MegaCore®

SerialLite II

MegaCore Function User Guide

ALTERA®

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

MegaCore Version:	8.0
Document Date:	May 2008

I.S. EN ISO 9001

UG-0705-1.7

Contents

1. About This MegaCore Function

Release Information

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

Table 1–1. SerialLite II Release Information

Item	Description
Version	8.0
Release Date	May 2008
Ordering Code	IP-SLITE2
Product ID	00AD
Vendor ID	6AF7

Altera verfies that the current version of the Quartus® II software compiles the previous version of each MegaCore function. The *MegaCore IP Library Release Notes and Errata* report any exceptions to this verification. Altera does not verify compilation with MegaCore function versions older than one release.

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 II MegaCore function to each Altera device family.

Table 1–2. Device Family Support

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

Features

- Physical layer features
  - 622 Mbps to 6.375 Gbps per lane
  - Single or multiple lane support (up to 16 lanes)
  - 8-, 16-, or 32-bit data path per lane
  - Symmetric, asymmetric, unidirectional/simplex or broadcast mode
  - Optional payload scrambling
  - Full-duplex or self-synchronizing link state machine
  - Channel bonding scalable up to 16 lanes
  - Synchronous or asynchronous operation
    - Automatic clock rate compensation for asynchronous use
    - ±100 and ±300 parts per million (ppm)
- Link layer features
  - Atlantic™ interface compliant
  - Support for two user packet types: data packet and priority packet
  - Optional packet integrity protection using cyclic redundancy code (CRC-32 or CRC-16)
  - Optional link management packets
    - Retry-on-error for priority packets
    - Individual port (data/priority) flow control
- Unrestricted data and priority packet size
- Support for TimeQuest timing analyzer
- Polarity reversal for Stratix II GX and Stratix IV
- Lane order reversal
- IP functional simulation models for use in Altera-supported VHDL and Verilog HDL simulators
- Support for OpenCore® Plus evaluation

General Description

SerialLite II is a light weight protocol suitable for packet and streaming data in chip-to-chip, board-to-board, and backplane applications. This protocol offers low gate count and minimum data transfer latency. It provides reliable, high-speed transfers of packets between devices over serial links. The SerialLite II protocol defines packet encapsulation at the link layer and data encoding at the physical layer. This protocol integrates transparently with existing networks without software support.

The SerialLite II MegaCore function is a simple, high-speed, low-latency, low-resource, point-to-point serial data communication link with performance up to 3.125 Gbps in Arria GX and Stratix GX devices and up to 6.375 Gbps with a transfer size of 4 in Stratix II GX and Stratix IV GX devices. The SerialLite II MegaCore function is highly configurable, and provides a wide range of functionality suited to moving data in many different environments.

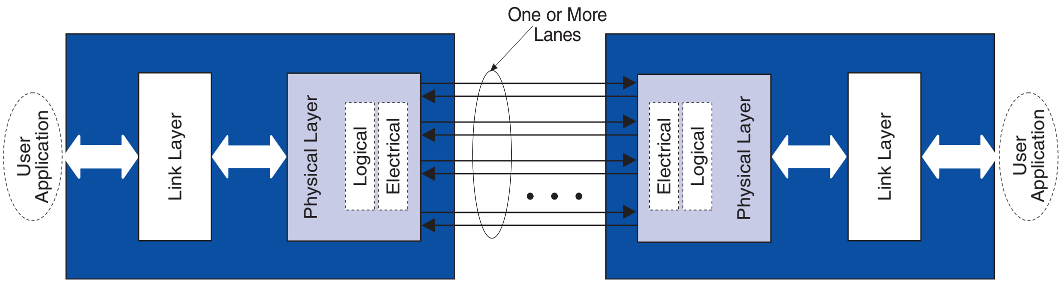
The SerialLite II MegaCore function provides a simple and lightweight way to move data from one point to another reliably at high speeds. It consists of a serial link of up to 16 bonded lanes, with logic to provide a number of basic and optional link support functions. The Atlantic interface is the primary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

The SerialLite II protocol specifies a link that is simple to build, uses as little logic as possible, and requires little work for a logic designer to implement. The SerialLite II MegaCore function uses all of the features available in the SerialLite II protocol. You can parameterize the MegaCore function using the MegaWizard® Plug-In Manager.

A link built using the SerialLite II MegaCore function operates at from 622 Mbps to 6.375 Gbps per lane. Link reliability is enhanced by the 8B10B encoding scheme and optional CRC capabilities. You can achieve further reductions in the bit-error rate by using the optional retry-on-error feature. Data rate and consumption mismatches can be accommodated using the optional flow-control feature to ensure that no data is lost.

Figure 1–1 shows that the SerialLite II MegaCore function is divided into two main blocks: a protocol processing portion (data link layer) and a high-speed front end (physical layer).

Figure 1–1. SerialLite II MegaCore Function High-Level Block Diagram



You can use the SerialLite II MegaCore function in the following applications:

- Chip-to-chip connectivity
- Board-to-board connectivity
- Shelf-to-shelf connectivity
- Backplane communication
- Bridging applications
- Streaming video applications
- Imaging applications

Figures 1–2 and 1–3 show two examples of bridging applications.

Figure 1–2. Typical Application—Bridging Functions

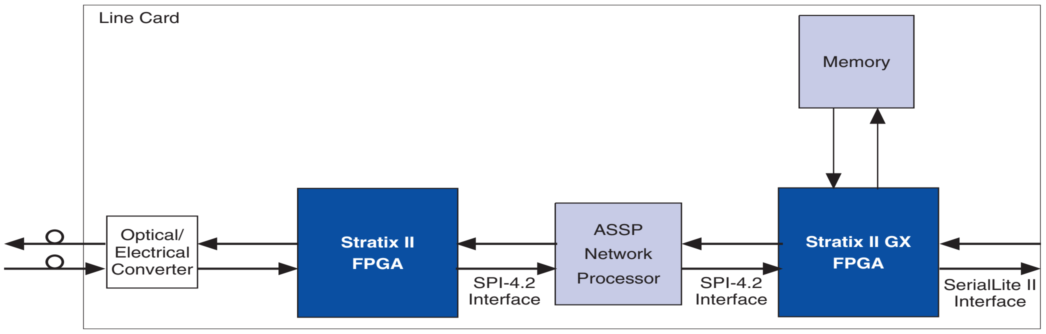
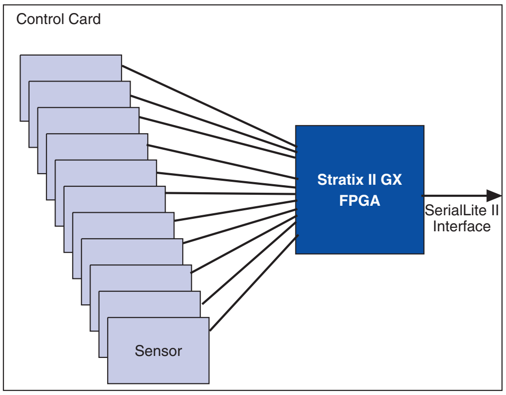


Figure 1–3. Typical Application—Unidirectional Bridging Application



Performance and Resource Utilization

Table 1–3 lists the resources and internal core speeds for a selection of variations using a 1,024-byte first-in first-out (FIFO) buffer. These results were obtained using the Quartus® II software version 8.0 for the following device: Stratix II GX (EP2GX90FF1508C3)

Table 1–3. Performance (Part 1 of 2)

Parameters								Combin-ational ALUTs	Logic Reg.	Memory Blocks		f_{MAX} (MHz)	Throughput Mbps (2)
Data Flow	Number of Lanes	Data/ Type	Packet Type	Transfer Size (1)	CRC	Flow Control	Retry on Error			M512	M4K		
Full-Duplex	1	Packet	Data	1	No	No	No	756	741	9	10	267	1250
Full-Duplex	1	Packet	Data	2	No	No	No	768	754	0	11	285	3125
Full-Duplex	1	Packet	Data	4	No	No	No	863	818	11	11	273	6375
Full-Duplex	4	Packet	Data	1	No	No	No	1215	1031	15	11	239	1250
Full-Duplex	4	Packet	Data	2	No	No	No	1507	1113	15	22	249	3125
Full-Duplex	4	Packet	Data	4	No	No	No	2089	1554	2	48	247	6375
Full-Duplex	16	Packet	Data	2	No	No	No	4101	2809	17	87	199	3125

Table 1–3. Performance (Part 2 of 2)

Parameters								Combin-ational ALUTs	Logic Reg.	Memory Blocks		fMAX (MHz)	Throughput Mbps (2)
Data Flow	Number of Lanes	Data/ Type	Packet Type	Transfer Size (1)	CRC	Flow Control	Retry on Error			M512	M4K		
Full-Duplex	16	Packet	Data	4	No	No	No	6347	4493	1	180	185	6375
Full-Duplex	1	Packet	Data	1	32	Yes	No	1077	1028	9	12	276	1250
Full-Duplex	1	Packet	Data	2	32	Yes	No	1181	1019	1	12	239	3125
Full-Duplex	1	Packet	Data	4	32	Yes	No	1381	1075	12	12	215	6375
Full-Duplex	4	Packet	Data	1	32	Yes	No	1787	1306	16	12	215	1250
Full-Duplex	4	Packet	Data	2	32	Yes	No	2387	1446	16	23	192	3125
Full-Duplex	4	Packet	Data	4	32	Yes	No	3384	1907	3	49	177	6375
Full-Duplex	1	Packet	Priority	2	16	Yes	Yes	1448	1236	1	22	228	3125
Full-Duplex	1	Packet	Priority	4	16	Yes	Yes	1675	1284	12	22	225	6375
Full-Duplex	4	Packet	Priority	2	16	Yes	Yes	2573	1659	17	41	212	3125
Full-Duplex	4	Packet	Priority	4	16	Yes	Yes	3528	2110	17	41	160	6375

Notes to Table 1–3:
(1) A transfer size of 1 is used for 1,250 Mbps, 2 is used for 3,125 Mbps, and 4 is used for 6,375 Mbps.
(2) Total throughput equals the value in the Throughput column multiplied by the value in the Number of Lanes column.

Table 1–4 lists the resources and internal core speeds for a selection of variations using 1,024-byte FIFO buffers. These results were obtained using the Quartus II software version 8.0 for the following device: Stratix GX (EP1SGX40GF1020C5)

Table 1–4. Performance (Part 1 of 3)

Parameters								LEs	Memory Blocks		f_{MAX} (MHz)	Throughput Mbps (2)
Data Flow	Number of Lanes	Data Type	Packet Type	Transfer Size (1)	CRC	Flow Control	Retry on Error		M512	M4K		
Full-Duplex	1	Packet	Data	1	No	No	No	1065	9	10	178	1250
Full-Duplex	1	Packet	Data	2	No	No	No	1098	0	11	182	3125
Full-Duplex	4	Packet	Data	1	No	No	No	1711	15	11	186	1250
Full-Duplex	4	Packet	Data	2	No	No	No	2706	23	22	180	3125
Full-Duplex	16	Packet	Data	2	No	No	No	8328	50	87	158	3125
Full-Duplex	1	Packet	Data	1	32	Yes	No	1687	10	10	172	1250
Full-Duplex	1	Packet	Data	2	32	Yes	No	1728	2	11	163	3125
Full-Duplex	4	Packet	Data	1	32	Yes	No	2496	17	11	151	1250
Full-Duplex	4	Packet	Data	2	32	Yes	No	3848	25	22	127	3125
Full-Duplex	1	Packet	Data and Priority	2	No	Both	No	2169	2	19	181	3125
Full-Duplex	1	Packet	Data and Priority	2	32	Both	No	2538	2	19	165	3125
Full-Duplex	4	Packet	Data and Priority	2	No	Both	No	3911	33	38	181	3125
Full-Duplex	4	Packet	Data and Priority	2	32	Both	No	4797	33	38	125	3125
Full-Duplex	1	Packet	Priority	2	No	Yes	No	1350	2	11	188	3125

Table 1–4. Performance (Part 2 of 3)

Parameters								LEs	Memory Blocks		f_{MAX} (MHz)	Throughput Mbps (2)
Data Flow	Number of Lanes	Data Type	Packet Type	Transfer Size (1)	CRC	Flow Control	Retry on Error		M512	M4K		
Full-Duplex	4	Packet	Priority	2	No	Yes	No	2993	24	22	173	3125
Full-Duplex	8	Packet	Priority	2	No	Yes	No	4787	28	44	167	3125
Full-Duplex	1	Packet	Priority	2	16	Yes	Yes	2241	2	21	163	3125
Full-Duplex	4	Packet	Priority	2	16	Yes	Yes	4373	26	40	144	3125
Full-Duplex	1	Streaming	Data	1	No	No	No	198	0	0	253	1250
Full-Duplex	1	Streaming	Data	2	No	No	No	243	0	0	246	3125
Full-Duplex	4	Streaming	Data	1	No	No	No	763	4	0	183	1250
Full-Duplex	4	Streaming	Data	2	No	No	No	1681	12	0	194	3125
Simplex Tx	1	Streaming	Data	1	No	No	No	27	0	0	422	1250
Simplex Tx	1	Streaming	Data	2	No	No	No	35	0	0	422	3125
Simplex Rx	1	Streaming	Data	1	No	No	No	98	0	0	282	1250
Simplex Rx	1	Streaming	Data	2	No	No	No	128	0	0	240	3125
Asymm Tx	4	Packet	Data	1	No	No	No	1392	8	11	177	1250
Asymm Tx	4	Packet	Data	2	No	No	No	1908	17	17	168	3125
Asymm Tx	8	Packet	Data	2	No	No	No	2292	13	29	169	3125
Asymm Rx	4	Packet	Data	1	No	No	No	1604	9	11	195	1250
Asymm Rx	4	Packet	Data	2	No	No	No	2559	23	16	177	3125
Broad-cast Rx	4	Streaming	Data	1	No	No	No	561	0	0	187	1250

Table 1–4. Performance (Part 3 of 3)

Parameters								LEs	Memory Blocks		f_{MAX} (MHz)	Throughput Mbps (2)
Data Flow	Number of Lanes	Data Type	Packet Type	Transfer Size (1)	CRC	Flow Control	Retry on Error		M512	M4K		
Broad-cast Rx	4	Streaming	Data	2	No	No	No	729	0	0	200	3125
Broad-cast Rx	8	Streaming	Data	2	No	No	No	1359	0	0	181	3125

Notes to Table 1–4:
(1) A transfer size of 1 is used for 1,250 Mbps and 2 is used for 3,125 Mbps.
(2) Total throughput equals the value in the Throughput column multiplied by the value in the Number of Lanes column.

Installation and Licensing

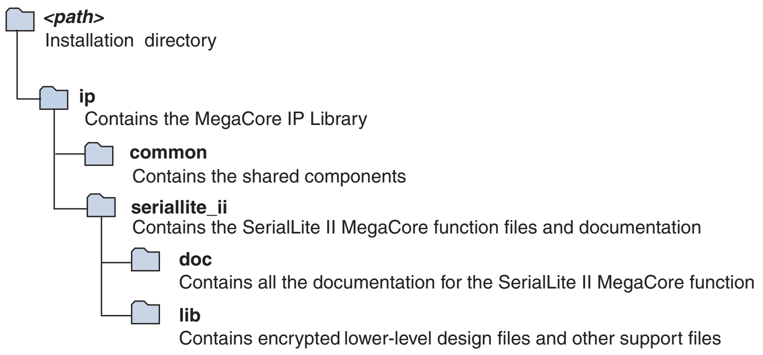
The SerialLite II MegaCore function is part of the MegaCore® IP Library, which is distributed with the Quartus® II software and downloadable from the Altera® website, www.altera.com.

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

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

Figure 1–4 shows the directory structure after you install the SerialLite II MegaCore Function User Guide, where <*path*> is the installation directory. The default installation directory on Windows is **c:\altera\80**; on UNIX and Solaris it is **/opt/altera/80**.

Figure 1–4. SerialLite II MegaCore Function Directory Structure


<path>
Installation directory
ip
Contains the MegaCore IP Library
common
Contains the shared components
seriallite_ii
Contains the SerialLite II MegaCore function files and documentation
doc
Contains all the documentation for the SerialLite II MegaCore function
lib
Contains encrypted lower-level design files and other support files

For details on installation and licensing, refer to *Quartus II Installation & Licensing for Windows* or *Quartus II Installation & Licensing for UNIX & Linux Workstations*.

OpenCore Plus Evaluation

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

- Simulate the behavior of a megafunction (Altera MegaCore function or AMPP^{SM} megafunction) within your system
- Verify the functionality of your design, as well as evaluate its size and speed quickly and easily
- Generate time-limited device programming files for designs that include megafunctions
- Program a device and verify your design in hardware

OpenCore Plus Timeout Behavior

OpenCore Plus hardware evaluation supports the following two operation modes:

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

All megafunctions in a device timeout simultaneously when the most restrictive evaluation time is reached. If there is more than one megafunction in a design, a specific megafunction's timeout behavior may be masked by the timeout 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; the SerialLite II MegaCore function is forced into reset.

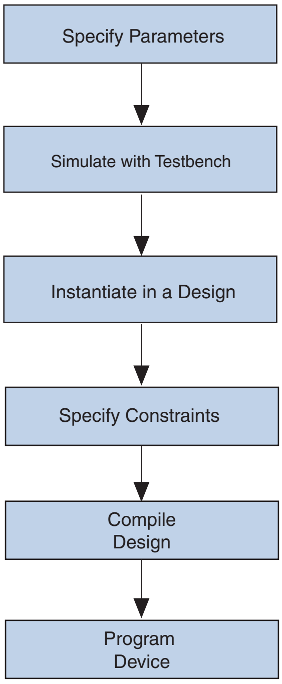
For more information on OpenCore Plus hardware evaluation, see *AN 320: OpenCore Plus Evaluation of Megafunctions*.

2. Getting Started

Design Flow

Figure 2–1 outlines the high-level steps required to create a design that includes the SerialLite II MegaCore function. Each step is explained in detail in the walkthrough below.

Figure 2–1. SerialLite II MegaCore Design Flow



This chapter explains how to create a SerialLite II MegaCore function using the MegaWizard® Plug-In Manager and the Quartus II software. When you finish generating a custom variation of the SerialLite II MegaCore function, you can incorporate it into your overall project.

This walkthrough requires the following steps:

- Create a New Quartus II Project
- Launch the MegaWizard Plug-In Manager
- Parameterize
- Set Up Simulation
- Generate Files

Create a New Quartus II Project

You can create a new Quartus II project with the **New Project Wizard**, which specifies the working directory for the project, assigns the project name, and designates the name of the top-level design entity.

To create a new project follow these steps:

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

2. On the File menu, click **New Project Wizard**.

3. Click **Next** in the **New Project Wizard Introduction** (the introduction does not display if you turned it off previously).

4. In the **New Project Wizard: Directory, Name, Top-Level Entity** page, enter the following information:

   a. Specify the working directory for your project. For example, this walkthrough uses the **c:\altera\projects\slite2_project** directory.

   b. Specify the name of the project. This walkthrough uses **example** for the project name.

   ☞ The Quartus II software automatically specifies a top-level design entity that has the same name as the project. This walkthrough assumes that the names are the same.

5. Click **Next** to display the **New Project Wizard: Add Files** page.

   ☞ When you specify a directory that does not already exist, a message asks if the specified directory should be created. Click **Yes** to create the directory.

6. Click **Next** to close this page and display the **New Project Wizard: Family and Device Settings** page.

7. On the **New Project Wizard: Family and Device Settings** page, choose the target device family in the **Family** list. For this walkthrough, select **Stratix II GX**.

8. The remaining pages in the **New Project Wizard** are optional. Click **Finish** to complete the Quartus II project.

Launch the MegaWizard Plug-In Manager

To launch the MegaWizard Plug-In Manager in the Quartus II software, follow these steps:

1. Click **MegaWizard Plug-In Manager** on the Tools menu.

   ☞ Refer to 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. Under **Installed Plug-Ins,** expand **Interfaces** and then, select **SerialLite II v8.0**.

4. Select the output file type for your design; the MegaWizard Plug-In Manager supports VHDL and Verilog HDL. For this example, select **Verilog HDL**.

5. The MegaWizard Plug-In Manager shows the project path that you specified in the **New Project Wizard**. Append a variation name for the MegaCore function output files <*project path*>\<*variation name*>. For this example, type `example` as the variation name.

6. Click **Next** to display the **Parameter Settings** page for the SerialLite II MegaCore function.

Parameterize

This section shows how to parameterize the example SerialLite II MegaCore function and describes the results of various options. A comprehensive description of all parameters is contained in Chapter 3, Parameter Settings.

☞ The following parameters are ordered as they appear in the MegaWizard Plug-In Manager. Not all parameters are supported by, or are relevant for, every MegaCore function variation.

To parameterize your MegaCore function, follow these steps:

1. Click **Parameter Settings** in the MegaWizard Plug-In Manager. The **Physical Layer** page appears.

2. Enter a data rate in megabits per second (Mbps). The SerialLite II MegaCore function supports data rates of 622 to 6,375 Mbps per lane.

   The data rate must be an acceptable range for the **Transfer size**. SerialLite II returns a warning or an error message if you specify a data rate that is not within the range for the specified **Transfer size**.

3. Choose a **Transfer size**. The **Transfer size** determines the number of contiguous data columns. The **Transfer size** also determines the serialization/deserialization (SERDES) factor and internal data path width:

   - A **Transfer size** of **1** equates to an internal data path of 8 bits (Recommended for less than 2.5 gigabits per second (Gbps))
   - A **Transfer size** of **2** equates to an internal data path of 16 bits (Recommended for less than or equal to 3.125 Gbps)
   - A **Transfer size** of **4** equates to an internal data path of 32 bits (Typically for greater than 3.125 Gbps, and only available for Stratix II GX and Stratix IV devices)

4. Specify the **Reference Clock Frequency**. The value changes with the date rate but the input reference clock must be within 50 MHz and 622 MHz.

5. Select a **Port Type**. You have three choices: **Bidirectional**, **Transmitter only**, and **Receiver only**.

   ☞ If you choose **Transmitter only** or **Receiver only**, the self-synchronized link-up parameter (link state machine) is enabled by default.

6. Turn on or off the **Self-Synchronized Link-Up** option. This parameter allows the receiver on the far end of the link to synchronize itself to incoming data streams, rather than on an exchange of status information with the transmitter. Note that the **Self-Synchronized Link-Up** feature is only for single lane applications.

7. Under **Transmitter Settings**, select the number of lanes for the transmitter.

8. Turn on or off the **Scramble** and **Broadcast mode** options.

9. Under **Receiver Settings**, select the number of lanes for the receiver.

   Table 2–1 shows the allowable number of lanes depending on the chosen parameters.

Table 2–1. Number of Transmit Lanes

Self-Synchronized Link-Up	Broadcast	Number of Lanes
✓	✓	2 – 16
✓	–	1
–	✓	2 – 16
–	–	1 – 16

10. Turn on or off the **De-scramble** option.

11. Turn on or off the **Enable frequency offset tolerance** option. If you turn on this option, select an offset tolerance of ± **100** or ± **300** parts per million (ppm).

12. Click **Configure Transceiver** to display the **Configure Transceiver** page. Select the following parameters on the **Configure Transceiver** page to configure the altgx megafunction.

See "Transceiver Configuration" on page 3–28 for a more detailed description of the transceiver parameters.

   a. For the transmitter, select the **Voltage Output Differential (VOD)** control setting value.

   b. Under **Pre-emphasis**, select a value for **Specify pre-emphasis control setting**.

   c. In the **Bandwidth mode** list, select **high** or **low** for the Tx PLL bandwidth.

   d. Select a value for the **Transmitter Buffer Power (VCCH)**.

   e. Under **Receiver Functionality**, select a value for **Specify equalizer control setting**.

   f. In the **Bandwidth mode** list, select **high**, **medium** or **low** for the Rx PLL bandwidth.

   g. To reconfigure functionality settings, specify a **Starting channel number.**

   h. Click **Finish**.

13. Click **Next** to open the **Link Layer** page.

14. Under **Data Type**, select **Packets** or **Streaming**.

15. If you select **Packets**, select a packet type: **Priority packets and data packets**, **Priority packets**, or **Data packets**.

16. If you select a packet type that includes priority packets, follow these substeps; otherwise, skip to Step 17.

    a. Turn on or off the **Retry-on-error** option.

    b. If you turned on **Retry-on-error**, specify a value for **Timeout** and **Segment size**.

    c. Under **Buffer Size**, specify a value for **Transmitter** and **Receiver**.

    d. Turn on or off the **Enable flow control** option.

    e. If you turned on **Enable flow control**, specify the values for the following settings:

       - **Pause quantum time**
       - **Threshold**
       - **Refresh period**

    If you selected **priority packets only**, skip to Step 18.

17. If you selected a packet type that includes data packets, follow these substeps:

    a. Turn on or off the **Enable flow control** option.

    b. If you turned on flow control, specify the values for the following settings:

       - **Pause quantum time**
       - **Threshold**
       - **Refresh period**

    For information on setting these parameters, refer to "Flow Control" on page 3–18.

    c. Select the transmitter and receiver buffer sizes (bytes).

18. If your transmitter or receiver requires cyclic redundancy code (CRC) checking, turn on the **Enable CRC** option for your chosen packet type and specify a value for **CRC Type**.

19. Click **Next**.

Set Up Simulation

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

CAUTION You may use these models only for simulation and not for synthesis or any other purposes. Using these models for synthesis creates a nonfunctional design.

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

1. On the **EDA** page, under **Simulation Libraries**, turn on **Generate Simulation Model**.

2. Some third-party synthesis tools can use a netlist that contains only the structure of the MegaCore function, but not detailed logic, to optimize performance of the design that contains the MegaCore function. If your synthesis tool supports this feature, turn on **Generate netlist**.

3. Click **Next** to display the **Summary** page.

Generate Files

You can use the check boxes on the **Summary** page to enable or disable the generation of specified files. A gray check mark indicates a file that is automatically generated; other check marks indicate optional files.

To generate your parameterized MegaCore function, follow these steps:

1. Turn on the files you want to generate.

2. To generate the specified files and close the MegaWizard interface, click **Finish**. The generation phase can take several minutes to complete.

The Quartus II IP File (**.qip**) is a file generated by the MegaWizard interface or SOPC Builder that contains information about a generated IP core. You are prompted to add this **.qip** file to the current Quartus II project at the time of file generation. In most cases, the **.qip** file contains all of the necessary assignments and information required to process the core or system in the Quartus II compiler. Generally, a single **.qip** file is generated for each MegaCore function and for each SOPC Builder system. However, some more complex SOPC Builder components generate a separate **.qip** file, so the system **.qip** file references the component **.qip** file.

A report file, <*variation name*>**.html**, in you project directory lists each file generated and provides a description of its contents.

Simulate the Design

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

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

Altera also provides a Verilog HDL demonstration testbench that shows you how to instantiate a model in a design for all configurations. Altera also provides a VHDL demonstration testbench for a restricted number of configurations. The testbench stimulates the inputs and checks the outputs of the interfaces of the SerialLite II MegaCore function, allowing you to evaluate the MegaCore function's basic functionality. The testbench is described in detail in Chapter 5, Testbench.

Instantiate the MegaCore

You can now integrate your custom MegaCore function variation into your design and simulate your complete design using your own custom testbench.

Specify Constraints

This example design applies constraints to create virtual pins and set up timing analysis.

Assign Virtual Pins

If you are compiling the SerialLite II MegaCore function variation as a standalone component, you must specify virtual pin assignments. The MegaWizard interface generates a tool command language (Tcl) script that automates this task. Follow these steps to run the script:

1. On the Tools menu, click **Tcl Scripts** to open the **Tcl Scripts** dialog box.
2. In the project directory, select *<variation_name>*_**constraints**.
3. Click **Run**.

☞ The script assumes the default names for the virtual pins. If you have connected the pins to names other than the default names, you must edit this script and change the virtual pin names when the core is still compiled in stand-alone mode.

Fitter Constraints

The Tcl script also optimizes fitter settings to produce the best performance ($\mathbf{f_{MAX}}$). Use this script as a guide to set constraints for the SerialLite II MegaCore function variation in your design. The timing constraints are currently set for the SerialLite II MegaCore function variation as a standalone component, thus you must update the script with hierarchy information for your own design. The Tcl script also points to the generated Synopsys Design Constraints (SDC) timing constraint script if the TimeQuest timing analyzer is enabled. The Fitter optimizes your design based on the requirements in the **.sdc** files in your project.

The script uses the `FITTER_EFFORT "STANDARD FIT"` Fitter setting.

☞ This fitter setting may conflict with your Quartus II software settings.

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

Timing Constraints

The SerialLite II MegaCore generates an ASCII file (with the **.sdc** extension) that contains design constraints and timing assignments in the industry-standard SDC format. The constraints in the **.sdc** file are described using the Tcl tool command language and follow Tcl syntax rules.

To specify the TimeQuest timing analyzer as the default timing analyzer, on the Assignments menu, click **Timing Analysis Settings**. In the **Timing Analysis Settings** page, turn on **Use TimeQuest Timing Analyzer during compilation**.

The TimeQuest timing constraints are currently set for the SerialLite II MegaCore function variation as a standalone component. You must update the script with hierarchy information if your own design is not a standalone component.

Refer to the *Quartus II TimeQuest Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook* for more information on how to use the TimeQuest Timing Analyzer.

Compile and Program

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

3. Parameter Settings

Table 3–1 shows the function parameters, which can be set only in the MegaWizard Plug-In Manager (see "Parameterize" on page 2–3). The following sections describe these parameters.

Table 3–1. Default SerialLite II Variation (Part 1 of 2)

Parameter	Default Configuration
Physical Layer	
Device family	Depends on the family specified in the MegaWizard Plug-in Manager
Data rate	3,125 megabits per second (Mbps)
Transfer size	2 Columns
Reference Clock Frequency	156.25 MHz
Port type	Bidrectional
Self-synchronized link up	Disabled
Number of lanes (Transmitter and Receiver)	1
Scramble/Descramble	Disabled
Broadcast mode	Disabled
Frequency offset tolerance	Disabled
Link Layer	
Data type	Packets
Packet type	Data packets
Flow control	Disabled
Buffer size (Transmitter and Receiver)	1,024 bytes
CRC generation	Disabled
Transceiver Configuration	
Voltage output differential (VOD)	0
Pre-emphasis control settings	0
Transmitter Buffer Power (VCCH)	1.5
Equalizer control settings	0

Table 3–1. Default SerialLite II Variation (Part 2 of 2)

Parameter	Default Configuration
Bandwidth mode (Transmitter and Receiver)	Low
Starting Channel number for reconfiguration	0

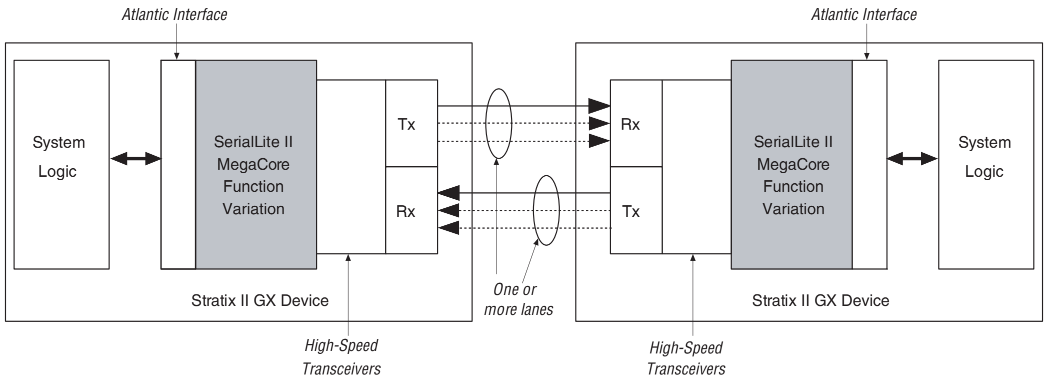
To configure your own variation of the SerialLite II MegaCore function, you must decide the following issues:

- High-level link configuration
- Bandwidth required
- Whether to use CRC checking
- Whether to implement flow control
- How to size the FIFO buffers

Link Consistency

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

Figure 3–1. Complete SerialLite II Link



Physical Layer Configuration

This section describes the options available to parameterize the physical layer of your SerialLite II MegaCore function variation.

Data Rate

The SerialLite II MegaCore function supports a data rate range of 622 to 6,375 Mbps per lane. In Arria GX and Stratix GX devices, the data rate must be less than 3,125 Mbps; in Stratix II GX and Stratix IV devices, the data rate must be less than 6,375 Mbps. The data rate range varies based on based on the device and the transfer size (TSIZE) as Table 3–2 illustrates.

Table 3–2. Data Rate Dependencies on Transfer Size

Data Rate (Gbps)	Arria GX/Stratix GX	Stratix II GX/Stratix IV
<= 2.5	TSIZE= 1, 2	TSIZE= 1, 2
<= 3.125	TSIZE= 2	TSIZE= 2
<= 5	Not Supported	TSIZE= 2*(1)*, 4
<= 6.375	Not Supported	TSIZE= 4

Note to Table 3–2:
(1) Symmetric mode (*p_RX_NUM_LANES* == *p_TX_NUM_LANES*) only.

Transfer size

The **Transfer size** parameter defines many important characteristics of the MegaCore function variation. **Transfer size** determines the number of contiguous data columns and the internal data path width per lane, where:

- A transfer size of 1 equates to an internal data path of 8 bits (Recommended for less than 2.5 Gbps)
- A transfer size of 2 equates to an internal data path of 16 bits (Recommended for less than or equal to 3.125 Gbps)
- A transfer size of 4 equates to an internal data path of 32 bits (Typically for greater than 3.125 Gbps, and only available for Stratix II GX and Stratix IV FPGAs)

and it determines the width of the SERDES block, where:

- A transfer size of 1 equates to a 10 bit-wide SERDES block
- A transfer size of 2 equates to a 20 bit-wide SERDES block
- A transfer size of 4 equates to a 40-bit wide SERDES block

The 40:1 deserialization factor is only available in Stratix II GX and Stratix IV devices, and is recommended when the data rate exceeds 3.125 Gbps and must be used when the data rate exceeds 5 Gbps.

Reference Clock Frequency

The **Reference Clock Frequency** parameter allows you to select the available input reference clock frequencies in (OIF) CEI PHY interface mode. Valid values change with the data rate but the reference input clock frequency must be within 50 MHz and 622 MHz.

- The general formula to determine frequency:

  Frequency = Data rate/m, where m can be 50, 40, 32, 25, 20, 16, 10, 8, 5 or 4

- Any value less than 50 MHz or more than 622 MHz is discarded
- This parameter is only applicable if you chose Stratix II GX, Stratix IV or Arria GX device family.

Port Type

The **Port Type** parameter offers three options: **Bidirectional**, **Transmitter only**, and **Receiver only**. If you turn on the **Bidirectional** option, you must specify values for **Transmitter Settings** and **Receiver Settings**. Under **Transmitter Settings**, you need to specify the **Number of lanes**, and select whether or not to enable the **Scramble** and **Broadcast mode**. Under **Receiver Settings**, you must specify the settings for the **Number of lanes**, and select whether or not to enable the **De-Scramble** option. If you turn on **Transmitter only** option, you must specify values for **Transmitter Settings only**, and if you turn on **Receiver only option**, you must specify values for **Receiver Settings** only.

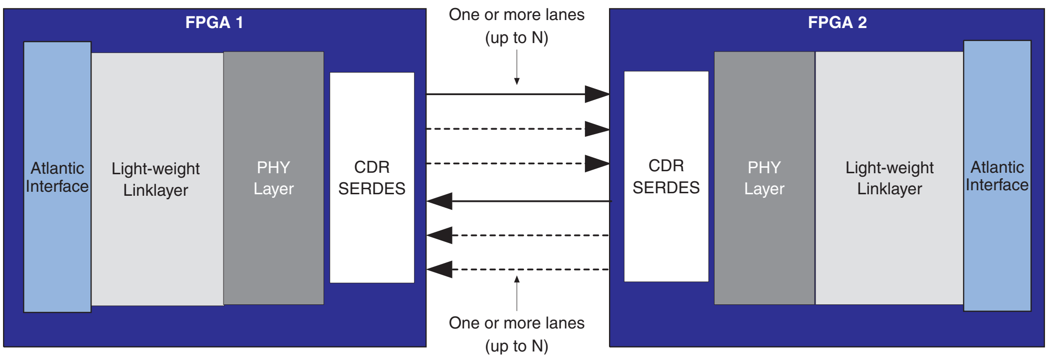
The **Number of lanes** parameter dictates the number of serial links, essentially the number of external inputs and outputs (I/Os) for the MegaCore function.

If you set the **Number of lanes** for the transmitter and receiver settings to the same value, you configure the MegaCore function to operate in symmetric, bidirectional mode. See Figures 3–2 and 3–3 on page 3–5.

If you set the **Port Type** to **Receiver only** or **Transmitter only**, you configure the MegaCore function to operate in unidirectional mode, transmitter, or receiver only. See Figures 3–4 and 3–5 on page 3–6.

If you set the **Port Type** to **Bidirectional**, but have the number of lanes set to a value other than zero, but not equal to the other function's value, you configure the MegaCore function to operate in asymmetric mode. See Figures 3–6 and 3–7 on page 3–7.

Figure 3–2. Symmetric Mode Block Diagram

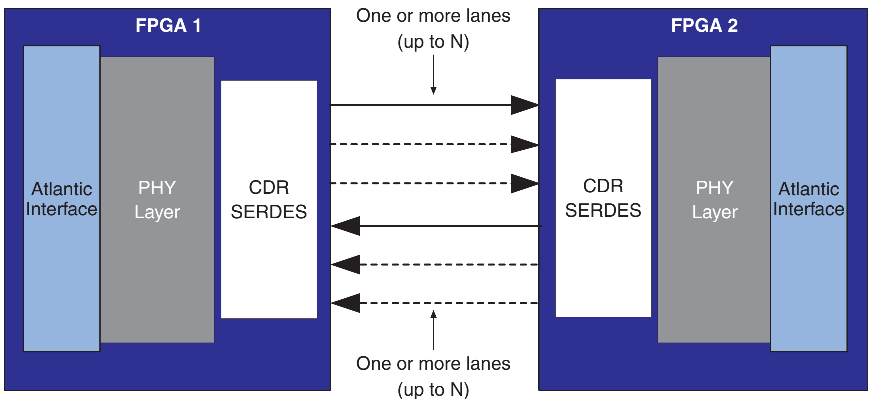


Notes to Figure 3–2:

(1) A full line indicates a mandatory lane.

(2) A dashed line indicates an optional lane.

Figure 3–3. Streaming Symmetric Mode Block Diagram

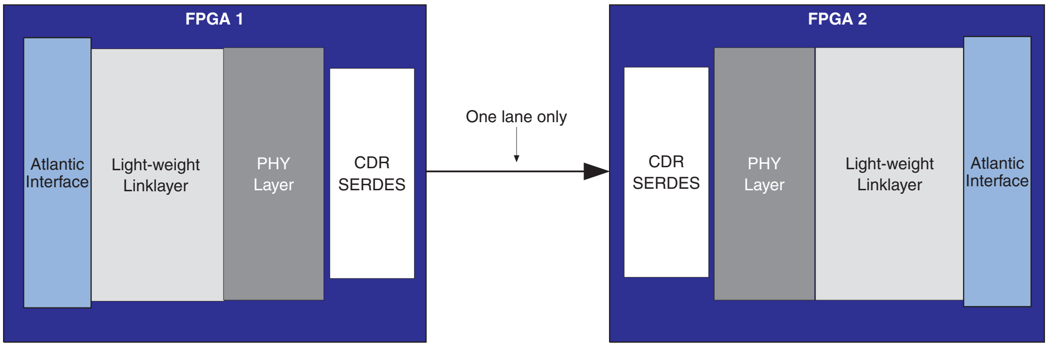


Notes to Figure 3–3:

(1) A full line indicates a mandatory lane.

(2) A dashed line indicates an optional lane.

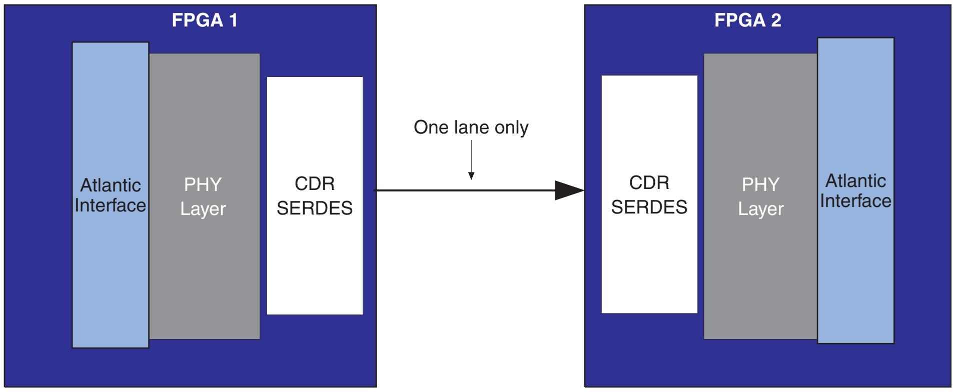
Figure 3–4. Simplex Mode Block Diagram



Note to Figure 3–4:

(1) A full line indicates a mandatory lane.

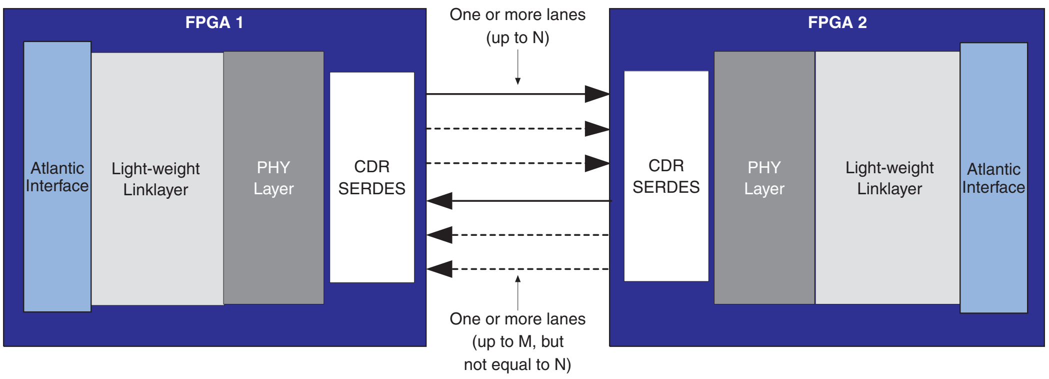
Figure 3–5. Streaming Simplex Mode Block Diagram



Note to Figure 3–5:

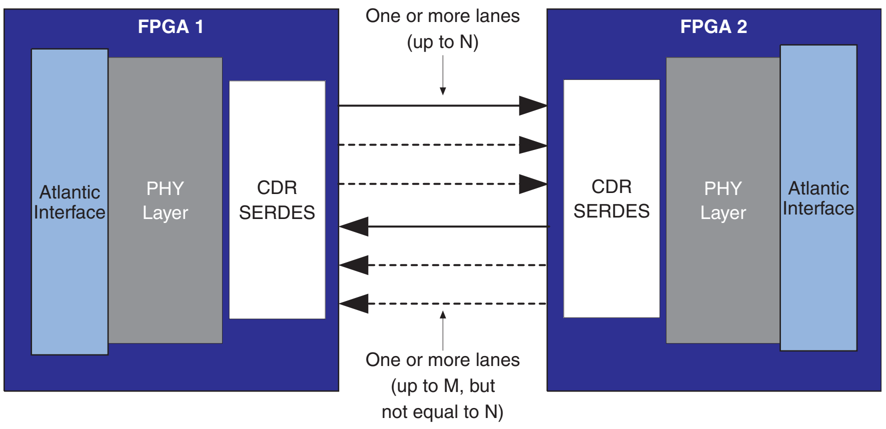
(1) A full line indicates a mandatory lane.

Figure 3–6. Asymmetric Mode Block Diagram



Notes to Figure 3–6:
(1) A full line indicates a mandatory lane.
(2) A dashed line indicates an optional lane.

Figure 3–7. Streaming Asymmetric Mode Block Diagram



Notes to Figure 3–7:
(1) A full line indicates a mandatory lane.
(2) A dashed line indicates an optional lane.

Self Synchronized Link Up

The receiver on the far end must synchronize itself to incoming data streams. To do so, it uses the self-synchronizing link state machine (LSM), a light-weight implementation that is especially useful when data is streaming. As there is no handshaking or exchange of status information between the receiver and transmitter, this parameter uses considerably fewer logic elements than the full-duplex link state machine. The self-synchronizing LSM can be used in all modes, except asymmetric mode, but this mode can only support one lane.

This parameter is enabled by default when the MegaCore function operates in unidirectional mode because the duplex LSM cannot be used when there is no return path.

The `ctrl_tc_force_train` signal must be asserted for the training patterns to be sent. Negate the signal once the adjacent receiver has locked, if this status information can be made available, or after a user-defined period of time when the link status of the adjacent receiver is not known or cannot be known. The LSM links up after receiving 64 consecutive valid, error-free characters. The link goes down after receiving four consecutive errors. At this time, the `ctrl_tc_force_train` signal should be reasserted until the receiver relocks.

The required hold time for the `ctrl_tc_force_train` signal largely depends on when the altgx megafunction completes the power-on reset cycle. Therefore, the self-synchronizing link-up state machine does not look at the incoming stream until the altgx reset is complete.

For example, the following procedure shows the altgx megafunction reset sequence in a Stratix GX device (you can also apply the same technique to Arria GX, Stratix II GX, and Stratix IV):

1. Wait for the pll_locked signal (`stat_tc_pll_locked`) to be asserted, which happens when the PLL in the altgx megafunction locks to the reference clock (`trefclk`). The reference clock must be characterized; 10 µS or less is normal.

2. Wait for the rx_freqlocked signal (`stat_rr_freqlock`) to be asserted, which happens when the altgx megafunction locks onto the serial stream; 5 ms or less is normal.

3. The Rx digital reset needs to complete; this reset normally takes one million internal `tx_coreclock` cycles after `rx_freqlocked` is asserted. The `stat_tc_rst_done` signal is asserted to indicate that the reset sequence has been completed.

   ☞ The normal time values are much shorter in simulation (For example, using the IP Functional Simulation Model), but not in gate-level simulation. Gate-level simulation uses the hardware equivalent times.

As you have full visibility of the above signals (via the SignalTap® II Logic Analyzer and the port list), you should characterize the timing of these signals to set up the size of your `ctrl_tc_force_train` counter. The MegaCore function also has a reset done status signal (`stat_tc_rst_done`) that can be useful for measurements. The following MegaCore function status output signals correspond to each step above:

- `stat_tc_pll_locked`
- `stat_rr_freqlock`
- `stat_tc_rst_done` (to see when `rx_digitalreset` has been negated).

After the reset controller has completed, the MegaCore function waits for the altgx megafunction aligner to detect and align the control (k28.5) character in the training sequence. Once the altgx megafunction detects this character, the count starts at every k28.5 that is received (basically, counting every training pattern). Once 64 error-free training sequences have been received, the MegaCore function reports linkup. Any errors (for example, disparity or 8B/10B errors) that are received reset the count, and the MegaCore function continues to wait until 64 error-free training patterns are received.

☞ The self-synchronizing LSM also locks onto the clock compensation sequence. Turning on the **Clock Compensation** option allows the receiver to automatically relock if the link goes down, therefore the transmitter is not required to assert `ctrl_tc_force_train` to retrain the link (which may be impossible in a unidirectional link because the transmitter does not necessarily detect that the receiver has lost the link).

Number of Lanes

Because each lane operates at the bit rate, you can increase the bandwidth by adding lanes. Adding lanes—up to a maximum of 16—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, thereby mitigating possible high-speed design issues and making it easier to meet performance. For Arria GX, the maximum number of lanes is 12 only. For the other device families, the maximum number is 16.

Scramble

Scrambling the data eliminates repeating characters, which affect the EMI substantially at high data rates. A linear feedback shift register (LSFR) is used as a pseudo-random number generator to scramble the data, using the $G(x) = X^{16} + X^5 + X^4 + X^3 + 1$ polynomial.

The transmitted bits are `XOR`ed with the output of the LFSR in the data stream. At the receiver, the data stream is again `XOR`ed with an identical scrambler to recover the original bits. To synchronize the transmitter to the receiver, the COM character initializes the LFSR with the initial seed of `0xFFFF XOR`ed with the lane number (LN).

Scrambling is recommended for data rates greater than 3,125 Mbps, and is optional for lower data rates (622 to 3,125 Mbps inclusive). This parameter applies only to the transmitter, and allows for scrambling (like CRC) to be enabled in one direction only, as required.

De-Scramble

This parameter applies only to the receiver, and allows for descrambling (like CRC) to be enabled in one direction only, as required. Descrambling is required if the incoming data stream is scrambled.

Broadcast Mode

If you enable the broadcast mode parameter for the transmitter, you configure the MegaCore function to use a single shared transmitter and multiple receivers in the master device, as shown in Figures 3–8 and 3–9. The number of receivers is determined by the number of lanes chosen for the slave receiver. The master transmitter uses its output lanes to broadcast identical messages to all slave receivers, and each slave responds individually by sharing the master’s lanes.

Figure 3–8. Broadcast Mode Block Diagram

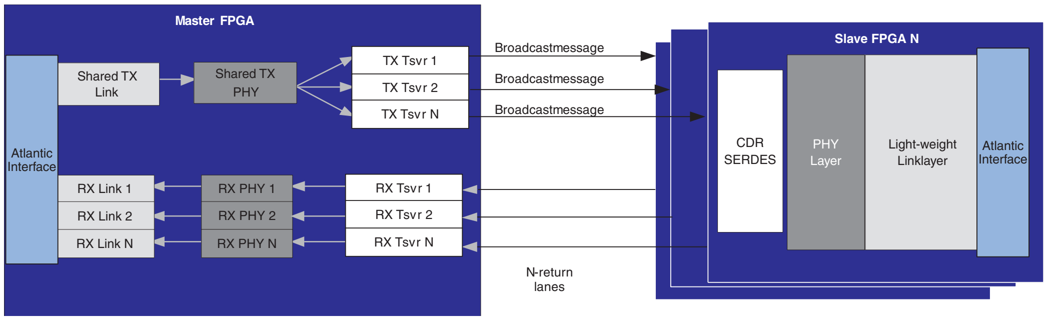
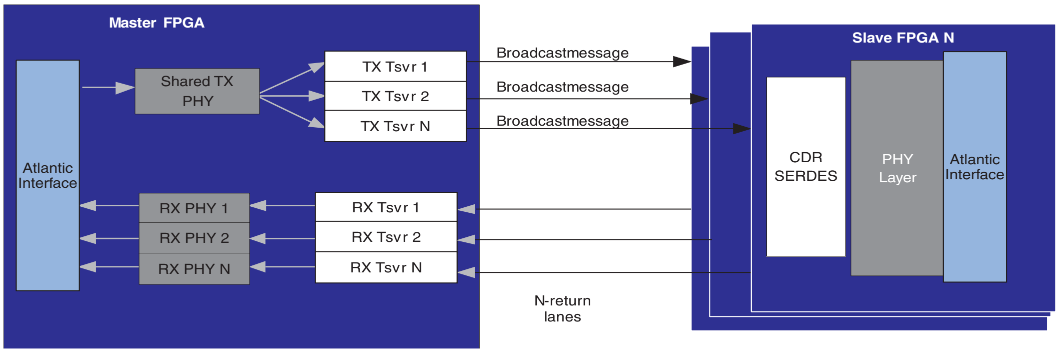


Figure 3–9. Streaming Broadcast Mode Block Diagram



Clock Compensation

The clock compensation value determines when the clock compensation sequence is inserted into, or deleted from, the high-speed serial data stream to compensate for ppm frequency differences between different clock crystals when the **Clock Compensation** option is enabled.

The frequency offset removal (`foffre`) block includes a FIFO buffer overflow status output signal: `err_rr_foffre_oflw.` If this signal toggles, you may need to adjust the ppm setting.

Lane Polarity and Order Reversal

The SerialLite II protocol optionally allows the link to recover from some connection problems. Lane polarity and lane order are reversed automatically.

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 a layout error or because it simplifies layout. The SerialLite II 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.

For training sequence one, the TID field normally read as /T1/ (D10.2) is read as /!T1/ (D21.5) when the lane polarity is inverted. Likewise for training sequence two, the TID field normally read as /T2/ (D5.2) is read as /!T2/ (D26.5) when the lane polarity is inverted. In these training sequences, the /COM/ character is followed by seven valid data characters. The last character of the sequence is used to determine the parity. If any of the parity identifiers in any lane is either /!T1/ (D21.5) or /!T2/ (D26.5), the receiver for that lane inverts the polarity.

☞ The Stratix GX device family is unable to reverse polarity. If the polarity of a pair is reversed, it is considered a catastrophic error. The core asserts the `err_rr_pol_rev_required` signal to identify this error condition.

Lane Order

It is possible that the order of lanes may be incorrect due to layout errors. It may also be reversed, with the most significant lane of one end of the link connected to the least significant lane of the other end, due to layout constraints. The SerialLite II logic always detects a lane order mismatch, and compensates for the 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.

The SerialLite II logic only corrects reversed lane order. If the lane order is scrambled, the receiving end cannot unscramble it. Example 3–1 illustrates two, possible four-lane systems.

Example 3–1. SerialLite II Lane Reversal

SerialLite II can reverse the following four-lane system:

```
Lane 0 -> Lane 3
Lane 1 -> Lane 2
Lane 2 -> Lane 1
Lane 3 -> Lane 0
```

SerialLite II cannot reverse the following four-lane system. The link will never come up:

```
Lane 0 -> Lane 1
Lane 1 -> Lane 3
Lane 2 -> Lane 0
Lane 3 -> Lane 2
```

Frequency Offset Tolerance

The **Enable frequency offset tolerance** parameter sets the value for the frequency offset tolerance (clock compensation). This parameter also determines whether the system is configured for synchronous or asynchronous clocking operation. If you enable this parameter, the values available are ±100 ppm and ±300 ppm.

Link Layer Configuration

This section describes the options available to parameterize the link layer of the SerialLite II MegaCore function variation.

Data Type:Packets

Packet mode for packet-based protocols. The data port expects data to arrive in packets, marked by asserting start of packet (SOP) at the beginning and end of packet (EOP) at the end of the packet. The receiver passes these packets to the user logic via the Atlantic interface, with the packet boundaries marked by SOP and EOP.

Data Type:Streaming

The regular data port allows data to be formatted as a stream or in packets. Streaming data has no beginning or end. It acts like an infinite-length packet and represents an unending sequence of data bytes. The only Atlantic signals present are `txrdp_ena`, `txrdp_dav`, and `txrdp_dat` (valid and data) in the transmitter, and `rxrdp_ena` and `rxrdp_dat` for a receiver instantiation. There is no backpressure for the receiver function; consequently, the user logic must accept the data when `rxrdp_ena` is high. There is only backpressure in the transmitter function if clock compensation is enabled (`txrdp_dav` is negated when the clock compensation sequence is inserted).

Once system link up is complete, your logic should provide data continuously. The SerialLite II MegaCore function does not encapsulate streaming data. Streaming mode does not include link-layer functions.

If this parameter is enabled, all link layer basic parameters, including data and priority ports, and buffering are disabled (grayed out).

Regular Data Port

A cut-through data flow is implemented for data packets. Packet data is transmitted as soon as enough data is received to fill a bonded CDR column, without waiting for the entire packet to be delivered to the transmitter. This approach provides the lowest latency. There is no packet size limitation.

Priority Port

The data flow for priority packets depends upon whether **Retry-on-error** is enabled.

Retry-on-error Disabled

A cut-through data flow is implemented for priority packets. Priority packet data is transmitted as soon as enough data is received to fill a bonded CDR column, without waiting for the entire packet to be delivered to the transmitter. This approach provides the lowest latency. There is no packet size limitation. As the name implies, priority packets have precedence over data packets. The SerialLite II MegaCore function inserts high priority packets within a data packet that is already in transmission (nesting packets).

Retry-on-error Enabled

A store-and-forward data flow is implemented for priority packet segments. Priority packets are broken into `SEGMENT_SIZE` bytes that are buffered and sent across the link. The transmission of data does not start until a segment or an end of packet has been delivered to the transmitter. Therefore, priority packets less than or equal to `SEGMENT_SIZE` bytes are buffered before transmission. This buffering is required to support the **Retry-on-error** option, which is only allowed for priority packets. As the name implies, priority packets have precedence over data packets. The SerialLite II MegaCore function inserts high priority packets within a data packet that is already in transmission (nesting packets). There is also no maximum packet size limitation.

If a packet is larger than a `SEGMENT_SIZE`, a full segment must be queued before it can be transmitted. This queueing may result in mid-packet backpressure on the priority port Atlantic interface. Segment interleaving, priority segments destined for different ports, is fully supported, as long as the address change occurs on a segment boundary.

Segment Size

Segment size is only applicable when the **Retry-on-error** parameter is turned on. Priority packets are broken into segments of `SEGMENT_SIZE` bytes and sent across the link. Priority packets less than or equal to `SEGMENT_SIZE` bytes and without an end marker are buffered before transmission.

The `SEGMENT_SIZE` parameter settings range from 8 to 2,048 bytes in 2^n increments, and the default value is 256 bytes.

Retry-on-error

The SerialLite II MegaCore function allows you to improve the bit error rate of your data by using the **Retry-on-error** parameter. This parameter is only available on the priority data port. The parameter provides for segments with errors to be retransmitted, so that only good segments are delivered to the Atlantic receive interface.

When the **Retry-on-error** parameter is turned on, all segments sent by the transmitter are acknowledged. There are two types of acknowledgement:

– ACK: The received segment is good and error-free.

– NACK: The received segment contains an error. If the **Retry-on-error** parameter is turned on, the transmitter retransmits all segments starting from the segment with errors. (If the **Retry-on-error** parameter is turned off, the receiver raises a data error.)

The segment buffers in the transmitting logic hold segments until they have been acknowledged. Once a segment has been acknowledged by `ACK`, it is released from the buffer so that the buffer can be used for another segment. If a segment is acknowledged by `NACK`, that segment and all segments sent after that segment are retransmitted.

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

The retry-on-error operation proceeds as follows:

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

2. Any data errors cause the segment to be acknowledged as errored (`NACK`). Once that happens, the receiver ignores all incoming data until it receives the retransmitted segment.

3. All segments are numbered internally with a segment ID. The receiver knows which segment it expects next, so if the next expected segment has been corrupted or lost, the next received segment has the wrong segment number and the receiver requests a retransmission of the sequence starting with the segment ID it was expecting.

4. The oldest outstanding segment to be acknowledged has an associated timer, set by the **Timeout** value on the **Link Layer** page in the MegaWizard Plug-In Manager. If an acknowledgment (`ACK` or `NACK`) is lost or corrupted in transit, the timer expires causing the affected segment and all subsequent segments to be retransmitted.

5. The transmitter knows which segment it expects to be acknowledged next. If the next acknowledgment is not for the expected segment, the transmitter infers that the expected acknowledgment was lost and retransmits the segment in question and all subsequent segments. Only segments that have the correct segment ID are buffered. The timer starts when the segment is identified as the next segment to be acknowledged.

6. 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 MegaWizard Plug-In Manager, 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. However, do not set the **Timeout** to be too short because then the system always times out and the link never remains up. The timeout value is based primarily on the round trip latency (that is, from the time a packet is sent to when the `ACK` is returned to that transmitter). The exact value of the round trip latency is undetermined, pending device characterization, but a value of 1,024 columns is recommended.

Implementation of the retry-on-error mechanism is optional for the priority port. If the **Retry-on-error** parameter is turned off, no segment acknowledgments are generated or expected, and all segments are transmitted without any acknowledgements from the receiver.

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

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

Option	Description
Turned on	Logic is created to acknowledge segments and retransmit segments when errors occur. Eight transmit segment buffers are created. Available only if the priority data port is enabled.
Turned off	Logic is not created to acknowledge segments. Available only if the priority data port is enabled. This is the default setting.

Retry-on-error Responses

Table 3–4 summarizes the response to various transmission errors.

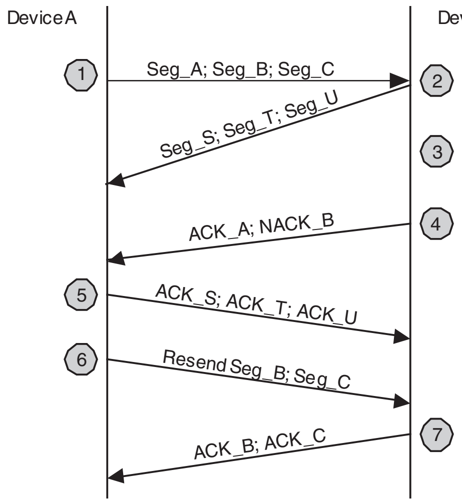
Table 3–4. Retry-on-error Responses

Error	Response
Invalid 8b/10b codes groups	Far end transmitter issues `NACK`
Running disparity errors	Far end transmitter issues `NACK`
Unsupported valid code groups	Far end transmitter issues `NACK`
CRC errored segments with {EGP} sequence	Far end transmitter issues `NACK`
Out of order segment	Far end transmitter issues `NACK`
Out of order acknowledgment	Near end transmitter starts re-send

Retry-on-error Operation Example

Figure 3–10 on page 3–18 shows an example of the retry-on-error operation.

Figure 3–10. Retry-On-Error Example



Notes to Figure 3–10

(1) Device A transmits Seg_A, Seg_B, and Seg_C to Device B.
(2) At the same time, Device B transmits Seg_S, Seg_T, and Seg_U to Device A.
(3) Device B properly receives Seg_A, but detects an error with Seg_B.
(4) Device B returns positive acknowledge for Seg_A, but requests retransmission of Seg_B. Device B discards all subsequently received segments until Seg_B is received again.
(5) Device A acknowledges the proper reception of Seg_S; Seg_T; and Seg_U.
(6) Device A resends all segments starting from Seg_B.
(7) Finally, Device B acknowledges the proper reception of Seg_B and Seg_C.

Flow Control

The SerialLite II MegaCore function provides the **Enable flow control** parameter as an optional means of exerting backpressure on a data source when data consumption is too slow. Use this parameter to ensure that the receive FIFO buffers do not overflow.

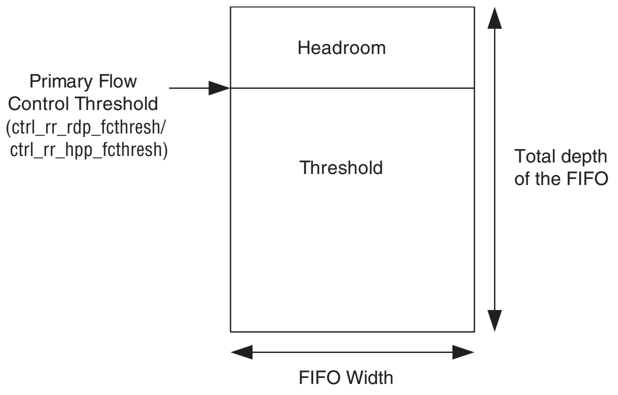
Flow control is only needed when the system logic on the receiving end of the link is reading the data more slowly than the system logic on the transmitting end of the link is sending data.

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

Flow Control Operation

When flow control is used, the FIFO buffer is structured as two sections, threshold and headroom. Figure 3–11 shows the FIFO buffer structure with flow control enabled.

Figure 3–11. FIFO Buffer Structure (Flow Control Enabled)


Headroom
Primary Flow Control Threshold (ctrl_rr_rdp_fcthresh/ ctrl_rr_hpp_fcthresh)
Threshold
Total depth of the FIFO
FIFO Width

The threshold value determines if a Flow Control `PAUSE` is requested. You control the size of this threshold by setting the flow control threshold per port using the MegaWizard interface to fall within the total depth of the FIFO. The value for the flow control threshold signals (`ctrl_rr_rdp_fcthresh` and `ctrl_rr_hpp_fcthresh`) must be within the total FIFO depth. The value must also ensure required headroom to compensate for the delays for the flow control request to take effect, and for the remaining data already in the system to be stored in the FIFO. Refer to section "Selecting the Proper Threshold Value" on page 3–23 for further analysis.

The total depth of the FIFO (in bytes) is derived by the MegaWizard interface using the following formula :

```
Total Depth = FIFO SIZE / (TSIZE * RX_NUMBER_LANES)
```

In this example, set the `FIFO SIZE` on the **Parameter Settings** tab, **Link Layer** page by selecting a value in the **Receiver** field of the **Buffer Size** section.

`TSIZE` and the `RX_NUMBER_LANES` are set on the **Physical Layer** page. Under **Data Settings**, select the `TSIZE` by selecting a number in the **Transfer size** option. To set `RX_NUMBER_LANES`, specify a value for the **Number of lanes** option in the **Receiver Settings** section.

If in this example you select a high-priority FIFO size of 1024B, and a `TSIZE` of 2 in a 4-Lane SerialLite II configuration, you compute the `Total Depth` as follows:

```
Total Depth =1024/2*4 =128
```

Based on the above result, for this example, you must set the **Threshold** value in the MegaWizard interface to be less than 128 elements. Set the **Threshold** value in the appropriate packet settings section on the **Link Layer** page.

When flow control is enabled, the SerialLite II 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 the data that has already been transmitted to be stored in the receive FIFO buffer. 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. Refer to section "Flow Control Operation Example" on page 3–21" for more details.

If the far receive FIFO buffer is still in breach of the threshold when the flow control refresh period timer expires, the far receiver automatically renews the `pause` to extend the flow control period. This renewal occurs until the fill level of far receive FIFO is no longer greater than the threshold. When the renewed flow control packet reaches the near transmitter before the current `pause` expires, the pause time is refreshed.

- This refresh time must be set so that the renewed flow control packets are received by the near transmitter before the current `pause` time completes. Set the value of **Refresh period** to be smaller than **Pause quantum time** in the **Priority Packet Settings** or **Data Packet Settings** section on the **Link Layer** page.

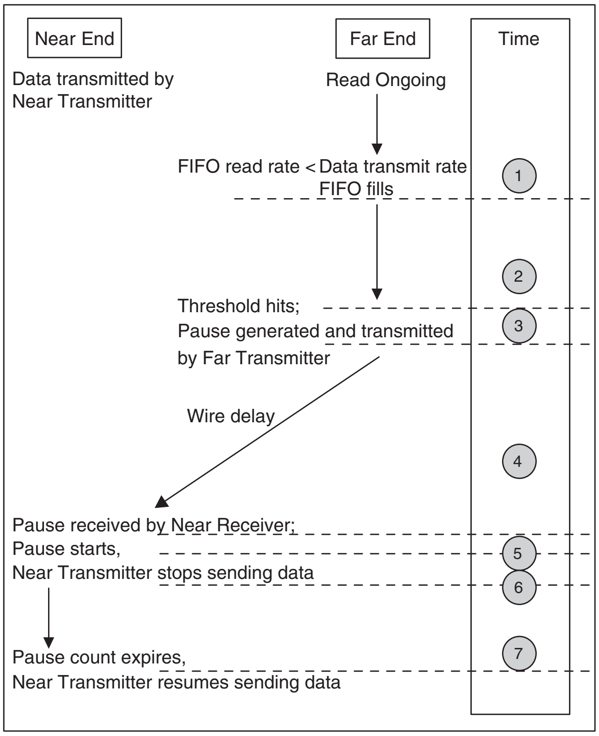
- If the Refresh period is small, more flow control packets are sent on the link, possibly degrading the performance of an alternate active port. This is a trade off for the link bandwidth performance.

To overcome head-of-line blocking, every port has its own flow control that suspends the flow of data to either the priority port or the regular data port, depending on the FIFO buffer status. For example, if the near transmitter receives a flow control pause request for the priority port, the data on the regular port is transmitted (as long as the regular port is not also being requested to pause).

Flow Control Operation Example

Figure 3–12 on page 3–22 shows an example of a flow control operation.

Figure 3–12. Typical Flow Diagram of FC Operation


Near End
Far End
Time
Data transmitted by Near Transmitter
Read Ongoing
FIFO read rate < Data transmit rate
FIFO fills
Threshold hits;
Pause generated and transmitted by Far Transmitter
Wire delay
Pause received by Near Receiver;
Pause starts,
Near Transmitter stops sending data
Pause count expires,
Near Transmitter resumes sending data
1
2
3
4
5
6
7

Notes to Figure 3–12

(1) Near transmitter starts sending data to far receiver when the link is up. The FIFO inside the far receiver reads the data. When the user logic on the receiving end of the link is reading the data out of the FIFO slower than the rate at which the data is being written into the FIFO, the FIFO starts to fill.

(2) The far receiver FIFO fill level breaches the Flow Control Threshold value.

(3) The far transmitter generates and sends the flow control packet with a `FC_TIME` pause request amount. There is some internal transmit latency (`tlate_fc_transmit`) for the flow control packet to hit the serial link.

(4) The flow control packet reaches the near receiver after some wire delay period (`t_wd`).

(5) There is some latency for the flow control packet to come from the serial link until the near receiver completes processing the packet (`tlate_fc_receive`).

(6) The near transmitter stops sending data to the far receiver either as soon as the flow control packet is received, or after the current active segment has been sent (for Priority packet with Retry-on Error enabled) for the specified pause duration. This latency accounts for the amount of additional data that has been already transmitted before the `PAUSE` request was received (`tlate_stop_data`).

(7) After the Pause quantum time specified by the users expired, the pause stops and the near transmitter continue sending the data (assuming that no other pause requests have been received).

Selecting the Proper Threshold Value

Table 3–5 on page 3–23 defines the specification value for flow control internal latency, as mentioned in the previous example. Use this information to determine the minimum FIFO threshold size avoiding starvation during the flow control. To calculate latency numbers in terms of time units, multiply the number in Table 3–5 by the `tx_coreclock` clock period.

Table 3–5. SerialLite II Flow Control Internal Latency

Internal Latency	Description	Latency Value (cycles)
`tlate_fc_transmit`	Latency that occurs during `RX FIFO` breach up to the point where the associated flow control link management packet is sent out on the link. This includes the time for the core to generate the link management packet and the time through the transceiver.	24
`t_wd`	Wire delay	*(1)*
`tlate_fc_receive`	Latency that occurs in the duration when the flow control link management packet reaches the transceiver pins until the the core processes the request.	23 + *deskew cycles (2)*
`tlate_stop_data`	Overall system core latency (indicates the amount of data that may still be in the system when the `PAUSE` begins). This data must still be stored in the RX FIFO.	Regular data: 20 + `spreadsheet_TX` + `spreadsheet_RX` Priority data: 20 + `seg_TX` *(3)* + `seg_TX` *(3)*+ `spreadsheet_TX` + `spreadsheet_RX`

Note to Table 3–5

(1) `t_wd` specifies the wire delay between the devices. This value depends on the data rate and trace lengths in the application.

(2) deskew cycles = 0 for single lane configuration;
deskew cycles = worst case lane to lane skew in the transceiver, refer to "SerialLite II Deskew Support" on page 4–7

(3) `seg_TX` and `seg_RX` are taken into account only for **Priority packets** with **Retry-on-error** feature. If a priority packet with retry-on-error feature is in transfer, flow control begins immediately after the current segment of the priority packet has been sent.
`seg_TX` = [segment size/`TSIZE* TX_NUMBER_LANES]`
`seg_RX` = [segment size/`TSIZE* RX_NUMBER_LANES]`
The **Segment size** value is specified by users in the **Parameters Settings** tab on the **Link Layer** page.

If you want to avoid FIFO starvation during flow control, then the **Threshold** must be set to a definite value. If you do not set it to a proper value, you can cause an inefficient system. The FIFO does not get new data when the FIFO fill level is breached, and a flow control is requested. If the FIFO does not hold enough data, the reading halts after the FIFO empties while waiting for the pause to expire.

The proper threshold value can be derived by subtracting the depth of the FIFO from the total latency.

Total Latency = [`tlate_fc_transmit + t_wd + tlate_fc_receive + tlate_stop_data`] cycles

☞ The ratio between one element and one cycle is equal to one. When you write one element to the FIFO, it takes one clock cycle. Therefore one cycle is one element.

Therefore, the **Threshold value** should be set based on the following formula:

Threshold value = Total Depth of FIFO (elements) - Total Latency (clock cycles)]

Selecting the Proper Pause Duration/Refresh Value

Activation of flow control causes a pause in transmission. You can specify the duration of this pause in terms of columns. You can specify a pause duration from to 8 to 2,040 columns. In elements, this value is 8/`TSIZE` to 2040/`TSIZE` elements. Set the pause duration based on the rate that 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 is too long.

As an example, assume a theoretical pause needs to be 100 elements long. As a designer, you would not likely know that at design time, so you must estimate a reasonable value. The effect of a `TSIZE-2`, 120-element pause (240 columns on the GUI) causes more delay than needed. However, an 80-element delay (160 columns on the GUI) results in the pause being renewed after 80 elements, for a total 160 elements of delay, even longer than the 120-element pause.

The flow control refresh period determines the number of columns before a flow control packet can be retransmitted (for example if a flow control link management packet is lost or corrupted). This refresh period must be less than the pause quantum time. The packet is retransmitted if the FIFO buffer is still breached.

For example, after the first flow control link management packet is sent out (256 columns on the GUI), the far transmitter monitors the `stat_tc_rdp_thresh_breach` or `stat_tc_hpp_thresh_breach` (which indicates that the RX FIFO is still breached). If these signals are still asserted after the `FC_Refresh_Period` (208 columns on the GUI), the far transmitter generates another flow control packet (256 columns on the GUI) and sends it out, `stat_fc_hpp_retransmit` or `stat_tc_fc_hpp_retransmit` is asserted.

External Flow Control (When RX FIFO Size is 0).

The SerialLite II MegaCore function supports an external flow control when the RX FIFO size is zero. The `rxrdp_dav` and `rxhpp_dav` input signals are provided to activate flow control to pause the data transmission when the corresponding **Regular** port or **Priority** data port is selected.

Drive `rxrdp_dav` low when the fill level of your external FIFO has been breached; this action triggers the flow control pause request. When this signal is high, no flow control requests is generated.

Transmit/Receive FIFO Buffers

The transmit FIFO buffers are used by the transmitting end of the SerialLite II link to store data to be transmitted across the high-speed serial link. The receive FIFO buffers are used by the receiving end of the SerialLite II link to store data for presentation to the Atlantic interface and eventual consumption by the system logic.

The SerialLite II MegaCore function automatically sets the width of the receive FIFO buffers at `TSIZE` bytes per lane. You specify the buffer size in the MegaWizard interface.

FIFO Buffer Size

The size of the FIFO buffers is based on the factors listed in Table 3–6.

Table 3–6. Factors Affecting Receive FIFO Buffer Size

Factor	Description
Flow control	If flow control is enabled, the FIFO buffer size should change to account for the thresholds that must be set.
Pause duration	When optimizing against starvation during flow control, the pause duration affects the FIFO buffer size.

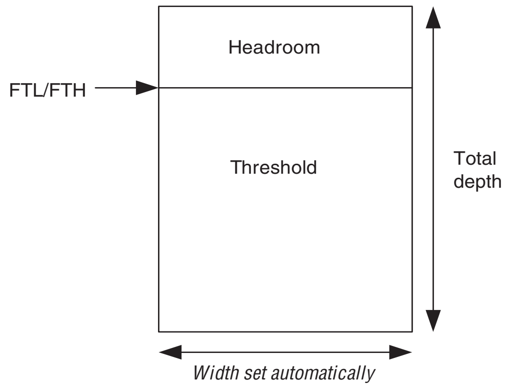
Table 3–6. Factors Affecting Receive FIFO Buffer Size

Factor	Description
Number of packets (and packet sizes)	If you want to use a store-and-forward FIFO (using the `eop_dav` and a high threshold), the FIFO must be big enough to hold a full packet at minimum.
Wire delay and bit rate	The wire propagation delay and the bit rate change the wire latency, which must be accommodated if flow control is used.

FIFO Buffer Structure

Figure 3–13 shows the Atlantic FIFO buffer structure.

Figure 3–13. Atlantic FIFO Buffer Structure



The FIFO buffer threshold low (`ctl_rxrdp_ftl/ctl_rxhpp_ftl`) value for receiver variations controls when the `rxrdp_dav/rxhpp_dav` signals are asserted for the read side of the FIFO buffer, respectively. If the fill level of the buffer is higher than the FTL value, the `rxrdp_dav/rxhpp_dav` signal is asserted indicating that there is a burst of data available.

The receiver Atlantic FIFO buffers include an end-of-packet based data available feature which can be turned on by asserting the `ctl_rxrdp_eopdav/ctl_rxhpp_eopdav` signals. The end-of-packet feature determines whether the `dav` remains high: if the signal is asserted, and there is an end-of- packet beneath the FTL threshed, the `dav` signal remains high until the end-of-packet is read out of the FIFO. Otherwise, if the signal is not asserted, the `dav` signal only remains high when the fill level of the buffer is higher than the FTL value.

☞ There is no requirement to wait for the `rxrdp_dav` signal to be asserted; you can read from the buffer at any time by asserting the `rxrdp_ena` signal at all times and qualifying the data with the `rxrdp_val/rxhpp_val` signal. The FIFO has built-in underflow protection, such that an underflow condition does not exit.

The FIFO buffer threshold high (`ctl_txrdp_fth/ctl_txhpp_fth`) value for transmitter variations controls when the `txrdp_dav/txhpp_dav` signals are asserted and deasserted for the write side of the FIFO buffer, respectively. The `txrdp_dav` signal indicates when there is room available to write new data into the FIFO buffer, and is asserted when the fill level of the FIFO is less than the FTH setting, and deasserted when the fill level of the FIFO is greater than the FTH.

For example, if FTH is five, and the fill level is four, the `txrdp_dav/txhpp_dav` signal is high, indicating that the user can write data into the FIFO. If the fill level for this example is six, the `txrdp_dav/txhpp_dav` signal is low, indicating that the user should stop writing data into the FIFO.

☞ The threshold units are in FIFO elements.

Data Integrity Protection: CRC

If you need error protection, you may add CRC checking to your packet. The CRC is automatically generated in transmission and is automatically checked on reception. On the data port, a CRC check failure results in the packet being marked as bad using the `rxrdp_err/rxhpp_err` signal on the Atlantic interface. You decide independently for each port whether CRC usage is enabled.

16-Bit Versus 32-Bit CRC

The SerialLite II MegaCore function supports 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 polynomial:

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

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

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

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

Tables 3–7 and 3–8 show the different CRC options.

Table 3–7. 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.

Table 3–8. CRC Type Options

Option	Description
16-bit	Generates a two-byte CRC. Adequate for packets of around 1 KBytes or smaller. This is the default algorithm once CRC usage has been enabled.
32-bit	Generates a 4-byte CRC. Should only be used for packets larger than about 1 KBytes or when extreme protection is required, because it is resource-intensive.

Transceiver Configuration

This section describes the optional, configurable Altera altgx gigabit transceiver megafunctions. The transceiver megafunction offers several configuration options that can be set based on board-level conditions, design constraints, or other application-specific requirements, to ensure the proper operation of the serial link.

For more information on Altera gigabit transceiver (altgxb) megafunction, refer to the *Stratix GX Transceiver User Guide* in volume 2 of the *Stratix GX Device Handbook*.

For more information on Altera gigabit transceiver (altgx) megafunction, refer to the *Stratix II GX Transceiver User Guide* in volume 2 of the *Stratix II GX Device Handbook* and the *Stratix IV Transceiver Architecture* in volume 2 of the *Stratix IV Device Handbook*.

Voltage Output Differential (VOD) Control Settings

The Stratix GX, Stratix II GX, and Stratix IV transceivers allow you to set the VOD to handle different length, backplane, and receiver requirements. A range from 400 to 1,600 mV is supported for Stratix GX devices. The range is 200 to 1,400 mV in Stratix II GX and Stratix IV. Arria GX has a fixed value, which cannot be changed. The range is decoded using a 3-bit value and determined by the on-chip transmitter programmable termination values.

Table 3–9 shows how the VOD value chosen in MegaWizard interface corresponds to the mV value. The VOD value is set to 0 by default.

Table 3–9. VOD Control Settings

VOD Value (Per Lane)	100 Ohms (mV) Stratix GX	100 Ohms (mV) Stratix II GX	
		1.2 V	1.5 V
0	400	N/A	200
1	800	320	400
2	1,000	480	600
3	1,200	640	800
4	1,400	800	1,000
5	1,600	960	1,200
6	N/A	N/A	1,400

This parameter is disabled when the number of lanes in the transmit direction is equal to zero. For Arria GX, this parameter cannot be changed.

Pre-Emphasis Control Settings

The programmable **Pre-emphasis** setting boosts the high frequencies in the transmit data signal, which may be attenuated by the transmission medium. The **Pre-emphasis** setting maximizes the data eye opening at the far-end receiver, which is particularly useful in lossy transmission mediums. **Specify pre-emphasis control setting** parameter is set to 0 by default.

This parameter is disabled when the number of lanes in the transmit direction is equal to zero.

For Stratix GX devices, the 0–5 pre-emphasis control values (and what these integer numbers ultimately translate into) can be found in the the *Stratix Gigabit Transceiver User Guide* in volume 2 of the *Stratix GX Device Handbook*.

For Stratix II GX and Stratix IV GX devices, the pre-emphasis control values supported are 0,1,2,3,4, and 5. For 0, pre-emphasis is turned off. For 1, the pre-emphasis is the maximum negative value. For 2, pre-emphasis is the medium negative value. The value 3 is a special value in which only the first post-tap is set (set to the maximum), while the other taps are off. A value of 4 yields a medium positive value, while 5 sets the pre-emphasis values to the maximum positive supported values. For Arria GX, the **Pre-emphasis** setting cannot be changed.

Transmitter Buffer Power (VCCH)

This setting is for information only and is used to calculate the VOD from the buffer power supply and the transmitter termination to derive the proper VOD range. The selections available for Stratix II GX are 1.2 V and 1.5 V. The 1.2 V setting is only available for data rates less than 3,125 Mbps. The VCCH value is fixed at 1.4 V for Stratix IV and 1.5 V for Stratix_GX and Arria GX.

Equalizer Control Settings

The transceiver offers an equalization circuit in each receiver channel to increase noise margins and help reduce the effects of high frequency losses. The programmable **Equalizer** compensates for inter-symbol interference (ISI) and high frequency losses that distort the signal and reduce the noise margin of the transmission medium by equalizing the frequency response.

For Stratix II GX and Stratix IV GX devices, the equalization control values supported are 0, 1, 2, 3, and 4. These values correspond to lowest/off (0), between medium and lowest (1), medium (2), between medium and high (3), and high (4). For Arria GX, the equalization cannot be changed.

For Stratix GX devices, Table 3–10 on page 3–31 shows how the receiver equalizer value chosen in MegaWizard interface corresponds to the MegaCore function’s input signal. Some values are reserved and should not be configured.

The equalizer value is set to 0 by default.

Table 3–10. Equalizer Control Settings

Receiver Equalizer Value	Equalizer Incoming Signal
0	`3'b000`
1	`3'b010`
2	`3'b100`
3	`3'b101`
4	`3'b111`

Bandwidth Mode

The transmitter and receiver PLLs in the altgxb megafunction offer programmable bandwidth settings. The PLL bandwidth is the measure of its ability to track the input clock and jitter, determined by the -3 dB frequency of the PLL's closed-loop gain.

The transmitter offers two settings: **high** or **low**. The receiver offers three settings: **high**, **medium**, or **low**.

- The **high** bandwidth setting provides a faster lock time and tracks more jitter on the input clock source which passes it through the PLL to help reject noise from the voltage control oscillator (VCO) and power supplies.
- The **low** bandwidth setting filters out more high frequency input clock jitter, but increases lock time. The PLL is set to the low setting by default.
- The **medium** setting balances the lock time and noise rejection/jitter filtering between the high and low settings.

If the number of lanes in the transmit or receive direction is equal to zero, the bandwidth mode for that direction is disabled. This parameter is also disabled for Arria GX.

Starting Channel number

The range for the dynamic reconfiguration starting channel number setting is 0 to 156, in multiples of four. This range is in multiples of four because the dynamic reconfiguration interface is per transceiver block.

The range of 0 to 156 is the logical channel address, based purely on the number of possible alt2gxb instances. This parameter is only applicable for the Stratix II GX device family.

Instantiating an alt2gxb Reconfiguration Block

The alt2gxb interface allows you modify the parameter interface with a reconfiguration block. When you use a Stratix II GX device, the `alt2gxb_reconfig` block lets you dynamically reconfigure the following physical media attachment (PMA) settings:

- pre-emphasis
- equalization
- VOD

The `alt2gxb_reconfig` block is not instantiated, but the MegaWizard- generated wrapper provides the ports that interface to the `alt2gxb_reconfig` block. If you choose to use an `alt2gxb_reconfig` block, you must instantiate the `alt2gxb_reconfig` block and connect the associated signals to the corresponding SerialLite II MegaCore function top level signals (tie the `reconfig_fromgxb` and, `reconfig_clk` and `reconfig_togxb` ports to the `alt2gxb_reconfig` block).

If you are not using the `alt2gxb_reconfig` block, you can ignore the `reconfig_fromgxb` bus (do not connect the port to any logic). The `reconfig_clk` should be tied to `1'b0` and the `reconfig_togxb` port should be tied to `3'b010`.

altgx Support Signals

This section describes the altgx support signals, which are only present on variants that use the Arria GX, Stratix II GX, or Stratix IV integrated PHY. They are connected directly to the altgx instance. In many cases these signals must be shared with altgx instances that are implemented in the same device. The following signals exist:

- `cal_blk_clk`
- `reconfig_clk`
- `reconfig_togxb`
- `reconfig_fromgxb`
- `gxb_powerdown`

Table 3–11 describes these altgx support signals.

Table 3–11. altgx Support Signals

Signal	I/O	Description
`cal_blk_clk`	I	The `cal_blk_clk` input signal is connected to the altgx calibration block clock (`cal_blk_clk`) input. All instances of altgx in the same device must have their `cal_blk_clk` inputs connected to the same signal because there is only one calibration block per device. This input should be connected to a clock operating as recommended by the *Stratix II GX Device Handbook*.
`reconfig_clk`	I	The `reconfig_clk` input signal is the altgx dynamic reconfiguration clock. This signal must be connected as described in the *Stratix II GX Device Handbook* if the altgx dynamic reconfiguration block is used. Otherwise, this signal must be set to `1'b0`.
`reconfig_togxb`	I	The `reconfig_togxb[2:0]` input bus is the altgx dynamic reconfiguration data input. This signal must be connected as described in the *Stratix II GX Device Handbook* if the altgx dynamic reconfiguration block is used. Otherwise, this signal must be set to `3'b010`.
`reconfig_fromgxb`	O	The `reconfig_fromgxb` output signal is the altgx dynamic reconfiguration data output. This signal must be connected as described in the *Stratix II GX Device Handbook* if the altgx dynamic reconfiguration block is used. If no external `altgx_reconfig` block is used, then you must leave this signal unconnected.
`gxb_powerdown`	I	`gxb_powerdown` resets and powers down all circuits in the transceiver block. This signal does not affect the refclk buffers and reference clock lines. All the `gxb_powerdown` input signals of cores placed in the same quad should be tied together. The `gxb_powerdown` signal should be tied low or should remain asserted for at least 2 ms whenever it is asserted.

Error Handling

The SerialLite II MegaCore function does error checking and has an interface to view local errors. The errors are categorized, and the effect of an error depends on the type of error that occurs.

The SerialLite II MegaCore function has three error types:

- Data error
- Link error
- Catastrophic error

The causes and results of these errors are summarized in Table 3–12.

Table 3–12. Error Summary

Error Type	Cause	Action
Catastrophic	● Link state machine cannot reverse polarity ● Link state machine cannot reorder lanes	SerialLite II enters nonrecoverable state
Link	● Eight consecutive {|TS1|} sequences received in all lanes simultaneously ● Loss of character alignment ● Loss of lane alignment ● Loss of characters from underflow/overflow ● Data error threshold exceeded ● Retry-on-error timer expired three times	Trigger link initialization
Data	● Invalid 8b/10b codes groups ● Running disparity errors ● Unsupported valid code groups ● Link protocol violation ● LMP with BIP error ● CRC error ● Unexpected channel number ● Out of order packet ● Out of order acknowledgment (if retry-on-error enabled)	Two possibilities: ● If **Retry-on-error** is enabled and the packet is a priority packet, request retransmission ● Otherwise, mark the packet as bad and forward it to the user link layer
Packets Marked Bad	{EBP} marked packet	Received packet is marked as bad and forwarded to the user link layer

Error signals, such as `txrdp_err` and `txhpp_err`, are asserted by user logic. When `txrdp_err` is asserted with `txrdp_eop`, the packet is marked with the end of bad packet (EBP) marker. The `txrdp_err` signal is ignored when it is not asserted with `txrdp_eop`.

When the `txhpp_err` is asserted and the **Retry-on-error** feature is turned off, the packet is marked with the EBP marker. When the `txhpp_err` is asserted and the **Retry-on-error** feature is turned on, the packet is not transmitted and is silently dropped.

Optimizing the Implementation

There are a number of steps you can take to optimize your design, depending on your goals. The features selected in your SerialLite II configuration have a substantial impact on both resource utilization and performance. Because of the number of different combinations of options that are available, it is difficult to generalize the performance or resource requirements of a design. In addition, the performance of a SerialLite II link in isolation is different from the performance of the same link instantiated alongside large amounts of other logic in the device.

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 the SerialLite II MegaCore function 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.

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).
- CRC—the CRC generation and checking logic degrades 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.
- 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 to the required frequency, try running different placement seeds. This technique often yields a better result. For information on seed specification and improving speed, you can refer to the *Command-Line Scripting* chapter and the *Design Space Explorer* chapter in volume 2 of the *Quartus II Handbook* respectively.

Limiting Fanout

Depending on the number of lanes and the size of memories you choose, fanout can impact performance. 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 volume 1 of the *Quartus II Handbook* for more information on limiting fanout.

Floorplanning

The SerialLite II MegaCore function does not come with any placement constraints. The critical paths depend on where the Fitter places SerialLite II logic in the device, as well as the other logic in the device. You can use standard floorplanning techniques to improve performance. Refer to volume 2 of the *Quartus II Handbook* for more information on floorplanning.

Minimizing Logic Utilization

The amount of logic required for a SerialLite II link depends heavily on the features you choose.

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.
- Flow control—this feature requires logic to monitor the FIFO buffer levels and to generate and act upon `PAUSE` instructions.
- Streaming mode—use this mode if packet encapsulation is not required. The link-layer portion of the SerialLite II MegaCore function contains a significant amount of logic, which is reduced to zero in streaming mode.

Minimizing Memory Utilization

The amount of memory required for a SerialLite II link depends heavily on the features you choose. 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:

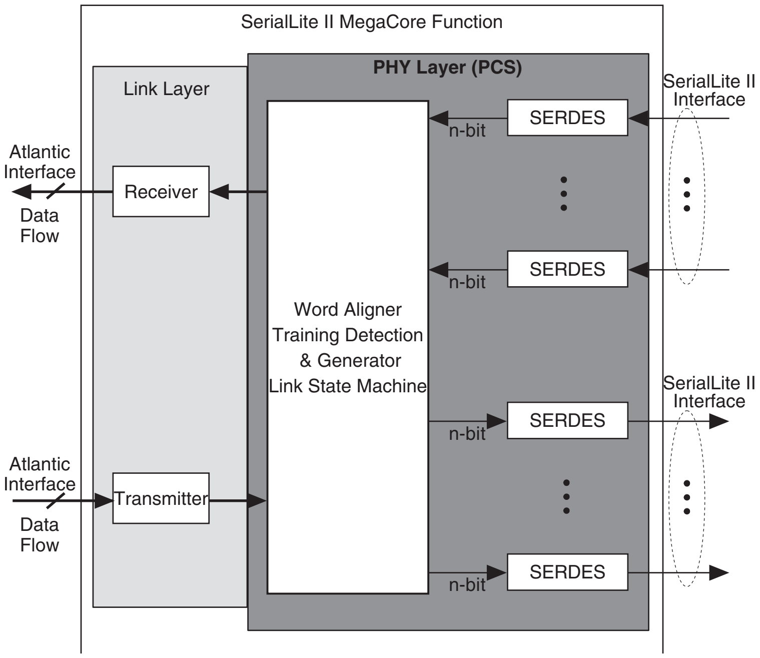
- Lane count—this establishes the bus widths internally, and most memories used scale almost directly with the number of lanes selected. Running fewer lanes at higher bit rates, if possible, uses less memory (but places more of a burden on meeting performance).
- Receive FIFO buffer size—you can minimize memory usage by not adding significant amounts of margin to the minimum specified sizes.
- Use streaming mode if packet encapsulation is not required. The link-layer portion of the SerialLite II MegaCore function contains a significant amount of logic, which is reduced to zero in streaming mode.

4. Functional Description

The SerialLite II 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. Figure 4–1 shows a block diagram of the SerialLite II MegaCore function.

See Chapter 5, Testbench for more information on the testbench.

Figure 4–1. SerialLite II MegaCore Function Block Diagram



As shown in Figure 4–1, the SerialLite II MegaCore function is divided into two main blocks: a protocol processing portion (data link layer) and a high-speed front end (physical layer). The protocol processing portion features Atlantic™ FIFO buffers for data storage or clock domain crossing, as well as data encapsulation and extraction logic. The high-speed front end contains a link state machine (LSM) and serializer/deserializer

(SERDES) blocks. The SERDES blocks contain optional high-speed serial clock and data recovery (CDR) logic implemented with high-speed serial transceivers.

Interface Overview

The SerialLite II MegaCore function has two interfaces, the Atlantic interface and a high-speed serial interface.

Atlantic Interface

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

The width of the Atlantic interface is determined by the number of lanes and the transfer size.

The SerialLite II 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. Also, in the transmit direction of each type of port, an Atlantic dual clock domain FIFO buffer is implemented. The receiver dual clock domain Atlantic FIFO buffer is optional.

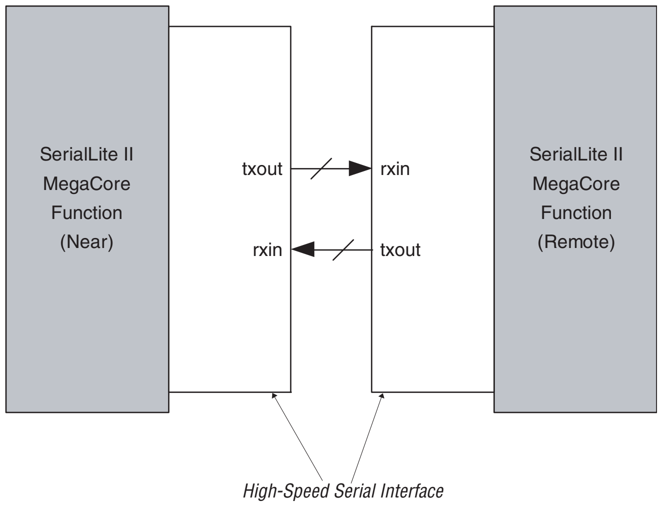
The SerialLite II MegaCore function is an Atlantic interface slave when the Atlantic FIFO buffer is implemented (when the function is not in streaming mode, and the buffer size is not zero). Otherwise, the SerialLite II MegaCore function is an Atlantic interface master. This user guide refers to the logic that drives data into the SerialLite II MegaCore function or receives data from the SerialLite II MegaCore function as the *system logic*. The Atlantic interface signals are described in Table 4–4 on page 4–25.

For more information on this interface, refer to the *FS 13: Atlantic Interface*.

High-Speed Serial Interface

The high-speed serial interface always appears at the external device pins. The high-speed interface consists of the differential signals that carry the high-speed data between the two ends of a link, as shown in Figure 4–2.

Figure 4–2. High-Speed Serial Interface Connections


SerialLite II MegaCore Function (Near)
txout
rxin
rxin
txout
SerialLite II MegaCore Function (Remote)
High-Speed Serial Interface

The high-speed serial interface signals are detailed in Table 4–2 on page 4–22.

Clocks and Data Rates

A SerialLite II link has two distinct clock rates: the core clock rate and the bit rate. The core clock rate is the rate of the clock the internal logic is running at. This clock controls the FPGA logic and is a derived clock from the phase-locked loops (PLLs). The transmitter and receiver both have their own core clocks, `tx_coreclock` and `rrefclk` respectively.

To determine the clock frequency required for `tx_coreclock` and `rrefclk`, use the following formula:

Core clock frequency = `Data Rate (Mbps) / (TSIZE * 10)`

For example, if the data rate is 3125 Mbps, and the `TSIZE` is 2, then:

Core clock frequency = `3125 / (2*10) = 156.25 MHz`

Aggregate Bandwidth

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

For example, calculate the bandwidth for a variation using 8B/10B encoding and an internal data path of 8 bits (transfer size is equal to 1), and the number of lanes is equal to 4.

In this mode, the input data bus into the processor portion is 36 bits wide (32 bits of raw data and 8 bits of control information). With the additional bits per byte (due to 8B/10B encoding) for control information, the data bus size being transmitted from the byte alignment logic into the protocol-processing portion of the MegaCore function is equal to the number of lanes × 10 (due to 8B/10B encoding). Thus for 4 lanes, the data bus size is equal to 40 bits (4×10 =40).

For example, a 32-byte packet. Count the number of 32-bit wide rows that are transmitted into the protocol-processing portion. The result is 8 rows (32 bytes/4 bytes) of solid data, plus one additional row for the start-of-packet marker row and the end-of-packet marker row (no CRC) which equals 9 rows of 40 bits.

For a 32-byte packet, given a link rate of 800 Mbps × 4 = 3.2 Gbps, the transfer is equal to the following:

- data bits: 256
- bits sent: 360
- 256/360 × 3.2 = 2.276 Gbps

For 64-byte packets, the transfer is the following:

- data bits: 512
- bits sent: 680
- 512/680 × 3.2 = 2.409 Gbps

For 128-byte packets, the transfer is the following:

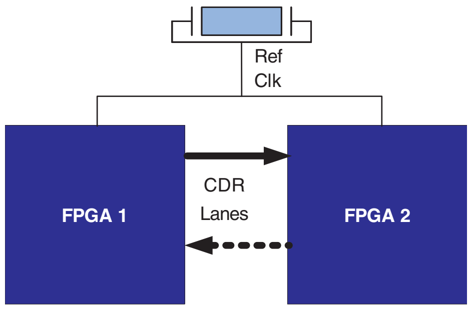
- data bits: 1,024
- bits sent: 1,320
- 1,024/1,320 × 3.2 = 2.482 Gbps

External Clock Modes

You can configure the SerialLite II MegaCore function to use one of two clock modes: synchronous or asynchronous.

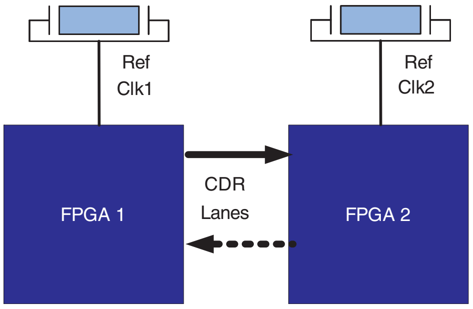
A synchronous configuration is typically used for a link where both ends are on the same board or on two boards driven by the same system clock (see Figure 4–3).

Figure 4–3. Synchronous Mode



An asynchronous configuration is typically used when the two ends of the link are on different boards, each having its own independent clock source (see Figure 4–4).

Figure 4–4. Asynchronous Mode



SerialLite II Internal Clocking Configurations

This section contains diagrams illustrating internal clocking configurations.

For Stratix II GX configurations with more than one lane, you must ensure that the PPM clock group settings enable the internal phase compensation FIFOs within the transceiver block.

Note that when you generate a custom MegaCore function, a Tcl script, named <*variation name*>**_constraints.tcl**, is generated that contains the assignments in Example 4–1. These constraints are automatically written to your project directory when you run the generated Tcl script. If you do not use the generated Tcl script, you must specify appropriate assignments manually. As Example 4–1 shows, you can enter these assignments directly in the Tcl console window.

Example 4–1. Tcl Code to Set Instance Assignments

if (RX_NUM_LANES > 1 and Stratix II GX)

```
{

set_instance_assignment -name GXB_0PPM_CLOCK_GROUP_DRIVER 1 -to\
*rx_clkout_wire[0]
set_instance_assignment -name GXB_0PPM_CLOCK_GROUP 1 -to\
*xcvr2_inst|alt2gxb:\
|alt2gxb_component|channel_rec[*].receive

}
```

if (TX_NUM_LANES > 1 and Stratix II GX)

{

```
set_instance_assignment -name GXB_0PPM_CLOCK_GROUP_DRIVER 0 -to\
*tx_clkout_int_wire[0]
set_instance_assignment -name GXB_0PPM_CLOCK_GROUP 0 -to\
*xcvr2_inst|alt2gxb:alt2gxb_component|channel_tx[*].transmit
```

}

SerialLite II Deskew Support

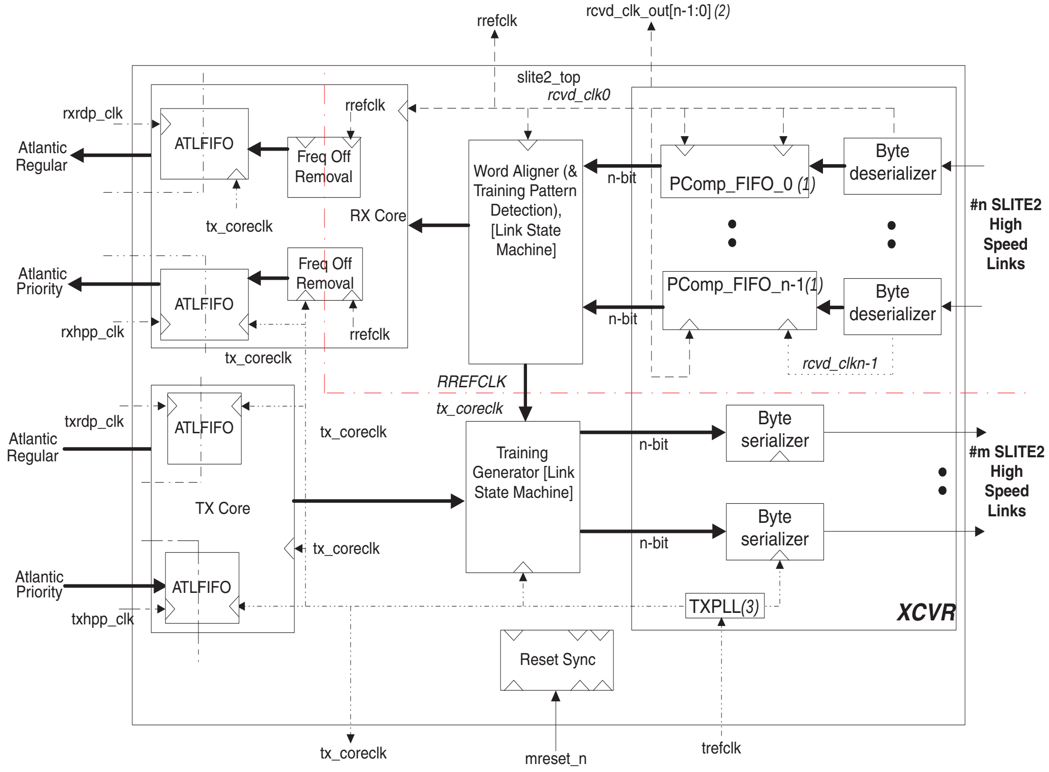
The Table 4–1 defines the parameters for the maximum receiver lane–to–lane deskew tolerance for the SerialLite II MegaCore as specified at the FPGA pins. You can use this information to ensure trace length differences do not exceed the timing budget. The values include worst case lane–to–lane skew in the transceivers. To calculate in terms of time units, multiply the value in Table 4–1 by the `tx_coreclock` clock period.

Table 4–1. SerialLite II Deskew Tolerance

Transfer Size	Max Deskew (Cycles)
1	14
2	6
4	2

Figures 4–5 through Figure 4–10 show the MegaCore function clock structures, which vary based on the configuration parameters.

Figure 4–5. Full-Featured Clock Structure



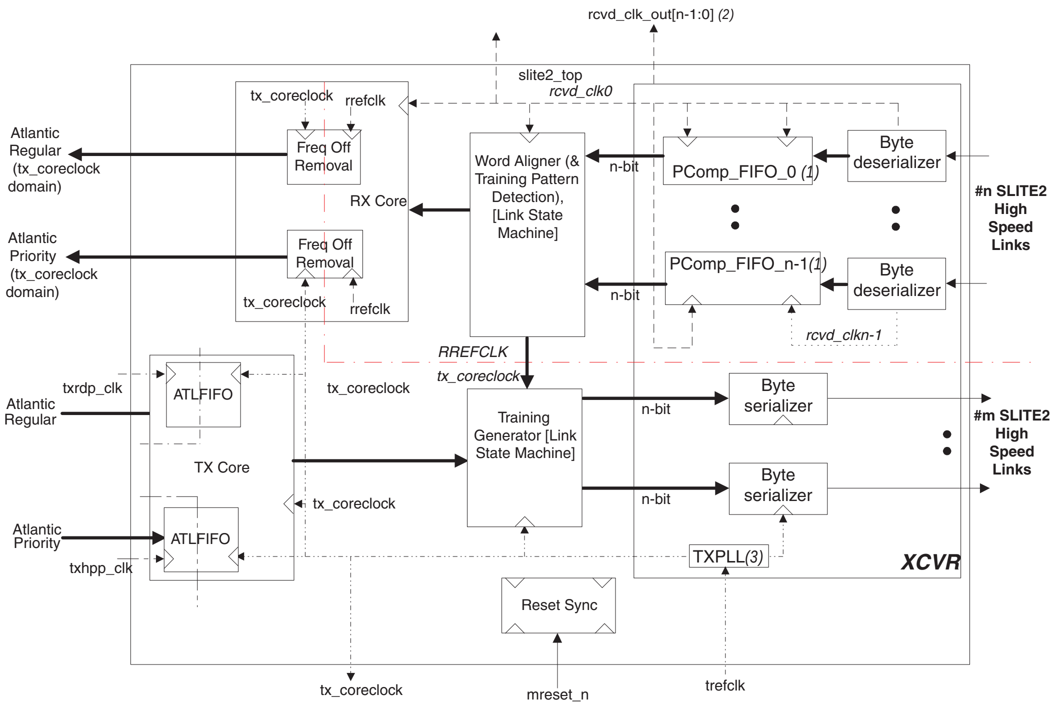
Notes to Figure 4–5:

(1) For Stratix GX, this phase compensation FIFO is outside the transceiver. For Stratix II GX and Stratix IV, you must set the zero (0) PPM clock group settings to use the internal phase compensation FIFOs within the transceiver block.

(2) Individual recovered clocks (one per channel).

(3) For Stratix GX, `tx_coreclk` is tied directly to `trefclk`. (It does not use the output of `TXPLL`.)

Figure 4–6. No Receiver FIFO Buffers Clock Structure



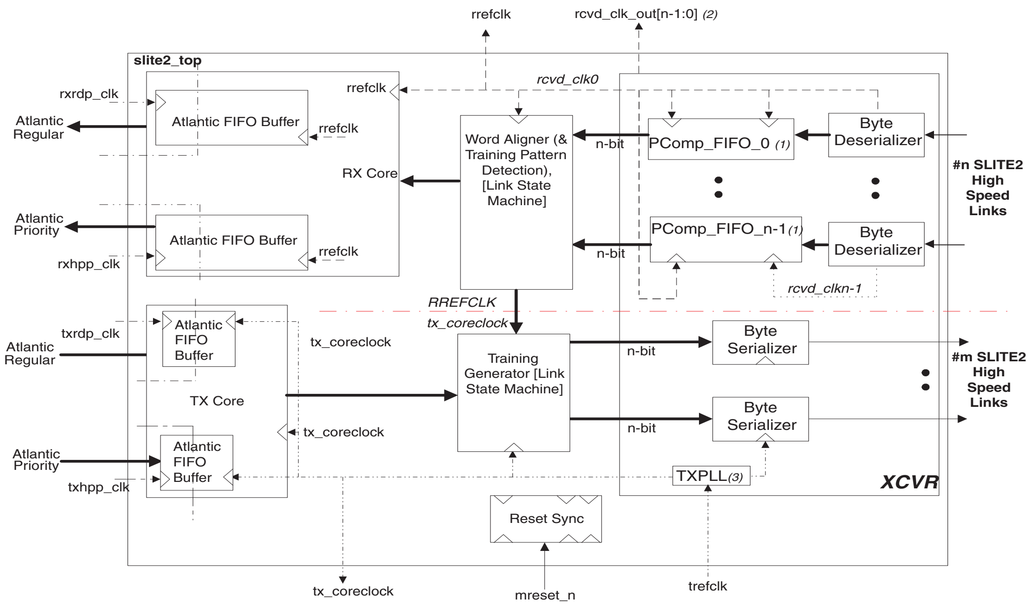
Notes to Figure 4–6:

(1) For Stratix GX, this phase compensation FIFO is outside the transceiver. For Stratix II GX and Stratix IV, you must set the zero (0) PPM clock group settings to use the internal phase compensation FIFOs within the transceiver block.

(2) Individual recovered clocks (one per channel).

(3) For Stratix GX, `tx_coreclock` is tied directly to `trefclk`. (It does not use the output of `TXPLL`.)

Figure 4–7. Full-Featured No Frequency Offset Clock Structure



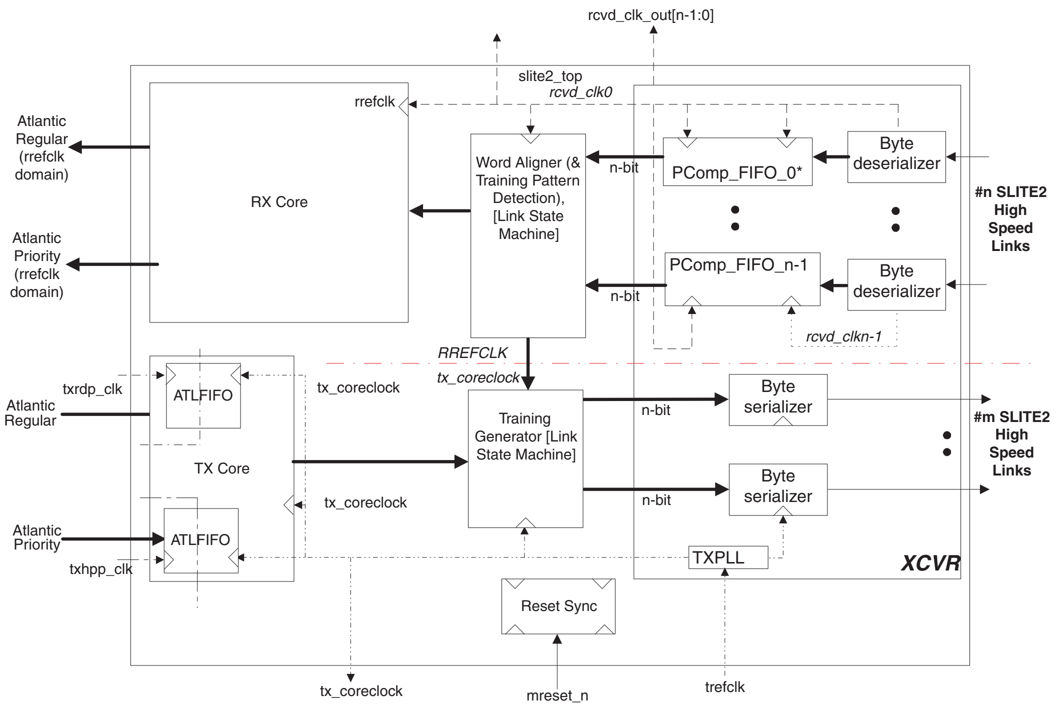
Notes to Figure 4–7:

(1) For Stratix II GX and configurations with more than one lane, this Phase Compensation FIFO is outside the transceiver. For Stratix II GX and Stratix IV, you must set the zero (0) PPM clock group settings to use the internal phase compensation FIFOs within the transceiver block.

(2) Individual recovered clock (one per channel).

(3) For Stratix GX, `tx_coreclock` is directly tied to `trefclk` (does not use output of `TXPLL`).

Figure 4–8. No Receiver FIFO Buffers No Frequency Offset Clock Structure



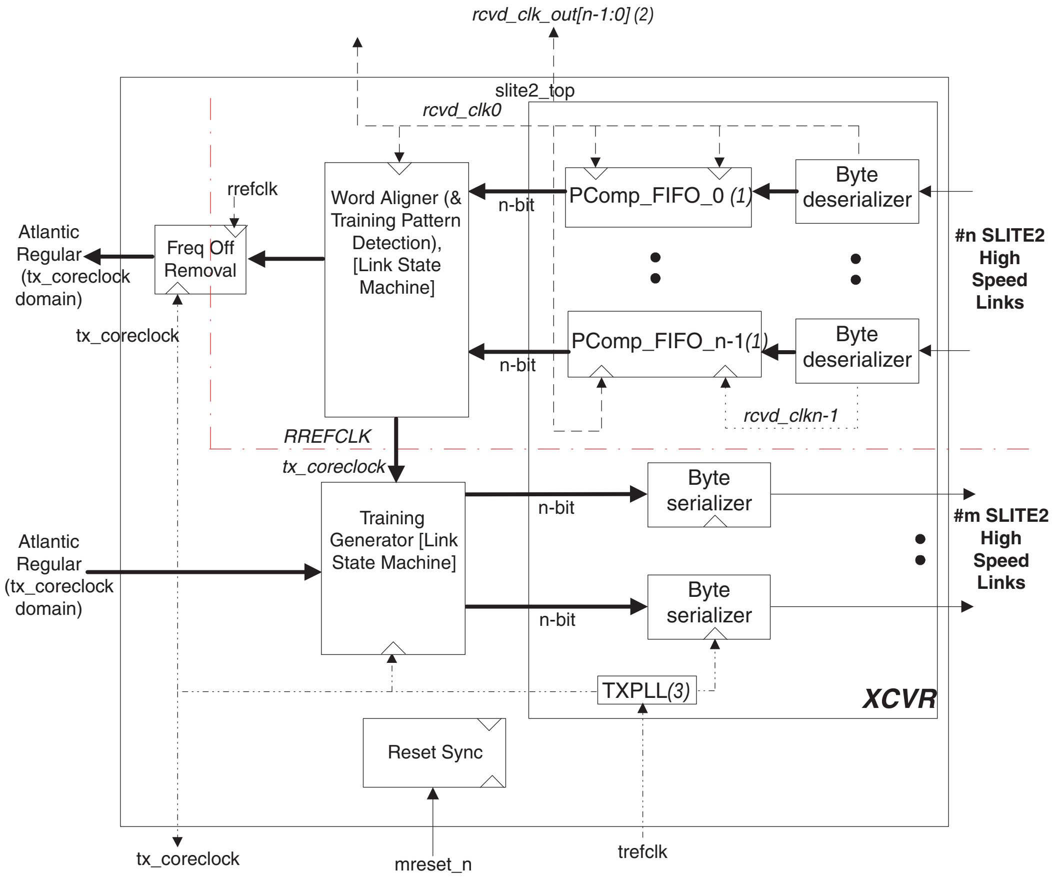
Notes to Figure 4–8:

(1) For Stratix GX, this phase compensation FIFO is outside the transceiver. For Stratix II GX and Stratix IV, you must set the zero (0) PPM clock group settings to use the internal phase compensation FIFOs within the transceiver block.

(2) Individual recovered clocks (one per channel).

(3) For Stratix GX, `tx_coreclock` is directly tied to `trefclk`. (It does not use the output of `TXPLL`.)

Figure 4–9. Streaming Full-Featured Clock Structure



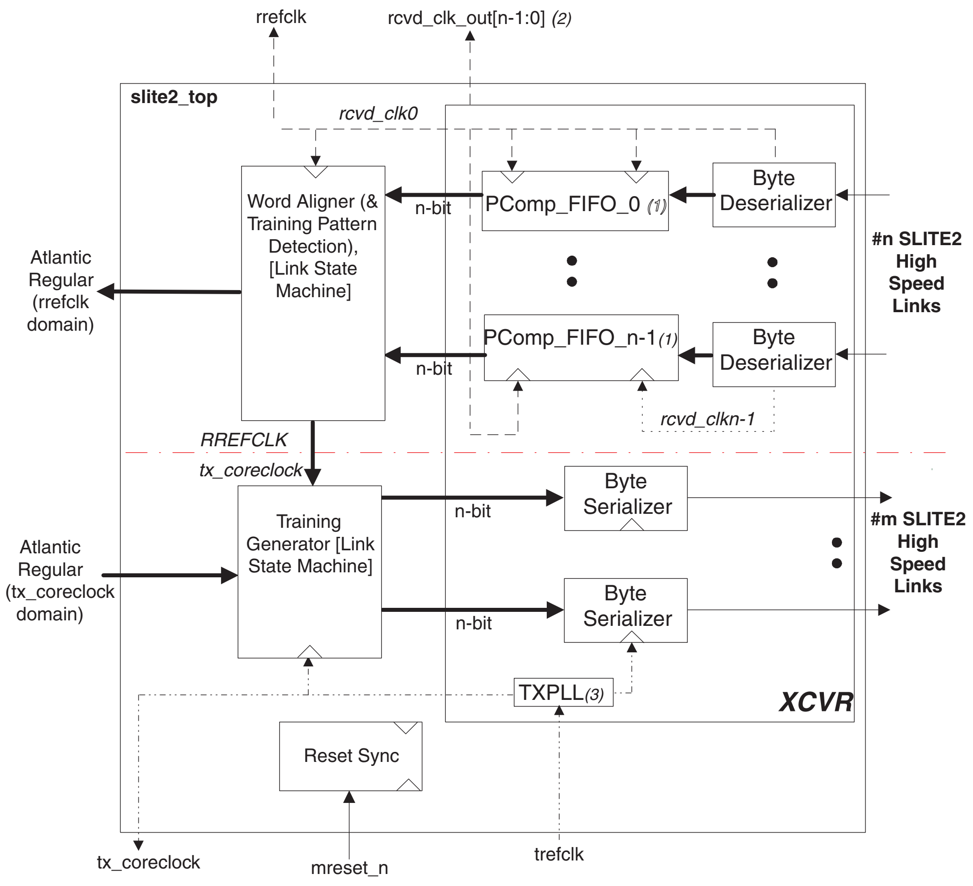
Notes for Figure 4–9:

(1) For Stratix GX, the phase compensation FIFO is outside the transceiver. For Stratix II GX and Stratix IV, you may need to set the zero (0) PPM clock group settings to use the internal phase compensation FIFOs within the transceiver block.

(2) Individual recovered clocks (one per channel).

(3) For Stratix GX, `tx_coreclk` directly tied to `trefclk`. (It does not use the output of `TXPLL`.)

Figure 4–10. Streaming No Frequency Offset Clock Structure



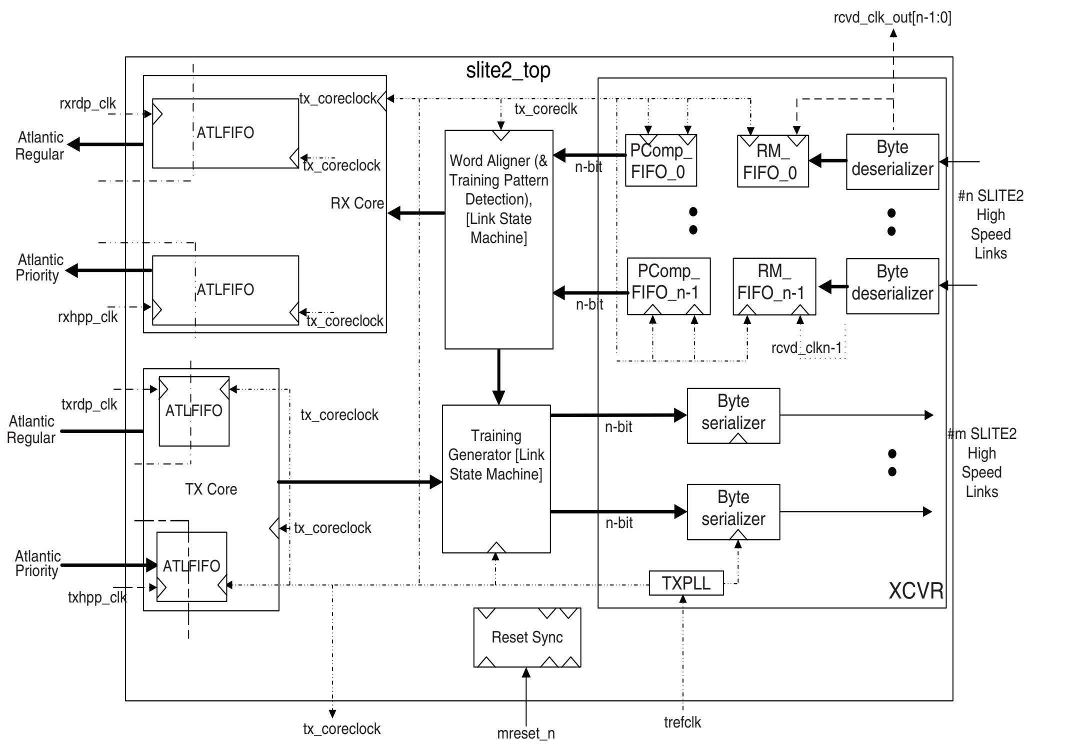
Notes to Figure 4–10:

(1) For Stratix GX and configurations with more than one lane, this Phase Compensation FIFO is outside the transceiver. You must set the zero (0) PPM clock group settings to use the internal phase compensation FIFOs within the transceiver block.

(2) Individual recovered clocks (one per channel).

(3) For Stratix GX, `tx_coreclock` is directly tied to `trefclk` (It does not use output of `TXPLL`).

Figure 4–11. Full Featured Clock Structure for 5G Symmetrical With TSIZE = 2



SerialLite II MegaCore Pin-Out Diagrams

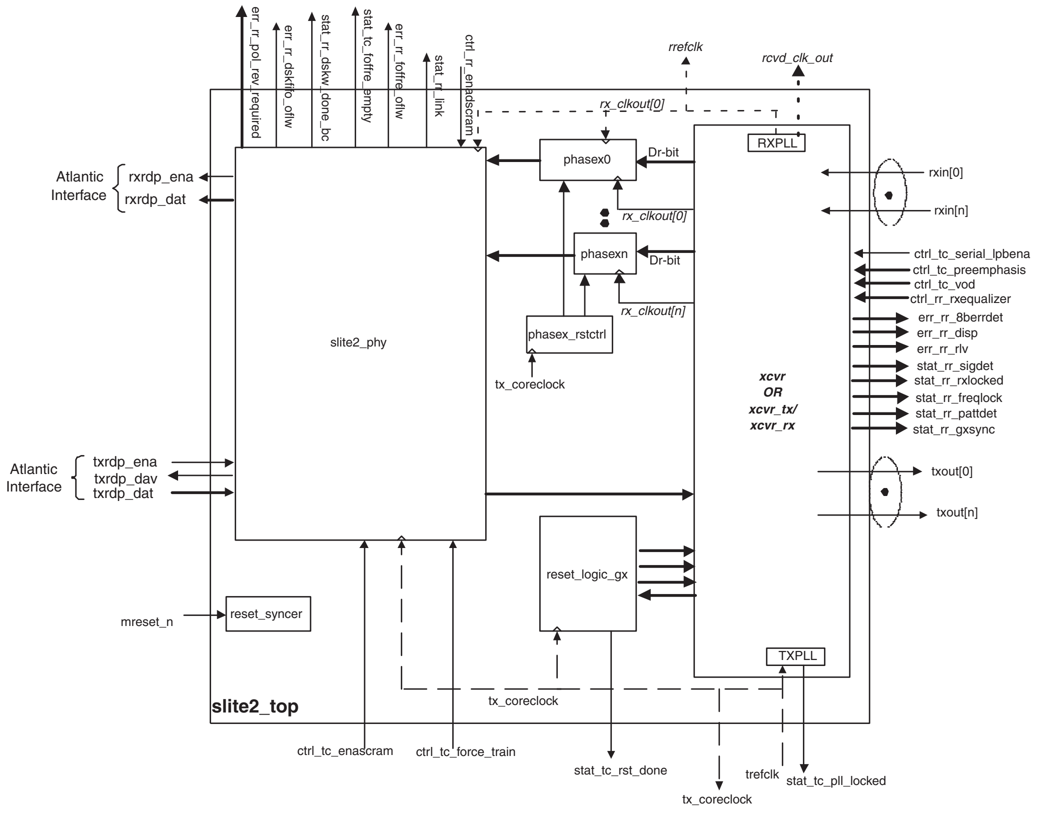
This section shows pin-out diagrams for the SerialLite II MegaCore function. The following diagrams are included:

- Stratix GX PHY Layer
- Stratix II GX/Stratix IV PHY Layer
- Transmitter Link Layer
- Receiver Layer With No FIFO
- Receiver Link Layer With FIFO

Your SerialLite II MegaCore function design always contains a PHY layer, based on the device you select. The link layer portions is present if the **Data Type** option is set to **Packets**. The inclusion of receiver and transmitter components is determined by the **Port Type** option that you select (**Bidirectional**, **Transmitter only**, **Receiver only**). For example, if **Data Type** is **Packets**; **Port Type** is **Bidirectional**; the receiver FIFO is set to **0** bytes; and the device family is **Stratix II GX**, refer to the following diagrams:

- Stratix II GX/Stratix IV PHY Layer
- Transmitter Link Layer
- Receiver Layer With No FIFO

Figure 4–12. Stratix GX PHY Layer



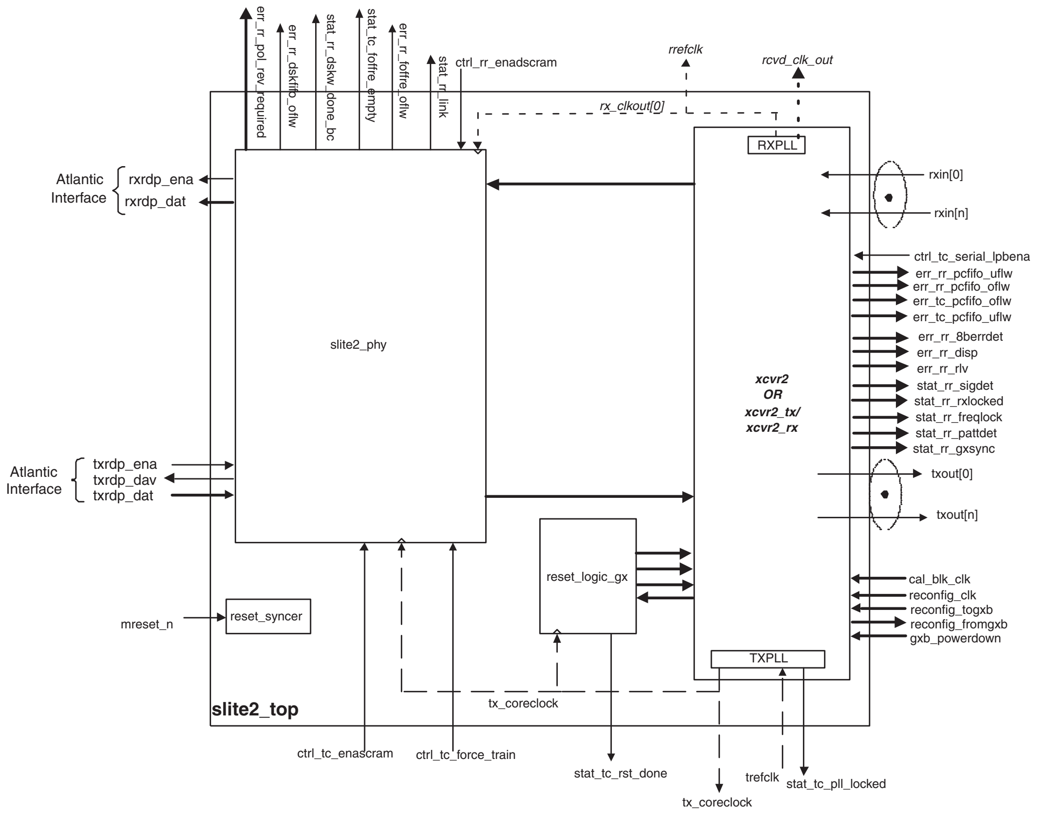
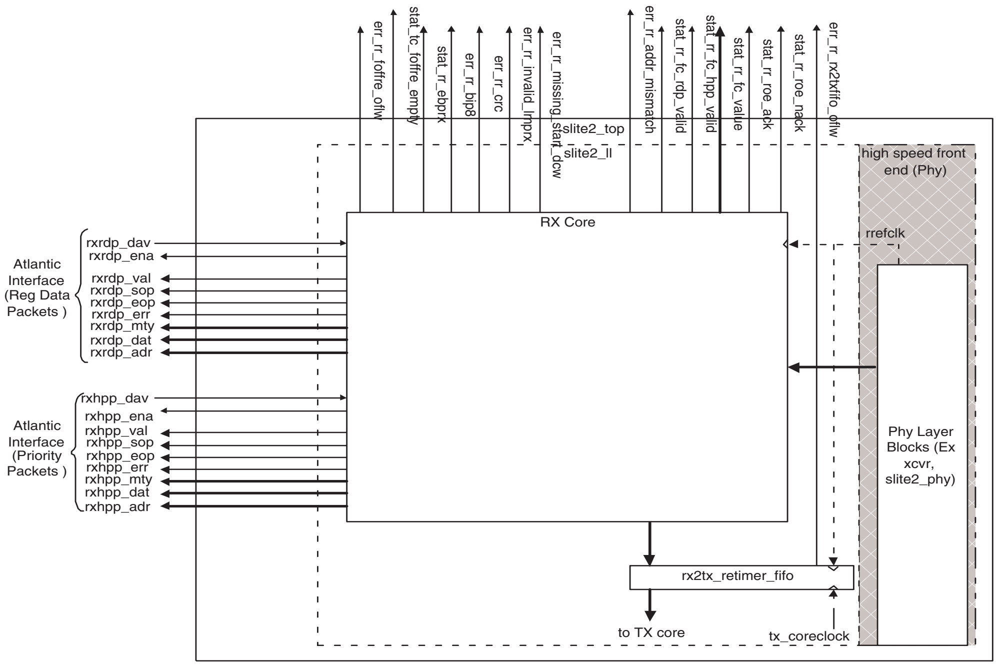


Figure 4–13. Stratix II GX/Stratix IV PHY Layer

Figure 4–14. Receiver Layer With No FIFO



Note to Figure 4–14:

(1) Signals are present if flow control is enabled. Drive the signal high to indicate that a flow control Link Management Packet is requested.

Figure 4–15. Receiver Link Layer With FIFO

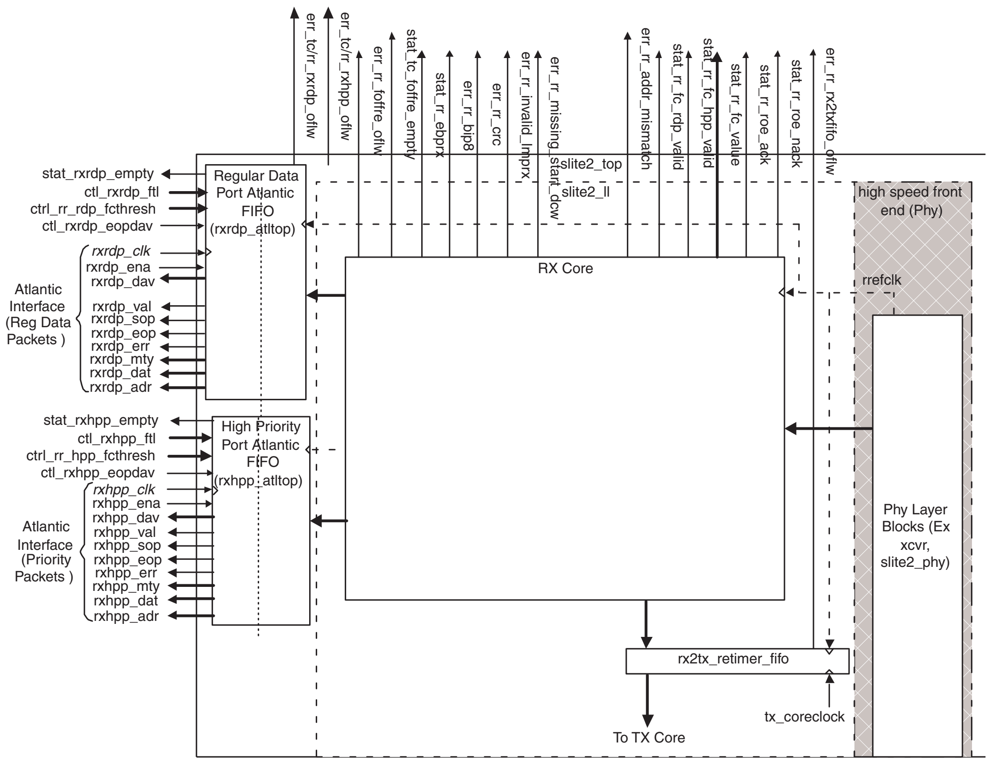
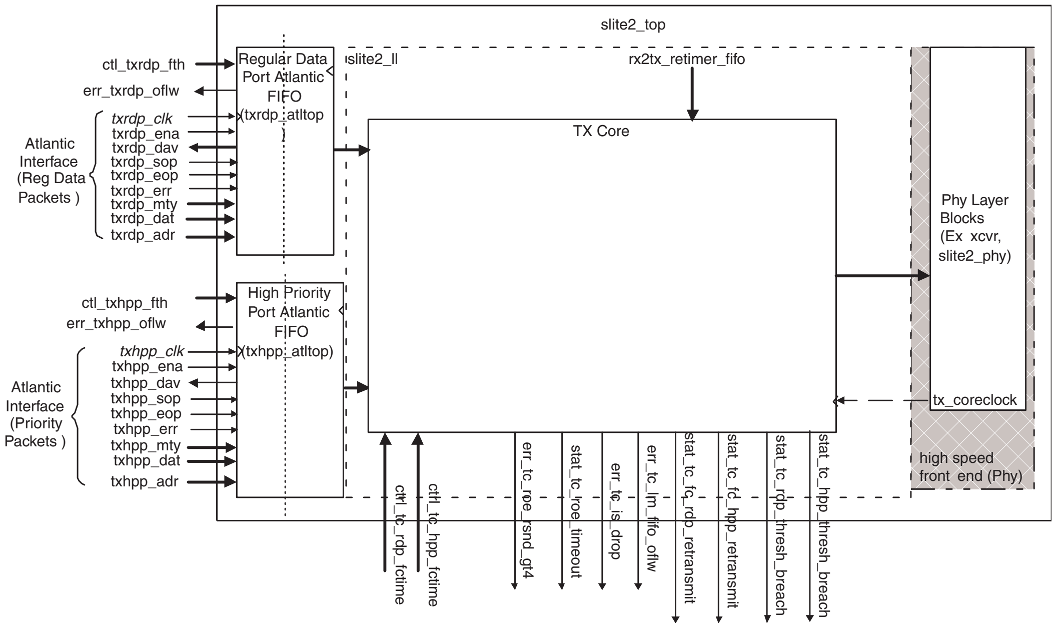


Figure 4–16. Transmitter Link Layer



Initialization and Restart

Before the SerialLite II link can operate, the MegaCore function must properly reset the GX transceiver. The SerialLite II MegaCore function must then be initialized and trained. The SerialLite II training sequence can 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. The reset of the GX transceiver is controlled by the `mreset_n` and `gxb_powerdown` signals. The minimum pulse width is determined by characterization. Currently, a 2 ms pulse width is sufficient for the `gxb_powerdown` input, and three cycles for the `mreset_n` signal. For simulation, a reset duration of several clock cycles (for example, 10) is sufficient.

A link only restarts on its own if a link error occurs during normal operation. A hardware reset using the `mreset_n` signal also brings down the link when the reset is asserted low and re-establishes the link when the reset is released. 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 after it sees eight successive training sequences.

Multiple Core Configuration

When you instantiate multiple SerialLite II MegaCore functions, you must apply the following additional guidelines to create a working design.

- If you use the Tcl constraints to make assignments for the MegaCore functions, you must edit the Tcl script associated with each generated SerialLite II MegaCore function to update the hierarchal paths to each clock node and signal inside the TCL scripts. You can use the generated scripts as a guide. You must also make these changes to the generated Synopsys Design Constraints File (**.sdc)** if you intend to use the TimeQuest Timing Analyzer.

  Note that the Tcl scripts assume a top-level name for several clocks, such as: `trefclk`, `rxrdp_clk`, `rxhpp_clk`, `txrdp_clk`, and `txhpp_clk`. You must edit `Set Clock Names` in the scripts if the clock name connected to these inputs does not match. If the multiple cores are connected to the same clocks at the top-level file, you must make sure `Set Clock Names` and `clock settings` are only available in one script. You must always set to run this script first in the projects. You must edit the Tcl script and the **.sdc** file if you plan to use the Timequest timing analyzer.

- For Arria GX, Stratix II GX and Stratix IV designs, you must ensure that the `cal_blk_clk` input to each SerialLite II MegaCore function is driven by the same calibration clock source. In addition, ensure that the SerialLite II MegaCore function and other MegaCore variants in the system that use the altgx megafunction have the same clock source connected to their respective `cal_blk_clk` ports.

- In Arria GX, Stratix II GX, and Stratix IV designs that include multiple SerialLite II cores in a single transceiver block, the same signal must drive `gxb_powerdown` to each of the SerialLite II MegaCore variants.

Signals

Tables 4–2 through 4–7 show the SerialLite II MegaCore function signals.

☞ The signals required for a given configuration, as well as the appropriate bus widths, are created automatically by the MegaWizard interface based upon the parameter values you select.

Table 4–2 shows the high-speed serial interface signals.

Table 4–2. High-Speed Serial Interface Signals

Signal	Direction	Clock Domain	Description
`rxin[n-1]` *(1)*	Output	–	SerialLite II differential receive data bus. Bus carries packets, cells, or in-band control words.
`txout[m-1]` *(2)*	Output	–	SerialLite II differential transmit data bus. Bus carries packets, cells, or in-band control words.
`rrefclk` *(3)*	Output	`rrefclk`	Receive core output PLL-derived clock.
`trefclk`	Input	`trefclk`	Reference clock used to drive the transmitter PLL. The PLL is used to generate the transmit core clock (`tx_coreclock`).
`tx_coreclock`	Output	`tx_coreclock`	Transmitter core output clock. Directly connected to `trefclk` in Stratix GX designs. In Arria GX, Stratix II GX, and Stratix IV designs, the TX PLL output clock and the primary clock are used for the TX logic.
`mreset_n`	Input	Asynchronous	Master reset pin, active low. Asserting this signal causes the entire SerialLite II MegaCore function, including the Atlantic FIFO buffers, to be reset.
`ctrl_tc_force_train`	Input	`tx_coreclock`	Force training patterns to be sent. Negate once the receiver has locked. Only used in self-synchronizing mode. Otherwise, this signal is currently reserved (tie this signal to `1'b0`).
`stat_tc_pll_locked`	Output	`tx_coreclock`	PLL locked signal. Indicates that the `ALTGXB` PLL has locked to the `trefclk`.
`stat_rr_link` *(3)*	Output	`rrefclk`	Link Status. When high, the link is enabled.

Notes to Table 4–2:
(1) n = RX number of lanes
(2) m = TX Number of lanes
(3) In broadcast mode, these signals will have the corresponding receiver function number post-fixed. For example, `err_rr_crc0` is the CRC error signal from SerialLite II receiver block 0.

Table 4–3 shows the transceiver megafunction signals.

For more information on Altera gigabit transceiver (altgxb) megafunction, refer to the *Stratix GX Transceiver User Guide* in volume 2 of the *Stratix GX Device Handbook*.

For more information on Altera gigabit transceiver (altgx) megafunction, refer to the *Stratix II GX Transceiver User Guide* in volume 2 of the *Stratix II GX Device Handbook* and the *Stratix IV Transceiver Architecture* in volume 2 of the *Stratix IV Device Handbook*.

Table 4–3. altgxb and alt2gxb Megafunction Signals (Part 1 of 3)

Signal *(1)*, *(2)*	Direction	Clock Domain	Description
`ctrl_tc_serial_lpbena`	Input	`tx_coreclock`	Serial Loopback (`TXOUT` internally connected to `RXIN`). Tie signal to `1'b0` to NOT use loopback, tie to `1'b1` to Use Serial Loopback.
`rcvd_clk_out [`*`rxnl-1:0`*`]`	Output		Per lane recovered clock.
`err_rr_8berrdet [`*`srx-1:0`*`]`	Output	`rrefclk`	8B/10B error detection signal.
`err_rr_disp[`*`srx-1:0`*`]`	Output	`rrefclk`	Disparity error detection signal
`err_rr_pcfifo_uflw [`*`rxnl-1:0`*`]`	Output	`rrefclk`	Interface/phase compensation FIFO buffer underflow signal (Arria GX, Stratix II GX, and Stratix IV GX devices only).
`err_rr_pcfifo_oflw [`*`rxnl-1:0`*`]`	Output	`rrefclk`	Interface/phase compensation FIFO buffer overflow signal (Arria GX, Stratix II GX, and Stratix IV GX devices only).
`err_rr_rlv[`*`rxnl-1:0`*`]`	Output	`rrefclk`	Run length violation status signal.
`err_tc_pcfifo_uflw [`*`txnl-1:0`*`]`	Output	`tx_coreclock`	Interface/phase compensation FIFO buffer underflow signal (Arria GX, Stratix II GX, and Stratix IV GX devices only).
`err_tc_pcfifo_oflw [`*`txnl-1:0`*`]`	Output	`tx_coreclock`	Interface/phase compensation FIFO buffer overflow signal (Arria GX, Stratix II GX and Stratix IV GX devices only).
`stat_rr_sigdet [`*`rxnl`*`-1:0]`	Output	`rrefclk`	This signal is for debugging purposes only and can be ignored.
`stat_rr_gxsync [`*`srx`*`-1:0]`	Output	`rrefclk`	Gives the status of the pattern detector and word aligner.

Table 4–3. altgxb and alt2gxb Megafunction Signals (Part 2 of 3)

Signal (1), (2)	Direction	Clock Domain	Description
`stat_rr_rxlocked [`*`rxnl`*`-1:0]`	Output	`rrefclk`	Receiver PLL locked signal. Indicates whether or not the receiver PLL is phase locked to the CRU reference clock. When the PLL locks to data, which happens some time after the transceiver's `rx_freqlocked` signal is asserted high, this signal has little meaning because it only indicates lock to the reference clock. This signal is active low for Stratix GX devices and active high for Arria GX, Stratix II GX, and Stratix IV GX devices.
`stat_rr_freqlock [`*`rxnl`*`-1:0]`	Output	`rrefclk`	Frequency locked signal from the CRU. Indicates whether the transceiver block receiver channel is locked to the data mode in the `rxin` port.
`stat_rr_pattdet [`*`srx`*`-1:0]`	Output	`rrefclk`	Pattern detection signal
`reconfig_fromgxb [`*`recon_quad`*` - 1:0]` (3)	Output	`reconfig_clk` (4)	alt2gxb Reconfig from the GXB Bus. If the `alt2gxb_reconfig` block is not used, do not connect this output. If the alt2gxb_reconfig block is used, connect this signal to the `reconfig_fromgxb` port on the alt2gxb_reconfig module.
`reconfig_togxb [2:0]`	Input	`reconifg_clk`	alt2gxb Reconfig to the GXB Bus. If the alt2gxb_reconfig block is not used, connect this bus to `3'b010`. If the alt2gxb_reconfig block is used, connect this signal to the `reconfig_togxb` port on the alt2gxb_reconfig module.
`reconfig_clk`	Input	—	alt2gxb Reconfig Clock to the GXB. If the alt2gxb_reconfig block is not used, connect this clock to `1'b0`. If the alt2gxb_reconfig block is used, connect this signal to the `reconfig_clk` port on the alt2gxb_reconfig module.
`cal_blk_clk`	Input	—	Calibration clock for the termination resistor calibration block. The frequency range of `cal_blk_clk` is 10 to 125 MHz. The quality of the calibration clock is not an issue, so PLD local routing is sufficient to route the calibration clock.

Table 4–3. altgxb and alt2gxb Megafunction Signals (Part 3 of 3)

Signal *(1)*, *(2)*	Direction	Clock Domain	Description
`gxb_powerdown`	Input	—	Transceiver block reset and power down. This signal resets and powers down all circuits in the transceiver block. This does not affect the refclk buffers and reference clock lines. All the `gxb_powerdown` input signals of cores placed in the same transceiver block should be tied together.The `gxb_powerdown` signal should be tied low or should remain asserted for at least 2ms whenever it is asserted.

Notes to Table 4–3:

(1) rxnl is the receive number of lanes; txnl is the transmit number of lanes.

(2) srx is the transfer size × the receive number of lanes.

(3) recon_quad is the total number of Quads being used.

(4) If the `alt2gxb_reconfig` block is not used, the signal will not toggle (set to a fixed value) and thus is not on any clock domain. If the `alt2gxb_reconfig` block is used, this signal is on the `reconfig_clk` domain.

Table 4–4 shows the Atlantic interface signals.

For more information on this interface, refer to the *FS13: Atlantic Interface*.

These signals are only present when the Link Layer mode is enabled and the Atlantic FIFO buffer is used.

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

Signal	Direction	Clock Domain	Description
`rxrdp_clk` *(1)*	Input	–	Atlantic receive regular data port clock.
`txrdp_clk`	Input	–	Atlantic transmit regular data port clock.
`rxhpp_clk` *(1)*	Input	–	Atlantic receive high priority port clock.
`txhpp_clk`	Input	–	Atlantic transmit high priority port clock.
`rxrdp_ena` *(1)*	Input	`rxrdp_clk`	Enable signal on the Atlantic interface. Indicates that the data is to be read on the next clock cycle.
`rxrdp_dav` *(1)*	Input	`rxrdp_clk`	Input (No FIFO buffer) determines whether flow control is required on this port. When this signal is low, the fill level has been breached. When this signal is high, the FIFO buffer has enough space for more words.

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

Signal	Direction	Clock Domain	Description
`rxrdp_dav` *(1)*	Output	`rxrdp_clk`	Output (With FIFO buffer) represents the buffer's fill level. This signal is high when the level is above FTL or if an EOP is in the buffer.
`rxrdp_val` *(1)*	Output	`rxrdp_clk`	The output data is valid.
`rxrdp_sop` *(1)*	Output	`rxrdp_clk`	Start of packet indicator on the Atlantic interface.
`rxrdp_eop` *(1)*	Output	`rxrdp_clk`	End of packet indicator on the Atlantic interface.
`rxrdp_err` *(1)*	Output	`rxrdp_clk`	Error indicator on the Atlantic Interface. This signal is not necessarily held high until `rxrdp_eop` is asserted.
`rxrdp_mty[`*m*`–1:0]` *(1)*, *(3)*	Output	`rxrdp_clk`	Number of empty bytes in the data word.
`rxrdp_dat[`*d*`–1:0]` *(1)*, *(2)*	Output	`rxrdp_clk`	User data bits.
`rxrdp_adr[7:0]` *(1)*	Output	`rxrdp_clk`	User-defined packet ID. Only valid with `rxrdp_sop`.
`txrdp_ena`	Input	`txrdp_clk`	Enable signal on the Atlantic interface. Indicates that the data is valid.
`txrdp_dav`	Output	`txrdp_clk`	Indicates that the input FIFO buffer is not full.
`txrdp_sop`	Input	`txrdp_clk`	Start of packet indicator on the Atlantic interface.
`txrdp_eop`	Input	`txrdp_clk`	End of packet indicator on the Atlantic interface.
`txrdp_err`	Input	`txrdp_clk`	Error indicator on the Atlantic interface.
`txrdp_mty[`*tm*`–1:0]` *(4)*	Input	`txrdp_clk`	Number of empty bytes in the data word.
`txrdp_dat[`*td*`–1:0]` *(5)*	Input	`txrdp_clk`	User data bits.
`txrdp_adr[7:0]`	Input	`txrdp_clk`	User-defined packet ID.
`rxhpp_ena` *(1)*	Input	`rxhpp_clk`	Enable signal on the Atlantic interface. Indicates that the data is to be read on the next clock cycle.
`rxhpp_dav` *(1)*	Input	`rxhpp_clk`	Input (No FIFO buffer) determines whether flow control is required on this port.When this signal is low, the fill level has been breached. When this signal is high, the FIFO buffer has enough space for more words.
`rxhpp_dav` *(1)*	Output	`rxhpp_clk`	Output (With FIFO buffer) represents the buffer's fill level. This signal is high when the level is above FTL or if an EOP is in the buffer.
`rxhpp_val` *(1)*	Output	`rxhpp_clk`	The output data is valid.

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

Signal	Direction	Clock Domain	Description
`rxhpp_sop` *(1)*	Output	`rxhpp_clk`	Start of packet indicator on the Atlantic interface.
`rxhpp_eop` *(1)*	Output	`rxhpp_clk`	End of packet indicator on the Atlantic interface.
`rxhpp_err` *(1)*	Output	`rxhpp_clk`	Error indicator on the Atlantic Interface. This signal is not necessarily held high until `rxhpp_eop` is asserted.
`rxhpp_mty[m-1:0]` *(1)*, *(3)*	Output	`rxhpp_clk`	Number of empty bytes in the data word.
`rxhpp_dat[d-1:0]` *(1)*, *(2)*	Output	`rxhpp_clk`	User data bits.
`rxhpp_adr[3:0]` *(1)*	Output	`rxhpp_clk`	User-defined packet ID. Only valid with `rxhpp_sop`.
`txhpp_ena`	Input	`txhpp_clk`	Enable signal on the Atlantic interface. Indicates that the data is valid.
`txhpp_dav`	Output	`txhpp_clk`	Indicates that the input FIFO buffer is not full.
`txhpp_sop`	Input	`txhpp_clk`	Start of packet indicator on the Atlantic interface.
`txhpp_eop`	Input	`txhpp_clk`	End of packet indicator on the Atlantic interface.
`txhpp_err`	Input	`txhpp_clk`	Error indicator on the Atlantic interface.
`txhpp_mty[tm-1:0]` *(4)*	Input	`txhpp_clk`	Number of empty bytes in the data word.
`txhpp_dat[td-1:0]` *(5)*	Input	`txhpp_clk`	User data bits.
`txhpp_adr[3:0]`	Input	`txhpp_clk`	User-defined packet ID.

Notes to Table 4–4:

(1) In broadcast mode, these pins will have the corresponding receiver function number post-fixed. For example, `err_rr_crc0` is the CRC error signal from SerialLite II receiver block 0.
(2) d is the data width, which is 8 × transfer size × the RX number of lanes.
(3) m is the empty value, which is log2 (data width).
(4) tm is the empty value, which is log2 (data width).
(5) td is the data width, which is 8 × transfer size × the TX number of lanes.

Table 4–5 shows the Atlantic interface signals for streaming mode.

Table 4–5. Atlantic Interface Signals for Streaming Mode

Signal	Direction	Clock Domain	Description
rxrdp_dat [d-1:0] (1), (2)	Output	rrefclk	Received user data bits.
rxrdp_ena (1)	Output	rrefclk	Enable signal on the Atlantic interface. Indicates that thed data is valid on the current clock cycle.
txrdp_dat [td-1:0] (3)	Input	tx_coreclock	User data bits to be transmitted.
txrdp_ena	Input	tx_coreclock	Enable signal on the Atlantic interface. Indicates that the data is valid.
txrdp_dav	Output	tx_coreclock	Indicates that the core is requesting the user data to stop while the core inserts the clock compensation sequence. If **Clock Compensation** is not enabled, this signal will always be high while the link is up.

Table 4–6 shows the protocol processor's error, status, and control signals.

Table 4–6. Protocol Processor's Error, Status and Control Signals (Part 1 of 3)

Signal	Direction	Clock Domain	Description
`err_rr_rxrdp_oflw`	Output	`rrefclk`	Indicates that the Atlantic FIFO buffer has overflowed and data has been lost when **Clock Compensation** is disabled (regular data port).
`err_rr_rxhpp_oflw`	Output	`rrefclk`	Indicates that the Atlantic FIFO buffer has overflowed and data has been lost when **Clock Compensation** is disabled (priority data port).
`err_tc_rxrdp_oflw`	Output	`tx_coreclock`	Indicates that the Atlantic FIFO buffer has overflowed and data has been lost when **Clock compensation** is enabled (regular data port).

Table 4–6. Protocol Processor's Error, Status and Control Signals (Part 2 of 3)

Signal	Direction	Clock Domain	Description
`err_tc_rxhpp_oflw`	Output	`tx_coreclock`	Indicates that the Atlantic FIFO buffer has overflowed and data has been lost when the **Clock Compensation** is enabled (priority data port).
`err_txrdp_oflw`	Output	`txrdp_clk`	Indicates that the Atlantic FIFO buffer has overflowed and data has been lost (regular data port).
`err_txhpp_oflw`	Output	`txhpp_clk`	Indicates that the high-priority Atlantic FIFO buffer has overflowed and data has been lost. If the **Retry-on-error** parameter is turned on, this signal remains high until the FIFO buffer has been emptied by the SerialLite II MegaCore function.
`stat_rxrdp_empty` *(1)*	Output	`rxdrp_clk`	Indicates that the internal Atlantic FIFO buffer is empty, and the read request is ignored.
`stat_rxhpp_empty` *(1)*	Output	`rxhpp_clk`	Indicates that the internal Atlantic FIFO buffer is empty, and the read request is ignored.
`ctl_rxhpp_ftl [n-1:0]` *(2)*	Input	`rxhpp_clk`	Receive high priority port FIFO threshold low (`dav` control). Determines when to inform the user logic that data is available via the `rxhpp_dav` signal. This threshold applies to all buffers. Units are in elements. Only change at reset.
`ctl_rxrdp_ftl [n-1:0]` *(2)*	Input	`rxrdp_clk`	Receive regular data port FIFO threshold low (`dav` control). Determines when to inform the user logic that space is available via the `rxrdp_dav` signal. This threshold applies to all buffers. Units are in elements. Only change at reset.
`ctl_rxhpp_eopdav` *(1)*	Input	`rxhpp_clk`	Receive high priority port FIFO buffer end-of-packet (EOP)-based `dav` control. Assert to turn on `dav` when there is an end of packet below the FTL threshold. Value applies to all Atlantic buffers. Only change at reset.
`ctl_rxrdp_eopdav` *(1)*	Input	`rxrdp_clk`	Receive regular data port FIFO buffer EOP-based `dav` control. Assert to turn on `dav` when there is an end of packet below the FTL threshold. Value applies to all Atlantic buffers. Only change at reset.
`ctl_txhpp_fth [tn-1:0]`*(3)*	Input	`txhpp_clk`	Transmit high priority port FIFO buffer threshold high `dav` control.

Table 4–6. Protocol Processor's Error, Status and Control Signals (Part 3 of 3)

Signal	Direction	Clock Domain	Description
`ctl_txrdp_fth [tn-1:0]` *(3)*	Input	`txrdp_clk`	Transmit regular data port FIFO buffer threshold high `dav` control.

Notes to Table 4–6:

(1) In broadcast mode, these signals will have the corresponding receiver function number post-fixed. For example, `err_rr_crc0` is the CRC error signal from SerialLite II receiver block 0.

(2) n is = FIFO SIZE / (TSIZE * RX Number of Lanes).

(3) tn is = FIFO SIZE / (TSIZE * TX Number of Lanes).

Table 4–7 shows the troubleshooting signals. These signals do not necessarily need to be connected to external logic. In general, they are for diagnostic purposes. Some signals in Table 4–7 are only available in certain configurations.

Table 4–7. Troubleshooting Signals (Part 1 of 3)

Signal	Direction	Clock Domain	Description
`stat_tc_rst_done`	Output	`tx_coreclock`	Reset controller logic `Done` signal. When high, the reset controller has completed the altgxb reset sequence successfully.
`err_rr_foffre_oflw` *(1)*	Output	`rrefclk`	Frequency offset tolerance FIFO buffer has overflowed. Link restarts.
`stat_tc_foffre_empty` *(1)*	Output	`tx_coreclock`	Frequency offset tolerance FIFO buffer has underflowed. The link does not go down. IDLE characters are inserted. This does not have a negative impact on the core, and is simply for diagnostic purposes.
`stat_rr_ebprx` *(1)*	Output	`rrefclk`	An end of bad packet character was received.
`err_rr_bip8` *(1)*	Output	`rrefclk`	A BIP-8 error was detected in the received link management packet.
`err_rr_crc` *(1)*	Output	`rrefclk`	A CRC error was detected in the received segment/packet.
`err_rr_fcrx_bne` *(1)*	Output	`rrefclk`	A flow control link management packet was received, but flow control is not enabled.
`err_rr_roerx_bne` *(1)*	Output	`rrefclk`	A retry-on-error link management packet was received, but **Retry-on-error** parameter is not enabled.

Table 4–7. Troubleshooting Signals (Part 2 of 3)

Signal	Direction	Clock Domain	Description
`err_rr_invalid_lmprx` *(1)*	Output	`rrefclk`	An invalid link management packet was received.
`err_rr_missing_start_dcw` *(1)*	Output	`rrefclk`	Data byte(s) received, but a start of data control word (DCW) is missing.
`err_addr_mismatch` *(1)*	Output	`rrefclk`	The start and end address fields do not match. Segments are marked with an error. Possible packets are destined for an invalid address.
`err_rr_pol_rev_required` *(1)*	Output	`rrefclk`	Catastrophic error. Polarity on the input altgxb lines is reversed. The MegaCore function cannot operate.
`err_rr_dskfifo_oflw` *(1)*	Output	`rrefclk`	Deskew FIFO buffer has overflowed. Link restarts.
`stat_rr_dskw_done_bc` *(1)*	Output	`rrefclk`	A bad column was received after successful deskew completion. Link is restarted.
`stat_tc_rdp_thresh_breach` *(1)*	Output	`tx_coreclock`	The receiver regular data port FIFO buffer is breached, transmit flow control link management packet.
`stat_tc_hpp_thresh_breach` *(1)*	Output	`tx_coreclock`	The receiver priority data port FIFO buffer is breached, transmit flow control link management packet.
`err_tc_roe_rsnd_gt4`	Output	`tx_coreclock`	Transmitter has transmitted a segment four times without receiving an ACK for that segment. Link is restarted.
`stat_tc_roe_timeout`	Output	`tx_coreclock`	**Retry-on-error** only: The transmitter MegaCore function has timed out waiting for ACK for a packet. The MegaCore function sends that packet again.
`stat_tc_fc_rdp_retransmit`	Output	`tx_coreclock`	The receiver FIFO buffer is still breached, and the refresh timer has reached maximum. Retransmitting flow control link management packet (regular data port).
`stat_tc_fc_hpp_retransmit`	Output	`tx_coreclock`	The receiver FIFO buffer is still breached, and the refresh timer has reached maximum. Retransmitting flow control link management packet (priority data port).
`err_tc_is_drop`	Output	`tx_coreclock`	Irregular segment received (**Segment size** boundary violation).

Table 4–7. Troubleshooting Signals (Part 3 of 3)

Signal	Direction	Clock Domain	Description
`err_tc_lm_fifo_oflw`	Output	`tx_coreclock`	Link management FIFO buffer has overflowed. Link management packets are lost.
`err_rr_rx2txfifo_oflw`	Output	`rrefclk`	Receiver to transmitter link management status information FIFO buffer has overflowed.
`stat_rr_fc_rdp_valid`	Output	`rrefclk`	A flow control link management packet was received (regular data port).
`stat_rr_fc_hpp_valid`	Output	`rrefclk`	A flow control link management packet was received (priority data port).
`stat_rr_fc_value[7:0]`	Output	`rrefclk`	The RAW `FC_TIME` value is embedded in the valid flow control link management packet. Decode with the `stat_rr_fc_rdp_valid` and `stat_rr_fc_hpp_valid` signals.
`stat_rr_roe_ack`	Output	`rrefclk`	A retry-on-error link management packet was received of type ACK.
`stat_rr_roe_nack`	Output	`rrefclk`	A retry-on-error link management packet was received of type NACK.

Note to Table 4–7:

(1) In broadcast mode, these signals will have the corresponding receiver function number post-fixed. For example, `err_rr_crc0` is the CRC error signal from SerialLite II receiver block 0.

MegaCore Verification

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

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

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

5. Testbench

General Description

This chapter describes the features and applications of the SerialLite II testbench to help you successfully design and verify your design implementation.

This demonstration testbench is available in Verilog HDL for all configurations and in VHDL for restricted configurations. The testbench shows you how to instantiate a model in a design, it stimulates the inputs and checks the outputs of the interfaces of the SerialLite II MegaCore® function, demonstrating basic functionality.

The demonstration testbench is generic and can be used with any Verilog HDL or VHDL simulator. The scripts allow you to run the testbench in the standard edition (SE) or the Altera® edition (AE) of the ModelSim® software.

Figure 5–1 on page 5–4 shows the block diagram of the SerialLite II testbench. The shaded blocks are provided with the SerialLite II testbench.

Features

The SerialLite II testbench includes the following features:

- Easy to use simulation environment for any standard Verilog HDL or VHDL simulator. For VHDL configurations where the VHDL demonstration testbench is not generated, a mixed language simulator is required to simulate the Verilog HDL testbench with the VHDL IP Functional Simulation models.
- Open source Verilog HDL or VHDL testbench files.
- Flexible SerialLite II functional model to verify your application that uses any SerialLite II MegaCore function.
- Simulates all basic SerialLite II transactions.

SerialLite II Testbench Files

The Verilog HDL demonstration testbench and associated scripts are generated when you create a MegaCore function variation in the MegaWizard Plug-In Manager, as described in "Generate Files" on page 2–7.

The VHDL demonstration testbench and the scripts to run it are generated when you create a MegaCore function variation that meets the following criteria:

- The language is VHDL.
- Broadcast mode is disabled.
- The data type is packets (streaming mode is disabled).
- Data packets are selected. (Priority packets are disabled.)
- The number of Rx lanes and Tx lanes is the same.
- The Rx buffer size is not equal to zero.

The SerialLite II testbench comprises the following files:

- Verilog HDL or VHDL top-level testbench file: <*variation_name*>**_tb.v** or <*variation_name*>**_tb.vhd**
- Verilog HDL or VHDL IP functional simulation model of the device under test (DUT): <*variation_name*>**.vo** or **.vho**
- Verilog HDL or VHDL IP functional simulation model of the SISTER MegaCore function used as a bus functional model for testing the DUT: <*variation_name*>**_sister_slite2_top.vo** or **.vho**

All utilities are included in the testbench file: <*variation_name*>**_tb.v** or <*variation_name*>**_tb.vhd**.

Testbench Specifications

This section describes the modules used by the SerialLite II testbench. See Figure 5–1 on page 5–4 for a block diagram of the SerialLite II testbench. The SerialLite II testbench has the following modules:

- Atlantic™ generators
- Device under test (DUT)
- Sister device
- Atlantic monitors
- Clock and reset generator
- Pin monitors

If your application requires a feature that is not supported by the SerialLite II testbench, you can modify the source code to add the feature. You can also modify the existing behavior to fit your application needs.

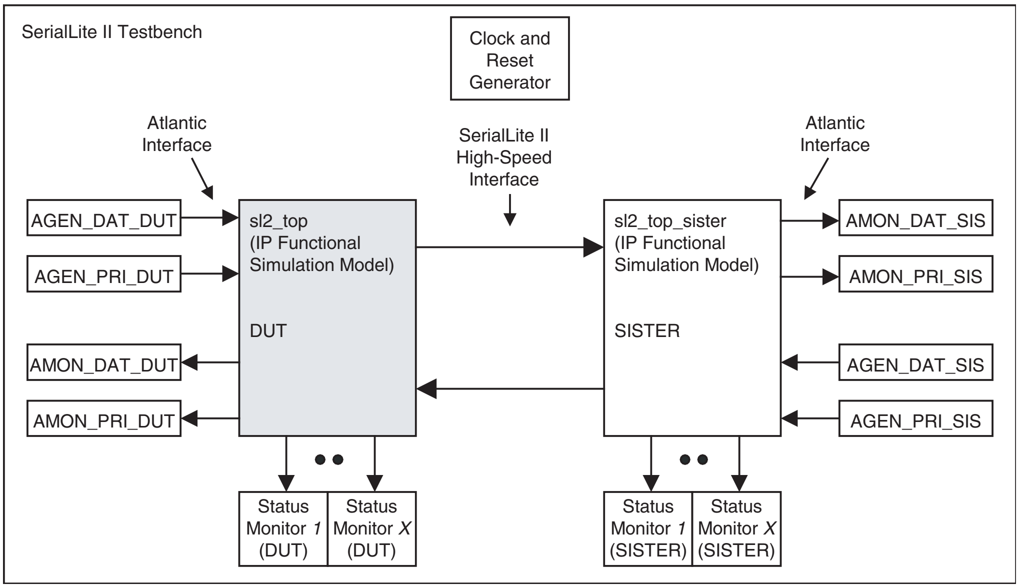
The testbench environment (`tb`) shown in Figure 5–1 on page 5–4 generates traffic through the Atlantic generators (`agen_dat_dut`, `agen_pri_dut`) and sends it through the SerialLite II MegaCore function— the device under test (DUT). The SerialLite II interface of the DUT is connected to the SerialLite II interface of a second SerialLite II MegaCore function—the SISTER. Data flows through the SISTER MegaCore function and is received and checked on the Atlantic interface of the SISTER MegaCore function (`amon_dat_sis`, `amon_pri_sis`). A similar data path exists in the opposite direction, where the SISTER's Atlantic generators (`agen_dat_sis`, `agen_pri_sis`) send data through the SerialLite II SISTER MegaCore function to the DUT, and data is received on the DUT's Atlantic interface (`amon_dat_dut`, `amon_pri_dut`).

Because there is no Atlantic to Atlantic verification, the received data's integrity is ensured in the following ways:

- Each Atlantic generator generates a certain number of packets or streaming bytes which the corresponding Atlantic monitor receives.
- The generated data follows a pseudo-random sequence (Verilog HDL) or incrementing data sequence (VHDL) that is checked by the Atlantic monitors.
- Each packet has an incrementing identifier (first byte in the packet) that is checked by the Atlantic monitor.

The SISTER MegaCore function is a SerialLite II MegaCore function with parameters derived from the DUT parameters. If the DUT is symmetrical (receiver's parameters matching transmitter's parameters), the SISTER's parameters match the DUT parameters. If the DUT is asymmetrical, the SISTER's parameters are different than the DUT's parameters, so that the DUT's transmitter parameters match the SISTER's receiver parameters and vice-versa. For a broadcast DUT, there are multiple SISTER instantiations. Pin monitor utilities monitor the SerialLite II status and error pins of the DUT and SISTER(s).

Figure 5–1. SerialLite II Testbench Environment (Non-Broadcast)



Notes to Figure 5–1:

(1) The DUT and the SISTER MegaCore functions may have different parameters; depending on the DUT parameters, and some components may be missing.

(2) _DAT = Regular Data Port; _PRI = High Priority Port; _DUT = Refers to DUT side; _SIS = Refers to SISTER side.

Depending on the SerialLite II link variation you choose (for example, using the single, broadcast, or asymmetric mode) the SerialLite II testbench environment may change, but the basic functionality is unchanged: data is sent or received on the Atlantic interface of the SerialLite II DUT IP model and received or sent on the Atlantic interface of the SerialLite II SISTER IP model.

Figure 5–2 shows the testbench environment for a SerialLite II single mode–transmitter only, non-broadcast MegaCore function. The SISTER model contains a receiver.

Figure 5–2. SerialLite II Testbench Environment (Single Mode–Transmitter Only, Verilog HDL Only, Non-Broadcast)

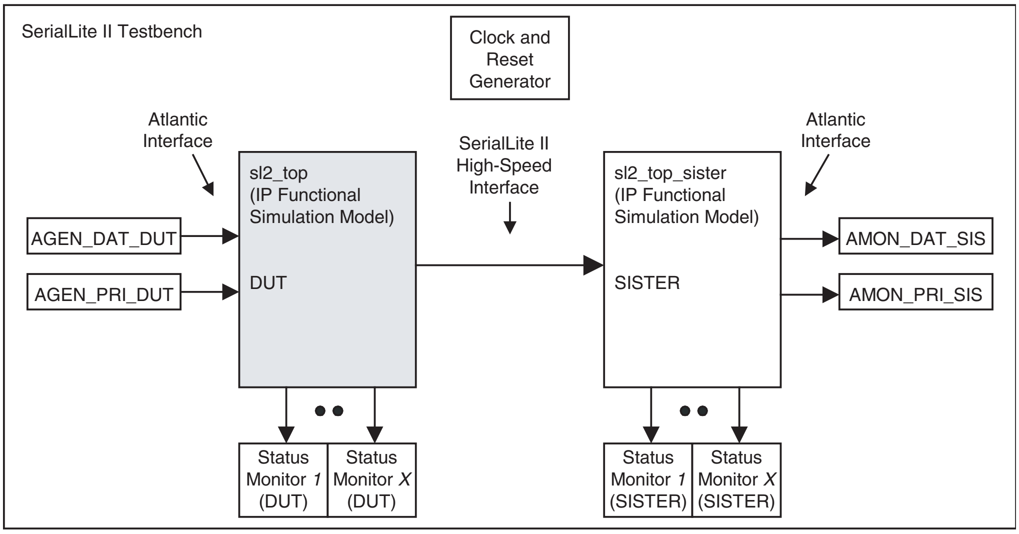

SerialLite II Testbench
Clock and Reset Generator
Atlantic Interface
SerialLite II High-Speed Interface
Atlantic Interface
sl2_top (IP Functional Simulation Model)
DUT
sl2_top_sister (IP Functional Simulation Model)
SISTER
AGEN_DAT_DUT
AGEN_PRI_DUT
AMON_DAT_SIS
AMON_PRI_SIS
Status Monitor 1 (DUT)
Status Monitor X (DUT)
Status Monitor 1 (SISTER)
Status Monitor X (SISTER)

Figure 5–3 on page 5–6 shows the testbench environment for a SerialLite II single mode–receiver only, non-broadcast MegaCore function. The SISTER model contains a transmitter.

Figure 5–3. SerialLite II Testbench Environment (Single Mode–Receiver Only, Verilog HDL Only, Non-Broadcast)

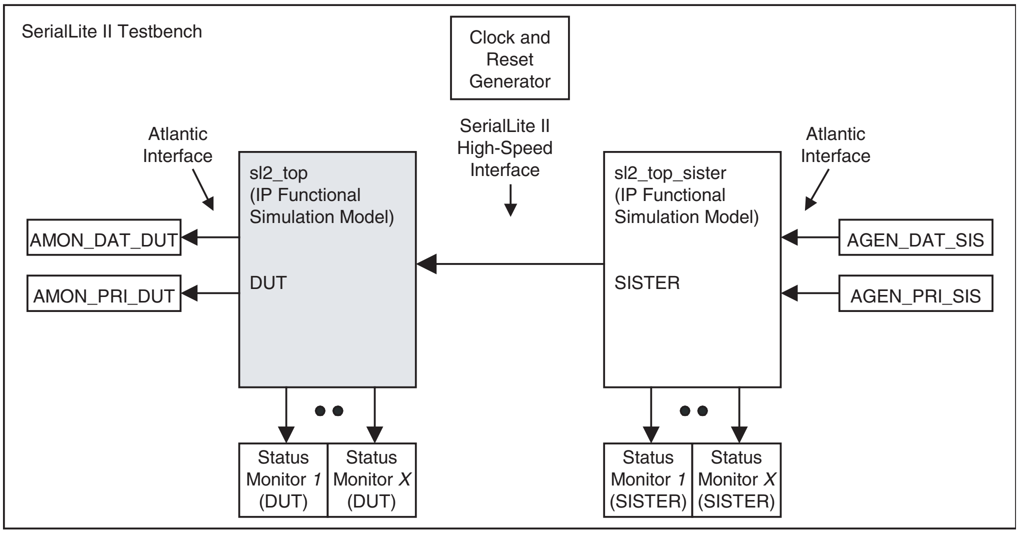
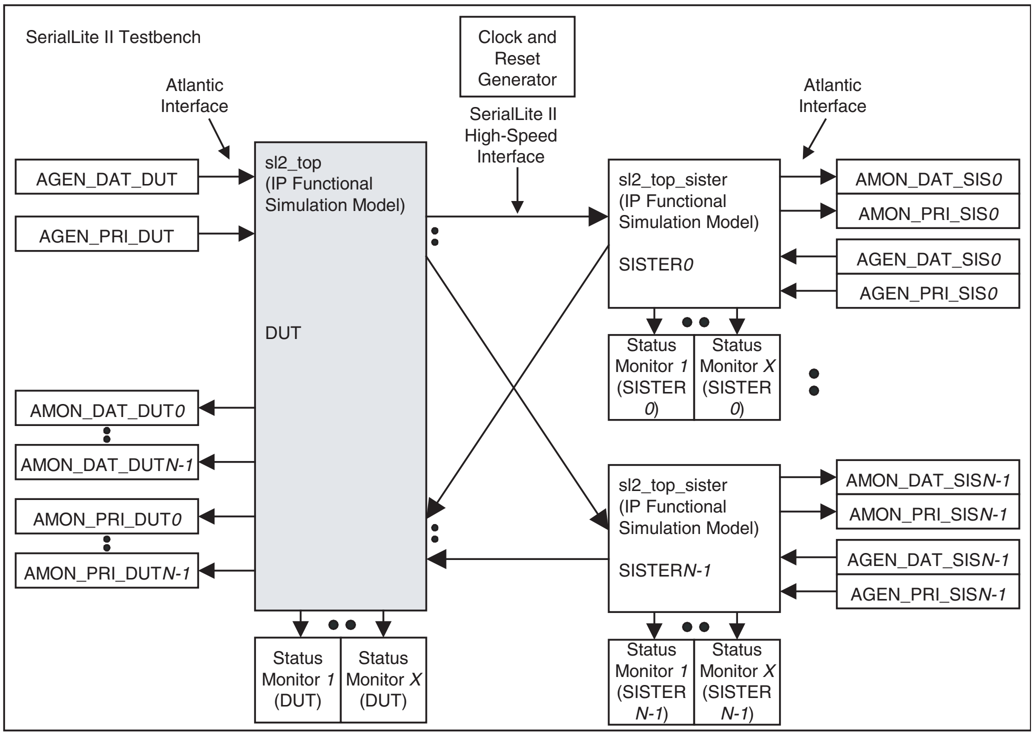

SerialLite II Testbench
Clock and Reset Generator
Atlantic Interface
SerialLite II High-Speed Interface
Atlantic Interface
sl2_top (IP Functional Simulation Model)
DUT
sl2_top_sister (IP Functional Simulation Model)
SISTER
AMON_DAT_DUT
AMON_PRI_DUT
AGEN_DAT_SIS
AGEN_PRI_SIS
Status Monitor 1 (DUT)
Status Monitor X (DUT)
Status Monitor 1 (SISTER)
Status Monitor X (SISTER)

Figure 5–4 on page 5–7 shows the testbench environment for a SerialLite II standard broadcast mode MegaCore function with multiple SISTER instances that have one receive and transmit port.

Figure 5–4. SerialLite II Testbench Environment, Verilog HDL Only (Standard Broadcast Mode)



Simulation Flow

This section describes the basic steps to use the SerialLite II testbench. The SerialLite II testbench performs the following tests, if applicable:

- The testbench waits for the main reset sequence to end.
- The testbench waits for both SerialLite II links to come up (DUT and SISTER).
- If the regular data port is enabled, the testbench begins to send data from the data port Atlantic generators (DUT and SISTER side). The data Atlantic monitors check that the first data matches the first data sent from the generators and so on, until all the data is sent.

- In Verilog HDL only, if the priority data port is enabled, the testbench begins to send data from the priority port Atlantic generators. The priority Atlantic monitors 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 all monitors receive the last packet, the testbench finishes.

You can use the SerialLite II testbench as a template for creating your own testbench or modify it to increase the testing coverage.

Running a Simulation

Altera provides a ModelSim simulation script that 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 you have set the Quartus II project directory to be the working directory.

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

2. Execute the simulation run, by typing the appropriate command:

`do <variation name>_run_modelsim.tcl` (Verilog HDL)

or

`do <variation_name>_run_modelsim_vhdl.tcl` (VHDL)

The testbench creates the **run_modelsim.log** file as an output file.

☞ If you are using ModelSim versions later than 6.1g SE, the `vsim` command requires an additional switch `-voptargs=+acc` in order for the testbench to simulate successfully. This switch is automatically added in the **run_modelsim.tcl** file if the testbench is generated from the Quartus II software version 8.0 or later.

Simulation Pass and 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:

- Create data to be transported through the link

- Verify that the data arrived with or without errors
- Verify that the various protocols were honored in the delivery of the data
- Confirm 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 (*amon* modules) have not detected any errors in the received packets.

If no errors are detected, and all packets are received, the testbench issues a message stating that the simulation was successful.

If errors were detected, a message states that the testbench has failed. If not all packets have been detected, the testbench eventually times out (time limit set by WATCHTIME), which causes an error and the testbench to fail.

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 II 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. Example 5–1 shows the ModelSim log for a successful run.

Example 5–1. run_modelsim.log

```
*****************************************************************************
#CORE DUT : Comming out of RESET
# Note : CMU PLL is reset
# Time: 0 ns  Instance: tb.slite2_top_sis.nlOiOlO.m_cdr.m_rxpll
# Note : CMU PLL is reset
# Time: 0 ns  Instance: tb.slite2_top_dut.nlil1i.m_cdr.m_rxpll
# 0 ns VERIFY 0 of 1: example_tb
# ****************************************************************************
# CORE DUT : In RESET
# ****************************************************************************
# ****************************************************************************
# CORE SIS : In RESET
# ******************************************************************************
# Note : CMU PLL is reset
# Time: 2 ns  Instance: tb.slite2_top_dut.n1il1O
# Note : CMU PLL is reset
# Time: 2 ns  Instance: tb.slite2_top_sis.nlOiO0l
# ******************************************************************************
# CORE DUT : Comming out of RESET
# ******************************************************************************
# ******************************************************************************
# CORE SIS : Comming out of RESET
# ******************************************************************************
# Reset DONE = 1
# **************************
# ******* Link is up. ******
# **************************
# Linked Up, Utils ON
# AGEN_DAT_DUT   4: sent packet id=0 addr=0x14 size=268 err=1, time: 7276 ns
# AGEN_DAT_SIS  11: sent packet id=0 addr=0x9b size=282 err=0, time: 7428 ns
# AGEN_DAT_DUT   6: sent packet id=0 addr=0xe6 size=293 err=1, time: 7434 ns
# AGEN_DAT_DUT   7: sent packet id=0 addr=0xf7 size=379 err=1, time: 8176 ns
# AGEN_DAT_DUT   2: sent packet id=0 addr=0x62 size=373 err=1, time: 8244 ns
# AGEN_DAT_DUT  13: sent packet id=0 addr=0xdd size=446 err=0, time: 8402 ns
# 10000 ns : tb progressing..
# AMON_DAT_SIS   4: received packet id=0 addr=0x14 err=1, time: 11288 ns
# AMON_DAT_SIS   6: received packet id=0 addr=0xe6 err=1, time: 11572 ns
# AMON_DAT_SIS   7: received packet id=0 addr=0xf7 err=1, time: 12960 ns
# AMON_DAT_DUT  11: received packet id=0 addr=0x9b err=0, time: 12989 ns
# AMON_DAT_SIS   2: received packet id=0 addr=0x62 err=1, time: 13102 ns
# AMON_DAT_SIS  13: received packet id=0 addr=0xdd err=0, time: 13290 ns
# AMON_DAT_SIS:  Received ALL          5 packets, time: 13290 ns
# AGEN_DAT_SIS   5: sent packet id=0 addr=0xd5 size=645 err=1, time: 15328 ns
# AGEN_DAT_SIS  15: sent packet id=0 addr=0x0f size=678 err=0, time: 16059 ns
# AGEN_DAT_SIS   8: sent packet id=0 addr=0x98 size=848 err=0, time: 18330 ns
# AGEN_DAT_SIS   4: sent packet id=0 addr=0xa4 size=916 err=1, time: 18686 ns
# 20000 ns : tb progressing..
# AMON_DAT_DUT   5: received packet id=0 addr=0xd5 err=1, time: 21964 ns
# AMON_DAT_DUT  15: received packet id=0 addr=0x0f err=0, time: 22726 ns
# AMON_DAT_DUT   8: received packet id=0 addr=0x98 err=0, time: 25070 ns
# AMON_DAT_DUT   4: received packet id=0 addr=0xa4 err=1, time: 25263 ns
# AMON_DAT_DUT:  Received ALL          5 packets, time: 25263 ns
# 25263 ns RUNNING TESTCASE_END #1: example_tb
# ***************************************************************
# $$$ End of testbench example_tb at : 25263 ns
# $$$ AUTHOR: unknown
# $$$ DATE: `DATE
# RUNNING ACTUAL_TC =           1  RUNNING EXPECTED_TC =          1
# RUNNING ACTUAL_ERR =           0,
# $$$ Exit status for testbench example_tb :  TESTBENCH_PASSED
# ***************************************************************
# ** Note: Data structure takes 74588614 bytes of memory
#          Process time 495.56 seconds
#          $finish    : example_tb.v(1070)
#    Time: 25263352 ps  Iteration: 0  Instance: /tb
```

Value Change Dump (VCD) File Generation (For the Verilog HDL Testbench)

The simulation allows **.vcd** file generation if WAVEFORM is tick defined. All signals are included in the dump file (**dumpfile.vcd**).

Example 5–2. .vcd File Generation

```
# add the following tick define to the testbench to
# create a VCD
`define WAVEFORM

# add the following to the simulator command line to
# create a VCD dump file.
+define+WAVEFORM
```

Testbench Timeout

The testbench uses a maximum simulation time to guard against infinite loops or stuck simulations. The default value of 50,000,000 picoseconds is sufficient for most simulation runs. However, if more time is needed for a particularly long run, you can increase the WATCHTIME value. For example, change the already defined WATCHTIME inside the testbench main section to `` `define WATCHTIME 100,000,000 `` for Verilog HDL or for VHDL edit the <*variation_name*>**_tb.vhd** to change the constant `WATCHTIME : time := 100000000 ns;`

In Verilog HDL, an alternative to increasing the `WATCHTIME` is to reset the watch timer from time to time (for example, after each test case or even after each packet is sent) by adding the following line, as needed, to the testbench main section:

```
 reset_watchdog_timer;
```

Every time the `reset_watchdog_timer` task is called, the testbench timeout resets with another WATCHTIME time.

Special Simulation Configuration Settings

The SerialLite II MegaCore function contains few 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 count overrides the default value of 20,000.

- The clock compensation value determines when the clock compensation sequence is inserted into the high-speed serial stream (if **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 parts per million (ppm).

Atlantic Receiver Behavior

The receiver (Rx) Atlantic interface signals, other than `rxhpp/rxrdp_val`, can be x when the `rxhpp/rxrdp_val` is zero. Therefore, if the user logic uses the receive Atlantic interface when `rxhpp/rxrdp_val` is zero, the receiver MegaCore function can transmit x's when data is not valid. This invalid data should not be used during simulation.

To ensure valid data transmission, the receive Atlantic interface should only be sampled when the `rxhpp/rxrdp_val` is `1`.

Testbench Components Description

This section describes the testbench components.

DUT

The Verilog HDL or VHDL IP functional simulation model of the device under test (DUT).

SISTER

A Verilog HDL or VHDL IP functional simulation model used to test the DUT. When the DUT is asymmetric (for example, the number of receiving lanes is different than the number of transmitting lanes), is configured in single mode (receiver or transmitter only), or is configured in broadcast mode, the SISTER parameters may not match the DUT parameters, or multiple SISTER MegaCore functions may need to be instantiated.

AGEN

This testbench includes separate versions of the `AGEN` module for Verilog HDL and VHDL.

Verilog HDL

This Verilog HDL version of the AGEN module generates Atlantic data for the SerialLite II demonstration testbench (agen_dat_dut, agen_pri_dut, agen_dat_sis, agen_pri_sis, and so on). The data pattern is based on a linear feedback shift register (LFSR) to create a predictable but non-incrementing (pseudo-random) pattern.

This module features few tasks, the main one being the send_packet task that transmits packets into the SerialLite II MegaCore function. It also supports the streaming mode if the data port is configured as such.

The first byte of each generated packet is a sequential identifier (*id*) that seeds the LFSR. Every time the send_packet task is called, the *agen id* is incremented by one.

The module operates in one of two modes: data port or priority port. When in priority port mode, the Atlantic dav signal is ignored for all but the first transfer of a packet.

There can be multiple *agen* instantiations (for data and priority port, DUT and SISTER), depending on the DUT's chosen parameters.

AGEN Tasks

This sections defines the AGEN tasks.

– send_packet(addr,size[31:0],err)

send_packet is the main AGEN task. It causes a packet of a specified size and destined for a particular address to be transmitted. The err bit may also be assigned a value. The data is based on a LFSR.

Table 5–1 describes the send_packet task fields.

Table 5–1. Send_Packet Task Field Description

Field Location in Task	Field	Valid Values	Description
1	addr	0 to 0xFF (data) 0 to 0xF (priority)	Set to 0.
2	size	0 to 0xFFFF_FFFF (bytes)	The size field sets the size, in bytes, of the current packet being sent by this task.
3	err	1'b0 or 1'b1	The err field determines whether an Atlantic error is asserted at the end of a packet when eop is asserted. You can optionally set it to 1'b1 to set the error flag for that packet.

– `ipg(min[31:0],max[31:0])`

If the `gap` task is called, successive packets are separated by a a random number of idle cycles.

Table 5–2. GAP Task Field Description

Field Location in Task	Field	Valid Values	Description
1	`min`	0 to 0xFFFF_FFFF (cycles)	The `min` field sets the minimum value, in Atlantic clock cycles, for a random gap between two packets.
2	`max`	0 to 0xFFFF_FFFF (cycles)	The `max` field sets the maximum value, in Atlantic clock cycles, for a random gap between two packets. *A* `max` field greater than or equal to the `min` field is required. When `max==min`, no gap occurs.

– `gap(prob[31:0],min[31:0],max[31:0])`

If the `iptg` task is called, idle cycles are inserted between write operations. The probability of idles between write cycles decreases with larger values of `prob`.

Table 5–3. IPTG Task Field Description

Field Location in Task	Field	Valid Values	Description
1	`prob`	0 to 0xFFFF_FFFF (integer)	The `prob` field sets the probability of a transaction gap. The probability decreases with a larger value of `prob`. Before each transaction, a random number between 0 and `prob` is generated and compared to `prob/2`. If they match, a random gap is inserted; if not, no gap is inserted.
2	`min`	0 to 0xFFFF_FFFF (cycles)	The `min` field sets the minimum value, in Atlantic clock cycles, for a random gap between AGEN write transactions.
3	`max`	0 to 0xFFFF_FFFF (cycles)	The `max` field sets the maximum value, in Atlantic clock cycles, for a random gap between AGEN write transactions. A `max` field greater than or equal to the `min` field is required. When `max==min`, no gap occurs.

– `verbose(bit_value)`

The `verbose` task enables or disables the display of AGEN verbose messages.

Table 5–4. Verbose Task Field Description

Field Location in Task	Field	Valid Values	Description
1	`bit_value`	1'b0 or 1'b1	Setting `bit_value` to 1, enables the display of verbose messages. Setting `bit_value` to 0, disables the display of verbose messages (default).

– `corrupt_sop`

The `corrupt_sop` task corrupts the start of packet (SOP) of the next packet. When called, it waits for the SOP and corrupts it (makes SOP==0). All the subsequent packets are not corrupted.

– `corrupt_eop`

The `corrupt_eop` task corrupts the end of packet (EOP) of the next packet. When called, it waits for the EOP and corrupts it (makes EOP==0). All the subsequent packets are not corrupted.

AGEN Parameters

These parameters are set by the MegaWizard® Plug-In Manager, based on the selected configuration, and are fixed for a given SerialLite II configuration.

☞ These parameters are documented for reference purposes only. Do not modify them.

– **PRIORITY**

A value of one causes the model to generate data intended for a priority port, so that Atlantic `dav` signal is ignored for all but the first transfer of a packet. A value of zero causes the model to generate data intended for a data port, so `dav` is always obeyed.

Example 5–3. defparam PRIORITY

```
defparam agen_dat_dut.PRIORITY=0;

defparam agen_pri_dut.PRIORITY=1;
```

– **PORT_NAME**

A string used to distinguish between verbose messages coming from multiple instances of AGEN.

Example 5–4. defparam PORT_NAME

```
defparam agen_dat_dut.PORT_NAME = "AGEN_DAT_DUT";

defparam agen_pri_sis.PORT_NAME = "AGEN_PRI_SIS";
```

VHDL

The VHDL version of the `AGEN` module generates Atlantic data for the SerialLite II demonstration testbench (`agen_dat_dut`, `agen_dat_sis`). The data generated is based on an incrementing pattern.

The first element (at SOP) contains a decoded packet size for the packet. Once the packet is transmitted, the packet size count increases by one for the next packet so that successively larger packets are sent.

The AGEN generator sends packets until the internal packet count reaches the value of the `packets_to_end` input integer. Inner packet gaps can be optionally enabled by driving the `ipg` input to the module with a one. Doing so changes the behavior of the Atlantic write enable so that it is controlled by the output of a pseudo random generator. Verbose mode for the utility can be enabled by setting the `verbose` integer in the generic map to one.

AMON

This testbench includes separate versions of the `AMON` module for Verilog HDL and VHDL.

Verilog HDL

This Verilog HDL version of the `AMON` module monitors the Atlantic data received (instances: `amon_dat_dut`, `amon_pri_dut`, `amon_dat_sis`, `amon_pri_sis`, and so on). The data pattern received must be based on a LFSR that has produced a predictable but non-incrementing pattern.

The `AMON` monitor does the following basic checks:

- Data checking: checks that the received data follows the LFSR pattern
- *id* checking: checks that the packet identifier (first byte of each packet) is an incrementing number.
- Number of packets checking: checks that the expected number of regular data or high priority packets have been received. The expected number of packets is set via tasks.
- Start or end of packet checking: checks Atlantic packets for missing SOP and EOP signals

The module operates in one of two modes: data port or priority port. When in priority port mode, the `dav` signal is ignored for all but the first transfer of a packet.

There can be multiple `AMON` instantiations (for data and priority port, DUT and SISTER), depending on the DUT's chosen parameters.

AMON Tasks

– `data_checking(bit_value)`

This task enables or disables the data checking.

Table 5–5. Data_Checking Task Field Description

Field Location in Task	Field	Valid Values	Description
1	`bit_value`	1'b0 or 1'b1	Setting `bit_value` to 1, enables the data checking (default). Setting `bit_value` to 0, disables the data checking.

– `id_checking(bit_value)`

This task enables or disables the packet `id` checking.

Table 5–6. id_checking Task Field Description

Field Location in Task	Field	Valid Values	Description
1	`bit_value`	1'b0 or 1'b1	Setting `bit_value` to 1, enables the packet `id` checking (default). Setting `bit_value` to 0, disables the packet `id` checking.

– `wait_all_packets(number[31:0])`

This task waits until all packets (when in packet mode) or streaming bytes (when in streaming mode) are received.

Table 5–7. Wait_All_Packets Task Field Description

Field Location in Task	Field	Valid Values	Description
1	`number`	0 to 0xFFFF_FFFF	If in packet mode, this field sets the expected number of packets to be received. The task waits until all `number` of packets are received. If in streaming mode, this field sets the expected number of streaming bytes to be received. The task waits until all `number` of streaming bytes are received.

– `mp_checking(bit_value)`

This task enables or disables the missing SOP and EOP checking.

Table 5–8. MP_Checking Task Field Description

Field Location in Task	Field	Valid Values	Description
1	`bit_value`	1'b0 or 1'b1	Setting bit_value to 1, enables the missing SOP or EOP checking (default). Setting bit_value to 0, disables the missing SOP or EOP checking.

– `gap(prob[31:0],min[31:0],max[31:0])`

If this task is called, `amon` read operations may have some gaps between them. The probability of gaps between read cycles decreases with larger values of `prob`.

Table 5–9. Read_Transaction_Gap Task Field Description

Field Location in Task	Field	Valid Values	Description
1	`prob`	0 to 0xFFFF_FFFF (integer)	The `prob` field sets the probability for a read transaction gap to happen. Probability decreases with a larger value of `prob`. Before each read transaction a random number between 0 and `prob` is generated and compared to `prob/2.` If they match, a random gap is inserted in the read operation (`ena` goes low); if not, no gap is inserted.
2	`min`	0 to 0xFFFF_FFFF (cycles)	The `min` field sets the minimum value, in Atlantic clock cycles, for a random gap between `AMON` read transactions.
3	`max`	0 to 0xFFFF_FFFF (cycles)	The `max` field sets the maximum value, in Atlantic clock cycles, for a random gap between `AMON` read transactions. *A* `max` field greater than or equal to the `min` field is required. When `max==min`, no gap occurs.

– `verbose (bit_value)`

This task enables or disables the display of verbose messages.

Table 5–10. Verbose Task Field Description

Field Location in Task	Field	Valid Values	Description
1	`bit_value`	1'b0 or 1'b1	Setting `bit_value` to 1, enables the display of verbose messages. Setting `bit_value` to 0, disables the display of verbose messages (default).

AMON Parameters

These parameters are set by the MegaWizard Plug-In Manager, based on the selected configuration, and are fixed for a given SerialLite II configuration.

☞ These parameters are documented for reference purposes only. Do not modify them.

– **PRIORITY**

A value of one causes the model to receive data intended for a priority port, so that Atlantic `dav` signal is ignored for all but the first transfer of a packet. A value of zero causes the model to receive data intended for a data port, so `dav` is always obeyed.

Example 5–5. defparam Priority

```
defparam amon_dat_dut.PRIORITY=0;

defparam amon_pri_dut.PRIORITY=1;
```

– **PORT_NAME**

A string used to distinguish between verbose messages coming from multiple instances of AMON.

Example 5–6. defparam Port_name

```
defparam amon_dat_dut.PORT_NAME = "AMON_DAT_DUT";

defparam amon_pri_sis.PORT_NAME = "AMON_PRI_SIS";
```

VHDL

The VHDL version of the `AMON` module monitors the Atlantic data received (instances: `amon_dat_dut`, `amon_dat_sis`). The data received is based on a incrementing pattern.

The AMON monitor performs the following functions:

- Validates transmission of *individual packets* by extracting the intended packet size from the SOP and checking it against the actual value of the packet size counter in the EOP.
- Counts the *total number of packets* (provided as an output) to ensure that the all packets sent are also received.
- Checks Atlantic packets for missing SOP and EOP signals.

If any errors are detected by the `AMON` monitor, the `error_detect` output signal is asserted.

Inner packet read gaps can optionally be enabled by driving the `ipg` input to the module with a one. Doing so changes the behavior of the Atlantic read enable so that it is instead controlled by the output of a pseudo random generator. Verbose mode for the utility is enabled by setting the `verbose` integer in the generic map to one.

Status Monitors (pin_mon)

The simulation includes status pin monitors for the DUT and SISTERs (`pin_mon_`<*pin_name*>). When enabled (by default), the status monitor compares the received data against the expected data. If the expected value is different from the current value, the monitor flags an error.

Set the `en` input pin high to enable a pin monitor, low to disable a pin monitor, or for Verilog HDL only use the tasks. The Verilog HDL pin monitor expected value can be set by a task.

Pin_mon Tasks - Verilog HDL

Table 5–11 shows the function of the `pin_mon` tasks.

Table 5–11. Pin_mon Tasks

Task	Function
`on`	This task enables monitoring (the `en` input pin must also be set high to enable monitoring).
`off`	This task disables monitoring (regardless of the value of the `en` input pin).
`verbose_on`	This task enables the display of verbose messages.
`verbose_off`	This task disables the display of verbose messages.
`set_expect (bit_value)`	This task sets the expected pin value.

Clock and Reset Generator

The DUT and the SISTER use a common clock, with the frequency set by the MegaWizard interface.

There is one master reset signal (`reset_n`) that resets all the logic in the demonstration testbench (DUT, SISTER(s), AGENs, AMONs and status monitors).

Ensure `reset_n` to the MegaCore function starts high at `Time=0`, and then goes low for proper reset of the simulation model. Some simulators do not detect the transition if `reset_n` is asserted low at `T=0`.

To allow for easy modification, the reset section of the testbench is marked by start–end comment tags:

```
SERIALLITE2_TB_RESET_START

SERIALLITE2_TB_RESET_END
```

The clock and reset utilities are included in the testbench top-level file.

Example Testbench – Verilog HDL

To allow for easy modification of the demonstration testbench, its main section is marked by start–end tags:

```
//SERIALLITE2_TB_MAIN_START

//SERIALLITE2_TB_MAIN_END
```

Because there is no Atlantic to Atlantic score-boarding, the demonstration testbench focuses on passing error-free data rather than errored data. Any error condition that involves dropped or errored packets, must be handled in the testbench by setting proper expectations.

This section shows and explains a demonstration testbench main section example, allowing you to easily modify the testbench. You can change the packet size, port address, number of packets, and so on, or force certain behavior.

This example testbench may not match your testbench exactly.

Table 5–12. **Example of a Demonstration Testbench** *(Part 1 of 6)*

Main Section	Comments
`//SERIALLITE2_TB_MAIN_START`	Start of the testbench main section; the only section intended to be modified.

`integer pkt_cnt_dat_dut;`
`integer pkt_cnt_pri_dut;`
`integer pkt_cnt_dat_sis;`
`integer pkt_cnt_pri_sis;`
	Declare packet counters.

Table 5–12. Example of a Demonstration Testbench *(Part 2 of 6)*

Main Section	Comments
`//------------------------------------ --------------------- //Define the number of packets / streaming bytes to be sent //------------------------------------ --------------------- integer packets_to_send; initial packets_to_send = 5; integer streaming_bytes; initial streaming_bytes = 1500; //------------------------------------ ---------------------`	Defines the number of packets (5) or streaming bytes (1,500) to be sent.
`initial begin #1;`	Main initial block.
`exp_tc_cnt = 1;`	Sets expectation for the number of test cases (checks); this number must match the number of *tc_start/tc_end* pairs in the testbench, otherwise the testbench is declared INCOMPLETE.
`err_limit = 0;`	Sets expectation for the number of errors.
``tc_start(`TBID);``	Testcase start.
`wait (reset_n == 1);`	Waiting for the reset to complete; the reset is asserted in a separate initial block.
`// initialize packet counters`	
`pkt_cnt_dat_dut = packets_to_send;`	Sets the number of packets to be sent to the regular data port of the DUT MegaCore function.
`pkt_cnt_pri_dut = packets_to_send;`	Sets the number of packets to be sent to the high priority port of the DUT MegaCore function.
`pkt_cnt_dat_sis = packets_to_send;`	Sets the number of packets to be sent to the regular data port of the SISTER MegaCore function.
`pkt_cnt_pri_sis = packets_to_send;`	Sets the number of packets to be sent to the high priority port of the SISTER MegaCore function.
`wait (linked_up == 1);`	Wait for DUT and SISTER to go into link-up.
`fork`	Launch multiple send packet loops in parallel.

Table 5–12. **Example of a Demonstration Testbench** *(Part 3 of 6)*

Main Section	Comments
`begin`	Send regular data packets (on Atlantic interface) to the DUT.
`///////////////////////////////////////` `///////`	AGEN and AMON instantiations are set to display verbose messages.
` // Generate RDP packets for DUT`	Set AGEN to insert random inner packet gaps.
`///////////////////////////////////////` `///////`	
` @(posedge trefclk);`	Launch two processes in parallel:
` agen_dat_dut.verbose(1);`	- Send regular data packets to the DUT.
` agen_dat_dut.ipg(0,5);`	
` amon_dat_sis.verbose(1);`	Define packet size, error, address.
` fork`	

` while (pkt_cnt_dat_dut > 0) begin :`
`send_loop_dat_dut`
	Packet address is a random number from 0 to 255.
	Packet size is a random number from 1 to 1,024.
	Packet err is a random number from 0 to 1.

` integer size;`	
` integer err;`	Call the AGEN send packet task (regular data, DUT).
` reg [7:0] addr;`	Reset watchdog with every packet being sent. Repeat this loop *pkt_cnt_dat_dut* times.
` addr = $dist_uniform(seed,0,255);`	
` size = $dist_uniform(seed,1,1024);`	- Wait for the other side (Atlantic interface of the SISTER) to receive all these packets.
` err = $dist_uniform(seed,0,1);`	

` agen_dat_dut.send_packet(ad-`
`dr,size,err);`	

` reset_watchdog_timer;`	
` pkt_cnt_dat_dut = pkt_cnt_dat_dut - 1;`	
` end`	
` begin`	
` fork`	

`amon_dat_sis.wait_all_packets(packets_t`
`o_send);`	

` join`	
` end`	
` join`	
` end`	

Table 5–12. Example of a Demonstration Testbench *(Part 4 of 6)*

Main Section	Comments

`begin`
 	
`////////////////////////////////////////////////`
`// Generate RDP packets for SISTER`
 	
`////////////////////////////////////////////////`
`@(posedge trefclk);`
`agen_dat_sis.verbose(1);`
`agen_dat_sis.ipg(0,5);`
`amon_dat_dut.verbose(1);`
`fork`
`while ( pkt_cnt_dat_sis > 0 ) begin : send_loop_dat_sis`
`integer size;`
`integer err;`
`reg [7:0] addr;`
 	
`addr = $dist_uniform(seed,0,255);`
`size = $dist_uniform(seed,1,1024);`
`err = $dist_uniform(seed,0,1);`
`agen_dat_sis.send_packet(ad-dr,size,err);`
`reset_watchdog_timer;`
`pkt_cnt_dat_sis = pkt_cnt_dat_sis - 1;`
`end`
`begin`
 	
`amon_dat_dut.wait_all_packets(packets_to_send);`
`end`
`join`
`end`
	Send regular data packets (on Atlantic interface) to the SISTER MegaCore function.
	AGEN and AMON instantiations are set to display verbose messages.
	Set AGEN to insert random inner packet gaps.
	
	Launch two processes in parallel:
	- Send regular data packets to the SISTER.
	
	Define packet size, error, address.
	
	Packet address is a random number from 0 to 255.
	Packet size is a random number from 1 to 1,024.
	Packet err is a random number from 0 to 1.
	
	Call the AGEN send packet task (regular data, SISTER).
	Reset watchdog with every packet being sent.
	Repeat this loop *pkt_cnt_dat_sis* times.
	
	- Wait for the other side (Atlantic interface of the DUT) to receive all these packets.

Table 5–12. **Example of a Demonstration Testbench** ***(Part 5 of 6)***

Main Section	Comments

`begin`
 	
`//////////////////////////////////////////////`
`// Generate HPP priority packets for DUT`
`//////////////////////////////////////////////`
`agen_pri_dut.verbose(1);`
`agen_pri_dut.ipg(0,5);`
`amon_pri_sis.verbose(1);`
`fork`
`while ( pkt_cnt_pri_dut > 0 ) begin : send_loop_pri_dut`
`integer size;`
`integer err;`
`reg [3:0] addr;`
 	
`addr = $dist_uniform(seed,0,15);`
`size = $dist_uniform(seed,1,780);`
`err = ( $dist_uniform(seed,0,8) == 4 ) ? 1'b1 : 1'b0;`
`agen_pri_dut.send_packet(addr,size,err);`
 	
`reset_watchdog_timer;`
`pkt_cnt_pri_dut = pkt_cnt_pri_dut - 1;`
`end`
`begin`
`fork`
 	
`amon_pri_sis.wait_all_packets(packets_to_send);`
`join`
`end`
`join`
`end`
	Send high priority packets (on Atlantic interface) to the DUT MegaCore function.
	AGEN and AMON instantiations are set to display verbose messages.
	Set AGEN to insert random inner packet gaps.
	
	Launch two processes in parallel:
	- Send high priority packets to the DUT.
	
	Define packet size, error, address.
	
	Packet address is a random number from 0 to 15.
	Packet size is a random number from 1 to 780.
	Packet err is a random number from 0 to 1.
	
	Call the AGEN send packet task (high priority, DUT)
	Reset watchdog with every packet being sent.
	Repeat this loop *pkt_cnt_pri_dut* times.
	
	- Wait for the other side (Atlantic interface of the SISTER) to receive all these packets.

Table 5–12. **Example of a Demonstration Testbench** *(Part 6 of 6)*

Main Section	Comments
`begin` `/////////////////////////////////////////////////` `// Generate HPP priority packets for SISTER` `/////////////////////////////////////////////////` `agen_pri_sis.verbose(1);` `agen_pri_sis.ipg(0,5);` `amon_pri_dut.verbose(1);` `fork` `while ( pkt_cnt_pri_sis > 0 ) begin : send_loop_pri_sis` `integer size;` `integer err;` `reg [3:0] addr;` `addr = $dist_uniform(seed,0,15);` `size = $dist_uniform(seed,1,780);` `err = ( $dist_uniform(seed,0,8) == 4 ) ? 1'b1 : 1'b0;` `agen_pri_sis.send_packet(addr,size,err);` `reset_watchdog_timer;` `pkt_cnt_pri_sis = pkt_cnt_pri_sis - 1;` `end` `begin` `amon_pri_dut.wait_all_packets(packets_to_send);` `end` `join` `end`	Send high priority packets (on Atlantic interface) to the SISTER MegaCore function. AGEN and AMON instantiations are set to display verbose messages. Set AGEN to insert random inner packet gaps. Launch two processes in parallel: - Send high priority packets to the SISTER. Define packet size, error, address. Packet address is a random number from 0 to 15. Packet size is a random number from 1 to 780. Packet err is a random number from 0 to 1. Call the AGEN send packet task (high priority, SISTER). Reset watchdog with every packet being sent. Repeat this loop *pkt_cnt_pri_sis* times. - Wait for the other side (Atlantic interface of the DUT) to receive all these packets.
`join`	All loops must finish (receive all packets) before exiting.
`tc_end(`TBID);` `exit;` `end`	End of test case. Main initial block end.
`endmodule`	End of testbench main section.
`//SERIALLITE2_TB_MAIN_END`	

Additional Information

Document Revision History

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

Document Revision History

Chapter	Date & Document Version	Changes Made	Summary of Changes
All	May 2008 v8.0	● Added additional alt2gxb parameters - VCCH, Reference Input Clk Frequency and Reconfiguration Channel Number ● Added Stratix IV device support	–
All	October 2007 v7.2.0	● Added gxb_powerdown signal to ease merging of multiple SerialLite II cores into the same transceiver block. ● Added VHDL testbench for some configurations. ● Moved the alt2gxb megafunction instantiation into a separate file to ease post-generation changes	Organized to present critical information first. Clarified a number of technical details.
	October 2005 v1.0.0	● Initial release of this user guide.	–
1	October 2007 v7.2.0	● Updated the release information. ● Updated the performance information.	–
	May 2007 v7.1	● Updated the release information.	–
	December 2006 v7.0	● Updated the release information.	–
	December 2006 v6.1	● Updated the release information. ● Updated the performance information.	–
	April 2006 v1.1.0	● Updated the release information. ● Updated the performance information.	–
	December 2005 v1.0.1	● Updated the release information. ● Updated the performance information.	–

Document Revision History

Chapter	Date & Document Version	Changes Made	Summary of Changes
2	December 2006 v7.0	● Updated the release information.	–
	December 2006 v6.1	● Updated design flow with MegaWizard® interface.	–
	April 2006 v1.1.0	● Updated the format. ● Added Table 2–1 on page 2–5. ● Updated Generated Files Table..	–
	December 2005 v1.0.1	● Updated the system requirements.	–
3	October 2007 v7.2	● Corrected error in Table 3-12: for the `reconfig_togxb` input signal, "If no external `altt2gxb_reconfig` block is used, then you can tie this bus to 3'b010." ● Added information describing core's ability to re-order lanes	–
	May 2007 v7.1	● Fixed floating `refclk` in Figure 3-9. Streaming Full Featured Clock Structure ● Added `rxhpp_oflw` signal ● Explained error signal behavior for signals such as `txrdp_err` and `txhpp_err` when such signals are asserted by user logic.	–
	December 2006 v7.0	● Updated the release information.	–
	December 2006 v6.1	● Updated signal tables. ● Added information for al2gxb interface.	–
	April 2006 v1.1.0	● Made changes throughout this chapter. ● Added clock structure figures. ● Added and updated signals.	–
	December 2005 v1.0.1	● Added streaming mode diagrams. ● Added retry-on-error information.	–

Document Revision History

Chapter	Date & Document Version	Changes Made	Summary of Changes
4	October 2007 v7.2.0	• Added information on `gxb_powerdown` signal • Added information on using multiple cores	–
	May 2007 v7.1	• Clarified `reset` requirements for testbench simulation	–
	December 2006 v7.0	• Updated the release information.	–
	December 2006 v6.1	• Added information about the Atlantic interface receiver signals	–
	April 2006 v1.1.0	• Made minor changes to some task and field names.	–
	December 2005 v1.0.1	• Added pin monitors. • Updated the testbench example (Table 5–12).	–
5	October 2007 v7.2.0	• Added VHDL testbench for some configurations	–

How to Contact Altera

For the most up-to-date information about Altera products, refer to the following table.

Contact *(1)*	Contact Method	Address
Technical support	Website	www.altera.com/support
Technical training	Website	www.altera.com/training
	Email	custrain@altera.com
Product literature	Website	www.altera.com/literature
Non-technical support (General)	Email	nacomp@altera.com
(Software Licensing)	Email	authorization@altera.com

Note to table:

(1) You can also contact your local Altera sales office or sales representative.

Typographic Conventions

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: **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`, for example, `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 (for example, the VHDL keyword `BEGIN`), as well as logic function names (for example, `TRI`) are shown in Courier.
1., 2., 3., and a., b., c., etc.	Numbered steps are used in a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■ ● •	Bullets are used in a list of items when the sequence of the items is not important.
✓	The checkmark indicates a procedure that consists of one step only.
☞	The hand points to information that requires special attention.
⚠ CAUTION	A caution calls attention to a condition or possible situation that can damage or destroy the product or the user’s work.
⚠ WARNING	A warning calls attention to a condition or possible situation that can cause injury to the user.
↵	The angled arrow indicates you should press the Enter key.
👣	The feet direct you to more information on a particular topic.