RapidIO Physical Layer

MegaCore Function User Guide

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
www.altera.com

MegaCore Version:	2.2.2
Document Version:	2.2.2 rev. 1
Document Date:	April 2005

UG-MC_RIOPHY-1.6

Contents

About This User Guide ... **vii**
Revision History ... vii
How to Contact Altera ... ix

Chapter 1. About This MegaCore Function
Release Information ... 1–1
Device Family Support ... 1–1
Introduction ... 1–2
New in Version 2.2.2 ... 1–2
Features ... 1–2
General Description ... 1–3
OpenCore Plus Evaluation ... 1–3
Performance ... 1–4

Chapter 2. Getting Started
System Requirements ... 2–1
Design Flow ... 2–1
Obtain & Install the RapidIO Physical Layer MegaCore Function ... 2–2
Download the RapidIO Physical Layer MegaCore Function ... 2–2
Install the RapidIO Physical Layer MegaCore Function Files ... 2–3
Directory Structure ... 2–4
RapidIO Physical Layer MegaCore Function Walkthrough ... 2–4
Create a New Quartus II Project ... 2–5
Launch IP Toolbench ... 2–6
Step 1: Parameterize ... 2–7
Step 2: Set Up Simulation ... 2–12
Step 3: Generate ... 2–14
Simulate the Design ... 2–17
IP Functional Simulation Model ... 2–23
Compile the Design ... 2–24
Program a Device ... 2–25
Set Up Licensing ... 2–25
Append the License to Your license.dat File ... 2–26
Specify the License File in the Quartus II Software ... 2–26

Chapter 3. Serial Specifications
Functional Description ... 3–1
Layer 1 Features ... 3–1
Layer 2 Features ... 3–1
Layer 3 Features ... 3–2

Interfaces & Protocols 3–3
Clock Domains 3–5
Resets 3–7
Baud Rates 3–9
Layer 1 3–10
Layer 2 3–15
Layer 3 3–17
OpenCore Plus Time-Out Behavior 3–22
Parameters 3–23
Signals 3–23
Registers 3–28
Master Register Description 3–29

Chapter 4. Parallel Specifications

Functional Description 4–1
Layer 1 Features 4–1
Layer 2 Features 4–1
Layer 3 Features 4–1
Interfaces & Protocols 4–3
Clock Domains 4–5
Dynamic Phase Alignment 4–7
Layer 1 4–11
Layer 2 4–16
Layer 3 4–19
OpenCore Plus Time-Out Behavior 4–23
Parameters 4–24
Signals 4–25
Registers 4–30
Transmitter Register Description 4–31
Receiver Register Description 4–32
Master Register Description 4–32
MegaCore Verification 4–35

Appendix A. Pin Constraints & Board Design

Pin Constraints A–1
Board Design Configuration A–1

Appendix B. Static & Dynamic Phase Alignment

Static vs Dynamic Alignment B–1
Static Alignment B–1
Dynamic Alignment B–2
Altera Solutions B–3
Static Alignment B–3
Dynamic Phase Alignment (DPA) B–3
AC Timing Analysis B–5

Appendix C. Compliance

Compliance Statement C–1
General Compliance (Physical Layer) C–1
General Compliance (Transport and Logical Layers) C–3

About This User Guide

Revision History

The table below displays the revision history for the chapters in this user guide.

Chapter	Date	Version	Changes Made
1	April 2005	2.2.2	● Updated the release information and device family support tables.
	January 2005	2.2.1	● Updated the release information.
	December 2004	2.2.0	● Updated the release information. ● Updated the features. ● Updated the performance information.
	March 2004	2.1.0	● Updated the release information and device family support tables. ● Moved the Configuration Options table to Chapters 3 and 4. ● Moved Interfaces and Protocols descriptions to Chapters 3 and 4. ● Added OpenCore® Plus description. ● Updated the performance information.
2	April 2005	2.2.2	● Updated the system requirements.
	January 2005	2.2.1	● No change.
	December 2004	2.2.0	● Updated the system requirements. ● Added IP CD installation instructions. ● Updated the walkthrough instructions. ● Added the 4× serial parameter. ● Added the receive priority retry threshold parameters. ● Removed the receive buffer control interface parameter. ● Removed the Set Constraints section.
	March 2004	2.1.0	● Added Linux instructions. ● Moved the configuration parameters description to Chapters 3 and 4. ● Updated the walkthrough instructions. ● Added IP functional simulation models information.

Chapter	Date	Version	Changes Made
3	April 2005	2.2.2	● Updated the Error Handling section. ● Added the Forced Compensation Sequence Insertion section. ● Added more description to the `atxovf` signal.
	January 2005	2.2.1	● No change.
	December 2004	2.2.0	● Updated the Functional Description section, to add the 4× serial feature and description. ● Removed the receive buffer control interface. ● Updated the description of sub-layers 1 and 3. ● Updated the Parameters, Signals, and some Registers tables.
	March 2004	2.1.0	● Added OpenCore Plus time-out behavior description. ● Added Interfaces and Protocols descriptions. ● Added Parameters description (table). ● Updated, renamed, and deleted some signals. ● Updated the register set.
4	April 2005	2.2.2	● Updated the Error Handling section. ● Added more description to the `atxovf` signal.
	January 2005	2.2.1	● No change.
	December 2004	2.2.0	● Updated the features, Figure 4-1, and Figure 4-2. ● Removed the receive buffer control interface. ● Updated the description of sub-layers 2 and 3. ● Updated the Parameters, Signals, and some Registers tables. ● Added the MegaCore Verification section.
	March 2004	2.1.0	● Removed 16-bit port width feature, related description and figures. ● Added OpenCore Plus time-out behavior description. ● Added Interfaces and Protocols descriptions. ● Added Parameters description (table). ● Updated, renamed, and deleted some signals. ● Updated the register set.
A	April 2005	2.2.2	● No change.
	January 2005	2.2.1	● No change.
	December 2004	2.2.0	● Added Stratix® II references.
	March 2004	2.1.0	● Removed all references to APEX™ II device family. ● Updated literature references.
B	April 2005	2.2.2	● No change.
	January 2005	2.2.1	● No change.
	December 2004	2.2.0	● Added Stratix II references. ● Removed Stratix timing information.
	March 2004	2.1.0	● Removed all references to APEX II device family, including static timing information.

Chapter	Date	Version	Changes Made
C	April 2005	2.2.2	● No change.
	January 2005	2.2.1	● No change.
	December 2004	2.2.0	● Removed the packet retry transmission order compliance issue from Table C3. ● Added the port link time-out control issue to Table C2.
	March 2004	2.1.0	● Added this Compliance appendix.

How to Contact Altera

For the most up-to-date information about Altera® products, go to the Altera world-wide web site at www.altera.com. For technical support on this product, go to www.altera.com/mysupport. For additional information about Altera products, consult the sources shown below.

Information Type	USA & Canada	All Other Locations
Technical support	www.altera.com/mysupport/	www.altera.com/mysupport/
	(800) 800-EPLD (3753) (7:00 a.m. to 5:00 p.m. Pacific Time)	+1 408-544-8767 7:00 a.m. to 5:00 p.m. (GMT -8:00) Pacific Time
Product literature	www.altera.com	www.altera.com
Altera literature services	literature@altera.com	literature@altera.com
Non-technical customer service	(800) 767-3753	+ 1 408-544-7000 7:00 a.m. to 5:00 p.m. (GMT -8:00) Pacific Time
FTP site	ftp.altera.com	ftp.altera.com

This document uses the typographic conventions shown below.

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Command names, dialog box titles, checkbox options, and dialog box options are shown in bold, initial capital letters. Example: **Save As** dialog box.
bold type	External timing parameters, directory names, project names, disk drive names, filenames, filename extensions, and software utility names are shown in bold type. Examples: $\mathbf{f_{MAX}}$, **\qdesigns** directory, **d:** drive, **chiptrip.gdf** file.
Italic Type with Initial Capital Letters	Document titles are shown in italic type with initial capital letters. Example: *AN 75: High-Speed Board Design.*
Italic type	Internal timing parameters and variables are shown in italic type. Examples: t_{PIA}, $n + 1$. Variable names are enclosed in angle brackets (< >) and shown in italic type. Example: *<file name>*, *<project name>***.pof** file.

Visual Cue	Meaning
Initial Capital Letters	Keyboard keys and menu names are shown with initial capital letters. Examples: Delete key, the Options menu.
“Subheading Title”	References to sections within a document and titles of on-line help topics are shown in quotation marks. Example: “Typographic Conventions.”
`Courier type`	Signal and port names are shown in lowercase Courier type. Examples: `data1`, `tdi`, `input`. Active-low signals are denoted by suffix `n`, e.g., `resetn`. Anything that must be typed exactly as it appears is shown in Courier type. For example: `c:\qdesigns\tutorial\chiptrip.gdf`. Also, sections of an actual file, such as a Report File, references to parts of files (e.g., the AHDL keyword `SUBDESIGN`), as well as logic function names (e.g., `TRI`) are shown in Courier.
1., 2., 3., and a., b., c., etc.	Numbered steps are used in a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■ ● •	Bullets are used in a list of items when the sequence of the items is not important.
✓	The checkmark indicates a procedure that consists of one step only.
☞	The hand points to information that requires special attention.
⚠ CAUTION	The caution indicates required information that needs special consideration and understanding and should be read prior to starting or continuing with the procedure or process.
⚠	The warning indicates information that should be read prior to starting or continuing the procedure or processes
↵	The angled arrow indicates you should press the Enter key.
👣	The feet direct you to more information on a particular topic.

1. About This MegaCore Function

Release Information

Table 1–1 provides information about this release of the RapidIO Physical Layer MegaCore® function.

Table 1–1. RapidIO Physical Layer Release Information

Item	Description
Version	2.2.2
Release Date	April 2005
Ordering Code	IP-RIOPHY
Product ID	0095
Vendor ID	6AF7

Device Family Support

MegaCore functions provide either full or preliminary support for target Altera® device families, as described below:

- Full support means the MegaCore function meets all functional and timing requirements for the device family and may be used in production designs
- Preliminary support means the MegaCore function meets all functional requirements, but may still be undergoing timing analysis for the device family; it may be used in production designs with caution.

Table 1–2 shows the level of support offered by the RapidIO Physical Layer MegaCore function to each Altera device family.

Table 1–2. Device Family Support

Device Family	Support
Stratix® II	Full
Stratix GX	Preliminary
Stratix	Full
Other device families	No support

Introduction

The RapidIO™ interconnect—an open standard developed by the RapidIO Trade Association—is a high-performance packet-switched interconnect technology designed to pass data and control information between microprocessors, digital signal processors (DSPs), communications and network processors, system memories, and peripheral devices. Its small silicon footprint makes it ideal for field-programmable gate arrays (FPGAs).

New in Version 2.2.2

- Maintenance release

For details, refer to the release notes document available at: **www.altera.com/products/ip/iup/rapidio/m-alt-riophy.html.**

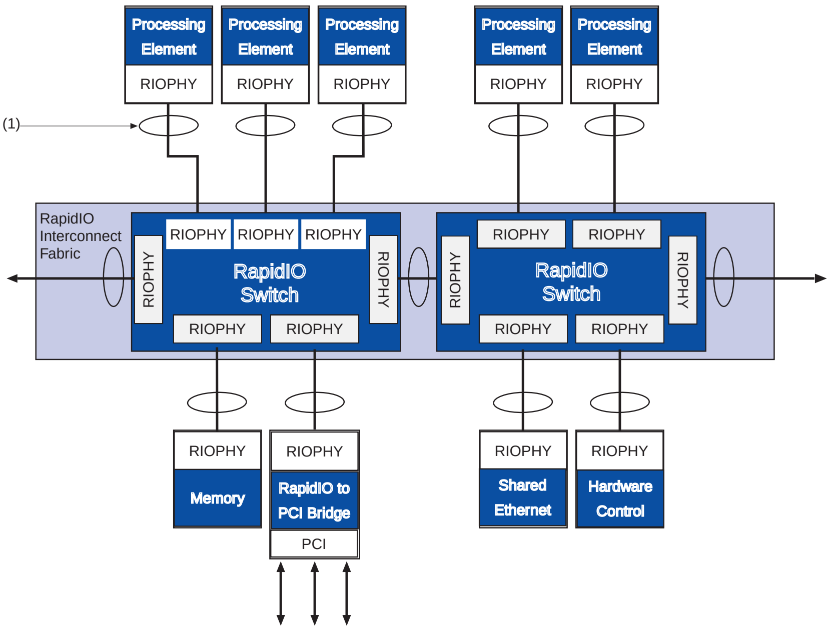
Features

- 1× Serial Features
  - One lane (1×) serial differential signaling
  - 500 Mbaud to 3.125 Gbaud nominal rates
    - 400 Mbps to 2.5 Gbps decoded data rates
- 4× Serial Features
  - Four lane (4×) serial differential signaling
  - 500 Mbaud to 2.5 Gbaud nominal rates
    - 1.6 to 8.0 Gbps aggregate decoded data rates
- Parallel True-LVDS™ high-speed interface ports
  - 8 bits at up to 1 Gbps port data rate
    - 500-MHz DDR clock for up to 8 Gbps throughput rate in each direction
  - Integrated DPA hardware module for use in Stratix GX and Stratix II device families
- 32- and 64-bit Atlantic™ interface
- Asymmetric buffering for receiver and transmitter
- Packet buffering, flow control, error detection, packet assembly and delineation
  - Configurable buffers up to 32 Kbytes
- Compliant with all applicable standards, including:
  - RapidIO Trade Association, *RapidIO™ Interconnect Specification, Revision 1.2,* June 2002.
    - *Part IV: Physical Layer 8/16 LP-LVDS Specification*
    - *Part VI: Physical Layer 1×/4× LP Serial Specification.*
  - RapidIO Trade Association, *RapidIO™ Specification, Revision 1.2,* Errata 1, May 2003.
  - Altera Corporation, *Atlantic Interface Specification*.
  - Altera Corporation, *AIRbus Interface Specification.*
- Easy-to-use IP Toolbench interface
- IP functional simulation models for use in Altera-supported VHDL and Verilog HDL simulators
- Support for OpenCore® Plus evaluation

General Description

The RapidIO Physical Layer MegaCore function is targeted for high-performance, multi-computing, high-bandwidth input/output applications. Figure 1–1 shows an example system implementation.

Figure 1–1. Typical Application



Note to Figure 1–1
(1) All ellipses represent RapidIO interfaces.

OpenCore Plus Evaluation

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

- Simulate the behavior of a MegaCore function within your system using the Quartus® II software and Altera supported VHDL and Verilog HDL simulators
- Verify the functionality of your design, as well as evaluate its size and speed quickly and easily
- Generate time-limited device programming files for designs that include MegaCore functions
- Program a device and verify your design in hardware

You only need to purchase a license for the MegaCore function when you are completely satisfied with its functionality and performance, and want to take your design to production.

For more information on OpenCore Plus hardware evaluation using the RapidIO Physical Layer, see "OpenCore Plus Time-Out Behavior" on page 3–22 (serial) or page 4–23 (parallel), and *AN 320: OpenCore Plus Evaluation of Megafunctions*.

Performance

Table 1–3 shows typical expected performance for the RapidIO Physical Layer MegaCore functions, respectively for Stratix II EP2S30F672C3, and Stratix GX EP1SGX25CF672C5 devices for parallel and EP1SGX40DF1020C5 for serial RapidIO functions. Results were generated using the Quartus II software version 5.0.

This MegaCore function does not use any M-RAM resources.

Table 1–3. RapidIO Utilization & Performance (Part 1 of 2)

Parameters	Stratix GX				Stratix II			
	LEs	Memory		f_{MAX} (MHz)	ALUTs	Memory		f_{MAX} (MHz)
		M512	M4K			M512	M4K	
Serial								
1× Serial, Baud rate=3.125 Gbaud, Atlantic interface port width=32 bits, Tx buffer size=16 Kbytes, Rx buffer size=16 Kbytes	7,135	10	79	100 *(1)(2)*	–	–	–	–
4× Serial, Baud rate=1.25 Gbaud, Atlantic interface port width=32 bits, Tx buffer size=8 Kbytes, Rx buffer size=4 Kbytes	7,846	11	32	125 *(3)*	–	–	–	–
4× Serial, Baud rate=2.5 Gbaud, Atlantic interface port width=64 bits, Tx buffer size=8 Kbytes, Rx buffer size=4 Kbytes	11,561	11	35	125 *(3)*	–	–	–	–

Table 1–3. RapidIO Utilization & Performance (Part 2 of 2)

Parameters	Stratix GX				Stratix II			
	LEs	Memory		f_{MAX} (MHz)	ALUTs	Memory		f_{MAX} (MHz)
		M512	M4K			M512	M4K	
Parallel								
LVDS data rate=1Gbps, DPA=Yes, Atlantic interface port width=64 bits, Tx buffer size=16 Kbytes, Rx buffer size=16 Kbytes	11,920	7	74	125 *(3)*	8,406	7	74	158
LVDS data rate=500 Mbps, Atlantic interface port width=32 bits, Tx buffer size=16 Kbytes, Rx buffer size=16 Kbytes	7,527	7	76	126 *(3)*	5,525	7	76	142

Notes to Table 1–3:
(1) Requires an f_{Max} of 78.125 MHz for 3.125 Gbaud.
(2) Atlantic clocks `arxclk` and `atxclk` can operate up to 125 MHz.
(3) For best results, follow the Quartus II Timing Optimization Advisor recommendations.

2. Getting Started

System Requirements

The instructions in this section require the following hardware and software:

- A PC running the Windows NT/2000/XP, Red Hat Linux 7.3 or 8.0, or Red Hat Enterprise Linux 3.0 operating system; or a Sun workstation running the Solaris 8 or 9 operating system
- Quartus® II software version 5.0 or higher

Design Flow

To evaluate the RapidIO Physical Layer MegaCore function using the OpenCore® Plus feature, the design flow involves the following steps:

1. Obtain and install the RapidIO Physical Layer MegaCore function.

2. Create a custom variation of the RapidIO Physical Layer MegaCore function using IP Toolbench.

   IP Toolbench is a toolbar from which you can quickly and easily view documentation, specify parameters, and generate all of the files necessary for integrating the parameterized MegaCore® function into your design. You can launch IP Toolbench from within the Quartus II software.

3. Implement the rest of your design using the design entry method of your choice.

4. Use the IP Toolbench-generated IP functional simulation model to verify the operation of your design.

For more information on IP functional simulation models, see the *Simulating Altera in Third-Party Simulation Tools* chapter in Volume 3 of the *Quartus II Handbook*.

5. Use the Quartus II software to compile your design and perform static timing analysis.

   You may also generate an OpenCore Plus time-limited programming file, which you can use to verify the operation of your design in hardware.

6. Purchase a license for the RapidIO Physical Layer MegaCore function.

Once you have purchased a license for the RapidIO Physical Layer MegaCore function, the design flow involves the following additional steps:

1. Set up licensing.

2. Generate a programming file for the Altera® device(s) on your board.

3. Program the Altera device(s) with the completed design.

4. Complete system verification.

Obtain & Install the RapidIO Physical Layer MegaCore Function

Before you can start using Altera MegaCore functions, you must obtain the MegaCore files and install them on your computer. Altera MegaCore functions can be installed from the MegaCore IP Library CD-ROM either during or after Quartus II installation, or downloaded individually from the Altera web site and installed separately.

☞ The following instructions describe the process of downloading and installing the RapidIO Physical Layer MegaCore function. If you have already installed the RapidIO Physical Layer MegaCore function from the MegaCore IP Library CD-ROM, skip to "Directory Structure" on page 2–4.

Download the RapidIO Physical Layer MegaCore Function

If you have Internet access, you can download MegaCore functions from Altera's web site at **www.altera.com.** Follow the instructions below to obtain the RapidIO Physical Layer MegaCore function via the Internet. If you do not have Internet access, contact your local Altera representative to obtain the MegaCore IP Library CD-ROM.

1. Point your web browser to **www.altera.com/ipmegastore.**

2. Type `RapidIO` in the **IP MegaSearch** box.

3. Click **Go**.

4. Choose **Parallel & Serial RapidIO Physical Layer** from the search results page. The product description web page displays.

5. Click **Download Free Evaluation** on the top right of the product description web page.

6. Fill out the registration form and click **Submit Request**.

7. Read the Altera MegaCore license agreement, turn on the **I have read the license agreement** check box, and click **Proceed to Download Page**.

8. Follow the instructions on the RapidIO Physical Layer MegaCore function download and installation page to download the MegaCore function and save it to your hard disk.

   ☞ There is a specific MegaCore function download file for each supported operating system.

Install the RapidIO Physical Layer MegaCore Function Files

The following instructions describe how you install the RapidIO Physical Layer MegaCore function on computers running the Windows, Linux, or Solaris operating systems.

Windows

Follow these steps to install the RapidIO Physical Layer MegaCore function on a PC running a supported version of the Windows operating system:

1. Choose **Run** (Windows Start menu).

2. Type <*path name*>`\rio-v2.2.2.exe`, where <*path name*> is the location of the downloaded MegaCore function.

3. Click **OK**. The **RapidIO Physical Layer Installation** dialog box appears. Follow the on-screen instructions to finish installation.

Solaris & Linux

Follow these steps to install the RapidIO Physical Layer MegaCore function on a computer running supported versions of the Solaris and Linux operating systems:

1. Move the compressed files to the desired installation directory and make that directory your current directory.

2. Decompress the package by typing the following command:

```
gzip -d rio-v2.2.2_linux.tar.gz↵
```

or

```
gzip -d rio-v2.2.2_solaris.tar.gz↵
```

3. Extract the package by typing the following command:

```
tar xvf rio-v2.2.2_linux.tar↵
```

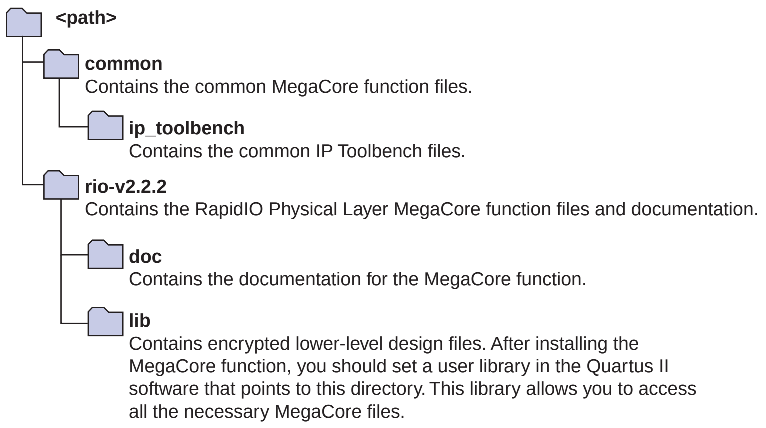
or

```
tar xvf rio-v2.2.2_solaris.tar↵
```

Directory Structure

Figure 2–1 shows the directory structure for the RapidIO Physical Layer MegaCore function, where <*path*> is the installation directory.

Figure 2–1. Directory Structure



<path>
common
Contains the common MegaCore function files.
ip_toolbench
Contains the common IP Toolbench files.
rio-v2.2.2
Contains the RapidIO Physical Layer MegaCore function files and documentation.
doc
Contains the documentation for the MegaCore function.
lib
Contains encrypted lower-level design files. After installing the MegaCore function, you should set a user library in the Quartus II software that points to this directory. This library allows you to access all the necessary MegaCore files.

RapidIO Physical Layer MegaCore Function Walkthrough

This walkthrough explains how to create a RapidIO Physical Layer MegaCore function using the Altera RapidIO Physical Layer IP Toolbench and the Quartus II software on a PC. When you are finished generating a RapidIO Physical Layer MegaCore function, you can incorporate it into your overall project.

This walkthrough involves the following steps:

- “Create a New Quartus II Project” on page 2–5
- “Launch IP Toolbench” on page 2–6
- “Step 1: Parameterize” on page 2–7
- “Step 2: Set Up Simulation” on page 2–12
- “Step 3: Generate” on page 2–14

Create a New Quartus II Project

Before you begin, you must create a new Quartus II project. With the New Project wizard, you specify the working directory for the project, assign the project name, and designate the name of the top-level design entity. You will also specify the RapidIO Physical Layer MegaCore function user library. To create a new project, follow these steps:

1. Choose **Programs > Altera > Quartus II** <*version*> (Windows Start menu) to run the Quartus II software. You can also use the Quartus II Web Edition software.

2. Choose **New Project Wizard** (File menu).

3. Click **Next** in the introduction (the introduction will not display if you turned it off previously).

4. Specify the working directory for your project. This walkthrough uses the directory **c:\temp**.

5. Specify the name of the project. This walkthrough uses **example**.

   You must specify the same name for the project name and the top-level design entity name.

6. Click **Next**.

Steps 7 to 10 are only required if you are running the Solaris or Linux operating system.

7. Click **User Libraries**.

8. Type <*path*>`\rio-v2.2.2\lib\` into the **Library name** box, where <*path*> is the directory in which you installed the RapidIO Physical Layer MegaCore function. The default installation directory is **c:\MegaCore**.

9. Click **Add**.

10. Click **OK**.

11. Click **Next**.

12. Choose the target device family in the **Family** list.

13. Click **Finish**.

You have finished creating your new Quartus II project.

Launch IP Toolbench

To launch IP Toolbench in the Quartus II software, follow these steps:

1. Start the MegaWizard® Plug-In Manager by choosing **MegaWizard Plug-In Manager** (Tools menu). The **MegaWizard Plug-In Manager** dialog box is displayed.

   ☞ Refer to the Quartus II Help for more information on how to use the MegaWizard Plug-In Manager.

2. Specify that you want to create a new custom megafunction variation and click **Next**.

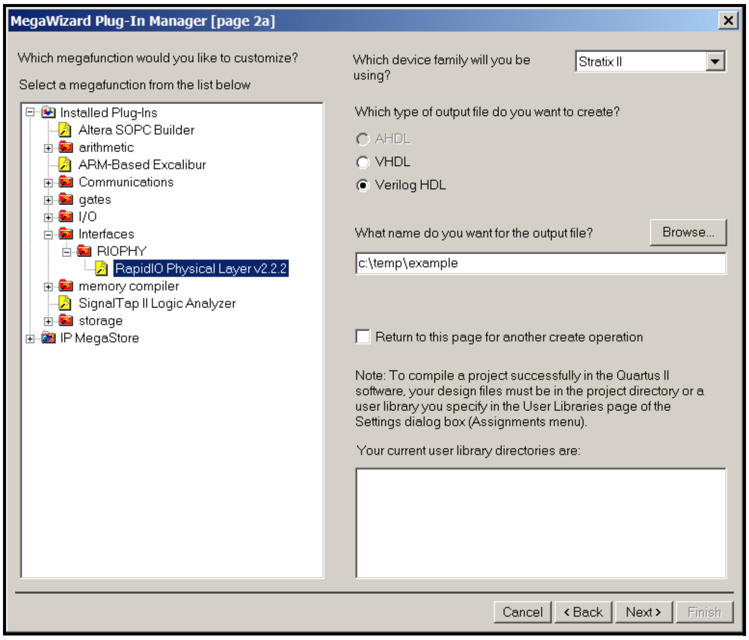
3. Choose **RapidIO Physical Layer v2.2.2** in the **Interfaces > RapidIO** directory.

4. Select the output file type for your design; the wizard supports VHDL and Verilog HDL.

5. Specify a name for the MegaCore function files, <*directory name*>\<*variation name*>. Figure 2–2 on page 2–7 shows the wizard after you have made these settings.

   ☞ The <*variation name*> must be a different name from the project name and the top-level design entity name.

Figure 2–2. Select the MegaCore Function


MegaWizard Plug-In Manager [page 2a]

Which megafunction would you like to customize?

Select a megafunction from the list below

- Installed Plug-Ins
  - Altera SOPC Builder
  - arithmetic
  - ARM-Based Excalibur
  - Communications
  - gates
  - I/O
  - Interfaces
    - RIOPHY
      - RapidIO Physical Layer v2.2.2
  - memory compiler
  - SignalTap II Logic Analyzer
  - storage
- IP MegaStore

Which device family will you be using? Stratix II

Which type of output file do you want to create?

AHDL / VHDL / [✓ Verilog HDL]

What name do you want for the output file? c:\temp\example

Return to this page for another create operation

Note: To compile a project successfully in the Quartus II software, your design files must be in the project directory or a user library you specify in the User Libraries page of the Settings dialog box (Assignments menu).

Your current user library directories are:

6. Click **Next** to launch IP Toolbench.

Step 1: Parameterize

This section describes the parameters available to parameterize a RapidIO Physical Layer MegaCore function, and the benefits of different options. The parameters are ordered as they appear in IP Toolbench.

Not all parameters are supported by, or are relevant for, every MegaCore function variation.

Parallel Walkthrough

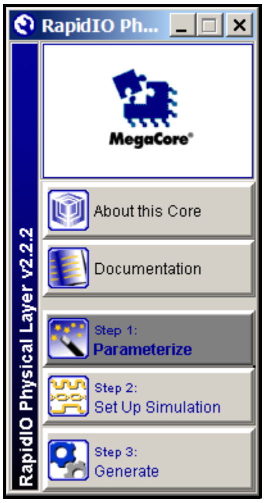
This walkthrough uses the parallel mode. For the serial mode, see page 2–10.

For more information on the parallel parameters, refer to "Parameters" on page 4–24.

To parameterize your MegaCore function, follow these steps:

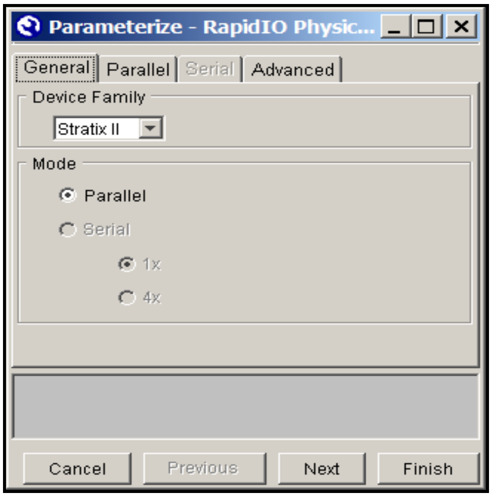
1. Click **Step 1: Parameterize** in IP Toolbench (see Figure 2–3).

Figure 2–3. IP Toolbench



2. Click the **General Parameters** tab (see Figure 2–4).

Figure 2–4. General Parameters



Device Family: Stratix II

Mode
- [✓ Parallel]
- Serial
  - [✓ 1x]
  - 4x

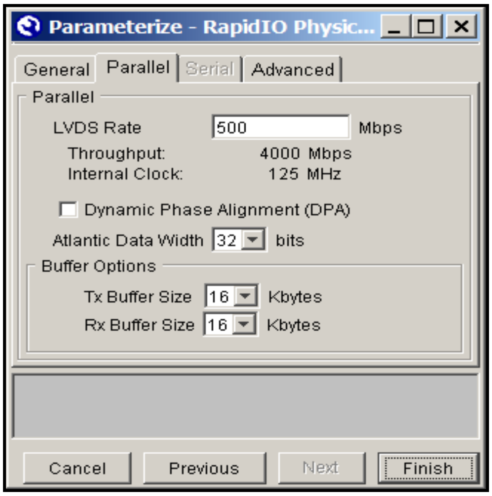
a. Choose the device family.

The RapidIO Physical Layer MegaCore function parallel variations can be targeted to any device in the Stratix® series.

b. Select **Parallel** as the mode of operation.

3. Click the **Parallel** tab (see Figure 2–5).

Figure 2–5. Parallel Parameters



Parallel

LVDS Rate 500 Mbps

Throughput: 4000 Mbps

Internal Clock: 125 MHz

☐ Dynamic Phase Alignment (DPA)

Atlantic Data Width 32 bits

Buffer Options

Tx Buffer Size 16 Kbytes

Rx Buffer Size 16 Kbytes

a. Enter an LVDS rate in Mbps from 300 to 1000.

b. If your chosen LVDS rate is greater than 750 Mbps, and your target device is a Stratix GX or Stratix II FPGA, turn on the **Dynamic Phase Alignment (DPA)** check box.

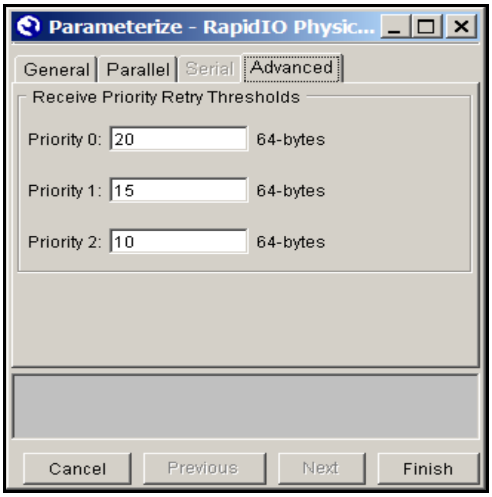
c. Choose an Atlantic™ data width.

d. Choose a transmit and receive buffer size.

4. If you want to adjust the receive retry thresholds, click the **Advanced** tab (see Figure 2–6 on page 2–10). Otherwise, click **Finish** and skip to "Step 2: Set Up Simulation" on page 2–12.

For more information regarding the advanced parallel parameters, see "Receiver" on page 4–19.

Figure 2–6. Advanced Parameters



Receive Priority Retry Thresholds

Priority 0: 20 64-bytes

Priority 1: 15 64-bytes

Priority 2: 10 64-bytes

a. Enter a value for **Receive Priority 0 Retry Threshold**.

b. Enter a value for **Receive Priority 1 Retry Threshold**.

c. Enter a value for **Receive Priority 2 Retry Threshold**.

5. Click **Finish** to complete the parameterization of your parallel MegaCore function variation.

6. Skip to "Step 2: Set Up Simulation" on page 2–12.

Serial Walkthrough

This walkthrough uses the serial mode. For the parallel mode. see page 2–7.

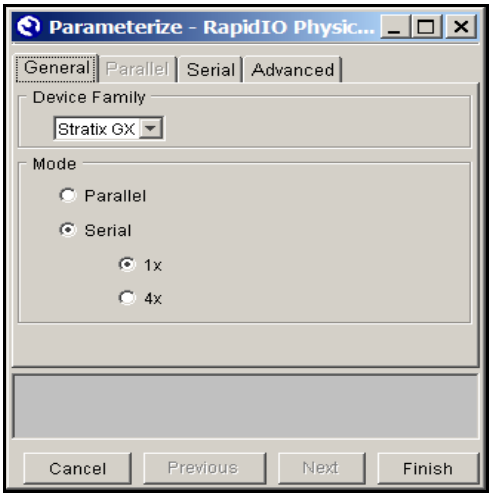
For more information on the serial parameters, refer to "Parameters" on page 3–23.

To parameterize your MegaCore function, follow these steps:

1. Click **Step 1: Parameterize** in IP Toolbench (see Figure 2–3 on page 2–8).

2. Click the **General Parameters** tab (see Figure 2–7 on page 2–11).

Figure 2–7. General Parameters



Device Family: Stratix GX

Mode:
- ○ Parallel
- ● Serial
  - ● 1x
  - ○ 4x

a. Choose the device family.

☞ The RapidIO Physical Layer MegaCore function serial variations can only be targeted to Stratix GX devices.

b. Select the mode of operation as **Serial**.

c. Select the number of lanes to use, either 1 or 4.

3. Click the **Serial** tab (see Figure 2–8 on page 2–12).

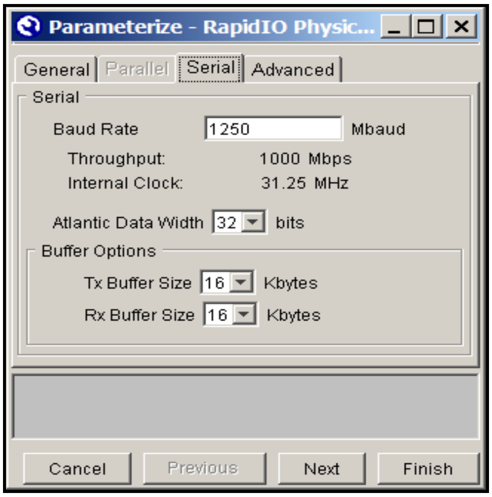
a. Enter a baud rate between 500 and 3,125 Mbaud for 1× variations; or between 500 and 2,500 Mbaud for 4× variations.

☞ Only 1,250, 2,500, and 3,125 Mbaud are defined by the RapidIO specification.

b. Choose an Atlantic data width.

c. Choose a transmit and receive buffer size.

Figure 2–8. Serial Parameters



Serial

Baud Rate 1250 Mbaud

Throughput: 1000 Mbps

Internal Clock: 31.25 MHz

Atlantic Data Width 32 bits

Buffer Options

Tx Buffer Size 16 Kbytes

Rx Buffer Size 16 Kbytes

4. If you want to adjust the receive retry thresholds, click the **Advanced** tab (see Figure 2–6 on page 2–10). Otherwise, click **Finish** and proceed to "Step 2: Set Up Simulation".

For more information regarding the advanced serial parameters, see "Receiver" on page 3–17.

   a. Enter a value for **Receive Priority 0 Retry Threshold**.

   b. Enter a value for **Receive Priority 1 Retry Threshold**.

   c. Enter a value for **Receive Priority 2 Retry Threshold**.

5. Click **Finish** to complete the parameterization of your serial MegaCore function variation.

Step 2: Set Up Simulation

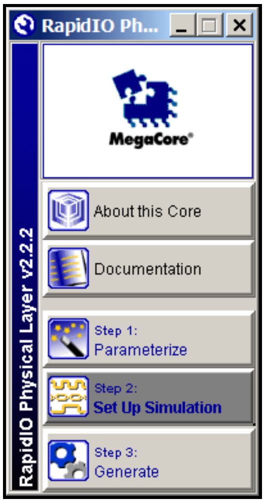
An IP functional simulation model is a cycle-accurate VHDL or Verilog HDL model file produced by the Quartus II software. It allows for fast functional simulation of IP using industry-standard VHDL and Verilog HDL simulators.

☞ You may only use these simulation model output files for simulation purposes and expressly not for synthesis or any other purposes. Using these models for synthesis creates a non-functional design.

To generate an IP functional simulation model for your MegaCore function, follow these steps:

1. Click **Step 2: Set Up Simulation** in IP Toolbench (see Figure 2–9).

Figure 2–9. Set Up Simulation



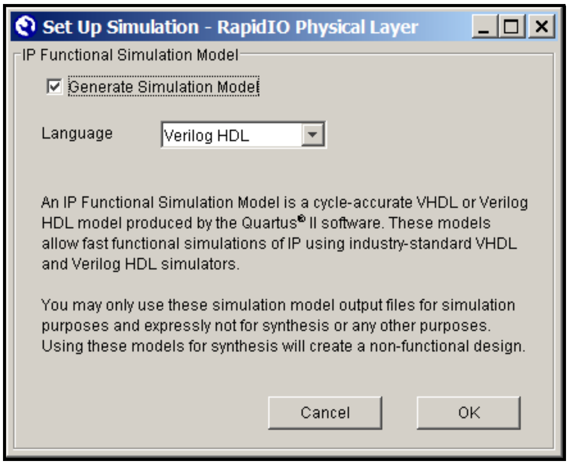
2. Turn on **Generate Simulation Model** (see Figure 2–10 on page 2–14).

3. Choose the language in the **Language** list.

   ☞ To use the IP Toolbench-generated testbench, choose the same language as you chose for your variation.

4. Click **OK**.

Figure 2–10. Generate Simulation Model



Set Up Simulation - RapidIO Physical Layer

IP Functional Simulation Model

☑ Generate Simulation Model

Language: Verilog HDL

An IP Functional Simulation Model is a cycle-accurate VHDL or Verilog HDL model produced by the Quartus® II software. These models allow fast functional simulations of IP using industry-standard VHDL and Verilog HDL simulators.

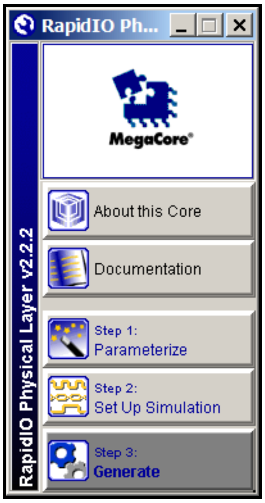
You may only use these simulation model output files for simulation purposes and expressly not for synthesis or any other purposes. Using these models for synthesis will create a non-functional design.

Step 3: Generate

To generate your MegaCore function, follow these steps:

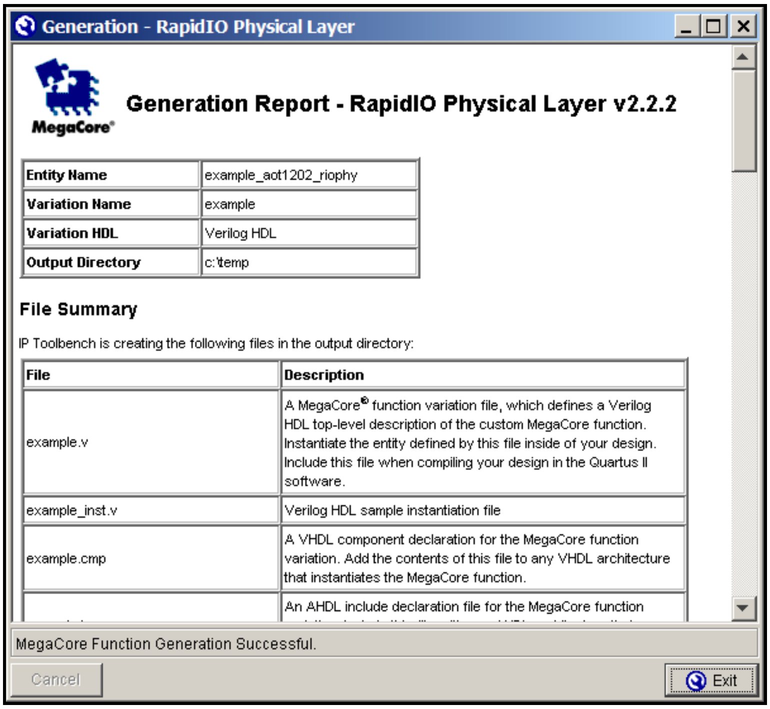
1. Click **Step 3: Generate** in IP Toolbench (see Figure 2–11).

Figure 2–11. IP Toolbench—Generate



2. The generation report lists the design files that IP Toolbench creates (see Figure 2–12). Click **Exit**.

Figure 2–12. Generation



Generation Report - RapidIO Physical Layer v2.2.2

Entity Name	example_aot1202_riophy
Variation Name	example
Variation HDL	Verilog HDL
Output Directory	c:\temp

File Summary

IP Toolbench is creating the following files in the output directory:

File	Description
example.v	A MegaCore® function variation file, which defines a Verilog HDL top-level description of the custom MegaCore function. Instantiate the entity defined by this file inside of your design. Include this file when compiling your design in the Quartus II software.
example_inst.v	Verilog HDL sample instantiation file
example.cmp	A VHDL component declaration for the MegaCore function variation. Add the contents of this file to any VHDL architecture that instantiates the MegaCore function.
	An AHDL include declaration file for the MegaCore function

MegaCore Function Generation Successful.

Table 2–1 describes the IP Toolbench-generated files.

For full details, see the generation report or the **.html** generation report file.

Table 2–1. IP Toolbench-Generated Files (Part 1 of 2) Note (1)

Extension (2)	Description
.**vhd**, or **.v**	A MegaCore variation file, which defines a VHDL or Verilog HDL top-level description of the custom MegaCore function. Instantiate the entity defined by this file inside of your design. Include this file when compiling your design in the Quartus II software.
_inst.vhd or **_inst.v**	VHDL or Verilog HDL sample instantiation file.

Table 2–1. IP Toolbench-Generated Files (Part 2 of 2) Note (1)

Extension (2)	Description
.cmp	A VHDL component declaration for the MegaCore variation. Add the contents of this file to any VHDL architecture that instantiates the MegaCore function.
.inc	An AHDL include declaration file for the MegaCore variation. Include this file with any AHDL architecture that instantiates the MegaCore function.
_bb.v	Verilog HDL black-box file for the MegaCore variation. Use this file when using a third-party EDA tool to synthesize your design.
.bsf	Quartus II symbol file for the MegaCore variation. You can use this file in the Quartus II block diagram editor.
_tb.v	Verilog HDL testbench file.
_simfiles.vnc	Verilog HDL include file list for testbench.
_hutil.iv	Verilog HDL include file for testbench.
_demo_util.iv	Verilog HDL include file for testbench.
_demo_hookup.iv	Verilog HDL include file for testbench.
_run_modelsim_verilog	Script to run testbench.
_run_modelsim_vhdl	Script to run testbench.
riophy_dcore.ocp	OpenCore Plus file.
_aot####_riophy_constraints.tcl	Tool command language (Tcl) script used to set constraints.
_aot####_riophy_altlvds_rx.v	Verilog HDL RTL for MegaCore variation.
_aot####_riophy_altlvds_tx.v	Verilog HDL RTL for MegaCore variation.
_aot####_riophy.v	Verilog HDL RTL for MegaCore variation.
_aot####_riophy_dcore.v	Verilog HDL RTL for MegaCore variation.
.vo or **.vho**	VHDL or Verilog HDL IP functional simulation model.
.html	MegaCore report file.
_pin.html	Variation-specific signals.

Notes to Table 2–1:
(1) These files are variation dependent, some may be absent or their names may change.
(2) The aot#### number represents a unique code assigned to each release of this MegaCore function.

You can now integrate your custom megafunction variation into your design, simulate, and compile.

Constraints are automatically set by the MegaWizard Plug-In Manager.

Simulate the Design

You can simulate your design using the IP Toolbench-generated VHDL and Verilog HDL functional simulation models.

For more information on IP functional simulation models, see the *Simulating Altera in Third-Party Simulation Tools* chapter in Volume 3 of the *Quartus II Handbook*.

Altera also provides a Verilog HDL demonstration testbench, including scripts to compile and run the demonstration testbench using a variety of simulators and models. This testbench demonstrates how to instantiate a model in a design, and includes some simple stimulus to control the user interfaces of the RapidIO interface.

For a complete list of models or libraries required to simulate the RapidIO Physical Layer MegaCore function, refer to the <*variation name*>**_run_modelsim_verilog** or <*variation name*>**_run_modelsim_vhdl** scripts provided with the demonstration testbench.

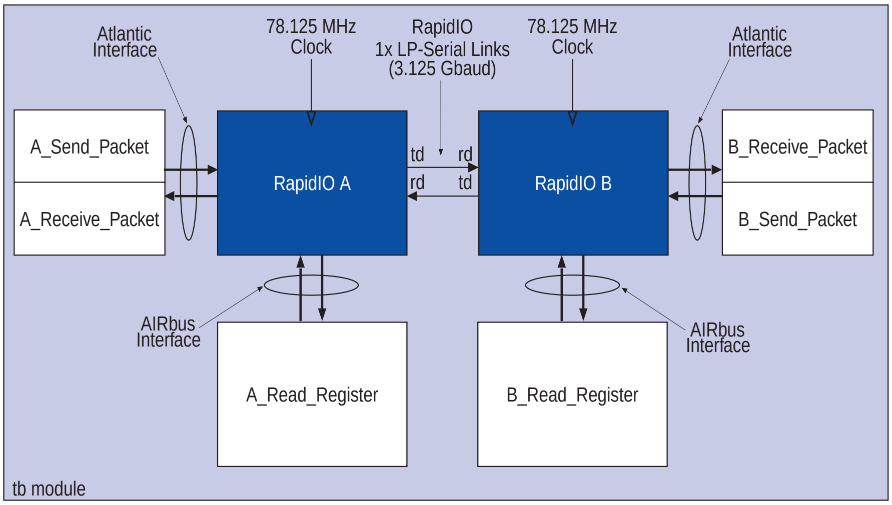
Serial RapidIO Demonstration Testbench Description

The demonstration testbench provided with the serial RapidIO MegaCore function tests the following functions:

- Port initialization process
- Transmission, reception, and acknowledgment of packets with 8 to 256 bytes of data payload
- Writing to and reading from the Atlantic slave interfaces
- Reading from the software interface registers

The testbench consists of two RapidIO MegaCore functions interconnected through their high-speed serial interfaces, see Figure 2–13 on page 2–18. (Each MegaCore function's `td` output is connected to the other MegaCore function's `rd` input.) The testbench module provides clocking and reset control along with tasks to write to and read from the MegaCore function's Atlantic interfaces, and a task to read from the command and status register (CSR) set.

Figure 2–13. Serial RapidIO Demonstration Testbench *Note (1)*



Note to Figure 2–13:
(1) The external blocks, shown in white, are Verilog HDL tasks.

The testbench starts with the MegaCore functions in a reset state. A 78.125 MHz reference clock is provided to all clock inputs. After coming out of reset, the MegaCore functions start the port initialization process to detect the presence of a partner and establish bit synchronization and code group boundary alignment.Once the MegaCore functions have asserted their `port_initialized` output signals, the testbench checks that the port initialization process completed successfully by reading the Error and Status CSR to confirm the expected values of the `PORT_OK` and `PORT_UNINIT` register bits.

Packets with 8 to 256 bytes of data payload are then transmitted from one MegaCore function to the other. The receiving MegaCore function sends the proper acknowledgment symbols and the received packets are checked in the expected sequence for data integrity.

The format of the transmitted packets is described in Table 2–2.

Table 2–2. Serial Packets Format

Packet Byte	Format	Description
First Header word	*{AckID[4:0],Reserved[2:0], prio[1:0],tt[1:0],ftype[3:0]}*	`AckID` is set to zero and is replaced by the transmitting MegaCore function. The `prio` field is used by the receiver to select the output queue. The `tt` and `ftype` fields are for use by the transport and logical layers and are ignored by the physical layer MegaCore functions, except I/O logical maintenance packet type.
DestinationID	*{DestinationID[15:0]}*	These fields are for use by the Transport and Logical layers and are transferred unchanged by the physical layer MegaCore functions.
SourceID	*{SourceID[15:0]}*	
Last Header word	*{Transaction[3:0],Size[3:0],TID[7:0]}*	
Payload bytes	8 to 256 bytes	The payload bytes in the packet are set to an incrementing sequence starting at 0.

The received packets' format is similar, but cyclic redundancy codes (CRCs) and padding (when required) are appended to the packet and an intermediate CRC is inserted in the packets after the first 80 bytes, when the packet's size exceeds 80 bytes.

Table 2–3 lists the tasks used to write packets to a MegaCore function for transmission, read and check a received packet, and read the value from a register and compare it to an expected value.

Table 2–3. Serial Tasks

Function	Prototype	Comments

Write Packet to an Atlantic slave sink.	task send_packet;
	input [1:0] prio;
	input [1:0] tt;
	input [3:0] ftype;
	input [8:0] payload_sizes;
		The `payload_size` should be an even number between 8 and 256 inclusive.
		
		The actual name of the task is pre-pended with A_ or B_ depending on which MegaCore function it should act.
		
		`prio`—packet priority
		`tt`—transport type
		`ftype`—packet format type
		`payload size`—size of the packet payload

Read and check a packet from an Atlantic slave source.	task receive_packet;
	input [1:0] prio;
	input [1:0] tt;
	input [3:0] ftype;
	input [8:0] payload_size;	

Read from Register	task read_register;
	input [15:0]address;
	input [31:0]expected;
		The read value is compared to the expected value, any difference is flagged as an error. "don't care" values can be specified by putting 'x's in the corresponding bit position.

All of the packets are sent contiguously, in sequence. After all packets have been sent, the idle symbols are transmitted until the end of the simulation.

The testbench concludes by checking that all of the packets have been received. If no error is detected and all packets are received, the testbench issues a "`TESTBENCH PASSED`" message stating that the simulation was successful.

If an error is detected, a "`TESTBENCH FAILED`" message is issued to indicate that the testbench has failed. A "`TESTBENCH INCOMPLETE`" message is issued if the expected number of checks is not made. For example, if not all packets are received before the testbench is terminated. The variable tb.exp_chk_cnt determines the number of checks done to insure completeness of the testbench.

To get a value change dump file called **dump.vcd** for all viewable signals, simply uncomment the line "``//`define MAKEDUMP``" in the *<variation name>***_tb.v** file.

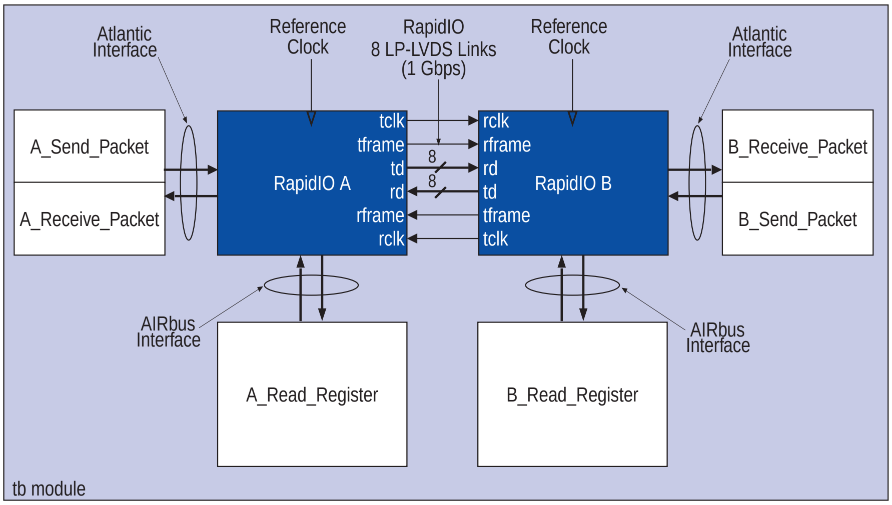
Parallel RapidIO Demonstration Testbench Description

The testbench provided with the parallel RapidIO MegaCore function tests the following functions:

- Port Initialization and training
- Transmission, reception, and acknowledgment of packets with 8 to 256 bytes of data payload
- Writing to and reading from the Atlantic slave interfaces
- Reading from the software interface registers

The testbench consists of two RapidIO MegaCore functions interconnected through their parallel RapidIO interfaces, see Figure 2–14 on page 2–21. (Each MegaCore function's `tclk`, `td`, and `tframe` outputs are connected to the other MegaCore function's `rclk`, `rd`, and `rframe` inputs, respectively.) The tb module provides clocking and reset control along with tasks to write to and read from the MegaCore function's Atlantic interfaces, and tasks to read from the command and status register (CSR) set.

Figure 2–14. Parallel RapidIO Demonstration Testbench Note (1)



Note to Figure 2–14:
(1) The external blocks, shown in white, are Verilog HDL tasks.

The testbench starts with the MegaCore functions in a reset state. A reference clock is provided to all clock inputs. After coming out of reset, the MegaCore functions start the link initialization process to detect the presence of a partner and establish sampling window and 32-bit boundary alignment. Once the MegaCore functions have asserted the `train_done` output signals, the testbench checks that the port initialization process completed successfully by reading the Error and Status CSR to confirm the expected values of the `PORT_OK` and `PORT_UNINIT` register bits.

Packets with 8 to 256 bytes of data payload are then transmitted from one MegaCore function to the other. The receiving MegaCore function sends the proper acknowledgment symbols and the received packets are checked in the expected sequence for data integrity.

The format of the transmitted packets is described in Table 2–4.

Table 2–4. Parallel Packets Format

Packet Byte	Format	Description
First Header word	*{S, AckID[2:0], Reserved1, S_BAR, Reserved2[1:0], prio[1:0], tt[1:0], ftype[3:0]}*	`AckID` is set to zero and is replaced by the transmitting MegaCore function. `S` is set to zero and `S_BAR` is set to one. The `prio` field is used by the receiver to select the output queue. The `tt` and `ftype` fields are for use by the transport and logical layers and are ignored by the physical layer MegaCore functions, except I/O logical maintenance packet type. The `Reserved`, `prio`, `tt`, and `ftype` fields are set to zero in the demonstration testbench.
DestinationID	*{DestinationID[15:0]}*	These fields are for use by the Transport and Logical layers and are transferred unchanged by the physical layer MegaCore functions. They are set to easily recognizable patterns for testing purposes.
SourceID	*{SourceID[15:0]}*	
Last Header word	*{Transaction[3:0],Size[3:0],TID[7:0]}*	
Payload bytes	8 to 256 bytes	The payload bytes in the packet are set to an incrementing sequence starting at 0.

The received packets' format is similar, but CRCs and padding (when required) are appended to the packet and an intermediate CRC is inserted into the packets after the first 80 bytes, when the packet's size exceeds 80 bytes.

Table 2–5 lists the tasks used to write packets to a MegaCore function for transmission, read and check a received packet, and read the value from a register and compare it to an expected value.

Table 2–5. Parallel Tasks

Function	Prototype	Comments

Write Packet to an Atlantic slave sink.	task send_packet;
	input [1:0] prio;
	input [1:0] tt;
	input [3:0] ftype;
	input [8:0] payload_sizes;
		The payload_size should be an even number between 8 and 256 inclusive.
		
		The actual name of the task is pre-pended with A_ or B_ depending on which MegaCore function it should act.
		
		`prio`—packet priority
		`tt`—transport type
		`ftype`—packet format type
		`payload size`–size of the packet payload

Read and check a packet from an Atlantic slave source.	task receive_packet;
	input [1:0] prio;
	input [1:0] tt;
	input [3:0] ftype;
	input [8:0] payload_size;	

Read from Register	task read_register;
	input [15:0]address;
	input [31:0]expected;
		The read value is compared to the expected value, any difference is flagged as an error. “don’t care” values can be specified by putting “x”s in the corresponding bit position.

All of the packets are sent contiguously, in sequence. After all packets have been sent, the idle symbols are transmitted until the end of the simulation.

The testbench concludes by checking that all of the packets have been received. If no error is detected and all packets are received, the testbench issues a “`TESTBENCH PASSED`” message stating that the simulation was successful.

If an error is detected, a “`TESTBENCH FAILED`” message is issued to indicate that the testbench has failed. A “`TESTBENCH INCOMPLETE`” message is issued if the expected number of checks is not made. For example, if not all packets are received before the testbench is terminated. The variable tb.exp_chk_cnt determines the number of checks done to insure completeness of the testbench.

To get a value change dump file called **dump.vcd** for all viewable signals, simply uncomment the line “`` //`define MAKEDUMP ``” in the *<variation name>*__tb.v__ file.

IP Functional Simulation Model

To use the demonstration testbench with IP functional simulation models in the ModelSim® simulator, follow these steps:

For Solaris or Linux operating systems:

1. Turn the *<variation name>*_**run_modelsim_verilog** or *<variation name>*_**run_modelsim_vhdl** scripts into executable files and change the permissions by typing:

   ```
   chmod +x <variation name>_run_modelsim_verilog
   ```

   or

   ```
   chmod +x <variation name>_run_modelsim_vhdl
   ```

2. Run the scripts by typing:

   ```
   ./<variation name>_run_modelsim_verilog
   ```

   or

   ```
   ./<variation name>_run_modelsim_vhdl
   ```

☞ These scripts are only examples. You can modify them to use other simulators.

☞ In all cases, the testbench itself is in Verilog HDL, therefore a license to run mixed language simulations is required to run the testbench with the VHDL model.

In addition to the specified model, the scripts make use of a few clear-text source files: (See Table 2–1 on page 2–15.)

- *<variation name>*_**tb.v** is the top level testbench file
- *<variation name>*_**hutil.iv** defines a few general purpose testing utilities
- *<variation name>*_**demo_hookup.iv** connects the two instantiations of the MegaCore functions together and generates the required clock and reset signals
- *<variation name>*_**demo_util.iv** defines the tasks to read and write on the AIRbus or Atlantic interfaces.

The 220model, `altera_mf`, `altgxb`, and `sgate` simulation libraries required for simulation are also provided.

Compile the Design

You can use the Quartus II software to compile your design. Refer to Quartus II Help for instructions on performing compilation.

Program a Device

After you have compiled your design, program your targeted Altera device, and verify your design in hardware.

With Altera's free OpenCore Plus evaluation feature, you can evaluate the RapidIO Physical Layer MegaCore function before you purchase a license. OpenCore Plus evaluation allows you to generate an IP functional simulation model, and produce a time-limited programming file.

For more information on IP functional simulation models, see the *Simulating Altera in Third-Party Simulation Tools* chapter in Volume 3 of the *Quartus II Handbook*.

You can simulate the RapidIO Physical Layer MegaCore function in your design, and perform a time-limited evaluation of your design in hardware.

For more information on OpenCore Plus hardware evaluation using the RapidIO Physical Layer MegaCore function, see "OpenCore Plus Time-Out Behavior" on page 3–22 (serial) or page 4–23 (parallel), and *AN 320: OpenCore Plus Evaluation of Megafunctions*.

Set Up Licensing

You need to purchase a license for the MegaCore function only when you are completely satisfied with its functionality and performance, and want to take your design to production.

After you purchase a license for RapidIO Physical Layer MegaCore function, you can request a license file from the Altera web site at **www.altera.com/licensing** and install it on your computer. When you request a license file, Altera e-mails you a **license.dat** file. If you do not have Internet access, contact your local Altera representative.

To install your license, you can either append the license to your **license.dat** file or you can specify the MegaCore function's **license.dat** file in the Quartus II software.

Before you set up licensing for the RapidIO Physical Layer MegaCore function, you must already have the Quartus II software installed on your computer with licensing set up.

Append the License to Your license.dat File

To append the license, follow these steps:

1. Close the following software if it is running on your PC:

    - Quartus II software
    - MAX+PLUS® II software
    - LeonardoSpectrum™ synthesis tool
    - Synplify software
    - ModelSim simulator

2. Open the RapidIO Physical Layer MegaCore function license file in a text editor. The file should contain one `FEATURE` line, spanning 2 lines.

3. Open your Quartus II **license.dat** file in a text editor.

4. Copy the `FEATURE` line from the RapidIO Physical Layer MegaCore function license file and paste it into the Quartus II license file.

    ☞ Do not delete any `FEATURE` lines from the Quartus II license file.

5. Save the Quartus II license file.

    ☞ When using editors such as Microsoft Word or Notepad, ensure that the file does not have extra extensions appended to it after you save (e.g., **license.dat.txt** or **license.dat.doc**). Verify the filename in a DOS box or at a command prompt.

Specify the License File in the Quartus II Software

To specify the MegaCore function's license file, follow these steps:

☞ Altera recommends that you give the file a unique name, e.g., <*MegaCore name*>**_license.dat**.

1. Run the Quartus II software.

2. Choose **License Setup** (Tools menu). The **Options** dialog box opens to the **License Setup** page.

3. In the **License file** box, add a semicolon to the end of the existing license path and filename.

4. Type the path and filename of the MegaCore function license file after the semicolon.

    ☞ Do not include any spaces either around the semicolon or in the path/filename.

5. Click **OK** to save your changes.

3. Serial Specifications

Functional Description

This section describes the serial RapidIO MegaCore® function, which is divided into three sub-layers. Layer 1 is the first layer of the three layer partition of the 1× or 4× serial physical layer. It provides a full-duplex interface with serial differential ports to a serial RapidIO™ device or MegaCore function. Layer 1 uses many features provided by the Stratix® GX high-speed transceiver.

Layer 1 Features

- Port initialization
- Receiver
  - One or four lane high-speed data deserialization (up to 3.125 Gbaud for 1× serial with 32-bit Atlantic interface; up to 4× 2.5 Gbaud for 4× serial with 64-bit Atlantic interface)
  - Clock and data recovery
  - Lane synchronization
  - 8B/10B decoding
  - Packet/control symbol delineation
  - Cyclic redundancy code (CRC) checking on packets
  - Control symbol CRC-5 checking
  - Error detection
  - Idle character deletion
- Transmitter
  - One or four lane high-speed data serialization (up to 3.125 Gbaud for 1× serial with 32-bit Atlantic interface; up to 4× 2.5 Gbaud for 4× serial with 64-bit Atlantic interface)
  - 8B/10B encoding
  - Packet/control symbol assembly
  - CRC generation on packets
  - Control symbol CRC-5 generation
  - Pseudo-random idle sequence generation

Layer 2 Features

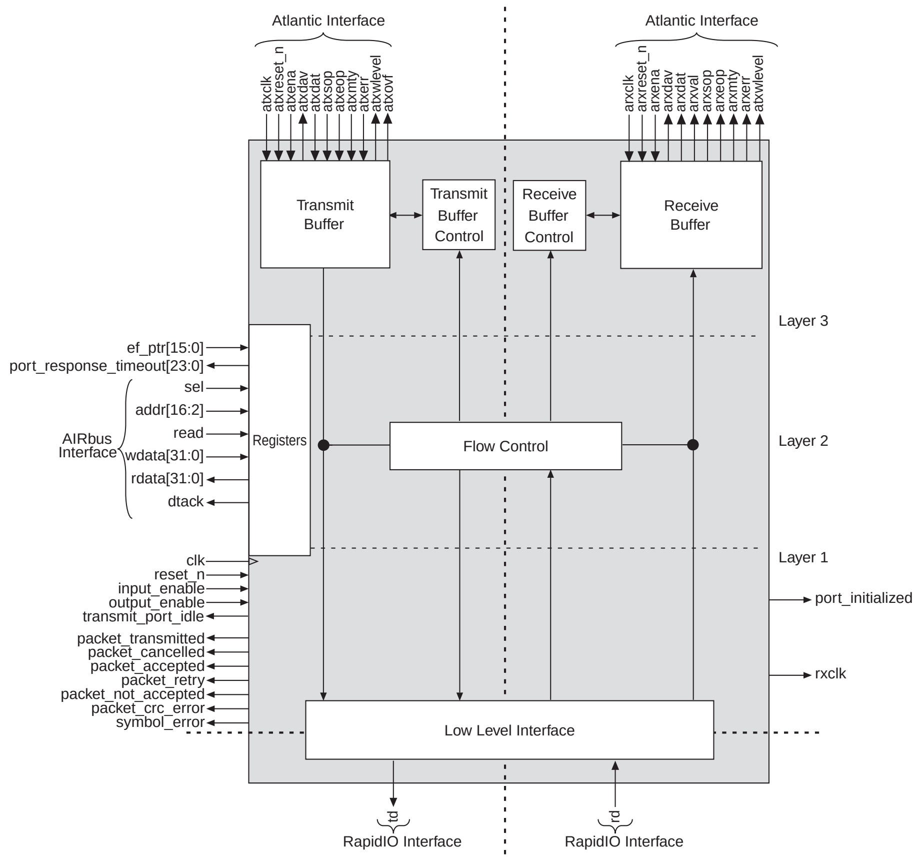
- Processor access (registers)
  - Status/control
- Flow control (`AckID` window tracking)
  - Time-out on acknowledgements
- Order of retransmission maintenance, and acknowledgements
- `AckID` assignment
- Error management

Layer 3 Features

- Atlantic interface with clock decoupling
- First-in first-out (FIFO) buffer level output port
- Asymmetric buffer sizes
- Transmitter
  - Four transmission queues, and four retransmission queues to handle packet prioritization
  - Up to 32-Kbyte buffers
- Receiver
  - Up to 32-Kbytes buffers

Figure 3–1 shows a high-level block diagram of the serial RapidIO MegaCore function.

Figure 3–1. Serial RapidIO Block Diagram



Interfaces & Protocols

Three interfaces support the RapidIO MegaCore function: the RapidIO™ interface, the Atlantic interface, and the access to internal registers (AIRbus) interface.

RapidIO Interface

RapidIO is a packet-switched interconnect protocol defined by the RapidIO Trade Association. The protocol is divided into a three-layer hierarchy: physical layer, transport layer, and logical layer.

The RapidIO Physical Layer MegaCore function implements only the physical layer, which is further divided into three sub-layers: Layer 1, Layer 2, and Layer 3.

Table 3–1 shows the different layers and sub-layers, and their respective functions.

Table 3–1. RapidIO Layers

RapidIO Layer		OSI Layer	Description
Logical		Transport and upper layers	End point operation protocols
Transport		Network layer	Point to point packet delivery addressing scheme
Physical	Layer 3	Physical and Data link layer	Buffering
	Layer 2		Flow control
	Layer 1		Electrical interface, CDR, differential AC coupling, error detection, packet assembling and delineation

More detailed information on the RapidIO interface is available from the RapidIO Trade Association's web site at **www.rapidio.org**.

Atlantic Interface

The Atlantic interface, an Altera® protocol, is the user interface. It also connects the different sub-layers of the RapidIO MegaCore function. For 1× serial variations the Atlantic interface is always 32 bits. For 4× serial variations supporting up to 1.25 GBaud of throughput, the Atlantic interface is 32 bits. For 4× serial variations supporting up to 2.5 GBaud of throughput, the Atlantic interface is 64 bits.

The Atlantic interface is a full-duplex synchronous protocol. The transmit Atlantic interface supports 32- and 64-bit packet data transfers. It works as a slave-sink interface. The receive Atlantic interface supports 32- and 64-bit packet data transfers. It works as a slave-source interface.

The `arxdav` signal is asserted when a full packet is available to be read from the receive buffer.

If the `arxena` signal is asserted when the `arxdav` signal is not asserted, the first word becomes available on the Atlantic interface, and the `arxval` signal is asserted as soon as the first 64-byte block of a packet (or the full packet if it is smaller than 64 bytes) is ready to be read out of the receive buffer. Thus, the MegaCore function does not wait for the full packet before reading out the first block.

Error Handling

The `arxerr` signal can be asserted for a variety of reasons, listed below. As an Atlantic signal, it is synchronous to `arxclk` and is only valid when `arxval` is asserted. Once asserted, `arxerr` stays asserted until the end of the packet when `arxeop` is asserted.

- CRC error—When a CRC error is detected, the `packet_crc_error` signal is asserted for one `rxclk` clock period. The `packet_not_accepted` signal is asserted when the `packet_not_accepted` symbol is transmitted. The `arxerr` signal is also asserted when the packet is read out of the Atlantic interface.

- Stomp—The `arxerr` signal is asserted if a `stomp` control symbol is received in the midst of a packet, causing it to be prematurely terminated. The `arxerr` signal is also asserted for any packet received between the `stomp` symbol and the following `restart-from-retry` symbol.

- Packet size—If a received packet exceeds the allowable size, it is cut short to the maximum allowable size (276 bytes total), and `arxerr` and `arxeop` are asserted on the last word.

- Outgoing symbol buffer full—Under some congestion conditions, there may be no space in the outgoing symbol buffer for the `packet_accepted` symbol. If this happens, the packet cannot be acknowledged and will have to be retried. Thus, `arxerr` is asserted to indicate to the downstream circuit that the received packet should be ignored because it will be retried.

- Symbol error —If an embedded symbol is errored, `arxerr` is asserted and the packet in which it is embedded is retried.

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

AIRbus Interface

The AIRbus interface provides access to internal registers using a simple synchronous internal processor bus protocol. This consists of separate read data (`rdata[31:0]`) and write data (`wdata[31:0]`) buses, a data transfer acknowledge (`dtack`) signal, and a block-select (`sel`) signal. An address (`addr[16:2]`) bus and read (`read`) signal indicate the location and type of access within the block. The `rdata` buses and `dtack` signals can be merged from multiple blocks using a simple `OR` function. The `dtack` signal is sustained until the `sel` signal is removed (four-way handshaking), meaning the AIRbus can cross clock domain boundaries. All registers are 32-bits wide.

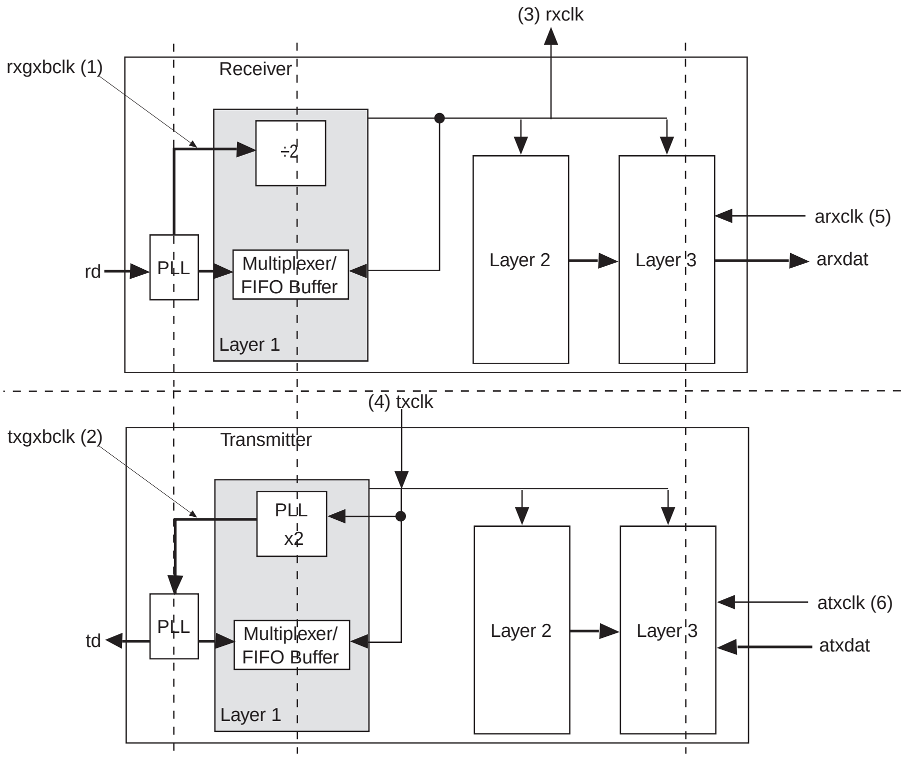
Although the AIRbus interface specification lists a clock (`clk`) and an interrupt request (`irq`) as part of its signals, the RapidIO MegaCore function does not have an AIRbus-specific clock, or an irq signal.

More detailed information on the Atlantic and AIRbus interfaces is available from the Altera web site at **www.altera.com**.

Clock Domains

In addition to the high-speed clock domains inside the Stratix GX transceiver, the 1× serial RapidIO MegaCore function comprises six clock domains: two Stratix GX transceiver clocks, two internal global clocks, and two Atlantic interface clocks, as illustrated in Figure 3–2 on page 3–6.

Figure 3–2. 1× Serial Clock Domains



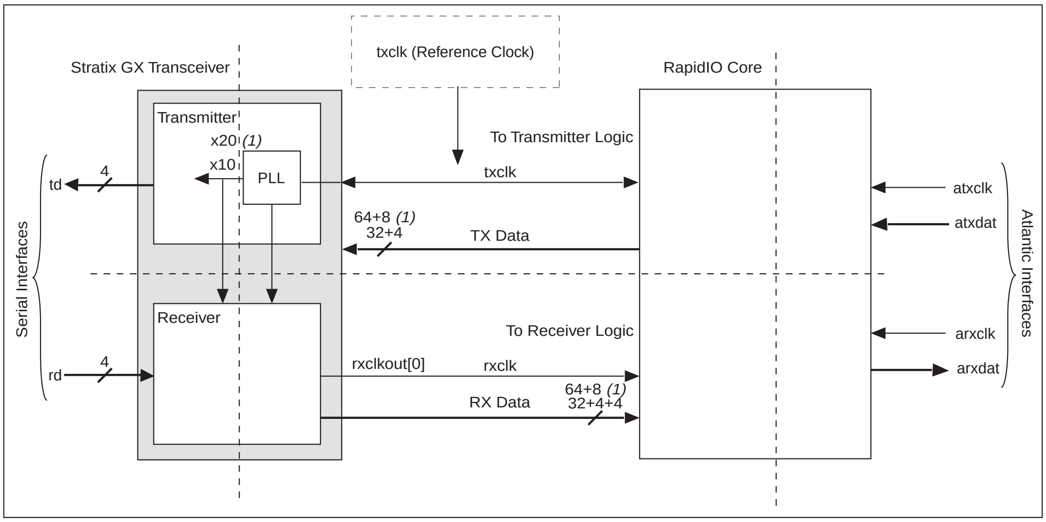
Notes to Figure 3–2:

(1) `rxgxbclk`: Receiver transceiver clock
(2) `txgxblck`: Transmitter transceiver clock
(3) `rxclk`: Receiver internal global clock
(4) `txclk`: Transmitter internal global clock (same as system clock)
(5) `arxclk`: Atlantic interface clock—greater than, or equal to `rxclk`
(6) `atxclk`: Atlantic interface clock—greater than, or equal to `txclk`

The 4× serial RapidIO MegaCore function comprises six clock domains: a main system clock (`txclk`), an internal receiver-side recovered clock (`rxclk`), two Atlantic interface clocks (`atxclk` and `arxclk`), and two high-speed Stratix GX transceiver clocks. Figure 3–3 on page 3–7 shows a top-level view of the clock domains and how they relate to each other.

The main system clock drives the transmit-side logic, and serves as a reference clock for the Stratix GX transceiver's PLL. The PLL generates the high-speed transmit clock and the reference clocks for the receive-side high-speed deserializer clock and recovery unit (CRU). The CRU generates the recovered clock (`rxclk`) that drives the receive-side logic.

Figure 3–3. 4× Serial Clock Domains



Note to Figure 3–3:

(1) For 64-bit Atlantic interface.

Resets

All reset signals can be asserted asynchronously to any clock. However, most reset signals must be deasserted synchronously to a specific clock. The Atlantic interface resets, for example, should be deasserted on the rising edge of the corresponding clock.

See "Signals" on page 3–23 for further details.

The serial RapidIO MegaCore function has a dedicated reset control module called riophy_reset. This module is provided in the **riophy_reset.v** clear-text Verilog HDL source file, and is instantiated inside the top-level riophy module found in the clear text **riophy.v** Verilog HDL source file.

The riophy_reset module controls all of the RapidIO MegaCore function's internal reset signals. In particular, it generates the recommended reset sequence for the `altgxb` high-speed serial I/O megafunction, as described in the *Reset Control & Power Down* chapter of the *Stratix GX Transceiver User Guide*.

The sequence of events when **reset_n** is asserted, and as the MegaCore function comes out of reset after **reset_n** is deasserted, are described in the following steps.

When **reset_n** is asserted:

1. The **txpll_areset** signal is asserted to reset the PLL (for 1× variations only).

2. The internal signals **rxreset_n** and **txreset_n** are asserted to keep the riophy_dcore module in reset until the clocks it relies on are stable.

3. The **gxbpll_areset**, **txdigitalreset**, **rxdigitalreset** and **rxanalogreset** signals are asserted.

When **reset_n** is deasserted:

1. Deassert **txpll_areset** (for 1× variations only).

2. Wait for **txpll_locked** to be asserted, indicates that the PLL's output clock has stabilized (1× variations only).

3. Wait at least 1 millisecond (ms).

4. Deassert **gxbpll_areset.**

5. Wait for **gxbpll_locked** to be asserted.

6. Deassert **txdigitalreset** and **rxanalogreset.**

7. Wait for **rx_freqlocked** to be asserted.

8. Wait for at least 2 ms.

9. Deassert **rxdigitalreset.**

10. Deassert **rxreset_n** and **txreset_n.**

The MegaCore function is now operating normally.

When, as part of its normal operation, the Initialization State Machine (described in paragraph 4.6.3 of *Part VI Physical Layer 1x/4x LP-Serial, Revision 1.2*) transitions to the SILENT state and drives the **link_drvr_oe** signal low, the **txdigitalreset** signal of the `altgxb` megafunction is asserted to simulate turning off the output driver. This causes a steady stream of K28.5 idle characters all of identical disparity to be transmitted.

This, in turn, causes the receiving end to detect several disparity errors and forces the state machine to re-initialize, thus achieving the desired result of the SILENT state.

If two adjacent MegaCore functions are reset one after the other, one of the MegaCore functions may enter the Input Error Stopped state because one of the MegaCore functions is in the SILENT state while the other is already initialized. The initialized MegaCore function is the one to enter the Input Error Stopped state, and subsequently recover.

Baud Rates

The serial RapidIO specification specifies baud rates of 1.25, 2.5, and 3.125 Gbaud. Table 3–2 shows the relationship between baud rates and internal clock rates for 1× serial.

Table 3–2. Baud Rates and Internal Clock Rates for 1× Serial

Baud Rates (Gbaud)	Transceiver Clocks (MHz) (txgxbclk/rxgxbclk)	Internal Clocks (MHz) (txclk/rxclk)
3.125	156.25	78.125
2.5	125.00	62.50
1.25	62.50	31.25

Table 3–3 shows the relationship between baud rates and internal clock rates for 4× serial.

Table 3–3. Baud Rates and Internal Clock Rates for 4× Serial

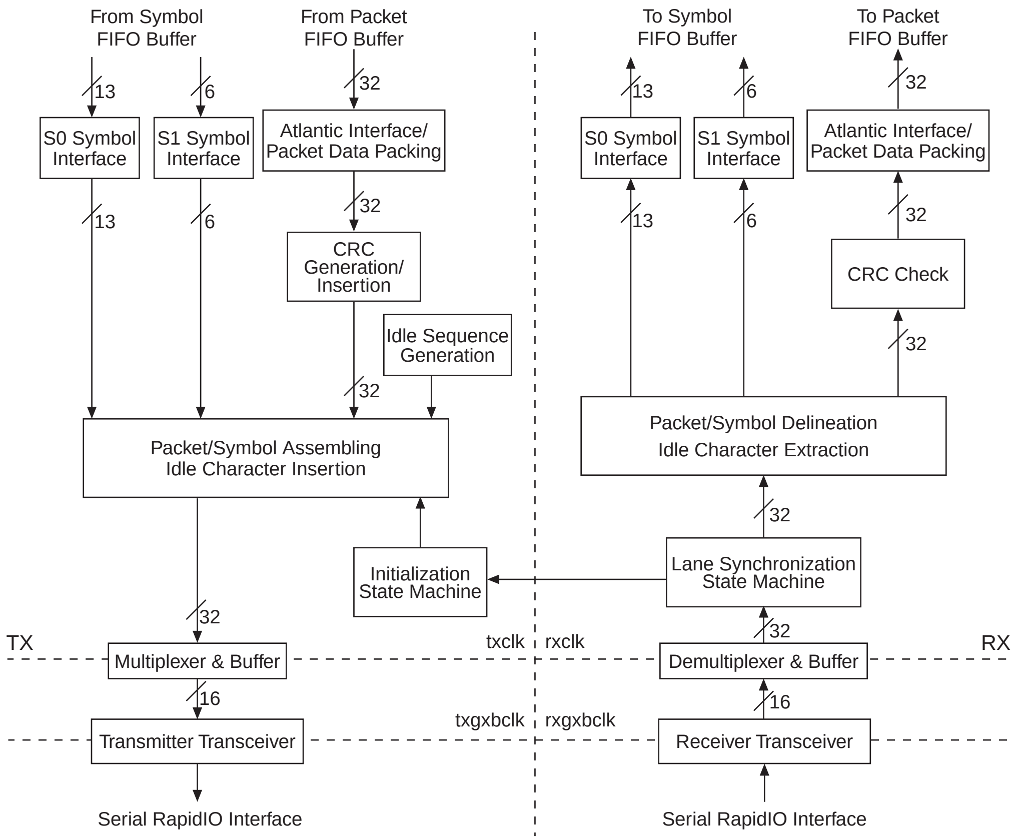
Baud Rates (Gbaud)	Internal Clocks (MHz) (txclk/rxclk)	
	32-Bit Atlantic Interface	64-Bit Atlantic Interface
2.5	–	125
1.25	125	62.5

For more information on using high-speed transceiver blocks, refer to *AN 237: Using High-Speed Transceiver Blocks in Stratix GX Devices.*

Layer 1

The layer 1 sub-layer is designed to be a full-duplex interface with serial differential ports to a serial RapidIO device or MegaCore function. This section gives a block-by-block description of the layer 1 functions. Figure 3–4 on page 3–10 shows a detailed block diagram of the layer 1.

Figure 3–4. Layer 1 Data Flow Block Diagram



Receiver

The layer 1 receiver sub-layer receives and passes packets to the layer 3, and passes control symbols to the layer 2.

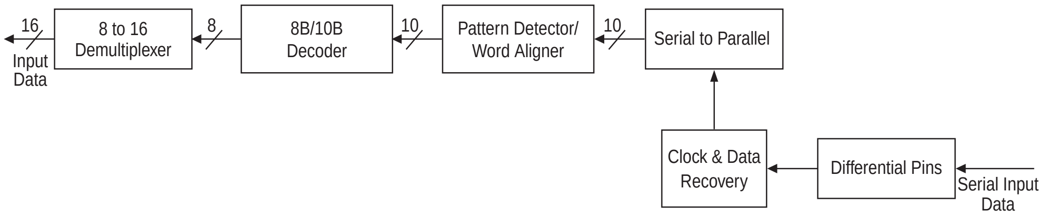
Clock & Data

The layer 1 receiver requires two clock domains: a Stratix GX megafunction clock (`rxgxbclk`), and an internal global clock (`rxclk`).

Receiver Transceiver

The receiver transceiver is an embedded megafunction within the Stratix GX FPGA. Serial data from differential input pins is fed into the clock and recovery unit (CRU) to detect clock and data. Recovered data is deserialized into 10-bit code groups and sent to the pattern detector and word aligner block to detect word boundaries. Properly aligned 10-bit code groups are then 8B/10B decoded into 8-bit characters and converted to 16-bit data via the 8-to-16 demultiplexer. Figure 3–5 shows the structure and the data flow of the receiver transceiver.

Figure 3–5. Receiver Transceiver Structure



Lane Synchronization State Machine

The lane synchronization state machine monitors the lane synchronization status. If the signal `lane_sync` is asserted, then the lane is synchronized and the valid data is presented at the input path.

Packet/Symbol Delineation & Idle Character Extraction

The packet/symbol delineation and idle character extraction block delineates the input data into two data streams. One goes into the packet FIFO buffer, and the other goes into the symbol FIFO buffer.

This block also extracts idle characters from the data stream. It detects stomp symbol and packet size error, and asserts the corresponding error signals to layer 2. This block checks the 5-bit CRC at the end of the 24-bit symbol that covers the first 19 bits. The polynomial $x^5+x^4+ x^2+ 1$ is used. If the CRC is incorrect, the error signal `sym_err` is asserted.

CRC Check

The CCITT polynomial $x^{16} + x^{12} + x^5 + 1$ is used for CRC checking. This block checks 16-bit CRCs that cover all packet header bits (except the first six bits), and all data payload. 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 CRC is used and appended at the end.

For packets longer than 80 bytes, two CRCs are used. The first CRC is appended after the first 80 bytes; the second CRC is a continuation of the calculation of the first CRC and is appended at the end of the packet.

This block also flags CRC errors, and packet size errors.

Atlantic Interface/Packet Data Packing

This block sends 32- or 64-bit data to the upper layer via a 32- or 64-bit master-source Atlantic interface. It generates all required handshake signals for the interface.

S0 & S1 Symbol Interface

These blocks receive 13-bit stype0 control symbols and 6-bit stype1 control symbols, respectively. These blocks send control symbols to the upper layer via a simple dual-port FIFO interface.

Transmitter

The layer 1 transmitter sub-layer assembles packets and control symbols, received over a slave-source Atlantic interface, into one message and passes it to the serial RapidIO interface.

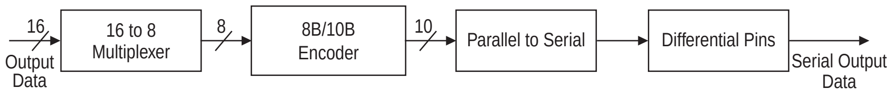
Clock and Data

The layer 1 transmitter uses two clocks: a Stratix GX megafunction clock (`txgxbclk`), and an internal global clock (`txclk`).

Transmitter Transceiver

The transmitter transceiver is an embedded megafunction within the Stratix GX FPGA. The 16-bit parallel output data is internally multiplexed to 8-bit data and 8B/10B encoded. The 10-bit encoded data is then serialized and sent to differential output pins. Figure 3–6 on page 3–13 shows the transmitter transceiver structure and data flow direction.

Figure 3–6. Transmitter Transceiver Structure



Initialization State Machine
The serial port must be initialized before it can receive valid data. This state machine works closely with the lane synchronization state machine to monitor the `lane_sync` signal. When the `lane_sync` signal is asserted, the state machine enters the 1×_MODE or 4×_MODE state. The `port_initialized` signal is also asserted.

Packet/Symbol Assembling & Idle Character Insertion
The packet/symbol assembling and idle character insertion block assembles packet data and control symbol into a proper output format, with corresponding delimiting symbols and special characters. It generates 5 bit CRCs to cover the 19-bit symbol and appends the CRC at the end of the symbol. The polynomial $x^5+x^4+ x^2+ 1$ is used. It inserts an idle sequence if both the packet FIFO and symbol FIFO buffers are empty. During port initialization, it continues to send idle characters until the port is initialized. This module is also responsible for inserting status control symbols at least once every 1,024 transmitted code groups.

Idle Sequence Generation
When there is no data to transmit, layer 1 automatically inserts idle characters to transmit. As stated on page 43 of *Part VI: Physical Layer 1×/4× LP Serial Specification* of the *RapidIO™ Interconnect Specification, Revision 1.2, June 2002.*

"The 1x idle sequence consists of a sequence of the code-groups /K/, /A/, and /R/ (the idle code-groups) and shall be used by ports in operating is 1x mode. The 4x idle sequence consists of a sequence of the columns ||K||, ||A||, ||R|| (the idle columns) and shall be used by ports operating in 4x mode. Both sequences shall comply with the following requirements:

> *1. The first code-group (column) of an idle sequence generated by a port operating in 1x mode (4x mode) shall be a /K/ (||K||). The first code-group (column) shall be transmitted immediately following the last code-group (column) of a packet or delimited control symbol.*

2. At least once every 5000 code-groups (columns) transmitted by a port operating in 1x mode (4x mode), an idle sequence containing the /K/R/R/R/ code-group sequence (||K||R||R||R|| column sequence) shall be transmitted by the port. This sequence is referred to as the "compensation sequence".

3. When not transmitting the compensation sequence, all code-groups (columns) following the first code-group (column) of an idle sequence generated by a port operating in 1x mode (4x mode) shall be a pseudo-randomly selected sequence of /A/, /K/, and /R/ (||A||, ||K||, and ||R||) based on a pseudo-random sequence generator of 7th order or greater and subject to the minimum and maximum /A/ (||A||) spacing requirements.

4. The number of non /A/ code-groups (non ||A|| columns) between /A/ code-groups (||A|| columns) in the idle sequence of a port operating in 1x mode (4x mode) shall be no less than 16 and no more than 32. The number shall be pseudo-randomly selected, uniformly distributed across the range and based on a pseudo-random sequence generator of 7th order or greater."

CRC Generation & Insertion

The CCITT polynomial $x^{16} + x^{12} + x^5 + 1$ is used for CRC generation. This block generates a CRC that covers all packet header bits, (except the first six bits) and all data payload. 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 CRC is used and appended at the end.

For packets longer than 80 bytes, two CRCs are used. The first CRC is appended after the first 80 bytes; the second CRC is a continuation of the calculation of the first CRC and is appended at the end of the packet.

Atlantic Interface/Packet Data Packing

The transmitter receives packet data from upper layers via a 32- or 64-bit master-sink Atlantic interface. It generates all required handshake signals for the interface.

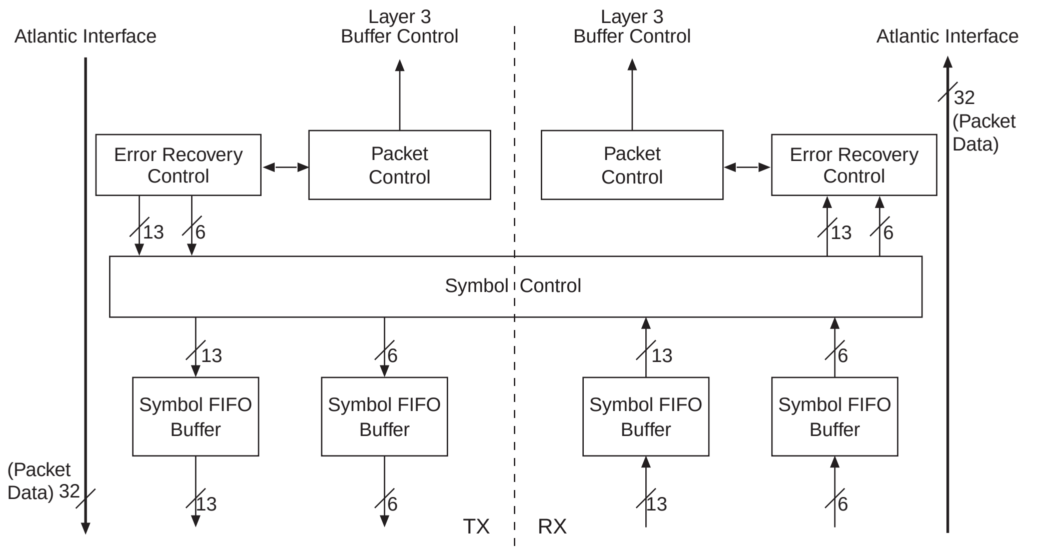
S0 & S1 Symbol Interface

The transmitter receives control symbols from upper layers via 13-bit and 6-bit FIFO interfaces. The 13-bit interface is for stype0 control symbols, and the 6-bit interface is for stype1 control symbols. It also decodes packet termination symbols: stomp, restart from retry, and link request.

Layer 2

The layer 2 sub-layer provides flow control for the serial RapidIO physical layer. This section gives a block-by-block description of the layer 2. Figure 3–7 on page 3–15 shows a detailed block diagram of the layer 2.

Figure 3–7. Layer 2 Data Flow Block Diagram



Receiver

The layer 2 receiver sub-layer is responsible for processing incoming control symbols. It also monitors incoming packet `ackID`s to maintain proper flow.

Clock and Data

The layer 2 receiver comprises one clock domain: an internal global clock (`rxclk`).

Symbol FIFO Buffer

Incoming 13-bit stype0 control symbols and 6-bit stype1 control symbols are stored in their respective symbol FIFO buffers by the layer 1. These symbols are retrieved by the layer 2 for further processing.

Symbol Control

On the receive side, the layer 2 keeps track of the sequence of `ackID`s and tells the layer 3 which packets have been acknowledged, and which packets to drop.

Packet Control

The packet control block uses a sliding window protocol to handle incoming and outgoing packets. Each incoming and outgoing packet has an attached 5-bit `ackID` in the header field. The value of `ackID` is zero at reset. It increments after each packet is sent out, and rolls over to zero after it has reached 31. All packets can only be accepted by the receiver in the sequential order specified by the `ackID`. If a packet is lost at the receiver, a packet retry request with the lost `ackID` is sent to the sender. The sender then retransmits all packets starting from the lost `ackID`.

Error Recovery Control

A packet or control symbol corrupted by an incorrect CRC, or by a CRC-5 error, must be recovered. During the error recovery process, two interdependent state machines are required to operate the input and output ports, respectively.

When an incoming packet is corrupted, the receiver sends a `packet not accepted` symbol to the sender. The sender then retransmits all packets starting from the retried `ackID` of the corrupted packet.

When an incoming control symbol is corrupted, the receiver sends a `packet not accepted` control symbol to inform the sender of the internal status, and the expected `ackID`. The sender then proceeds to retransmit the control symbol.

Transmitter

The layer 2 transmitter sub-layer is responsible for creating and transmitting outgoing control symbols. It also monitors outgoing packet `ackID`s to maintain proper flow.

Clock and Data

The layer 2 transmitter comprises one clock domain: an internal global clock (`txclk`).

Symbol FIFO Buffer

The layer 2 provides these symbol FIFO buffers to store outgoing 13-bit stype0 control symbols and 6-bit stype1 control symbols. These symbols are retrieved by the layer 1, and sent out via the serial RapidIO interface.

Symbol Control

On the transmit side, the layer 2 keeps track of the sequence of `ackID`s and tells the layer 3 which packet to send with what `ackID`. The layer 2 also tells the layer 3 which packet has been acknowledged, and thus can be discarded in the buffers.

Packet Control

The packet control block uses a sliding window mechanism to handle incoming and outgoing packets. This block also sets the time-out counters for each outgoing packet. When time-out occurs to an outgoing packet, the packet control block treats it as an unexpected acknowledge control symbol, and starts the packet retry process.

Error Recovery Control

An uncorrupted protocol violating control symbol, or a control symbol corrupted by an incorrect CRC, or by a CRC-5 error must be recovered. During the error recovery process, two interdependent state machines are required to operate the input and output ports, respectively.

For error recovery, transmitted packets are held by the output port for possible retransmission in case an error is detected by the receiving device. The packets are held until the sending device receives a packet-accepted control symbol for that packet. If a packet is retransmitted, the time-out counter is reset for that retransmitted packet.

Layer 3

The layer 3 sub-layer provides buffers, and buffer management for packet data. This section briefly describes the layer 3 functions.

Receiver

The layer 3 receiver sub-layer accepts packet data from the layer 1 sub-layer, and stores it in its buffers for the user. The receiver buffer is partitioned in 64-byte blocks that are allocated from a free queue as required, and returned to the free queue when no longer needed. Up to five 64-byte blocks can be used to store a packet.

The RapidIO Specification requires that at least one packet, of higher priority than all previously transmitted packets, always be able to pass through.

To meet this requirement, the layer 3 receiver sub-layer accepts or retries received packets based on their priority and the receive buffer's fill level.

The receiver bases its decision to accept or retry packets on three programmable threshold levels: Threshold_2, Threshold_1, and Threshold_0.

- Packets of priority 2′b11 (highest priority) are retried only if the receiver buffer is full.
- Packets of priority 2′b10 are retried only if the number of available free 64-byte blocks is less than Threshold_2.
- Packets of priority 2′b01 are retried only if the number of available free 64-byte blocks is less than Threshold_1.
- Packets of priority 2′b00 (lowest priority) are retried only if the number of available free 64-byte blocks is less than Threshold_0.

The default threshold values are:

- Threshold_2 = 10
- Threshold_1 = 15
- Threshold_0 = 20

The threshold values are programmed through clear-text internal input ports in the riophy module, and can be set on the **Advanced** tab of IP Toolbench.

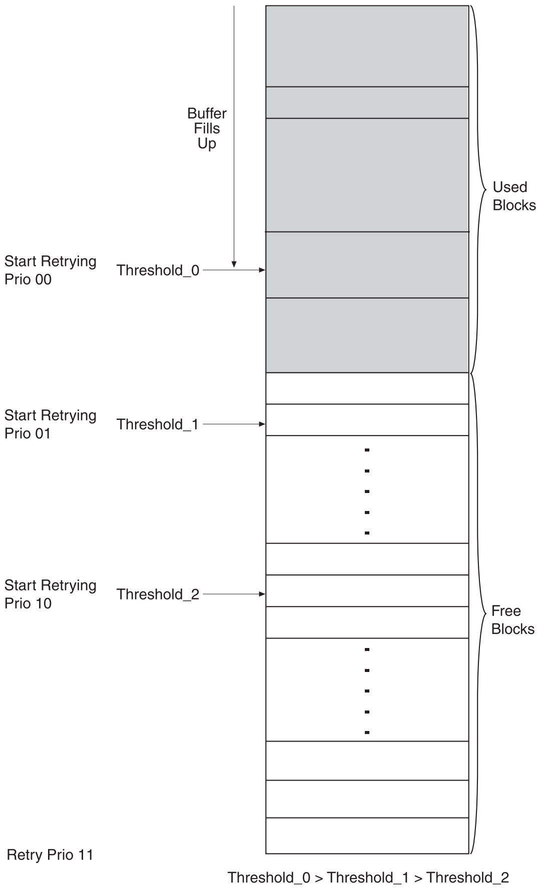
- `rx_threshold_0[p_rxbuf_addr_width:0]`
- `rx_threshold_1[p_rxbuf_addr_width:0]`
- `rx_threshold_2[p_rxbuf_addr_width:0]`

Where `p_rxbuf_addr_width` corresponds to the internal effective address width.

To comply with the RapidIO specification, the threshold values must increase monotonically by at least the size of one packet (see Figure 3–8 on page 3–19). The MegaWizard Plug-In generates the following parameters, and enforces the consistency checks.

- p_rx_threshold_2 > 9
- p_rx_threshold_1 > p_rx_threshold_2 + 4
- p_rx_threshold_0 > p_rx_threshold_1 + 4
- p_rx_threshold_0 < (2 ** p_rxbuf_addr_width)

Figure 3–8. Receiver Threshold Levels



Clock & Data

The layer 3 receiver sub-layer comprises two clock domains: an internal global clock (`rxclk`), and an Atlantic interface clock (`arxclk`). The buffer provides clock decoupling.

Receiver Buffers

The buffer size can be configured to 4, 8, 16, or 32 Kilobytes.

See Table 1–3 on page 1–4 for examples of memory usage depending on on buffer size.

Transmitter

The layer 3 transmitter sub-layer accepts packet data from the Atlantic interface, and stores it into its buffer for the layer 1 sub-layer.

The RapidIO Specification requires that newly arrived, higher priority packets be transmitted ahead of the retransmission of previously transmitted, but not acknowledged, (retried) lower priority packets. The Specification also requires that at least one packet, of higher priority than all previously transmitted packets, always be able to pass through. To meet these requirements, the layer 3 transmitter sub-layer includes four transmit queues and four retransmit queues, one for each priority level.

Transmit & Retransmit Queues

As packets are written to the transmitter's Atlantic interface, they are added to the tail end of the appropriate priority transmit queue. The transmitter always transmits the packet at the head of the highest priority non-empty queue. Once transmitted, the packet is moved to the corresponding priority retransmit queue.

When a packet-accepted control symbol is received for the first non-acknowledged transmitted packet, the accepted packet is removed from its retransmit queue.

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

The layer 2 ensures that no more than 31 unacknowledged packets are transmitted, and that the AckIDs are used and acknowledged in incrementing order. The layer 3 removes packets from the appropriate retransmit queues when they are acknowledged.

Transmit Buffer

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

Clock & Data

The layer 3 transmitter sub-layer requires two clock domains: an internal global clock (`txclk`), and an Atlantic interface slave clock (`atxclk`). The Atlantic interface provides clock decoupling.

Forced Compensation Sequence Insertion

As packet data is written to the transmit Atlantic interface, it is stored in 64-byte blocks. To minimize the latency introduced by the RapidIO MegaCore function, 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 in lieu of idles as the true idle sequence can only be inserted in between packets, and not inside a packet. This, along with other embedded symbols, such as packet-accepted symbols, causes the transmission of the packet to be stretched in time.

Compensation sequences must be inserted every 5,000 code groups or columns, and must be inserted between packets. The serial RapidIO MegaCore function checks whether the 5,000 code group deadline is approaching before the transmission of every packet, and inserts a compensation sequence when the number of code groups or columns, left before the required compensation sequence insertion, falls below a specified threshold.

That threshold is chosen to allow time for the transmission of a packet of maximum legal size (276 bytes), even when it is stretched by the insertion of a significant number of embedded symbols. (Up to 37 embedded symbols, or 148 bytes, theoretically need to be embedded should the traffic in the other direction consist of minimum-sized packets.)

In some cases, for example when using an extremely slow transmit Atlantic clock, the transmission of a packet can be stretched beyond the threshold, to the point where a compensation sequence must be inserted. When this occurs, the packet transmission is aborted with a `stomp` symbol, the compensation sequence is inserted, and normal transmission resumes.

When the receive side receives the stomped packet, it simply marks it as errored by asserting `arxerr`.

No traffic is lost and no protocol violation occurs, but an unexpected `arxerr` assertion occurs.

OpenCore Plus Time-Out Behavior

OpenCore® Plus hardware evaluation can support the following two modes of operation:

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

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

For MegaCore functions, the untethered timeout is 1 hour; the tethered timeout value is indefinite.

Your design stops working after the hardware evaluation time expires. After that time, the RapidIO Physical Layer MegaCore function behaves as though its Atlantic interface signals `atxena` and `arxena` are tied low.

As a result, it is impossible for the RapidIO MegaCore function to transmit new packets (it will only transmit idles and status control symbols), or read packets out of the Atlantic interface. If the far end continues to transmit packets, the RapidIO MegaCore function starts refusing new packets by sending `packet_retry` control symbols once its receiver buffer fills up beyond the corresponding threshold.

For more information on OpenCore Plus hardware evaluation using the RapidIO Physical Layer, see *AN 320: OpenCore Plus Evaluation of Megafunctions*.

Parameters

Table 3–4 shows the RapidIO Physical Layer function parameters, which can only be set in IP Toolbench (see "Step 1: Parameterize" on page 2–7).

Table 3–4. Serial RapidIO Physical Layer Parameters

Parameter	Value	Description
`Device family`	Stratix GX	The Stratix GX device family with its clock and data recovery input/outputs is the only family that meets the serial RapidIO specifications.
`Number of lanes`	1× or 4×	One or four lane high-speed data serialization or deserialization (up to 3.125 Gbps for 1x serial; up to 2.5 Gbps for 4x serial)
`Baud rate`	From 500 to 3.125 Mbaud	The baud rate is the external device's serial data rate. The serial RapidIO specification specifies baud rates of 1.25 to 3.125 Mbaud. Table 3–2 on page 3–9 shows the relationship between baud rates and internal clock rates.
`Atlantic interface port width`	32 or 64 bits	The Atlantic interface packet data path can be configured for 32 or 64 bits. The 1× serial only supports 32 bits, the 4× serial supports both 32 and 64 bits. See "Atlantic Interface" on page 3–3 for further details.
`Tx buffer size`	8, 16, or 32 Kbytes *(1)*	The Tx buffer size parameter allows you to select the transmitter buffer size.
`Rx buffer size`	4, 8, 16, or 32 Kbytes *(1)*	The Rx buffer size parameter allows you to select the receiver buffer size.
`Receive priority 0/1/2 retry threshold`	Threshold_2 > 9 Threshold_1 > Threshold_2 + 4 Threshold_0 > Threshold_1 + 4 Threshold_0 < (2 ** p_rxbuf_addr_width)	When the number of available free 64-byte blocks in the receive buffer is less than one of these thresholds, the receiver refuses incoming packets of the corresponding priority level by sending packet-retry symbols.

Note to Table 3–4:
(1) Buffers are implemented in M512 and M4K RAM blocks. Depending on the size of the device used, the maximum buffer size may be limited by the number of available RAM blocks.

Signals

Tables 3–5 through 3–10 list the pins used by the serial RapidIO Physical Layer MegaCore function, with the I/Os shown in Figure 3–1 on page 3–2. The active-low signals are indicated by `_n`.

For signals and bus widths specific to your variation, refer to the HTML file generated by IP Toolbench (see Table 2–1 on page 2–15).

Tables 3–5 and 3–6 list the signals used in the serial layer 1.

Table 3–5. Serial RapidIO Interface Layer 1 Signals

Signal	Direction	Description
`rd`	Input	Receive data—a unidirectional data receiver. It is connected to the `td` bus of the transmitting device.
`td`	Output	Transmit data—a unidirectional point-to-point driver to transmit the packet information. The `td` bus of one device is connected to the `rd` bus of the receiving device.

Table 3–6. Serial Layer 1 Global Signals

Signal	Direction	Description
`clk`	Input	Reference clock.
`reset_n`	Input	Active-low reset. `reset_n` can be asserted asynchronously but must be deasserted synchronously with `clk`.
`rxclk`	Output	Receive-side recovered clock.
`input_enable`	Input	Enables the inputs. This signal is an input port driven by user logic. The signals driving this input port must be in the `clk` clock domain. Logically `OR`ed with the register bit `IN_PENA`. The `input_enable` signal is used to enable the input ports directly, removing the need to set up the register bits via the AIRbus interface.
`output_enable`	Input	Enables the outputs. This signal is an input port driven by user logic. The signals driving this input port must be in the `clk` clock domain. Logically `OR`ed with the register bit `OUT_PENA`. The `output_enable` signal is used to enable the output ports directly, removing the need to set up the register bits via the AIRbus interface.
`port_initialized`	Output	This signal indicates that the serial RapidIO initialization sequence has completed successfully. This is a level signal asserted high while the initialization state machine is in the 1X_MODE or 4X_MODE state, as described in *paragraph 4.6 of Part VI of the RapidIO Specification.*
`simulation_speedup`	Input	This port should be tied low in normal operation. When this port is tied high, some delays are shortened to reduce simulation time. Fore example, the millisecond delays required for PLLs to stabilize are shortened to a handful of clock cycles.

Table 3–7 lists the signals used in the layer 2.

Table 3–7. Serial Layer 2 AIRbus Interface Signals *Note (1)*

Signal	Direction	Description
`sel`	Input	AIRbus interface selects.
`addr`	Input	AIRbus address bus.
`read`	Input	AIRbus read.
`wdata`	Input	AIRbus write data bus.
`rdata`	Output	AIRbus read data bus.
`dtack`	Output	AIRbus interface data transfer acknowledge.

Note to Table 3–7:

(1) Although the AIRbus interface specification lists a clock (`clk`) and an interrupt request (`irq`) as part of its signals, the RapidIO MegaCore function does not have an AIRbus-specific clock, or an `irq` signal.

Tables 3–8 through 3–9 list the signals used in the layer 3. .

Table 3–8. Serial Layer 3 Atlantic Receive Interface Signals

Signal	Direction	Description
`arxclk`	Input	Receive clock.
`arxreset_n`	Input	Receive active-low reset. `arxreset_n` can be asserted asynchronously but should be deasserted on the rising edge of `arxclk`.
`arxena`	Input	Receive enable.
`arxdav`	Output	Receive data available. The `arxdav` signal is asserted when at least one complete packet is available to be read from the receive buffer. It is deasserted when the receive buffer does not have at least one complete packet available.
`arxdat`	Output	Receive data bus.
`arxval`	Output	Receive data valid.
`arxsop`	Output	Receive start of packet.
`arxeop`	Output	Receive end of packet.
`arxmty`	Output	Number of invalid bytes on `arxdat`.
`arxerr`	Output	Receive data error.
`arxwlevel` *(1)*	Output	Receive buffer write level (number of free 64-byte blocks in the receive buffer).

Note to Table 3–8:

(1) The following equation: `log2(size of the receive buffer in bytes/64)` determines the number of bits. For example, a receive buffer size of 16 would give: `log2(16×1024/64)= 8 bits (i.e., [7:0])`.

Table 3–9. Serial Layer 3 Atlantic Transmit Interface Signals

Signal	Direction	Description
`atxclk`	Input	Transmit clock.
`atxreset_n`	Input	Transmit active-low reset. `atxreset_n` can be asserted asynchronously but should be deasserted on the rising edge of `atxclk`.
`atxena`	Input	Transmit enable.
`atxdav`	Output	Transmit data available. `atxdav` is asserted when the transmit buffer has space to accept at least one maximum size packet (i.e., 276 bytes). It is deasserted when it does not have space to accept at least one maximum size packet.
`atxdat`	Input	Transmit data bus.
`atxsop`	Input	Transmit start of packet.
`atxeop`	Input	Transmit end of packet.
`atxmty`	Input	Number of invalid bytes on `atxdat`.
`atxerr`	Input	Transmit data error.
`atxwlevel` *(1)*	Output	Transmit buffer write level (number of free 64-byte blocks in the transmit buffer).
`atxovf`	Output	Transmit buffer overflow. If a new packet is started by asserting `atxena` and `atxsop` three or more `atxclk` clock cycles after `atxdav` is deasserted, `atxovf` is asserted and the packet is ignored.

Note to Table 3–9:

(1) The following equation: `log2(size of the transmit buffer in bytes/64)-1` determines the number of bits. For example, a transmit buffer size of 16 would give: `log2(16×1024/64)-1 = 7 bits (i.e., [6:0])`.

Table 3–10 shows the packet and error monitoring signals for the serial RapidIO MegaCore function.

Table 3–10. Packet and Error Monitoring Signals (Part 1 of 2)

Signal	Direction	Clock Domain	Description
`packet_transmitted`	Output	`clk`	Pulsed high for one clock cycle when a packet's transmission completes normally.
`packet_cancelled`	Output	`clk`	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 symbol.
`packet_accepted`	Output	`clk`	Pulsed high for one clock cycle when a packet-accepted symbol is being transmitted.

Table 3–10. Packet and Error Monitoring Signals (Part 2 of 2)

Signal	Direction	Clock Domain	Description
`packet_retry`	Output	`clk`	Pulsed high for one clock cycle when a packet-retry symbol is being transmitted.
`packet_not_accepted`	Output	`clk`	Pulsed high for one clock cycle when a packet-not-accepted symbol is being transmitted.
`packet_crc_error`	Output	`rxclk`	Pulsed high for one clock cycle when a CRC error is detected in a received packet.
`symbol_error`	Output	`rxclk`	Pulsed high for one clock cycle when a corrupted symbol is received.
`char_err`	Output	`rxclk`	Pulsed for one clock cycle when an invalid character or a valid but illegal character is detected.

Table 3–11 shows the register-related signals for the serial RapidIO MegaCore function.

Table 3–11. Register-Related Signals

Signal	Direction	Description
`ef_ptr[15:0]`	Input	Most significant bits [31:16] of the PHEAD0 register.
`port_response_timeout[23:0]`	Output	Most significant bits [31:8] of PRTCTRL register.

Table 3–12 shows the `ALTGXB` and `ALTPLL` megafunction signals for the serial RapidIO MegaCore function.

Table 3–12. ALTGXB & ALTPLL Megafunction Signals

Signal	Direction	Clock Domain	Description
`tx_preemphasisctrl`	Input	`clk`	These ports are directly connected to the corresponding input ports of the `ALTGXB` megafunction. Refer to the `ALTGXB` megafunction's on-line documentation for details.
`rx_equalizerctrl`	Input	`clk`	
`tx_vodctrl`	Input	`clk`	
`gxbpll_locked`	Output	`clk`	Connected to the `pll_locked` output port of the `ALTGXB` megafunction.
`txpll_locked`	Output	`clk`	Connected to the `locked` output port of the `ALTPLL` megafunction.

Registers

All addresses access 32-bit registers and are shown as hexadecimal values. The access addresses for each register increment by units of 4. Table 3–13 shows the memory map for the serial RapidIO Physical Layer function.

Table 3–13. Master Memory Map

Address	Name	Description
`'h100`	`PHEAD0`	Port Maintenance Block Header 0
`'h104`	`PHEAD1`	Port Maintenance Block Header 1
`'h120`	`PLTCTRL`	Port Link Time-out Control CSR
`'h124`	`PRTCTRL`	Port Response Time-out Control CSR
`'h13C`	`PGCTRL`	Port General Control CSR
`'h158`	`ERRSTAT`	Port 0 Error and Status CSR
`'h15C`	`PCTRL0`	Port 0 Control CSR

Table 3–14 lists the access codes used to describe the type of register bits.

Table 3–14. 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

Master Register Description

Tables 3–15 through 3–21 describe the registers for the master functions of the serial RapidIO MegaCore function. The offset values are as defined by the RapidIO standard.

Table 3–15. PHEAD0—Port Maintenance Block Header 0—'h100

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 input from the `ef_ptr` input signal.	`ef_ptr`
`EF_ID`	15:0	`RO`	Hard-wired extended features ID.	`‘h0004`

Table 3–16. PHEAD1—Port Maintenance Block Header 1—'h104

Field	Bits	Access	Function	Default
`RSRV`	31:0	`URO`	Reserved.	0

Table 3–17. PLTCTRL—Port Link Time-Out Control CSR—'h120

Field	Bits	Access	Function	Default
`VALUE`	31:8	`RW`	Time-out interval value.	`'hffffff`
`RSRV`	7:0	`URO`	Reserved.	0

Table 3–18. PRTCTRL—Port Response Time-Out Control CSR—'h124

Field	Bits	Access	Function	Default
`VALUE`	31:8	`RW`	Time-out internal value. This value is not used by the RapidIO Physical Layer MegaCore function. The contents of this register are brought out to the `port_response_timeout` output signal.	`'hffffff`
`RSRV`	7:0	`URO`	Reserved.	0

Table 3–19. PGCTRL—Port General Control CSR—'h13C

Field	Bits	Access	Function	Default
HOST	31	RW	A host device is a device that is responsible for system exploration, initialization, and maintenance. Agent or slave devices are typically initialized by host devices. 'b0—agent or slave device. 'b1—host device.	0
ENA	30	RW	The master enable bit controls whether or not a device is allowed to issue requests into the system. If the master enable is not set, the device may only respond to requests. 'b0—processing element cannot issue requests. 'b1—processing element can issue requests.	0
DISCOVERED	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.	0
RSRV	28:0	UR0	Reserved.	0

Table 3–20. ERRSTAT—Port 0 Error and Status CSR—'h158 (Part 1 of 2)

Field	Bits	Access	Function	Default
RSRV	31:21	UR0	Reserved.	0
OUT_RTY_ENC	20	RW1C	Output port has encountered a retry condition. This bit is set when bit 18 is set.	0
OUT_RETRIED	19	RO	Output port has received a packet-retry control symbol and cannot make forward progress. This bit is set when bit 18 is set. This bit is cleared when a packet-accepted or a packet-not-accepted control symbol is received.	0
OUT_RTY_STOP	18	RO	Output port has received a packet-retry control symbol and is in the output retry-stopped state.	0
OUT_ERR_ENC	17	RW1C	Output port has encountered (and possibly recovered from) a transmission error. This bit is set when bit 16 is set.	0
OUT_ERR_STOP	16	RO	Output port is in the output error-stopped state.	0
RSRV1	15:11	UR0	Reserved.	0
IN_RTY_STOP	10	RO	Input port is in the input retry-stopped state.	0

Table 3–20. ERRSTAT—Port 0 Error and Status CSR—'h158 (Part 2 of 2)

Field	Bits	Access	Function	Default
`IN_ERR_ENC`	9	`RW1C`	Input port has encountered (and possibly recovered from) a transmission error. This bit is set when bit 8 is set.	0
`IN_ERR_STOP`	8	`RO`	Input port is in the input error-stopped state.	0
`RSRV2`	7:5	`UR0`	Reserved.	0
`PWRITE_PEND`	4	`UR0`	This register is not implemented and is reserved. It is always set to zero.	0
`RSRV3`	3	`UR0`	Reserved.	0
`PORT_ERR`	2	`RW1C`	Input or output port has encountered an error from which hardware was unable to recover.	0
`PORT_OK`	1	`RO`	Input and output ports are initialized and the port is exchanging error-free control symbols with the adjacent device.	0
`PORT_UNINIT`	0	`RO`	Input and output ports are not initialized. This bit and bit 1 are mutually exclusive.	`'b1`

Table 3–21. PCTRL0—Port 0 Control CSR—'h15C (Part 1 of 2)

Field	Bits	Access	Function	Default
`PORT_WIDTH`	31:30	`UR0`	Hardware width of the port: `'b00`—Single-lane port. `'b01`—Four-lane port. `'b10`–'`b11`—Reserved.	0
`INIT_WIDTH`	29:27	`UR0`	Width of the ports after initialized: `'b000`—Single lane port, lane 0. `'b001`—Single lane port, lane 2. `'b010`—Four lane port. `'b011–'b111`—Reserved.	
`PWIDTH_OVRIDE`	26:24	`UR0`	Soft port configuration to override the hardware size: `'b000`—No override. `'b001`—Reserved. `'b010`—Force single lane, lane 0. `'b011`—Force single lane, lane 2. `'b100–'b111`—Reserved.	0
`PORT_DIS`	23	`RW`	Port disable: `'b0`—port receivers/drivers are enabled. '`b1`—port receivers are disabled, causing the drivers to send out idles.	0

Table 3–21. PCTRL0—Port 0 Control CSR—'h15C (Part 2 of 2)

Field	Bits	Access	Function	Default
`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 any packets.	1
`IN_PENA`	21	`RW`	Input port receive enable: `'b0`—port is stopped and only enabled to respond I/O logical maintenance packets, depending upon the functionality of the processing element. Other packets generate packet-not-accepted control symbols to force an error condition to be signalled by the sending device. Control symbols are not affected and are received and handled normally. `'b1`—port is enabled to respond to any packet.	1
`ERR_CHK_DIS`	20	`RW`	This bit disables 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.	0
`MULTICAST`	19	`RW`	Send incoming multicast-event control to this port (multiple port devices only).	0
`RSRV2`	18:1	`UR0`	Reserved.	0
`PORT_TYPE`	0	`UR1`	This indicates the port-type, parallel or serial. `'b0`—Parallel port. `'b1`—Serial port.	1

4. Parallel Specifications

Functional Description

This section describes the parallel RapidIO MegaCore® function, which is subdivided into three sub-layers. The following is a list of features for the three sub-layers.

Layer 1 Features

- 8-bit parallel True-LVDS™ high-speed interface
- I/O port training function to set up byte alignment
- Receiver
  - Packet/control symbol delineation
  - Parity checking on control symbols
  - Cyclic redundancy code (CRC) checking on packets
  - Idle symbol extraction
- Transmitter
  - Packet/control symbol assembly
  - Parity generation on control symbols
  - CRC generation on packets
  - Idle symbol insertion

Layer 2 Features

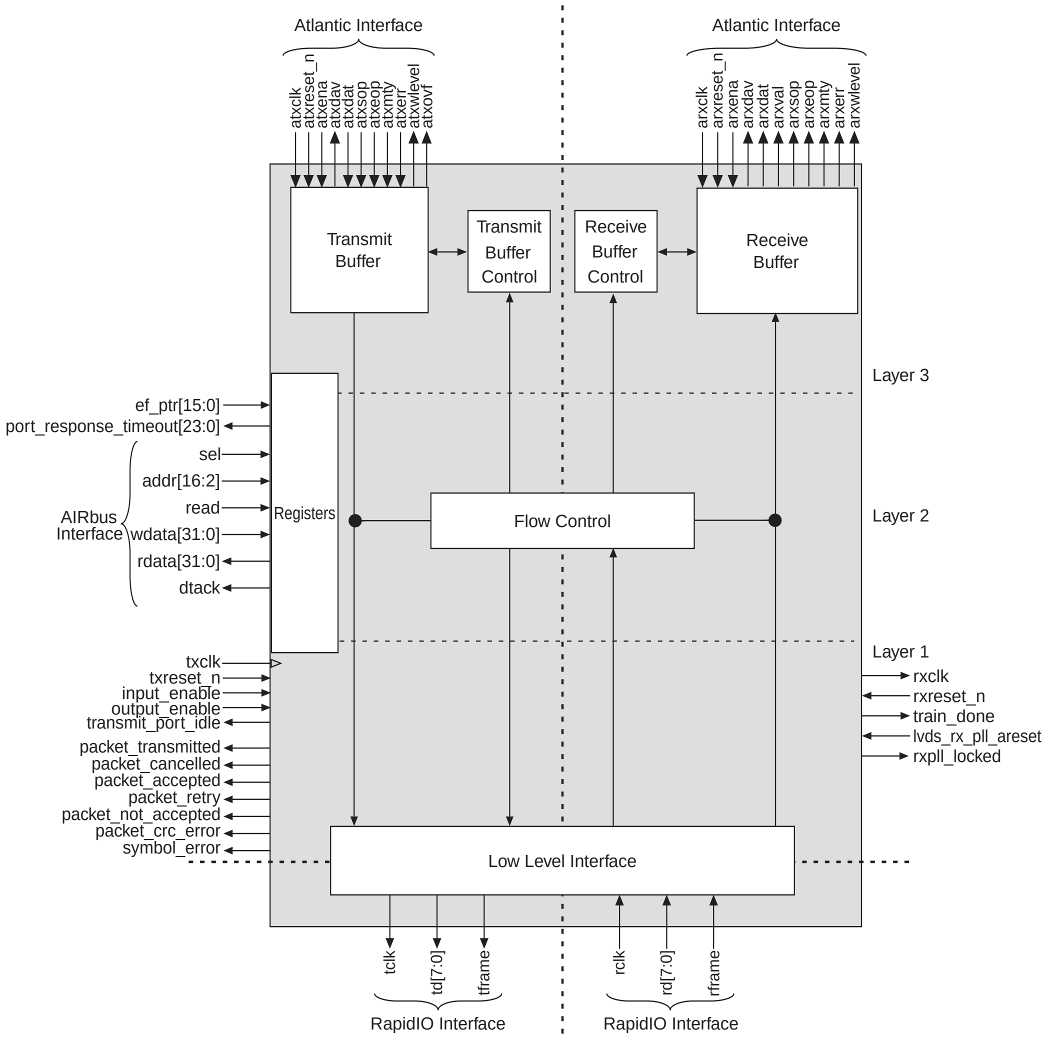
- Processor access
  - Symbol queue, status
- Flow control (`AckID` window tracking)
  - Time-out on acknowledgements
- Order of retransmission maintenance, and acknowledgements
- `AckID` assignment
- Error management

Layer 3 Features

- Atlantic™ interface with clock decoupling
- Extended tags
- FIFO buffer level output port
- Asymmetric buffer sizes
- Transmitter
  - Four transmission queues, and four retransmission queues to handle packet prioritization
  - Up to 32 Kbyte buffers
- Receiver
  - Up to 32 Kbyte buffers

Figure 4–1 shows a high-level block diagram of the parallel RapidIO MegaCore function.

Figure 4–1. Parallel RapidIO Block Diagram



Interfaces & Protocols

Three interfaces support the RapidIO MegaCore function: the RapidIO™ interface, the Atlantic interface, and the access to internal registers (AIRbus) interface.

RapidIO Interface

RapidIO is a packet-switched interconnect protocol defined by the RapidIO Trade Association. The protocol is divided into a three-layer hierarchy: physical layer, transport layer, and logical layer.

The RapidIO Physical Layer MegaCore function implements only the physical layer, which is further divided into three sub-layers: Layer 1, Layer 2, and Layer 3.

Table 4–1 shows the different layers and sub-layers, and their respective functions.

Table 4–1. RapidIO Layers

RapidIO Layer		OSI Layer	Description
Logical		Transport and upper layers	End point operation protocols
Transport		Network layer	Point to point packet delivery addressing scheme
Physical	Layer 3	Physical and Data link layer	Buffering
	Layer 2		Flow control
	Layer 1		Electrical (True-LVDS) interface, clock and data recovery (CDR), error detection, packet assembling and delineation

More detailed information on the RapidIO interface is available from the RapidIO Trade Association's web site at **www.rapidio.org**.

Atlantic Interface

The Atlantic interface, an Altera® protocol, is the user interface. It also connects the different sub-layers of the RapidIO MegaCore function. For parallel configurations, the width of this interface can be configured for 32 or 64 bits.

The transmit Atlantic interface supports 32- or 64-bit packet data transfers. It works as a slave-sink interface. The receive Atlantic interface supports 32- or 64-bit packet data transfers. It works as a slave-source interface.

The Layer 2 monitors packet data flow between the Layer 1 and the Layer 3 to provide flow control and generate the appropriate control symbols.

The `arxdav` signal is asserted when a full packet is available to be read from the receive buffer.

If the `arxena` signal is asserted when the `arxdav` signal is not asserted, the first word becomes available on the Atlantic interface, and the `arxval` signal is asserted as soon as the first 64-byte block of a packet (or the full packet if it is smaller than 64 bytes) is ready to be read out of the receive buffer. Thus, the MegaCore function does not wait for the full packet before reading out the first block.

Error Handling

The `arxerr` signal can be asserted for a variety of reasons, listed below. As an Atlantic signal, it is synchronous to `arxclk` and is only valid when `arxval` is asserted. Once asserted, `arxerr` stays asserted until the end of the packet when `arxeop` is asserted.

- CRC error—When a CRC error is detected, the `packet_crc_error` signal is asserted for one `rxclk` clock cycle. The `packet_not_accepted` signal is asserted when the `packet_not_accepted` symbol is transmitted. The `arxerr` signal is also asserted when the packet is read out of the Atlantic interface.

- Stomp—The `arxerr` signal is asserted if a `stomp` control symbol is received in the midst of a packet, causing it to be prematurely terminated. The `arxerr` signal is also asserted for any packet received between the `stomp` symbol and the following `restart-from-retry` symbol.

- Packet size—If a received packet exceeds the allowable size, it is cut short to the maximum allowable size (276 bytes total), and `arxerr` and `arxeop` are asserted on the last word.

- Outgoing symbol buffer full—Under some congestion conditions, there may be no space in the outgoing symbol buffer for the `packet_accepted` symbol. If this happens, the packet cannot be acknowledged and will have to be retried. Thus, `arxerr` is asserted to indicate to the downstream circuit that the received packet should be ignored because it will be retried.

- Symbol error —If an embedded symbol is errored, `arxerr` is asserted and the packet in which it is embedded is retried.

AIRbus Interface

The AIRbus interface, in Layer 2, provides access to internal registers using a simple synchronous internal processor bus protocol. This consists of separate read data (`rdata[31:0]`) and write data (`wdata[31:0]`) buses, a data transfer acknowledge (`dtack`) signal, and a block-select (`sel`) signal. An address (`addr[16:2]`) bus and read (`read`) signal indicate the location and type of access within the block. The `rdata` buses and `dtack` signals can be merged from multiple blocks using a simple `OR` function. The `dtack` signal is sustained until the `sel` signal is removed (four-way handshaking), meaning the AIRbus can cross clock domain boundaries. All registers are 32-bits wide.

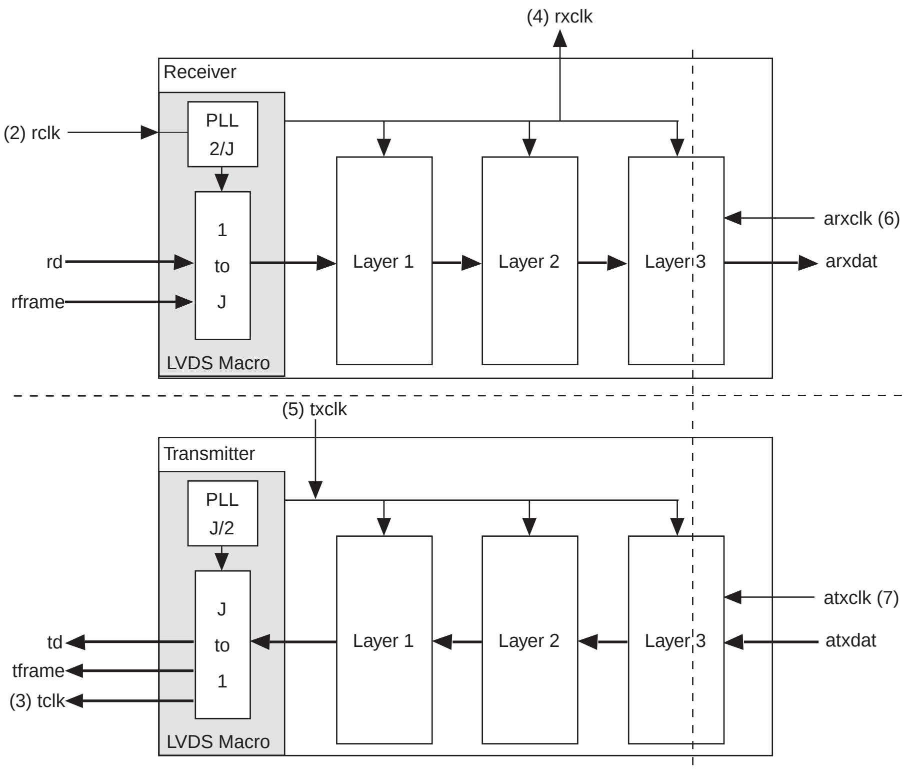
Although the AIRbus interface specification lists a clock (`clk`) and an interrupt request (`irq`) as part of its signals, the RapidIO MegaCore function does not have an AIRbus-specific clock, or an irq signal.

More detailed information on the Atlantic and AIRbus interfaces is available from the Altera web site at **www.altera.com**.

Clock Domains

The RapidIO MegaCore function comprises six clock domains: four interface clocks, and two global clocks, as illustrated in Figure 4–2 on page 4–6.

Figure 4–2. Clock Domains Note (1)



Notes to Figure 4–2:

(1) The `J` setting controls the width of the data bus driven into the transmitter or out of the receiver. `J` is the deserialization factor, and can be equal to 4 or 8.
(2) `rclk`: RapidIO interface clock.
(3) `tclk`: RapidIO interface clock.
(4) `rxclk`: Receiver internal global clock; `rxclk` is received from `rclk`.
(5) `txclk`: Transmitter global clock—reference for `tclk` generation.
(6) `arxclk`: Atlantic interface clock.
(7) `atxclk`: Atlantic interface clock.

All reset signals can be asserted asynchronously to any clock. However, most reset signals must be deasserted synchronously to a specific clock. The Atlantic interface resets, for example, should be deasserted on the rising edge of the corresponding clock.

See “Signals” on page 4–25

For configurations that have a 32-bit Atlantic width and dynamic phase alignment (DPA) enabled, the `rxreset_n` signal is an asynchronous reset signal and therefore can be deasserted asynchronously.

Table 4–2 shows the minimum external and internal clock required by the RapidIO specification.

Table 4–2. External and Internal Clock Rates

LVDS Data Rates	Double Data Rate (DDR) Clock	RapidIO Port Width	Internal Rate (`rxclk` & `txclk`)		Throughput Bit Rate
			32-Bit Width	64-Bit Width	
500 Mbps	250 MHz	8 bits	125 MHz	62.5 MHz	4 Gbps
750 Mbps	375 MHz	8 bits	–	93.75 MHz	6 Gbps
1 Gbps	500 MHz	8 bits	–	125 MHz	8 Gbps

For more information on using high-speed I/O, refer to *AN 202: Using High-Speed Differential I/O Interfaces in Stratix Devices*.

For more information on using high-speed transceiver blocks, refer to *AN 237: Using High-Speed Transceiver Blocks in Stratix GX Devices.*

Dynamic Phase Alignment

DPA is used for source-synchronous interface protocols that operate at increasingly higher data rates (e.g., 1 Gbps). DPA allows devices to respond to changes in the operational board skew. Devices equipped with DPA continuously check the incoming data and adjust the phase of the clock to align with it. Several industry standards for chip-to-chip interfaces, including RapidIO, have recognized the value of DPA, and have included or recommended it in their specifications.

Every Stratix GX and Stratix II receiver channel features an embedded DPA block. For Stratix GX devices, the DPA blocks are located in I/O banks 1 and 2; for Stratix II devices, the DPA blocks are located in banks 1, 2, 5 and 6. A complete, FPGA-integrated hard-silicon DPA solution offers several benefits to system designers. It is implemented for each data channel, such that each channel receives its own phase-adjusted clock. This individual alignment for each channel minimizes the chance for errors introduced by mismatches in signal propagation paths. Also, it does not require a training mode; rather, it continuously realigns the clock to the data during device operation. The training patterns specified by standards such as RapidIO are supported, but no training pattern is required when using DPA with other interfaces.

The RapidIO MegaCore function also features an integrated DPA block on its receiving path, between the high-speed serial bus and the parallel bus. The RapidIO DPA block functions include: data deserialization and clock division, dynamic phase alignment, and byte alignment.

The RapidIO DPA block makes use of the DPA capability of Stratix II or Stratix GX devices, supporting data rates of up to 1 Gbps. The DPA block is configured to support 9 hi-speed channels, and internal data widths of 32 or 64 bits.

Features

- Dynamic clock-data synchronization (phase alignment) to compensate the clock-channel and channel-channel skew
- Dynamic byte alignment using training patterns
- Supports the RapidIO protocol
- Corrects the data skew difference of up to +/- 0.75 bits (1.5 bits)
- Supports a serialization/deserialization (SERDES) factor of 8 and 4
- Supports data rates from 350 Mbps to 1 Gbps

☞ Stratix II or Stratix GX devices and DPA are required for performance beyond 750 Mbps.

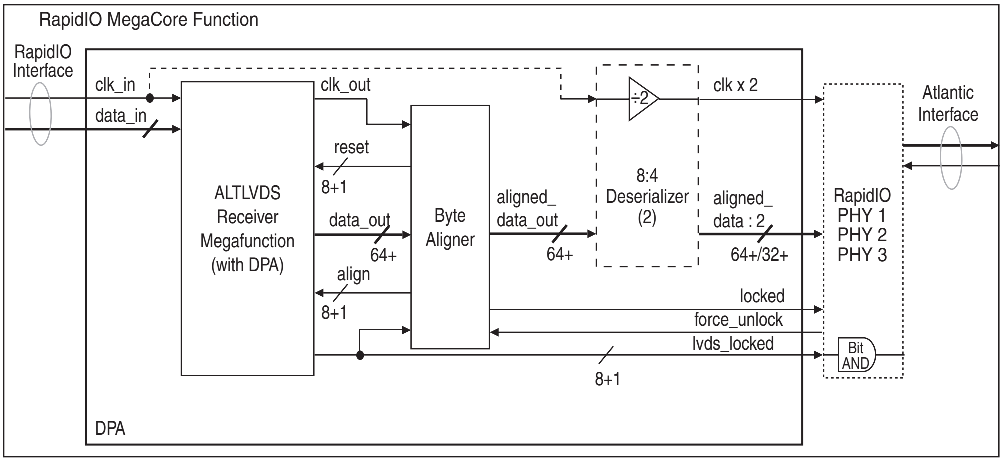
Functional Description

The DPA block takes in the serial hi-speed phase/channel/byte misaligned serial data, and outputs the phase/channel/byte aligned parallel data and clock.

The block consists of the following sub-blocks: an ALTLVDS receiver megafunction (with the DPA feature enabled), a byte aligner, and an 8:4 deserializer (needed to achieve a deserialization factor of 4 in Stratix GX devices only). It also consists of two status signals: `locked` and `lvds_locked` (one bit per channel), and one control signal: `force_unlock`.

The `rxdpa_locked` and `lvds_ch_locked` (bit `AND` of `lvds_locked`) status signals are accessible to the user. The `force_unlock` control signal is not accessible to the user. See Figure 4–3 on page 4–9.

Figure 4–3. DPA Block Diagram Note (1)



Notes to Figure 4–3:

(1) The width of the data path for the `data_out`, `aligned_data_out`, and `aligned_data:2` signals depends on the SERDES factor.

(2) Exists only for a deserialization factor of 4 and for Stratix GX devices.

ALTLVDS Receiver Megafunction

The ALTLVDS receiver megafunction performs the phase equalization (data and clock), the deserialization and the bit slipping (per channel) if requested by the byte aligner sub-block, via the `align` signal (one bit for each channel). The ALTLVDS receiver megafunction is generated by the MegaWizard® Plug-in Manager in the Quartus® II software.

Byte Aligner

The byte aligner sub-block aligns the parallel data (channel by channel). It pulses the `align` signal until all channels are aligned (based on the training patterns). For every `align` pulse, the ALTLVDS receiver megafunction sub-block shifts the data on the corresponding channel by one bit to the left. The byte aligner retimes the parallel data.

Alignment is done once at start-up, and whenever requested by the RapidIO processor by asserting the `force_unlock` signal (based on the RapidIO protocol link-response symbol time-out).

☞ If lock cannot be achieved after the state machine has received many `force_unlock` signals, it is a good indication that the LVDS is not locked on some or all channels, that the `lvds_locked` signal was deasserted during training, or that the `lvds_locked` signal is asserted but the channel-to-channel skew is greater than the maximum supported skew.

The byte aligner works in parallel, with one state machine per channel.

The byte aligner state machines begin the alignment process once the ALTLVDS receiver megafunction asserts the `lvds_locked` signal (high). During the alignment process, the byte aligner does not monitor the `lvds_locked` signal, and should it become deasserted (low) during alignment, the output data may be misaligned. Therefore it is recommended that the master RapidIO MegaCore function monitor the `lvds_locked` signal if alignment cannot be achieved, or if it detects too many DIP errors, indicating a misalignment.

RapidIO Training Pattern

The RapidIO training pattern consists of four 1s and four 0s (on all channels) repeating 256 times (the framing channel could also be inverted: four 0s and four 1s). The number of channels is 8+1 (data and framing), depending on the serialization factor.

For example, for 8+1 channels and a serialization factor of 8, the parallel data looks as shown in Table 4–3.

Table 4–3. Training Pattern Example

Cycle	Framing [7:0]	Data (Channel 7 … 0, bit [7:0])						
	Ch8	Ch7	Ch6	Ch5	Ch4	Ch3	...	Ch0
0	‘hF0 *(1)*	‘hF0	‘hF0	‘hF0	‘hF0	‘hF0	...	‘hF0
1	‘hF0 *(1)*	‘hF0	‘hF0	‘hF0	‘hF0	‘hF0	...	‘hF0
...	...	...	...	...	...	...	...	...

Note to Table 4–3:

(1) Could also be ‘h0F (see the *RapidIO™ Interconnect Specification*).

8:4 Deserializer

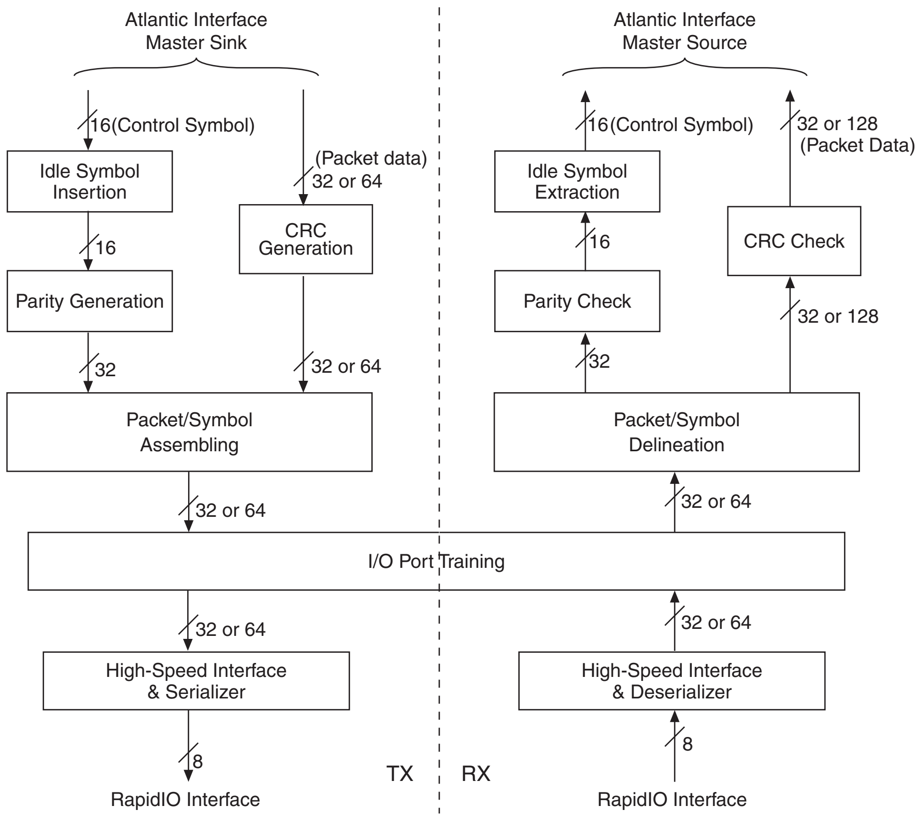
The 8:4 deserializer sub-block is only required in Stratix GX devices to support a deserialization factor of 4. It consists of a register-based clock divide-by-two circuit and 4-bit demultiplexers, one for each channel.

For more information on using dynamic phase alignment, refer to *AN 236: Using Source-Synchronous Signaling with DPA in Stratix GX Devices*.

Layer 1

The layer 1 sub-layer is designed to be a full-duplex interface with 8-bit unidirectional True-LVDS ports. This section gives a block-by-block description of the layer 1 functions. Figure 4–4 shows a detailed block diagram of the layer 1.

Figure 4–4. Layer 1 Data Flow Block Diagram



Receiver

The layer 1 receiver sub-layer receives and passes packets to the layer 3, and passes control symbols to the layer 2 over a slave-sink Atlantic interface.

Clock & Data

The layer 1 receiver requires two clock domains: a RapidIO interface clock (`rclk`), and an internal global clock (`rxclk`).

The 8-bit data is received from the RapidIO interface on True-LVDS pins (ports). The data associated with the clock is DDR.

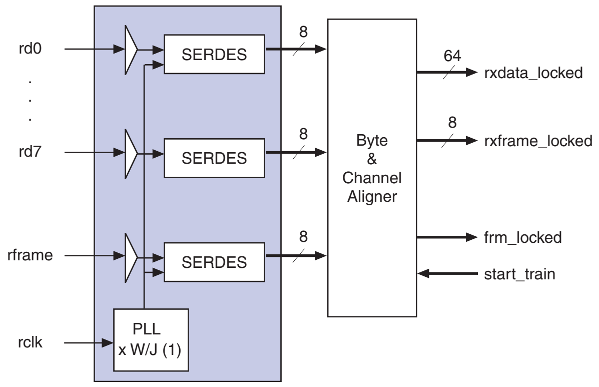
High-Speed Interface & Deserializer

Using the high-speed serial interface (HSSI) features of the Stratix devices, the 8-bit bus is deserialized to a 32- or 64-bit bus. The deserialization factor can be 4 or 8.

The system receives the serial data and clock on its input True-LVDS pins. Data words arrive on the `rd` bus at 2 × `rclk` (DDR). A high-frequency clock, generated by a PLL, shifts the serial data through the receiver's SERDES module. The deserialized data (`rd` line) is driven out in parallel with a low-frequency clock—the receiver's global clock (`rxclk`)—which drives the internal logic elements (LEs).

Figure 4–5 shows the layer 1 input port configuration: 8 True-LVDS pin data, 64-bit internal data path, rclk/4 internal clock.

Figure 4–5. Layer 1 8-/64-Bit Input Port Configuration



Note to Figure 4–5:
(1) W = 2; J = 8

Time Division Multiplexing

Input data packets are multiplexed in the same order as they are received on the input pins. The Stratix deserializer block outputs data pin-by-pin and the first bit received is the most significant bit (MSB).

I/O Port Training

The RapidIO interface is source synchronous, and requires link initialization to ensure that the input port receives the packets and control symbols properly. Link initialization is required in the following circumstances:

- System power-up
- Normal operation: system reset or error recovery (by request)
- Connecting to another processing element

To ensure proper reception, the receiver performs two procedures, which can be done in parallel, to initialize the input ports.

Sampling window alignment: During initialization, a predefined training pattern is applied to the input port to set it up for reliable data sampling. The training pattern is a special bit pattern that differs from the control symbols and packets, it is easily recognized by the input port logic when initialization is required, but ignored when not required.

The training pattern includes the following characteristics:

- 32 bits of binary 1 followed by 32 bits of binary 0
- A frame signal switches at the same time as the data bits
- The frame signal does not have to transition high to low or low to high in phase with the data bits
- The same pattern is sent from the output port for 256 times followed by an idle symbol for each send-training request.

☞ A training pattern cannot be embedded within, or terminate, a packet.

32-bit boundary alignment: In order to ensure that the input port is aligned to the 32-bit boundary of the connected output port, all control symbols are delineated on this 32-bit boundary, thus allowing for an aligned and steady stream of frame signals.

I/O port training (initialization) requires two state machines: input and output port training state machines.

☞ A port can only transition from its unitialized state to its OK state when both state machines are in their OK states.

The layer 1 uses the I/O training pattern to perform byte alignment, under the control of the layer 2.

Packet/Symbol Delineation

The packet/symbol delineation block delineates the input data on a 32-bit boundary when the port is in its OK state, and the data is valid. It takes the input data stream and divides it into two streams: control symbols which go to the parity check block, and packet data which goes to the CRC check block.

This block also detects start of packet (SOP), EOP control symbol, and `s` bit errors.

Parity Check

The `s` bit is protected by an odd parity bit. The aligned control symbol is protected by a 16-bit inverted value. This block checks the inverted value and the `s` bit to ensure data integrity.

Idle Symbol Extraction

The idle symbol extraction block extracts and discards all control symbols with an undefined stype field, and sets a symbol valid signal if the appropriate control symbol is not an idle control symbol, and the `s` bit and the control symbol parity are correct.

CRC Check

The CCITT polynomial $x^{16} + x^{12} + x^5 + 1$ is used for CRC checking. 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 CRC is used and appended at the end.

For packets longer than 80 bytes, two CRCs are used. The first CRC is appended after the first 80 bytes; the second CRC is appended at the end of the packet.

This block also flags CRC errors, and packet size errors.

Transmitter

The layer 1 transmitter sub-layer assembles packets and control symbols, received over a slave-source Atlantic interface, into one message and passes it to the parallel RapidIO interface.

Clock and Data

The layer 1 transmitter uses two clocks: a global clock (`txclk`), and a RapidIO interface clock (`tclk`).

The `tclk` signal is a data output of the True-LVDS pins. It is generated by the deserialization of `txclk`.

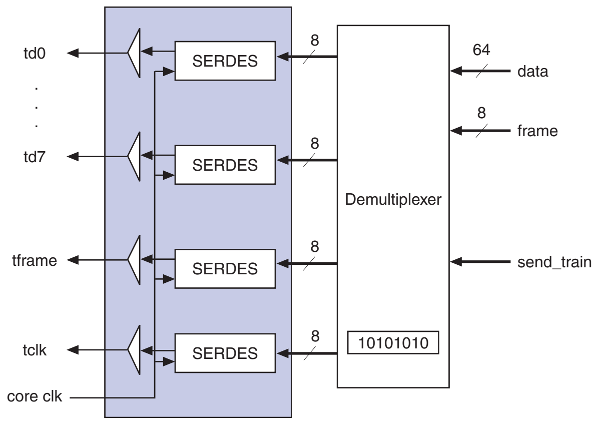
The data is transmitted on True-LVDS pins. The data associated with the clock is DDR.

High-Speed Interface and Serializer

Using the HSSI features of the Stratix devices, the 32- or 64-bit bus is serialized to an 8-bit bus. Data words are sent on the `td` data bus at 2×`tclk` rate. A SERDES module serializes the word inputs into output high-speed `td/tframe` lines.

Figure 4–6 shows the layer 1 output port configuration: 8 True-LVDS pin data, 64-bit internal data path, internal clock, and a core clock.

Figure 4–6. Layer 1 8-/64-bit Output Port Configuration *Note (1)*



Note to Figure 4–6:
(1) The deserialization factor is 8.

I/O Port Training

The output port sends out a training pattern on power-up, or when a link request/send-training control symbol is received.

The layer 1 generates the training pattern under the control of the layer 2.

See "I/O Port Training" on page 4–13, for more details.

Packet/Control Symbol Assembling

The packet/control symbol assembling block assembles the control symbol and packet data streams into the required output format, with corresponding framing signal.

Parity Generation

16-bit control symbols are followed by a bit-wise inversion of themselves for alignment on a 32-bit boundary, but this inversion also serves for parity error checking. This block generates the 16-bit inverted value.

Idle Symbol Insertion

The symbol insertion block inserts an idle symbol if a throttle request is received to add a wait state to the output packet, or if no data (packet or control symbol) is available for transfer.

CRC Generation

The CRC generation block generates a CRC over the entire packet header and data payload, except for the first six bits of the first packet, which are covered by protocol and parity.

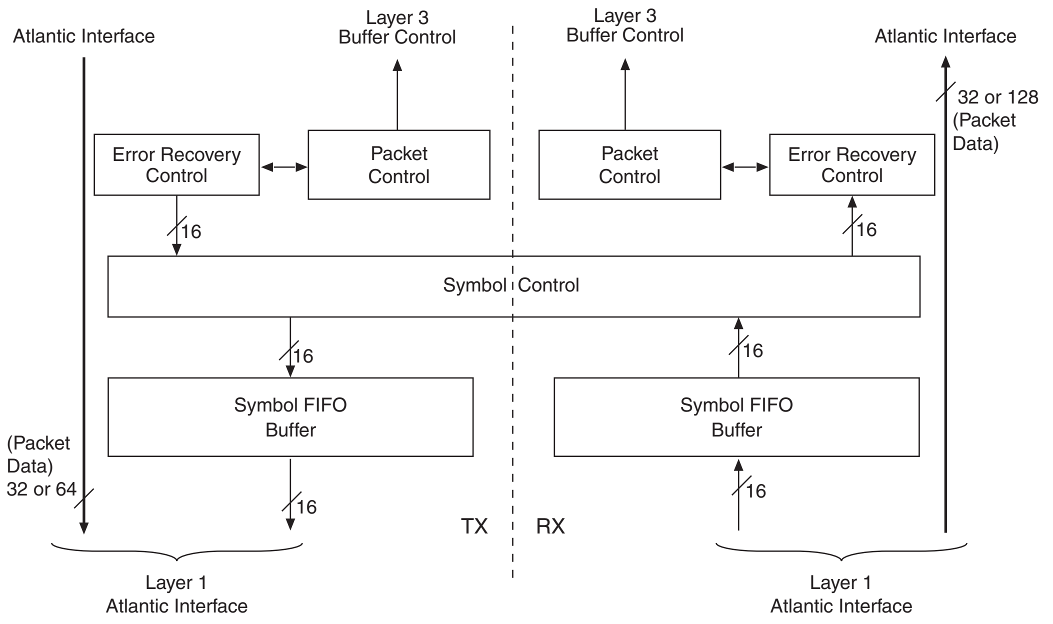
For packets of 80 bytes or fewer—header and payload data included—a single CRC is used and appended at the end.

For packets longer than 80 bytes, two CRCs are generated. The first CRC is appended after the first 80 bytes; the second CRC is appended at the end of the packet. The second CRC is a continuation of the calculation of the first CRC.

Layer 2

The layer 2 sub-layer provides flow control for the parallel RapidIO physical layer. This section gives a block-by-block description of the layer 2. Figure 4–7 on page 4–17 shows a detailed block diagram of the layer 2.

Figure 4–7. Layer 2 Data Flow Block Diagram



Receiver

The layer 2 receiver sub-layer is responsible for processing incoming control symbols. It also monitors incoming packet `ackID`s to maintain proper flow.

Clock and Data

The layer 2 receiver comprises one clock domain: an internal global clock (`rxclk`).

Symbol FIFO Buffer

Incoming symbols are stored in the receiver's symbol FIFO buffer by the layer 1. These symbols are retrieved by the layer 2 for further processing.

The AIRbus interface allows the processor to insert or extract symbols from the FIFO buffer. The receiver FIFO buffer connects to the layer 1 via an Atlantic slave interface.

Symbol Control

On the receive side, the layer 2 keeps track of the sequence of `ackID`s and tells the layer 3 which packets have been acknowledged, and which packets to drop.

Packet Control

The packet control block uses a sliding window protocol to handle incoming and outgoing packets. Each incoming and outgoing packet has an attached 3-bit `ackID` in the header field. The value of `ackID` is zero at reset. It increments after each packet is sent out, and rolls over to zero after it has reached seven. All packets can only be accepted by the receiver in the sequential order specified by the `ackID`. If a packet is lost at the receiver, a packet retry request with the lost `ackID` is sent to the sender. The sender then retransmits all packets starting from the lost `ackID`.

Error Recovery Control

A packet or control symbol corrupted by an incorrect CRC, or by a parity error, must be recovered. During the error recovery process, two interdependent state machines are required to operate the input and output ports, respectively.

When an incoming packet is corrupted, the receiver sends a packet not accepted acknowledgment to the sender. The sender then retransmits all packets starting from the retried `ackID` of the corrupted packet.

When an incoming control symbol is corrupted, the receiver sends a packet not accepted control symbol to inform the sender of the internal status, and the expected `ackID`. The sender then proceeds to retransmit the control symbol.

Transmitter

The layer 2 transmitter sub-layer is responsible for creating and transmitting outgoing control symbols. It also monitors outgoing packet `ackID`s to maintain proper flow.

Clock and Data

The layer 2 transmitter comprises one clock domain: an internal global clock (`txclk`).

Symbol FIFO Buffer

The layer 2 provides this symbol FIFO buffer to store outgoing symbols. These symbols are retrieved by the layer 1, and sent out via the output port.

The AIRbus interface allows the processor to insert or extract symbols from the FIFO buffer. The transmitter FIFO buffer connects to the layer 1 via an Atlantic slave interface.

Symbol Control

On the transmit side, the layer 2 keeps track of the sequence of transmitted and acknowledged `AckID`s.

Packet Control

The packet control block uses a sliding window mechanism to handle incoming and outgoing packets. This block also sets the time-out counters for each outgoing packet.

Error Recovery Control

An uncorrupted protocol violating control symbol, or a control symbol corrupted by an incorrect CRC, or by a parity error must to be recovered.

For error recovery, transmitted packets are held by the output port for possible retransmission in case an error is detected by the receiving device.

Layer 3

The layer 3 sub-layer provides buffers, and buffer management for packet data. This section gives a block-by-block description of the layer 3 functions.

Receiver

The layer 3 receiver sub-layer accepts packet data from the layer 1 sub-layer, and stores it in its buffers for the user. The receiver buffer is partitioned in 64-byte blocks that are allocated from a free queue as required, and returned to the free queue when no longer needed. Up to five 64-byte blocks can be used to store a packet.

The RapidIO Specification requires that at least one packet, of higher priority than all previously transmitted packets, always be able to pass through.

To meet this requirement, the layer 3 receiver sub-layer accepts or retries received packets based on their priority and the receive buffer's fill level.

The receiver bases its decision to accept or retry packets on three programmable threshold levels: Threshold_2, Threshold_1, and Threshold_0.

- Packets of priority 2′b11 (highest priority) are retried if the receiver buffer is full.
- Packets of priority 2′b10 are retried only if the number of available free 64-byte blocks is less than Threshold_2.
- Packets of priority 2′b01 are retried only if the number of available free 64-byte blocks is less than Threshold_1.
- Packets of priority 2′b00 (lowest priority) are retried only if the number of available free 64-byte blocks is less than Threshold_0.

The default threshold values are:

- Threshold_2 = 10
- Threshold_1 = 15
- Threshold_0 = 20

The threshold values are programmed through clear-text internal input ports in the riophy module, and can be set on the **Advanced** tab of IP Toolbench.

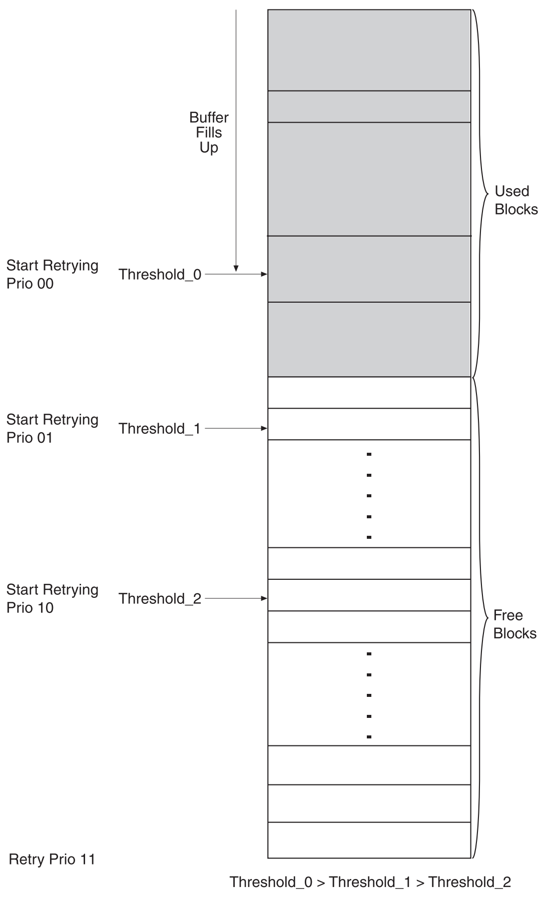
- `rx_threshold_0[p_rxbuf_addr_width:0]`
- `rx_threshold_1[p_rxbuf_addr_width:0]`
- `rx_threshold_2[p_rxbuf_addr_width:0]`

☞ Where `p_rxbuf_addr_width` corresponds to the internal effective address width.

To comply with the RapidIO specification, the threshold values increase monotonically by at least the size of one packet (see Figure 4–8 on page 4–21). The MegaWizard Plug-In generates the following parameters, and enforces the consistency checks.

- p_rx_threshold_2 > 9
- p_rx_threshold_1 > p_rx_threshold_2 + 4
- p_rx_threshold_0 > p_rx_threshold_1 + 4
- p_rx_threshold_0 < (2 ** p_rxbuf_addr_width)

Figure 4–8. Receiver Threshold Levels



Clock & Data

The layer 3 receiver sub-layer comprises two clock domains: an internal global clock (`rxclk`), and an Atlantic interface clock (`arxclk`). The buffer provides clock decoupling.

Receiver Buffers

The buffer size can be configured to 4, 8, 16, or 32 Kilobytes.

Refer to Table 1–3 on page 1–4 for examples of memory usage depending on buffer size.

128 to 64 Atlantic Adapter
The 128 to 64 Atlantic adapter block is optional, and is used to convert a 128-bit Atlantic slave-source into a 64-bit Atlantic slave-source interface.

Transmitter

The layer 3 transmitter sub-layer accepts packet data from the Atlantic interface, and stores it into its buffers for the layer 1 sub-layer.

The RapidIO Specification requires that newly arrived, higher priority packets be transmitted ahead of the retransmission of previously transmitted, but not acknowledged, (retried) lower priority packets. The Specification also requires that at least one packet, of higher priority than all previously transmitted packets, always be able to pass through. To meet these requirements, the layer 3 transmitter sub-layer includes four transmit queues and four retransmit queues, one for each priority level.

Transmit & Retransmit Queues
As packets are written to the transmitter's Atlantic interface, they are added to the tail end of the appropriate priority transmit queue. The transmitter always transmits the packet at the head of the highest priority non-empty queue. Once transmitted, the packet is moved to the corresponding priority retransmit queue.

When a packet-accepted control symbol is received for the first non-acknowledged transmitted packet, the accepted packet is removed from its retransmit queue.

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

The layer 2 ensures that no more than 7 unacknowledged packets are transmitted, and that the AckIDs are used and acknowledged in incrementing order. The layer 3 removes acknowledged packets from the appropriate retransmit queues when they are acknowledged.

Transmit Buffer
The transmit buffer is the main memory in which the packets are stored. It is partitioned into 64-byte blocks.

Clock & Data
The layer 3 transmitter sub-layer requires two clock domains: an internal global clock (`txclk`), and an Atlantic interface slave clock (`atxclk`). The Atlantic interface provides clock decoupling.

OpenCore Plus Time-Out Behavior

OpenCore® Plus hardware evaluation can support the following two modes of operation:

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

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

For MegaCore functions, the untethered timeout is 1 hour; the tethered timeout value is indefinite.

Your design stops working after the hardware evaluation time expires. After that time, the RapidIO Physical Layer MegaCore function behaves as though its Atlantic interface signals `atxena` and `arxena` are tied low.

As a result, it is impossible for the RapidIO MegaCore function to transmit new packets (it will only transmit idles and status control symbols), or read packets out of the Atlantic interface. If the far end continues to transmit packets, the RapidIO MegaCore function starts refusing new packets by sending `packet_retry` control symbols once its receiver buffer fills up beyond the corresponding threshold.

For more information on OpenCore Plus hardware evaluation using the RapidIO Physical Layer, see *AN 320: OpenCore Plus Evaluation of Megafunctions*.

Parameters

Table 4–4 shows the RapidIO Physical Layer function parameters, which can only be set in IP Toolbench (see "Step 1: Parameterize" on page 2–7).

Table 4–4. Parallel RapidIO Physical Layer Parameters (Part 1 of 2)

Parameter	Value	Description
`Device Family`	Stratix II, Stratix GX, or Stratix	The Stratix II and Stratix GX device families with their faster port speeds (up to 1 Gbps) and dynamic phase alignment features are ideal for the parallel RapidIO MegaCore function. The Stratix device families with its True-LVDS buffers meets the high data rate and low power consumption requirements of the RapidIO standard by using a low-voltage differential signal capable of travelling at rates up to 840 Mbps. Also, the Stratix devices offer a high-speed serial interface (HSSI) combined with serialization/deserialization (SERDES) capability and frequency multiplication all in one circuit.
`Dynamic phase alignment`	Yes/ No	RapidIO specifications require DPA at 750 Mbps and above.
`LVDS data rate` *(1)*	300 Mbps to 1 Gbps	The `LVDS data rate` parameter offers True-LVDS data rates of up to 1 Gbps for 8-bit port widths. Data is clocked on both rising and falling edges, for throughput rates of up to 8 Gbps in each direction.
`Atlantic interface port width` *(2)*	32 or 64 bits	The Atlantic interface packet data path can be configured for 32 or 64 bits. See "Atlantic Interface" on page 4–3 for further details.
`Tx buffer size`	4, 8, 16, or 32 Kbytes	The Tx buffer size parameter allows you to select the transmitter buffer size.
`Rx buffer size`	4, 8, 16, or 32 Kbytes	The Rx buffer size parameter allows you to select the receiver buffer size.

Table 4–4. Parallel RapidIO Physical Layer Parameters (Part 2 of 2)

Parameter	Value	Description
`Receive priority 0/1/2 retry threshold`	Threshold_2 > 9 Threshold_1 > Threshold_2 + 4 Threshold_0 > Threshold_1 + 4 Threshold_0 < (2 ** p_rxbuf_addr_width)	When the number of available free 64-byte blocks in the receive buffer is less than one of these thresholds, the receiver refuses incoming packets of the corresponding priority level by sending packet-retry symbols.

Notes to Table 4–4:

(1) Stratix II or Stratix GX devices and DPA are required for performance above 750 Mbps.

(2) Not all combinations of options are available because of clock frequency and pin constraints on FPGAs.

Signals

Tables 4–5 through 4–12 list the pins used by the parallel RapidIO Physical Layer MegaCore function, with the I/Os shown in Figure 4–1 on page 4–2. The active-low signals are indicated by `_n`.

For signals and bus widths specific to your variation, refer to the HTML file generated by IP Toolbench (see Table 2–1 on page 2–15).

Tables 4–5 through 4–7 list the I/O signals used in the parallel layer 1.

Table 4–5. Parallel Layer 1 Receive Signals

Signal	Direction	Description
`rclk`	Input	Receive clock—free-running input clock for the 8-bit port. `rclk` connects to `tclk` of the transmitting device.
`rd`	Input	Receive data—the receive data is a unidirectional packet data input bus. It is connected to the `td` bus of the transmitting device.
`rframe`	Input	Receive frame—this control signal indicates a special packet framing event on the `rd` pins. `rframe` is sampled with respect to `rclk`.

Table 4–6. Parallel Layer 1 Transmit Signals (Part 1 of 2)

Signal	Direction	Description
`tclk`	Output	Transmit clock—free-running clock for the 8-bit port. `tclk` connects to `rclk` of the receiving device.

Table 4–6. Parallel Layer 1 Transmit Signals (Part 2 of 2)

Signal	Direction	Description
`td`	Output	Transmit data—the transmit data is a unidirectional point-to-point bus designed to transmit the packet information along with the associated `tclk` and `tframe`. The `td` bus of one device is connected to the `rd` bus of the receiving device. `td[0-7]` is always asserted with a fixed relationship to `tclk`, as defined in the AC section.
`tframe`	Output	Transmit framing signal—when issued as active, this signal indicates a packet control event. `tframe` is connected to `rframe` of the receiving device. `tframe` is always asserted with a fixed relationship to `tclk`, as defined in the AC section.

Table 4–7. Parallel Layer 1 Global Signals (Part 1 of 2)

Signal	Direction	Description
`rxclk`	Output	Receive reference clock.
`txclk`	Input	Transmit reference clock.
`rxreset_n`	Input	Receive active-low reset. `rxreset_n` can be asserted asynchronously but should be deasserted on the rising edge of `rxclk`.
`txreset_n`	Input	Transmit active-low reset. `tx_reset_n` can be asserted asynchronously but should be deasserted on the rising edge of `txclk`.
`input_enable`	Input	Enables the inputs. This signal is an input port driven by user logic. The signals driving this input port must be in the `txclk` clock domain. Logically `OR`ed with the register bit `IN_PENA`. The `input_enable` signal is used to enable the input ports directly, removing the need to set up the register bits via the AIRbus interface.
`output_enable`	Input	Enables the outputs. This signal is an input port driven by user logic. The signals driving this input port must be in the `txclk` clock domain. Logically `OR`ed with the register bit `OUT_PENA`. The `output_enable` signal is used to enable the output ports directly, removing the need to set up the register bits via the AIRbus interface.
`train_done`	Output	Indicates that port training has been completed. This signal is pulsed high for one clock period when the last of either the Input port training state machine or the Output port training state machine transitions to the '`OK`' state, as described in *Appendix A of part IV of the RapidIO Specification*.

Table 4–7. Parallel Layer 1 Global Signals (Part 2 of 2)

Signal	Direction	Description
`lvds_rx_pll_areset`	Input	Receive active-high asynchronous reset. Can be asserted or deasserted asynchronously.
`rxpll_locked`	Output	Receive PLL locked.

Table 4–8 shows the DPA signals used by the receiver.

☞ Only applicable when using the DPA circuitry of a Stratix GX or Stratix II device.

Table 4–8. DPA Status Signals (Receiver Only)

Signal	Direction	Description
`rxdpa_locked`	Output	The DPA byte aligner is locked so all channels are aligned.
`rxlvds_ch_locked`	Output	All channels are locked to the DPA mode. This signal is the bit `AND` of the Stratix II `altlvds_rx rx_dpa_locked [number of channels-1..0]` signal.

Table 4–9 lists the I/O signals used in the parallel layer 2.

Table 4–9. Parallel Layer 2 AIRbus Interface Signals Note (1)

Signal	Direction	Description
`sel`	Input	AIRbus interface selects
`addr`	Input	AIRbus address bus
`read`	Input	AIRbus read
`wdata`	Input	AIRbus write data bus
`rdata`	Output	AIRbus read data bus
`dtack`	Output	AIRbus interface data acknowledge

Note to Table 4–9:

(1) Although the AIRbus interface specification lists a clock (`clk`) and an interrupt request (`irq`) as part of its signals, the RapidIO MegaCore function does not have an AIRbus-specific clock, or an `irq` signal.

Tables 4–10 through 4–11 list the I/O signals used in the parallel layer 3..

Table 4–10. Parallel Layer 3 Atlantic Receive Interface Signals

Signal	Direction	Description
`arxclk`	Input	Receive clock
`arxreset_n`	Input	Receive active-low reset. `arxreset_n` can be asserted asynchronously but should be deasserted on the rising edge of `arxclk`.
`arxena`	Input	Receive enable
`arxdav`	Output	Receive data available. The `arxdav` signal is asserted when at least one complete packet is available to be read from the receive buffer. It is deasserted when the receive buffer does not have at least one complete packet available.
`arxdat`	Output	Receive data bus.
`arxval`	Output	Receive data valid.
`arxsop`	Output	Receive start of packet.
`arxeop`	Output	Receive end of packet.
`arxmty`	Output	Number of invalid bytes on the receive data bus.
`arxerr`	Output	Receive data error.
`arxwlevel`*(1)*	Output	Receive buffer level. The number of bits corresponds to the number of free 64-byte blocks in the receive buffer.

Note to Table 4–10:
(1) The following equation: `log2(size of the receive buffer in bytes/64)` determines the number of bits. For example, a receive buffer size of 16 gives: `log2(16×1024/64)= 8 bits`.

Table 4–11. Parallel Layer 3 Atlantic Transmit Interface Signals (Part 1 of 2)

Signal	Direction	Description
`atxclk`	Input	Transmit clock.
`atxreset_n`	Input	Transmit active-low reset. `atxreset_n` can be asserted asynchronously but should be deasserted on the rising edge of `atxclk`.
`atxena`	Input	Transmit enable.
`atxdav`	Output	Transmit data available. `atxdav` is asserted when the transmit buffer has space to accept at least one maximum size packet (i.e., 276 bytes). It is deasserted with it does not have space to accept at least one maximum size packet.
`atxdat`	Input	Transmit data bus.
`atxsop`	Input	Transmit start of packet.
`atxeop`	Input	Transmit end of packet.

Table 4–11. Parallel Layer 3 Atlantic Transmit Interface Signals (Part 2 of 2)

Signal	Direction	Description
`atxmty`	Input	Number of invalid bytes on the transmit data bus.
`atxerr`	Input	Transmit data error.
`atxwlevel` *(1)*	Output	Transmit buffer level. The number of bits corresponds to the number of free 64-byte blocks in the transmit buffer.
`atxovf`	Output	Transmit buffer overflow. If a new packet is started by asserting `atxena` and `atxsop` three or more `atxclk` clock cycles after `atxdav` is deasserted, `atxovf` is asserted and the packet is ignored.

Note to Table 4–11:

(1) The following equation: `log2(size of the transmit buffer in bytes/64)-1` determines the number of bits. For example, a transmit buffer size of 16 gives: `log2(16×1024/64)-1 = 7 bits`.

Table 4–12 shows the packet and error monitoring signals for the parallel RapidIO MegaCore function.

Table 4–12. Packet and Error Monitoring Signals

Signal	Direction	Clock Domain	Description
`packet_transmitted`	Output	`clk`	Pulsed high for one clock cycle when a packet's transmission completes normally.
`packet_cancelled`	Output	`clk`	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 symbol.
`packet_accepted`	Output	`clk`	Pulsed high for one clock cycle when a packet-accepted symbol is being transmitted.
`packet_retry`	Output	`clk`	Pulsed high for one clock cycle when a packet-retry symbol is being transmitted.
`packet_not_accepted`	Output	`clk`	Pulsed high for one clock cycle when a packet-not-accepted symbol is being transmitted.
`packet_crc_error`	Output	`rxclk`	Pulsed high for one clock cycle when a CRC error is detected in a received packet.
`symbol_error`	Output	`rxclk`	Pulsed high for one clock cycle when a corrupted symbol is received.

Table 4–13 shows the register-related signals for the parallel RapidIO MegaCore function.

Table 4–13. Register-Related Signals

Signal	Direction	Description
`ef_ptr[15:0]`	Input	Most significant bits [31:16] of the PHEAD0 register.
`port_response_timeout[23:0]`	Output	Most significant bits [31:8] of PRTCTRL register.

Registers

All addresses access 32-bit registers and are shown as hexadecimal values. The access addresses for each register increment by units of 4. Tables 4–14 through 4–16 show the memory maps for the parallel RapidIO MegaCore function.

Table 4–14. Transmitter Memory Map

Address	Name	Description
`'h10000`	`TXCTRL`	Transmitter Control
`'h10004`	`TXSTAT`	Transmitter Status
`'h10008`	`TXSYM`	Transmitter Symbol

Table 4–15. Receiver Memory Map

Address	Name	Description
`'h10020`	`RXCTRL`	Receiver Control
`'h10024`	`RXSTAT`	Receiver Status
`'h10028`	`RXSYM`	Receiver Symbol

Table 4–16. Master Memory Map

Address	Name	Description
`'h100`	`PHEAD0`	Port Maintenance Block Header 0
`'h104`	`PHEAD1`	Port Maintenance Block Header 1
`'h120`	`PLTCTRL`	Port Link Time-out Control CSR
`'h124`	`PRTCTRL`	Port Response Time-out Control CSR
`'h13C`	`PGCTRL`	Port General Control CSR
`'h158`	`ERRSTAT`	Port 0 Error and Status CSR
`'h15C`	`PCTRL0`	Port 0 Control CSR

Table 4–17 lists the access codes used to describe the type of register bits.

Table 4–17. 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

Transmitter Register Description

Tables 4–18 through 4–20 describe the registers for the transmitter section of the RapidIO MegaCore function. The offset values are as defined by the RapidIO standard.

Table 4–18. TXCTRL—'h10000

Field	Bits	Access	Function	Default
SYM_INS	0	RW	When a transition from 0 to 1 occurs, inserts one symbol into the symbol queue. After insertion, this bit is cleared.	0

Table 4–19. TXSTAT—'h10004

Field	Bits	Access	Function	Default
RSRV	31:0	UR0	Reserved.	0

Table 4–20. TXSYM—'h10008

Field	Bits	Access	Function	Default
`SYMBOL`	15:0	`RW`	16-bit symbol to be inserted into the symbol queue.	0

Receiver Register Description

Tables 4–21 through 4–23 describe the registers for the receiver section of the RapidIO MegaCore function. The offset values are as defined by the RapidIO standard.

Table 4–21. RXCTRL—'h10020

Field	Bits	Access	Function	Default
`SYM_EXT`	0	`RW`	Symbol extraction control.	0

Table 4–22. RXSTAT—'h10024

Field	Bits	Access	Function	Default
`RSRV`	31:0	`UR0`	Reserved.	0

Table 4–23. RXSYM—'h10028

Field	Bits	Access	Function	Default
`SYMBOL`	15:0	`RO`	Extracted symbol. The AIRbus interface does not assert the `dtack` signal until symbol is received.	0

Master Register Description

Tables 4–24 through 4–30 describe the registers for the master functions of the RapidIO MegaCore function. The offset values are as defined by the RapidIO standard.

Table 4–24. PHEAD0—Port Maintenance Block Header 0—'h100

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 input from the `ef_ptr` input signal.	`ef_ptr`
`EF_ID`	15:0	`RO`	Hard wired extended features ID.	`'h0001`

Table 4–25. PHEAD1—Port Maintenance Block Header 1—'h104

Field	Bits	Access	Function	Default
`RSRV`	31:0	`UR0`	Reserved.	0

Table 4–26. PLTCTRL—Port Link Time-out Control CSR—'h120

Field	Bits	Access	Function	Default
`VALUE`	31:8	`RW`	Time-out interval value.	`'hffffff`
`RSRV`	7:0	`UR0`	Reserved.	0

Table 4–27. PRTCTRL—Port Response Time-out Control CSR—'h124

Field	Bits	Access	Function	Default
`VALUE`	31:8	`RW`	Time-out interval value. This value is not used by the RapidIO Physical Layer MegaCore function. The contents of this register are brought out to the `port_response_timeout` output signal.	`'hffffff`
`RSRV`	7:0	`UR0`	Reserved.	0

Table 4–28. PGCTRL—Port General Control CSR—'h13C

Field	Bits	Access	Function	Default
`HOST`	31	`RW`	A host device is a device that is responsible for system exploration, initialization, and maintenance. Agent or slave devices are typically initialized by host devices. `'b0`—agent or slave device `'b1`—host device	0
`ENA`	30	`RW`	The master enable bit controls whether or not a device is allowed to issue requests into the system. If the master enable is not set, the device may only respond to requests. `'b0`–processing element cannot issue requests. `'b1`—processing element can issue requests.	0
`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.	0
`RSRV`	28:0	`UR0`	Reserved.	0

Table 4–29. ERRSTAT—Port 0 Error and Status CSR—'h158

Field	Bits	Access	Function	Default
`RSRV`	31:21	`URO`	Reserved.	0
`OUT_RTY_ENC`	20	`RW1C`	Output port has encountered a retry condition.	0
`OUT_RETRIED`	19	`RO`	Output port has received a packet-retry control symbol and cannot make forward progress.	0
`OUT_RTY_STOP`	18	`RO`	Output port has been stopped due to a retry and is trying to recover.	0
`OUT_ERR_ENC`	17	`RW1C`	Output port has encountered (and possibly recovered from) a transmission error.	0
`OUT_ERR_STOP`	16	`RO`	Output port has been stopped due to a transmission error and is trying to recover.	0
`RSRV1`	15:11	`URO`	Reserved.	0
`IN_RTY_STOP`	10	`RO`	Input port has been stopped due to a retry.	0
`IN_ERR_ENC`	9	`RW1C`	Input port has encountered (and possibly recovered from) a transmission error.	0
`IN_ERR_STOP`	8	`RO`	Input port has been stopped due to a transmission error.	0
`RSRV2`	7:4	`URO`	Reserved.	0
`PORT_PRES`	3	`RO`	The port is receiving the free-running clock on the input port.	0
`PORT_ERR`	2	`RW1C`	Input or output port has encountered an unrecoverable error and has shut down (turned off both port enables).	0
`PORT_OK`	1	`RO`	Input and output ports are initialized and can communicate with the adjacent device.	0
`PORT_UNINIT`	0	`RO`	Input and output ports are not initialized and are in training mode.	`'b1`

Table 4–30. PCTRL0—Port 0 Control CSR—'h15C (Part 1 of 2)

Field	Bits	Access	Function	Default
`OUT_PWIDTH`	31	`RO`	Opening width of the port: `'b0`—8-bit port. This register is always set to 0.	0
`OUT_PENA`	30	`RW`	Output port transmit enable: `'b0`—port is stopped and not enabled to issue any packets, except to respond to I/O logical maintenance packets. `'b1`—port is enabled to issue any packets.	1

Table 4–30. PCTRL0—Port 0 Control CSR—'h15C (Part 2 of 2)

Field	Bits	Access	Function	Default
`OUT_PDRIV_DIS`	29	`RO`	Output port driver disable: `'b0`—output port drivers are turned on, and drive the pins normally. This register is always set to 0.	0
`RSRV`	28	`UR0`	Reserved.	0
`IN_PWIDTH`	27	`RO`	Operating width of the port: `'b0`—8-bit port. This register is always set to 0.	0
`IN_PENA`	26	`RW`	Input port receive enable: `'b0`—port is stopped and only enabled to respond to I/O logical maintenance requests. Other requests return packet-not-accepted control symbols to force an error condition to be signalled by the sending device. `'b1`—port is enabled to respond to any packet.	1
`IN_PRECV_DIS`	25	`RW`	Input port receiver enable: `'b0`—input port receivers are enabled. `'b1`—input port receivers are disabled and are unable to receive any packets or control symbols.	0
`RSRV1`	24	`UR0`	Reserved.	0
`ERR_CHK_DIS`	23	`RW`	This bit disables all RapidIO transmission error checking: `'b0`—Error checking and recovery is enabled. `'b1`—Error checking and recovery is disabled. Device behavior is undefined when error checking and recovery are disabled and an error condition occurs.	0
`MULTICAST`	22	`RW`	Send incoming multicast-event control to this port (multiple port devices only).	0
`RSRV2`	21:1	`UR0`	Reserved.	0
`PORT_TYPE`	0	`UR0`	This indicates the port-type, parallel or serial. `'b0`—Parallel port. `'b1`—Serial port.	0

MegaCore Verification

The parallel RapidIO Physical Layer MegaCore function has been rigorously tested and verified in hardware for different platforms and environments. Each environment has individual test suites, that are designed to cover the following categories:

- Link initialization
- Packet format
- Packet priority
- Flow control

- Stomp error recovery
- Endurance
- Throughput

These test suites contain several testbenches, that are grouped and focused on testing specific features of the parallel RapidIO Physical Layer MegaCore function. These individual testbenches set unique parameters for each specific feature test.

Results of the hardware verification tests are gathered in I-tested reports available for different ASSP devices. Contact your local Altera® sales representative, or field applications engineer (FAE), for further information.

Appendix A. Pin Constraints & Board Design

Pin Constraints

The pinouts for the Stratix® GX, and Stratix device families include dedicated True-LVDS™ clock input pins located on either the RXCLK_IN1n/p bank, or the RXCLK_IN2n/p bank. For the Stratix II device family, the dedicated True-LVDS clock input pins are located on a minimum of two banks. All pins for the receiver must be kept on the same bank. These True-LVDS banks require a clean, filtered power supply.

Board Design Configuration

For detailed board layout guidelines, refer to *AN 224: High-Speed Board Layout Guidelines*, available at **www.altera.com**.

For detailed board layout guidelines in Stratix II devices, refer to the *High-Speed Board Layout Guidelines* and *High-Speed Differential I/O Interfaces with DPA in Stratix II Devices* chapters of the *Stratix II Device Handbook*, available at **www.altera.com**.

For detailed board layout guidelines in Stratix devices, refer to the *High-Speed Differential I/O Interfaces in Stratix Devices* chapter of the *Stratix Device Handbook*, available at **www.altera.com**.

A parallel combination of 0.1, 0.01, and 0.001µF capacitors should be used to decouple the high-speed PLL power and ground planes.

For the output clock (`tdclk`), the user should not use the True-LVDS output clock pins, but should use an appropriate True-LVDS data pair instead. Clock pins can be treated as data pins, because the serializer/deserializer (SERDES) is preloaded with a binary 1010 pattern that guarantees an appropriate skew between the clock and data.

As for True-LVDS traces running at 1 Gbps, the standard board layout guidelines for laying out high-speed True-LVDS traces should apply.

☞ Special attention should be given to status channel lines to ensure setup and hold time requirements are met. Trace lengths should match.

Appendix B. Static & Dynamic Phase Alignment

Static vs Dynamic Alignment

The RapidIO Parallel Physical Layer MegaCore® function implements either static or dynamic alignment. Both follow the same electrical specifications, but differ in their timing requirements.

Static Alignment

In a Stratix® device, the RapidIO parallel physical payer MegaCore function implements static alignment. The reference material for the static alignment portion of this appendix was obtained from the RapidIO™ Trade Association's *Enhancements to the RapidIO AC Specification*, Item 01-05-001.13.

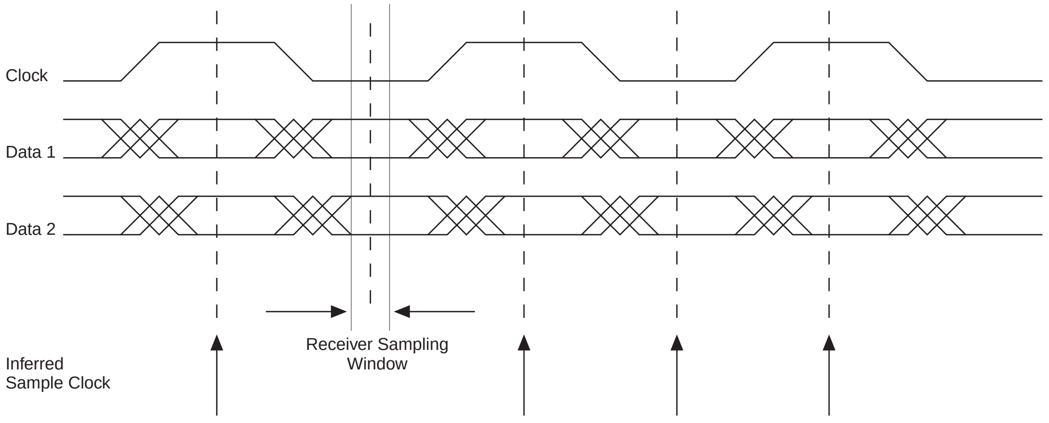
The timing margin for a statically-aligned system is calculated by subtracting all of the delays from the overall period of the clock. These delays include:

- Transmitter clock-to-data skews
- Receiver sampling windows
- Jitter components

The remaining time is allocated to the connection between parts. All of the differential delays between traces—caused by factors such as board routing, transmission line effects, and connector skews—consume this margin of time. Static alignment is appropriate for areas where such factors can be well controlled. For example, the connection between adjacent devices is generally short, and can be controlled—at layout time—to within a few millimetres.

Figure B–1 on page B–2 shows an example of static alignment.

Figure B–1. Static Alignment Timing Diagram



Dynamic Alignment

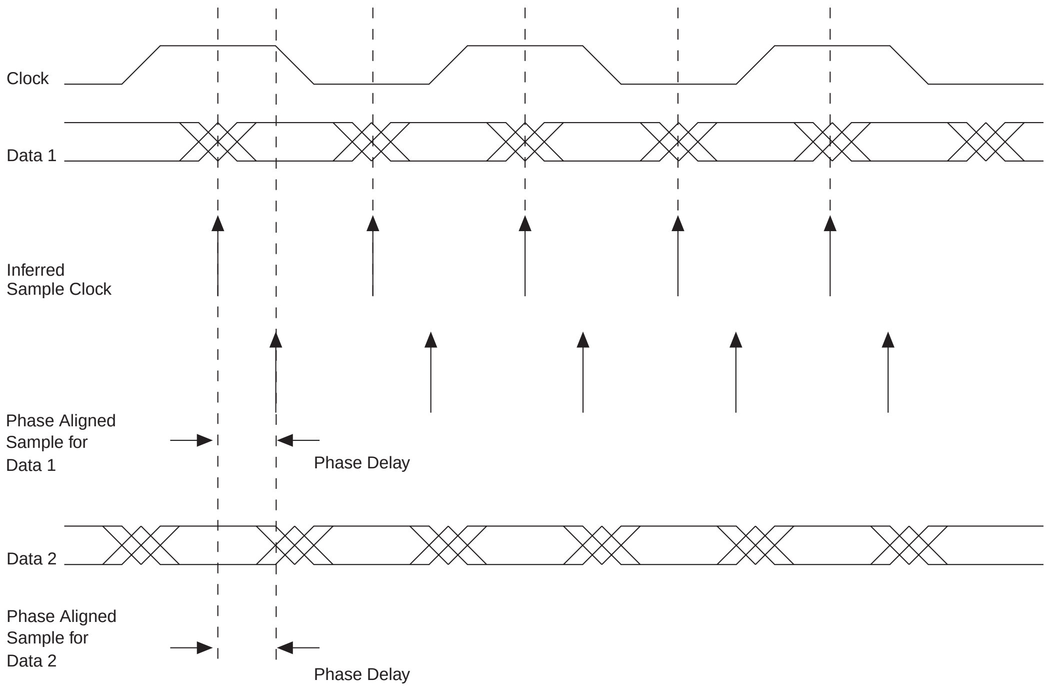
Dynamic alignment allows for greater skew between the inputs. At the receiver, the frequency (hence sampling rate) is known, but the actual phase of the data is not. Each receiver channel looks onto the incoming data and samples at the center of the data eye. Additional logic is required to account for the skews in sampling the data. These skews arise when the data is realigned in time to reconstitute the data as it was originally sent.

The timing margin for a dynamically-aligned system generally excludes the differential skews between data signals. In this case, the timing margin is calculated from the clock frequency by subtracting the receiver sampling window, jitter components, and any sampling errors introduced into the receiver. Transmitter output delays or skews, or interconnection skews can be ignored because the receiver's dynamic phase aligner compensates for them.

Dynamic alignment is appropriate where the skews between signals cannot be controlled. This is common where signals pass through multiple connectors, or where devices can be interchanged. It typically provides a much larger timing margin than static alignment.

Figure B–2 on page B–3 shows an example of dynamic alignment.

Figure B–2. Dynamic Alignment Timing Diagram


Clock
Data 1
Inferred Sample Clock
Phase Aligned Sample for Data 1
Phase Delay
Data 2
Phase Aligned Sample for Data 2
Phase Delay

Altera Solutions

Altera supports both static and dynamic alignment as a system solution.

Static Alignment

The devices in the Stratix series have built-in high-speed interface macros. Using DDR clocking, these macros allow the devices to receive data at rates exceeding 800 Mbps. Built-in logic within the macros is used to present this data to the MegaCore function logic—at lower frequencies—for subsequent protocol processing. A reference sampling clock is used to recover the data. This sampling clock is selected at design time, and cannot be changed when the device is operating. It is possible, however, to compensate for fixed interconnection skews by sampling different receivers on skewed phases of the original receiver reference.

Dynamic Phase Alignment (DPA)

As high-speed interfaces with source-synchronous clocking schemes approach 700 Mbps and beyond, the margin for clock-to-channel and channel-to-channel skew contracts significantly. To stay within the permitted budget, designers must use precise printed circuit board (PCB) design techniques because the slightest mismatch in trace lengths or the use of connectors could result in erroneous data transfer. Additionally, skew inducing effects such as process, voltage, and temperature variations compound the problem, making static phase alignment techniques ineffective.

DPA technology has been developed to address the inadequacies of static alignment methods. The goal of DPA is to allow devices to actively respond to changes in the operational board skew. Devices equipped with DPA continuously check the incoming data and adjust the phase of the clock to align with it. Several industry standards responsible for defining chip-to-chip interfaces, including RapidIO, have recognized the value of DPA, and have included or recommended it in their specifications.

Every Stratix GX and Stratix II receiver channel features an embedded DPA block. For Stratix GX devices, the DPA blocks are located in I/O banks 1 and 2; for Stratix II devices, the DPA blocks are located in banks 1, 2, 5 and 6. A complete, FPGA-integrated hard-silicon DPA solution offers several benefits to system designers. It is implemented for each data channel, such that each channel receives its own phase-adjusted clock. This individual alignment for each channel minimizes the chance for errors introduced by mismatches in signal propagation paths. Also, it does not require a training mode; rather, it continuously realigns the clock to the data during device operation.

The RapidIO MegaCore function also features an integrated DPA block on its receiving path, between the high-speed serial bus and the parallel bus. The RapidIO DPA block functions include: data deserialization and clock division, dynamic phase alignment, and byte alignment.

The RapidIO DPA block makes use of the DPA capability of Stratix II or Stratix GX devices, supporting data rates of up to 1 Gbps. The DPA block is configured to support 9 hi-speed channels, and internal data widths of 32 or 64 bits.

For more information on the use of DPA in the RapidIO Physical Layer MegaCore function, see “Dynamic Phase Alignment” on page 4–7.

For more information on using dynamic phase alignment, refer to the following documents:

- *High-Speed Differential I/O Interfaces with DPA in Stratix II Devices* chapter of the *Stratix II Device Handbook*
- *AN 236: Using Source-Synchronous Signaling with DPA in Stratix GX Devices*

- *The Need for Dynamic Phase Alignment in High-Speed FPGAs White Paper*
- *Advantages of the Embedded DPA Circuitry in Stratix GX Devices White Paper*

AC Timing Analysis

Specifications for this interface allow two sets of timing relationships between the sender and receiver: static and dynamic mode. In the static alignment mode, all data obeys a common set of timing parameters (e.g. set up and hold times with respect to a sampling clock). In the dynamic alignment mode, a per-bit timing relationship applies.

This section describes the timing analysis for various configurations and components. These timing components are referenced to Figure B–3 on page B–5. This figure shows the timing path as related to the paths followed by the clock and data signals through the user's system. Figure B–4 on page B–6 references the timing values to the clock and data edges.

Figure B–3. Timing Analysis Model

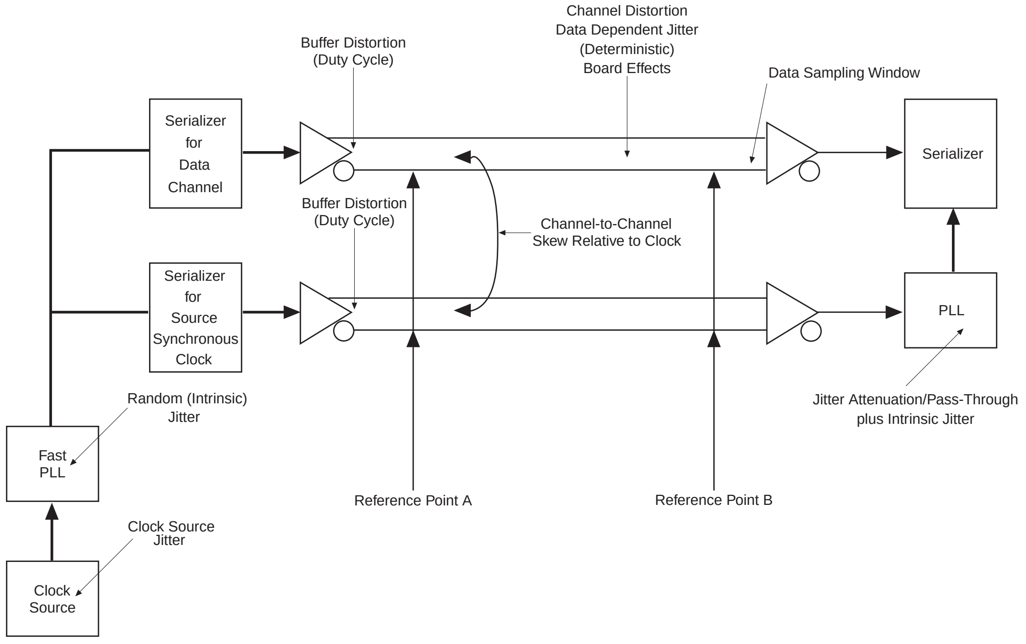
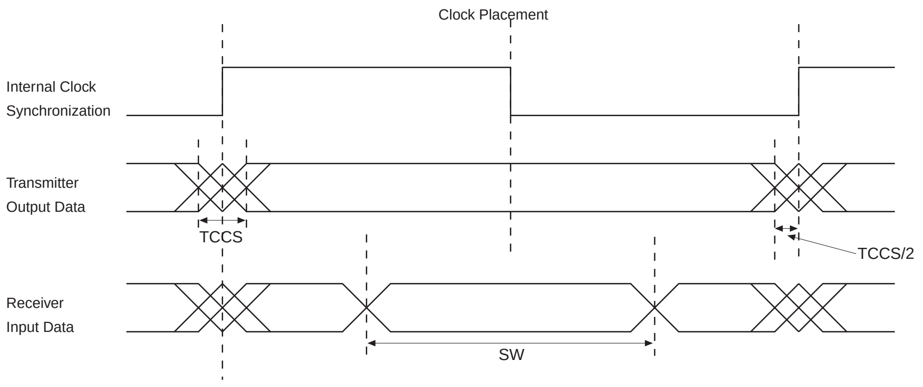


Figure B–4. Timing Diagram



For further timing information on the RapidIO interface, refer to the *RapidIO Trade Association, RapidIO™ Interconnect Specification, Revision 1.2, June 2002.*

- *Parallel—Part IV: Physical Layer 8/16 LP-LVDS Specification*
- *Serial—Part VI: Physical Layer 1×/4× LP Serial Specification*

Appendix C. Compliance

Compliance Statement

This appendix states the items of the RapidIO™ specifications with which the Altera® RapidIO Physical Layer MegaCore® function is non-compliant. Most of these non-compliance statements are based on the *RapidIO™ 8/16 LP-LVDS Device Compliance Checklists, rev 0.4, 3/2001.*

☞ Unless otherwise specified in this appendix, the RapidIO Physical Layer MegaCore function is compliant with the RapidIO specifications.

General Compliance (Physical Layer)

Tables C–1 through C–4 summarize the checklist's requirements, and provide commentary regarding the MegaCore function's non-compliance..

Table C–1. General Specification Compliance (Section 1.1.1)

Item Number	Compliance Item Summary	Comment
1	All device generated packets must comply to logical, transport and physical layer specifications.	The MegaCore function only implements physical layer specifications
9 – 25	CAR/CSR implementation	The CAR and CSR registers are implemented as local registers, and are not available from the RapidIO interface
14	Reads to reserved CAR bits return logic 0s when read.	The values of undefined registers are not guaranteed.
22	Reads to reserved CSR and Extended Feature bits return logic 0s when read.	The values of undefined registers are not guaranteed.

Table C–2. Basic Functionality List (Section 1.1.2.2) (Part 1 of 2)

Item Number	Compliance Item Summary	Comment
2	Free running `clk` output	The clock is not available when the device is reset, or not configured.
6	AC specifications	To be determined based on device characterization. See "AC Timing Analysis" on page B–5 for Stratix® timing information.

Table C–2. Basic Functionality List (Section 1.1.2.2) (Part 2 of 2)

Item Number	Compliance Item Summary	Comment
9	16 bit ports can act as 8 bit ports	The MegaCore function no longer supports 16 bits, and therefore no longer supports the port width downgrade feature.
13	Assignment of read and write functions to priorities	This is a logical level function, and is not implemented.
14	Physical layer register set	These registers are implemented as local registers, and are not available from the RapidIO interface.
14E	Port General Control CSR	Bit 2 as defined in the specification is not implemented, and the output bits are always enabled. This is an FPGA device limitation. Bit 9 as defined in the specification has a slightly different meaning, See Table 3–19 on page 3–30 and Table 4–28 on page 4–33 for details.
15	System Initialization	The MegaCore function only implements physical layer specifications
N/A *(1)*	Port Link Time-Out Control CSR According to the RapidIO specification, the reset value represents between three (3) and six (6) seconds for serial, and three (3) and five (5) seconds for parallel.	The RapidIO Physical Layer MegaCore function appends eight zero bits to the least significant bit (LSB) of the register value. It forms a 32-bit counter that runs at the internal clock frequency, thus the maximum timer interval is much longer than the required six (6) seconds stated in the RapidIO specification.

Note to Table C–2:

(1) This item is not listed in the RapidIO Compliance Checklist, it is in non-compliance with the RapidIO Specifications.

Table C–3. Packet Transmission List (Section 1.1.2.4)

Item Number	Compliance Item Summary	Comment
9	Switch preserves error coverage	The MegaCore function only implements physical layer specifications

Table C–4. Packet Reception List (Section 1.1.2.5)

Item Number	Compliance Item Summary	Comment
9F	TOD-sync symbol	This symbol is not processed. It is received and can be flagged through an interrupt to the local microprocessor. If you are not using the broadcast feature, non-compliance should not be an issue.

General Compliance (Transport and Logical Layers)

☞ Since the RapidIO Physical Layer MegaCore function does not implement the transport or logical layers, it is not compliant with sections 1.1.3, 1.1.4, 1.3, 1.4, or 2.0.