



SerialLite II IP Core

User Guide



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

UG-0705-2014.07.09

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

14.0
July 2014



[Subscribe](#)

© 2014 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.



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

Release Information

Table 1–1. SerialLite II Release Information

Item	Description
Version	14.0
Release Date	July 2014
Ordering Code	IP-SLITE2
Product ID	00AD
Vendor ID	6AF7

Altera verifies that the current version of the Quartus® II software compiles the previous version of each IP core. The [MegaCore IP Library Release Notes and Errata](#) report any exceptions to this verification. Altera does not verify compilation with IP core versions older than one release.

Device Family Support

MegaCore functions IP cores provide the following support for Altera device families:

- *Preliminary support*—Altera verifies the IP core with preliminary timing models for this device family. The core meets all functional requirements, but might still be undergoing timing analysis for the device family. It can be used in production designs with caution.
- *Final support*—Altera verifies the IP core with final timing models for this device family. The core meets all functional and timing requirements for the device family and can be used in production designs.

[Table 1–2](#) shows the level of support offered by the SerialLite II IP core to each Altera device family.

Table 1–2. Device Family Support (Part 1 of 2)

Device Family	Support
Arria® II GX	Final
Arria V	Preliminary
Arria V GZ	Preliminary

Table 1–2. Device Family Support (Part 2 of 2)

Device Family	Support
Cyclone® V	Preliminary
Stratix® IV	Final
Stratix V	Preliminary
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 (LSM)
 - 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
- Lane order reversal
- IP functional simulation models for use in Altera-supported VHDL and Verilog HDL simulators
- Support for OpenCore® Plus evaluation

- Streaming video applications
- Imaging applications

Figure 1-2 and Figure 1-3 show two examples of bridging applications.

Figure 1-2. Typical Application—Bridging Functions

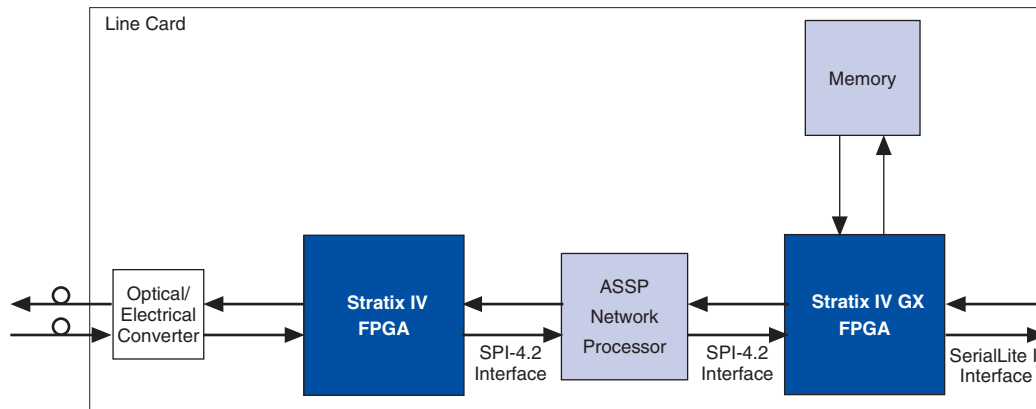
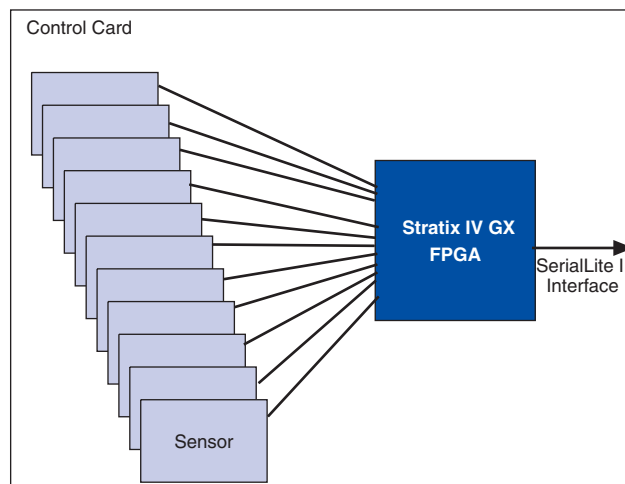


Figure 1-3. Typical Application—Unidirectional Bridging Application

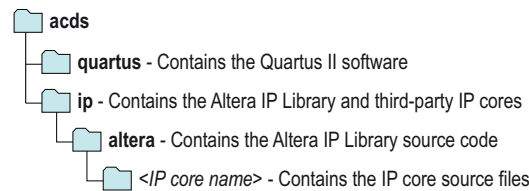


Installing and Licensing IP Cores

The Quartus II software includes the Altera IP Library. The library provides many useful IP core functions for production use without additional license. You can fully evaluate any licensed Altera IP core in simulation and in hardware until you are satisfied with its functionality and performance.

Some Altera IP cores, such as MegaCore® functions, require that you purchase a separate license for production use. After you purchase a license, visit the [Self Service Licensing Center](#) to obtain a license number for any Altera product. For additional information, refer to [Altera Software Installation and Licensing](#).

Figure 1–4. IP core Installation Path



The default installation directory on Windows is `<drive>:\altera\<version number>`; on Linux it is `<home directory>/altera/<version number>`.

OpenCore Plus IP Evaluation

Altera's free OpenCore Plus feature allows you to evaluate licensed MegaCore IP cores in simulation and hardware before purchase. You need only purchase a license for MegaCore IP cores if you decide to take your design to production. OpenCore Plus supports the following evaluations:

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

OpenCore Plus evaluation supports the following two operation modes:

- Untethered—run the design containing the licensed IP for a limited time.
- Tethered—run the design containing the licensed IP for a longer time or indefinitely. This requires a connection between your board and the host computer.

All IP cores using OpenCore Plus in a design time out simultaneously when any IP core times out.

Upgrading Outdated IP Cores

Each IP core has a release version number that corresponds to its Quartus II software release. When you include IP cores from a previous version of the Quartus II software in your project, click **Project > Upgrade IP Components** to identify and upgrade any outdated IP cores.

The Quartus II software prompts you to upgrade an IP core when the latest version includes port, parameter, or feature changes. The Quartus II software also notifies you when IP cores are unsupported or cannot upgrade in the current version of the Quartus II software. Most Altera IP cores support automatic simultaneous upgrade, as indicated in the **Upgrade IP Components** dialog box. IP cores unsupported by auto-upgrade may require regeneration in the parameter editor, as indicated in the **Upgrade IP Components** dialog box.

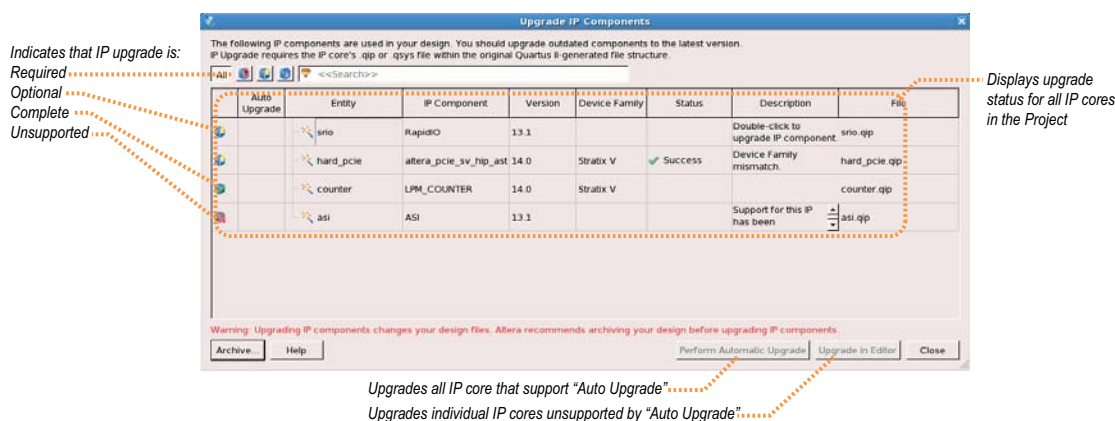
Upgrading IP cores changes your original design files. If you have not already preserved your original source files, click **Project > Archive Project** and save the project archive.

To upgrade outdated IP cores, follow these steps:

1. In the latest version of the Quartus II software, open the Quartus II project containing an outdated IP core variation.
2. Click **Project > Upgrade IP Components**. The **Upgrade IP Components** dialog box displays all outdated IP cores in your project, along with basic instructions for upgrading each core.
3. To simultaneously upgrade all IP cores that support automatic upgrade, click **Perform Automatic Upgrade**. The IP variation upgrades to the latest version.
4. To upgrade IP cores unsupported by automatic upgrade, follow these steps:
 - a. Select the IP core in the **Upgrade IP Components** dialog box.
 - b. Click **Upgrade in Editor**. The original parameter editor appears.
 - c. Click **Finish** or **Generate** to regenerate the IP variation and complete the upgrade. The version number updates when complete.

Example designs provided with any Altera IP core regenerate automatically whenever you upgrade the IP core in the **Upgrade IP Components** dialog box.

Figure 1-5. Upgrading IP Cores



Upgrading IP Cores at the Command Line

Alternatively, you can upgrade IP cores at the command line. To upgrade a single IP core, type the following command:

```
quartus_sh --ip_upgrade -variation_files <my_ip_path> <project>
```


To upgrade a list of IP cores, type the following command:

```
quartus_sh --ip_upgrade -variation_files  
"<my_ip>.qsys;<my_ip>.<hdl>; <project>"
```

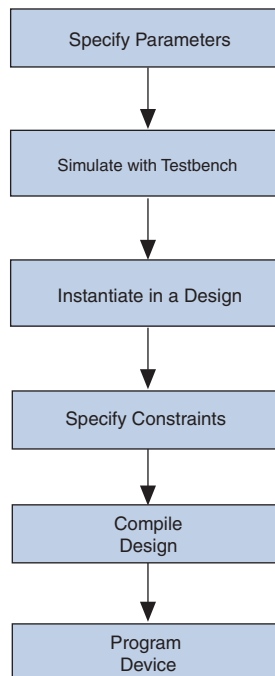


IP cores older than Quartus II software version 12.0 do not support upgrade. Altera verifies that the current version of the Quartus II software compiles the previous version of each IP core. The *MegaCore IP Library Release Notes* reports any verification exceptions for MegaCore IP. The *Quartus II Software and Device Support Release Notes* reports any verification exceptions for other IP cores. Altera does not verify compilation for IP cores older than the previous two releases.

Design Flow

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

Figure 2–1. SerialLite II IP Core Design Flow




This chapter explains how to define and instantiate a custom variation of the SerialLite II IP core using the IP Catalog and parameter editor.

IP Catalog and Parameter Editor

The Quartus II IP Catalog (**Tools > IP Catalog**) and parameter editor help you easily customize and integrate IP cores into your project. You can use the IP Catalog and parameter editor to select, customize, and generate files representing your custom IP variation.

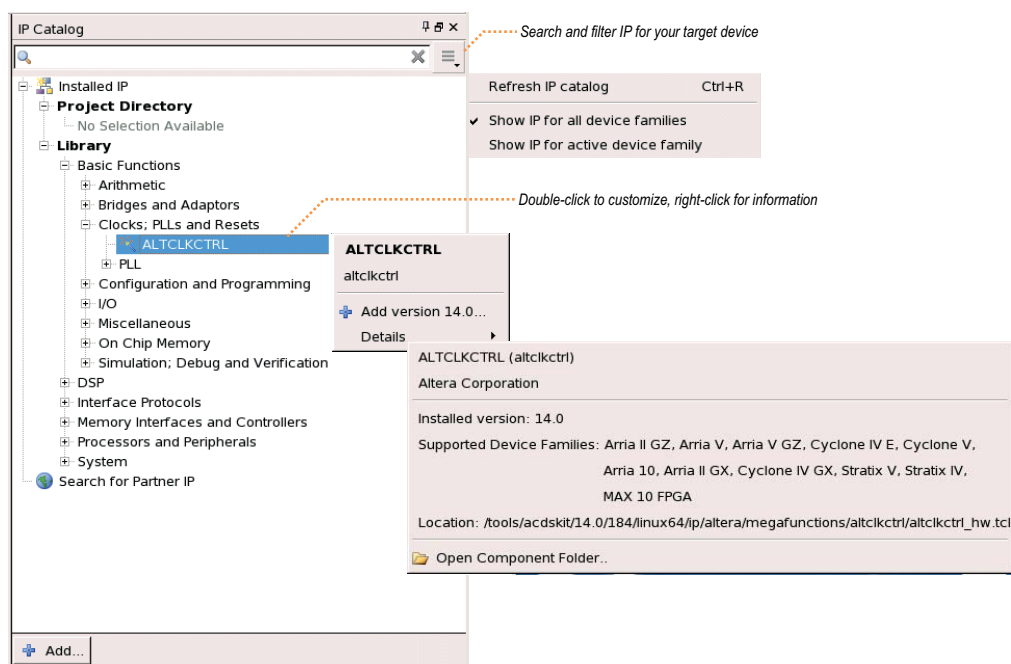
The IP Catalog automatically displays the IP cores available for your target device. Double-click any IP core name to launch the parameter editor and generate files representing your IP variation. The parameter editor prompts you to specify your IP variation name, optional ports, architecture features, and output file generation options. The parameter editor generates a top-level **.qsys** or **.qip** file representing the IP core in your project. Alternatively, you can define an IP variation without an open Quartus II project. When no project is open, select the **Device Family** directly in IP Catalog to filter IP cores by device.


 The IP Catalog is also available in Qsys (**View > IP Catalog**). The Qsys IP Catalog includes exclusive system interconnect, video and image processing, and other system-level IP that are not available in the Quartus II IP Catalog.

Use the following features to help you quickly locate and select an IP core:

- Filter IP Catalog to **Show IP for active device family** or **Show IP for all device families**.
- Search to locate any full or partial IP core name in IP Catalog. Click **Search for Partner IP**, to access partner IP information on the Altera website.
- Right-click an IP core name in IP Catalog to display details about supported devices, installation location, and links to documentation.

Figure 2–2. Quartus II IP Catalog



 The IP Catalog and parameter editor replace the MegaWizard™ Plug-In Manager in the Quartus II software. The Quartus II software may generate messages that refer to the MegaWizard Plug-In Manager. Substitute “IP Catalog and parameter editor” for “MegaWizard Plug-In Manager” in these messages.

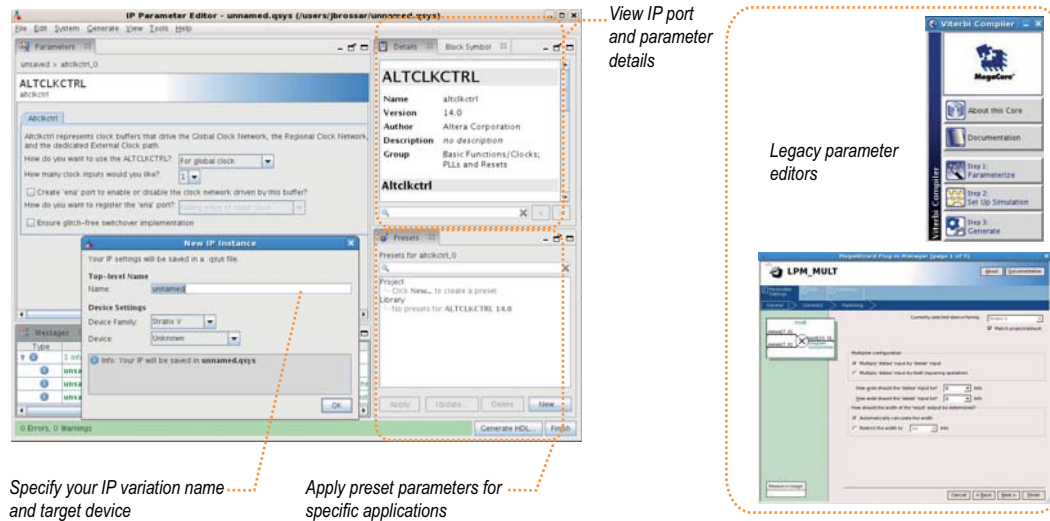
Using the Parameter Editor

The parameter editor helps you to configure your IP variation ports, parameters, architecture features, and output file generation options:

- Use preset settings in the parameter editor (where provided) to instantly apply preset parameter values for specific applications.
- View port and parameter descriptions and links to detailed documentation.

- Generate testbench systems or example designs (where provided).

Figure 2-3. IP Parameter Editors



Customizing and Generating IP Cores

You can customize IP cores to support a wide variety of applications. The Quartus II IP Catalog displays IP cores available for the current target device. The parameter editor guides you to set parameter values for optional ports, features, and output files.

To customize and generate a custom IP core variation, follow these steps:

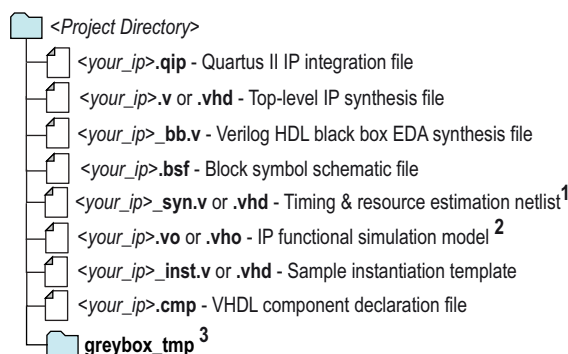
1. In the IP Catalog (**Tools > IP Catalog**), locate and double-click the name of the IP core to customize. The parameter editor appears.
2. Specify a top-level name for your custom IP variation. This name identifies the IP core variation files in your project. If prompted, also specify the target Altera device family and output file HDL preference. Click **OK**.
3. Specify the desired parameters, output, and options for your IP core variation:
 - Optionally select preset parameter values. Presets specify all initial parameter values for specific applications (where provided).
 - Specify parameters defining the IP core functionality, port configuration, and device-specific features.
 - Specify options for generation of a timing netlist, simulation model, testbench, or example design (where applicable).
 - Specify options for processing the IP core files in other EDA tools.
4. Click **Finish** or **Generate** to generate synthesis and other optional files matching your IP variation specifications. The parameter editor generates the top-level **.qip** or **.qsys** IP variation file and HDL files for synthesis and simulation. Some IP cores also simultaneously generate a testbench or example design for hardware testing.

When you generate the IP variation with a Quartus II project open, the parameter editor automatically adds the IP variation to the project. Alternatively, click **Project > Add/Remove Files in Project** to manually add a top-level .qip or .qsys IP variation file to a Quartus II project. To fully integrate the IP into the design, make appropriate pin assignments to connect ports. You can define a virtual pin to avoid making specific pin assignments to top-level signals.

Files Generated for Altera IP Cores

The Quartus II software generates the following files during generation of your IP core variation.

Figure 2-4. IP Core Generated Files



Notes:

1. If supported and enabled for your IP variation
2. If functional simulation models are generated
3. Ignore this directory

Simulating IP Cores

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

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

For more information about simulating Altera IP cores, refer to *Simulating Altera Designs* in volume 3 of the *Quartus II Handbook*.

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 IP core variation as a standalone component, you must specify virtual pin assignments. The SerialLite II parameter editor 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 (f_{MAX}). Use this script as a guide to set constraints for the SerialLite II IP core variation in your design. The timing constraints are currently set for the SerialLite II IP core 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 IP core variation into your design and simulate and compile.

Timing Constraints

The SerialLite II IP core 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 IP core 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

Click **Start Compilation** 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.

Table 3–1 shows the function parameters, which can be set only in the SerialLite II parameter editor (refer to “Parameterize” on page 2–3). The following sections describe these parameters.

Table 3–1. Default SerialLite II Variation

Parameter	Default Configuration
Physical Layer	
Device family	Depends on the family specified in the SerialLite II parameter editor
Data rate	3,125 megabits per second (Mbps)
Transfer size	2 Columns
Reference Clock Frequency	156.25 MHz
Port Type	Bidirectional
Self-Synchronized Link-Up	Disabled
Number of lanes (Transmitter and Receiver Settings)	1
Scramble/De-Scramble	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 (Transmitter and Receiver)	Disabled
Transceiver Configuration	
Voltage Output Differential (V_{OD})	0
Pre-emphasis control setting	0
Transmitter Buffer Power (V_{CCH})	1.5
Equalizer control setting	0
Bandwidth mode (Transmitter and Receiver)	Low
Starting channel number for reconfiguration	0

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

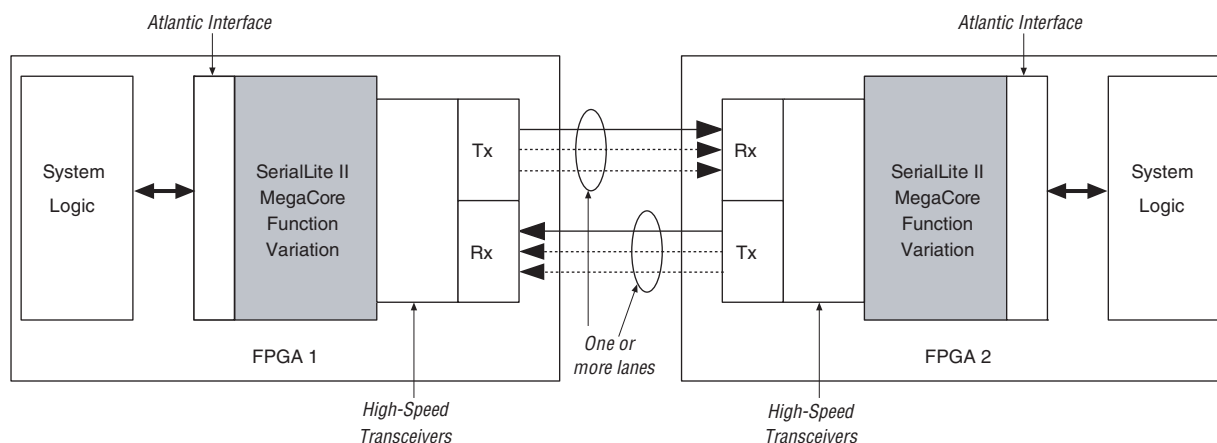
- High-level link configuration
- Bandwidth required
- Whether to use CRC checking
- Whether to implement flow control

- 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 IP core variation.

Data Rate

The SerialLite II IP core supports a data rate range of 622 to 6,375 Mbps per lane. In Arria II GX devices, the data rate must be less than 3,750 Mbps, and in Stratix IV devices, less than 6,375 Mbps. The data rate range varies based on the device and the transfer size (TSIZE) as [Table 3-2 on page 3-2](#) illustrates.

Table 3-2. Data Rate Dependencies on Transfer Size

Devices	Data Rate				
	2.5 Gbps	3.125 Gbps	3.75 Gbps	5 Gbps	6.375 Gbps
Arria II GX	TSIZE= 1, 2	TSIZE= 2	TSIZE= 2	Not Supported	Not Supported
Stratix IV GX	TSIZE= 1, 2	TSIZE= 2	TSIZE= 4	TSIZE= 4	TSIZE= 4
Stratix IV GT	—	TSIZE= 2	TSIZE= 4	TSIZE= 4	TSIZE= 4

 For Arria V, Cyclone V, and Stratix V devices, refer to [Table 4-2 on page 4-19](#).

The data rates for an individual Arria II GX device are limited to the respective speed grades, refer to [Table 3-3](#).

Table 3-3. Arria II GX Speed Grade-Data Rate Limits

Device Speed Grade	Minimum Data Rate (Mbps)	Maximum Data Rate (Mbps)
C4	600	3,750
C5	600	3,125
C6	600	3,125

Transfer size

The **Transfer size** parameter defines many important characteristics of the IP core 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 (only available for Stratix IV FPGA with transfer size greater than 3.125 Gbps, and must be used when the data rate exceeds 5 Gbps)

A transfer size 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

Reference Clock Frequency

The **Reference Clock Frequency** parameter defines the frequency of the reference clock for the Arria II GX or Stratix IV internal transceiver. 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:

$$\text{Frequency} = p \times \text{Data Rate} / (2 \times m)$$
 where $p = 1$ or 2 , and $m = 4, 5, 8, 10, 16, 20$, or 25
 Condition for frequency to be valid: $(50 \times p) < \text{Frequency} < 622$
- This parameter is only applicable if you chose Arria II GX or Stratix IV devices.
- If you select a reference clock frequency that is not equal to the data rate/(transfer size) * 10, the **Clock Compensation** option is disabled if the **Receiver only** port type option is turned on.

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**.

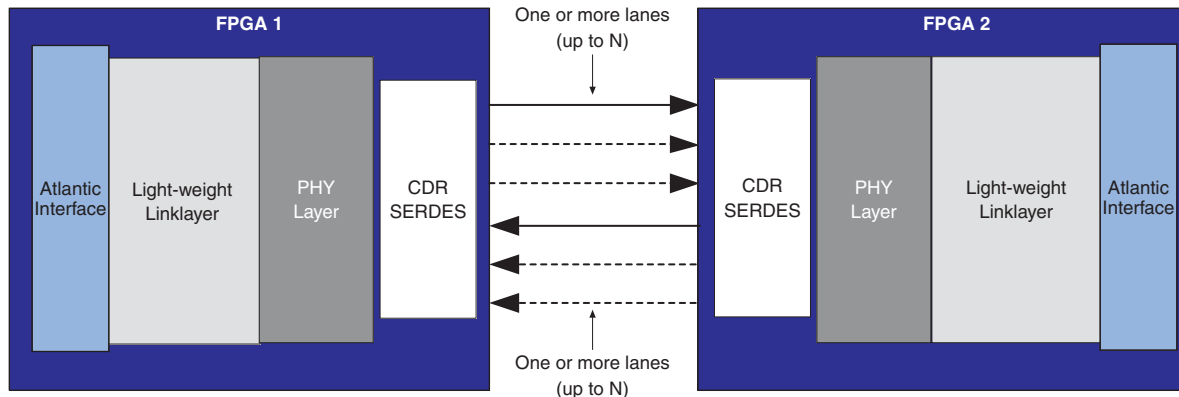
The **Number of lanes** parameter dictates the number of serial links, essentially the number of external inputs and outputs (I/Os) for the IP core.

If you set the **Number of lanes** for the transmitter and receiver settings to the same value, you configure the IP core to operate in symmetric, bidirectional mode. Refer to [Figure 3-2](#) and [Figure 3-3](#) on page 3-5.

If you set the **Port Type** to **Receiver only** or **Transmitter only**, you configure the IP core to operate in unidirectional mode, transmitter, or receiver only. Refer to [Figure 3-4](#) and [Figure 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 IP core to operate in asymmetric mode. Refer to [Figure 3-6](#) and [Figure 3-7](#) on page 3-7.

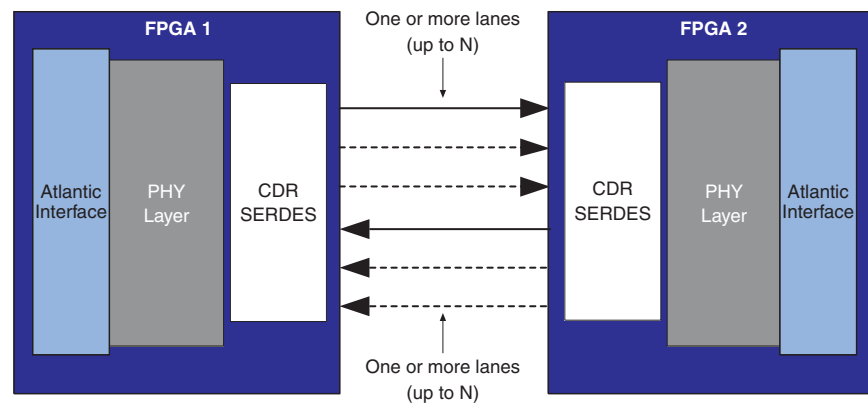
Figure 3-2. Symmetric Mode Block Diagram



Notes to Figure 3-2:

- (1) A full line indicates a mandatory lane.
- (2) A dashed line indicates an optional lane.

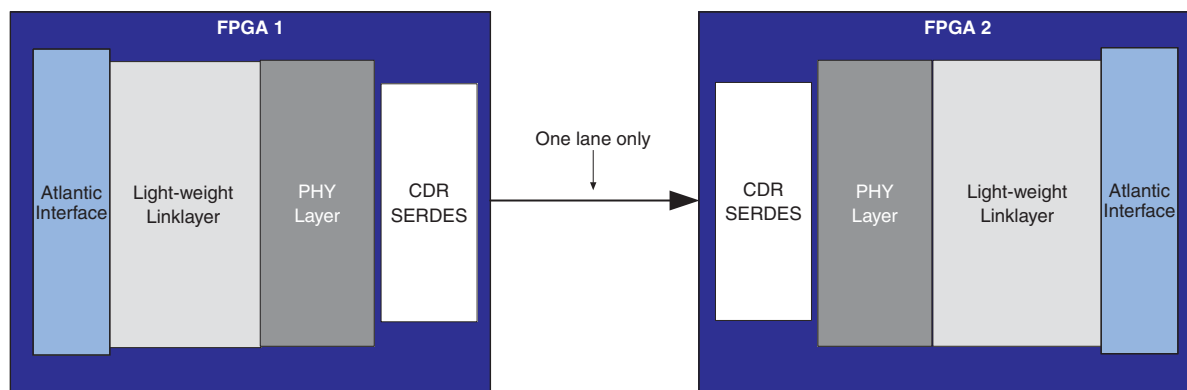
Figure 3–3. Streaming Symmetric Mode Block Diagram



Notes to Figure 3–3:

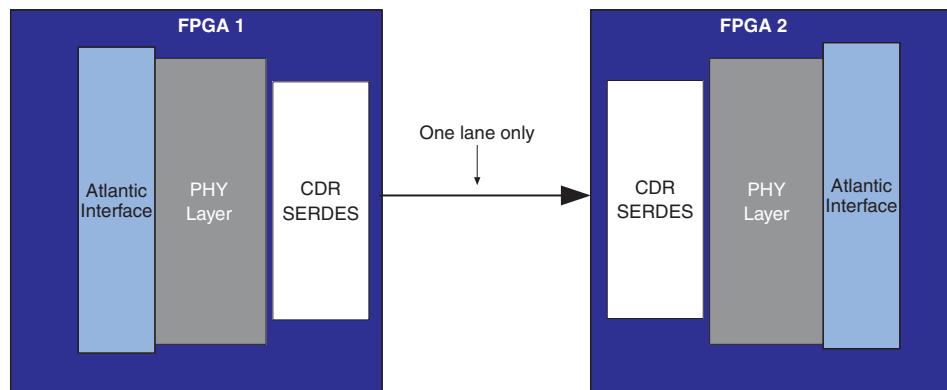
- (1) A full line indicates a mandatory lane.
- (2) A dashed line indicates an optional lane.

Figure 3–4. Simplex Mode Block Diagram

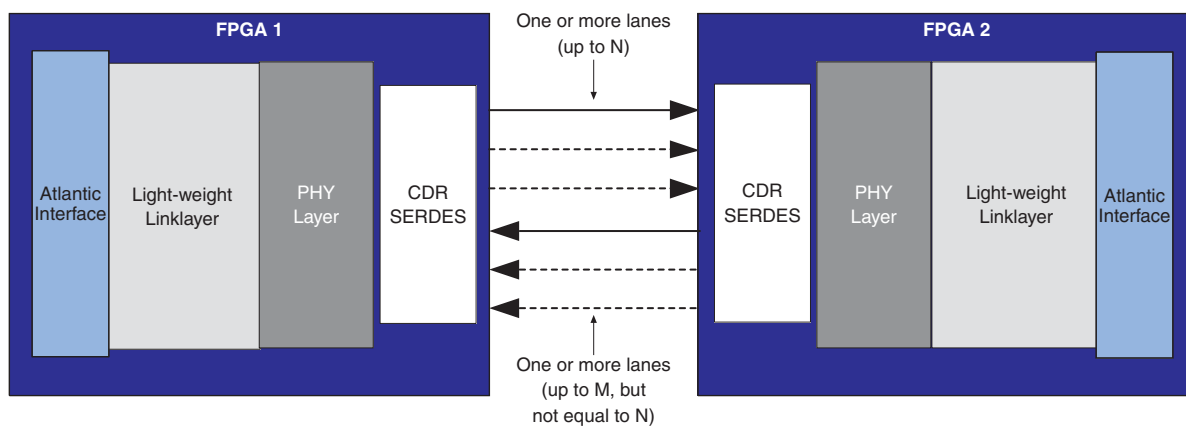


Note to Figure 3–4:

- (1) A full line indicates a mandatory lane.

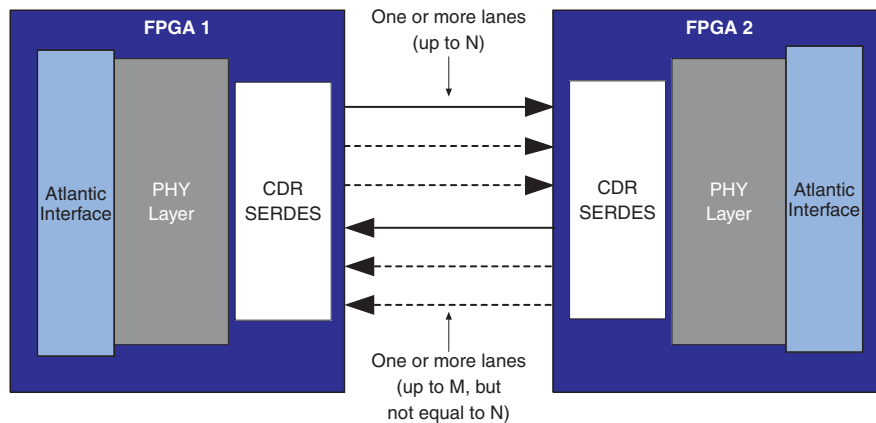
Figure 3-5. Streaming Simplex Mode Block Diagram**Note to Figure 3-5:**

- (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 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 LSM. 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 IP core 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 IP core completes the power-on reset cycle. Therefore, the self-synchronizing link-up state machine does not look at the incoming stream until the transceiver reset is complete.

For example, the following procedure shows the transceiver reset sequence in an Arria or Stratix transceiver device:

1. Wait for the `p11_locked` signal (`stat_tc_p11_locked`) to be asserted, which happens when the PLL in the ALTGX IP core locks to the reference clock (`trfclk`). The reference clock must be characterized; 10 ms or less is normal.
2. Wait for the `rx_freqlocked` signal (`stat_rr_freqlock`) to be asserted, which happens when the ALTGX IP core 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 IP core also has a reset done status signal (`stat_tc_rst_done`) that can be useful for measurements. The following IP core 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 completes, the IP core waits for the transceiver byte aligner to detect and align the control (k28.5) character in the training sequence. Once the transceiver detects this character, the count starts at every k28.5 that is received (basically, counting every training sequence). Once 64 error-free training sequences have been received, the IP core reports linkup. Any errors (for example, disparity or 8B/10B errors) that are received reset the count, and the IP core continues to wait until 64 error-free training patterns are received.



The self-synchronizing LSM also locks onto the clock compensation sequence. As turning on the **Clock Compensation** option allows the receiver to automatically relock if the link goes down, 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.

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 following polynomial equation:

$$G(x) = X^{16} + X^5 + X^4 + X^3 + 1$$

The transmitted bits are XORED with the output of the LFSR in the data stream. At the receiver, the data stream is again XORED 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 XORED 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 IP core to use a single shared transmitter and multiple receivers in the master device, as shown in Figure 3-8 and Figure 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

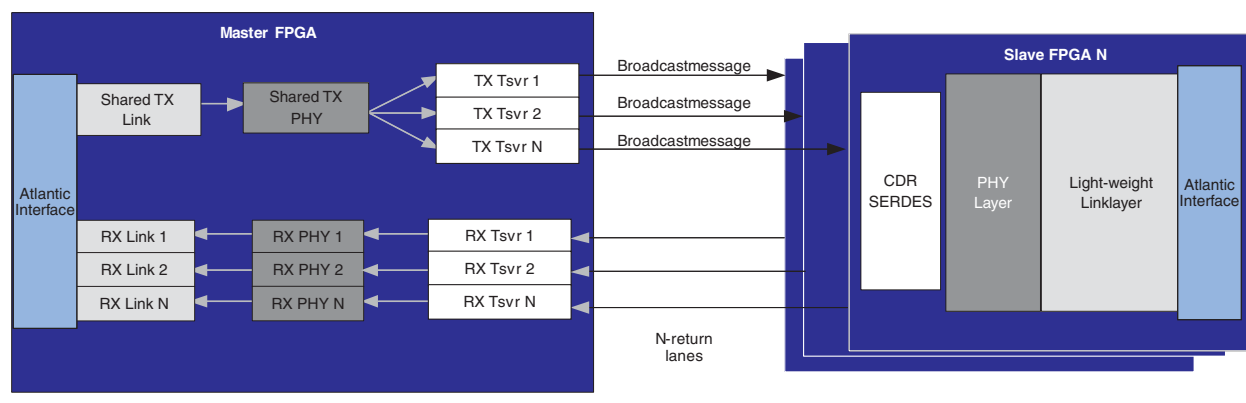
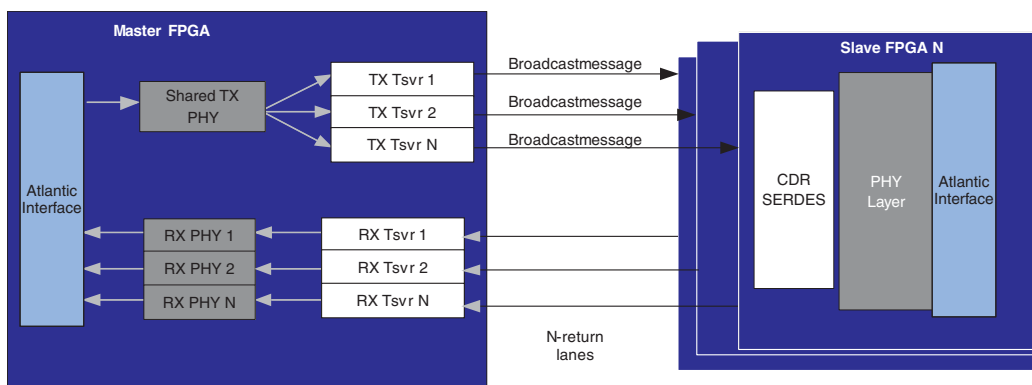


Figure 3–9. Streaming Broadcast Mode Block Diagram

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.



The **Clock Compensation** option is disabled if the value chosen for the **Reference Clock Frequency** option does not equal $\text{data rate} / (\text{transfer size} * 10)$, and the **Receiver Only** port type option is turned on.

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.

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. The following example shows a possible four-lane system, where Serial Lite II can reverse the four-lane system:

Example 3–1. SerialLite II Lane Reversal

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

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 IP core 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 IP core 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 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 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 IP core 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 IP core 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 2ⁿ increments, and the default value is 256 bytes.

Retry-on-error

The SerialLite II IP core 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 you turn on the **Retry-on-error** parameter, the transmitter retransmits all segments starting from the segment with errors. (If you turn off the **Retry-on-error** parameter, 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 seven 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 SerialLite II parameter editor. 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 SerialLite II parameter editor, and it is good practice not to set the time-out 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 time-out 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 3-4 shows the retry-on-error options for the priority data port.

Table 3-4. 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 3-5 summarizes the response to various transmission errors.

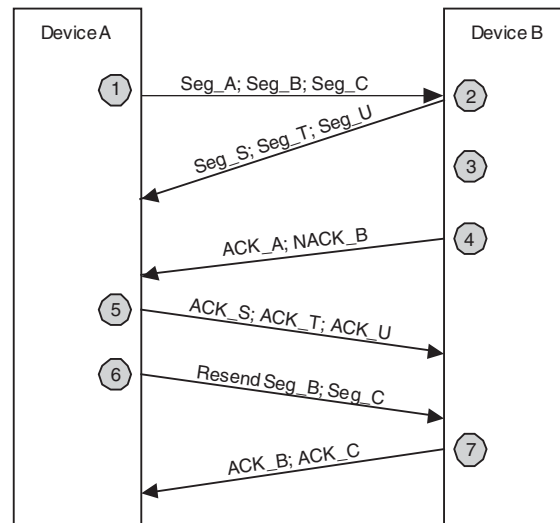
Table 3-5. Retry-on-error Responses

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 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 IP core 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 slower than the system logic on the transmitting end of the link is sending data.

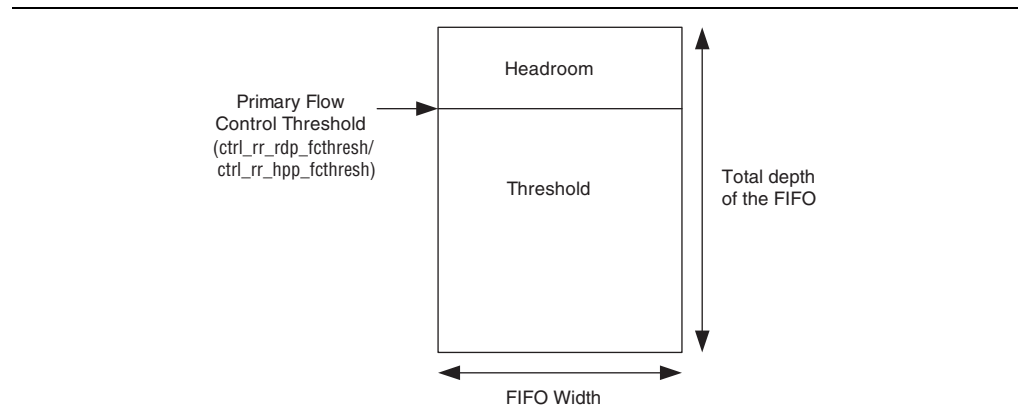
The flow control feature in the SerialLite II IP core operates by having the receiving end of the link issue a PAUSE instruction to the transmitting end of the link when threshold of the receiver's FIFO buffer is breached. The instruction causes the transmitter to cease transmission for specified pause 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)



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 SerialLite II parameter editor 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 3–18 for further analysis.

The total depth of the FIFO (in bytes) is derived by the SerialLite II parameter editor using the following formula :

$$\text{Total Depth} = \text{FIFO SIZE} / (\text{TSIZE} * \text{RX_NUMBER_LANES})$$

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 1,024B, and a TSIZE of 2 in a four-lane SerialLite II configuration, you compute the Total Depth as follows:

$$\text{Total Depth} = 1024 / 2 * 4 = 128$$

Based on the above result, for this example, you must set the **Threshold** value in the SerialLite II parameter editor 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 IP core 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 3-17](#)” 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.

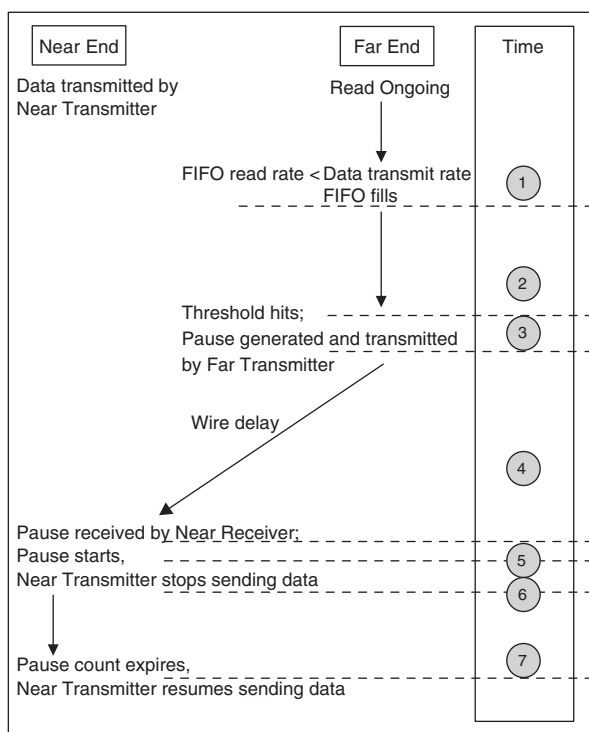
- This refresh time must be set so that the renewed flow control packets are received by the near transmitter before the current pause time completes. Set the value of **Refresh period** to be smaller than **Pause quantum time** in the **Priority Packet Settings** or **Data Packet Settings** section 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 3-17 shows an example of a flow control operation.

Figure 3-12. Typical Flow Diagram of FC Operation



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 expires, the pause stops and the near transmitter continues sending the data (assuming that no other pause requests have been received).

Selecting the Proper Threshold Value

Table 3-6 on page 3-18 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 latency values in [Table 3-6](#) by the tx_coreclock clock period.

Table 3-6. 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: 41 Priority data: 41 + seg_TX (3) + seg_RX (3)

Notes to Table 3-6:

- (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 4-6
- (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.

$$\text{seg_TX} = \lceil \text{segment size} / (\text{T_SIZE} * \text{TX_NUMBER_LANES}) \rceil$$

$$\text{seg_RX} = \lceil \text{segment size} / (\text{T_SIZE} * \text{RX_NUMBER_LANES}) \rceil$$
The **Segment size** value is specified by users in the **Parameters Settings** tab on the **Link Layer** page.

The **Threshold** parameter must be set to a value such that the FIFO does not completely empty during a flow control operation (this can cause inefficiencies in the system), and leave enough room in the FIFO to ensure any remaining data in the system can be safely stored in the FIFO without the FIFO overflowing.

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

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 8 to 2,040 columns. In elements, this value is $8/\text{TSIZE}$ to $2,040/\text{TSIZE}$ 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. Part of determining the pause duration is the read rate of the RX FIFO.

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 $\text{TSIZE}-2$, 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.

Selecting the Proper Refresh Value

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.

The `stat_tc_rdp_thresh_breach`, `stat_tc_hpp_thresh_breach`, `stat_fc_hpp_retransmit` and `stat_tc_fc_hpp_retransmit` status signals indicate whether the refresh period is set appropriately. If `stat_tc_rdp_thresh_breach` or `stat_tc_hpp_thresh_breach` (which indicates that the RX FIFO is still breached) is still asserted after the FC refresh period (based on the value set), the far transmitter generates another flow control packet (based on the value set at the **Pause Quantum Time** option) and sends it out, causing the `stat_fc_hpp_retransmit` or `stat_tc_fc_hpp_retransmit` to be asserted.

External Flow Control (When RX FIFO Size is 0)

The SerialLite II IP core 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 IP core automatically sets the width of the receive FIFO buffers at TSIZE bytes per lane. You specify the buffer size in the SerialLite II parameter editor.

FIFO Buffer Size

The size of the FIFO buffers is based on the factors listed in [Table 3-7 on page 3-20](#).

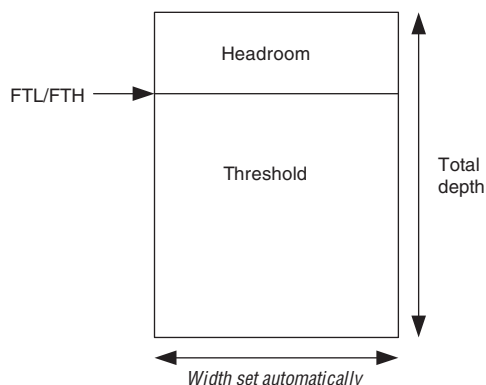
Table 3-7. Factors Affecting Receive FIFO Buffer Size

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.
Number of packets (and packet sizes)	If you want to use a store-and-forward FIFO (using the <code>eop_dav</code> 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 (FTL), `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.



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

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_rxhdp_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 threshold, the `dav` signal remains high until the end-of-packet is read out of the FIFO buffer. 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.

`ctl_rxhdp_fth` and `ctl_rxrdp_fth` are the threshold levels for the high priority and regular data ports on the receiver Atlantic FIFO buffers. When the data fill level is higher than the threshold level set by `ctl_rxhdp_fth` or `ctl_rxrdp_fth`, or `dav = 1`, it means that there is a large amount of data ready to be fetched at the FIFO buffer. You must set these threshold levels based on your design requirements, and ensure that the FIFO buffer does not underflow. You may also set the threshold levels to segment size of a priority packet; or to the lowest level so that you can fetch data as soon as it is stored in the FIFO buffer.

You can set `ctl_rxhdp_ftl` to 1 element unit so that it fetches the data from the RX FIFO buffer as soon as there is data available. If you want to store some data before fetching it, you can raise the threshold level.

The FIFO buffer threshold high (`ctl_txrdp_fth/ctl_txhdp_fth`) value for transmitter variations controls when the `txrdp_dav/txhdp_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/txhdp_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/txhdp_dav` signal is low, indicating that the user should stop writing data into the FIFO.

`ctl_txhdp_fth` and `ctl_txrdp_fth` are the threshold levels for the high priority and regular data ports on the transmitter Atlantic FIFO buffers. When the data fill level at the FIFO buffer is lower than the threshold level set by `ctl_txhdp_fth` or `ctl_txrdp_fth`, or `dav = 1`, it means that there are plenty of spaces available for data to write into the buffer. You must set these threshold levels high so that the user logic knows whenever the FIFO buffer has available spaces for data buffering and to ensure that overflow does not occur. However, these threshold settings should not exceed the FIFO depth.

For example, if the transmitter buffer size is 4,096 bytes, and the transmitter FIFO depth is 2,048 element units, you should set the level of `ctl_txhdp_fth` to 250 element units.

$TSIZE = 2$, and one FIFO element = 2 bytes

Maximum TX FIFO level (TX 8 lane) = $2,048 / 8 = 256$ element units



You can set any value below 256 element units for `ctl_txhdp_fth`; Altera recommends a level of 250 element units or 8'hFA.

The threshold levels on both the transmitter and receiver Atlantic FIFO buffers differ according to implementation. They may depend on the data traffic, the FIFO depth, and the clock frequencies for read and write. Based on your design, you can gauge the usual fill level of the FIFO buffers and determine the appropriate threshold levels.

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 IP core 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 following polynomial equation:

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

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

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

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.

Table 3–8 and Table 3–9 show the different CRC options.

Table 3–8. 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 3–9. 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 IP cores. The transceiver IP core 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.



If you select Arria V or Stratix V as the target device family, you are required to instantiate the Custom PHY IP core as the hard transceiver. For more information about this configuration, refer to [“Configuration for Arria V, Cyclone V, and Stratix V Devices”](#) on page 4-19.



For more information on Altera gigabit transceiver (ALTGX) IP core, refer to the [Arria II GX Transceiver Architecture](#) section in volume 2 of the *Arria II GX Device Handbook*, and the [Stratix IV Transceiver Architecture](#) section in volume 2 of the *Stratix IV Device Handbook*.

Voltage Output Differential (V_{OD}) Control Settings

The Stratix IV transceivers allow you to set the V_{OD} to handle different length, backplane, and receiver requirements. A range from 200 to 1,200 mV is supported for Stratix IV devices. Arria II GX devices have a fixed value, which cannot be changed. The range is decoded using the GUI integer value and the on-chip transmitter programmable termination values.

[Table 3-10](#) shows how the V_{OD} value you chose in the SerialLite II parameter editor corresponds to the mV value. The V_{OD} value is 0 by default.

Table 3-10. V_{OD} Control Settings

V_{OD} Value (Per Lane)	100 Ohms (mV) Stratix IV
0	200
1	400
2	600
3	700
4	800
5	900
6	1,000
7	1,200

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

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 IV devices, the pre-emphasis control values supported are 0,1,2,3,4, and 5. For 0, **Pre-emphasis** option 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 II GX devices, the **Pre-emphasis** setting cannot be changed.

Transmitter Buffer Power (V_{CCH})

This setting is for information only and is used to calculate the V_{OD} from the buffer power supply and the transmitter termination to derive the proper V_{OD} range.

The selections available are 1.5 V for Arria II GX devices.

For Stratix IV devices, the Quartus II software automatically selects 1.4 V or 1.5 V. If you want to set your preferred V_{CCH} value to the transmit and receive pins, perform the following steps:

1. In the Quartus II window, on the Assignments menu, click **Assignment Editor**.
2. In the <<new>> row, in the **To** column, double-click and type rxin to set value for the receive pin.
3. Double-click in the **Assignments Name** column, and click **I/O Standard (Accepts wildcards/groups)**. The entry is set to **I/O Standard**.
4. Double-click in the **Value** column and click **1.4-V PCML** or **1.5-V PCML**.
5. In the new <<new>> row, repeat steps 2 to 4 to set the value for the transmit pin (txout).

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 IV 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 II GX devices, the equalization cannot be changed.

Bandwidth Mode

The transmitter and receiver PLLs in the ALTGX IP core 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 II GX devices.

Starting Channel number

The range for the dynamic reconfiguration starting channel number setting is 0 to 380 for Stratix IV GX devices. These ranges are in multiples of four because the dynamic reconfiguration interface is per transceiver block. The range 0 to 380 is the logical channel address, based purely on the number of possible transceiver instances. This parameter is not applicable for Arria II GX devices.

Instantiating a Transceiver Reconfiguration Block

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

- Pre-emphasis
- Equalization
- V_{OD}
- Offset cancellation



For analog settings, there are no restrictions on using dynamic reconfiguration.

When you use a transceiver-based device, the ALTGX interface allows you to modify the parameter interface with a reconfiguration block. The `altgx_reconfig` block is not instantiated, but the parameter editor-generated wrapper provides the ports that interface to the `altgx_reconfig` block. If you choose to use an `altgx_reconfig` block, you must instantiate the `altgx_reconfig` block and connect the associated signals to the corresponding SerialLite II IP core top-level signals (tie the `reconfig_fromgxb`, `reconfig_clk`, and `reconfig_togxb` ports to the `altgx_reconfig` block).



You must instantiate the transceiver reconfiguration block on an Arria II GX or a Stratix IV device, because these device transceivers require offset cancellation. Your Arria II GX or Stratix IV design can compile without the dynamic reconfiguration block but it cannot function correctly in hardware.



For more information about the following topics, refer to the respective documents:

- Dynamic reconfiguration and offset cancellation for Arria II GX devices, refer to the *AN 558: Implementing Dynamic Reconfiguration in Arria II GX*.

- Dynamic reconfiguration and offset cancellation for Stratix IV devices, refer to the *Stratix IV Dynamic Reconfiguration* chapter in the *Stratix IV Device Handbook*.
- Dynamic reconfiguration and using the Altera Reconfiguration Controller for Arria V, Cyclone V, and Stratix V devices, refer to the *Altera Transceiver PHY IP Core User Guide*.

ALTGX Support Signals

This section describes the ALTGX support signals, which are only present on variants that use the Arria II GX and 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 <i>Arria II GX Device Handbook</i> or the <i>Stratix IV Device Handbook</i> .
reconfig_clk	I	The reconfig_clk input signal is the ALTGX dynamic reconfiguration clock. This signal must be connected as described in the <i>Arria II GX Device Handbook</i> or the <i>Stratix IV Device Handbook</i> if the ALTGX dynamic reconfiguration block is used. Otherwise, this signal must be set to 1'b0.
reconfig_togxb	I	The reconfig_togxb [N:0] input signal is driven from an external dynamic reconfiguration block. The signal supports the selection of multiple transceiver channels for dynamic reconfiguration. This signal must be connected as described in the <i>Arria II GX Device Handbook</i> or the <i>Stratix IV Device Handbook</i> if the external dynamic reconfiguration block is used. Otherwise, you must set this signal to 4'b0010 for Arria II GX and Stratix IV devices. N value is 3 for Arria II GX and Stratix IV devices.

Table 3–11. ALTGX Support Signals

Signal	I/O	Description
reconfig_fromgxb	0	<p>The reconfig_fromgxb output signal is driven to an external dynamic reconfiguration block. The width of this bus depends on the number of lanes (it may require multiple transceiver QUAD blocks), and the device family (for Arria II GX and Stratix IV, the bus is wider due to offset cancelation support).</p> <p>This signal identifies the transceiver channel whose settings are being transmitted to the dynamic reconfiguration. This signal must be connected as described in the <i>Arria II GX Device Handbook</i> or the <i>Stratix IV Device Handbook</i> if the external dynamic reconfiguration block is used. Otherwise, leave this signal unconnected.</p> <p>For Arria II GX and Stratix IV, you must use the dynamic reconfiguration block because they require offset cancelation.</p>
gxb_powerdown	1	<p>gxb_powerdown resets and powers down all circuits in the transceiver block. This signal does not affect the refclk buffers and reference clock lines.</p> <p>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.</p>

Error Handling

The SerialLite II IP core 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 IP core 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 (Part 1 of 2)

Error Type	Cause	Action
Catastrophic	<ul style="list-style-type: none"> ■ LSM cannot reverse polarity ■ LSM cannot reorder lanes 	SerialLite II enters nonrecoverable state
Link	<ul style="list-style-type: none"> ■ 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

Table 3–12. Error Summary (Part 2 of 2)

Error Type	Cause	Action
Data	<ul style="list-style-type: none"> 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) 	<p>Two possibilities:</p> <ul style="list-style-type: none"> 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 via the <code>txrdp_err</code> or <code>rxhpp_err</code> signals, 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 IP core 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* and the *Design Space Explorer* chapters 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 IP core 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 IP core 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.

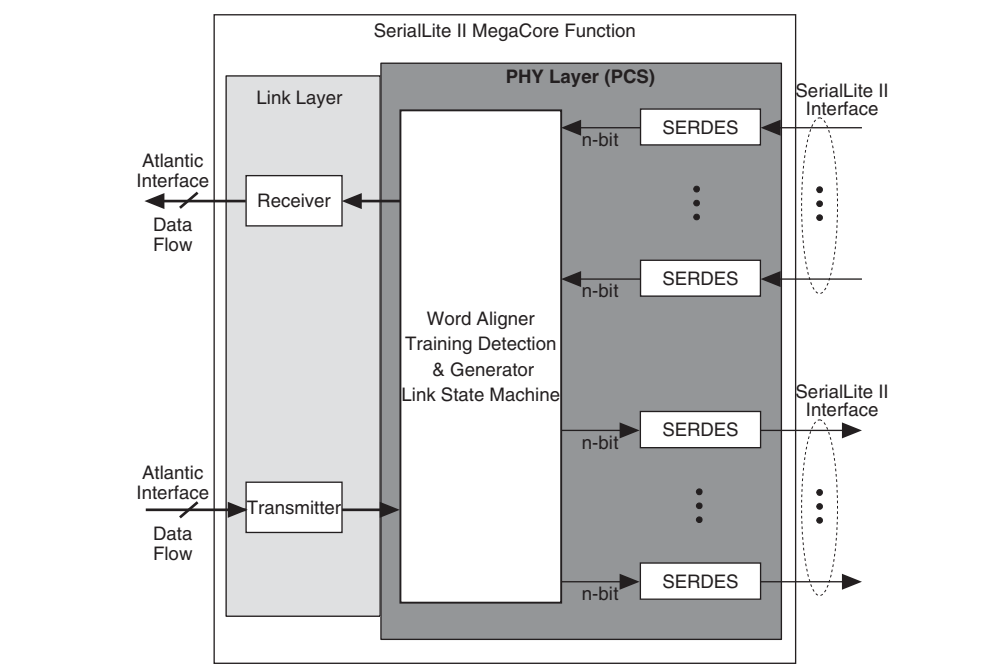
The following features have a significant impact on memory usage:

- 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 IP core contains a significant amount of logic, which is reduced to zero in streaming mode.

The SerialLite II IP core 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 IP core.

Refer to [Chapter 5, Testbench](#) for more information on the test bench.

Figure 4-1. SerialLite II IP Core Block Diagram



As shown in [Figure 4-1](#), the SerialLite II IP core 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 IP core 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 IP core. It is a full-duplex, synchronous point-to-point connection interface that supports a variety of data widths.

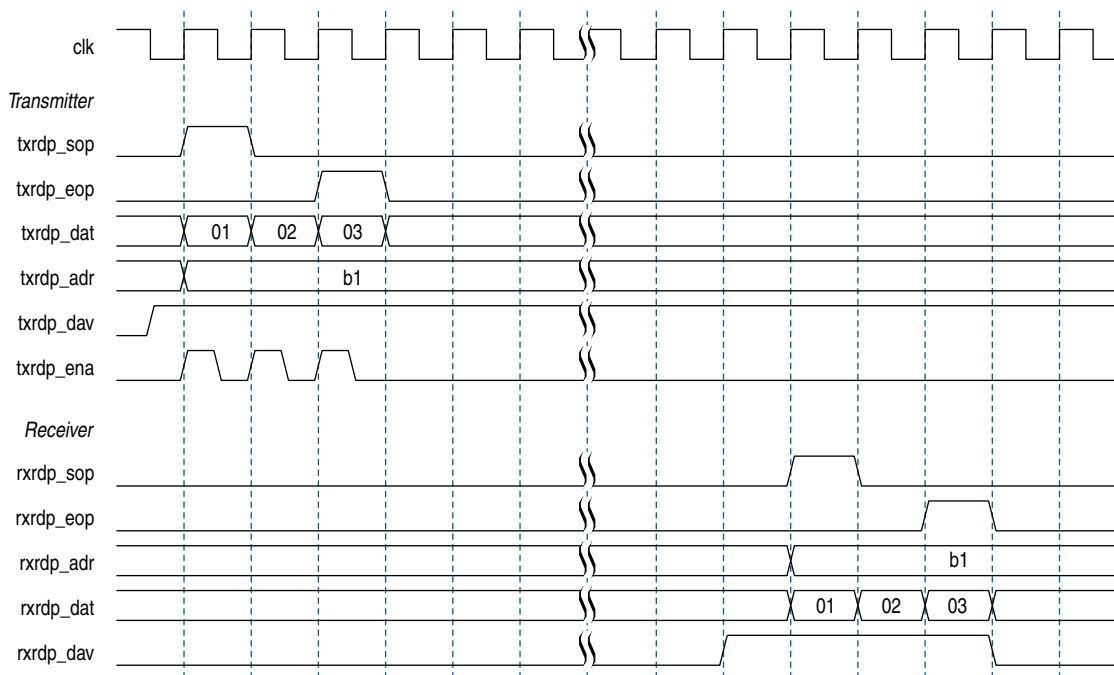
The width of the Atlantic interface is determined by the number of lanes and the transfer size.

The SerialLite II IP core 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 IP core 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 IP core is an Atlantic interface master. This user guide refers to the logic that drives data into the SerialLite II IP core or receives data from the SerialLite II IP core as the *system logic*.

Figure 4-2 shows how the data packets are transmitted and received through the Atlantic interface.

Figure 4-2. Transmitting and Receiving SerialLite II Data Packets



On the transmitter side, the user input data packets are sent to the Atlantic interface once the `txrdp_ena` signal is asserted (`txrdp_ena` pin is level triggered). The data packets go through several internal processes in the SerialLite II data link layer and physical layer, including all packet framing, CRC, and 8B/10B generation, and bit serializing. These internal processes produce some core latency of approximately 21 clock cycles to finally send the packets to the High Speed Serial Interface (HSSI) link. The latency calculation is based on the `tx_coreclock` frequency and is counted from the first data presented at the Atlantic interface on the transmitter side to the first data that appeared at the HSSI.

On the receiver side, the data packets are transmitted through the HSSI link and go through another SerialLite II IP core. In the other SerialLite II IP core, the same reverse processes are done in the SerialLite II data link layer and physical layer to strip off the framing and return the raw data back in the Atlantic interface. The data are presented at the Atlantic interface after approximately 25 clock cycles of latency. The latency is counted from the first data that appeared at the HSSI to the first data that reaches the Atlantic interface on the receiver side.

The Atlantic interface signals are described in [Table 4-7 on page 4-25](#).



However, these latencies are based on the simulations and parameters set in the testbench. The latencies vary depending on different designs and implementations, and the fill levels of the Atlantic FIFO buffer in designs where the fill levels are used.

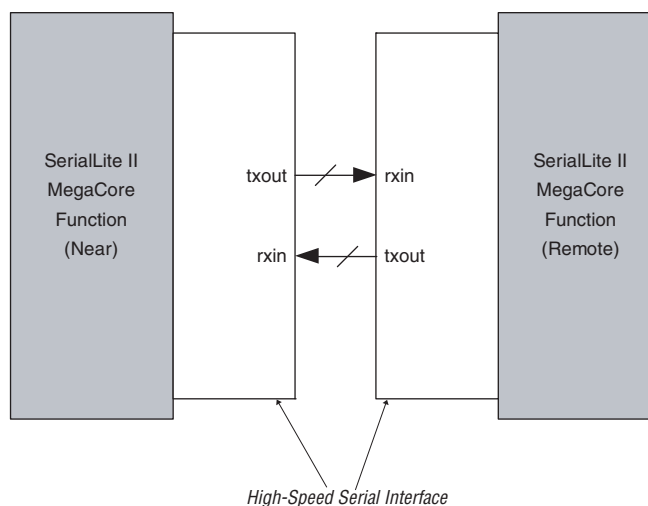


For more information on this interface, refer to the [FS 13: Atlantic Interface](#).

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 4-3](#).

Figure 4-3. High-Speed Serial Interface Connections



The high-speed serial interface signals are detailed in [Table 4-5 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 for `tx_coreclock` and `rrefclk`, use the following formula:

$$\text{Core clock frequency} = \text{Data Rate (Mbps)} / (\text{TSIZE} \times 10)$$

For example, if the data rate is 3,125 Mbps, and the TSIZE is 2, then:

$$\text{Core clock frequency} = 3,125 / (2 \times 10) = 156.25 \text{ MHz}$$

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 4 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 IP core is equal to the number of lanes \times 10 (due to 8B/10B encoding). Thus for 4 lanes, the data bus size is equal to 40 bits ($4 \times 10 = 40$).

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 $800 \text{ Mbps} \times 4 = 3.2 \text{ Gbps}$, the transfer is equal to the following:

- data bits: 256
- bits sent: 360
- $256 / 360 \times 3.2 = 2.276 \text{ Gbps}$

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

- data bits: 512
- bits sent: 680
- $512 / 680 \times 3.2 = 2.409 \text{ Gbps}$

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

- data bits: 1,024
- bits sent: 1,320

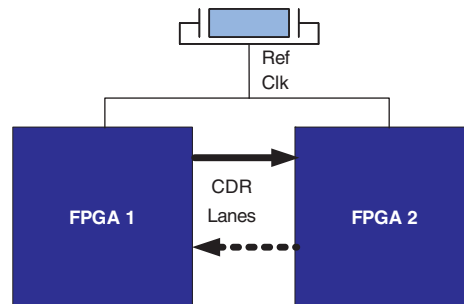
■ $1,024/1,320 \times 3.2 = 2.482 \text{ Gbps}$

External Clock Modes

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

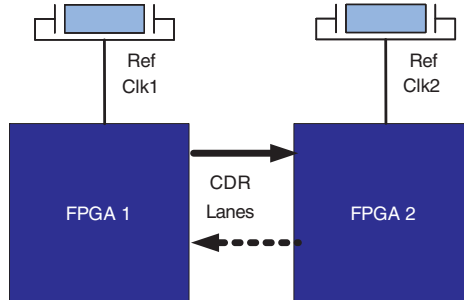
A synchronous configuration is typically used for a link where both ends are on the same board or on two boards driven by the same system clock (refer to [Figure 4-4](#)).

Figure 4-4. 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 (refer to [Figure 4-5](#)).

Figure 4-5. Asynchronous Mode



SerialLite II Internal Clocking Configurations

This section contains diagrams illustrating internal clocking configurations.

For Arria V, Cyclone V, and Stratix V configurations, you must identify the PLL reference clock frequency of the Custom PHY IP core and set the value accordingly in the `.sdc` file of the SerialLite II IP core for design integration between both cores.

When you generate a custom IP core, a Tcl script, named `<variation name>_constraints.tcl`, is generated that contains the PPM clock group settings 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 the PPM clock group assignments manually. You can type the assignments in [Example 4-1](#) directly into the Tcl console window.

Example 4-1.

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 IP core 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.

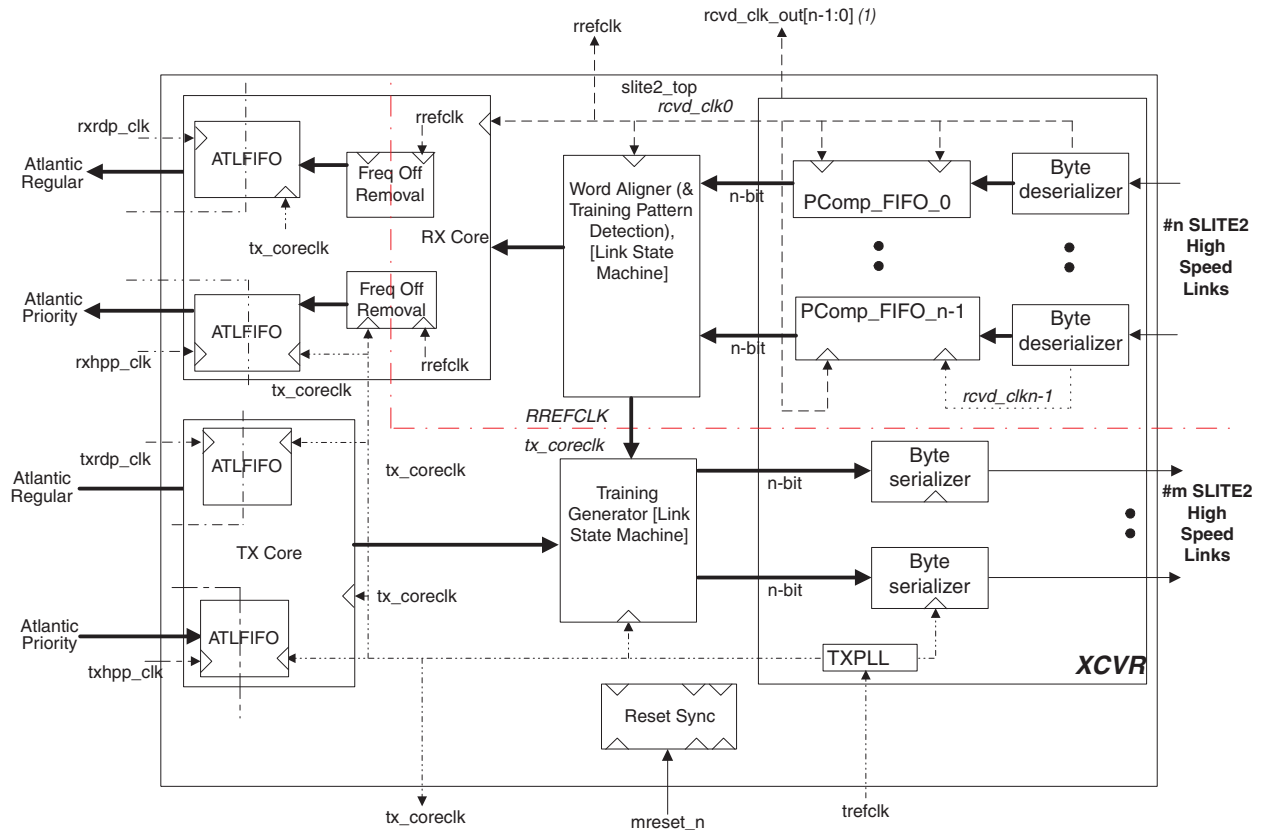
Table 4-1. SerialLite II Deskew Tolerance

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

SerialLite II Clocking Structure

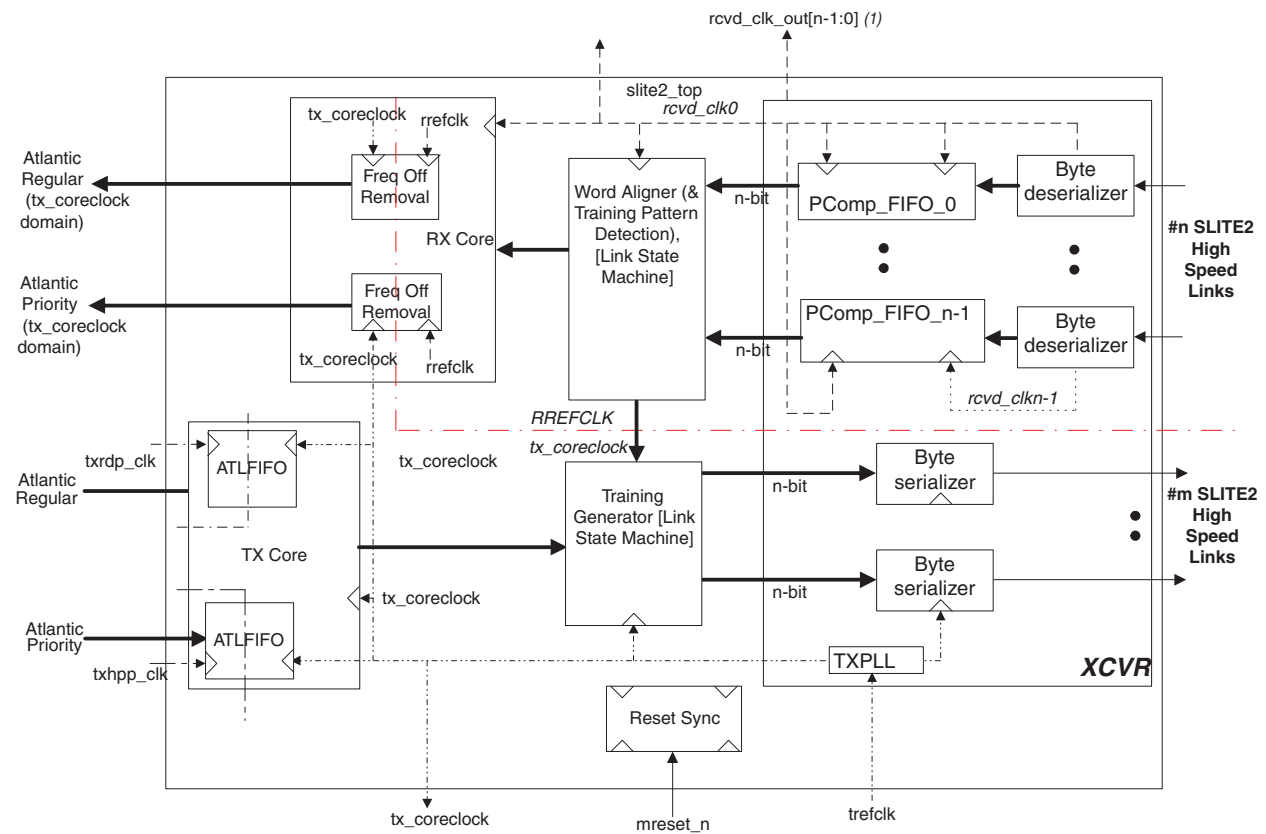
Figure 4-6 through Figure 4-12 show the IP core clock structures, which vary based on the configuration parameters.

Figure 4-6. Full-Featured Clock Structure



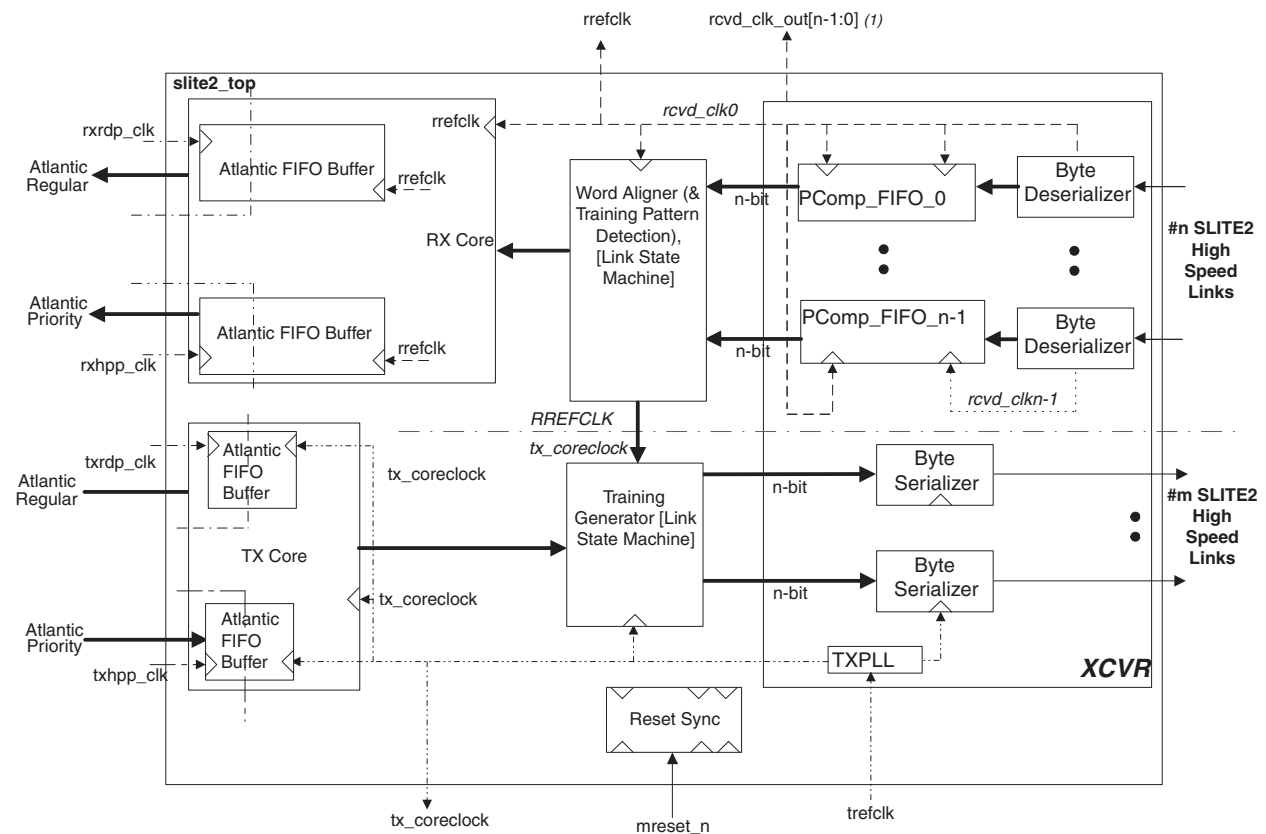
Note to Figure 4-6:

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

Figure 4-7. No Receiver FIFO Buffers Clock Structure**Note to Figure 4-7:**

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

Figure 4-8. Full-Featured No Frequency Offset Clock Structure



Note to Figure 4-8:

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

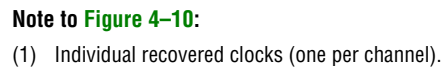
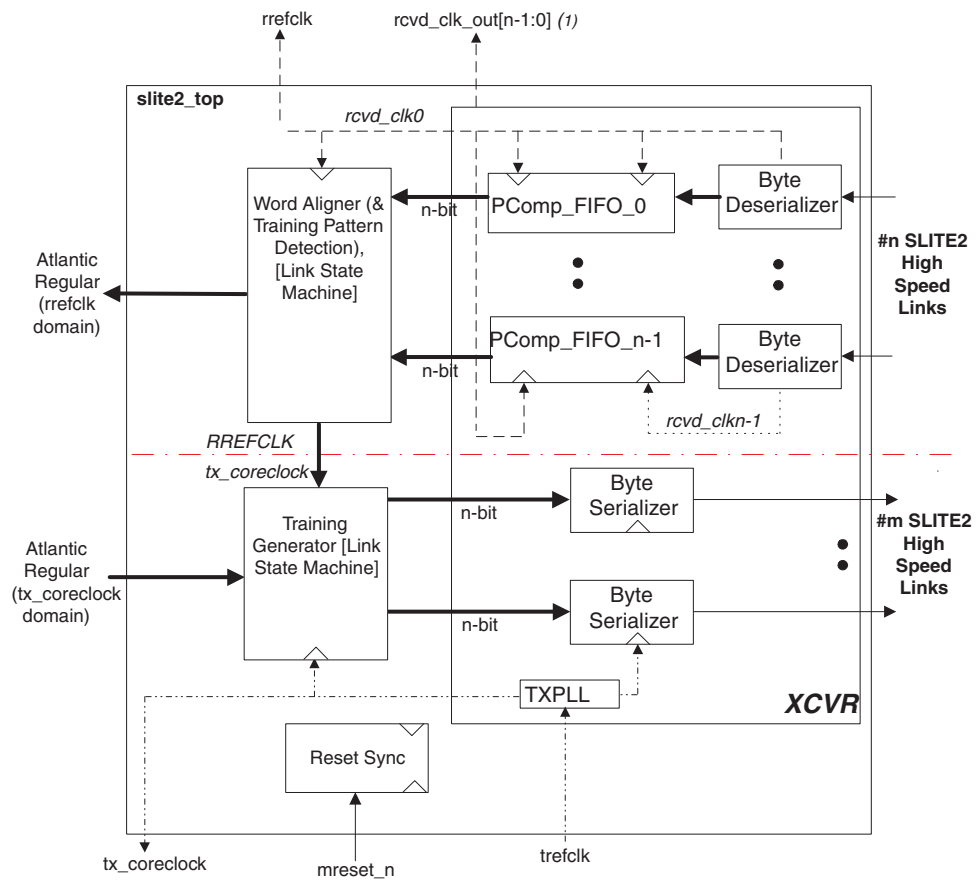
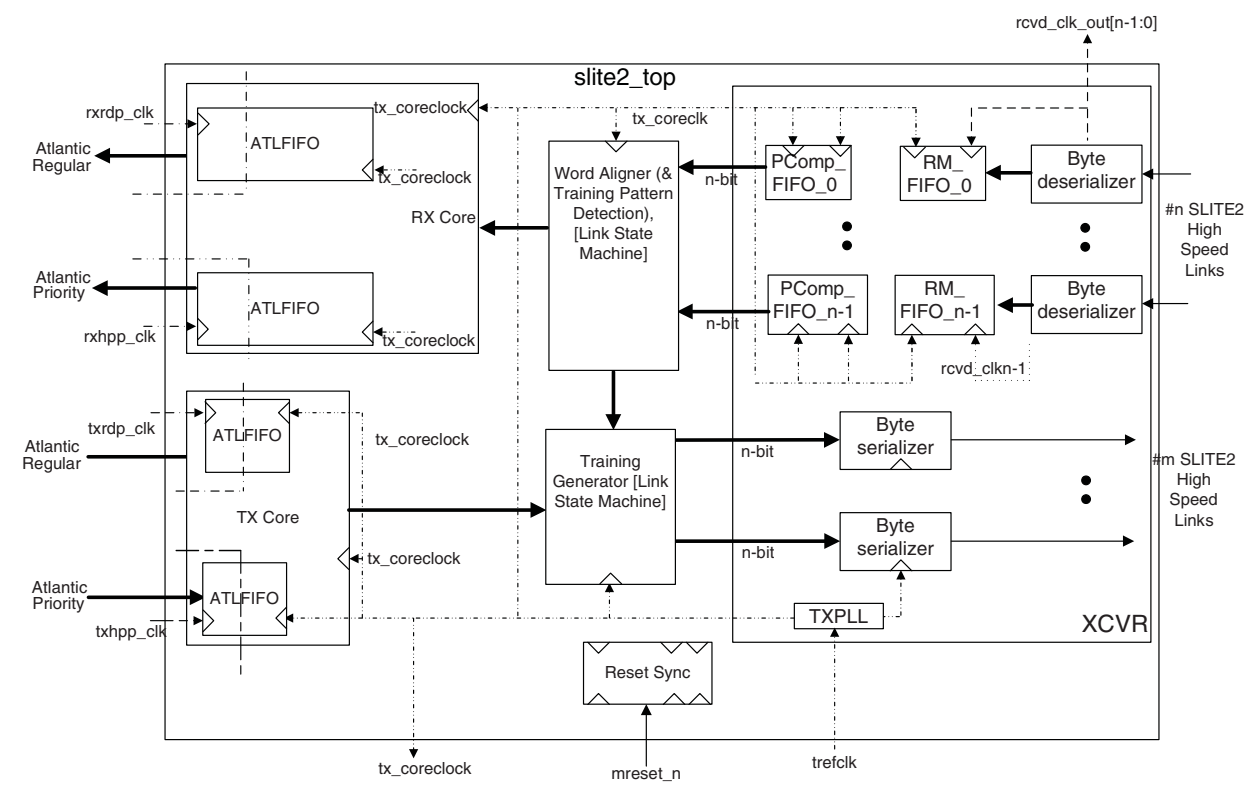


Figure 4–11. Streaming No Frequency Offset Clock Structure**Note to Figure 4–11:**

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

Figure 4-12. Full Featured Clock Structure for 5G Symmetrical With TSIZE = 2



Arria V, Cyclone V, and Stratix V Transceiver Clocking

For Arria V, Cyclone V, and Stratix V configurations, you must integrate the transceiver to the SerialLite II IP core manually.

When you configure the transceiver to work in more than 1 lane per SerialLite II instance, the `tx_clkout(0)` signal from the TX channel (PHY IP) must drive the SerialLite II input clock (`tx_coreclk`) and the input port (`tx_coreclk_in`) of all TX channels (PHY IP). Similarly, if your design requires more than 1 RX channel per SerialLite II instance, the `rx_clkout(0)` from the RX channel (PHY IP) must drive the SerialLite II input clock (`rx_coreclk`) and the input port (`rx_coreclk_in`) of all RX channels (PHY IP).

SerialLite II IP Core Pin-Out Diagrams

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

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

Your SerialLite II IP core 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:

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

Figure 4-13. Arria II GX/Stratix IV PHY Layer

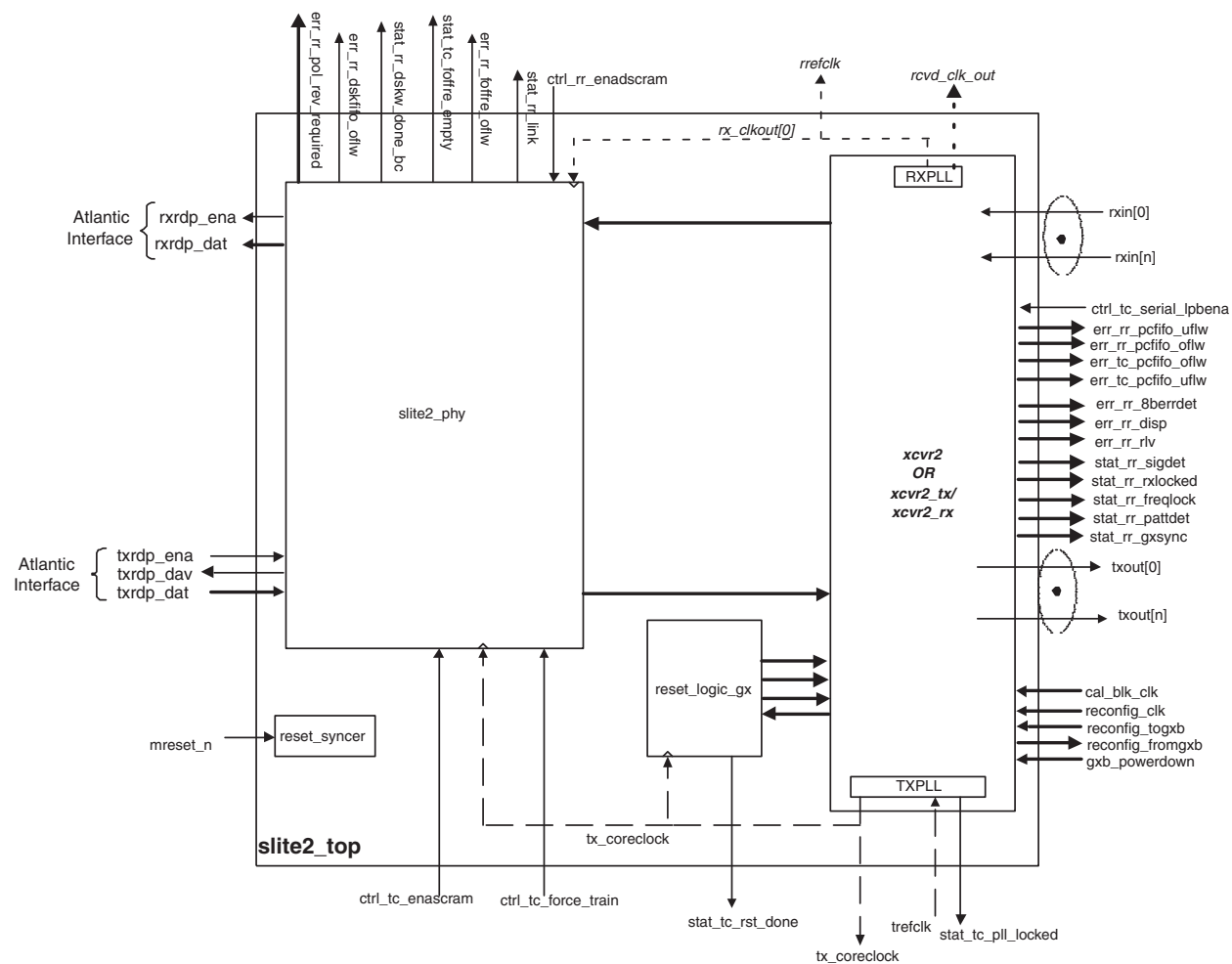
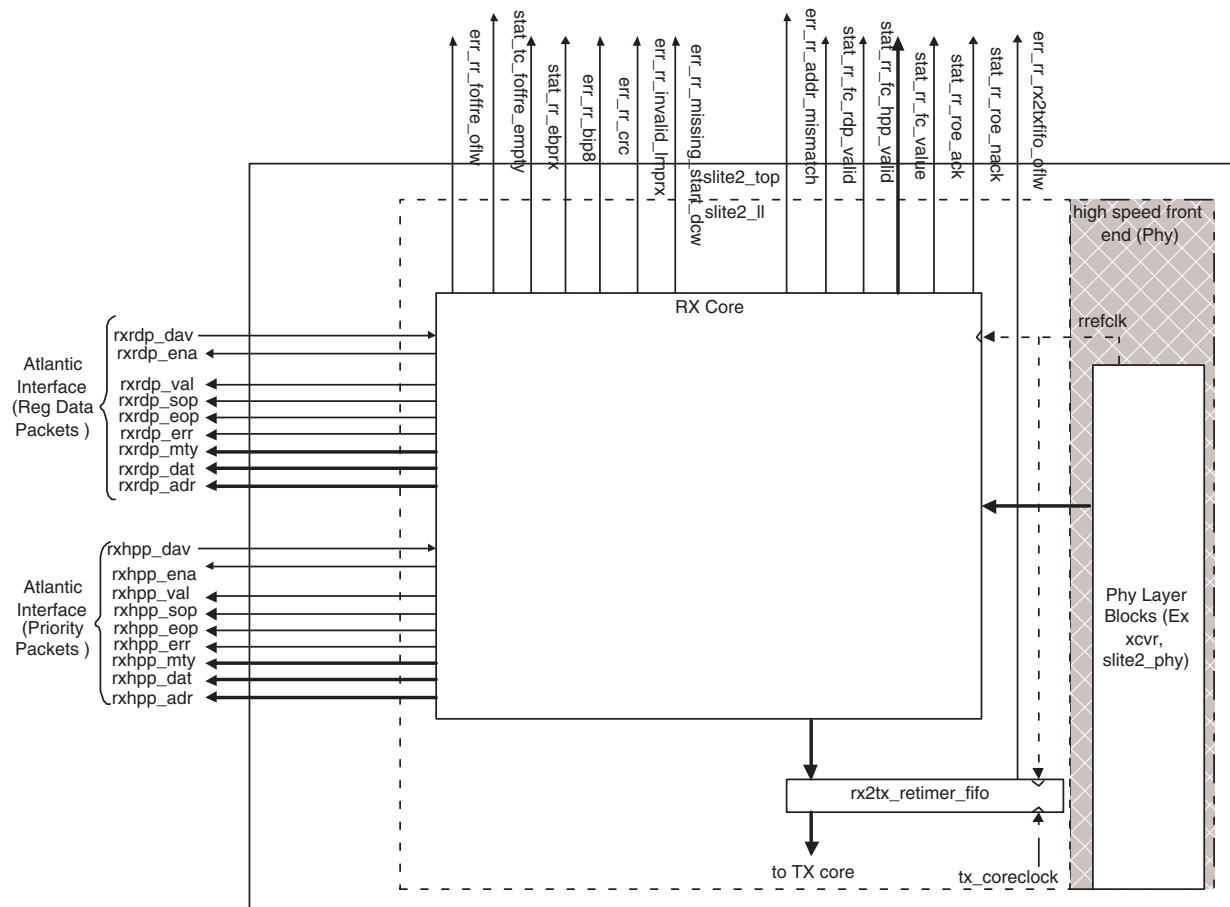


Figure 4-14. Receiver Layer With No FIFO



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.

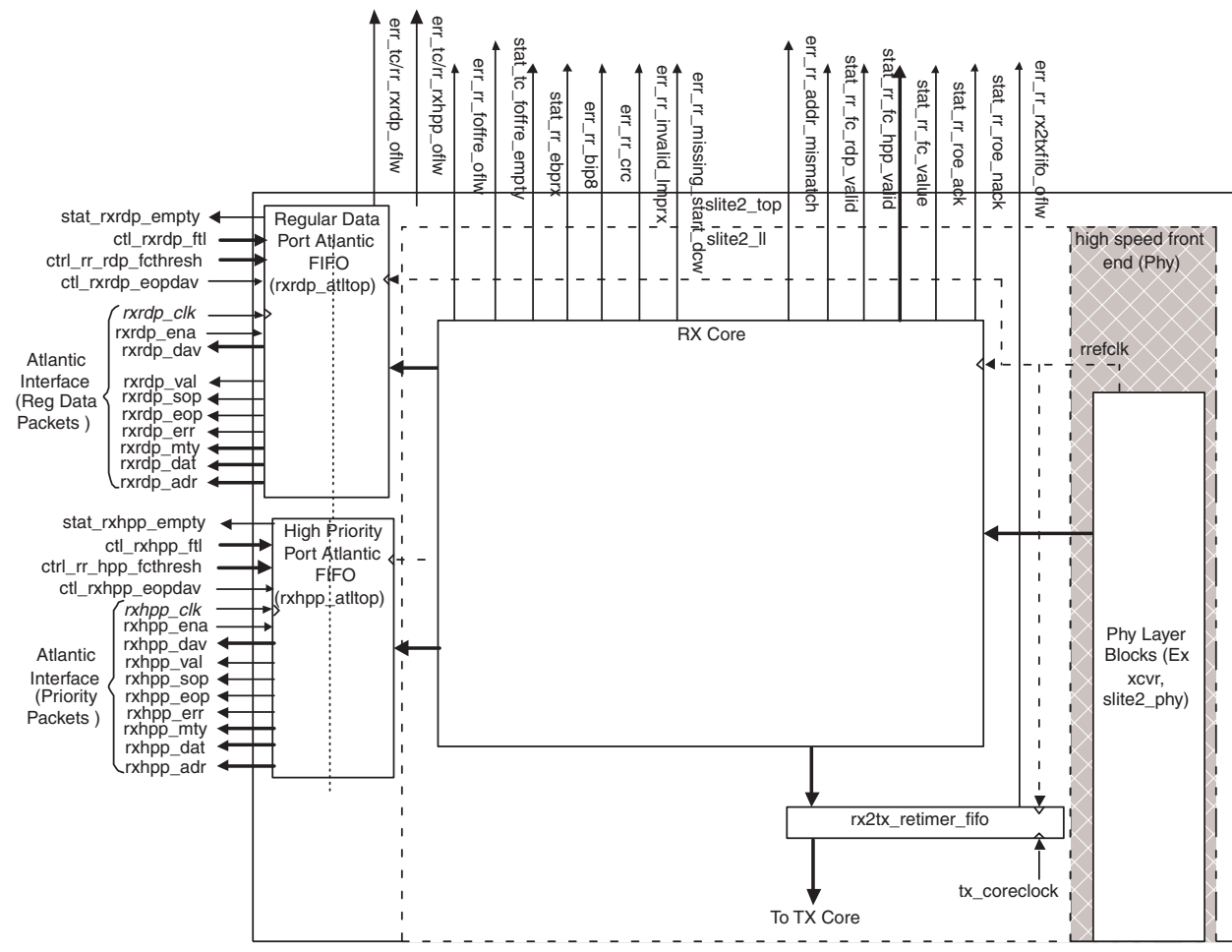
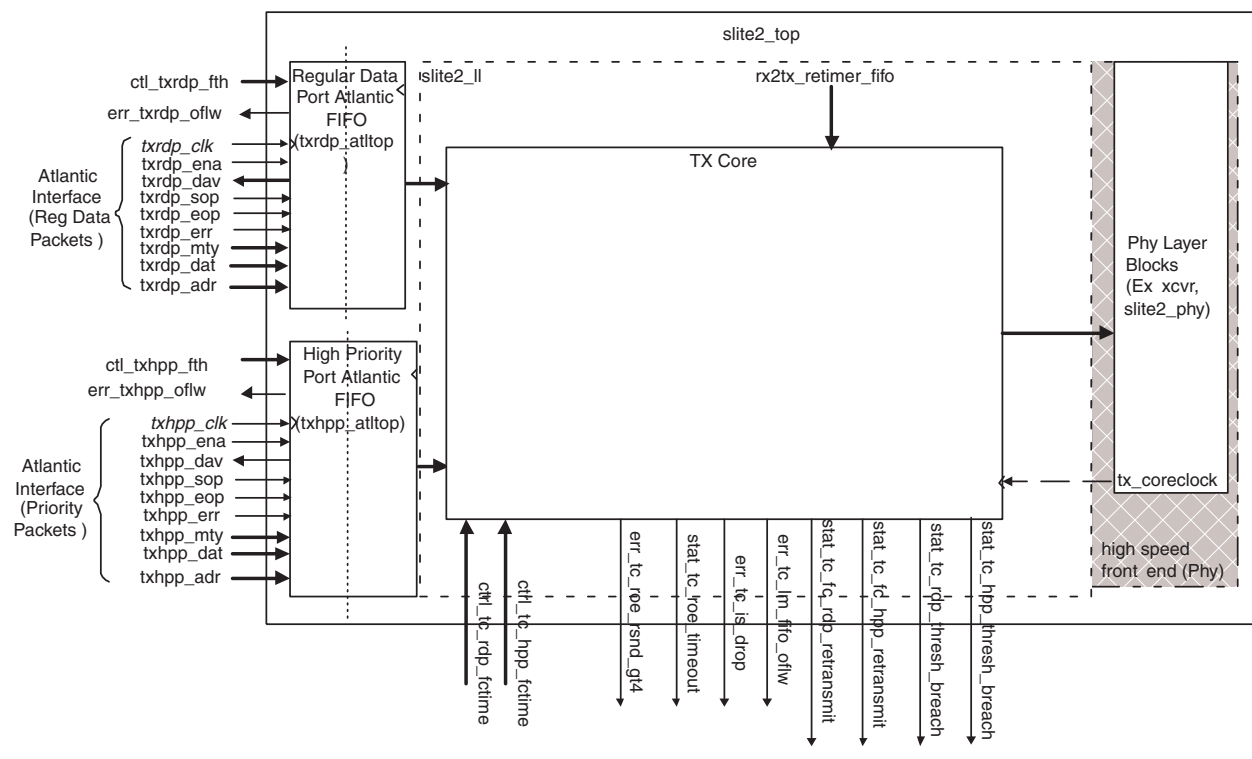
Figure 4-15. Receiver Link Layer With FIFO

Figure 4-16. Transmitter Link Layer



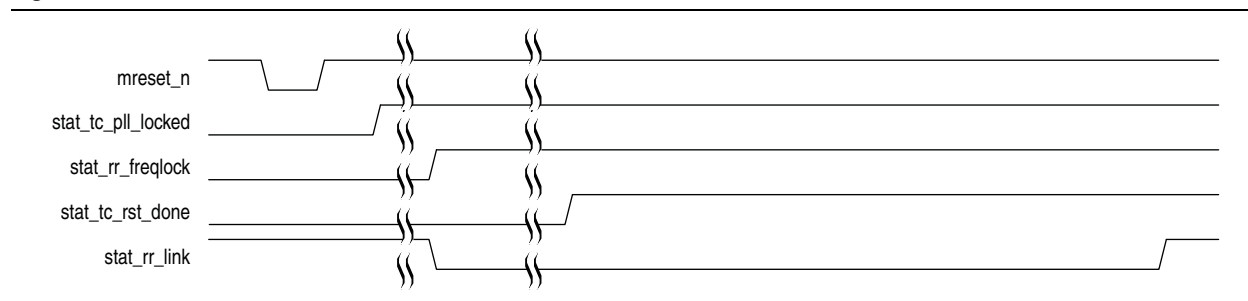
Initialization and Restart

Before the SerialLite II link can operate, the IP core must properly reset the GX transceiver. The SerialLite II IP core 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 reestablishes 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.

Figure 4-17 shows what happens when the SerialLite II IP core is initialized.

Figure 4-17. Initialization



When the `reset_n` input signal is asserted, the transceiver and the IP core start to reset and initialize the IP core. When the corresponding signals, `stat_tc_pll_locked`, `stat_rr_freqlock`, and the `stat_tc_rst_done` signal go high, a set of training sequence are transmitted across the link to align the characters and lanes. When everything is synchronized, the link is established and ready to be used, `stat_rr_link = 1`.

Multiple Core Configuration

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

- If you use the Tcl constraints to make assignments for the IP cores, you must edit the Tcl script associated with each generated SerialLite II IP core to update the hierarchical 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 II GX and Stratix IV designs, you must ensure that the `cal_blk_clk` input to each SerialLite II IP core is driven by the same calibration clock source. In addition, ensure that the SerialLite II IP core and other IP core variants in the system that use the ALTGX IP core have the same clock source connected to their respective `cal_blk_clk` ports.
- In Arria 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 IP core variants.

Configuration for Arria V, Cyclone V, and Stratix V Devices

The supported features for the SerialLite II IP core in Arria V, Cyclone V, and Stratix V devices are the same with the Stratix IV GX devices except for the hard transceiver features. Since there is no hard transceiver in this configuration, you need to instantiate the Custom PHY IP core and integrate both cores in your design. You can quickly locate the Custom PHY IP core in the IP Catalog.

Table 4-2 lists the Custom PHY IP core blocks and the respective data rate that the SerialLite II IP core utilizes for this configuration.

Table 4-2. Custom PHY IP Core Blocks and Data Rate Used by SerialLite II IP Core

FPGA Fabric Transceiver Interface Width	Blocks Enabled	Data Rate (Mbps) for Arria V GZ/ Arria V GX/ Stratix V	Data Rate (Mbps) for Cyclone V
32 (TSIZE = 4)	Word alignment mode: Manual ⁽¹⁾ /Automatic synchronization state machine ⁽²⁾	3,126 to 6,375	3,126 to 5,000
	Word alignment pattern: 10'h17c		
16 (TSIZE = 2)	8B/10B encoder/decoder	1,000 to 5,000	1,000 to 3,750
	Word alignment mode: Automatic synchronization state machine		
8 (TSIZE = 1)	Word alignment pattern: 10'h17c	622 to 2,500	622 to 1,875
	8B/10B encoder/decoder		

Notes to Table 4-2:

- (1) Assert the `rx_enapatternalign` register in Custom PHY through the Avalon-MM interface to trigger another alignment when synchronization is lost.
- (2) Applicable only for Arria V GZ and Stratix V devices.



For more information about the Custom PHY IP core, refer to the [Altera Transceiver PHY IP Core User Guide](#).

Design Consideration

When you instantiate the SerialLite II IP core and Custom PHY IP core, you must apply the following considerations to create a working design.

Compilation

If you use Tcl constraints to make assignments for the SerialLite II IP core, you must perform the following actions:

- Identify the Custom PHY IP core clock node
- Set the Custom PHY IP core reference clock frequency accordingly in the `.sdc` file for design integration between the SerialLite II IP core and Custom PHY IP core

Testbench

For the SISTER IP core instance, you are required to edit the SerialLite II IP core dynamically generated testbench to include the Custom PHY IP core instantiation. The testbench verifies whether the integration of both cores is functionally correct in the simulation.



The SISTER IP core is a SerialLite II IP core with parameters derived from the DUT parameters. For more information about the testbench, refer to “[Testbench Specifications](#)” on page 5–2.

Simulation Support

The Quartus II software generates the simgen netlist, which contains only the SerialLite II IP core soft logic. The hard transceiver instantiation logic is not included. You are required to add the Custom PHY IP core simulation files into the command line Tcl file (`<top level design name>_run_models.tcl`) to enable the simulation to work in the Modelsim simulator.



For more information about the compilation and simulation flow, refer to the design example for [SerialLite II implementation in Arria V and Stratix V devices](#).

Parameter Settings For SerialLite II and Custom PHY IP Cores

The parameters associated with the transceiver configuration ([Configure Transceiver](#) page) in the SerialLite II IP core are disabled since there is no hard transceiver in this configuration. Other parameters for the SerialLite II IP core remains the same and are enabled. Refer to “[Parameter Settings](#)” on page 3–1 for a more detailed description of the parameters.

The SerialLite II IP core requires specific features to be enabled on the Custom PHY IP core to support this configuration. [Table 4–3](#) list the options that you can set using the Custom PHY IP core parameter editor. Note that the required ports are essential for the Custom PHY IP core instantiation.

Table 4–3. Custom PHY IP Core Settings (Part 1 of 2)

Option	Description	Setting
pll_locked output port	Provides Tx PLL locking status in the Custom PHY IP core.	Optional
tx_ready output port	Indicates that the Custom PHY IP core is ready to transmit data.	Required
rx_ready output port	Indicates that the Custom PHY IP core is ready to receive data.	Required
Enable TX Bitflip	Provides control for bitflip functionality.	Off
Create rx_coreclkkin port	Provides transceiver clock output to the rx_coreclk signal in the SerialLite II IP core. For Arria V, Cyclone V, and Stratix V designs with more than 1 channel, connect transceiver PHY rx_clkout (0) to rx_coreclkkin (N-1:0).	Required
Create tx_coreclkkin port	Provides transceiver clock output to the tx_coreclk signal in the SerialLite II IP core. For Arria V, Cyclone V, and Stratix V designs with more than 1 channel, connect transceiver PHY tx_clkout (0) to tx_coreclkkin (N-1:0).	Required

Table 4-3. Custom PHY IP Core Settings (Part 2 of 2)

Option	Description	Setting
Create rx_recovered_clk port	Provides a recovered clock output for the transceiver.	Off
Create optional ports	Provide the following ports: <ul style="list-style-type: none"> ■ tx_forceelecidle ■ rx_is_lockedtohref ■ rx_is_lockedtodata ■ rx_signaldetect 	Optional
Avalon data interfaces	Enables support for Avalon-Streaming (ST) interface.	Optional
Enable embedded reset controller	Enables the controller to reset the transceiver.	Required
Create optional word aligner status ports	Provide the following word aligner status ports for the transceiver: <ul style="list-style-type: none"> ■ rx_syncstatus ■ rx_patterndetect 	Optional
Enable run length violation checking	Enables run length violation check to the err_rr_rlv signal in the SerialLite II IP core.	Required
Enable rate match FIFO	Enables support for rate match FIFO.	Optional
Create optional rate match FIFO status ports	Enable the status ports for rate match FIFO.	Optional
Enable 8B/10B encoder/decoder	Provide the following ports: <ul style="list-style-type: none"> ■ rx_runningdisp—provides running disparity status to the err_rr_disp signal in the SerialLite II core. ■ rx_dataok—indicates whether the rx_parallel_data output port contains data or control symbol. 	Required
Enable manual disparity control	Enables manual disparity control for the 8B/10B encoder/decoder.	Off
Create optional 8B/10B status ports	Provide the following status ports for the 8B/10B encoder/decoder operation: <ul style="list-style-type: none"> ■ rx_errdetect ■ rx_disperr 	Optional
Enable byte ordering block	Enables byte ordering pattern configuration.	Off
Enable byte ordering block manual control	Provides manual control for the byte ordering block.	Off
Allow PLL/CDR reconfiguration	Enables support for dynamic reconfiguration of Tx PLL and Rx CDR.	Off

 For more information about the Custom PHY IP core ports, refer to the [Altera Transceiver PHY IP Core User Guide](#).

Extra Signals Between SerialLite II and Custom PHY IP Cores

The SerialLite II IP core includes new signals to interface with the Custom PHY IP core for data communication.

Table 4–4 lists the new interface signals.

Table 4–4. New Interface Signals

Signal Name	Direction	Width	Description
rx_parallel_data_out	Input	(Datapath width) x (Number of receiver channels)	Data input from the hard receiver.
rx_coreclk	Input	1	Clock input from the hard receiver.
tx_parallel_data_in	Output	(Datapath width) x (Number of transmitter channels)	Data output for the hard transmitter.
tx_ctrlnable	Output	(Number of control bits) x (Number of transmitter channels)	Control signal to indicate the control word in tx_parallel_data_in signal.
tx_coreclk	Input	1	Clock input from the hard transmitter.
rx_ctrlldetect	Output	(Number of control bits) x (Number of receiver channels)	Control signal to indicate that control word is detected in the hard transceiver.
stat_rr_pattdet	Input	(Number of control bits) x (Number of receiver channels)	Pattern detect output for the hard transceiver.
err_rr_disp	Input	(Number of control bits) x (Number of receiver channels)	Disparity error output for the hard transceiver.
flip_polarity	Output	Number of receiver channels	Polarity inversion input for the hard transceiver.

Some transceiver signals are removed due to the exclusion of hard transceiver in this configuration. Refer to the next section for a more detailed description of the signals.

Signals

Table 4–5 through Table 4–10 show the SerialLite II IP core signals.



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

Table 4–5 shows the high-speed serial interface signals.

Table 4–5. High-Speed Serial Interface Signals (Part 1 of 2)

Signal	Direction	Clock Domain	Description
rxin[n-1] ^{(1), (4)}	Output	—	SerialLite II differential receive data bus. Bus carries packets, cells, or in-band control words.
txout[m-1] ^{(2), (4)}	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 ⁽⁴⁾	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. In Arria II GX and Stratix IV designs, the TX PLL output clock and the primary clock are used for the TX logic.

Table 4-5. High-Speed Serial Interface Signals (Part 2 of 2)

Signal	Direction	Clock Domain	Description
mreset_n	Input	Asynchronous	Master reset pin, active low. Asserting this signal causes the entire SerialLite II IP core, including the Atlantic FIFO buffers, to be reset. For Arria V, Cyclone V, and Stratix V designs, hold this signal asserted until the Custom PHY asserts the tx_ready and rx_ready output ports.
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 ALTGX PLL has locked to the trefclk.
stat_rr_link ⁽³⁾	Output	rrefclk	Link Status. When high, the link is enabled.

Notes to Table 4-5:

- (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.
- (4) This signal is removed in configurations targeted for Arria V, Cyclone V, and Stratix V devices due to the exclusion of hard transceivers.

Table 4-6 shows the transceiver IP core signals.



For more information on Altera gigabit transceiver (ALTGX) IP core, refer to the *Arria II GX Transceiver Architecture* section in volume 2 of the *Arria II GX Device Handbook*, and the *Stratix IV Transceiver Architecture* section in volume 2 of the *Stratix IV Device Handbook*.

Table 4-6. Transceiver IP core Signals (Part 1 of 3)

Signal ^{(1), (2)}	Direction	Clock Domain	Description
ctrl_tc_serial_lpbena ⁽⁵⁾	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 ⁽⁵⁾ [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 ⁽⁵⁾ [rxnl-1:0]	Output	rrefclk	Interface/phase compensation FIFO buffer underflow signal (Arria II GX and Stratix IV devices only).
err_rr_pcfifo_oflw ⁽⁵⁾ [rxnl-1:0]	Output	rrefclk	Interface/phase compensation FIFO buffer overflow signal (Arria II GX and Stratix IV devices only).
err_rr_rlv [rxnl-1:0]	Output	rrefclk	Run length violation status signal.

Table 4-6. Transceiver IP core Signals (Part 2 of 3)

Signal ⁽¹⁾ , ⁽²⁾	Direction	Clock Domain	Description
err_tc_pcfifo_uflw [txnl-1:0]	Output	tx_coreclock	Interface/phase compensation FIFO buffer underflow signal (Arria II GX and Stratix IV devices only).
err_tc_pcfifo_oflw [txnl-1:0]	Output	tx_coreclock	Interface/phase compensation FIFO buffer overflow signal (Arria II GX and Stratix IV devices only).
stat_rr_sigdet [rxnl-1:0]	Output	rrefclk	This signal is for debugging purposes only and can be ignored.
stat_rr_gxsync ⁽⁵⁾ [srx-1:0]	Output	rrefclk	Gives the status of the pattern detector and word aligner.
stat_rr_rxlocked ⁽⁵⁾ [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 high for Arria II GX and Stratix IV 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 ⁽³⁾ , ⁽⁵⁾ Arria II GX or Stratix IV GX: [recon_quad*17-1:0]	Output	reconfig_clk ⁽⁴⁾	ALTGX Reconfig from the GXB Bus. This signal is connected to the reconfig_fromgxb port on the altgx_reconfig module. If you use Arria II GX or Stratix IV device, you must connect this output to the altgx_reconfig module for offset cancelation.
reconfig_togxb ⁽⁵⁾ Arria II GX or Stratix IV GX: [3:0]	Input	reconfig_clk	ALTGX Reconfig to the GXB Bus. This signal is connected to the reconfig_togxb port on the altgx_reconfig module. If you use Arria II GX or Stratix IV device, you must connect this output to the altgx_reconfig module for offset cancelation.
reconfig_clk	Input	—	ALTGX Reconfig Clock to the GXB. This signal is connected to the reconfig_clk port on the altgx_reconfig module. If you use Arria II GX or Stratix IV device, you must connect this output to the altgx_reconfig module for offset cancelation.
cal_blk_clk ⁽⁵⁾	Input	—	Calibration clock for the termination resistor calibration block. The frequency range of cal_blk_clk is 10 to 125 MHz.

Table 4-6. Transceiver IP core Signals (Part 3 of 3)

Signal ^{(1), (2)}	Direction	Clock Domain	Description
gxb_powerdown ⁽⁵⁾	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 4-6:

- (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 altgx_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 altgx_reconfig block is used, this signal is on the reconfig_clk domain.
- (5) This signal is removed in configurations targeted for Arria V and Stratix V devices due to the exclusion of hard transceivers.

Table 4-7 on page 4-25 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.



There are no specific requirements for Atlantic clocks (rxrdp_clk, rxhpp_clk, txrdp_clk and txhpp_clk) as they are all system dependent. The Atlantic clocks at the read side must be fast enough to prevent backpressure which decreases bandwidth efficiency.

Table 4-7. 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.
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.

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

Signal	Direction	Clock Domain	Description
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), (2)}	Output	rxrdp_clk	Number of empty bytes in the data word.
rxrdp_dat[d-1:0] ^{(1), (3)}	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.
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), (2)}	Output	rxhpp_clk	Number of empty bytes in the data word.
rxhpp_dat[d-1:0] ^{(1), (3)}	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.

Table 4-7. Atlantic Interface Signals (Part 3 of 3)

Signal	Direction	Clock Domain	Description
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 4-7:

- (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.
- (2) m is the empty value, which is log2 (data width).
- (3) d is the data width, which is 8 × transfer size × the RX number of lanes.
- (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 4-8 shows the Atlantic interface signals for streaming mode.

Table 4-8. 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 the 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.

Notes to Table 4-8:

- (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 4-9 shows the protocol processor's error, status, and control signals.

Table 4-9. Protocol Processor's Error, Status and Control Signals (Part 1 of 2)

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).

Table 4–9. Protocol Processor's Error, Status and Control Signals (Part 2 of 2)

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

Notes to Table 4–9:

- (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 4–10 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 4–10 are only available in certain configurations.

Table 4–10. Troubleshooting Signals (Part 1 of 2)

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	Indicates that frequency offset tolerance FIFO buffer has overflowed. The link restarts.
stat_tc_foffre_empty ⁽¹⁾	Output	tx_coreclock	Indicates that 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	Indicates that an end of bad packet character was received.
err_rr_bip8 ⁽¹⁾	Output	rrefclk	Indicates that a BIP-8 error was detected in the received link management packet.
err_rr_crc ⁽¹⁾	Output	rrefclk	Indicates that a CRC error was detected in the received segment/packet.
err_rr_fcrx_bne ⁽¹⁾	Output	rrefclk	Indicates that a flow control link management packet was received, but flow control is not enabled.
err_rr_roerx_bne ⁽¹⁾	Output	rrefclk	Indicates that a retry-on-error link management packet was received, but Retry-on-error parameter is not enabled.
err_rr_invalid_lmprx ⁽¹⁾	Output	rrefclk	Indicates that an invalid link management packet was received.
err_rr_missing_start_dcw ⁽¹⁾	Output	rrefclk	Indicates that data byte(s) received, but a start of data control word (DCW) is missing.
err_addr_mismatch ⁽¹⁾	Output	rrefclk	Indicates that 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	May indicate catastrophic error. Polarity on the input ALTGXB lines is reversed; the IP core cannot operate. If you see the signal for the first time, you should manually reset the core. If the signal triggers again after you reset, then it confirms a catastrophic error.
err_rr_dskfifo_oflw ⁽¹⁾	Output	rrefclk	Indicates that deskew FIFO buffer has overflowed. Link restarts.
stat_rr_dskw_done_bc ⁽¹⁾	Output	rrefclk	Indicates that a bad column was received after successful deskew completion. Link is restarted.

Table 4-10. Troubleshooting Signals (Part 2 of 2)

Signal	Direction	Clock Domain	Description
stat_tc_rdp_thresh_breach ⁽¹⁾	Output	tx_coreclock	Indicates that the receiver regular data port FIFO buffer is breached, transmit flow control link management packet.
stat_tc_hpp_thresh_breach ⁽¹⁾	Output	tx_coreclock	Indicates that the receiver priority data port FIFO buffer is breached, transmit flow control link management packet.
err_tc_roe_rsnd_gt4	Output	tx_coreclock	Indicates that the transmitter has transmitted a segment four times without receiving an ACK for that segment. The link is restarted.
stat_tc_roe_timeout	Output	tx_coreclock	Retry-on-error only: Indicates that the transmitter IP core has timed out waiting for ACK for a packet. The IP core sends that packet again.
stat_tc_fc_rdp_retransmit	Output	tx_coreclock	Indicates that 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	Indicates that 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	Indicates that irregular segment received (segment size boundary violation).
err_tc_lm_fifo_oflw	Output	tx_coreclock	Indicates that the link management FIFO buffer has overflowed. Link management packets are lost.
err_rr_rx2txfifo_oflw	Output	rrefclk	Indicates that the receiver to transmitter link management status information FIFO buffer has overflowed.
stat_rr_fc_rdp_valid	Output	rrefclk	Indicates that a flow control link management packet was received (regular data port).
stat_rr_fc_hpp_valid	Output	rrefclk	Indicates that a flow control link management packet was received (priority data port).
stat_rr_fc_value[7:0]	Output	rrefclk	Indicates that 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	Indicates that a retry-on-error link management packet of type ACK was received.
stat_rr_roe_nack	Output	rrefclk	Indicates that a retry-on-error link management packet of type NACK was received.

Note to Table 4-10:

- (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.

IP Core Verification

The SerialLite II IP core 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 IP core. These individual testbenches set unique parameters for each specific feature test.

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 IP core, 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 5-3 shows the block diagram of the SerialLite II testbench. The shaded blocks are provided with the SerialLite II testbench.



For Arria V and Stratix V configurations, you are required to edit the dynamically generated testbench to include the Custom PHY IP core instantiation. For more information about this configuration, refer to “Configuration for Arria V, Cyclone V, and Stratix V Devices” on page 4-19.

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 IP core.
- Simulates all basic SerialLite II transactions.

SerialLite II Testbench Files

The Verilog HDL demonstration testbench and associated scripts are generated when you create a IP core variation in the parameter editor, as described in “Generate Files” on page 2-6.

The VHDL demonstration testbench and the scripts to run it are generated when you create a IP core 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 IP core 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. Refer to [Figure 5-1 on page 5-3](#) 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 5-3](#) generates traffic through the Atlantic generators (`agen_dat_dut`, `agen_pri_dut`) and sends it through the SerialLite II IP core—the device under test (DUT). The SerialLite II interface of the DUT is connected to the SerialLite II interface of a second SerialLite II IP core—the SISTER. Data flows through the SISTER IP core and is received and checked on the Atlantic interface of the SISTER IP core (`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 IP core 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.

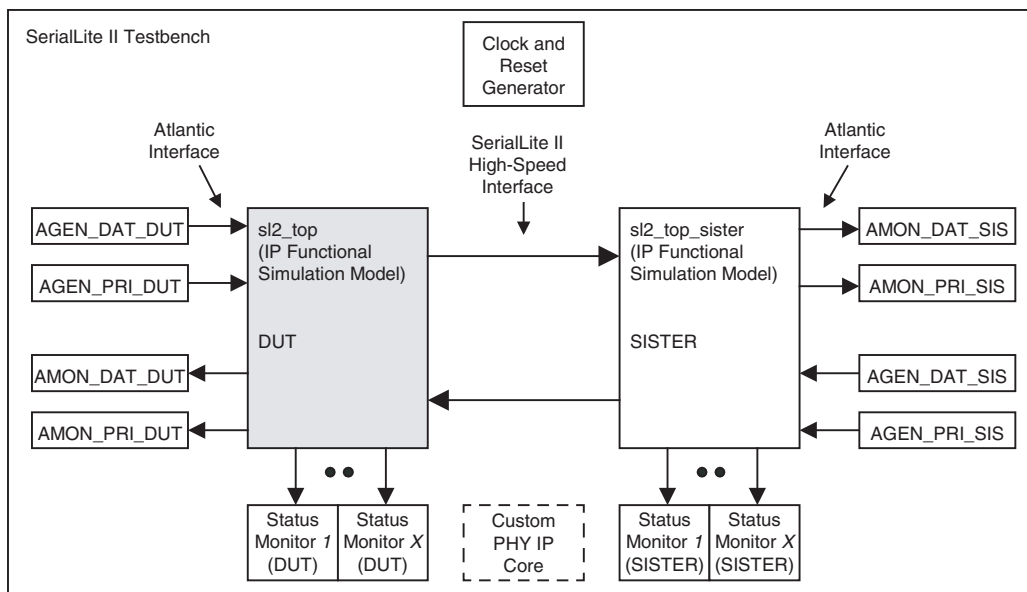
- The generated data follows a pseudo-random sequence (Verilog HDL) or incrementing data sequence (VHDL) that is checked by the Atlantic monitors.
- Each packet has an incrementing identifier (first byte in the packet) that is checked by the Atlantic monitor.

The SISTER IP core is a SerialLite II IP core 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).



The Custom PHY IP core is only applicable in configurations targeted for Arria V and Stratix V devices.

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



Notes to Figure 5–1:

- (1) The DUT and the SISTER IP cores 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 on page 5-4 shows the testbench environment for a SerialLite II single mode-transmitter only, non-broadcast IP core. The SISTER model contains a receiver.

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

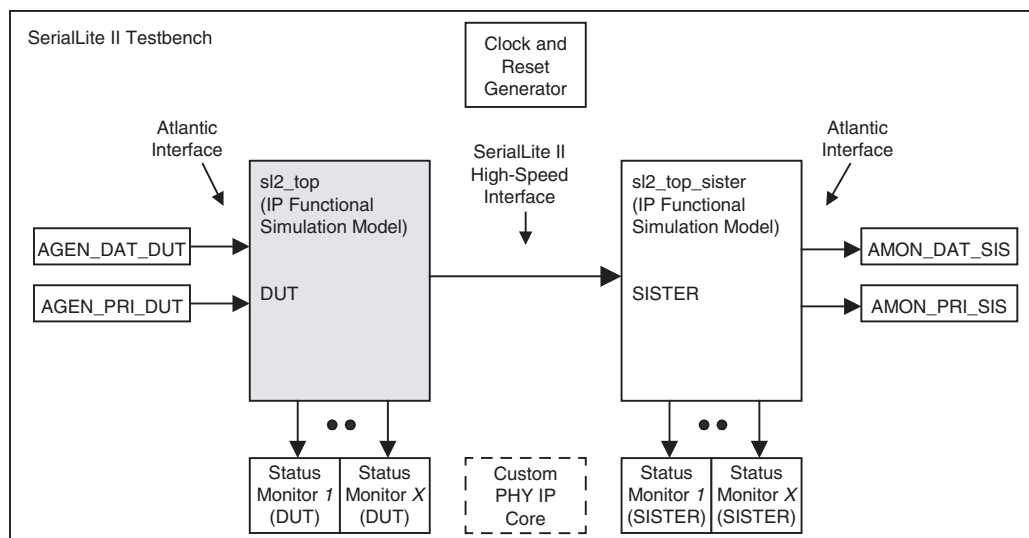


Figure 5-3 shows the testbench environment for a SerialLite II single mode-receiver only, non-broadcast IP core. The SISTER model contains a transmitter.

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

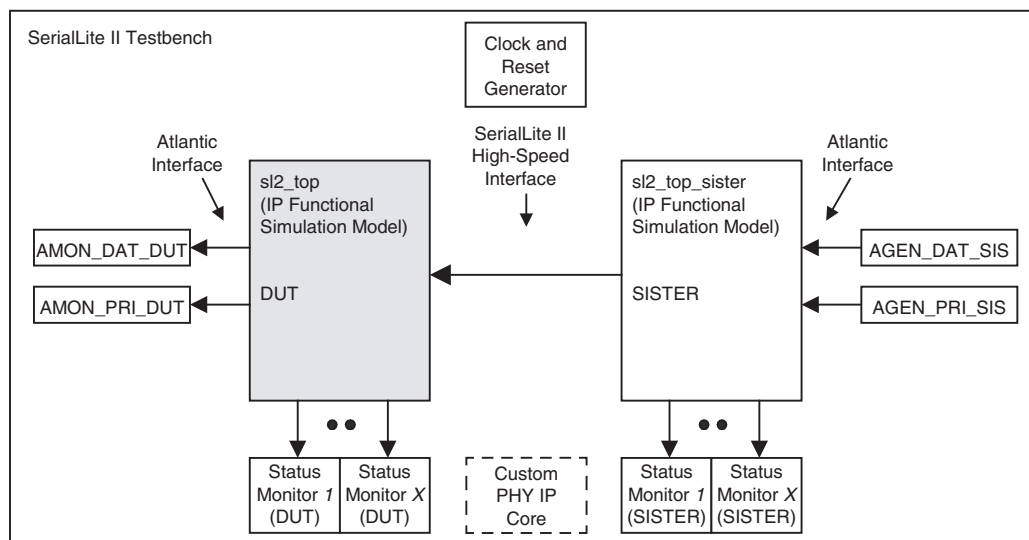
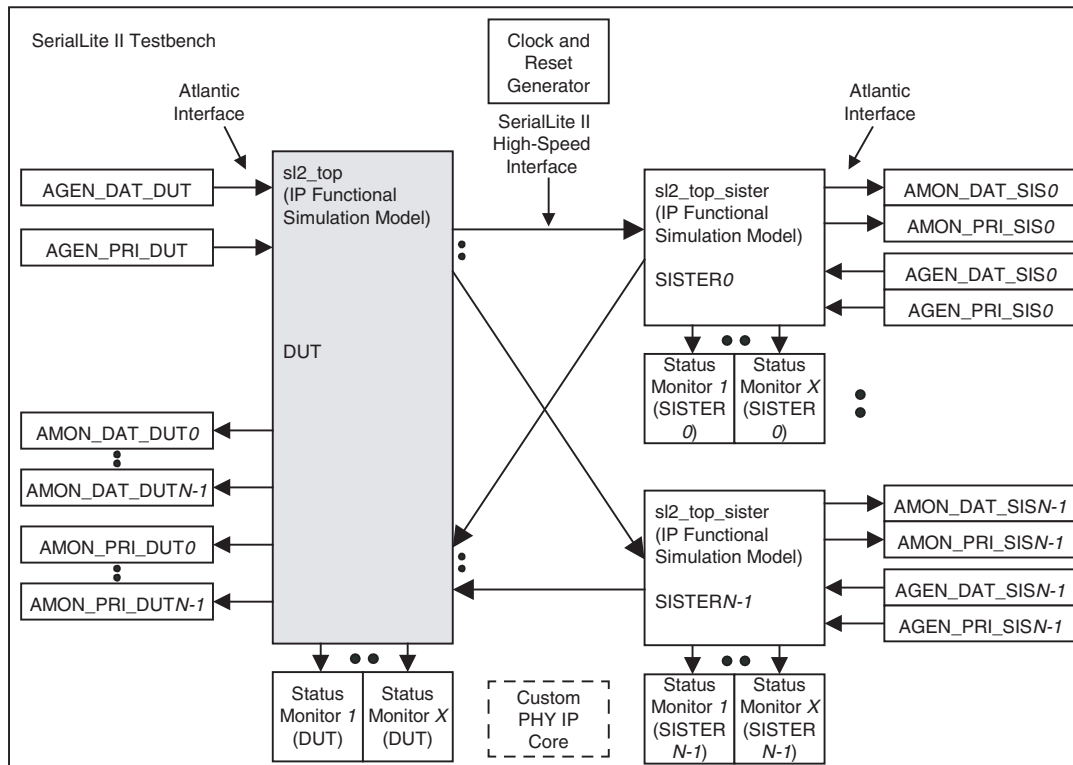


Figure 5-4 on page 5-5 shows the testbench environment for a SerialLite II standard broadcast mode IP core with multiple SISTER instances that have one receive and transmit port.

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 select Arria V or Stratix V as the target device family, you are required to add a list of the Custom PHY IP core simulation files into the command line Tcl file. For more information about the simulation support, refer to “[Configuration for Arria V, Cyclone V, and Stratix V Devices](#)” on page 4-19.

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 (Part 1 of 2)

```
*****
#CORE DUT : Comming out of RESET
# Note : CMU PLL is reset
# Time: 0 ns Instance: tb.slite2_top_sis.nl0i010.m_cdr.m_rxpll
# Note : CMU PLL is reset
# Time: 0 ns Instance: tb.slite2_top_dut.nl1l1i.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.nl1l10
# Note : CMU PLL is reset
# Time: 2 ns Instance: tb.slite2_top_sis.nl0i001
# *****
# 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
# 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
# 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
*****
```

Example 5-1. run_modelsim.log (Part 2 of 2)

```
# 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.

```
# 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 Time-Out

The testbench uses a maximum simulation time to guard against infinite loops or stuck simulations. The default value of 500,000,000 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 time-out resets with another WATCHTIME time.

Special Simulation Configuration Settings

The SerialLite II IP core contains few settings that have a reduced value in simulation:

- The internal counter that controls the duration of the digital resets to the ALTGX IP core 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 IP core 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 IP cores 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 an 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 IP core. 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.

Figure 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	<code>addr</code>	0 to 0xFF (data) 0 to 0xF (priority)	Set to 0.
2	<code>size</code>	0 to 0xFFFF_FFFF (bytes)	The <code>size</code> field sets the size, in bytes, of the current packet being sent by this task.
3	<code>err</code>	1'b0 or 1'b1	The <code>err</code> field determines whether an Atlantic error is asserted at the end of a packet when <code>eop</code> 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	<code>min</code>	0 to 0xFFFF_FFFF (cycles)	The <code>min</code> field sets the minimum value, in Atlantic clock cycles, for a random gap between two packets.
2	<code>max</code>	0 to 0xFFFF_FFFF (cycles)	The <code>max</code> field sets the maximum value, in Atlantic clock cycles, for a random gap between two packets. A <code>max</code> field greater than or equal to the <code>min</code> field is required. When <code>max==min</code> , 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	<code>prob</code>	0 to 0xFFFF_FFFF (integer)	The <code>prob</code> field sets the probability of a transaction gap. The probability decreases with a larger value of <code>prob</code> . Before each transaction, a random number between 0 and <code>prob</code> is generated and compared to <code>prob/2</code> . If they match, a random gap is inserted; if not, no gap is inserted.
2	<code>min</code>	0 to 0xFFFF_FFFF (cycles)	The <code>min</code> field sets the minimum value, in Atlantic clock cycles, for a random gap between AGEN write transactions.
3	<code>max</code>	0 to 0xFFFF_FFFF (cycles)	The <code>max</code> field sets the maximum value, in Atlantic clock cycles, for a random gap between AGEN write transactions. A <code>max</code> field greater than or equal to the <code>min</code> field is required. When <code>max==min</code> , 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	<code>bit_value</code>	1'b0 or 1'b1	Setting <code>bit_value</code> to 1, enables the display of verbose messages. Setting <code>bit_value</code> 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

The parameter editor sets these parameters based on the selected configuration, and the parameters 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.

```
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`.

```
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

The parameter editor sets these parameters based on the selected configuration, and the parameters 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.

```
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.

```
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
<code>on</code>	This task enables monitoring (the <code>en</code> input pin must also be set high to enable monitoring).
<code>off</code>	This task disables monitoring (regardless of the value of the <code>en</code> input pin).
<code>verbose_on</code>	This task enables the display of verbose messages.
<code>verbose_off</code>	This task disables the display of verbose messages.
<code>set_expect (bit_value)</code>	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 parameter editor.

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 IP core 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:

```
SERIALLLITE2_TB_RESET_START
SERIALLLITE2_TB_RESET_END
```



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

Custom PHY IP Core

The DUT and the SISTER use an external transceiver for Arria V and Stratix V configurations. You are required to separately instantiate the Custom PHY IP core using the parameter editor.

Example Testbench – Verilog HDL

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

```
//SERIALLLITE2_TB_MAIN_START
//SERIALLLITE2_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.

Table 5-12 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 5)

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.
//----- ----- //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 <i>tc_start</i> / <i>tc_end</i> 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 IP core.
pkt_cnt_pri_dut = packets_to_send;	Sets the number of packets to be sent to the high priority port of the DUT IP core.
pkt_cnt_dat_sis = packets_to_send;	Sets the number of packets to be sent to the regular data port of the SISTER IP core.
pkt_cnt_pri_sis = packets_to_send;	Sets the number of packets to be sent to the high priority port of the SISTER IP core.
wait (linked_up == 1);	Wait for DUT and SISTER to go into link-up.

Table 5-12. Example of a Demonstration Testbench (Part 2 of 5)

Main Section	Comments
fork	Launch multiple send packet loops in parallel.
<pre> begin // Generate RDP packets for DUT // Generate RDP packets for DUT @ (posedge trefclk); agen_dat_dut.verbose(1); agen_dat_dut.ipg(0,5); amon_dat_sis.verbose(1); fork while (pkt_cnt_dat_dut > 0) begin : send_loop_dat_dut 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_dut.send_packet(addr,size,err); reset_watchdog_timer; pkt_cnt_dat_dut = pkt_cnt_dat_dut - 1; end begin fork amon_dat_sis.wait_all_packets(packets_to_send); join end join end </pre>	<p>Send regular data packets (on Atlantic interface) to the DUT.</p> <p>AGEN and AMON instantiations are set to display verbose messages.</p> <p>Set AGEN to insert random inner packet gaps.</p> <p>Launch two processes in parallel:</p> <ul style="list-style-type: none"> - Send regular data packets to the DUT. <p>Define packet size, error, address.</p> <p>Packet address is a random number from 0 to 255.</p> <p>Packet size is a random number from 1 to 1,024.</p> <p>Packet err is a random number from 0 to 1.</p> <p>Call the AGEN send packet task (regular data, DUT).</p> <p>Reset watchdog with every packet being sent.</p> <p>Repeat this loop <i>pkt_cnt_dat_dut</i> times.</p> <ul style="list-style-type: none"> - Wait for the other side (Atlantic interface of the SISTER) to receive all these packets.

Table 5-12. Example of a Demonstration Testbench (Part 3 of 5)

Main Section	Comments
<pre> 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(addr,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 </pre>	<p>Send regular data packets (on Atlantic interface) to the SISTER IP core.</p> <p>AGEN and AMON instantiations are set to display verbose messages.</p> <p>Set AGEN to insert random inner packet gaps.</p> <p>Launch two processes in parallel:</p> <ul style="list-style-type: none"> - Send regular data packets to the SISTER. <p>Define packet size, error, address.</p> <p>Packet address is a random number from 0 to 255.</p> <p>Packet size is a random number from 1 to 1,024.</p> <p>Packet err is a random number from 0 to 1.</p> <p>Call the AGEN send packet task (regular data, SISTER).</p> <p>Reset watchdog with every packet being sent.</p> <p>Repeat this loop <i>pkt_cnt_dat_sis</i> times.</p> <p>- Wait for the other side (Atlantic interface of the DUT) to receive all these packets.</p>

Table 5-12. Example of a Demonstration Testbench (Part 4 of 5)

Main Section	Comments
<pre> 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 </pre>	<p>Send high priority packets (on Atlantic interface) to the DUT IP core.</p> <p>AGEN and AMON instantiations are set to display verbose messages.</p> <p>Set AGEN to insert random inner packet gaps.</p> <p>Launch two processes in parallel:</p> <ul style="list-style-type: none"> - Send high priority packets to the DUT. <p>Define packet size, error, address.</p> <p>Packet address is a random number from 0 to 15.</p> <p>Packet size is a random number from 1 to 780.</p> <p>Packet err is a random number from 0 to 1.</p> <p>Call the AGEN send packet task (high priority, DUT)</p> <p>Reset watchdog with every packet being sent.</p> <p>Repeat this loop <i>pkt_cnt_pri_dut</i> times.</p> <ul style="list-style-type: none"> - Wait for the other side (Atlantic interface of the SISTER) to receive all these packets.

Table 5-12. Example of a Demonstration Testbench (Part 5 of 5)

Main Section	Comments
<pre> 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 </pre>	<p>Send high priority packets (on Atlantic interface) to the SISTER IP core.</p> <p>AGEN and AMON instantiations are set to display verbose messages.</p> <p>Set AGEN to insert random inner packet gaps.</p> <p>Launch two processes in parallel:</p> <ul style="list-style-type: none"> - Send high priority packets to the SISTER. <p>Define packet size, error, address.</p> <p>Packet address is a random number from 0 to 15.</p> <p>Packet size is a random number from 1 to 780.</p> <p>Packet err is a random number from 0 to 1.</p> <p>Call the AGEN send packet task (high priority, SISTER).</p> <p>Reset watchdog with every packet being sent.</p> <p>Repeat this loop <i>pkt_cnt_pri_sis</i> times.</p> <ul style="list-style-type: none"> - Wait for the other side (Atlantic interface of the DUT) to receive all these packets.
<pre> join </pre>	<p>All loops must finish (receive all packets) before exiting.</p>
<pre> tc_end(`TBID); exit; end </pre>	<p>End of test case.</p> <p>Main initial block end.</p>
<pre> endmodule </pre>	<p>End of testbench main section.</p>
<pre> //SERIALLITE2_TB_MAIN_END </pre>	

This chapter provides additional information about the document and Altera.

Document Revision History

The following table lists the revision history for this document.

Date	Version	Changes
July 2014	2014.07.09	<ul style="list-style-type: none"> ■ Replaced MegaWizard Plug-In Manager information with IP Catalog. ■ Added standard information about upgrading IP cores. ■ Added standard installation and licensing information. ■ Removed obsolete device information.
November 2013	13.1	<ul style="list-style-type: none"> ■ Removed information about Arria GX, HardCopy IV, Stratix GX, and Stratix II GX devices. Altera no longer supports these devices. ■ Added Cyclone V device support. ■ Updated data rate information for Cyclone V devices.
July 2012	12.0	<ul style="list-style-type: none"> ■ Added Arria V and Stratix V device support. ■ Added information about MegaCore configuration for Arria V and Stratix V devices.
February 2011	10.1	<ul style="list-style-type: none"> ■ Updated Arria II GX and Stratix IV device support information. ■ Added information about FIFO threshold settings.
July 2010	10.0	Updated Stratix IV device support information.
November 2009	9.1	<ul style="list-style-type: none"> ■ Added HardCopy IV GX device support. ■ Added timing diagrams in Initialization and Restart, and Atlantic Interface sections.
March 2009	9.0	Added Arria II GX device support.
November 2008	8.1	Added requirement to configure a dynamic reconfiguration block with Stratix IV transceivers, to enable offset equalization.
May 2008	8.0	<ul style="list-style-type: none"> ■ Added additional ALT2GXB parameters - V_{CCH}, Reference Input Clk Frequency and Reconfiguration Channel Number. ■ Added Stratix IV device support.
October 2007	7.2.0	<ul style="list-style-type: none"> ■ 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 IP core instantiation into a separate file to ease post-generation changes.

How to Contact Altera

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


Contact ⁽¹⁾	Contact Method	Address
Technical support	Website	www.altera.com/support
Technical training	Website	www.altera.com/training
	Email	custrain@altera.com
Product literature	Website	www.altera.com/literature
Nontechnical support (general) (software licensing)	Email	nacomp@altera.com
	Email	authorization@altera.com







Note to Table:

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

Typographic Conventions

The following table shows the typographic conventions this document uses.

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Indicate command names, dialog box titles, dialog box options, and other GUI labels. For example, Save As dialog box. For GUI elements, capitalization matches the GUI.
bold type	Indicates directory names, project names, disk drive names, file names, file name extensions, software utility names, and GUI labels. For example, \qdesigns directory, D: drive, and chiptrip.gdf file.
<i>Italic Type with Initial Capital Letters</i>	Indicate document titles. For example, <i>Stratix IV Design Guidelines</i> .
<i>italic type</i>	Indicates variables. For example, $n + 1$. Variable names are enclosed in angle brackets (< >). For example, <file name> and <project name>.pof file.
Initial Capital Letters	Indicate keyboard keys and menu names. For example, the Delete key and the Options menu.
“Subheading Title”	Quotation marks indicate references to sections in a document and titles of Quartus II Help topics. For example, “Typographic Conventions.”
Courier type	Indicates signal, port, register, bit, block, and primitive names. For example, data1, tdi, and input. The suffix n denotes an active-low signal. For example, resetn. Indicates command line commands and anything that must be typed exactly as it appears. For example, c:\qdesigns\tutorial\chiptrip.gdf. Also indicates sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword SUBDESIGN), and logic function names (for example, TRI).
↵	An angled arrow instructs you to press the Enter key.
1., 2., 3., and a., b., c., and so on	Numbered steps indicate a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■ ■ ■	Bullets indicate a list of items when the sequence of the items is not important.
	The hand points to information that requires special attention.

Visual Cue	Meaning
	The question mark directs you to a software help system with related information.
	The feet direct you to another document or website with related information.
	The multimedia icon directs you to a related multimedia presentation.
	A caution calls attention to a condition or possible situation that can damage or destroy the product or your work.
	A warning calls attention to a condition or possible situation that can cause you injury.
	The envelope links to the Email Subscription Management Center page of the Altera website, where you can sign up to receive update notifications for Altera documents.

