and process RapidIO transactions.

The demonstration testbench 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
- Reading from the software interface registers
- Transmission and reception of multicast-event control symbols

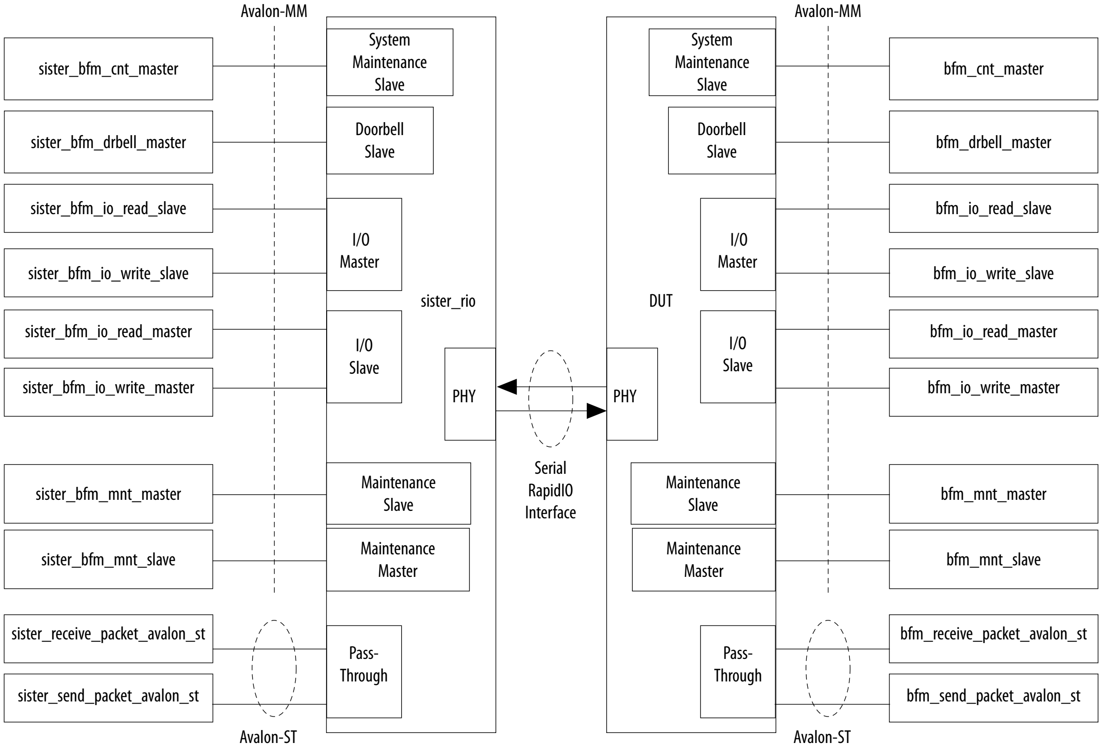
The testbench generates and monitors transactions on the Avalon-MM interfaces and Avalon-ST interface.

The testbench generates `MAINTENANCE`, `Input/Output`, or `DOORBELL` transactions if you select the corresponding modules during parameterization of the IP core. If your IP core variation includes an Avalon-ST pass-through interface, the testbench transfers Type 9 (Data Streaming) packets through that interface.

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.

The following figure is a block diagram of the testbench in which all of the available Avalon-MM interfaces are enabled. The two IP core modules communicate with each other using the RapidIO interface.

Figure 40. RapidIO IP Core Testbench



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)

Note: 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.

The figure illustrates the system specified in Verilog HDL in the testbench connections file (<*design_name*>`_hookup.iv` in variations other than Intel Arria 10 and Intel Cyclone 10 GX, and in the main testbench file <design_name>**_altera_rapidio_<*version*>.tb** in Intel Arria 10 and Intel Cyclone 10 GX variations).

Activity across the Avalon-MM interfaces is generated and checked by running tasks that are defined in the bus functional models (BFMs). In variations other than Intel Arria 10 and Intel Cyclone 10 GX, these models are implemented in the following files:

- <*design_name*>`_avalon_bfm_master.v`
- <*design_name*>`_avalon_bfm_slave.v`

In variations other than Intel Arria 10 and Intel Cyclone 10 GX, the file <*design_name*>`_tb.v` implements the code that performs the test transactions.

In Intel Arria 10 and Intel Cyclone 10 GX variations, the BFMs, the connections, and the test transaction generation are integrated in the main testbench file <design_name>**_altera_rapidio_<*version*>.tb**.

The code that performs the test transactions performs a reset and initialization sequence necessary for the DUT and sister_rio IP cores to establish a link and exchange packets.

7.1. Reset, Initialization, and Configuration

In the testbench generated for variations other than Intel Arria 10 and Intel Cyclone 10 GX, the clocks that drive the testbench are defined and generated in the <*design_name*>`_hookup.iv` file. In the testbench generated for Intel Arria 10 and Intel Cyclone 10 GX variations, the clocks are defined and generated in the main testbench file.

Note: Refer to <*design_name*>`_hookup.iv` or the main testbench file, as appropriate for your IP core variation, 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 121. Testbench Registers Programmed in both the DUT and the sister_rio IP Cores

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
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 this testbench example, the IP cores can exchange basic packets across the serial link.

Related Information

- Software Interface on page 138
  For a full description of each register.
- Testbench on page 167
  For information on RapidIO IP Core Testbench configuration.

7.2. 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.

7.3. 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.

Table 122. 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 these setting, any write operation presented across the Input/Output Avalon-MM slave interface on the rio module is translated to a RapidIO Streaming Write transaction.

Note: 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.

7.4. NWRITE_R Transactions

To perform NWRITE_R operations, one register in the IP core must be reconfigured as shown.

Table 123. 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 these settings, 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.

7.5. NWRITE Transactions

To perform NWRITE operations, one register in the IP core must be reconfigured as shown. With these settings, any write operation presented across the Input/Output Avalon-MM slave interface is translated into a RapidIO NWRITE transaction.

Table 124. NWRITE Transactions

Module	Register Address	Name	Value	Description
rio	0x1040C	Input/Output Slave Mapping Window 0 Control	32'h0055_0000 or 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.

7.6. 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.

7.7. 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.

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. 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. 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.

Related Information

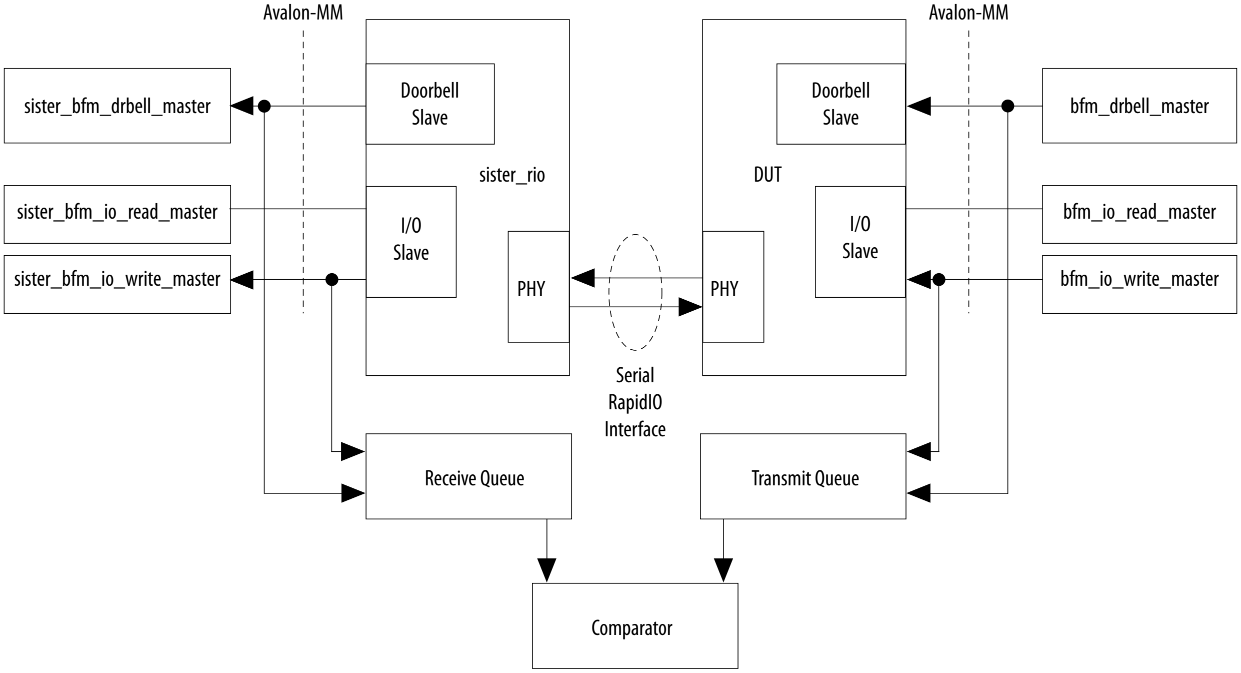
Doorbell Message Registers on page 163

7.8. Doorbell and Write Transactions With Transaction Order Preservation

The following figure 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 in non-

Intel Arria 10 and Intel Cyclone 10 GX variations **Prevent doorbell messages from passing write transactions** is turned on in the RapidIO parameter editor, the extra hardware is generated in the testbench.

Figure 41. 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, and Doorbell messages are sent and checked according to the test sequence. The additional hardware shown in the figure 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. 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 canceled transactions, the testbench checks that the I/O Slave Interrupt register reflects each cancelled transaction correctly.

Related Information

- SWRITE Transactions on page 172
- Doorbell Transactions on page 175

7.9. 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 following operations:

Table 125. 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.

7.10. 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.

8. Platform Designer (Standard) Design Example

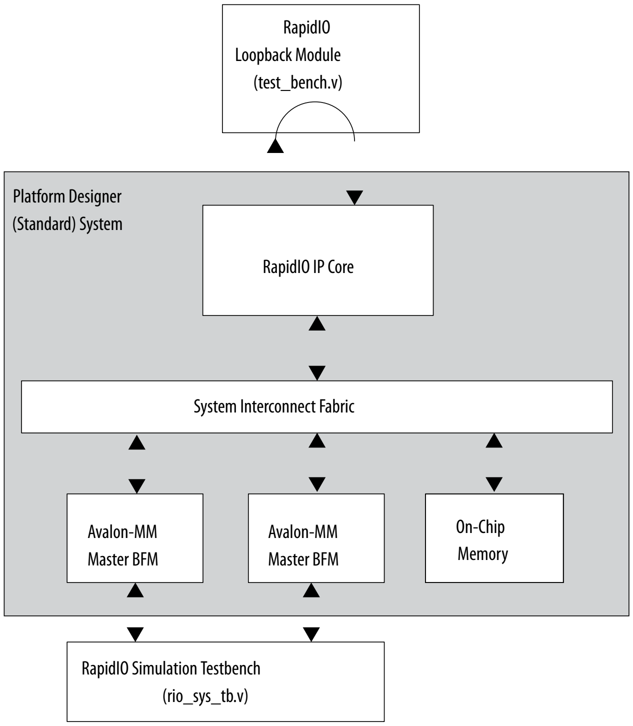
The design example in this chapter shows you how to use Platform Designer (Standard) to build a system that combines a RapidIO IP core with other Platform Designer (Standard) components. Platform Designer (Standard) automatically generates simulation models and HDL files that include all the specified components and interconnections. The design example includes a testbench to simulate the Platform Designer (Standard) system you generate. However, this design example does not support programming your target Intel FPGA and verifying your design in hardware.

The Platform Designer (Standard) design example is available only for variations other than Intel Arria 10 and Intel Cyclone 10 GX.

The design example explains how to use Platform Designer (Standard), an integral feature of the Intel Quartus Prime software, to generate a system containing the following components:

- RapidIO IP core
- On-Chip Memory
- Two Master BFMs

Figure 42. Simulation Example Platform Designer (Standard) System



The Platform Designer (Standard) design example is a simulation example. It does not support programming your target Intel FPGA and verifying your design in hardware.

This design example does not use all available parameters and options.

Related Information

- Platform Designer (Standard) Interconnect
  For more information about the system interconnect fabric.
- Creating a System with Platform Designer (Standard)
  For more information about the Platform Designer (Standard) integration tool.
- Parameter Settings on page 37
  For more information about specific parameters used in this design example.

8.1. Creating a New Intel Quartus Prime Project

You must create a new Intel Quartus Prime 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 **All Programs > Intel FPGA <*version*> Standard Edition** > **Quartus (Intel Quartus Prime <*version*>)** to run the Intel Quartus Prime Standard Edition 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 Intel Quartus Prime 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 Platform Designer (Standard) system.

   Note: The Intel Quartus Prime Standard Edition 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.
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)**

      Note: 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 Intel Quartus Prime project.

8.2. Running Platform Designer (Standard)

This section provides instructions to create and generate your own Platform Designer (Standard) system for the design example. The instructions teach you the process to create a Platform Designer (Standard) system.

If you prefer to run the design example without performing the steps to create your own Platform Designer (Standard) system, you can use the Platform Designer (Standard) system file (`.qsys`) provided in the design example directory.

To run Platform Designer (Standard) to generate your system, whether from your own `.qsys` file or the design example installation `.qsys` file, perform the following steps:

1. In the Intel Quartus Prime software, on the Tools menu, click **Platform Designer (Standard)**.
2. To use the Platform Designer (Standard) system provided with your Intel Quartus Prime installation in <*Quartus Prime_install_dir*>, perform the following steps:
   a. Copy the file <*Quartus Prime_install_dir*>**\ip\altera\rapidio\lib\rio\qsys_cust_demo\rio_sys.qsys** to your Intel Quartus Prime project directory.
   b. In Platform Designer (Standard), on the File menu, click **Open**.
   c. Browse to your Intel Quartus Prime project directory, if necessary, and click **rio_sys.qsys**.
   d. Click **Open**.
   e. Proceed directly to `Generating the System` and follow the instructions.
3. To learn how to create a Platform Designer (Standard) system by generating the design example Platform Designer (Standard) system manually, proceed to “`Adding and Parameterizing the RapidIO Component`” and follow the instructions.

Related Information

- Generating the System on page 189
- Adding and Parameterizing the RapidIO Component on page 182
- Creating a System with Platform Designer (Standard)
  For more information about how to use Platform Designer (Standard).
- Quartus Prime Help

8.2.1. Adding and Parameterizing the RapidIO Component

To instantiate and parameterize the RapidIO IP core in your system, perform the following steps:

1. In the IP Catalog, find and highlight the **RapidIO** IP core component and click **Add**.
2. To parameterize your IP core, follow these steps:
   a. On the **Physical Layer** page, specify the **Device Options** settings.

Table 126. 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.

Table 127. 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.

Table 128. 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.

f. Under **I/O Maintenance Logical Layer Module**, set the parameters.

Table 129. 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.

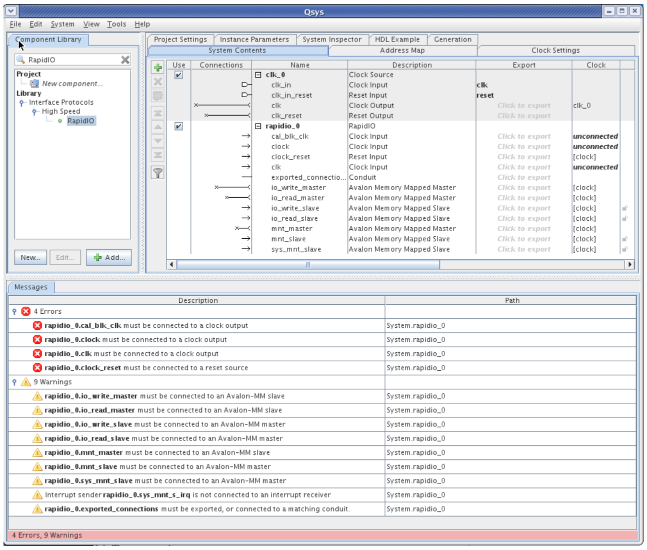
h. Click the **Capability Registers** tab. You can set the Device Register to match your system. Unless your design includes an additional extended feature block, keep the **Extended features pointer** default value of **0x0100**. You can keep the default values for all other parameters.

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 Platform Designer (Standard) 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.

Figure 43. RapidIO IP Core Added and Avalon-MM Ports Created



Use	Connections	Name	Description	Export	Clock
☑		clk_0	Clock Source		
		clk_in	Clock Input	clk	
		clk_in_reset	Reset Input	reset	
		clk	Clock Output	Click to export	clk_0
		clk_reset	Reset Output	Click to export	
☑		rapidio_0	RapidIO		
		cal_blk_clk	Clock Input	Click to export	unconnected
		clock	Clock Input	Click to export	unconnected
		clock_reset	Reset Input	Click to export	[clock]
		clk	Clock Input	Click to export	unconnected
		exported_connectio...	Conduit	Click to export	
		io_write_master	Avalon Memory Mapped Master	Click to export	[clock]
		io_read_master	Avalon Memory Mapped Master	Click to export	[clock]
		io_write_slave	Avalon Memory Mapped Slave	Click to export	[clock]
		io_read_slave	Avalon Memory Mapped Slave	Click to export	[clock]
		mnt_master	Avalon Memory Mapped Master	Click to export	[clock]
		mnt_slave	Avalon Memory Mapped Slave	Click to export	[clock]
		sys_mnt_slave	Avalon Memory Mapped Slave	Click to export	[clock]

Description	Path
4 Errors	
rapidio_0.cal_blk_clk must be connected to a clock output	System.rapidio_0
rapidio_0.clock must be connected to a clock output	System.rapidio_0
rapidio_0.clk must be connected to a clock output	System.rapidio_0
rapidio_0.clock_reset must be connected to a reset source	System.rapidio_0
9 Warnings	
rapidio_0.io_write_master must be connected to an Avalon-MM slave	System.rapidio_0
rapidio_0.io_read_master must be connected to an Avalon-MM slave	System.rapidio_0
rapidio_0.io_write_slave must be connected to an Avalon-MM master	System.rapidio_0
rapidio_0.io_read_slave must be connected to an Avalon-MM master	System.rapidio_0
rapidio_0.mnt_master must be connected to an Avalon-MM slave	System.rapidio_0
rapidio_0.mnt_slave must be connected to an Avalon-MM master	System.rapidio_0
rapidio_0.sys_mnt_slave must be connected to an Avalon-MM master	System.rapidio_0
Interrupt sender **rapidio_0.sys_mnt_s_irq** is not connected to an interrupt receiver	System.rapidio_0
rapidio_0.exported_connections must be exported, or connected to a matching conduit.	System.rapidio_0

4 Errors, 9 Warnings

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.

8.2.2. 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.

8.2.2.1. Adding the Master Maintenance BFM

To add the Master Maintenance BFM to your system, perform the following steps:

1. In the IP Catalog, 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 Platform Designer (Standard) 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.

8.2.2.2. 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.

Table 130. 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 Platform Designer (Standard) 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.

8.2.2.3. 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 Platform Designer (Standard) system.

8.2.3. Connecting Clocks and the System Components

You must 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.

8.2.3.1. 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.

Note: 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.

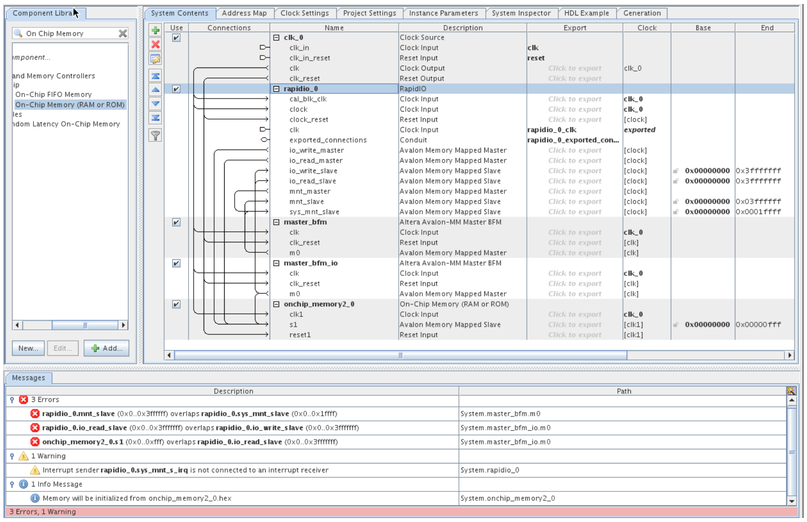
8.2.3.2. Connecting System Components

In Platform Designer (Standard), 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 below.

Table 131. 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

Figure 44. Complete System Connections



Use	Connections	Name	Description	Export	Clock	Base	End
☑		⊟ **clk_0**	Clock Source				
		clk_in	Clock Input	**clk**			
		clk_in_reset	Reset Input	**reset**			
		clk	Clock Output	*Click to export*	clk_0		
		clk_reset	Reset Output	*Click to export*			
☑		⊟ **rapidio_0**	RapidIO				
		cal_blk_clk	Clock Input	*Click to export*	**clk_0**		
		clock	Clock Input	*Click to export*	**clk_0**		
		clock_reset	Reset Input	*Click to export*	[clock]		
		clk	Clock Input	**rapidio_0_clk**	***exported***		
		exported_connections	Conduit	**rapidio_0_exported_con...**			
		io_write_master	Avalon Memory Mapped Master	*Click to export*	[clock]		
		io_read_master	Avalon Memory Mapped Master	*Click to export*	[clock]		
		io_write_slave	Avalon Memory Mapped Slave	*Click to export*	[clock]	**0x00000000**	0x3fffffff
		io_read_slave	Avalon Memory Mapped Slave	*Click to export*	[clock]	**0x00000000**	0x3fffffff
		mnt_master	Avalon Memory Mapped Master	*Click to export*	[clock]		
		mnt_slave	Avalon Memory Mapped Slave	*Click to export*	[clock]	**0x00000000**	0x03ffffff
		sys_mnt_slave	Avalon Memory Mapped Slave	*Click to export*	[clock]	**0x00000000**	0x0001ffff
☑		⊟ **master_bfm**	Altera Avalon-MM Master BFM				
		clk	Clock Input	*Click to export*	**clk_0**		
		clk_reset	Reset Input	*Click to export*	[clk]		
		m0	Avalon Memory Mapped Master	*Click to export*	[clk]		
☑		⊟ **master_bfm_io**	Altera Avalon-MM Master BFM				
		clk	Clock Input	*Click to export*	**clk_0**		
		clk_reset	Reset Input	*Click to export*	[clk]		
		m0	Avalon Memory Mapped Master	*Click to export*	[clk]		
☑		⊟ **onchip_memory2_0**	On-Chip Memory (RAM or ROM)				
		clk1	Clock Input	*Click to export*	**clk_0**		
		s1	Avalon Memory Mapped Slave	*Click to export*	[clk1]	**0x00000000**	0x00000fff
		reset1	Reset Input	*Click to export*	[clk1]		

Description	Path
3 Errors	
rapidio_0.mnt_slave (0x0..0x3ffffff) overlaps **rapidio_0.sys_mnt_slave** (0x0..0x1ffff)	System.master_bfm.m0
rapidio_0.io_read_slave (0x0..0x3fffffff) overlaps **rapidio_0.io_write_slave** (0x0..0x3fffffff)	System.master_bfm_io.m0
onchip_memory2_0.s1 (0x0..0xfff) overlaps **rapidio_0.io_read_slave** (0x0..0x3fffffff)	System.master_bfm_io.m0
1 Warning	
Interrupt sender **rapidio_0.sys_mnt_s_irq** is not connected to an interrupt receiver	System.rapidio_0
1 Info Message	
Memory will be initialized from onchip_memory2_0.hex	System.onchip_memory2_0

3 Errors, 1 Warning

Note: As described in *Reset for RapidIO IP Cores*, the circuitry necessary to ensure the correct behavior of the `reset_n` input signal to the RapidIO IP core is created automatically by Platform Designer (Standard). For this design example, you do not implement the logic described in the above figure , because Platform Designer (Standard) implements it for you.

The remaining errors are resolved as you modify the slave port base addresses, as described in the following section.

Related Information

Reset for RapidIO IP Cores on page 54

8.2.3.3. Assigning Addresses and Setting the Clock Frequency

To assign a specific address, follow these steps:

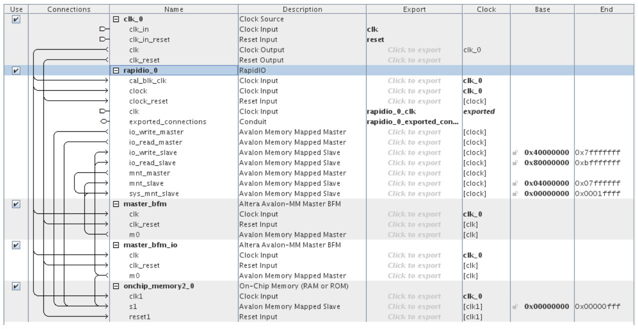
1. Click on the address that you want to change in the **Base** column, then type the address that you want to assign. Make the following address assignments.

Table 132. Assign Addresses

Port Name	Base Address
rapidio_0 io_write_slave	0x40000000
rapidio_0 io_read_slave	0x80000000
rapidio_0 mnt_slave	0x04000000
rapidio_0 sys_mnt_slave	0x00000000
onchip_mem... s1	0x00000000

2. On the File menu, click **Save** and type `rio_sys` to save the Platform Designer (Standard) system in the **rio_sys.qsys** file.

Figure 45. Complete Platform Designer (Standard) Example System



Use	Connections	Name	Description	Export	Clock	Base	End
☑		⊟ **clk_0**	Clock Source				
		clk_in	Clock Input	**clk**			
		clk_in_reset	Reset Input	**reset**			
		clk	Clock Output	*Click to export*	clk_0		
		clk_reset	Reset Output	*Click to export*			
☑		⊟ **rapidio_0**	RapidIO				
		cal_blk_clk	Clock Input	*Click to export*	**clk_0**		
		clock	Clock Input	*Click to export*	**clk_0**		
		clock_reset	Reset Input	*Click to export*	[clock]		
		clk	Clock Input	**rapidio_0_clk**	***exported***		
		exported_connections	Conduit	**rapidio_0_exported_con...**			
		io_write_master	Avalon Memory Mapped Master	*Click to export*	[clock]		
		io_read_master	Avalon Memory Mapped Master	*Click to export*	[clock]		
		io_write_slave	Avalon Memory Mapped Slave	*Click to export*	[clock]	**0x40000000**	0x7fffffff
		io_read_slave	Avalon Memory Mapped Slave	*Click to export*	[clock]	**0x80000000**	0xbfffffff
		mnt_master	Avalon Memory Mapped Master	*Click to export*	[clock]		
		mnt_slave	Avalon Memory Mapped Slave	*Click to export*	[clock]	**0x04000000**	0x07ffffff
		sys_mnt_slave	Avalon Memory Mapped Slave	*Click to export*	[clock]	**0x00000000**	0x0001ffff
☑		⊟ **master_bfm**	Altera Avalon-MM Master BFM				
		clk	Clock Input	*Click to export*	**clk_0**		
		clk_reset	Reset Input	*Click to export*	[clk]		
		m0	Avalon Memory Mapped Master	*Click to export*	[clk]		
☑		⊟ **master_bfm_io**	Altera Avalon-MM Master BFM				
		clk	Clock Input	*Click to export*	**clk_0**		
		clk_reset	Reset Input	*Click to export*	[clk]		
		m0	Avalon Memory Mapped Master	*Click to export*	[clk]		
☑		⊟ **onchip_memory2_0**	On-Chip Memory (RAM or ROM)				
		clk1	Clock Input	*Click to export*	**clk_0**		
		s1	Avalon Memory Mapped Slave	*Click to export*	[clk1]	**0x00000000**	0x00000fff
		reset1	Reset Input	*Click to export*	[clk1]		

8.2.4. 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. Select the **Generate HDL** option.
2. For **Create HDL design files for synthesis**, select **None**. This Platform Designer (Standard) system cannot run on hardware.
3. For **Create simulation model**, select **Verilog**.
4. Turn off **Create block symbol file (.bsf)** to expedite the generation process.
5. Click **Generate** to start the generation process.

   Note: 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 Platform Designer (Standard) system is generated successfully, the system HDL files are added to your project directory and are ready to be simulated with the Intel Quartus Prime software.
6. After generation completes successfully, click **Exit** to close Platform Designer (Standard).

   Note: Although this design example requires the Verilog HDL target output, you can alternatively select VHDL for a project of your own.

8.3. 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 Intel Quartus Prime installation directory to your Intel Quartus Prime 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 Intel Quartus Prime 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.

9. RapidIO Intel FPGA IP User Guide Archives

IP versions are the same as the Intel Quartus Prime Design Suite software versions up to v19.1. From Intel Quartus Prime Design Suite software version 19.2 or later, IP cores have a new IP versioning scheme.

If an IP core version is not listed, the user guide for the previous IP core version applies.

IP Core Version	User Guide
18.0	RapidIO Intel FPGA IP User Guide
17.1	RapidIO IP Core User Guide
17.0	RapidIO IP Core User Guide
14.0	RapidIO MegaCore Function User Guide

10. Document Revision History for the RapidIO Intel FPGA IP User Guide

Document Version	Intel Quartus Prime Version	IP Version	Changes
2021.09.15	20.3	19.2.0	Updated Table: *RapidIO Release Information* for clarity. The RapidIO Intel FPGA IP is no longer supported in Intel Quartus Prime Pro Edition software version 20.3 and later, and Intel Quartus Prime Standard Edition software version 18.1 and later.
2020.09.28	20.3	19.2.0	• Added the *Product Discontinuance Notice*. • Added IP version information.
2018.08.09	18.0	18.0	• Added `txclk` and `rxclk` frequencies into *Clock Frequencies for 4x RapidIO IP Core Variations*. • Added support for Intel Cyclone 10 GX devices. • Added Xcelium simulator support. • Renamed the document as *RapidIO Intel FPGA IP User Guide*. • Added `txclk`, `rxclk` values in *Clock Frequencies for 1x and 2x RapidIO IP Core Variations* table.

Date	Changes
November 2017	• Updated for Intel Quartus Prime 17.1 release. • Updated the resource utilization metrics for all device variations in *Performance and Resource Utilization* section. • Modified steps to generate the Platform Designer (Standard) system in *Generating the System* section. • Updated command in *Simulating the System* section to simulate system with the sample Verilog HDL testbench in ModelSim software.
May 2017	• Changed the document title from "RapidIO MegaCore Function User Guide" to "RapidIO IP Core User Guide". • Changed the document part number from UG-MC_RIOPHY-4.1 to UG-20078 . • Updated device support level for Arria V (GX, GT ,GZ, SX, and ST), Intel Arria 10, Cyclone V (GX, GT, SX, and ST), and Stratix V device family in *Table: Device Family Support*. • Updated the resource utilization metrics for all device variations in *Performance and Resource Utilization*. • Updated *Getting Started* chapter for the Intel Quartus Prime 17.0 software. • Added new section *RapidIO IP Core Testbench Files*. • Made following changes to *Simulating the Testbench with the ModelSim Simulator*: — Updated steps to simulate the testbench. — Updated testbench script location for all device variations. — Updated commands for all device variations. — Corrected TOP_LEVEL_NAME for all device variations. • Updated testbench script location for all device variations in *Simulating the Testbench with the VCS Simulator*. • Added new section *Transceiver PHY Reset Controller for Intel Arria 10 Variations*.

continued...

Date	Changes
	• Made following changes to *External Transceiver PLL*: — Clarified that an fPLL can also implement the required external transceiver PLL in an IP core that targets an Intel Arria 10 device. — Updated the location of the HDL code for an ATX PLL. — Added note to clarify the file name of ATX PLL HDL code for Intel Arria 10 devices. • Updated the value of rxgxbclk for Arria V and Cyclone V devices in *Table: Clock Frequencies for 1x and 2x RapidIO IP Core Variations*. • Clarified the value of default clock frequency for all device variations in *Baud Rates and Clock Frequencies*. • Added information about required parameter value for the Transceiver PHY Reset Controller that connects to the RapidIO IP core in *Reset Requirements for Intel Arria 10 Variations*. • Added new signals: — `input_enable` — `output_enable` — `no_sync_indicator` • Added information in *RapidIO IP Core Clocking* to confirm that RapidIO IP core can handle a difference of ±200 ppm between the rxclk and txclk clocks. • Added new parameter **Packet-Not-Accepted to Link Request timeout** in *Physical Layer Settings*. • Implemented Intel Rebranding.
August 2014	• Added support for Intel Arria 10 devices: New parameter **Enable transceiver dynamic reconfiguration** allows you to hide or make visible the Intel Arria 10 Native PHY IP core dynamic reconfiguration interface, an Avalon-MM interface for programming the hard registers in the Intel Arria 10 transceiver. New requirement to include TX PLL IP core in the Intel Arria 10 design. RapidIO IP core has new individual transceiver channel clock signals `tx_bonded_clocks_chN` to connect to an ATX PLL to support PLL sharing across the transceiver block. New requirement to include a reset controller in the Intel Arria 10 design. RapidIO IP core has new transceiver reset signals `tx_analogreset`, `rx_analogreset`, `tx_digitalreset`, and `rx_digitalreset` to connect to the reset controller. Only certain IP core variations support Intel Arria 10 devices. Refer to *Chapter: Parameter Settings* or *Upgrading a RapidIO Design to the Intel Arria 10 Device Family* for information about the supported and unsupported IP core variations. • Updated migration information for the v14.0 Intel Arria 10 Edition software release in *Appendix: Porting a RapidIO Design from the Previous Version of the Software*.
June 2014	• Removed device support, resource utilization numbers, and speed grade information for the following device families that the Quartus II software v13.1 and later no longer supports: Arria GX, Cyclone II, Stratix II, and Stratix II GX device families. • Removed device support for the following HardCopy device families: HardCopy II, HardCopy III, HardCopy IV E, and HardCopy IV GX device families. This device support was removed in the 13.1 release. • Removed device support, resource utilization numbers, and speed grade information for the following device families that the Quartus II software v14.0 and later no longer supports: Cyclone III, Cyclone III LS, and Stratix III device families. • Renamed and reordered resource utilization tables in *Performance and Resource Utilization* section. • Replaced "Serial RapidIO" with "RapidIO". The RapidIO IP core supports only the Serial RapidIO specification, since before the Quartus II software release 8.0 in 2008. • Modified *Chapter: Getting Started* to describe the new Quartus II software v14.0 IP design flow. • Updated migration information for the v14.0 software release in *Appendix: Porting a RapidIO Design from the Previous Version of the Software*.

continued...

Date	Changes
	• Removed support for Physical-layer only variations. The RapidIO IP core no longer supports Physical-layer only variations. Removed information about clocking, reset, and testbench for Physical-layer only variations. Removed the PHY Maintenance Avalon-MM slave interface signals (phy_mnt_s_clk, phy_mnt_s_chipselect, phy_mnt_s_waitrequest, phy_mnt_s_read, phy_mnt_s_write, phy_mnt_s_addresss, phy_mnt_s_writedata, phy_mnt_s_readdata). This interface is no longer a top-level interface. In variations with a Transport layer, software accesses the Physical layer registers through the system maintenance Avalon-MM slave interface instead. Removed the Calculating Resource Utilization for *Modular Configurations appendix*. • Corrected the description of the `atxwlevel`, `atxovf`, and `arxwlevel` Physical-layer buffer status output signals to indicate they are available. These signals were previously described incorrectly as being available only in Physical-layer-only variations. • Added section *Correcting the Synopsys Design Constraints File to Distinguish RapidIO IP Core Instances*. This section describes an additional requirement for ensuring a design with multiple RapidIO IP core instances functions correctly. • Added new section *Avalon-MM Interface Widths in the RapidIO IP Core*. • Clarified that `sys_mnt_s_address`, `mnt_s_address`, and `drbell_s_address` are word addresses and not byte addresses. They each address a four-byte (32-bit) word. Modified Maintenance TX address window translation calculation accordingly in *Maintenance Module*. • Corrected the width of `mnt_s_address` to 24 bits and in the Maintenance TX address window translation calculation description in *Maintenance Module*. • Clarified that `io_s_wr_address` and `io_s_rd_address` are word addresses (and not byte addresses) in RapidIO 1x variations, and are double-word addresses in 2x and 4x variations. Corrected header and note in to specify that the bit that provides the wdptr information is bit [0] rather than bit [2]. Corrected address window translation calculation and examples accordingly in *Input/Output Avalon-MM Slave Module*. • Updated parameter descriptions in *Chapter: Parameter Settings*. Updated capitalization and minor changes in parameter names. **Transceiver selection** is no longer modifiable. The remaining supported device families do not support an external transceiver option. Removed the **Transceiver Configuration** option. Removed **Enable Transport Layer** parameter. The RapidIO IP core no longer supports Physical-layer only variations: all variations have a Transport layer. Removed EDA Settings and Summary sections. These tabs no longer exist in the RapidIO parameter editor in the new Quartus II software v14.0 IP design flow. • Updated migration information in *Appendix: Porting a RapidIO Design from the Previous Version of the Software* and removed information about porting from SOPC Builder. • Removed dedicated appendix and other information about the XGMII external transceiver interfaces. This interface is not available for any of the current supported device families. • Removed some information about signals exported by Platform Designer (Standard) from *Chapter: Signals*, and redundant information about Platform Designer (Standard) renaming capability. Platform Designer (Standard) is documented in the *Quartus Prime Standard Edition Handbook*. • Corrected direction of `rxgxbclk` output clock signal in *Transceiver Signals*.
May 2013	• Removed SOPC Builder design flow, which is no longer available.
May 2013	• Added 2x mode for variations that target any Arria V, Cyclone V, or Stratix V device, including modification of the descriptions of the PORT_WIDTH and INIT_WIDTH fields in `Port 0 Control` CSR in to include 2x mode options. This feature is available in the Quartus II software 13.0 release and later. • Added support for Arria V GZ, Arria V SX, Arria V ST, Cyclone V SX, and Cyclone V ST devices. This support is available in the Quartus II software 12.1 release and later. • Updated resource utilization information for Arria V, Cyclone V, and Stratix V devices. • Updated Cyclone V GT and Stratix V speed grade support information. • Corrected entries in , *Write Request Size Encoding (32-bit datapath)* and, *Read Request Size Encoding (64-bit datapath)*. • Corrected and enhanced information about `gen_rx_empty` output signal .

continued...

Date	Changes
May 2012	• 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`.
November 2011	• 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: Platform Designer (Standard) Design Example*. • Enhanced description of `arxmty` signal. • Updated simulation sections in *Chapter: Getting Started*.
May 2011	• 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	• Added beta support for Platform Designer (Standard) system integration tool. • Added read-only version of Port 0 Local AckID CSR.
July 2010	• 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	• 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.
March 2009	• 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	• 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.

continued...

Date	Changes
November 2008	• 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	• 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	• 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.

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 Platform Designer (Standard) system integration tool, available with the Intel Quartus Prime software, to rapidly and easily build and evaluate your RapidIO system.

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. Porting a RapidIO Design from the Previous Version of Software

This appendix describes how to port your RapidIO design from the previous version of the RapidIO IP core and Intel Quartus Prime software.

B.1. Upgrading a RapidIO Design Without Changing Device Family

To upgrade your RapidIO design that you developed and generated using the RapidIO IP core v16.1, to the IP core v17.0, perform the following steps:

1. Follow the Intel FPGA IP upgrade instructions.

   Note the new device support restrictions and the fact that the RapidIO IP core no longer supports Physical layer only variations or external transceivers. To upgrade a RapidIO IP core variation that is no longer supported, you must regenerate it with v17.0 supported parameter values.

2. Proceed with simulation, adding the RapidIO timing constraints, and compilation.

Note: Before you add the RapidIO timing constraints, use the Assignment Editor to remove the old 0PPM assignments for this IP core. Otherwise, the new 0 ppm settings are not written.

Related Information

Upgrading IP Cores

B.2. Upgrading a RapidIO Design to the Intel Arria 10 and Intel Cyclone 10 GX Device Families

To upgrade your RapidIO design that you developed and generated to target another device family, to the newly supported Intel Arria 10 and Intel Cyclone 10 GX device families, you must manually re-parameterize and regenerate the RapidIO IP core.

Other IP cores in the design might have a migration path to the Intel Arria 10 and Intel Cyclone 10 GX device families.

The Intel Arria 10 and Intel Cyclone 10 GX device families supports different transceiver connections than previous device families. You will need to modify your design accordingly, including the addition of new supporting IP cores. The value you specify for the new parameter **Enable transceiver dynamic reconfiguration** affects the extent of the new ports.

The Intel Arria 10 and Intel Cyclone 10 GX device families supports fewer distinct variations of the RapidIO IP core than previous device families. If your design includes a RapidIO IP core that does not conform to any of the following restrictions, you must modify the design to accommodate a different RapidIO IP core variation. After you generate the new RapidIO IP core variation, you must connect any resulting new signals and redesign to remove connections to any newly removed signals in your Intel Arria 10 or Intel Cyclone 10 GX designs.

RapidIO IP cores that support an Intel Arria 10 device in the Intel Quartus Prime 14.0 Intel Arria 10 Edition software have the following new restrictions:

- You cannot use external transceivers. You must use the high-speed transceivers on the target device, which are configured with the RapidIO IP core. This change might affect connections.
- You cannot turn off automatic synchronization of transmitted ackID. This change does not affect connections.
- You cannot modify the default number 7 of link-request attempts before declaring a fatal error. This change does not affect connections.
- You cannot modify the default Physical layer receive buffer size of 32 KBytes. This change might affect resource utilization but does not affect connections.
- You cannot modify the default Physical layer transmit buffer size of 32 KBytes. This change might affect resource utilization but does not affect connections.
- You cannot generate a Physical layer only variation. This change might affect connections.
- You cannot turn on destination ID checking by default. However, Intel Arria 10 variations do support dynamic configuration of this feature.
- You cannot support a Maintenance Logical layer master port without supporting a Maintenance Logical layer slave port, and vice versa. This change might affect connections.
- You cannot modify the default number 16 of Maintenance transmit address translation windows. This change might affect resource utilization but does not affect connections.
- You cannot support `MAINTENANCE port-write` request reception without supporting `MAINTENANCE port-write` request transmission, and vice versa. This change does not affect connections.
- You must support order preservation between read and write operations in the I/O Avalon-MM Logical layer slave module. This change might affect resource utilization but does not affect connections.
- You cannot modify the default number 16 of Rx address translation windows. This change might affect resource utilization but does not affect connections.
- You cannot modify the default number 16 of Tx address translation windows. This change might affect resource utilization but does not affect connections.
- You cannot support `DOORBELL` message reception without supporting `DOORBELL` message transmission, and vice versa. This change does not affect connections.
- You must support a `DOORBELL` module Tx staging FIFO to support order preservation between `DOORBELL` messages and I/O write request transactions. This change might affect resource utilization but does not affect connections.

Related Information

- Introduction to Intel FPGA IP Cores
- Getting Started on page 17
- Device Options on page 37

- Transport Layer on page 40
- Input/Output Maintenance Logical Layer Module on page 41
- Port Write on page 42
- I/O Read and Write Order Preservation on page 43
- Avalon-MM Master on page 44
- Avalon-MM Slave on page 44
- Doorbell Slave on page 44