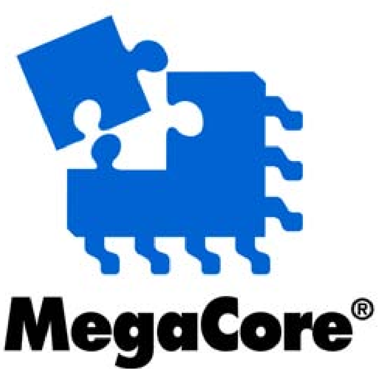


SerialLite

MegaCore Function User Guide

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

MegaCore Function Version: 1.1.0
Document Version: 1.1.0 rev. 1
Document Date: August 2005

Downloaded from Elcodis.com electronic components distributor

Copyright © 2005 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

nsai

I.S. EN ISO 9001

UG-SERIALLT-1.1

Downloaded from Elcodis.com electronic components distributor

Contents

About This User Guide v
Revision History v
How to Contact Altera v
Typographic Conventions vi

Chapter 1. About This MegaCore Function
Release Information 1–1
Device Family Support 1–1
New in Version 1.1.0 1–2
Features 1–2
- OpenCore Plus Evaluation 1–3

Performance 1–4

Chapter 2. Getting Started
System Requirements 2–1
Design Flow 2–1
Obtain & Install the SerialLite MegaCore Function 2–2
- Download the SerialLite MegaCore Function 2–2
- Install the SerialLite MegaCore Function Files 2–3
- Directory Structure 2–4

SerialLite MegaCore Function Walkthrough 2–5
- Create a New Quartus II Project 2–5
- Launch IP Toolbench 2–6
- Step 1: Parameterize 2–7
- Step 2: Set Up Simulation 2–13
- Step 3: Generate 2–15

Simulate the Design 2–17
Compile the Design 2–18
- Apply Constraints 2–18

Set Up Licensing 2–19
- Append the License to Your license.dat File 2–19
- Specify the License File in the Quartus II Software 2–20

Chapter 3. Specifications
Functional Description 3–1
- OpenCore Plus Time-Out Behavior 3–1

SerialLite Link Configuration 3–1
- Link Consistency 3–3
- Interface Overview 3–3
- Achieving the Desired Bandwidth 3–13

Downloaded from Elcodis.com electronic components distributor

Clock Compensation 3–19
Lane Polarity & Order Reversal 3–25
Choosing Ports 3–27
Streaming & Packet Data 3–29
Packet Sizes 3–30
Channel Multiplexing 3–33
Data Integrity Protection: CRC 3–36
Retry on Error 3–38
Flow Control 3–42
The Receive FIFO Buffers 3–45
Error Handling 3–54
Transceiver Settings 3–58
Optimizing the Implementation 3–66
Initialization & Restart 3–70

Chapter 4. SerialLite Testbench

General Description 4–1
Testbench Environment 4–1
Methodology Overview 4–2
Configuring the Simulation 4–3
ModelSim Simulator 4–3
Other Simulators 4–5
Sending & Receiving Data Tasks 4–8
User Packet Data 4–11
Running a Simulation 4–13
Simulation Pass & Fail Conditions 4–14

Downloaded from Elcodis.com electronic components distributor

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	August 2005	1.1.0	● Updated the release information. ● Updated the performance information.
2	August 2005	1.1.0	● Updated the system requirements. ● Updated Table 2-1. IP Toolbench-Generated Files. ● Added a reference to the Quartus® II Design Space Explorer optimization utility.
3	August 2005	1.1.0	● Corrected the THDAV signal description in Table 3–2. ● Added a reference to the SerialLite Bandwidth Calculator. ● Added the Clock Pad Restrictions section. ● Updated the description of the STATUS PORT[15..5] signal in Table 3–41. ● Added a description of the serial loopback (RX_SLPBK) signal to Table 3–54. ● Corrected the Initialization & Restart section.
4	August 2005	1.1.0	● Modified the "Configuring the Simulation" section to correct the testbench parameter names.
All	September 2004	1.0.0	First version of user guide.

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

Downloaded from Elcodis.com electronic components distributor

Information Type	USA & Canada	All Other Locations
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

Typographic Conventions

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.
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.

Downloaded from Elcodis.com electronic components distributor

Visual Cue	Meaning
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.

Downloaded from Elcodis.com electronic components distributor

Downloaded from Elcodis.com electronic components distributor

1. About This MegaCore Function

Release Information

Table 1–1 provides information about this release of the Altera® SerialLite MegaCore® function.

Table 1–1. SerialLite Release Information

Item	Description
Version	1.1.0
Release Date	August 2005
Ordering Code	IP-SERIALLITE
Product ID	00A6
Vendor ID	6AF7

Device Family Support

MegaCore functions provide either full or preliminary support for target Altera device families:

- 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 SerialLite MegaCore function to each Altera device family.

Table 1–2. Device Family Support

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

Downloaded from Elcodis.com electronic components distributor

Introduction

The SerialLite MegaCore function is a simple, high-speed, low-latency, low-resource, point-to-point serial data communication link. It implements the full SerialLite protocol with performance up to 3.125 Gbps across the serial data communication link. It is highly configurable, providing a wide range of functionality suited to moving data in many different environments.

New in Version 1.1.0

- Maintenance release

For details, refer to the *SerialLite MegaCore Function Errata Sheet* v1.0.0 and v1.1.0 available at **www.altera.com/literature/lit-es.jsp.**

Features

- 500 Mbps to 3.125 Gbps per lane
- Supports up to 16 lanes
- Support for two user packet types: data packet and priority packet
- Nesting of time-critical priority packets within regular data packets
- Unrestricted data packet size
- Priority packet size up to 256 bytes
- Optional lane polarity reversal
- Optional lane order reversal
- Optional packet integrity protection using cyclic redundancy code CRC-32 or CRC-16
- Optional priority packet retry-on-error
- Optional flow control
- Automatic handling of idles
- Synchronous or asynchronous operation
- Automatic clock rate compensation for asynchronous use
  - 100 and 300 parts per million (ppm)
- Error detection
- Atlantic™ interface compliant
- 8B/10B physical layer encoding and decoding
- Electricals based on familiar XAUI standard
- Low protocol overhead and logic usage
- Low point-to-point transfer latency
- No inter-frame gaps required
- 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

Downloaded from Elcodis.com electronic components distributor

General Description

The SerialLite MegaCore function provides a simple and lightweight way to move data from one point to another reliably at high speeds. It consists of a serial link of up to 16 bonded lanes, with logic to provide a number of basic and optional link support functions. The Atlantic interface is used as the primary access for delivering and receiving data.

The SerialLite protocol specifies a link that is simple to build, uses as little logic as possible, and requires little work for a logic designer to utilize. All of the features available in the SerialLite protocol have been implemented in the SerialLite MegaCore function and are parameterizable through a powerful MegaWizard® Plug-In Manager interface.

A link built using the SerialLite MegaCore function can operate at speeds from 500 Mbps to 3.125 Gbps. Link reliability is enhanced by the 8B/10B encoding scheme and optional cyclic redundancy code (CRC) capabilities. Further reductions in the bit-error rate can be achieved using the optional retry-on-error feature. Data rate and consumption mismatches can be accommodated using the optional clock-compensation and flow-control features to ensure that no data is lost.

The combination of optional capabilities makes the link well-suited to a wide variety of applications. It is intended to support chip-to-chip, board-to-board, and cross-backplane data transfers.

OpenCore Plus Evaluation

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

- Simulate the behavior of a MegaCore function within your system
- Verify the functionality of your design, as well as quickly and easily evaluate its size and speed
- 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 SerialLite MegaCore function, see "OpenCore Plus Time-Out Behavior" on page 3–1 and *AN 320: OpenCore Plus Evaluation of Megafunctions*.

Downloaded from Elcodis.com electronic components distributor

Performance

Table 1–3 shows typical expected performance for different parameters, using the Quartus® II software, version 5.0 Service Pack 1.

Table 1–3. Performance

Parameters							Logic Elements	Memory		f_{MAX} (MHz)
Number of Lanes	Regular Port	Priority Port	Mode	CRC	Flow Control	Retry on Error		M512	M4K	
1	✓	–	Streaming	–	–	–	795	2	1	>156
4	✓	–	Streaming	–	–	–	2,088	12	8	>156
1	✓	✓	Packet	CRC-16	✓	✓	2,319	5	15	>156
1	✓	–	Packet	CRC-16	–	–	1,072	2	3	>156
1	–	✓	Packet	CRC-16	✓	✓	1,878	3	12	>156
1	✓	✓	Packet	CRC-32	–	–	3,189	5	10	>156
1	✓	✓	Packet	CRC-32	✓	✓	4,117	7	16	>156
2	✓	✓	Packet	CRC-32	–	–	5,443	21	10	>156
2	✓	✓	Packet	CRC-32	✓	✓	6,452	19	18	>156
4	✓	✓	Packet	–	✓	–	3,686	27	17	>156
4	✓	✓	Packet	–	✓	✓	4,653	24	31	>156
4	✓	✓	Packet	CRC-16	✓	✓	5,196	26	32	>156
4	✓	–	Packet	CRC-16	–	–	2,880	22	5	>156
4	–	✓	Packet	CRC-16	✓	✓	4,050	20	25	>156
4	✓	✓	Packet	CRC-32	✓	✓	10,809	30	33	>156
4	–	✓	Packet	CRC-32	✓	✓	7,143	24	26	>156
4	✓	–	Packet	CRC-32	✓	–	6,350	24	7	>156
4	✓	✓	Packet	CRC-32	–	✓	10,703	30	33	139
4	✓	✓	Packet	CRC-32	–	–	9,560	36	17	>156
7	✓	✓	Packet	–	✓	✓	6,827	34	53	145
7	✓	✓	Packet	CRC-32	✓	✓	18,471	50	53	139
16	✓	✓	Packet	–	✓	✓	13,138	32	131	128
16	✓	✓	Packet	CRC-32	✓	✓	38,529	96	117	105

Downloaded from Elcodis.com electronic components distributor

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 Service Pack 1 or higher
- ModelSim® version 6.0c or higher for running the testbench
- Verilog 2000 support
- 2 GB of RAM highly recommended

Design Flow

The design flow to evaluate the SerialLite MegaCore® function using the OpenCore® Plus feature involves the following steps:

1. Obtain and install the SerialLite MegaCore function.

2. Create a custom variation of the SerialLite 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.

Downloaded from Elcodis.com electronic components distributor

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 SerialLite MegaCore function.

Once you have purchased a license for the SerialLite 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. Perform design verification.

Obtain & Install the SerialLite 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 SerialLite MegaCore function. If you have already installed the SerialLite MegaCore function from the MegaCore IP Library CD-ROM, skip to "Directory Structure" on page 2–4.

Download the SerialLite 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 SerialLite 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 `SerialLite` in the **IP MegaSearch** box.
3. Click **Go**.
4. Choose **SerialLite** from the search results page. The product description web page displays.

Downloaded from Elcodis.com electronic components distributor

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 SerialLite 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 SerialLite MegaCore Function Files

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

Windows

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

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

2. Type <*path name*>`\slite-v1.1.0.exe`, where <*path name*> is the location of the downloaded MegaCore function.

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

Solaris & Linux

Follow these steps to install the SerialLite 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.

Downloaded from Elcodis.com electronic components distributor

2. Decompress the package by typing the following command:

```
gzip -d slite-v1.1.0_linux.tar.gz↵
```

or

```
gzip -d slite-v1.1.0_solaris.tar.gz↵
```

3. Extract the package by typing the following command:

```
tar -xvf slite-v1.1.0_linux.tar↵
```

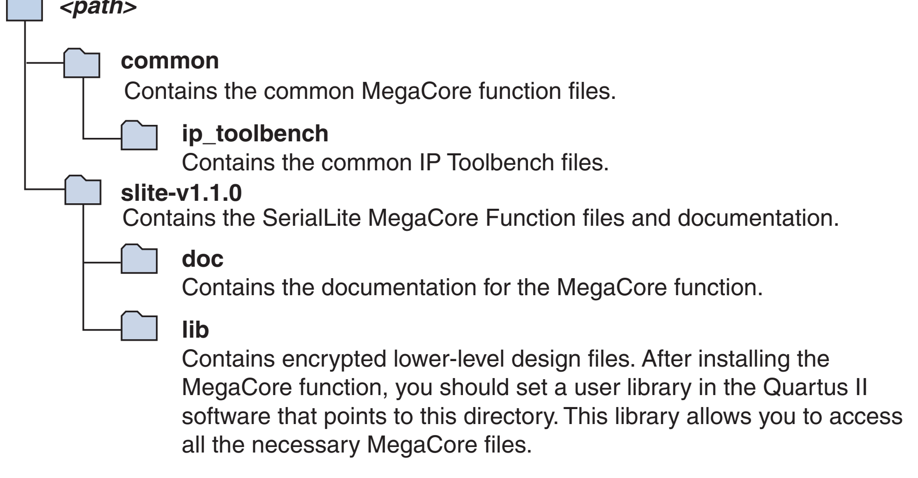
or

```
tar -xvf slite-v1.1.0_solaris.tar↵
```

Directory Structure

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

Figure 2–1. SerialLite MegaCore Function Directory Structure



Downloaded from Elcodis.com electronic components distributor

SerialLite MegaCore Function Walkthrough

This walkthrough explains how to create a SerialLite MegaCore function variation using the Altera SerialLite IP Toolbench and the Quartus II software on a PC. When you are finished generating a SerialLite MegaCore function variation, 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 2–13
- "Step 3: Generate" on page 2–15

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 also specify the SerialLite 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 does 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**.

Downloaded from Elcodis.com electronic components distributor

8. Type <*path*>`\slite-v1.1.0\lib\` into the **Library name** box, where <*path*> is the directory in which you installed the SerialLite MegaCore function. The default installation directory is **c:\MegaCore**.

9. Click **Add**.

10. Click **OK**.

11. Click **Next**.

12. Choose the Stratix® GX 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 megafunction variation and click **Next**.

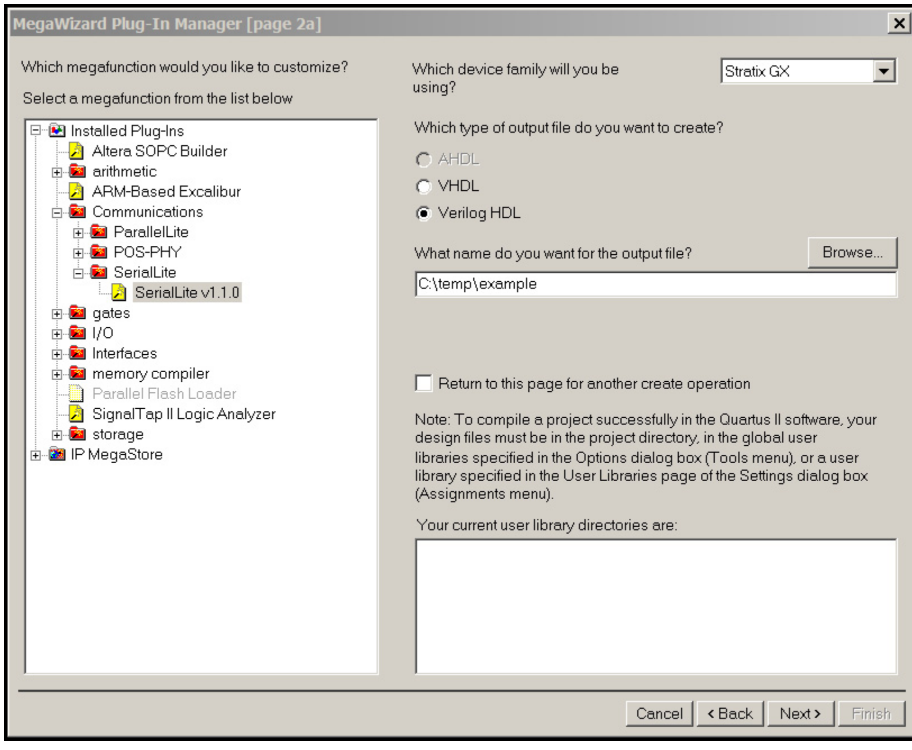
3. Choose **SerialLite v1.1.0** in the **Communications > SerialLite** directory.

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.

Downloaded from Elcodis.com electronic components distributor

Figure 2–2. Select the MegaCore Function



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

Step 1: Parameterize

There are four pages in the Parameterize - SerialLite MegaCore function wizard available for parameterizing your link. All of the settings for these pages are detailed in Chapter 3, Specifications.

No changes to the default configuration are made in this walkthrough.

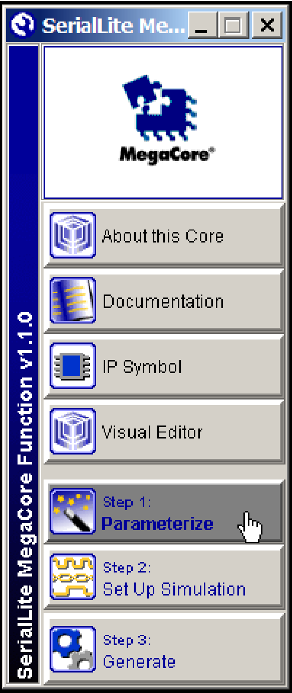
You move from page to page either by clicking the tabs at the top of the pages or by navigating using the First, Previous, Next, and Last buttons. The First button takes you to the Basic Configuration page, the Previous button takes you to the page whose tab is to the left of the current page, the Next button takes you to the page whose tab is to the right of the current page, and the Last button takes you to the Advanced Electricals page. You can make changes in any order, but if you move through the pages in the order indicated, no setting changes are required for you to revisit a page you have already completed.

To parameterize your MegaCore function, follow these steps:

Downloaded from Elcodis.com electronic components distributor

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

Figure 2–3. IP Toolbench—Parameterize



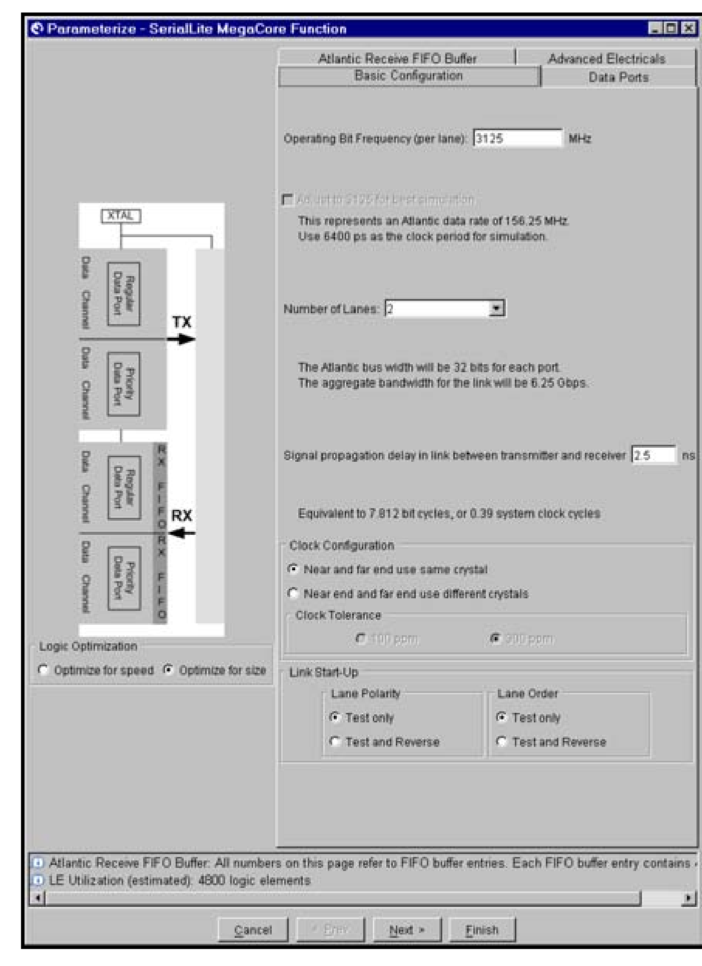
1. This brings up the Basic Configuration page (see Figure 2–4 on page 2–9).

   The Basic Configuration page allows you to configure the general characteristics of the link. Included on this page are settings for the following:

   - Bit rate
   - Lane count
   - Signal propagation delay
   - Clock configuration
   - Lane polarity reversal
   - Lane order reversal

Downloaded from Elcodis.com electronic components distributor

Figure 2–4. SerialLite MegaCore Function Basic Configuration Page



2. After you choose your settings on the Basic Configuration page, click **Next** to go to the Data Ports page.

   This page allows you to select and configure the data ports, as shown in Figure 2–5 on page 2–10. You can select the regular data port and the priority data port. For the regular data port, you can set the following:

   - Data mode (packet or streaming)
   - Channel multiplexing options
   - Cyclic redundancy code (CRC) options

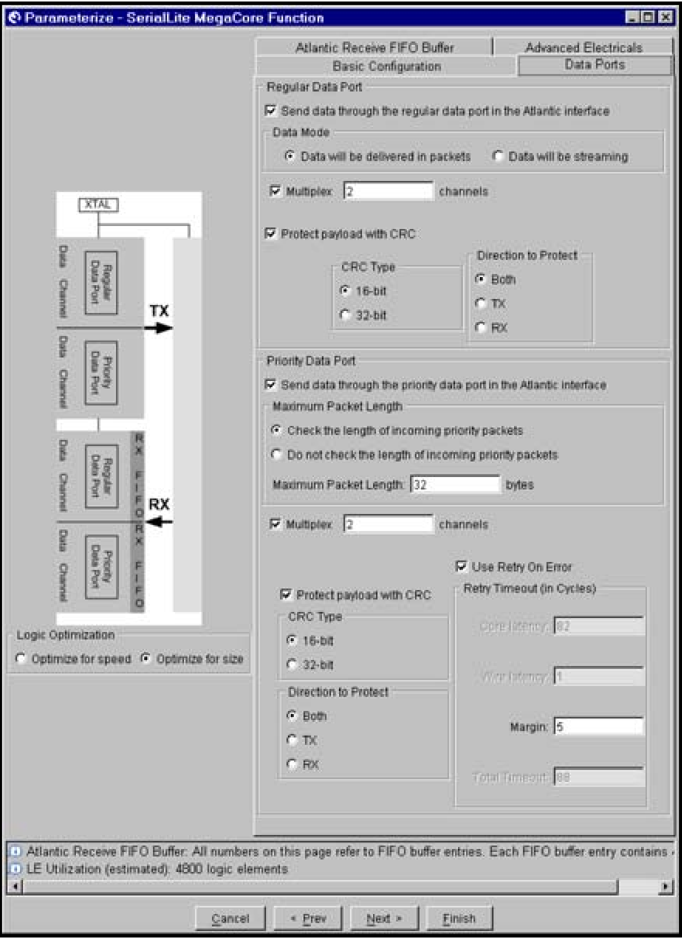
   For the priority port, you can set the following:

   - Maximum packet length
   - Channel multiplexing options

Downloaded from Elcodis.com electronic components distributor

- CRC options
- Retry-on-error options

Figure 2–5. SerialLite MegaCore Function Data Ports Page

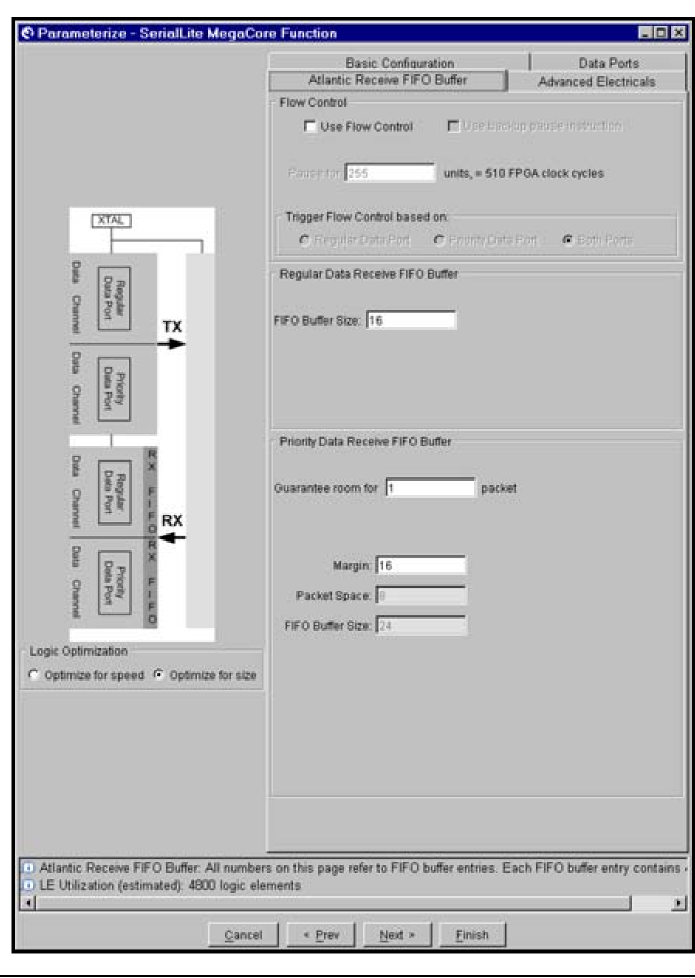


3. After you choose your settings on the Data Ports page, click **Next** to go to the Atlantic Receive FIFO Buffer page.

   On this page you can configure flow control and the receive FIFO buffer sizes. The options on this page depend heavily on whether or not flow control is enabled. Figure 2–6 on page 2–11 shows the options with flow control disabled. The default configuration, used in this walkthrough, uses no flow control.

Downloaded from Elcodis.com electronic components distributor

Figure 2–6. SerialLite MegaCore Function Atlantic Receive FIFO Buffer Page (Flow Control Disabled)

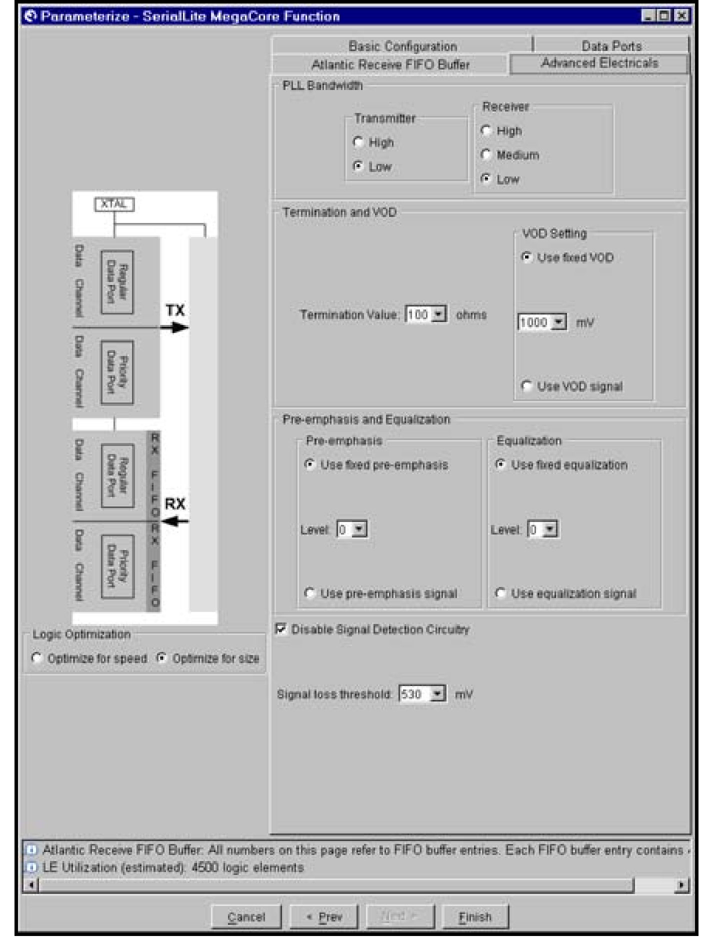


4. After you choose your settings on the Atlantic Receive FIFO Buffer page, click **Next** to go to the Advanced Electricals page. This page, shown in Figure 2–7 on page 2–12, allows you to configure electrical settings for the ALTGXB transceivers found in the Stratix GX devices. You can set the following characteristics:

   - Transmitter phase-locked loop (PLL) bandwidth
   - Receiver PLL bandwidth
   - Transmitter termination
   - Output differential voltage (V_{OD})
   - Pre-emphasis
   - Equalization
   - Signal loss behavior

Downloaded from Elcodis.com electronic components distributor

Figure 2–7. SerialLite MegaCore Function Advanced Electricals Page



You can also bias the design to optimize for speed or size. This selection is available independently of the four pages.

The wizard gives you an estimate of the logic used. This is only an estimate, and depends on what else is being instantiated in the Stratix GX device. For an accurate count of resources utilized, you must synthesize the design.

5. Continue with the default configuration and click **Finish**.

Downloaded from Elcodis.com electronic components distributor

Step 2: Set Up Simulation

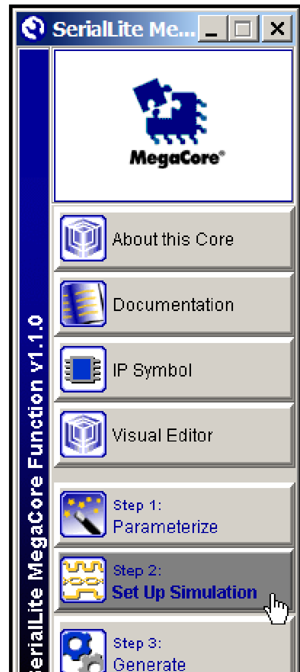
An IP functional simulation model is a cycle-accurate VHDL or Verilog HDL model file produced by the Quartus II software. It allows for fast functional simulation of IP using industry-standard VHDL and Verilog HDL simulators.

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 2–8).

Figure 2–8. IP Toolbench—Set Up Simulation



2. Turn on **Generate Simulation Model** (see Figure 2–9 on page 2–14).

Downloaded from Elcodis.com electronic components distributor

Figure 2–9. Generate Simulation Model

Set Up Simulation - SerialLite MegaCore Function

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.

Cancel OK

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.

   ☞ If VHDL is selected, your simulation environment must support mixed-language simulation to use the SerialLite testbench. If it does not, generate a second simulation model using Verilog HDL for use with the SerialLite testbench by repeating step 2.

4. Click **OK**.

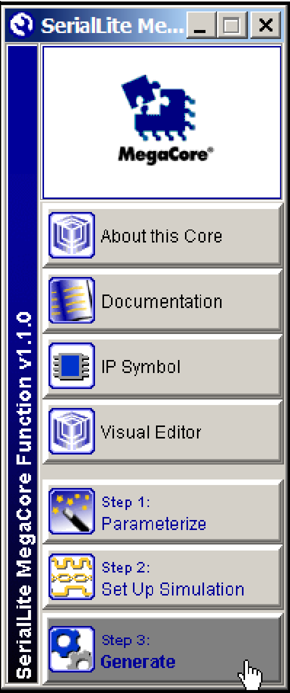
Downloaded from Elcodis.com electronic components distributor

Step 3: Generate

To generate your MegaCore function variation, follow these steps:

1. Click **Step 3: Generate** in IP Toolbench (see Figure 2–10).

Figure 2–10. IP Toolbench—Generate



2. The generation report lists the design files that IP Toolbench creates (see Figure 2–11 on page 2–16). Click **Exit**.

Downloaded from Elcodis.com electronic components distributor

Figure 2–11. Generation Report

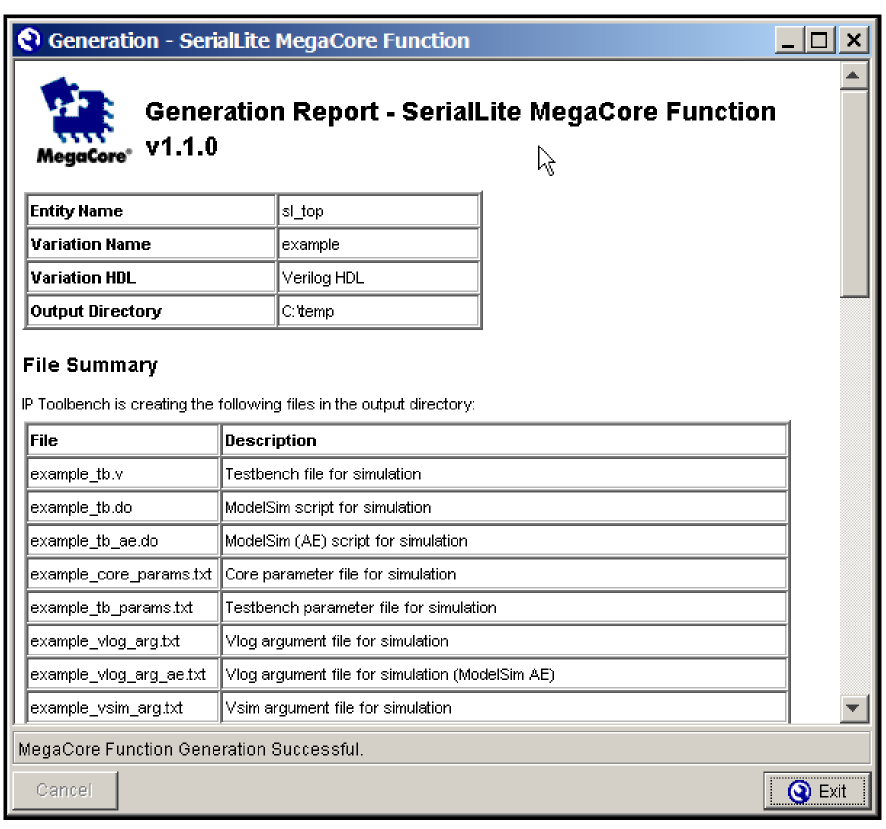


Table 2–1 describes IP Toolbench-generated files

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

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

Extension	Description
.vhd or **.v**	A MegaCore function variation file, which defines a VHDL or Verilog HDL top-level description of the MegaCore function variation. Instantiate the entity defined by this file inside of your design. Include this file when compiling your design in the Quartus II software.
.cmp	A VHDL component declaration file for the MegaCore function variation. Add the contents of this file to any VHDL architecture that instantiates the MegaCore function.
_bb.v	Verilog HDL black-box file for the MegaCore function variation. Use this file when using a third-party EDA tool to synthesize your design.
.bsf	Quartus II symbol file for the MegaCore function variation. You can use this file in the Quartus II block diagram editor.
.html	MegaCore function report file.

Downloaded from Elcodis.com electronic components distributor

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

Extension	Description
_sl_core.vo or **_sl_core.vho**	VHDL or Verilog HDL IP functional simulation model.
_inst.vhd or **_inst.v**	VHDL or Verilog HDL sample instantiation file.
_core_params.txt	SerialLite MegaCore function configuration information for use by the testbench.
_constraints.tcl	Tcl script for applying virtual pin constraints when compiling the SerialLite MegaCore function by itself.
_tb.do	SerialLite testbench script. Used for simulating a SerialLite MegaCore function variation using the SerialLite testbench in the Model Technology ModelSim (standard edition) simulation tool.
_tb_ae.do	SerialLite testbench script. Used for simulating a SerialLite MegaCore function variation using the SerialLite testbench in the Model Technology ModelSim-Altera simulation tool.
_tb.v	SerialLite testbench top-level file. Used when simulating a SerialLite MegaCore function variation using the SerialLite testbench.
_vsim_arg.txt	ModelSim simulation arguments file. Used when simulating a SerialLite MegaCore function variation using the SerialLite testbench.
_vlog_arg.txt	ModelSim compilation arguments file. Used when simulating a SerialLite MegaCore function variation using the SerialLite testbench.
_tb_params.txt	SerialLite testbench parameter file. You change parameter values in this file to control testbench behavior.
.inc	An AHDL `include` declaration file for the MegaCore function variation. Include this file with any AHDL architecture that instantiates the MegaCore function.

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

Simulate the Design

You can simulate your design using IP Toolbench-generated VHDL and Verilog HDL IP 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 configurable testbench for use in evaluating the SerialLite MegaCore function. The testbench is described in detail in Chapter 4, SerialLite Testbench.

Downloaded from Elcodis.com electronic components distributor

Compile the Design

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

Apply Constraints

If you are compiling the SerialLite MegaCore function variation by itself, the pins must be declared as virtual pins. A Tcl script handles that for you.

1. From the Tools menu, select **Tcl Scripts** to bring up the script browser.

2. In the project directory, select <*design name*>**_constraints.tcl**.

3. Click **Run**.

The Tcl script also adds several timing constraints and fitter guide settings that typically produce the best f_{MAX}. Use this script as a guide when setting constraints for the SerialLite MegaCore function variation when implementing an actual design. The timing constraints are currently for the SerialLite MegaCore function variation by itself, and must be updated with hierarchy information in your own design.

The fitter guide settings may cause conflicts with your Quartus II software settings. These following guide settings are used:

- `STRATIX_OPTIMIZATION_TECHNIQUE SPEED`
- `AUTO_PACKED_REGISTERS_STRATIX OFF`
- `MUX_RESTRUCTURE OFF`
- `STATE_MACHINE_PROCESSING AUTO`
- `FITTER_EFFORT "STANDARD FIT"`

If, after using the constraints Tcl script and the fitter guide settings, your design still does not meet timing, try using the Quartus II Design Space Explorer optimization utility.

For more information see the *Design Space Explorer* chapter in Volume 2 of the *Quartus II Handbook*.

You can now integrate your MegaCore function variation into your design and simulate and compile.

Downloaded from Elcodis.com electronic components distributor

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 SerialLite 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 SerialLite 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 SerialLite MegaCore Function, see "OpenCore Plus Time-Out Behavior" on page 3–1 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 the SerialLite 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 SerialLite 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

Downloaded from Elcodis.com electronic components distributor

- Synplify software
- ModelSim simulator

2. Open the SerialLite 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 SerialLite 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, for example, <*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.

Downloaded from Elcodis.com electronic components distributor

3. Specifications

Functional Description

The SerialLite MegaCore® function consists of parameterized logic and a parameterized testbench. The following sections detail the various possible configurations and things you should consider when deciding how to configure the link.

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. The `TENA` and `THENA` signals ignore attempts to write to the SerialLite MegaCore function, and the `RENA` and `RHENA` signals ignore attempts to read from the SerialLite MegaCore function.

For more information on OpenCore Plus hardware evaluation, see "OpenCore Plus Evaluation" on page 1–3 and *AN 320: OpenCore Plus Evaluation of Megafunctions* on **www.altera.com**.

SerialLite Link Configuration

The general decisions you must make for your SerialLite MegaCore function are:

- High-level link configuration
- Bandwidth required
- Which port(s) to use
- Whether to use packet or streaming data
- Whether to multiplex multiple channels

Downloaded from Elcodis.com electronic components distributor

- Whether to use CRC
- Whether to implement the retry-on-error feature
- Whether to implement flow control
- How to size the receive FIFO buffers
- Electrical characteristics of the Stratix® GX transceivers

IP Toolbench provides a fully functional default SerialLite MegaCore function variation ready for instantiation. The result is a link with the characteristics shown in Table 3–1. The following sections describe all of these features.

Table 3–1. Default SerialLite Link

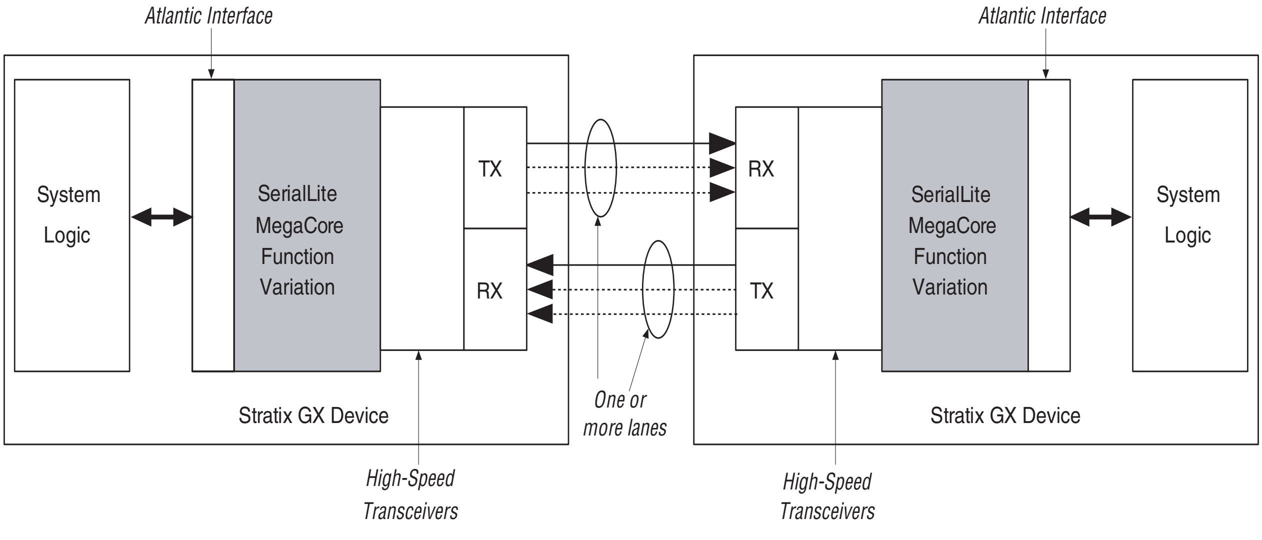
Feature	Default Configuration
Bit rate	3.125 Gbps
Lane count	1
Signal propagation delay	2.5 ns
Clock configuration	Both ends of link use the same clock source (no clock compensation)
Lane polarity reversal	Test only, no reversal
Lane order reversal	NA (only one lane)
Regular data port	Enabled
Data mode	Packet
Channel multiplexing (regular data port)	Disabled
CRC (regular data port)	Disabled
Priority data port	Disabled
Retry on error	NA (priority data port disabled)
Flow control	Disabled
Receive FIFO buffer size (regular data port)	Minimum (16 entries)
Transmitter phase-locked loop (PLL) bandwidth	Low
Receiver PLL bandwidth	Low
Transmitter termination	100 Ω
V_{OD}	1,000 mV
Pre-emphasis	0
Equalization	0
Signal detection	Disabled

Downloaded from Elcodis.com electronic components distributor

Link Consistency

A SerialLite link consists of two instantiations of logic implementing the SerialLite protocol. Each end of the link has a transmitter and a receiver, as shown in Figure 3–1.

Figure 3–1. Complete SerialLite Link



The SerialLite protocol is not a plug-and-play protocol. While there are many configurations to choose from, you must make sure that both ends of your link have the same configuration.

In particular, the SerialLite protocol specifies a symmetric link. The number of lanes in one direction must match the number of lanes in the other direction.

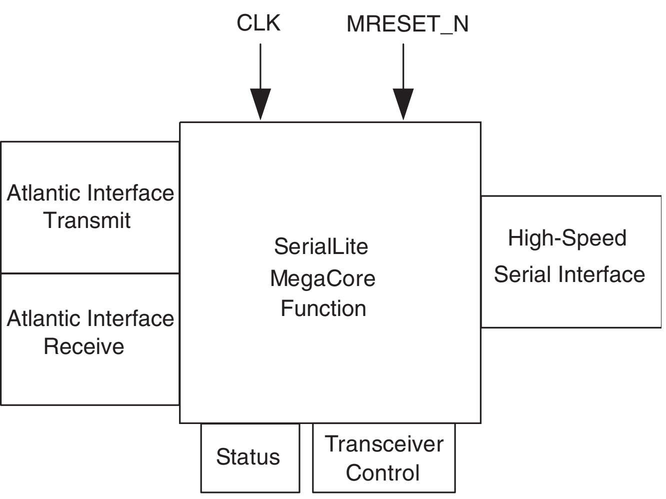
Interface Overview

The SerialLite MegaCore function has four interfaces, shown in Figure 3–2:

- The Atlantic™ interface
- The high-speed serial interface
- The status interface
- The transceiver control interface

Downloaded from Elcodis.com electronic components distributor

Figure 3–2. SerialLite MegaCore Function Interfaces



For clarity, the word *interface* is used to refer to the four interfaces noted above, and the word *port* is used to refer to the specific data ports on the Atlantic interface that allow for regular or priority data.

The status interface is discussed in "Status Interface" on page 3–56. The transceiver control interface is discussed in "Transceiver Settings" on page 3–58.

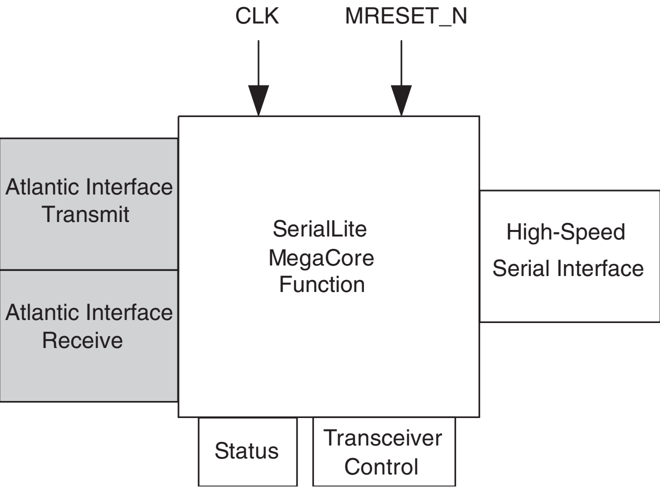
Downloaded from Elcodis.com electronic components distributor

Atlantic Interface

The Atlantic interface (see Figure 3–3) provides a standard mechanism for delivering data to and accepting data from the SerialLite MegaCore function. It is a full-duplex, synchronous point-to-point connection interface that supports a variety of data widths.

For more information on this interface, refer to *FS 13: Atlantic Interface*, available at **www.altera.com**.

Figure 3–3. Atlantic Interfaces



The SerialLite MegaCore function allows you to create one or two data ports: one for regular data and one for priority data. Each of these ports has a full Atlantic interface. Therefore, you may have one of the three configurations shown in Figures 3–4, 3–5, and 3–6. See “Choosing Ports” on page 3–27 for a full description of the behavior of these ports.

Downloaded from Elcodis.com electronic components distributor

Figure 3–4. Atlantic Interface for Regular Data Port Configuration

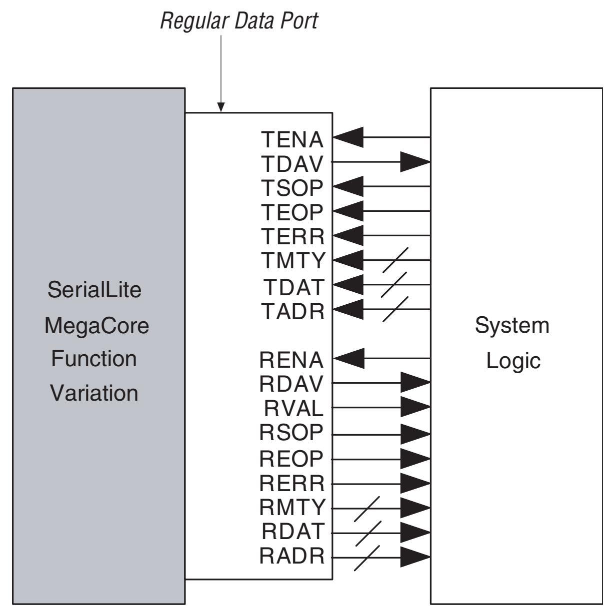
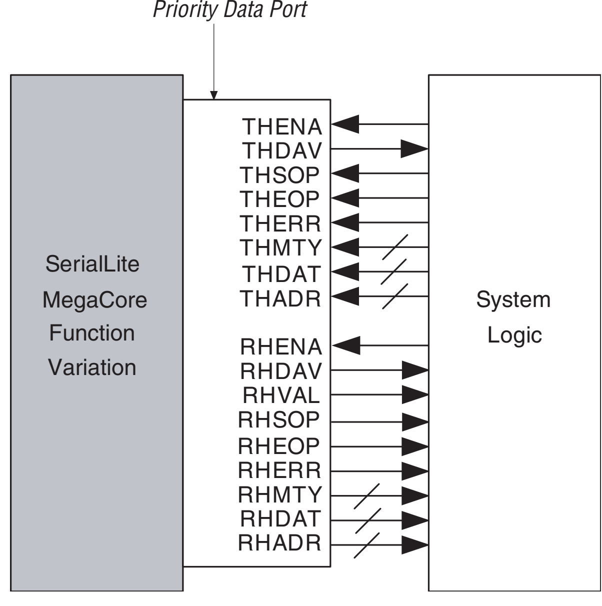
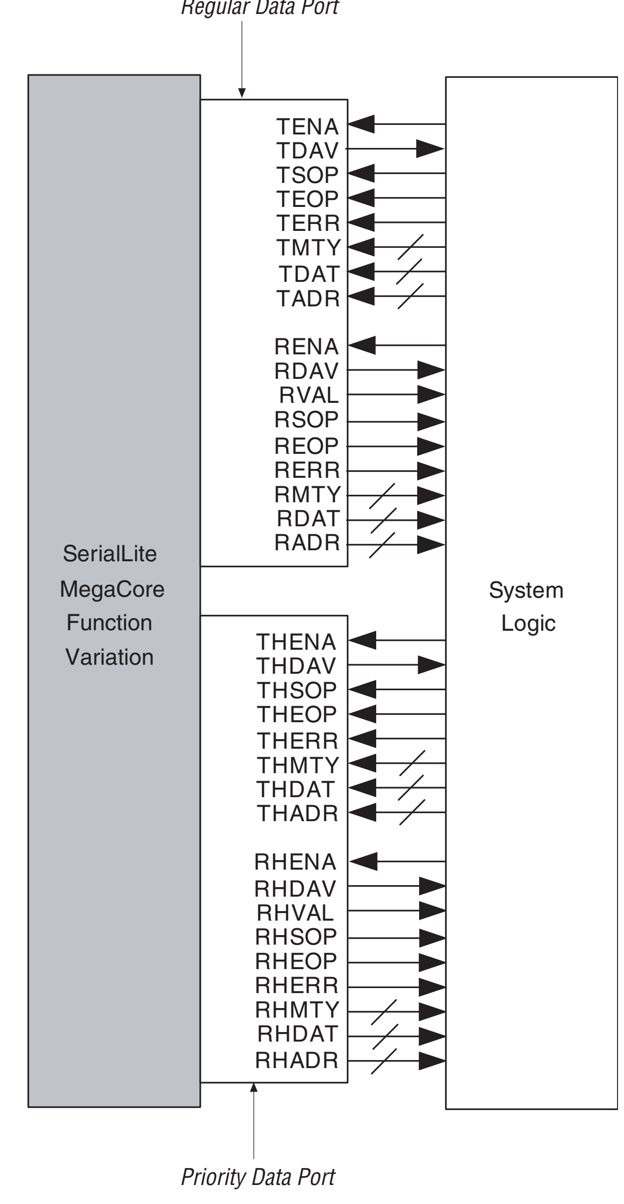


Figure 3–5. Atlantic Interface for Priority Data Port Configuration



Downloaded from Elcodis.com electronic components distributor

Figure 3–6. Atlantic Interface for Regular & Priority Data Port Configuration



The SerialLite MegaCore function is always an Atlantic interface slave. The logic on either side of the SerialLite link always acts as a master. This document refers to the logic that drives data into the SerialLite MegaCore function or receives data from the SerialLite MegaCore function as the "system logic."

Downloaded from Elcodis.com electronic components distributor

The Atlantic interface signals are described in Table 3–2. The signals required for a given configuration, as well as the appropriate bus widths, are created automatically by IP Toolbench based upon the features you select. All Atlantic interface signals operate in the system clock domain.

Table 3–2. Atlantic Interface Signals (Part 1 of 4)

Name	Direction	Description
TDAT[] THDAT[]	Input	Data buses. A data bus carries the main payload data. The width of the bus is determined by the number of lanes in the SerialLite MegaCore function configuration. The width, in bytes, is twice the number of lanes. For example, a 1-lane configuration is 2 bytes wide and a 4-lane configuration is 8 bytes wide. The system logic places data on the data bus for transmission, and reads data on the data bus for reception. Data is presented in big-endian order. Valid bytes are aligned with the most significant byte (MSB). For example, in a 2-lane configuration (which has a 4-byte-wide data bus), if only 3 bytes are valid on the final cycle of a packet, the valid data appears on bits `[31..8]` of the data bus, and the invalid byte is bits `[7..0]` of the data bus. `TDAT[]` is driven by the system logic to transmit data on the regular data port. `THDAT[]` is driven by the system logic to transmit data on the priority data port. `RDAT[]` is driven by the SerialLite MegaCore function to deliver received data on the regular data port. `RHDAT[]` is driven by the SerialLite MegaCore function to deliver received data on the priority data port.
RDAT[] RHDAT[]	Output	
TADR[] THADR[]	Input	The optional address buses. An address bus is only used on ports that enable Channel Multiplexing. The system logic places the channel number on the address bus for transmission, and reads the channel number from the address bus on reception. The width of the address bus is determined by the number of channels being multiplexed. The address bus must be valid at the same time as the data bus, and must remain constant throughout a complete packet. `TADR[]` is driven by the system logic to transmit the channel number on the regular data port if channel multiplexing is enabled for the regular data port. `THADR[]` is driven by the system logic to transmit the channel number on the priority data port if channel multiplexing is enabled for the priority data port. `RADR[]` is driven by the SerialLite MegaCore function to deliver the received channel number on the regular data port if channel multiplexing is enabled for the regular data port. `RHADR[]` is driven by the SerialLite MegaCore function to deliver the received channel number on the priority data port if channel multiplexing is enabled for the priority data port.
RADR[] RHADR[]	Output	

Downloaded from Elcodis.com electronic components distributor

Table 3–2. Atlantic Interface Signals (Part 2 of 4)

Name	Direction	Description
TMTY[] THMTY[]	Input	Word empty buses. A word-empty bus indicates the number of words that contain no data on the last cycle of a packet. The system logic places the appropriate value on the word-empty bus for transmission, and reads its value on reception. The word-empty bus should always be all zero except on the last cycle of a packet on the data bus. When the end-of-packet (EOP) signal is asserted, the number of invalid data bytes on the data bus is specified by the word-empty bus. The width of the word-empty bus is the number of bits required to represent the maximum possible number of empty bytes. For example, in a 4-lane link, there are 8 bytes of data and 7 possible invalid bytes (at least one byte must be valid). The word-empty bus is therefore 3 bits wide to represent values up to 7. TMTY[] is driven by the system logic to specify the number of empty bytes at the end of a packet to be transmitted on the regular data port (packet mode only). RMTY[] is driven by the SerialLite MegaCore function to specify the number of empty bytes at the end of a packet being received on the regular data port (packet mode only). THMTY[] is driven by the system logic to specify the number of empty bytes at the end of a packet to be transmitted on the priority data port. RHMTY[] is driven by the SerialLite MegaCore function to specify the number of empty bytes at the end of a packet being received on the priority data port.
RMTY[] RHMTY[]	Output	
TENA THENA	Input	Data transfer enable. The data transfer enable signal is driven by the system logic and controls the data flow across the interface. When transmitting data, the data transfer enable signal acts as a write-enable from the system logic to the SerialLite MegaCore function. The system logic asserts the data transfer enable signal and the data bus signals simultaneously. When the SerialLite MegaCore function observes the data transfer enable signal asserted on the rising clock edge, it immediately captures the Atlantic data interface signals. TENA acts as a write enable to the regular data port. THENA acts as a write enable to the priority data port.
RENA RHENA	Input	When receiving data, the data transfer enable signal acts as a read-enable from the system logic to the SerialLite MegaCore function. When the SerialLite MegaCore function observes the data transfer enable signal asserted on the rising clock edge, it drives on the next clock edge the Atlantic data interface signals and asserts the data-valid signal. The system logic captures the Atlantic data interface signals on the following rising clock edge. If the SerialLite MegaCore function is unable to provide new data, it deasserts the data valid signal for one or more clock cycles until it is prepared to drive the valid data interface signals. RENA acts as a read enable to the regular data port. RHENA acts as a read enable to the priority data port.

Downloaded from Elcodis.com electronic components distributor

Table 3–2. Atlantic Interface Signals (Part 3 of 4)

Name	Direction	Description
TDAV THDAV RDAV RHDAV	Output	Data available. The data-available signal is driven by the SerialLite MegaCore function to indicate readiness for transmitting or receiving data. When transmitting data, the data-available signal indicates that there is enough space in the SerialLite MegaCore function for one data cycle to be written; THDAV indicates there is enough space in the SerialLite MegaCore function for one packet to be written. When receiving data, the data-available signal indicates that there is at least one cycle's worth of data to be read from the SerialLite MegaCore function. TDAV indicates the SerialLite MegaCore function regular data port can accept at least one cycle of data. THDAV indicates the SerialLite MegaCore function priority data port can accept at least one packet of data. The THDAV is deasserted after the first write cycle of the next packet. However, most of the payload of the next packet can be written into the buffer provided that the THEOP is not written into the buffer before the THDAV is reasserted. The THDAV signal is reasserted after a new buffer becomes available. RDAV indicates the SerialLite MegaCore function regular data port has available at least one cycle of data. RHDAV indicates the SerialLite MegaCore function priority data port has available at least one cycle of data.
RVAL RHVAL	Output	Data valid. The data-valid signal is driven by the SerialLite MegaCore function, and is present only on the receive side of the interface. When high, the data-valid signal indicates valid data signals. The data-valid signal is updated on every clock cycle where the data transfer enable signal is found to be asserted, and holds its current value along with the data bus when the data transfer enable signal is found to be deasserted. Invalid data signals, indicated by the data-valid signal being low, must be disregarded. To determine whether new data has been received, the system logic must qualify the data-valid signal with the previous state of the data transfer enable signal. RVAL is driven by the SerialLite MegaCore function to indicate that valid data is being read on the regular data port. RHVAL is driven by the SerialLite MegaCore function to indicate that valid data is being read on the priority data port.
TSOP THSOP	Input	Start of packet. The start-of-packet signal is used to delineate the starting packet boundary on the data bus. When the start-of-packet signal is high, it indicates that the current data on the data bus is the first data of a new packet. The start-of-packet signal is asserted on the first data cycle of every packet. TSOP is driven by the system logic to start a new packet on the regular data port (packet mode only). THSOP is driven by the system logic to start a new packet on the priority data port. RSOP is driven by the SerialLite MegaCore function to indicate the beginning of a new packet on the regular data port (packet mode only). RHSOP is driven by the SerialLite MegaCore function to indicate the beginning of a new packet on the priority data port.
RSOP RHSOP	Output	

Downloaded from Elcodis.com electronic components distributor

Table 3–2. Atlantic Interface Signals (Part 4 of 4)

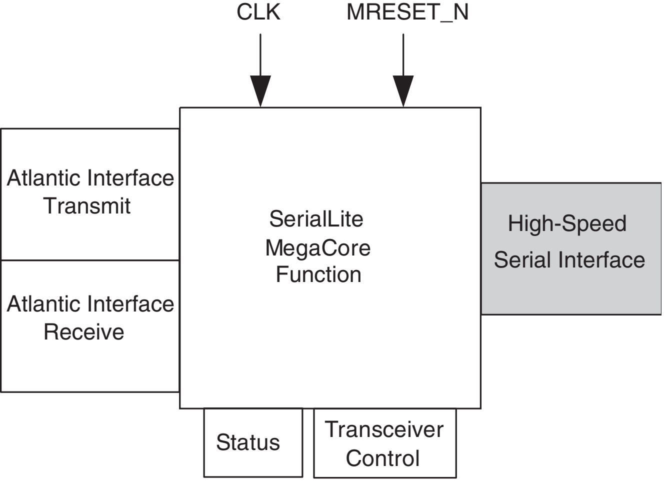
Name	Direction	Description
TEOP THEOP	Input	End of packet (EOP). The end-of-packet signal is used to delineate the ending packet boundary on the data bus. When the end-of-packet signal is high, it indicates that the current data on the data bus is the last data of a packet. The word-empty bus indicates the number of invalid bytes on the data bus when the end-of-packet signal is asserted. The end-of-packet signal is asserted on the last data cycle of every packet. TEOP is driven by the system logic to end a packet on the regular data port (packet mode only). THEOP is driven by the system logic to end a packet on the priority data port. REOP is driven by the SerialLite MegaCore function to indicate that the end of a packet is being read on the regular data port (packet mode only). RHEOP is driven by the SerialLite MegaCore function to indicate that the end of a packet is being read on the priority data port.
REOP RHEOP	Output	
TERR THERR	Input	Error indicator. The error indicator indicates that the current packet is aborted and should be discarded. The error indicator may be asserted at any time during the current packet, but once asserted, it can only be deasserted on the clock cycle after the end-of-packet signal is asserted. TERR is driven by the system logic to indicate that the packet being transmitted on the regular data port is invalid (packet mode only). THERR is driven by the system logic to indicate that the packet being transmitted on the priority data port is invalid. RERR is driven by the SerialLite MegaCore function to indicate that the packet being received on the regular data port is invalid (packet mode only). RHERR is driven by the SerialLite MegaCore function to indicate that the packet being received on the priority data port is invalid (only if the retry-on-error feature is not enabled).
RERR RHERR	Output	

Downloaded from Elcodis.com electronic components distributor

High-Speed Serial Interface

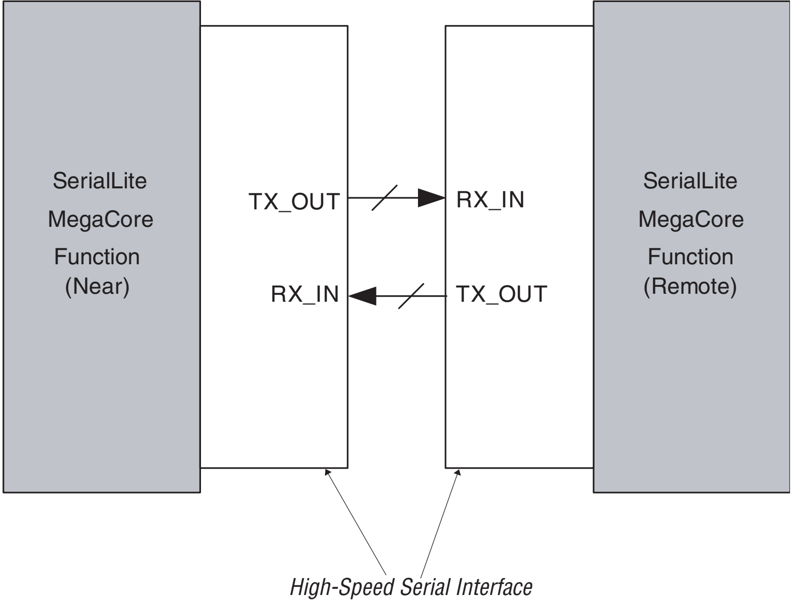
The high-speed serial interface (see Figure 3–7) always appears at the external device pins.

Figure 3–7. High-Speed Serial Interface



The high-speed serial interface consists of the differential signals that carry the high-speed data between the two ends of a link, as shown in Figure 3–8.

Figure 3–8. High-Speed Serial Interface Connections



Downloaded from Elcodis.com electronic components distributor

The high-speed serial interface signals are detailed in Table 3–3. The signals required for a given configuration, as well as the appropriate bus widths, are created by IP Toolbench.

Table 3–3. High-Speed Serial Interface Signals

Name	Direction	Clock Domain	Description
`TX_OUT[]`	Output	System clock, multiplied by 20	The transmit lane(s). These are outputs of the SerialLite MegaCore function. Connect these to the `RX_IN[]` inputs of the SerialLite MegaCore function on the remote end of the link. The width of the bus is equal to the number of lanes specified. In a multi-lane link, `TX_OUT[0]` corresponds to Lane 1. The number of actual wires is twice the number of lanes, since each lane consists of a differential pair of signals.
`RX_IN[]`	Input	Recovered clock	The receive lane(s). These are inputs to the SerialLite MegaCore function. Connect these to the `TX_OUT[]` outputs of the SerialLite MegaCore function on the remote end of the link. The width of the bus is equal to the number of lanes specified. In a multi-lane link, `RX_IN[0]` corresponds to Lane 1. The number of actual wires is twice the number of lanes, since each lane consists of a differential pair of signals.

Other Signals

Table 3–4 describes two miscellaneous signals.

Table 3–4. Miscellaneous Signals

Name	Direction	Clock Domain	Description
`CLK`	Input	System clock	System clock signal, rising-edge sensitive. This is the clock reference for the PLLs and the clock for the FPGA logic. It should operate at a frequency 1/20 of the desired serial bit rate.
`MRESET_N`	Input	Asynchronous	Master reset pin, active low. Asserting this signal brings the SerialLite link down. Once asserted, deasserting this signal causes the link to restart.

Achieving the Desired Bandwidth

The bandwidth that can be realized by a SerialLite link depends on the following:

- Clock rate
- Number of lanes
- Features selected

Downloaded from Elcodis.com electronic components distributor

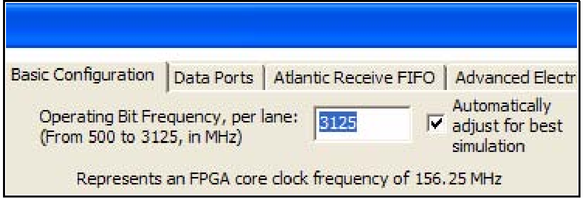
Clock & Data Rates

A SerialLite link can be characterized by a number of clock or data rates or frequencies. Clarification of these various rates and what they are called can help eliminate confusion.

System Clock & Bit Rate

A SerialLite link has two distinct clock rates: the system clock rate and the bit rate. The system clock rate is the rate of the clock you drive directly via the `CLK` input to the SerialLite MegaCore function. This clock controls the FPGA logic and acts as a reference clock to the PLLs. The PLLs boost this clock rate by 20 times to generate the bit rate, which drives the high-speed serial lines. The Parameterize - SerialLite MegaCore Function wizard asks you for the serial bit rate you want to use, and then reports to you the system clock rate required to generate that bit rate. For example, if you enter 3125 MHz as your desired bit rate, then you need to clock the `CLK` signal at 156.25 MHz (see Figure 3–9).

Figure 3–9. Bit Rate & System Clock Frequency



Each lane moves data at the bit rate (Table 3–5 shows the bit rate values). The overall bandwidth can be adjusted by changing the system clock rate, but other system requirements may make that impractical. Even if practical, the system clock rate cannot be set above 156.25 MHz (1/20 of the maximum rated bit rate, 3.125 Gbps). Additionally, if the SerialLite logic is being instantiated in a device that contains a lot of other logic or a slower-speed device, the maximum attainable speed may be lower than 156.25 MHz. In these circumstances, an additional lane is needed to increase bandwidth.

Table 3–5. Bit Rate Values (Mbps)

Minimum	Maximum	Default	Description
500	3125	3125	This is 20 times faster than the system clock, and somewhat faster than the effective data rate, depending on the features selected.

Downloaded from Elcodis.com electronic components distributor

Aggregate Bandwidth

The bit rate specifies the rate of data transmission on a single lane. In a multi-lane configuration, the total available bandwidth is the single-lane bit rate multiplied by the number of lanes.

Effective Data Rate

The bit rate sets the rate at which bits are sent across the high-speed serial link. This is not equivalent to the effective data rate. For example, because 8B/10B encoding is used, each 10 bits of transmitted data corresponds to only 8 bits of actual data.

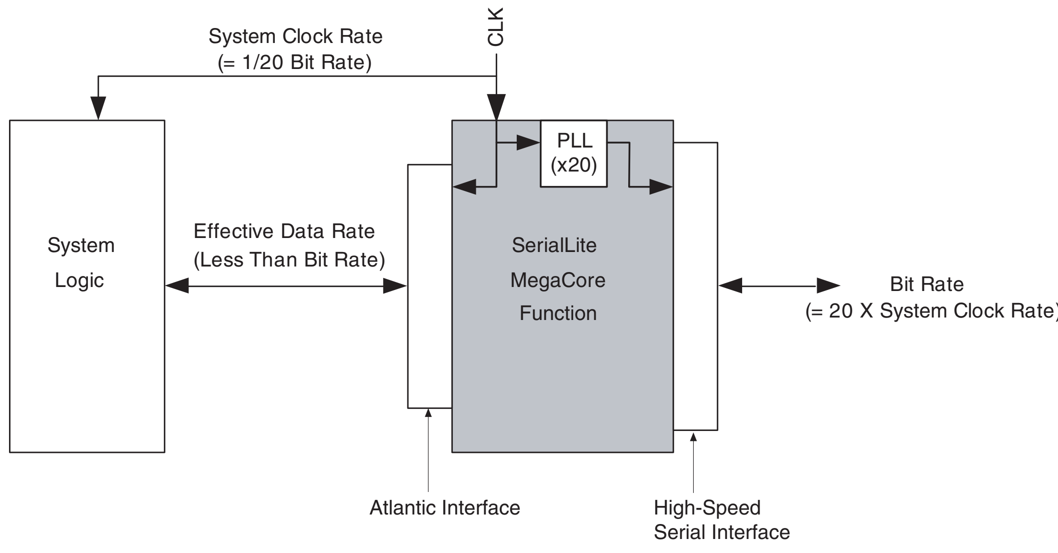
The effective data rate can also be further reduced by other features such as clock compensation, use of the priority port, and the retry-on-error feature. These and other features affect the data rate because they use the same link to transmit various control packets or priority packets in the middle of data packets. While the effect of these features should not be ignored, it may be very small.

The data rate can be further reduced by an inefficient implementation. In particular, using small packets on a link with many lanes, or setting the FIFO buffer sizes and flow control pauses such that the link spends too much time pausing are examples of implementations that reduce the data bandwidth. In these cases, you can improve bandwidth by making adjustments to improve efficiency. Altera provides a Microsoft Excel-based tool that can help you estimate and improve the efficiency of your SerialLite link. You can find the calculator and *Application Note 389: SerialLite MegaCore Function Bandwidth Calculator* at **www.altera.com/products/ip/communications/additional_functions_comm/m-alt-seriallite.html**.

The relationships between the various rates are illustrated in Figure 3–10 on page 3–16.

Downloaded from Elcodis.com electronic components distributor

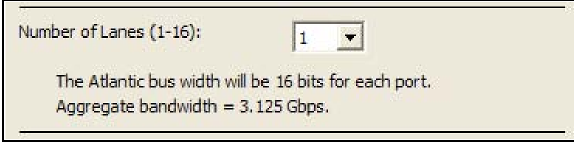
Figure 3–10. Clock Relationships



Scaling by Adding Lanes

Because each lane operates at the bit rate, you can increase bandwidth by adding lanes. This is a simple way to scale the link during system design. If adding a lane provides more bandwidth than needed, you can reduce the system clock rate, mitigating possible high-speed design issues and making it easier to meet performance. By setting an appropriate clock rate and lane width, a desired aggregate bit rate can be achieved. The aggregate bandwidth is reported in the Parameterize - SerialLite MegaCore function wizard, as shown in Figure 3–11.

Figure 3–11. Aggregate Bit Rate



Downloaded from Elcodis.com electronic components distributor

Table 3–6 shows the allowed lane count values.

Table 3–6. Lane Count Values

Minimum	Maximum	Default	Description
1	16	1	The number of lanes that are instantiated in the SerialLite MegaCore function variation.

Adjusting Frequency for Best Simulation

The SerialLite MegaCore function is implemented in Stratix GX devices, using the ALTGXB transceivers. In order for the transceiver to be simulated accurately cycle for cycle, the system clock rate must be convertible to a clock period that is an integral number of picoseconds. For example, 156.25 MHz corresponds to a period of 6,400 picoseconds, and 156 MHz corresponds to a period of 6,410.256 picoseconds, which is rounded to 6,410. In the latter case, the specified frequency and the underlying period do not correspond exactly, and the simulation may occasionally miss a byte. For this reason, best simulation is achieved by using a clock rate (or bit rate) that results in a picosecond-integer period.

If you enter a bit rate that does not correspond to a picosecond-integer period, then a checkbox is available that allows you to use the next highest frequency with a picosecond-integer period. Table 3–7 shows the adjustment options.

If you turn on the option, then that adjusted frequency is used. If you do not turn on the option, then the frequency you entered is used, and simulation could occasionally miss a byte.

The device always operates correctly, and never loses any bytes, regardless of the frequency selected. Only simulation is affected.

Table 3–7. Frequency Adjustment Options

Option	Description
Enabled	Ensures that the bit rate entered corresponds to a picosecond-integer system clock period. If necessary, the entered value is adjusted up to the closest bit rate with such a correspondence. This is the default setting.
Disabled	Makes no change to the bit rate entered. If the system clock period is not a picosecond integer, simulation may occasionally miss bytes. The device always operates correctly and never misses bytes.

Downloaded from Elcodis.com electronic components distributor

Signal Propagation Delay

The SerialLite link has latency inherent in its operation. This latency affects both the retry-on-error timeout value and the necessary sizes of the receive FIFO buffers when flow control is used. But the overall link latency must also account for the delay of the serial signals along their medium, whether traces on boards or cables between boards. This is referred to in the Parameterize - SerialLite MegaCore function wizard as the signal propagation delay. The phrase "signal propagation delay" is used whether or not a wire is physically used.

In order to ensure that the retry-on-error timeout and receive FIFO buffer sizes are correctly set, you must specify the signal propagation delay using a nanosecond value. In a multi-lane implementation, if there are significant differences between the lanes, use the worst-case (slowest) lane. This value is then converted to the equivalent number of clock cycles for use in latency calculations, and the results are reported in the Parameterize - SerialLite MegaCore function wizard, as shown in Figure 3–12.

Figure 3–12. Signal Propagation Delay

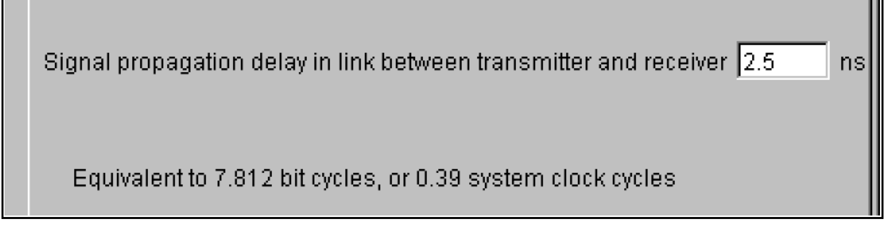


Table 3–8 shows the allowed signal propagation delay values.

Table 3–8. Signal Propagation Delay Values

Minimum	Maximum	Default	Description
0	N/A	2.5	The signal delay, in nanoseconds.

Downloaded from Elcodis.com electronic components distributor

Summary of Bandwidth-Related Settings

Table 3–9 summarizes the different ways you can change the bandwidth of the SerialLite link.

Table 3–9. Bandwidth-Related Settings

Setting	Description
Bit rate	Specifies the rate at which bits are sent on the high-speed serial interface. Equal to 20× the system clock rate, and faster than the effective data rate.
Automatically adjust for best simulation	Used to ensure that the bit rate chosen corresponds to a picosecond-integer system clock period.
Lane count	The number of lanes to be created. Each lane operates at the specified bit rate.
Signal propagation delay	The propagation delay of a signal from when it is placed on the `TX_OUT` pins of one end of the link to when it is received on the `RX_IN` pins of the other end of the link.

Clock Compensation

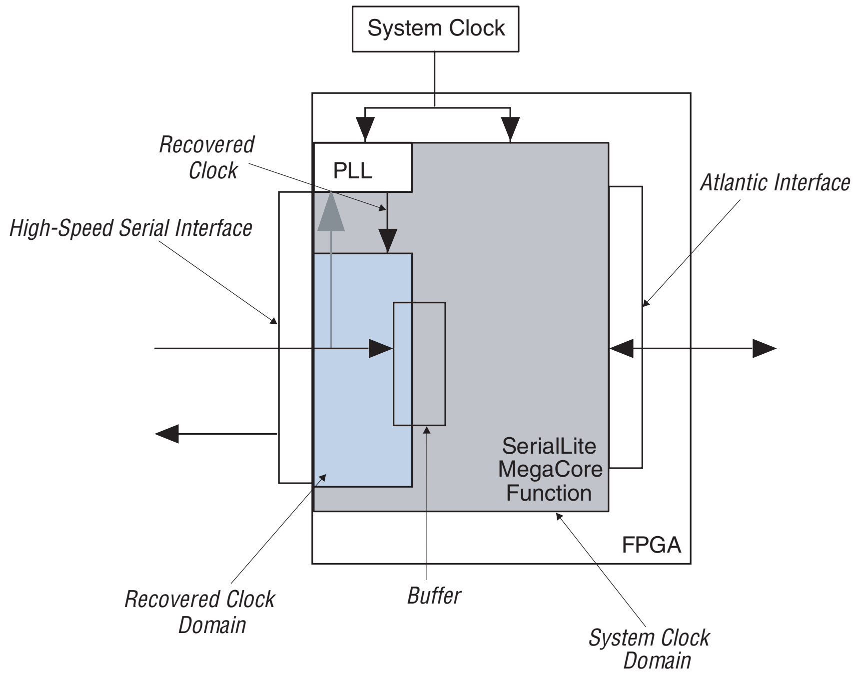
The configuration of your system clock or clocks determines whether or not you need to use clock compensation.

Clock Domains

The SerialLite internal logic contains two clock domains. The majority of the logic is clocked by the system clock via the `CLK` signal. However, a small part of the receiver logic is clocked by the clock signal recovered from the received data stream. The received data has to cross from this recovered clock domain into the system clock domain, as illustrated in Figure 3–13. A FIFO memory is used to buffer the data as it crosses from one domain to the other. Depending on the relationship between the system clock and the recovered clock, compensation may be required to ensure that no data is lost due to a frequency mismatch.

Downloaded from Elcodis.com electronic components distributor

Figure 3–13. Receiver Clock Domains



Clock Configuration

The SerialLite protocol supports clock compensation if the application needs it. Clock compensation is needed if the two ends of the link are being clocked by different clock sources.

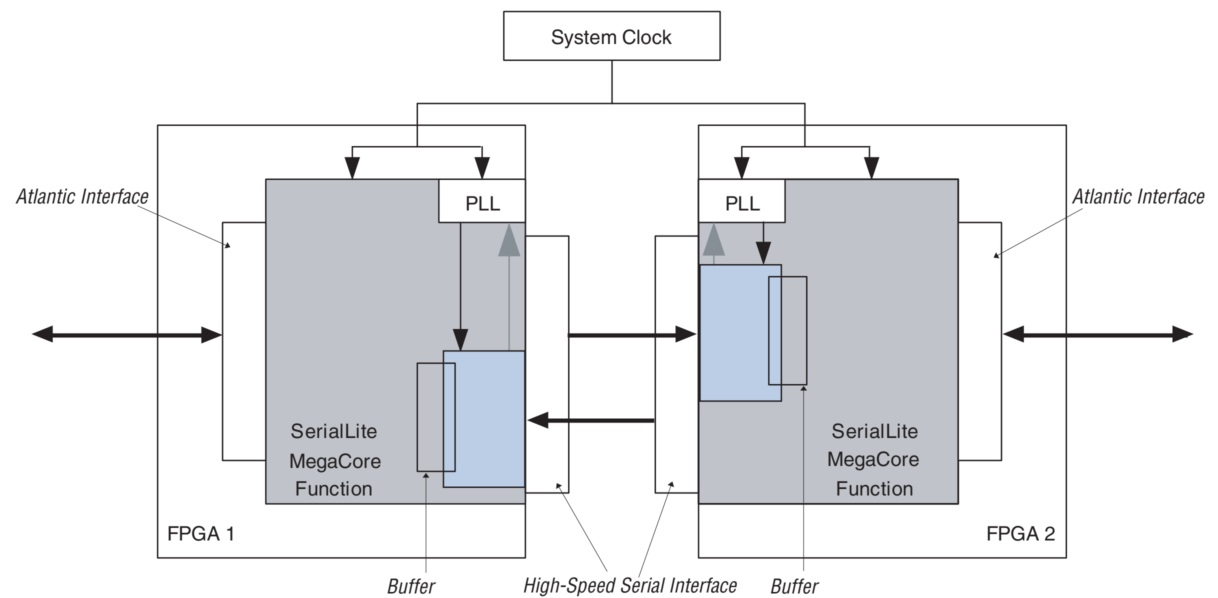
There are two configurations possible for clocking your link. The Parameterize - SerialLite MegaCore function wizard clock compensation selection is based on these configurations. By selecting the configuration that corresponds to your system, clock compensation is either included or not.

Single Clock Source

A single-clock configuration is typically used for a link where both ends are on the same board or on two boards driven by the same system clock (see Figure 3–14). Because both ends of the link use the same clock source, no clock compensation is needed. If you are using this configuration, select the **Near end and far end use the same crystal** option in the Clock Configuration portion of the wizard. This is the default selection.

Downloaded from Elcodis.com electronic components distributor

Figure 3–14. Single-Source Clock Configuration

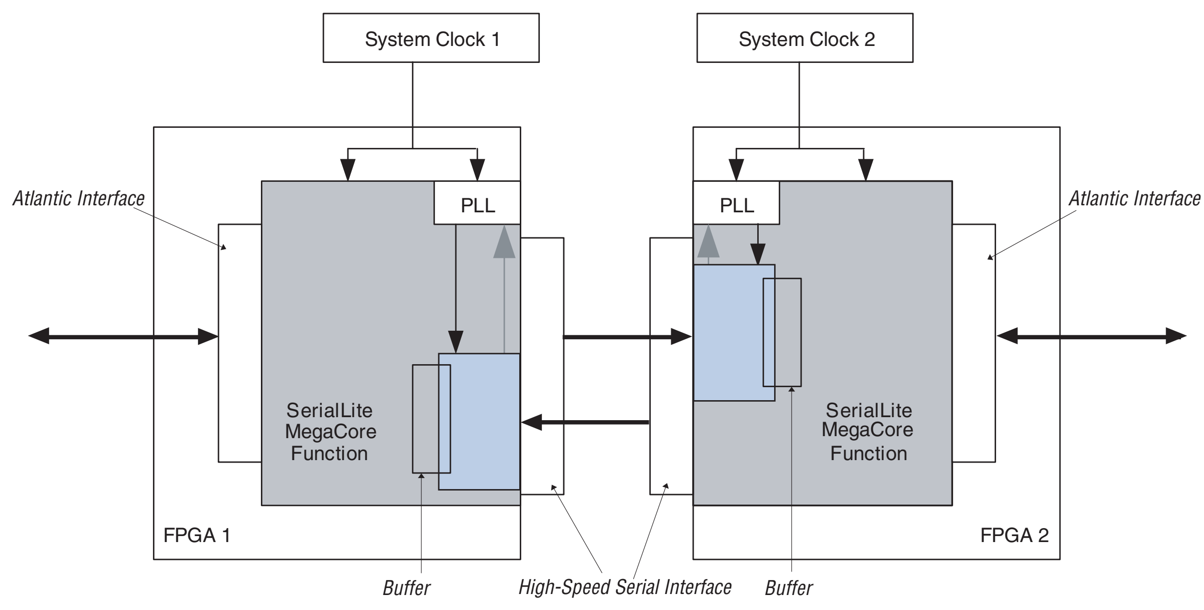


Two Clock Sources

Designs often use a two-clock configuration when the two ends of the link are on different boards, each having its own clock source (see Figure 3–15). The two clock sources must have the same nominal frequency, but may differ by a few hundred parts per million (ppm) because of clock tolerances. Clock compensation is required in this configuration to ensure that no data is lost on a system that is transmitting slightly faster than the receiver is processing. If you are using this configuration, select the **Near end and far end use different crystals** option in the Clock Configuration portion of the wizard.

Downloaded from Elcodis.com electronic components distributor

Figure 3–15. Independent Clock Sources



For this configuration, you can choose one of two clock tolerances: 100 ppm and 300 ppm. Make this choice based upon the clock sources you are using. When you select the 100 ppm tolerance, the clock compensation sequence is inserted less frequently, and clock compensation has less impact on bandwidth. Tables 3–10 and 3–11 show the clock compensation and tolerance options, respectively.

Table 3–10. Clock Compensation Options

Option	Description
Near end and far end use the same crystal	Used when a single clock source is used for both ends of the link. No clock compensation is implemented. This is the default setting.
Near end and far end use different crystals.	Used when the clock sources for the two ends of the link are independent, but of the same nominal frequency. Clock compensation is implemented.

Downloaded from Elcodis.com electronic components distributor

Table 3–11. Clock Tolerance Options

Option	Description
300 ppm	A clock compensation sequence is automatically inserted into the serial stream every 1667 clock cycles. Available only if the **Near end and far end use different crystals** setting has been selected for clock configuration. This is the default setting.
100 ppm	A clock compensation sequence is automatically inserted into the serial stream every 5000 clock cycles. Available only if the **Near end and far end use different crystals** setting has been selected for clock configuration.

Clock Pad Restrictions

The SerialLite MegaCore function is implemented in Stratix GX devices, using the ALTGXB transceivers. As the clock and reset operations of the SerialLite MegaCore function and the clock and reset operations of the ALTGXB transceivers are related, some clocking and reset issues may occur when the `REFCLKB` pins are used.

The `REFCLKB` pins are clock pads, available on every transceiver, that can drive the reference clock into the PLLs. You cannot use the `REFCLKB` pad on the active transceiver to drive the `CLK` input of the SerialLite MegaCore function.

The SerialLite MegaCore function implements a reset block that resets the ALTGXB transceiver. This reset block uses the SerialLite clock to drive the reset state machine. The reset state machine drives several signals, including the `PLL_ARESET` and `RX_ANALOGRESET` signals. When these reset signals are asserted, the `REFCLKB` pad goes into the reset state, so there is no active clock to the SerialLite MegaCore function. Without an active clock, the SerialLite MegaCore function can never come out of the reset state.

To avoid these issues, do not use the `REFCLKB` pad to clock the SerialLite MegaCore function. Instead, implement one of the following clocking options:

- Option 1—Use the `REFCLKB` pad of an unused transceiver to generate the clock. Figure 3–16 on page 3–24 shows a block diagram of this option.

- Option 2—Use another global clock resource. Figure 3–17 on page 3–24 shows a block diagram of this option.

Downloaded from Elcodis.com electronic components distributor

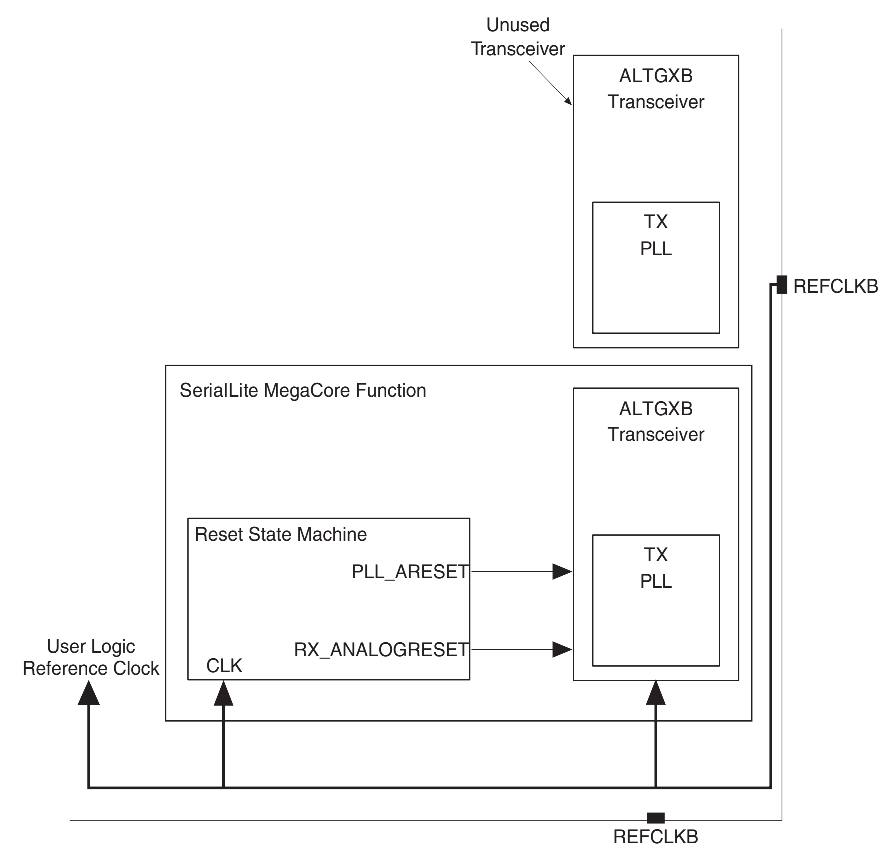
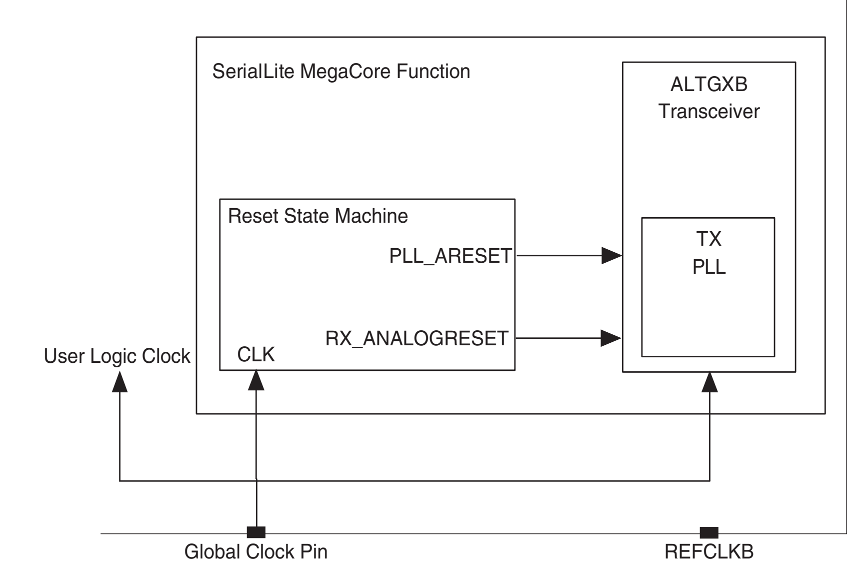


Figure 3–16. Option 1—Use an Unused Transceiver's REFCLKB Pad

Figure 3–17. Option 2—Use Another Global Clock Resource



Downloaded from Elcodis.com electronic components distributor

Lane Polarity & Order Reversal

The SerialLite protocol optionally allows the link to recover from some connection problems. Lane polarity and lane order can be reversed automatically if desired.

Lane Polarity

Each lane consists of a differential pair of signals. It is possible for the positive and negative sides of this pair to be reversed because of layout error or because it simplifies layout. Reversed is shown in Figures 3–18 and 3–19. The SerialLite logic always detects such a reversal.

Figure 3–18. Correct Lane Polarity

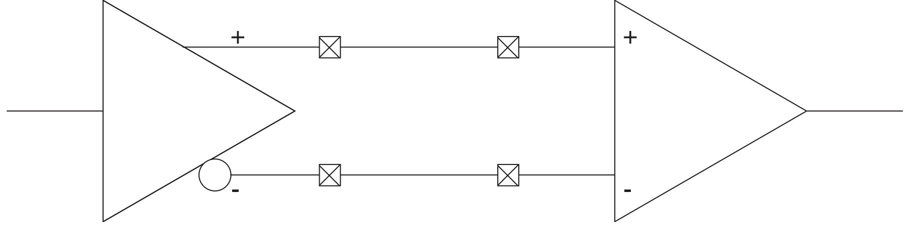
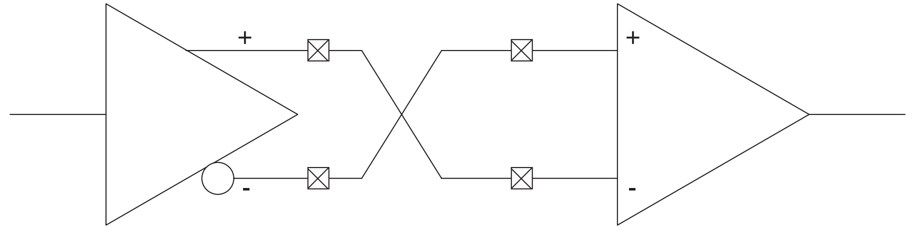


Figure 3–19. Reversed Lane Polarity



If desired, the SerialLite logic can compensate for such a reversed lane on the receive side. This reversal occurs during link initialization, and remains in place for as long as the link is active. Using this option adds logic to the SerialLite logic implementation size.

If reversal is not selected, and if the link detects that the polarity is incorrect, a catastrophic error is declared.

To configure the link to detect lane polarity errors but not to reverse the logic, select the **Test only** option for Lane Polarity in the Link Start-Up portion of the Parameterize - SerialLite MegaCore function wizard. This is the default selection.

Downloaded from Elcodis.com electronic components distributor

To configure the link to detect and automatically correct lane polarity errors, select the **Test and reverse** option for Lane Polarity in the Link Start-Up portion of the wizard. Table 3–12 shows the lane polarity options.

Table 3–12. Lane Polarity Options

Option	Description
Test only	Reversed polarity triggers a catastrophic error. This is the default setting.
Test and reverse	Reversed polarity is corrected at the receiver.

Lane Order

It is possible that the order of lanes may be incorrect due to layout errors. It may also be reversed (the most significant lane of one end of the link is connected to the least significant lane of the other end) due to layout constraints (shown in Figures 3–20 and 3–21). The SerialLite logic always detects a lane order mismatch.

If selected as an option, the SerialLite logic can compensate for reversed lane order on the receive side. This reversal occurs during link initialization, and remains in place for as long as the link is active.

Figure 3–20. Correct Lane Order

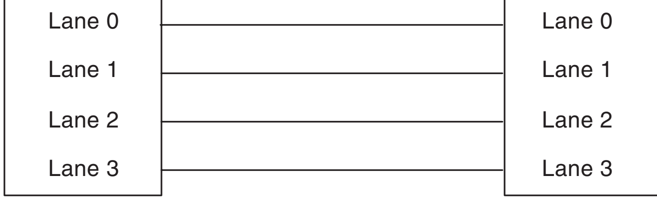


Figure 3–21. Reversed Lane Order

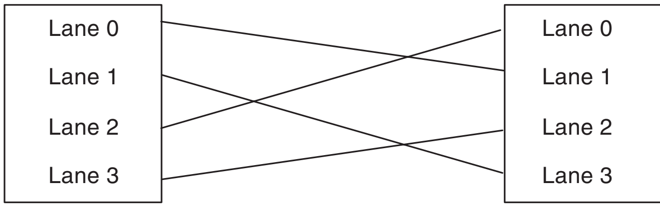
Downloaded from Elcodis.com electronic components distributor

If reversal is not selected and the link detects that the lane order is incorrect, a catastrophic error is declared by the SerialLite MegaCore logic.

Only reversed lane order can be corrected. If the lane order is scrambled, as shown in Figure 3–22, the receiving end cannot unscramble it, and a catastrophic error is declared by the SerialLite MegaCore logic.

Catastrophic errors are described in "Error Handling" on page 3–54.

Figure 3–22. Scrambled Lane Order



To configure the link to detect lane order errors but not to reverse the logic, select the **Test only** option for Lane Order in the Link Start-Up portion of the Parameterize - SerialLite MegaCore function wizard. This is the default selection.

To configure the link to detect and automatically correct reversed lane order, select the **Test and reverse** option for Lane Order in the Link Start-Up portion of the wizard. Table 3–13 shows the lane order options.

Table 3–13. Lane Order Options

Option	Description
Test only	Incorrect lane order triggers a catastrophic error. This is the default setting.
Test and reverse	Reversed lane order is corrected at the receiver. Scrambled lane order triggers a catastrophic error.

Choosing Ports

The SerialLite MegaCore function provides two ports: a regular data port and a priority data port. The ports operate differently, and have different strengths. You can use one or the other or both ports. At least one of the two must be used.

Downloaded from Elcodis.com electronic components distributor

There are a number of settings that can be specified for each port. These are summarized in Table 3–14 and described in more detail in the sections that follow.

Table 3–14. Available Port Settings

Setting	Description
Data mode	Regular data port only. Allows selection of streaming or packet data.
Packet size testing	Priority data port only. Depending on setting, determines allowed maximum priority packet sizes.
Maximum packet size	Priority data port only. Specifies the largest packet that is received on the priority port.
Channel multiplexing	Allows implementation of channel multiplexing and specification of the number of channels to multiplex.
CRC	Allows implementation of CRC and selection of various CRC options.
Retry on error	Priority data port only. Allows implementation of the retry-on-error feature and specification of the retry timeout.

Regular Data Port

Use the regular data port when:

- The lowest latency is required
- Unlimited packet sizes are desired
- Streaming data is required
- The retry-on-error feature is not required

The regular data port operates in cut-through mode, which means that as soon as data is received for transmission, it is transmitted without waiting for an entire packet to arrive. This results in very low latency. It also results in a smaller receive FIFO buffer. Table 3–15 shows the regular data port options.

Table 3–15. Regular Data Port Options

Option	Description
Enabled	Creates the signals and logic required to support a regular data port. This is the default setting.
Disabled	Does not create the signals and logic required to support a regular data port. If this option is chosen, then a priority data port must be created.

Downloaded from Elcodis.com electronic components distributor

Priority Data Port

Use the priority data port when:

- Certain data packets have higher priority than other data
- You want to use the retry-on-error feature

If both regular and priority data ports are provided, then the data delivered to the priority data port interrupts any data being transmitted via the regular data port. This behavior is referred to as packet nesting, because a priority data packet is nested within a regular data packet on a serial link. No regular data port data is lost during transmission of the priority packet. Once the priority data has been transmitted and no new priority data is available, transmission of the regular data packet resumes.

The priority data port operates in store-and-forward mode, which means that no data is transmitted across the link until the entire packet has been delivered through the Atlantic Transmit interface and stored in a buffer. When the retry-on-error feature is not enabled, two packet buffers are maintained. When the retry-on-error feature is enabled, eight packet buffers are maintained.

While a priority packet buffer is being filled, if there are no other full priority packet buffers, regular data continues to be transmitted. Regular data is only interrupted once a priority packet is ready for actual transmission.

Packet size on the priority data port is limited to 256 bytes. The priority data port cannot support streaming data. Table 3–16 shows the priority data port options.

Table 3–16. Priority Data Port Options

Option	Description
Enabled	Creates the signals and logic required to support a priority data port.
Disabled	Does not create the signals and logic required to support a priority data port. If this option is chosen, then a regular data port must be created. This is the default setting.

Streaming & Packet Data

The regular data port allows data to be formatted as a stream or in packets. Streaming data has no beginning or end. It represents an unending sequence of data bytes. Packets of data, by contrast, have a

Downloaded from Elcodis.com electronic components distributor

well-defined beginning and end. The data source determines whether streaming or packet data is used. For example, data from an antenna is most likely streaming; network traffic is most likely in packets.

If the source of streaming data cannot be paused, then any interruptions to the serial stream of data may result in lost data. The following items can interrupt the serial stream:

- Priority packets (if used)
- Flow control (if used, even if only implemented on the priority data port)
- Clock compensation sequences (if used)
- Retry on error packet acknowledgments (if used)

For example, the existence of a priority port alongside the streaming regular data port means that a priority packet interrupts the data stream because of the packet nesting behavior. If the source of that data stream cannot be paused, then any streaming data received while the priority packet is being transmitted is lost. In general, for best fidelity, any streaming data that can be interrupted should have a source that can be paused.

The priority data port can only accept packet data because of its store-and-forward architecture. Streaming data is not allowed on the priority data port. Table 3–17 shows the data mode options for the regular data port.

Table 3–17. Data Mode Options (Regular Data Port Only)

Option	Description
Packet	The regular data port expects data to arrive in packets, marked by asserting `TSOP` at the beginning and `TEOP` at the end of the packet. This is the default setting.
Streaming	The regular data port expects data to be streaming. The `TSOP`, `TEOP`, `TMTY[]`, `TADD[]`, `TERR`, `RSOP`, `REOP`, `RMTY[]`, `RADD[]`, and `RERR` signals are not available on the Atlantic interface.

Packet Sizes

The two data ports differ in their packet size requirements. The nature of the packets used in your application may affect your choice of which port to use.

Downloaded from Elcodis.com electronic components distributor

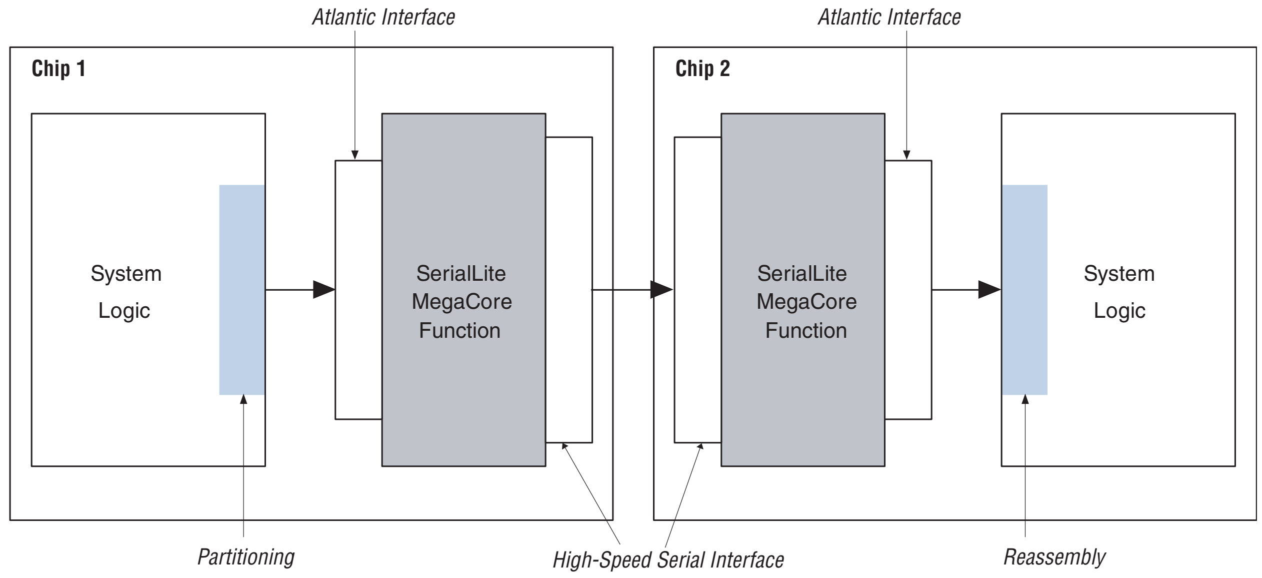
Maximum Packet Sizes

The regular data port does not have a limit on the size of the packets that can be transmitted. The use of larger or smaller packets on the regular data port does not affect the amount of logic or memory required to implement the port.

The priority data port has a packet size limit of 256 bytes because of its store-and-forward architecture. If your application has packets larger than this, they must be partitioned into multiple 256-byte packets when delivered to the priority Atlantic interface. These sub-packets can then be reassembled into the original larger packet after being received on the other end of the link from the remote Priority Atlantic interface.

The SerialLite MegaCore function does not perform any partitioning or reassembly of packets. System logic must perform these functions, as shown in Figure 3–23.

Figure 3–23. Breaking Up Large Priority Packets



Although the priority port packet size is limited to 256 bytes, a smaller maximum packet size can be specified. The priority packet buffers are sized to hold the biggest specified packet, so specifying a smaller packet requires less memory resources in the FPGA. Table 3–18 on page 3–32 shows the allowed size values for priority data packets.

Downloaded from Elcodis.com electronic components distributor

Only the maximum priority packet size is specified. Individual packets can be any size up to and including this maximum size.

Table 3–18. Maximum Priority Data Packet Size Values

Minimum	Maximum	Default	Description
2	256	32	The size, in bytes, of the largest priority packet that is allowed in the SerialLite implementation being created. Packets are not required to be this size, and may be smaller. Only available if the priority data port has been enabled.

Packet Size Testing on the Priority Data Port

Because packet sizes are limited on the priority data port, the priority packet buffers are built only big enough to hold the maximum packet specified. If a packet is delivered that exceeds the maximum size, the extra data is lost. The SerialLite MegaCore function is capable of testing incoming packets to ensure that any oversize packets are flagged as an error on the status interface.

As long as the maximum packet size is an even multiple of the bus width, individual packets that are smaller than the specified maximum can still be tested. If an oversize packet is received, it is discarded and bit 5 of the status interface is asserted.

If packets are not tested for size, then any maximum packet size can be specified. If an oversize packet is received, no error is flagged, and data may be lost.

Downloaded from Elcodis.com electronic components distributor

The Parameterize - SerialLite MegaCore function wizard manages this choice for you. If you select to test packet size, then invalid packet size choices are flagged as errors. If you select not to test packet size, you can enter any packet size up to and including 256 bytes. Table 3–19 shows the packet testing options for the priority data port.

Table 3–19. Packet Testing Options (Priority Data Port Only)	
Option	**Description**
Enabled	Logic is created to test incoming data for oversize packets. Oversize packets are discarded. Maximum packet size choices must be a multiple of the data bus width. This is the default setting.
Disabled	No logic is created to test incoming data for oversize packets. Oversize packets are not discarded, and are missing data. Any maximum packet size up to and including 256 bytes can be specified.

Channel Multiplexing

Channel multiplexing allows a single SerialLite link to carry more than one independent stream of data. Both the regular data port and the priority data port support channel multiplexing.

Channel multiplexing can be useful for such structures as multi-queue memories. Another use is in distinguishing system messages from standard payload packets on the priority port.

Up to 256 channels can be multiplexed on the regular data port, and up to 16 channels can be multiplexed on the priority data port. You decide independently for each port whether or not to enable channel multiplexing.

Channel multiplexing tags a packet with a channel number. The channel number is placed on the `TADR` or `THADR` bus for transmission, and is read off of the `RADR` or `RHADR` bus on reception. No processing is done on the channel number; the SerialLite MegaCore function simply passes the address through to the other end of the link.

Packets to different channels on the same port cannot interrupt each other. Once a packet for one channel has started, that packet must be completed before a new packet for a different channel can be started. Packets on the priority data port still interrupt packets on the regular data port, regardless of channel multiplexing.

Downloaded from Elcodis.com electronic components distributor

It is the responsibility of the transmitting system logic to multiplex the different channels of data and create the channel number. It is also the responsibility of the receiving system logic to read the channel number and demultiplex the output data accordingly. This is shown in Figure 3–24.

Figure 3–24. Using the Channel Multiplexing Feature

Chip 1
System Logic
Channel 1 Source
Channel 2 Source
Channel *n* Source
Atlantic Interface
TDAT (THDAT)
TADR (THADR)
SerialLite MegaCore Function
Chip 2
System Logic
Atlantic Interface
SerialLite MegaCore Function
RDAT (RHDAT)
RADR (RHADR)
Channel 1 Destination
Channel 2 Destination
Channel *n* Destination
Multiplex, Generate Channel Number
High-Speed Serial Interface
Read Channel Number, Demultiplex

Because the SerialLite MegaCore function does not actually do anything with the channel number, it is possible for the transmitting system logic to put any channel number desired on the address bus. For example, Channel 1 might direct a packet to Channel 2. The address bus can also be used to carry any other tag that might be required, as long as it is no more than 8 bits on the regular data port or 4 bits on the priority data port. The Parameterize - SerialLite MegaCore function wizard asks for a number of

Downloaded from Elcodis.com electronic components distributor

channels to multiplex, so if a six-bit tag is desired, for example, then specify 64 channels (=2^6). Table 3–20 shows the channel multiplexing options.

Table 3–20. Channel Multiplexing Options	
Option	**Description**
Enabled	If selected for the regular data port, `TADR` and `RADR` buses are created, and a number of channels from 2 to 256 can be specified. If selected for the priority data port, `THADR` and `RHADR` buses are created, and a number of channels from 2 to 16 can be specified. The width of the address buses is the minimum required to handle the number of channels specified. Channel multiplexing is specified independently for each port.
Disabled	If selected for the regular data port, `TADR` and `RADR` buses are not created. If selected for the priority data port, `THADR` and `RHADR` buses are not created. Channel multiplexing is specified independently for each port. This is the default setting.

Table 3–21 shows the channel count values for the regular data channel.

Table 3–21. Regular Data Channel Count Values			
Minimum	**Maximum**	**Default**	**Description**
2	256	2	The number of channels that are multiplexed through the Atlantic regular data port. `TADR` and `RADR` buses are created and sized with the number of bits required to carry the maximum channel number. For example, if 64 channels are specified, then the `TADR` and `RADR` buses are 6 bits wide. Only available if channel multiplexing is enabled on the regular data port.

Downloaded from Elcodis.com electronic components distributor

Table 3–22 shows the channel count values for the priority data channel.

Table 3–22. Priority Data Channel Count Values

Minimum	Maximum	Default	Description
2	16	2	The number of channels that are multiplexed through the Atlantic priority data port. THADR and RHADR buses are created and sized with the number of bits required to carry the maximum channel number. For example, if 12 channels are specified, then the TADR and RADR buses are 4 bits wide. Only available if channel multiplexing is enabled on the priority data port.

Data Integrity Protection: CRC

If you need error protection beyond that provided by 8B/10B encoding, you may add cyclic redundancy code (CRC) checking to your packet. The CRC is automatically generated in transmission, and is automatically checked on reception. On the regular data port, a CRC check failure results in the packet being marked as bad using the RERR signal on the Atlantic interface. On the priority data port, a CRC check failure either results in the packet being marked bad using the RHERR signal on the Atlantic interface, or, if the retry-on-error feature has been enabled, results in the resending of the packet in question. You decide independently for each port whether CRC usage is enabled.

CRC is not available for the streaming mode on the data port.

16-Bit Versus 32-Bit

The SerialLite MegaCore function supports the use of both 16-bit and 32-bit CRC algorithms. You decide which CRC algorithm to use independently for each port. The 16-bit algorithm generates a two-byte result, and uses the CCITT polynomial,

$$G(x) = X^{16} + X^{12} + X^5 + 1$$

The 32-bit algorithm generates a four-byte result, and uses the polynomial,

$$G(x) = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$$

Downloaded from Elcodis.com electronic components distributor

The 16-bit version provides excellent protection for packets smaller than about 1K bytes. For larger packets, CRC-32 can be considered, but it requires significantly more logic, especially on implementations requiring many lanes. At 16 lanes, CRC-32 logic may constitute as much as half of the logic of the entire SerialLite instantiation. Therefore CRC-32 should only be used when absolutely necessary.

Because of the 256-byte packet size limitation, CRC-32 is never really needed on the priority data port (although it is available).

Half Duplex

If you want to reduce the amount of logic used for implementing CRC logic, there are situations where you can cut it approximately in half. The SerialLite protocol specifies a symmetric full-duplex link, where traffic is passed in both directions. There may be applications, however, where the traffic in one direction is qualitatively different from that in the other direction. For instance, payload data may travel in one direction, with system messages or acknowledgments coming in the reverse direction.

In such systems, you may only be concerned about the data integrity in one direction. The Parameterize - SerialLite MegaCore function wizard allows you to specify, for a given application, to generate a CRC on transmission, but not to check on reception. Conversely, you can specify to check a CRC on reception, but not to generate on transmission. It is important that both ends of the link match each other. So if the near-end logic generates CRC, then the far-end logic must check CRC, and vice versa.

The wizard allows you to make this selection by asking which direction you wish to protect with CRC. The default provides protection in both directions. Alternatively, you can select to protect only the transmit (TX) direction or the receive (RX) direction. You decide independently for each port which directions you wish to protect.

Tables 3–23 through 3–25 show the different CRC options.

Table 3–23. CRC Options

Option	Description
Enabled	CRC logic is created. CRC usage is specified independently for each port.
Disabled	CRC logic is not created. CRC usage is specified independently for each port. This is the default CRC setting.

Downloaded from Elcodis.com electronic components distributor

Table 3–24. CRC Type Options

Option	Description
16-bit	Generates a two-byte CRC. Adequate for packets of around 1K bytes or smaller. CRC algorithm is specified independently for each port. Available on a given port only if CRC usage is enabled for that port. This is the default algorithm setting once CRC usage has been enabled.
32-bit	Generates a four-byte CRC. Should only be used for packets larger than about 1K bytes or when extreme protection is required, since it is resource-intensive. CRC algorithm is specified independently for each port. Available on a given port only if CRC usage is enabled for that port.

Table 3–25. CRC Direction Options

Option	Description
Both directions	Generates logic for generating and checking CRC. Available for both CRC-16 and CRC-32. CRC protection direction is specified independently for each port. Available on a given port only if CRC usage is enabled for that port. This setting is the default once CRC usage has been enabled.
TX direction	Generates logic for generating, but not checking, CRC. Match with a remote end of the link that has CRC only in the RX direction. Available for both CRC-16 and CRC-32. CRC protection direction is specified independently for each port. Available on a given port only if CRC usage is enabled for that port.
RX direction	Generates logic for checking, but not generating, CRC. Match with a remote end of the link that has CRC only in the TX direction. Available for both CRC-16 and CRC-32. CRC protection direction is specified independently for each port. Available on a given port only if CRC usage is enabled for that port.

Retry on Error

The SerialLite MegaCore function allows you to improve the bit error rate of your data by using the retry-on-error feature. This feature is only available on the priority data port. It provides for packets with errors to be resent so that only good packets are delivered to the Atlantic receive interface.

Downloaded from Elcodis.com electronic components distributor

Retry-on-Error Operation

When the retry-on-error feature is enabled, all packets sent by the transmitter are acknowledged by the receiver as having been received good (`ACK`) or bad (`NACK`). The packet buffers in the transmitting logic hold packets until they've been acknowledged. Once a packet has been acknowledged as received good (`ACK`), it is released from the buffer so that the buffer can be used for another packet. If a packet is acknowledged as received bad (`NACK`), then that packet and all packets sent after that packet are resent.

Up to eight packets awaiting acknowledgment can be held at once. If more packets arrive while all eight buffers are occupied, then the priority data port stalls until an acknowledgment is received, freeing up a buffer for the next packet.

The retry-on-error operation proceeds as follows:

1. When the receiver receives a good packet, the packet is delivered to the Atlantic interface and an `ACK` acknowledgment is sent back to the transmitter.

2. Any data errors cause the packet to be acknowledged bad (`NACK`). Once that happens, the receiver ignores all incoming data until it receives the resent packet.

3. All packets are numbered internally. The receiver knows which packet it expects next, so if the next expected packet has been corrupted or lost, then the next received packet has the wrong packet number, and the receiver requests a resend of the packet it was expecting.

4. Each transmitter packet buffer has an associated timer. If an acknowledgment (`ACK` or `NACK`) is lost or corrupted in transit, then the timer expires. This causes a resend of the packet in question and all subsequent packets.

5. If a lost acknowledgment is for a good packet (`ACK`), a resend occurs despite the fact that it actually is not necessary. But the receiver knows that it has already received that packet and discards the duplicate.

6. The transmitter knows which packet it expects to be acknowledged next. If the next acknowledgment is not for the expected packet, then the transmitter infers that the expected acknowledgment was lost and retransmits the packet in question and all subsequent packets.

Downloaded from Elcodis.com electronic components distributor

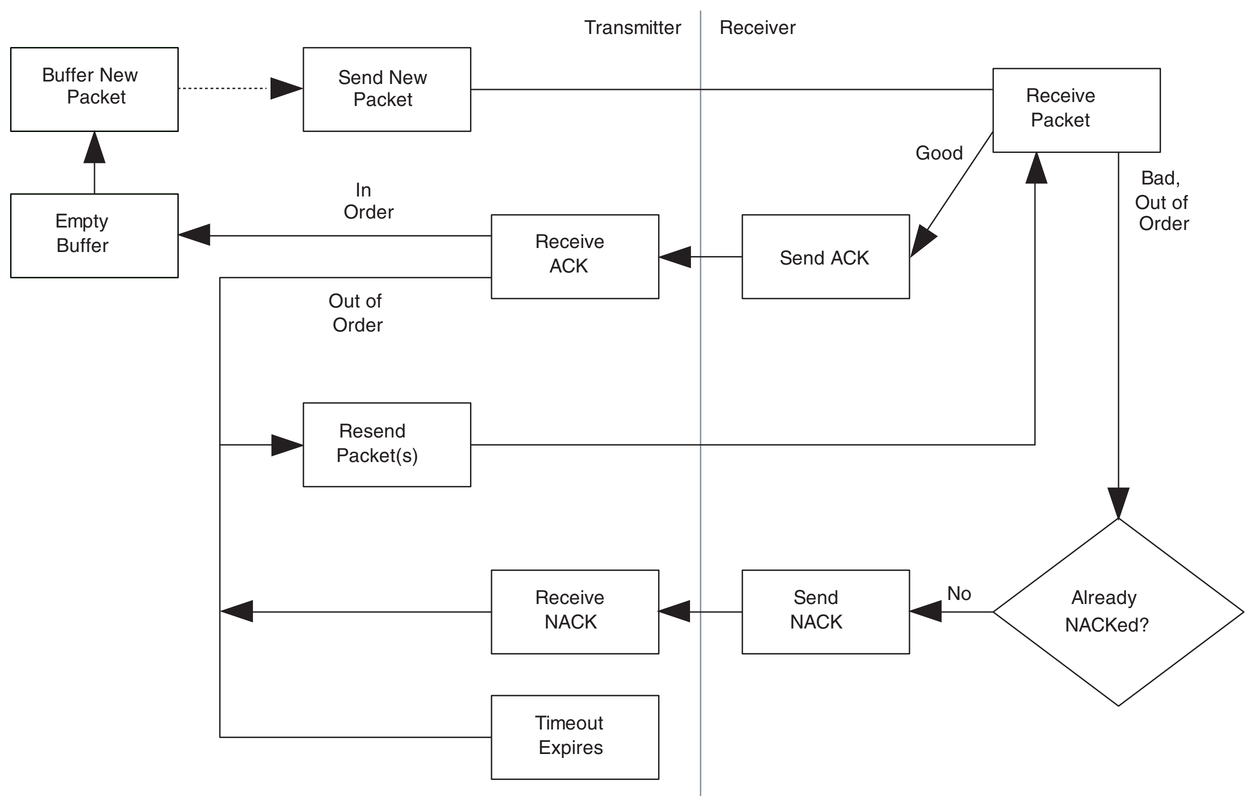
7. When a resent packet arrives at the receiver, if it also has errors, no further NACK acknowledgment is generated. This prevents bandwidth from being consumed by duplicate NACK messages. After a single NACK message, the only possible ensuing acknowledgment would be an ACK, indicating that the expected resent packet was received as good. If the resent packet is not received as good, the timer expires, causing the packet to be resent.

8. If the timer expires three times in succession, a link error is declared and the link is restarted. You can control the timeout limit in the Parameterize - SerialLite MegaCore function wizard, and it is good practice not to set the timeout to be too long so the system does not have to wait too long for such situations to resolve.

The steps outlined above ensure that exactly one good version of each packet gets delivered to the Atlantic receive port and that all packets are delivered in order. Any serious problems that make this impossible cause the link to restart.

Figure 3–25 shows the retry-on-error operation.

Figure 3–25. Retry-on-Error Operation



Downloaded from Elcodis.com electronic components distributor

If the retry-on-error feature is not selected, no packet acknowledgments are generated or expected. Only two packet buffers are created, and there are no timers associated with the packet buffers.

Table 3–26 shows the retry-on-error options for the priority data port.

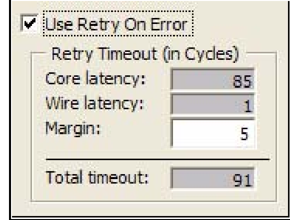
Table 3–26. Retry-on-Error Options (Priority Data Port Only)

Option	Description
Enabled	Logic is created to acknowledge packets and resend packets when errors occur. Eight transmit packet buffers with timers are created. A timeout value can be selected. Available only if the priority data port has been enabled.
Disabled	Logic is not be created to acknowledge packets. Two transmit packet buffers without timers are created. Available only if the priority data port has been enabled. This is the default setting.

Retry-on-Error Timeout Setting

Each packet buffer has a timer so that packets can be resent in the event of lost or corrupted acknowledgment packets. The timeout value must be set large enough to account for the time it takes for acknowledgments to be generated and arrive. This delay is caused by the delay through the SerialLite logic (the core latency) and latency of the link medium (the wire latency). The Parameterize - SerialLite MegaCore function wizard sets the minimum timeout value to ensure that the acknowledgments always have enough time to arrive. The wizard displays the components of the minimum timeout as shown in Figure 3–26.

Figure 3–26. Retry-on-Error Timeout Calculation



You can make the timeout longer than the minimum, although if it is too long, the link sits idle waiting for the timeout to expire. Adding a few clock cycles of margin to the minimum timeout is generally sufficient to ensure robust operation. The maximum total timeout is 2^{16} system clock cycles.

Downloaded from Elcodis.com electronic components distributor

Table 3–27 shows the margin values for the retry-on-error timeout.

Table 3–27. Retry-on-Error Timeout Margin Values

Minimum	Maximum	Default	Description
0	2^{16} -*n**	5	The number of system clock cycles beyond the required minimum that the retry-on-error circuitry waits before automatically resending data. Available only if the retry-on-error feature has been enabled. * *n* represents the required minimum timeout.

Flow Control

The SerialLite MegaCore function provides flow control as an optional means of exerting back-pressure on a data source when data consumption is too slow. Use it to ensure that the receive FIFO buffers do not overflow.

The flow control is not needed to handle the situation where two different clock sources are used on the transmitting and receiving sides of the links; clock compensation handles that without the need for flow control. Flow control is only needed when the system logic on the receiving end of the link is reading the data more slowly than the system logic on the transmitting end of the link is sending data.

If flow control is not enabled and a receive FIFO buffer overflows, a link error is declared, and the link is restarted. Link errors are described in the section "Error Handling" on page 3–54.

The flow control feature in the SerialLite MegaCore function works by having the receiving end of the link issue a `PAUSE` instruction to the transmitting end of the link when a receive FIFO buffer threshold is breached. The instruction causes the transmitter to cease transmission for a specified pause duration. Once the pause has expired, transmission resumes.

If the receive FIFO buffer is still in breach of the threshold when the pause is about to expire, the receiver automatically renews the pause in time to ensure that no data leaks out between pauses.

Downloaded from Elcodis.com electronic components distributor

Flow control suspends the flow of data through both the regular data port and the priority data port, regardless of which port had the full receive FIFO buffer.

Table 3–28 shows the flow control options.

Table 3–28. Flow Control Options

Option	Description
Enabled	Logic is created to implement flow control. The receive FIFO buffer sizing choices include threshold considerations for flow control.
Disabled	Logic is not created to implement flow control. The receive FIFO buffer sizing choices do not include any provision for setting thresholds. This is the default setting.

Table 3–29 lists the options available for configuring flow control.

Table 3–29. Flow Control Settings

Setting	Description
Backup option	Allows a backup threshold to be set as a fail-safe against lost `PAUSE` instructions.
Triggering port	Allows one or both ports to be used for triggering flow control.
Pause duration	Specifies the duration of a flow control pause.

Using a Backup PAUSE Instruction

There is a small but finite possibility that the `PAUSE` instruction sent to the transmitter may be lost or corrupted such that it won't take effect. The SerialLite MegaCore function allows you to specify a backup `PAUSE` instruction, as described in Table 3–30. This creates a second threshold above the main threshold. If the original `PAUSE` instruction takes effect, this second threshold is never breached. If the original `PAUSE` instruction does not take effect, then the second threshold is breached, and a second `PAUSE` instruction is issued. The chances are extremely remote that both `PAUSE` instructions would be corrupted (unless there was something seriously wrong with the link).

Downloaded from Elcodis.com electronic components distributor

Selecting the backup option results in a larger receive FIFO buffer and slightly more logic being used.

Table 3–30. Backup Pause Instruction Options

Option	Description
Enabled	A second FIFO buffer threshold is established, and logic is created to send a second `PAUSE` instruction. Available only if flow control is enabled.
Disabled	No second FIFO buffer threshold is established. Available only if flow control is enabled. This is the default setting.

Selecting the Trigger Port

Either or both of the data ports can be used to trigger flow control (see Table 3–31), although, once triggered, both data ports are affected by the flow control. Flow control increases the sizes of receive FIFO buffers. If both data ports are implemented, then memory usage can be reduced if your design requires only one of the ports to trigger flow control.

Table 3–31. Flow Control Trigger Options

Option	Description
Regular data port	Trigger flow control when the regular data port receive FIFO buffer breaches a threshold. Available only if flow control is enabled and the regular data port is enabled. If the priority data port is not enabled, then this is the only option allowed.
Priority data port	Trigger flow control when the priority data port receive FIFO buffer breaches a threshold. Available only if flow control is enabled and the priority data port is enabled. If the regular data port is not enabled, then this is the only option allowed.
Both data ports	Trigger flow control when either the regular data port receive FIFO buffer or the priority data port receive FIFO buffer breaches a threshold. Available only if flow control is enabled and both data ports are enabled. This is the default setting if both data ports are enabled.

Downloaded from Elcodis.com electronic components distributor

Selecting the Proper Pause Duration

Activation of flow control causes a pause in transmission whose duration you can specify in terms of pause units. Each pause unit is two system clock cycles. You can specify a pause duration from 1 to 255 pause units (equivalent to 2 to 510 clock cycles). Set the pause duration based upon your understanding of the rate at which your system logic consumes the data received. If a pause is too long, then overall system bandwidth is reduced. If a pause is too short, it may have to be renewed, which could result in an overall pause that's too long.

As an example, assume a theoretical pause needs to be 100 units long. As a designer, you would not likely know that at design time, so you have to use your judgment to pick a reasonable value. The effect of a 120-unit pause would be to cause more delay than needed. However, an 80-unit delay would result in the pause being renewed with a total of 160 units of delay, even longer than the 120-unit pause.

Table 3–32 shows the flow control pause values.

Table 3–32. Flow Control Pause Duration Values

Minimum	Maximum	Default	Description
1	255	255	The number of pause units (and half the number of) system clock cycles for which the transmitter stops sending data after it receives a `PAUSE` instruction. Available only if flow control has been enabled.

The Receive FIFO Buffers

The receive FIFO buffers are used by the receiving end of the SerialLite link to store data for presentation to the Atlantic interface and eventual consumption by the system logic. The appropriate size of the FIFO buffers depends on a number of factors. The Parameterize - SerialLite MegaCore function wizard handles these considerations automatically. The default FIFO buffers generated by the wizard work. You only need consider the following items if you wish to adjust the size of the FIFO buffers manually.

The width of the receive FIFO buffers is automatically set by the SerialLite MegaCore function at two bytes per lane. The depth is the only parameter that must be set in the wizard. The minimum FIFO buffer size depends on a number of factors, but the maximum FIFO buffer size theoretically allowed is 2^{32} entries. No device can provide that much memory, so the practical maximum is determined by the device resources available.

Downloaded from Elcodis.com electronic components distributor

Factors Affecting FIFO Buffer Size

The items described in Table 3–33 all have an impact on FIFO buffer size. As you change these parameters in the Parameterize - SerialLite MegaCore function wizard, you may notice that the FIFO buffer sizes change. The FIFO buffer sizing decisions made by the wizard are necessary for the correct operation of the link, given the choices you have made. The wizard handles the impact of these factors automatically.

Table 3–33. Factors Affecting Receive FIFO Buffer Size

Factor	Description
Flow control	If flow control is enabled, FIFO buffer sizing changes dramatically to account for the thresholds that need to be set. The entire set of sizing options on the Parameterize - SerialLite MegaCore function wizard screen depends on whether or not flow control is enabled.
Backup `PAUSE` instruction	Using a backup `PAUSE` instruction with flow control increases FIFO buffer size to accommodate the second threshold.
Pause duration	When optimizing against starvation during flow control, the pause duration affects the FIFO buffer size.
Optimizing against starvation	The FIFO buffers can be sized to reduce the amount of idle time during flow control. This setting is described in “Optimizing Against Starvation” on page 3–51.
Priority packet size	If the priority data port is enabled, an entire packet must fit in the receive FIFO buffer.
Number of priority packets to be stored	You can change the number of packets to be stored in the priority data port receive FIFO buffer.
Signal propagation delay and bit rate	The signal propagation delay and the bit rate change the wire latency, which must be accommodated if flow control is used.
Lane width	If the lane width changes, the width of the FIFO buffers changes, meaning the number of entries must change to accommodate the same amount of memory.
CRC	The use of CRC affects latency, which impacts FIFO buffer size if flow control is enabled.
Streaming vs. packet mode	This setting affects whether CRC can be utilized, which in turn affects latency.

FIFO Buffer Structure

When flow control is not used, the FIFO buffer is structured as a single block (Figure 3–27). When flow control is used, the FIFO buffer is structured as two sections, referred to as the “threshold” and the “headroom” (Figure 3–28 and Figure 3–29).

Downloaded from Elcodis.com electronic components distributor

Figure 3–27. FIFO Buffer Structure (Flow Control Not Enabled)

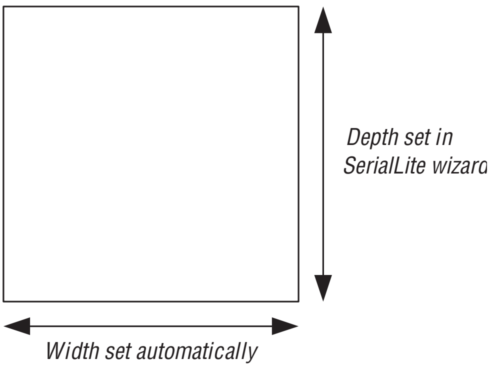
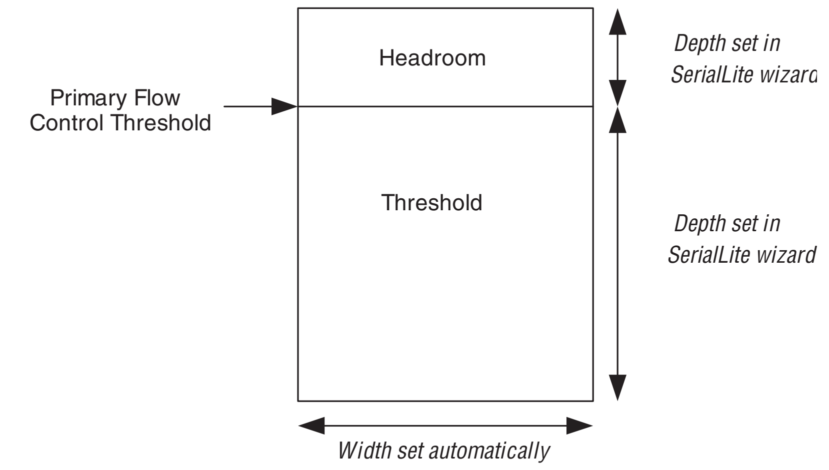
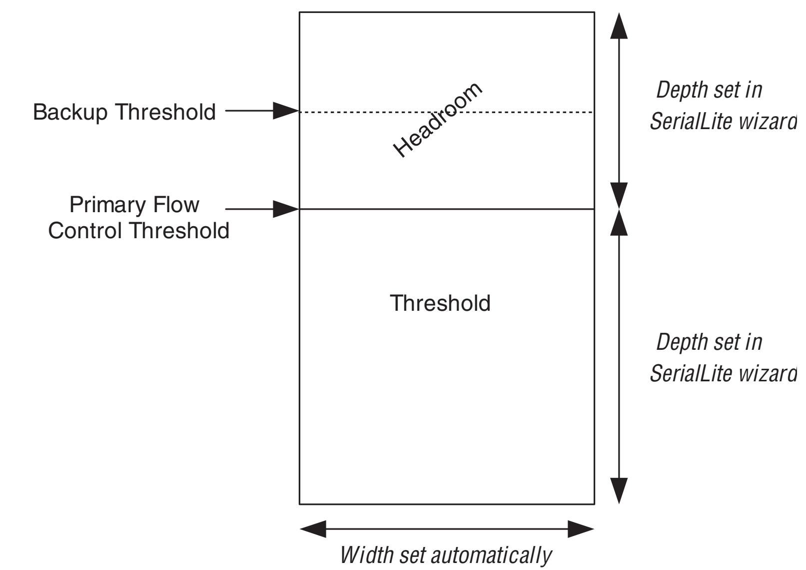


Figure 3–28. FIFO Buffer Structure (Flow Control Enabled Without Backup Pause)



Downloaded from Elcodis.com electronic components distributor

Figure 3–29. FIFO Buffer Structure (Flow Control Enabled With Backup Pause)



FIFO Buffer Thresholds

When flow control is enabled, a threshold is established for triggering flow control. You control the size of this threshold. The Parameterize - SerialLite MegaCore function wizard establishes a default that works, but you can create a larger threshold if you desire. The benefit of a larger threshold is that flow control occurs less often. The cost is the fact that more memory resources are required, and meeting performance could be more of a challenge.

If a backup pause is used, a second threshold is created (see Figure 3–29). This threshold is automatically set by the wizard, and cannot be changed. There is no reason to change it, and for that reason it is considered part of the headroom above the main threshold. The only threshold you need to control is the primary threshold that initiates flow control.

FIFO Buffer Headroom

When flow control is enabled, the SerialLite MegaCore logic monitors the triggering receive FIFO buffer, and when a threshold is reached, issues a `PAUSE` instruction. It takes some time for the `PAUSE` instruction to be issued, traverse the connection, and for transmission to be stopped. It takes more time for all data that has already been transmitted to be stored in the receive FIFO buffer.

Downloaded from Elcodis.com electronic components distributor

Therefore, there must be a certain amount of space left in the receive FIFO buffer above the threshold to hold the data that arrives during this delay. This headroom has contributions from the core latency and the wire latency. The Parameterize - SerialLite MegaCore function wizard automatically calculates the minimum headroom, displaying the components of that calculation (Figure 3–30). You can add more margin to the headroom if you wish.

If a backup `PAUSE` instruction is desired, then the minimum headroom must be effectively doubled (compare Figure 3–29 to Figure 3–28), because there must be room to accommodate the primary `PAUSE` instruction plus the backup `PAUSE` instruction, in case it's needed. Accommodations for the backup `PAUSE` instruction are automatically handled by the wizard.

Minimum & Maximum Buffer FIFO Sizes

The maximum FIFO buffer size for either port is 2^{32} entries. This is far more memory than any device can provide, meaning that there is effectively no limit to the FIFO buffer size within the bounds of available resources.

The minimum FIFO buffer size requirements are different for the two data ports. They are also heavily impacted by whether or not flow control is enabled.

Minimum Regular Data Receive FIFO Buffer
If flow control is not enabled, then the minimum regular data port receive FIFO buffer size is 16 entries (see Table 3–34).

If flow control is enabled, then there is no minimum threshold size (see Table 3–35). However, you should pick a reasonable number to avoid frequent triggering of flow control. The minimum headroom is automatically calculated by the Parameterize - SerialLite MegaCore function wizard based on core and wire latencies.

Table 3–34. Regular Data Receive FIFO Buffer Size Values (No Flow Control)

Minimum	Maximum	Default	Description
16	2^{32}	16	Determines the number of entries that are built into the regular data receive FIFO buffer. Available only if the regular data has been enabled. Applies only if flow control is disabled or the regular data has not been selected as a trigger for flow control.

Downloaded from Elcodis.com electronic components distributor

Table 3–35. Regular Data Receive Threshold & Headroom Margin Values (Flow Control)

Minimum	Maximum	Default	Description
0	2^{32} -*n**	5	The number of regular data receive FIFO buffer entries beyond the required minimum that are provided for the threshold and headroom. The threshold and headroom margins are independent. Available only if the data port has been enabled. Applies only if flow control is enabled and the regular data port has been selected as a trigger for flow control. * *n* represents the required minimum depth.

Minimum Priority Data Receive FIFO Buffer

The priority data port receive FIFO buffer must always be able to hold an entire packet. This requirement is driven by the need to avoid deadlock when both retry-on-error and flow control are used, but is always enforced to simplify the SerialLite MegaCore function design.

If flow control is not enabled, then the minimum priority data port receive FIFO buffer size is the size of one maximum priority packet but not less than 16 entries (see Table 3–36).The number of entries required for this depends on the number of lanes. This minimum is automatically calculated by the Parameterize - SerialLite MegaCore function wizard.

If flow control is enabled, then the minimum threshold size must be large enough to hold a maximum priority packet (see Table 3–37). The number of entries required for this depends on the number of lanes. This minimum is automatically calculated by the wizard. The minimum headroom is also automatically calculated by the wizard based on core and wire latencies (see Figure 3–30 on page 3–53).

Downloaded from Elcodis.com electronic components distributor

Creating FIFO Buffers Larger Than Minimum

To reduce the incidence of flow control, or to increase margin against overflow if flow control is not used, the receive FIFO buffers can be easily enlarged by adding margin to the minimums established by the Parameterize - SerialLite MegaCore function wizard. If flow control is not used, the size of the regular data FIFO buffer is increased by directly changing the size; the priority port FIFO buffer is increased by changing the margin. If flow control is used, then the threshold and headroom are adjusted by changing their respective margins.

Table 3–36. Priority Data Receive FIFO Buffer Size Margin Values (No Flow Control)

Minimum	Maximum	Default	Description
0	2^{32} -*n**	5	The number of priority data receive FIFO buffer entries beyond the required minimum that are provided in the priority data receive FIFO buffer. Available only if the priority data port has been enabled. Applies only if flow control is disabled or the priority data port has not been selected as a trigger for flow control. * *n* represents the required minimum depth.

Table 3–37. Priority Data Receive Threshold & Headroom Margin Values (Flow Control)

Minimum	Maximum	Default	Description
0	2^{32} -*n**	5	The number of priority data receive FIFO buffer entries beyond the required minimum that are provided for the threshold and headroom. The threshold and headroom margins are independent. Available only if the priority data port has been enabled. Applies only if flow control is enabled and the priority data port has been selected as a trigger for flow control. * *n* represents the required minimum depth.

Optimizing Against Starvation

When flow control is used, it is useful to balance the pause duration against the threshold size. If a long pause duration is used with a FIFO buffer that has a very small threshold, then flow control is easily triggered, and the FIFO buffer is likely emptied out long before the pause has expired. The FIFO buffer is therefore “starved,” since it has room for more data but data transmission is still suspended. This is wasteful of bandwidth, since the link is idle for no reason.

Downloaded from Elcodis.com electronic components distributor

The size of the FIFO buffers can be optimized to avoid the situation where a FIFO buffer is sitting empty with no data feeding it. The **Optimize FIFO buffer size to avoid starvation** option automatically sets a threshold size that minimizes this risk. Components that are taken into account are the core and wire latencies and the pause duration. These are automatically calculated by the Parameterize - SerialLite MegaCore function wizard, and the components of the calculation are displayed (see Figure 3–31).

Because the priority data receive FIFO buffer may have a large minimum size, determined by the maximum packet size, the FIFO may already be larger than required for avoiding starvation if the pause duration is short. Alternatively, optimizing against starvation may by itself create a FIFO buffer larger than the minimum required by the packet size. The wizard automatically creates the smallest FIFO buffer that satisfies both the minimum size requirement and the starvation setting for the priority receive FIFO buffer.

Table 3–38 shows the starvation optimization options.

Table 3–38. Starvation Optimization Options

Option	Description
Enabled	The FIFO buffer threshold sizing is automatically increased to avoid starvation during flow control. Available only if flow control is enabled. This is the default setting.
Disabled	The FIFO buffer threshold sizing is not increased to avoid starvation during flow control. Available only if flow control is enabled.

Storing Additional Priority Packets

A very simple way to increase the size of the priority data port receive FIFO buffer is to size it to hold multiples of the minimum packet size using the Parameterize - SerialLite MegaCore function wizard (see Table 3–39). There is no theoretical limit to the number of packets that can be selected, but in practice this can increase memory usage quickly, and make it harder to achieve performance.

Downloaded from Elcodis.com electronic components distributor

If flow control is disabled, this setting affects the FIFO buffer size directly. If flow control is enabled, this setting affects the size of the threshold.

Table 3–39. Number of Packets Stored in FIFO Buffer Values (Priority Data Port Only)

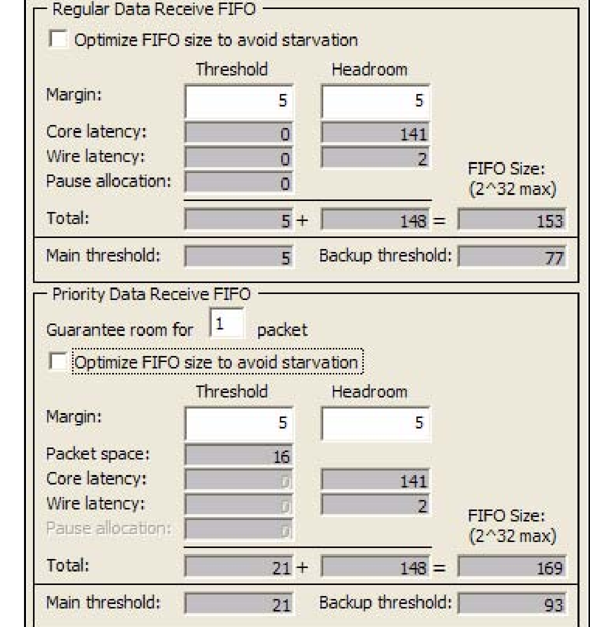
Minimum	Maximum	Default	Description
1	N/A	1	Ensures that the FIFO buffer (if flow control is disabled) or the FIFO buffer threshold (if flow control is enabled) is large enough to hold the specified number of packets. Available only if the priority data port is enabled.

FIFO Buffer Size

The sizes of the FIFO buffers, when flow control is enabled, along with the calculations of those sizes are displayed in the Parameterize - SerialLite MegaCore function wizard (Figure 3–30 and Figure 3–31). Figure 3–30 shows the case where optimization against starvation has not been selected. In this example, the maximum packet size determines the priority data receive FIFO buffer size.

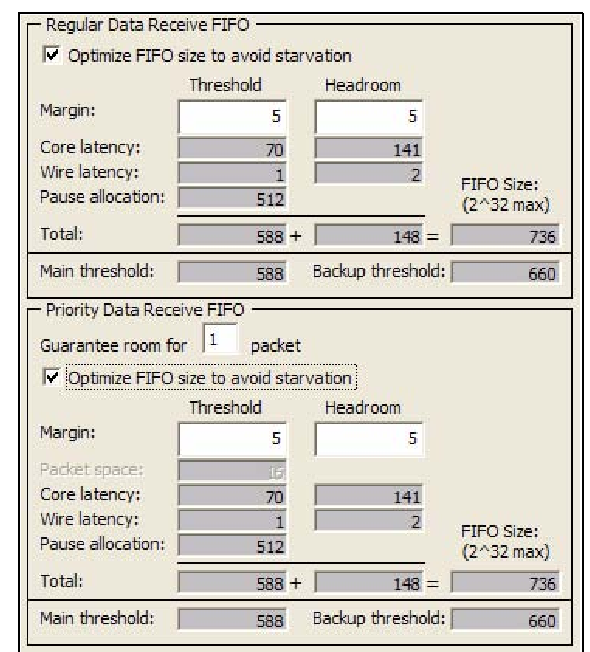
Figure 3–31 shows the case where optimization against starvation has been selected. In this case, the priority data receive FIFO buffer size is dominated by the starvation setting, creating a large enough FIFO buffer that the packet size has no impact.

Figure 3–30. FIFO Buffer Sizing (Flow Control Without Starvation Option)



Downloaded from Elcodis.com electronic components distributor

Figure 3–31. FIFO Buffer Sizing (Flow Control With Starvation Option)



Error Handling

The SerialLite MegaCore function provides error-checking capabilities and an interface for observing local errors. The error types are categorized, and the effect of an error depends on the kind of error encountered.

Error Types

The SerialLite MegaCore function has three levels of error:

- Data error
- Link error
- Catastrophic error

Downloaded from Elcodis.com electronic components distributor

The causes and results of these errors are summarized in Table 3–40.

Table 3–40. Error Causes & Results

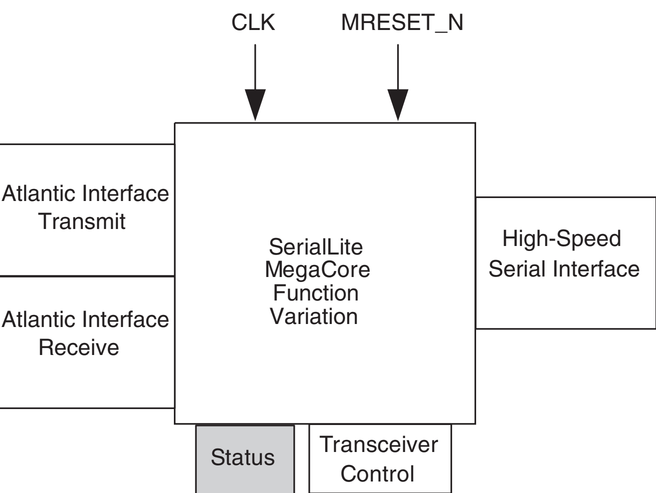
Error Type	Causes	Effects
Data	• Invalid 8B/10B code detected • Running disparity error • CRC error (if CRC implemented) • Packet marked bad by transmitting system logic asserting the `TERR` signal on the Atlantic interface (regular data port only).	For a packet being delivered to the regular data port, the `RERR` signal on the Atlantic interface is asserted for the packet. For a packet being delivered to the priority data port without the retry-on-error feature, the `RHERR` signal on the Atlantic interface is asserted for the packet. For a packet being delivered to the priority data port with the retry-on-error feature, the packet is acknowledged as bad, causing the packet to be resent. The `RHERR` signal does not exist if the retry-on-error feature is enabled.
Link	• FIFO memory overflow (Includes the regular and priority data ports, and the clock compensation sequence removal buffer) • Loss of character alignment • Loss of lane alignment • Loss of signal • Too many data errors in a short amount of time • A restart sequence is detected from the other end of the link	The link is brought down and restarted.
Catastrophic	• Reverse polarity detected and automatic polarity reversal not implemented • Reverse lane order detected and automatic lane reversal not implemented • Scrambled lane order detected	Unrecoverable. The link does not operate.

Downloaded from Elcodis.com electronic components distributor

Status Interface

The SerialLite MegaCore function provides a separate interface for accessing error information, as shown in Figure 3–32. The information provided through this interface reflects only the local side of the link. The status of the remote end of the link is not reflected on the local status interface because there is no mechanism to get the error information from the remote end of the link to the local end of the link.

Figure 3–32. Status Interface



Much of the logic that generates these signals exists in the recovered clock domain, but the signals are displayed in the system clock domain. However, there is no FIFO memory to buffer the domain crossing. Instead, most of the signals are metastability-hardened, and incur a two-three-cycle latency.

Downloaded from Elcodis.com electronic components distributor

The status interface is detailed in Table 3–41.

Table 3–41. Status Interface Signals

Name	Direction	Description
`STATUS_PORT[0]`	Output	Link up. Indicates that the local side of the link has been successfully initialized and is running. It is generated in the recovered clock domain internally, but is displayed in the system clock domain. Because the signal crosses domains and is metastability-hardened, there is a two- to three-cycle latency for the signal to be asserted and deasserted.
`STATUS_PORT[1]`	Output	Catastrophic error. Indicates that a catastrophic error was detected. This signal is asserted the clock cycle after the error is detected. It is generated in the recovered clock domain internally, but is displayed in the system clock domain. Because this condition causes the link to go into an unrecoverable state, this signal is not metastability-hardened.
`STATUS_PORT[2]`	Output	Link error. Indicates that a link error was detected. This signal is asserted high for one clock cycle when the link goes down. It is generated in the recovered clock domain internally, but is displayed in the system clock domain. Because the signal crosses domains and is metastability-hardened, there is a two- to three-cycle latency for the signal to be asserted and deasserted.
`STATUS_PORT[3]`	Output	Data error. Indicates that a data error was detected. This signal is asserted high for one clock cycle for each data error detected. Within one clock cycle, a single pulse occurs regardless of the number of errors occurring on the data bus during that clock cycle. It is generated in the recovered clock domain internally, but is displayed in the system clock domain. Because the signal crosses domains and is metastability-hardened, there is a two- to three-cycle latency for the signal to be asserted and deasserted. In addition, because of the domain crossing, two consecutive pulses may occasionally merge into a single pulse. For this reason, use this signal as a general indicator of the frequency of errors, but not to collect accurate statistics.
`STATUS_PORT[4]`	Output	Oversize packet discarded. Used only when the priority data port is enabled, and when priority packet testing has been enabled. Indicates that an oversize packet was received at the priority data port and was discarded. The signal is asserted high for one clock cycle starting one clock cycle after the oversize packet was detected. It is generated and displayed in the system clock domain, and does not cross a domain boundary.
`STATUS_PORT[15..5]`	Output	Reserved. Bit 5 is internally connected to the resynchronized `MRESET_N` signal. The other bits are tied to zero.

Downloaded from Elcodis.com electronic components distributor

Transceiver Settings

The transceivers used for the `TX_OUT` and `RX_IN` signals can be configured to adjust the electrical characteristics of the signals on the differential pairs. Table 3–42 describes the different characteristics that can be adjusted.

Table 3–42. Transceiver Settings

Setting	Description
Transmitter PLL bandwidth	Adjusts the responsiveness of the transmitter PLL to frequency changes
Receiver PLL bandwidth	Adjusts the responsiveness of the receiver PLL to frequency changes
Termination	Adjusts the termination on the transmitter differential pair
V_{OD}	Adjusts the output differential voltage
Pre-emphasis	Adjusts the amount of pre-emphasis given to the transmitted signal
Equalization	Adjusts the amount of equalization applied to the received signal
Signal loss detection	Adjusts how lost signals are handled

Transmitter & Receiver PLL Bandwidth

The Stratix GX transmitter PLL and receiver PLL in the transceiver block offer a programmable bandwidth setting. The PLL bandwidth is the measure of its ability to track the input clock and jitter. It is determined by the -3-dB frequency of the closed-loop gain of the PLL.

A high-bandwidth setting provides a faster lock time and tracks more jitter on the input clock source, which passes it through the PLL. This helps reject noise from the voltage-controlled oscillator (VCO) and power supplies. A low-bandwidth setting, on the other hand, filters out more high frequency input clock jitter, but increases lock time.

Downloaded from Elcodis.com electronic components distributor

PLL bandwidth settings are made in the Parameterize - SerialLite MegaCore function wizard. The -3-dB frequencies for these settings can vary due to the non-linear nature and frequency dependencies of the circuit. You can adjust the bandwidth to fine tune and customize the performance on specific systems. Tables 3–43 and 3–44 show the transmitter and receiver PLL bandwidth settings.

Table 3–43. Transmitter PLL Bandwidth Options
Low (default)
High

Table 3–44. Receiver PLL Bandwidth Settings
Low (default)
Medium
High

Transmitter Termination

The Stratix GX transmitter buffer allows you to program the on-chip differential termination resistor (see Table 3–45). The transmitter buffers are current-mode drivers, so the output differential voltage (V_{OD}) depends on the transmitter termination value. The Parameterize - SerialLite MegaCore function wizard allows you to make this selection.

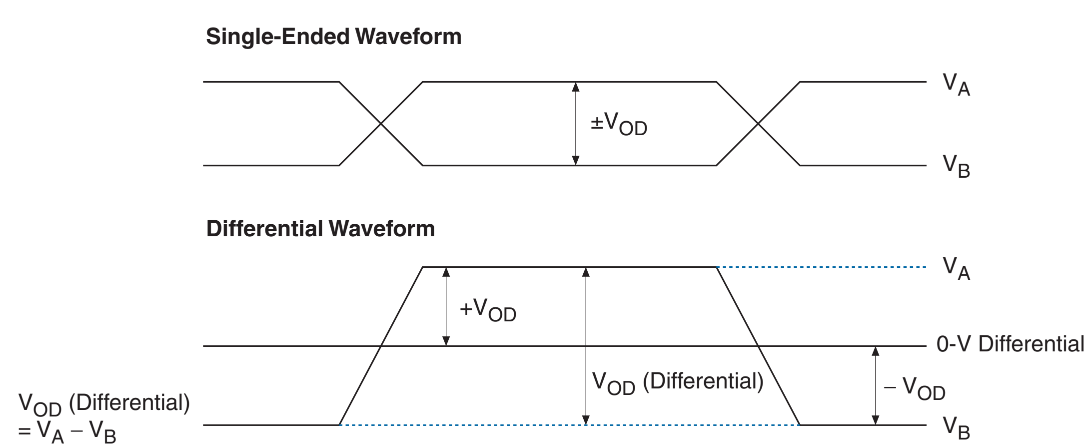
Table 3–45. Transmitter Termination Options
100 Ω (default)
120 Ω
150 Ω

Output Differential Voltage (V_{OD})

Stratix GX transceivers allow you to customize the output differential voltage (V_{OD}) to handle different length, backplane, and receiver requirements. V_{OD} is illustrated in Figure 3–33.

Downloaded from Elcodis.com electronic components distributor

Figure 3–33. V_{OD} (Differential) Signal Level



You can set the V_{OD} values statically during configuration or you can adjust them dynamically while the device is operating as shown in Table 3–46. You make this choice using the Parameterize - SerialLite MegaCore function wizard.

Table 3–46. V_{OD} Control Settings

Option	Description
Use fixed V_{OD}	A single static selected value is used for the V_{OD} for all lanes. This is the default setting.
Use V_{OD} signal	The `TX_VODCTRL` bus is created, with three bits per lane. The V_{OD} value is determined by the value placed on these signals, according to Table 3–45.

You can select the static V_{OD} value using the Parameterize - SerialLite MegaCore function wizard. The advantages of a static V_{OD} value are simplicity and fewer signals to be routed. The disadvantage of this mode is that the V_{OD} is set identically for all lanes, and cannot be changed without regenerating another programming file.

Downloaded from Elcodis.com electronic components distributor

Alternatively, if dynamic adjustment is selected, you can dynamically configure the V_{OD} setting while the device is operating. You can set the V_{OD} value by asserting encoded values on the `TX_VODCTRL` bus, which is instantiated in the transceiver control interface when you select this option. This configuration allows you to make quick performance evaluations of the various settings without needing to compile and regenerate multiple configuration files. Another advantage of this option is that it allows the V_{OD} of each lane to be configured independently.

Table 3–47 shows the V_{OD} settings available for each of the transmitter termination options, as well as the encoded values to be used for dynamic adjustment.

Table 3–47. V_{OD} Settings & Encoded Values

100 Ω (mV)	120 Ω (mV)	150 Ω (mV)	TX_VODCTRL[2..0]
400	480	600	000
800	960	1200	001
1000 (default)	1200	1500	010
1200	1440		011
1400			100
1600			101

The V_{OD} for the transmitter buffer cannot exceed 1600 mV. This voltage is the saturation point of the transmitter buffer. Settings beyond this value do not damage the buffer, but prevent the operation of the device from being represented accurately.

Pre-Emphasis & Equalization

The programmable pre-emphasis module in each transmit buffer boosts the high frequencies in the transmit data signal that may be attenuated in the transmission medium. This maximizes the data eye opening at the far-end receiver. Pre-emphasis is particularly useful in lossy transmission media.

Downloaded from Elcodis.com electronic components distributor

The transfer function of a transmission line can be represented in the frequency domain as a low-pass filter. Any frequency components below the -3-dB frequency pass through with minimal losses. Frequency components that are greater than the -3-dB frequency are attenuated. This variation in frequency response yields data-dependant jitter and other inter-symbol interference (ISI) effects. By applying pre-emphasis, the high-frequency components are boosted, or in other words, pre-emphasized, when transmitted. By applying equalization, the low-frequency components are attenuated. This equalizes the frequency response as seen at the receiver so that the difference between the low-frequency and high-frequency components is reduced, which in return minimizes the ISI effects from the transmission medium.

The programmable pre-emphasis settings can have one of six values. The programmable equalizer settings can have one of five values. You should experiment with the pre-emphasis and equalization values to determine the optimal settings based on your system variables.

You can set the pre-emphasis and equalization settings statically during configuration or you can adjust them dynamically while the device is operating, as shown in Tables 3–48 and 3–49. You make this choice using the Parameterize - SerialLite MegaCore function wizard.

Table 3–48. Pre-Emphasis Control Settings

Option	Description
Use fixed pre-emphasis	A single static selected value is used for the pre-emphasis for all lanes. This is the default setting.
Use pre-emphasis signal	The `TX_PREEMPHASISCTRL` bus is created, with three bits per lane. The pre-emphasis value is determined by the value placed on these signals, according to Table 3–50.

Table 3–49. Equalization Control Settings

Option	Description
Use fixed equalization	A single static selected value is used for the equalization for all lanes. This is the default setting.
Use equalization signal	The `RX_EQUALIZATIONCTRL` bus is created, with three bits per lane. The equalization value is determined by the value placed on these signals, according to Table 3–51.

Downloaded from Elcodis.com electronic components distributor

The advantages of a static pre-emphasis or equalization value are simplicity and fewer signals to be routed. The disadvantage of this mode is that the selected value is set identically for all lanes and cannot be changed without regenerating another programming file.

Alternatively, if dynamic adjustment is enabled in the wizard, you can dynamically configure the pre-emphasis or equalization setting while the device is operating. This configuration is done by asserting encoded values on the `TX_PREEMPHASISCTRL` and `RX_EQUALIZATIONCTRL` buses, which are instantiated in the transceiver control interface when you select this option. The encoded values are shown in Tables 3–50 and 3–51. This configuration allows you to make quick performance evaluations of the various settings without needing to compile and regenerate multiple configuration files. Another advantage of this option is that it allows the pre-emphasis or equalization of each lane to be configured independently of the other lanes.

Table 3–50. Pre-Emphasis Encoded Values

Pre-emphasis setting	TX_PREEMPHASISCTRL[2..0]
0 (default)	000
1	001
2	010
3	011
4	100
5	101

Table 3–51. Equalization Encoded Values

Equalization setting	TX_EQUALIZATIONCTRL[2..0]
0 (default)	000
1	010
2	100
3	101
4	111

Avoid pre-emphasis settings that, together with the V_{OD} setting, yield a value greater than 1600 mV. Settings beyond this value do not damage the buffer, but prevent the operation of the device from being represented accurately.

Downloaded from Elcodis.com electronic components distributor

Signal Detection & Signal Lost Threshold

The signal loss threshold detector senses whether the specified voltage level exists at the receiver buffer. This detector has a hysteresis response that filters out any high-frequency ringing that might be caused by inter-symbol interference (ISI) or any high-frequency losses in the transmission medium. If used, this feature allows the SerialLite MegaCore function to decide whether or not valid signals are present on the inputs

Each lane has a programmable signal lost threshold differential voltage level. Table 3–52 shows the four supported signal lost threshold settings, which are set in the Parameterize - SerialLite MegaCore function wizard. When the signal detector does not detect the signal, the SerialLite link is restarted.

Table 3–52. Signal Lost Threshold Options (mV)

530 (default)
700
740
840

If you have an environment where the voltage thresholds might not meet the lowest voltage threshold setting, you can disable the signal detection module in the wizard, which makes the SerialLite MegaCore function operate like there is always a signal available. Loss of signal is still detected through other means, such as data errors or the inability to initialize the link.

Table 3–53 describes the signal detection options.

Table 3–53. Signal Detection Options

Option	Description
Enabled	Signals with a differential voltage of less than 530 mV is considered lost, and a link error occurs.
Disabled	The internal flag defaults to indicate that a signal is always present. This is the default setting.

Downloaded from Elcodis.com electronic components distributor

Transceiver Control Interface

The SerialLite MegaCore function provides an interface to allow you to control the V_{OD}, pre-emphasis, and equalization settings of each lane individually and dynamically. You have the independent option for each of these characteristics to set the values either at configuration time or dynamically via a three-bit-per-lane bus. If you select a static setting in the Parameterize - SerialLite MegaCore function wizard, the control signals are not created. All lanes have the same settings.

The transceiver control interface (see Figure 3–34) gives you access to the signals needed to control the transceiver dynamically. If you select dynamic control, a three-bit bus is created per characteristic per lane. The values to be placed on these signals to achieve the desired settings are in Tables 3–47, 3–50, and 3–51.

Figure 3–34. Transceiver Control Interface

If, for example, you select dynamic control of pre-emphasis on a 4-lane link, 3 bits are created for each lane, resulting in a 12-bit bus for pre-emphasis control. The least-significant 3 bits (2 - 0) correspond to the controls for lane 1. The most-significant bits (11 - 9) correspond to the controls for lane 4. In the latter case, the settings for lane four would be read from the tables as if bits [11..9] were bits [2..0], so that a pre-emphasis setting of 1 for lane 4 would be achieved by placing 001 on bits [11..9].

Downloaded from Elcodis.com electronic components distributor

The control buses are described in detail in Table 3–54. These signals can be used as if asynchronous, although their effects cannot be observed if the link is not operating. The signals are intended for manual adjustment, not real-time changes, so instantaneous response should not be expected.

Table 3–54. Transceiver Control Interface Signals

Name	Direction	Description
TX_VODCTRL[lanes*3-1..0]	Input	V_{OD} control bus. Only instantiated if dynamic control for V_{OD} is selected. Three bits are assigned to each lane, with the least-significant bits (that is, [2..0]) corresponding to lane 1. Each three-bit group should have its values set per Table 3–47.
TX_PREEMPHASISCTRL[lanes*3-1..0]	Input	Pre-emphasis control bus. Only instantiated if dynamic control for pre-emphasis is selected. Three bits are assigned to each lane, with the least-significant bits (that is, [2..0]) corresponding to lane 1. Each three-bit group should have its values set per Table 3–50.
RX_EQUALIZATIONCTRL[lanes*3-1..0]	Input	Equalization control bus. Only instantiated if dynamic control for equalization is selected. Three bits are assigned to each lane, with the least-significant bits (that is, [2..0]) corresponding to lane 1. Each three-bit bus should have its values set per Table 3–51.
RX_SLPBK	Input	This signal dynamically enables the serial loopback on a channel-by-channel basis. When the RX_SLPBK signal is high, all blocks that are active when the signal is low remain active. Although serial loopback is enabled, data is output on the TX_OUT[] signal. When this pin is set, the data on TX_OUT[] is looped back onto the RX_IN[] signal, overriding any user data.

Optimizing the Implementation

There are a number of steps that can be taken to optimize your design, depending on your goals. The features selected in your SerialLite configuration have a substantial impact on both resource utilization and performance. Because of the large number of different combinations of options that are available, it is impossible to characterize in general how fast or how large a design will be. In addition, the performance of a SerialLite link in isolation is different from the performance of the same link instantiated alongside large amounts of other logic in a Stratix GX device.

Downloaded from Elcodis.com electronic components distributor

For the most part, the steps you take to improve performance or resource utilization are similar to the steps you would take for any other design. The following suggestions are intended to provide ideas, but should not be considered an exhaustive list.

Improving Performance

Performance is the factor that depends most on what other logic exists in the device. If SerialLite is competing with other logic for routing resources, inefficient routing could compromise speed. The following sections describe some things that can be considered if speed is an issue.

Optimize for Speed Versus Optimize for Size

The Parameterize - SerialLite MegaCore function wizard provides an option for gaining some speed at the expense of some resources. The amount of improvement varies, but the resource cost is modest, so if you are having trouble meeting performance, ensure this option is set for speed.

Performance becomes more of a challenge for links with many lanes, whereas conservation of resources tends to be more important for smaller links. Therefore, in designs involving four or more lanes, the wizard sets the default to optimize for speed; for designs having fewer than four lanes, the default is set to optimize for size. Table 3–55 shows the optimization settings.

Table 3–55. Optimization Settings

Option	Description
Optimize for speed	Various subtle aspects of the design use more flip-flops to provide higher speed. This is the default for designs with four or more lanes.
Optimize for size	Various subtle aspects of the design use fewer flip-flops to conserve resources. This is the default for designs with fewer than four lanes.

Downloaded from Elcodis.com electronic components distributor

Feature Selection

The following features impact speed more significantly. Your system may require some of these, but if any are optional or can be reconsidered, this may help your performance. Before making any changes, verify that the feature you want to change is in the critical speed path.

- Lane count: running more lanes more slowly reduces the operating frequency required (but uses more logic resources).
- Packet mode: streaming mode operates faster than packet mode. In general, streaming mode on the regular data port gives you the fastest, smallest implementation.
- CRC: the CRC generation and checking logic degrade performance and latency. In particular, if you are using CRC-32, evaluate carefully whether the extra protection over CRC-16 is really worthwhile, because CRC-16 has less impact on speed.
- Retry on error: the buffering and acknowledgment mechanisms can impact speed.
- Receive FIFO buffer size: large FIFO buffers increase fanout and may require longer routing to extend further inside the device.

Running Different Seeds

If your first attempt at hitting performance is close but is not quite enough, try running different placement seeds. This often yields a better result. The Quartus® II Design Space Explorer is also a useful tool for improving speed. Refer to your Quartus II documentation for more information on seed specification and the Design Space Explorer.

Limiting Fanout

Depending on the number of lanes and the size of memories chosen, fanout can become significant. Limiting the fanout during synthesis causes replication of high-fanout signals, improving speed. If high-fanout signals are the critical path, limiting the fanout allowed can help. Refer to your Quartus II documentation for more information on limiting fanout.

Floorplanning

The SerialLite MegaCore function does not come with any placement constraints. The critical paths depend on where in the device the SerialLite logic is placed as well as the other logic in the device. Standard floorplanning techniques can be used to improve performance. Refer to your Quartus II documentation for more information on floorplanning.

Downloaded from Elcodis.com electronic components distributor

Minimizing Logic Utilization

The amount of logic required for a SerialLite link depends heavily on the features chosen. The Parameterize - SerialLite MegaCore function wizard displays an estimate of LE usage to help you select features with visibility into the impact on LE utilization. This display is only an estimate and does not necessarily change with every feature setting change; if you want a more accurate measure of the logic required for your configuration, you must synthesize the design.

The following features have a significant impact on logic usage:

- Lane count: running fewer lanes at higher bit rates, if possible, uses less logic (but places more of a burden on meeting performance).
- CRC: significant savings can be made by eliminating CRC, or in particular, moving from CRC-32 to CRC-16 in high-lane-count designs. If you are using CRC-32, evaluate carefully whether the extra protection over CRC-16 is really worthwhile, because CRC-16 uses far fewer resources.
- Port selection: having two ports instantiated uses significantly more logic than having one port. If you don't need the nesting or retry-on-error features of the priority data port, consider using channel multiplexing on the regular data port to manage multiple streams without using the priority data port. Alternatively, if you require the retry-on-error feature and can forego packet nesting, you can eliminate the regular data port and use only the priority data port, with channel multiplexing allowing you to differentiate your data streams.
- Polarity reversal: if you have confidence that the SerialLite link connection medium has the correct differential pair polarity, opting to test polarity but not reverse it saves logic.
- Packet mode: packet mode requires more logic than streaming mode for data encapsulation. In general, streaming mode on the regular data port gives you the fastest, smallest implementation.
- Retry on error: this feature requires logic to generate acknowledgments and keep track of the packet flow.
- Flow Control: this feature requires logic to monitor the FIFO buffer levels and to generate and act upon `PAUSE` instructions.

Downloaded from Elcodis.com electronic components distributor

Minimizing Memory Utilization

The amount of memory required for a SerialLite link depends heavily on the features chosen. To obtain a measure of the memory required for your configuration, you must synthesize the design.

The following features have a significant impact on memory usage:

- Lane count: this establishes the bus widths internally, and most memories used scale almost directly with the number of lanes selected. Running fewer lanes at higher bit rates, if possible, uses less memory (but places more of a burden on meeting performance).
- Port selection: having two ports instantiated uses significantly more memory than having one port. If you don't need the nesting or retry-on-error features of the priority data port, consider using channel multiplexing on the regular data port to manage multiple streams without using the priority data port. Alternatively, if you require the retry-on-error feature and can forego packet nesting, you can eliminate the regular data port and use only the priority data port, with channel multiplexing allowing you to differentiate your data streams.
- Receive FIFO buffer size: you can minimize memory usage by not adding significant amounts of margin to the minimum specified sizes, or by not making room for more than one priority packet (if the priority data port is being used).
- Flow control: this feature requires larger receive FIFO buffers.
- Packet size: if you are using the priority data port, you can reduce the size of your receive FIFO buffers and the amount of buffer space required by reducing the maximum size of your priority packets.
- Optimizing against starvation: this feature makes long pauses more efficient by creating a larger receive FIFO buffer so that the receiver has enough data to process during the pause. Turning off this feature provides a smaller FIFO buffer, at the risk of less efficient operation, depending on your application. If possible, reducing the pause duration can restore some efficiency.
- Pause duration: when you wish to optimize against starvation, you can reduce the size of the receive FIFOs needed by shortening the pause duration.
- Retry-on-error: this feature requires eight packet buffers, as compared to two packet buffers if the feature is not used.

Initialization & Restart

Before the SerialLite link can operate, the MegaCore function must properly reset the gigabit transceiver. The SerialLite MegaCore function must then be initialized and trained. The SerialLite training sequence can

Downloaded from Elcodis.com electronic components distributor

generally bring the link up in a few hundred microseconds; the actual amount of time required varies according to PLL lock times, the number of lanes, the per-lane deskew, and other variation-specific factors.

A link only restarts on its own due to a link error encountered during operation. A hardware reset using the `MRESET_N` signal also causes the link to restart once the reset has been deasserted.

When one end of the link is brought down by either of these means, it brings the other end down by sending training sequences to the other end of the link. The other end of the link restarts once it sees eight successive training sequences.

While the initial training sequences are being received, they may be processed through the SerialLite logic and presented at the Atlantic interface. The result is that when a link error occurs, a few cycles of meaningless data may be consumed just prior to the link restarting.

Downloaded from Elcodis.com electronic components distributor

Downloaded from Elcodis.com electronic components distributor

4. SerialLite Testbench

General Description

You can simulate your design using IP Toolbench-generated VHDL and Verilog HDL IP 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*.

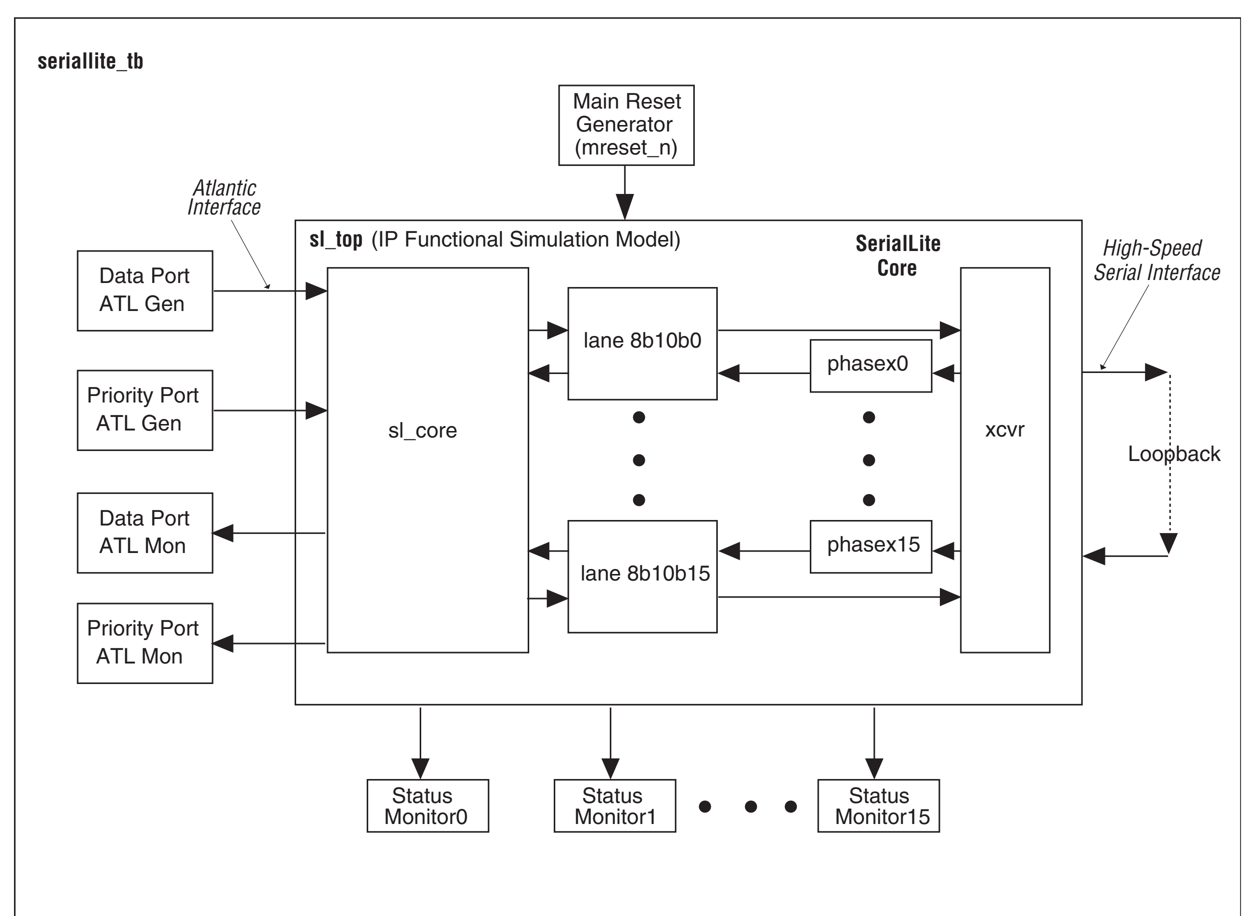
Altera provides models you can use for functional verification of the SerialLite MegaCore® function within your design. A Verilog HDL demonstration testbench, including scripts to run it, is also provided. This demonstration testbench, used with the ModelSim®-Altera simulator tool, demonstrates how to instantiate a model in a design. The demonstration testbench stimulates the inputs and checks the outputs of the interfaces of the SerialLite MegaCore function, demonstrating basic functionality.

Testbench Environment

The testbench (`seriallite_tb`) environment shown in Figure 4–1 generates traffic through the Atlantic™ port generators (`atl_gen`), sends it through the SerialLite logic, loops the data back on the high-speed serial interface, back into the receive side of the logic, and then checks the data as received at the Atlantic interface (`atl_mon`). The SerialLite status bus is monitored throughout the duration of the testbench by the status monitor (`stat_mon`).

Downloaded from Elcodis.com electronic components distributor

Figure 4–1. SerialLite Testbench Environment



Note to Figure 4–1:

(1) The main reset generator shown in the figure simply drives the `MRESET_N` core input with a synchronous (to CLK) reset.

Methodology Overview

There are three basic steps to using the SerialLite testbench.

1. Create a SerialLite link configuration using the Parameterize - SerialLite MegaCore function wizard.

2. Optionally modify the list of simulation parameters in the *<variation name>*`_tb_params.txt` file to reflect the simulations you wish to perform.

3. Execute the run `do` *<variation name>*`_tb.do` file using ModelSim.

Downloaded from Elcodis.com electronic components distributor

The <*variation name*>`_tb_params.txt` file contains the parameters that control the types of data, number of packets, and the size of packets. You edit the values in this file to set up the specific simulation you wish to perform. Changing the other parameters in the <*variation name*>`_tb_params.txt` file may result in simulation failure.

The SerialLite testbench performs the following tests, if applicable:

- The testbench waits for the main reset sequence to end.
- The testbench waits for the SerialLite link to come up.
- If the regular data port is enabled, the testbench begins to send data from the data port Atlantic generator. The data Atlantic monitor checks that the first data matches the first data sent from the generator and so on, until all the data is sent.
- If the priority data port is enabled, the testbench begins to send data from the priority port Atlantic generator. The priority Atlantic monitor checks that the first priority data matches the first priority data sent from the generator and so on, until all the data is sent.

Once a monitor receives the last packet, the testbench finishes.

You can use the SerialLite testbench as a template for creating your own testbench or modify it to increase the testing coverage. The tasks and parameters in the testbench are described in the following sections.

Configuring the Simulation

The simulation run is controlled by parameters. These parameters require no recompilation, making them fast and easy to change. The parameters are described in detail in the sections that follow.

ModelSim Simulator

If you are using the ModelSim simulator, the <*variation name*>`_tb_params.txt` and <*variation name*>`_core_params.txt` files contain the list of parameters to change and their values. By editing these lists of parameters according to the parameter descriptions below, you change how simulation proceeds. Not all parameters are available for any given link. For example, if you only instantiate the priority data port, editing the parameter list for the regular data port has no effect on the simulation.

There can be only one parameter per line. If you edit the parameter, edit only the value to the right of the equal (=) sign. If you inadvertently change the value to the left of the equal sign, restore it to its original name exactly, including case.

Downloaded from Elcodis.com electronic components distributor

Do not delete any parameters from or add any parameters to the parameter list.

Table 4–1 provides information about each parameter and its legal value.

Table 4–1. SerialLite IP Testbench Parameters (ModelSim Simulator) (Part 1 of 2)

Parameter	Minimum	Maximum	Default	Description
`data rate` `(_core_params.txt)`	400 Mbps	3,125 Mbps	3,125 Mbps	This is the operating data rate for a single serial lane. The input clock (`CLK`), and the Atlantic TX and Atlantic RX clocks are derived from this data rate. The clock is derived from the following formula: (1/(1,000,000 × value)) × 20 in picoseconds.
`IncrementDataValue`	0	1	1	For each packed on the regular and priority data ports, the count restarts at 0 at the beginning of a new packet. This parameter affects both data ports. The current data value is tracked separately for each port. The count restarts when the byte value is 8'hff. When set to 0, the data provided in packets is determined by user input. See “User Packet Data” on page 4–11.
`NumPriorityPackets`	1	(2^32)-1	10	The number of priority data packets. Controls the number of packets sent by the Atlantic generator to the priority data port. Only applicable if the priority data port is enabled.
`NumRegularPackets`	1	(2^32)-1	10	The number of regular data packets. Controls the number of packets sent by the Atlantic generator to the regular data port. Only applicable if the regular data port is enabled and packet mode is selected

Downloaded from Elcodis.com electronic components distributor

Table 4–1. SerialLite IP Testbench Parameters (ModelSim Simulator) (Part 2 of 2)

Parameter	Minimum	Maximum	Default	Description
`NumStreamTransactions`	1	(2^32)-1	12	The number of streaming transactions. Controls the number of transactions sent by the Atlantic generator to the regular data port. A transaction is considered to be 256-bytes of data. Only applicable if the regular data port is enabled and streaming mode is selected.
`PriorityPacketLength`	1 byte	65,535 bytes	10 bytes	The priority data packet size. Controls the number of bytes in a packet sent by the Atlantic generator to the priority data port. If used, all packets are of this size. Applicable only if the priority data port is enabled. Note: The maximum packet size supported by the priority port without errors is 256 bytes. Setting the packet size greater than 256 bytes results in a testbench error.
`RegularPacketLength`	1 byte	65,535 bytes	256 bytes	The regular data packet size. Controls the number of bytes in a packet sent by the generator to the regular data port. If used, all packets are of this size. Available only if the regular data port is enabled and packet mode is selected.

Other Simulators

If you are using a simulator other than the ModelSim tool, the *<variation name>*`_tb.v` file contains the list of parameters to change and their values. By editing these parameters according to the parameter descriptions below, you change how simulation proceeds. Not all parameters are available for any given link. For example, if you only instantiate the priority data port, editing the parameter list for the regular data port has no effect on the simulation.

Do not delete any parameters from the parameter list.

Downloaded from Elcodis.com electronic components distributor

Table 4–2 provides information about each parameter and its legal value.

Table 4–2. SerialLite IP Testbench Parameters (Other Simulator) (Part 1 of 2)

Parameter	Minimum	Maximum	Default	Description
ATLANTIC_TX_CLOCK_PERIOD	6,400	50,000	6,400	This is the main SerialLite MegaCore clock, which drives the input CLK and the Atlantic TX and RX clocks. The resultant clock also drives the ALTGXB transceiver. Thus, this clock determines the data rate of the high-speed serial link. The link speed (in Mbps) is determined as follows: (1000000/<value>)*20
DATA_INC_PATTERN	0	1	1	For each packed on the regular and priority data ports, the count restarts at 0 at the beginning of a new packet. This parameter affects both data ports. The current data value is tracked separately for each port. The count restarts when the byte value is 8'hff. When set to 0, the data provided in packets is determined by user input. See the section "User Packet Data" on page 4–11.
NUM_PRIORITY_PACKETS	1	(2^32)-1	10	The number of priority data packets. Controls the number of packets sent by the Atlantic generator to the priority data port. Only applicable if the priority data port is enabled.
NUM_DATA_PACKETS	1	(2^32)-1	10	The number of regular data packets. Controls the number of packets sent by the Atlantic generator to the regular data port. Only applicable if the regular data port is enabled and packet mode is selected

Downloaded from Elcodis.com electronic components distributor

Parameter	Minimum	Maximum	Default	Description
NUM_STREAM_TRANSACTIONS	1	(2^32)-1	12	The number of streaming transactions. Controls the number of transactions sent by the Atlantic generator to the regular data port. A transaction is considered to be 256-bytes of data. Only applicable if the regular data port is enabled and streaming mode is selected.
PRIORITY_PACKET_LENGTH	1 byte	65,535 bytes	10 bytes	The priority data packet size. Controls the number of bytes in a packet sent by the Atlantic generator to the priority data port. If used, all packets are of this size. Applicable only if the priority data port is enabled. Note: The maximum packet size supported by the priority port without errors is 256 bytes. Setting the packet size greater than 256 bytes results in a testbench error.
DATA_PACKETS_LENGTH	1 byte	65,535 bytes	256 bytes	The regular data packet size. Controls the number of bytes in a packet sent by the generator to the regular data port. If used, all packets are of this size. Available only if the regular data port is enabled and packet mode is selected.

Table 4–2. SerialLite IP Testbench Parameters (Other Simulator) (Part 2 of 2)

Downloaded from Elcodis.com electronic components distributor

Sending & Receiving Data Tasks

The testbench allows users to manipulate sending and receiving data through tasks.

Atlantic Generator

The Atlantic generator (`atl_gen`) features a send packet task (`send_pkt`) that transmits data and priority packets into the SerialLite MegaCore. The task also supports the streaming mode if the data port is configured as such.

To invoke the `send_pkt` task, use the following syntax from within the testbench:

```
seriallite_tb.atl_gen_dat_inst.send_pkt
(sop_ena, eop_ena, err_ena, 0, taddress,
tpktlength, user_data_control);
```

Table 4–3 describes the send packet task fields.

Table 4–3. Send Packet Task Field Descriptions (Part 1 of 2)

Field Location in Task	Field	Valid Values	Description
1	`sop_ena`	1'b0 or 1'b1	The `sop_ena` field determines if a start-of-packet (SOP) is asserted at the beginning of this packet. In packet mode, set this to 1'b1. In streaming mode, set this to 1'b0.
2	`eop_ena`	1'b0 or 1'b1	The `eop_ena` field determines if an end-of-packet (EOP) is asserted at the end of this packet. In packet mode, set this to 1'b1. In streaming mode, set this to 1'b0.
3	`err_ena`	1'b0 or 1'b1	The `err_ena` field determines if an error (ERR) is asserted at the end of a packet when EOP is asserted. In streaming mode, set this to 1'b0. In packet mode, set this to 1'b0. You can optionally set it to 1'b1 to set the error flag for that packet.
4	Reserved	0	Reserved for future use.
5	`taddress`	0 - 8'hFF (data) 0 - 4'hF (priority)	The `taddress` field sets the address for the current packet for use in channel-muxing mode. Set this to 0 when channel muxing is disabled.

Downloaded from Elcodis.com electronic components distributor

Table 4–3. Send Packet Task Field Descriptions (Part 2 of 2)			
Field Location in Task	**Field**	**Valid Values**	**Description**
6	tpktlength	1 - 65535 (bytes)	The tpktlength field sets the size of the current packet being sent by this task. For the priority port instantiation of ATLGEN, this length should not exceed 256-bytes to ensure error free operation.
7	user_data_control	1'b0 or 1'b1	The user_data_control field determines the data source for the packet payload. When set to 1'0, the payload defaults to incrementing payload, starting at 0 at the start of packet. When set to 1'b1, you must pre-fill an array with the packet contents to be transmitted. This task is described in detail in "User Packet Data" on page 4–11.

Example

```
seriallite_tb.atl_gen_dat_inst.send_pkt
(1'b1, 1'b1, 1'b0, 0, 8'h0, 16'd256, 1'b0);
```

When invoked, the above example generates a 256-byte packet with the Atlantic address field set to 0, and incrementing payload.

Atlantic Monitor

The Atlantic monitor (atl_mon) features a receive packet task (rcv_pkt) that checks data and priority packets from the SerialLite MegaCore. The task also supports the streaming mode if the data port is configured as such. By default, the monitor is configured for packet mode. To put the monitor into streaming mode, use a Verilog defparam on the packet_mode parameter, and set this to 0.

Example

```
defparam atl_mon_dat_inst.packet_mode = 0;
// Streaming mode.
```


Downloaded from Elcodis.com electronic components distributor

To invoke the `rcv_pkt` task, use the following syntax from within the testbench:

```
seriallite_tb.atl_mon_dat_inst.rcv_pkt(err_ena,
0, raddress, rpktlength, user_data_control);
```

Table 4–4 describes the receive packet task fields.

Table 4–4. Receive Packet Task Field Descriptions

Field Location in Task	Field	Valid Values	Description
1	`err_ena`	1'b0 or 1'b1	The `err_ena` field determines if the Atlantic `ERR` signal is expected to be asserted at the end of a packet when EOP is asserted. In streaming mode, set this to 1'b0. In packet mode, set this to 1'b0. You can optionally set it to 1'b1 when the packet is expected to have the error flag set.
2	Reserved	0	Reserved for future use.
3	`raddress`	0 - 8'hFF (data) 0 - 4'hF (priority)	The `raddress` field sets the expected address for the current packet for use in channel-muxing mode. Set this to 0 when channel muxing is disabled.
4	`rpktlength`	1 - 65535 (bytes)	The `rpktlength` field sets the expected size of the current packet being received by this task.
5	`user_data_control`	1'b0 or 1'b1	The `user_data_control` field determines the data source for the packet payload checking. When set to 1'0, the payload checking defaults to incrementing payload, starting at 0 at the start of packet. When set to 1'b1, the user must pre-fill an array with the packet contents to be transmitted. This task is described in detail in "User Packet Data" on page 4–11.

Example

```
seriallite_tb.atl_mon_dat_inst.rcv_pkt
(1'b0, 0, 8'h0, 16'd256, 1'b0);
```

When this example is invoked, the next incoming packet is checked to see if the packet is 256-bytes long of incrementing payload on address 0, with no `ERR` set.

Downloaded from Elcodis.com electronic components distributor

The send_pkt and rcv_pkt tasks are normally invoked using a Verilog fork and join statement. Each task must finish before invoking another similar task call; that is, you cannot fork two send_pkt tasks on the same Atlantic generator instantiation.

User Packet Data

If the DATA_INC_PATTERN parameter is set to 0, you can set the payload of each packet to contain any byte value. To set the payload, invoke the user_packet task before calling the send_pkt task. Once all the payload bytes of the packet have been set, the send_pkt task, with the user_data_control field set, uses the data contained in the internal array for the payload. Do not call the user_packet task again until the send_pkt task has completed the packet transfer, or else data corruption may occur.

To invoke the user_packet task, use the following syntax within the testbench:

```
seriallite_tb.atl_gen_dat_inst.user_packet
(user_data,byte_count);
```

Table 4–5. User Packet Task Field Descriptions

Field Location in task	Field	Valid Values	Description
1	User_data	8'h00 - 8'hFF	Contains the payload byte value.
2	Byte_count	16'h0000 - 16'hFFFF	Contains the byte location count for the payload byte.

Downloaded from Elcodis.com electronic components distributor

Example (5-byte packet)

```
seriallite_tb.atl_gen_dat_inst.user_packet(8'hFE,1
6'h0); // SOP
seriallite_tb.atl_gen_dat_inst.user_packet(8'hED,1
6'h1);
seriallite_tb.atl_gen_dat_inst.user_packet(8'hCA,1
6'h2);
seriallite_tb.atl_gen_dat_inst.user_packet(8'hBB,1
6'h3);
seriallite_tb.atl_gen_dat_inst.user_packet(8'h1E,1
6'h4); // EOP.
seriallite_tb.atl_gen_dat_inst.send_pkt(1'b1,
1'b1, 1'b0, 0, 8'h0, 16'd5, 1'b1);
```

When these tasks are invoked, a 5-byte packet with the payload `FEEDCABB1E` is transmitted into the SerialLite Data port. Byte location 0 is the SOP, and byte location 4 is the EOP. The transmission order is from byte count 0 and up.

Configuration of Status Monitors

The simulation includes a status port monitor for each bit of the status port. The monitor becomes active on a change to one of the status port bits. When active, the current value is checked against the expected value for that port. If the expected value is different from the current value, the monitor flags an error.

One status monitor per status bit is instantiated. To enable the monitor, set the `sp_en` port for each monitor to 1. For example,

```
sp_en[4..0] = 5'b11111;
```

enables the monitor for bits 0-4 of the `Status` bus.

To set the expected value inside each monitor, invoke the `set_expect` task. For example,

```
seriallite_tb.stat_mon_inst0.set_expect(1);
```

The only field in the `set_expect` task is the expected value for that monitor. In this example, the expected bit value for status port bit 0 (`stat_mon_inst0`) is set to 1.

By default, the expected value inside each status monitor is 1'b0.

Downloaded from Elcodis.com electronic components distributor

Special Simulation Configuration Settings

The SerialLite MegaCore contains two settings that have a reduced value in simulation.

The internal counter that controls the duration of the digital resets to the ALTGXB megafunction counts up to 20 in simulation. This overrides the default value of 1,000,000.

The clock compensation value determines when the clock compensation sequence is inserted into the high-speed serial stream (when clock compensation is enabled). In simulation, to minimize the time it takes for the sequence to occur, the value is always 100 cycles, independent of the actual clock compensation time value (100 or 300 ppm).

Waveform Generation

The simulation allows VCD file generation if `WAVEFORM` is tick defined. All signals are included in the dump file.

Example

Add `` `define WAVEFORM `` to the testbench or `+define+WAVEFORM` to the simulator command line to create a VCD dump file.

Testbench Timeout

The testbench uses a maximum simulation time to guard against infinite loops or stuck simulations. The default value of 15000000 system clock cycles is probably sufficient for most simulation runs. If more time is needed for a particularly long run, you can increase the `WATCHTIME` value.

For example, inside the testbench, add

```
`define WATCHTIME 55000000
```

Running a Simulation

A simulation script allows you to run a simulation based on the simulation configuration you have chosen. To run the simulation while in the ModelSim Tcl environment, first ensure that the working directory is the one you specified in step 4 of "Create a New Quartus II Project" on page 2–5.

1. Run ModelSim (`vsim`) to bring up the user interface.

Downloaded from Elcodis.com electronic components distributor

2. Run the simulation.

   a. In the Model Technology ModelSim-Altera simulation tool, run the simulation by typing the following command:

      `do` *<variation name>*`_tb_ae.do`↵

   b. In other versions of the ModelSim simulation tool, run the simulation by typing the following command:

      `do` *<variation name>*`_tb.do`↵

The testbench uses the file *<variation name>*`_tb_params.txt` as the simulation parameter input file and creates a file *<variation name>*`.log` as an output file.

Simulation Pass & Fail Conditions

The meaning of "pass" or "fail" can vary based on intent, so this section clarifies what it means when a simulation run ends and failure is reported.

The execution of a simulation run consists of the following components:

- Creating data to be transported through the link
- Verifying that the data arrived with or without errors
- Verifying that the various protocols were honored in the delivery of the data
- Confirming that the state of the link is consistent.

The testbench concludes by checking that all of the packets have been received. In addition, it checks that the Atlantic packet receivers (`atl_mon` modules, one for data port and one for priority port) have not detected any errors in the received packets. If no errors have been detected, and all packets have been received, the testbench issues a message stating that the simulation was successful.

If errors have been detected, a message states that the testbench has failed. If not all packets have been detected, a message states that the testbench is incomplete. The `tb.exp_chk_cnt` variable determines the number of checks done to insure completeness of the testbench. In packet mode, or when using the priority port, for each packet tested, the testbench performs one completeness check. In streaming mode on the data port, a check is registered every 256 bytes of data received by the monitor.

Downloaded from Elcodis.com electronic components distributor

In summary, the testbench checks the following:

- Were all expected stimulus generated?
- Did all expected packets arrive and was the data error-free?
- If errors occurred on the data, did the SerialLite logic detect the errors?
- Were there any protocol errors?
- Is there any evidence of the simulation running too long out of control?

If any of those checks detect a problem, the simulation is reported as failing. In a correctly operating testbench, the only reason for failing is the detection of deliberately inserted errors. There is a distinction between a simulation run failing and a test failing. If you insert errors and the errors are detected, the simulation fails. However, the test was successful because the errors were detected. For this reason, simulation failure is not by itself an indication of a problem.

Downloaded from Elcodis.com electronic components distributor

Downloaded from Elcodis.com electronic components distributor