SerialLite II IP Core User Guide

Subscribe

Send Feedback

Last updated for Quartus Prime Design Suite: 16.1

UG-0705

2019.01.09

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

ALTERA® now part of Intel

Contents

SerialLite II IP Core Overview

1

2019.01.09

UG-0705 Subscribe Send Feedback

The SerialLite II MegaCore function 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.

Table 1-1: SerialLite II IP Core Release Information

Information	Description
Version	16.1
Release Date	October 2016
Ordering Code	IP-SLITE2
Device Family Support	Intel® Stratix® 10, Arria® 10, Arria V, Arria II GX, Cyclone V, Stratix V and Stratix IV device families. **Note:** Intel Stratix 10 devices are indirectly supported by SerialLite II IP core version 16.1 and later. Arria 10 devices are indirectly supported by the SerialLite II IP core version 15.0 and later. If your design needs to implement SerialLite II interface in Intel Stratix 10 and Arria 10 devices, contact your local Altera representative or file a Service Request (SR) to obtain a design example, a guideline document, and a special license to enable the Quartus Prime software to generate the FPGA configuration file (`.sof`) for the Intel Stratix 10 and Arria 10 devices.

Altera verifies that the current version of the Quartus Prime software compiles the previous version of each IP core. The IP Core Release Notes and Errata report any exceptions to this verification. Altera does not verify compilation with IP core versions older than one release.

Table 1-2: SerialLite II IP Core Features

Features	Description
Physical layer features	• 622 Mbps to 6.375 Gbps per lane(1) • 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

Related Information

SerialLite II IP Core User Guide Archives on page 5-1
Provides a list of user guides for previous versions of the SerialLite II IP core.

General Description

The SerialLite II IP core is a simple, high-speed, low-latency, and low-resource point-to-point serial data communication link.

The SerialLite II IP core performs up to:

- 3.75 Gbps in Arria II GX devices
- 5 Gbps in Cyclone V devices
- 6.375 Gbps in Arria V, Stratix IV, and Stratix V devices
- More than 6.375 Gbps in Intel Stratix 10 and Arria 10 devices

(1) For Intel Stratix 10 and Arria 10 devices, the IP core supports higher than 6.375 Gbps per lane.

The SerialLite II IP core is highly configurable, and provides a wide range of functionality suited to moving data in many different environments.

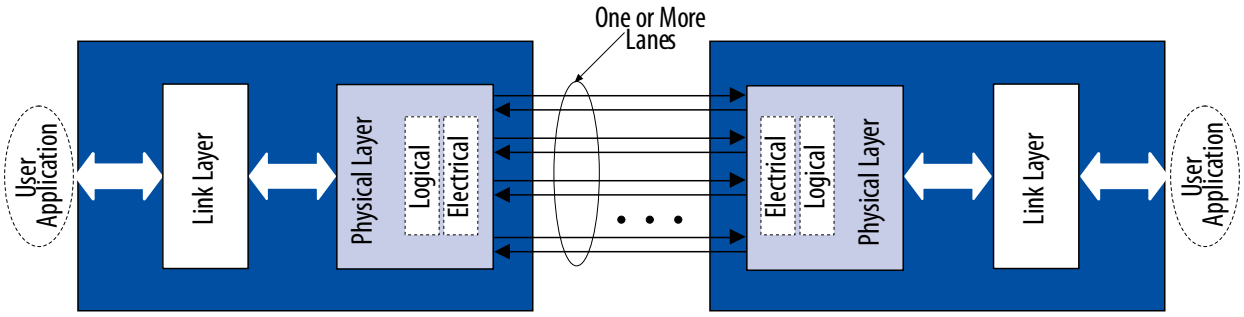
The IP core provides a simple and lightweight way to move data from one point to another reliably at high speeds. It consists of a serial link of up to 16 bonded lanes, with logic to provide a number of basic and optional link support functions. The Atlantic interface is the primary access for delivering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

The SerialLite II protocol specifies a link that is simple to build, uses as little logic as possible, and requires little work for a logic designer to implement. The SerialLite II MegaCore function uses all of the features available in the SerialLite II protocol. You can parameterize the IP core using the SerialLite II parameter editor.

A link built using the SerialLite II IP core operates at 622 Mbps to 6.375 Gbps per lane (or more for Intel Stratix 10 and Arria 10 devices). Link reliability is enhanced by the 8B10B encoding scheme and optional CRC capabilities. You can achieve further reductions in the bit-error rate by using the optional retry-on-error feature. Data rate and consumption mismatches can be accommodated using the optional flow-control feature to ensure that no data is lost.

Figure 1-1: SerialLite II IP Core High-Level Block Diagram

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).



You can use the SerialLite II IP core in the following applications:

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

The following diagrams show two examples of bridging applications.

Figure 1-2: Typical Application—Bridging Functions

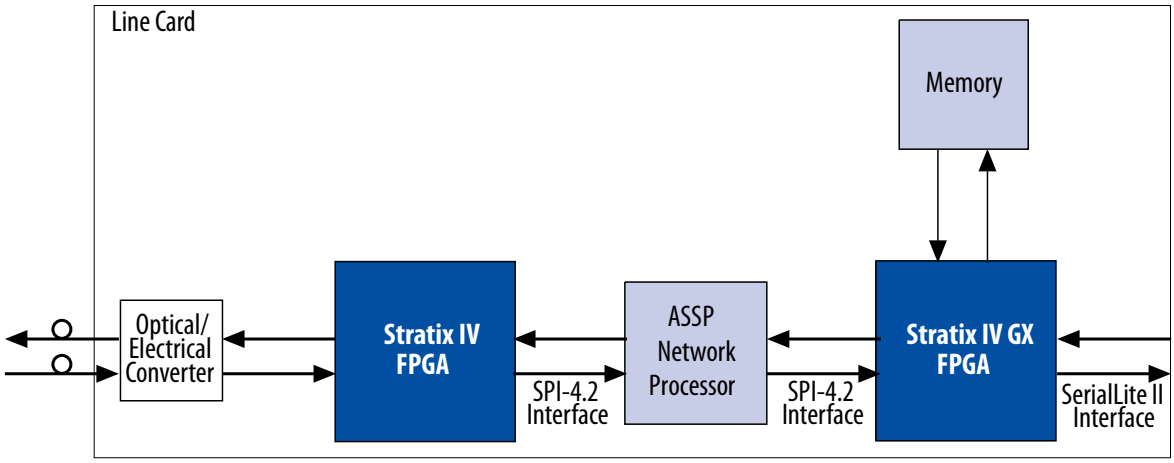
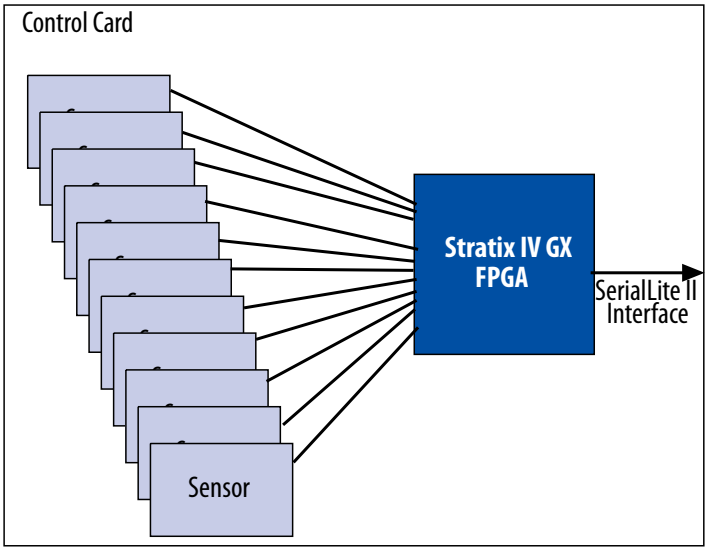


Figure 1-3: Typical Application—Unidirectional Bridging Application



Performance and Resource Utilization

The performance and resource utilization table lists the resources and internal core speeds for a selection of variations using a 1,024-byte FIFO buffer.

Table 1-3: Performance for Stratix IV Devices

These results were obtained using the Quartus Prime software version 16.0 for the Stratix IV device (EP4SGX70HF35C2).

Lane	Packet Type	Transfer Size	CRC	ROE	FC	Through-put at Mbps	RX Buffer Size (Data)	RX Buffer Size (Prio)	Combina-tional ALUT	Memory 9K	Memory ALUT	Logic Register
1	Data	1	0	0	0	1250	–	–	841	10	18	856
1	Data	2	0	0	0	3125	–	–	847	11	0	820
1	Data	4	0	0	0	6375	–	–	935	21	20	959
4	Data	1	0	0	0	1250	–	–	1399	21	56	1291
4	Data	2	0	0	0	3125	–	–	1636	31	112	1507
4	Data	4	0	0	0	6375	–	–	2184	50	0	1748
16	Data	2	0	0	0	3125	–	–	4416	87	308	3656
16	Data	4	0	0	0	6375	–	–	6685	181	0	5066
1	Data	1	32	0	1	1250	1024	–	1135	11	35	1265
1	Data	2	32	0	1	3125	1024	–	1189	12	16	1206
1	Data	4	32	0	1	6375	2048	–	1413	22	36	1357
4	Data	1	32	0	1	1250	2048	–	1814	22	72	1675
4	Data	2	32	0	1	3125	4096	–	2380	32	128	1995
4	Data	4	32	0	1	6375	8192	–	3402	51	16	2326
1	Priority	2	16	1	1	3125	–	1024	1519	21	16	1396
1	Priority	4	16	1	1	6375	–	2048	1745	32	36	1541
4	Priority	2	16	1	1	3125	–	4096	2687	51	128	2165
4	Priority	4	16	1	1	6375	–	8192	3661	71	488	3626

Table 1-4: Fmax for Stratix IV Devices

The performance results were obtained using these frequencies at 900 mV 85°C Corner.

Lane	Packet Type	Transfer Size	Fmax (txrdp_clk) MHz	Fmax (rxrdp_clk) MHz	Fmax (txhpp_clk) MHz	Fmax (rxhpp_clk) MHz
1	Data	1	252.78	261.3	–	–
1	Data	2	294.38	302.57	–	–
1	Data	4	292.06	298.06	–	–
4	Data	1	287.77	286.62	–	–
4	Data	2	285.47	299.94	–	–
4	Data	4	325.31	292.91	–	–

Lane	Packet Type	Transfer Size	Fmax (txrdp_clk) MHz	Fmax (rxrdp_clk) MHz	Fmax (txhpp_clk) MHz	Fmax (rxhpp_clk) MHz
16	Data	2	312.99	285.8	–	–
16	Data	4	260.35	247.52	–	–
1	Data	1	258.67	274.73	–	–
1	Data	2	281.61	256.02	–	–
1	Data	4	282.97	274.35	–	–
4	Data	1	291.12	271.67	–	–
4	Data	2	269.61	264.97	–	–
4	Data	4	271.67	267.02	–	–
1	Priority	2	–	–	287.69	252.33
1	Priority	4	–	–	271.59	283.77
4	Priority	2	–	–	259.61	288.1
4	Priority	4	–	–	337.5	272.33

SerialLite II IP Core Getting Started

2

2019.01.09

UG-0705

Subscribe Send Feedback

The SerialLite II IP core is installed as part of the Intel Quartus® Prime installation process.

You can select and parameterize any Altera IP core from the library. Altera provides an integrated parameter editor that allows you to customize the SerialLite II IP core to support a wide variety of applications.

Related Information

- **Introduction to Altera IP Cores**
  Provides general information about all Intel FPGA IP cores, including parameterizing, generating, upgrading, and simulating IP cores.
- **Creating Version-Independent IP and Qsys Simulation Scripts**
  Create simulation scripts that do not require manual updates for software or IP version upgrades.
- **Project Management Best Practices**
  Guidelines for efficient management and portability of your project and IP files.

Parameterize the IP Core

The SerialLite II IP core parameter editor guides you through the setting of parameter values and selection of optional ports.

1. Click **Parameter Settings** in the SerialLite II parameter editor. The **Physical Layer** page appears.
2. Key in a data rate in megabits per second (Mbps). The SerialLite II IP core supports data rates of 622 to 6,375 Mbps per lane.

   Note: For Intel Stratix 10 and Arria 10 devices, the IP core supports higher than 6.375 Gbps per lane.
3. Choose a **Transfer size**. The **Transfer size** determines the number of contiguous data columns. The **Transfer size** also determines the serialization/deserialization (SERDES) factor and internal data path width.
4. Specify the **Reference Clock Frequency**. This option defines the frequency of the reference clock for the Arria II GX or Stratix IV internal transceiver. You can select any frequency supported by the transceiver.

   This option is not available in Arria V, Cyclone V, and Stratix V configurations.
5. Select a **Port Type**. You have three choices: **Bidirectional**, **Transmitter only**, and **Receiver only**.

   If you choose **Transmitter only** or **Receiver only**, the **Self-Synchronized Link-Up** parameter (LSM) is enabled by default.

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

   This feature is only for single lane applications.
7. Under **Transmitter Settings**, select the number of lanes for the transmitter.
8. Turn on or off the **Scramble** and **Broadcast mode** options.
9. Under **Receiver Settings**, select the number of lanes for the receiver.

Table 2-1: Number of Transmit Lanes

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

10. Turn on or off the **De-scramble** option.
11. Turn on or off the **Enable frequency offset tolerance** option.
12. Click **Configure Transceiver**. Select the following parameters on the **Configure Transceiver** page to configure the ALTGX IP core for Arria II GX and Stratix IV devices.

    - For the transmitter, select the **Voltage Output Differential (VOD)** control setting value.
    - Under **Pre-emphasis**, select a value for **Specify pre-emphasis control setting**.
    - In the **Bandwidth mode** list, select **high** or **low** for the Tx PLL bandwidth.
    - Select a value for the **Transmitter Buffer Power (VCCH)**.
    - Under **Receiver Functionality**, select a value for **Specify equalizer control setting**.
    - In the **Bandwidth mode** list, select **high**, **medium** or **low** for the Rx PLL bandwidth.
    - To reconfigure functionality settings, specify a **Starting channel number**.
    - Click **Finish**.

    The **Configure Transceiver** page is disabled when you select Arria V, Cyclone V, or Stratix V as the target device family. To add a transceiver, you are required to instantiate the Custom PHY IP core.

    Note: If you want to use Intel Stratix 10, Arria 10 devices, refer to the SerialLite II IP core release information in **SerialLite II IP Core Overview** on page 1-1 for more details.
13. Click **Next** to open the **Link Layer** page.
14. Under **Data Type**, select **Packets** or **Streaming**.
15. If you select **Packets**, select a packet type: **Priority packets and data packets**, **Priority packets**, or **Data packets**.
16. If you select a packet type that includes priority packets, follow these substeps; otherwise, skip to Step 17.

- Turn on or off the `Retry-on-error` option.
- If you turned on **Retry-on-error**, specify a value for **Timeout** and **Segment size**.
- Under **Buffer Size**, specify a value for **Transmitter** and **Receiver**.
- Turn on or off the **Enable flow control** option.
- If you turned on **Enable flow control**, specify the values for **Pause quantum time**, **Threshold**, and **Refresh period**.
- If you selected **Priority packets only**, skip Step 17.

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

- Turn on or off the **Enable flow control** option.
- If you turned on **Enable flow control**, specify the values for **Pause quantum time**, **Threshold**, and **Refresh period**.
- Under **Buffer Size**, specify a value for **Transmitter** and **Receiver**.

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

Related Information

SerialLite II Parameter Settings on page 2-6

Set Up Simulation

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

To generate an IP functional simulation model for your IP core:

1. On the **EDA** page, under **Simulation Libraries**, turn on **Generate Simulation Model**.
2. Some third-party synthesis tools can use a netlist that contains only the structure of the IP core, but not detailed logic, to optimize performance of the design that contains the IP core. If your synthesis tool supports this feature, turn on **Generate netlist**.
3. Click **Next** to display the **Summary** page.

Note: For Arria V, Cyclone V, and Stratix V devices, the generated simulation model does not come with transceiver. You need to integrate yourself. When you generate the transceiver, also include the reset controller for the respective devices. For Intel Stratix 10 and Arria 10 devices, contact your local Altera representative or file a Service Request (SR).

Generate Files

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

To generate your parameterized IP core:

1. Turn on the files you want to generate.
2. To generate the specified files and close the SerialLite II parameter editor, click **Finish**. The generation phase can take several minutes to complete.
3. If you generate the IP core instance in a Quartus Prime project, you are prompted to add the Quartus Prime IP File (`.qip`) to the current Quartus Prime project.

   The `.qip` file is generated by the SerialLite II parameter editor and contains information about a generated IP core. In most cases, the `.qip` file contains all of the necessary assignments and information required to process the IP core or system in the Quartus Prime compiler. The SerialLite II parameter editor generates a single `.qip` file for each IP core.

   Note: For Arria V, Cyclone V, and Stratix V devices, you must also generate the Custom PHY and reset controller, and then add the transceiver `.qip` files in. You must manually integrate the transceiver to the SerialLite II IP core, and the reset controller to the transceiver. For Intel Stratix 10 and Arria 10 devices, contact your local Altera representative or file a Service Request (SR).
4. After your review the generation report, <*variation name*>`.html`, in your project directory, click **Exit** to close the SerialLite II parameter editor.

Simulate the Design

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

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

Note: For Arria V, Cyclone V, and Stratix V devices, the generated testbench is incorrect because the top level design has the transceiver integrated with it. The generated simulation model does not have the transceiver integrated with it, so you need to change the testbench accordingly. For these devices, you also need to modify the generated simulation script to add the Custom PHY transceiver files. For Intel Stratix 10 and Arria 10 devices, contact your local Altera representative or file a Service Request (SR).

Related Information

SerialLite II IP Core Testbench on page 4-1

Instantiate the IP Core

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

Compile and Program

After you are done with simulating and instantiating the IP core, you can compile and program your design

1. Click **Start Compilation** on the **Processing** menu in the Quartus Prime software to compile your design.
2. After successfully compiling your design, program the targeted Altera device with the **Programmer** in the **Tools** menu and verify the design in hardware.

Specify Constraints

The SerialLite II 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.

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**.

Note: 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, so 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 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.

Note: This fitter setting may conflict with your Quartus Prime 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:

1. On the **Assignments** menu, click **Timing Analysis Settings**.
2. 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.

Note: The `.sdc` generated for an Arria V, Cyclone V, or Stratix V device is incomplete. You need to change the `"set_clock_groups"` assignment which specifies `"<variant_name>*receive|clkout"` and `"<variant_nane>*transmit|clkout"` to the correct name of the `clkout` signals coming from the transceiver. Other similar clocks from the transceiver in the generated `.sdc` are also incorrect and need to be replaced by the actual name and path accordingly. For Intel Stratix 10 and Arria 10 devices, contact your local Altera representative or file a Service Request (SR).

SerialLite II Parameter Settings

You set the parameters using the SerialLite II parameter editor.

Table 2-2: SerialLite II Parameters

Parameter	Description
Physical Layer	
Device family	Select the targeted device family. **Note:** For Intel Stratix 10 and Arria 10 devices, contact your local Altera representative or file a Service Request (SR).
Data rate	Key in a data rate in megabits per second (Mbps). The SerialLite II IP core supports data rates of 622 to 6,375 Mbps per lane. **Note:** The data rate must be an acceptable range for the Transfer size. The parameter editor returns a warning or an error message if you specify a data rate that is not within the range for the specified Transfer size.
Transfer size	The **Transfer size** (TSIZE) parameter determines the number of contiguous data columns and the internal data path width per lane. • TSIZE 1— equates to an internal data path of 8 bits (Recommended for less than 2.5 gigabits per second (Gbps)) • TSIZE 2 —equates to an internal data path of 16 bits (Recommended for less than or equal to 3.125 Gbps) • TSIZE 4 —equates to an internal data path of 32 bits (Typically for greater than 3.125 Gbps, and only available for Arria V, Cyclone V, Stratix IV, and Stratix V devices) A transfer size also determines the width of the SERDES block: • TSIZE 1—10 bit-wide SERDES block • TSIZE 2—20 bit-wide SERDES block • TSIZE 4—40 bit-wide SERDES block

Parameter	Description
Reference clock frequency	This option defines the frequency of the reference clock for the Arria II GX or Stratix IV internal transceiver. You can select any frequency supported by the transceiver. **Note:** If you select a reference clock frequency that is not equal to the data rate/(transfer size) * 10, this option is disabled if you turned on the **Receiver only** port type option.
Port type	Select a port type: Bidirectional, Transmitter only, or Receiver only. **Note:** If you choose Transmitter only or Receiver only, the self-synchronized link-up parameter (LSM) is enabled by default.
Self-synchronized link-up	This parameter allows the receiver on the far end of the link to synchronize itself to incoming data streams, rather than on an exchange of status information with the transmitter. **Note:** This feature is only for single lane applications.
Number of lanes (Transmitter and Receiver settings)	Select the number of lanes for the transmitter and receiver. This parameter dictates the number of serial links, essentially the number of external inputs and outputs (I/Os) for the IP core. Because each lane operates at the bit rate, you can increase the bandwidth by adding lanes. **Note:** 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	Turn on to scramble the data. Scrambling the data eliminates repeating characters that affect the EMI substantially at high data rates. This parameter applies only to the transmitter, and allows for scrambling (like CRC) to be enabled in one direction only, as required. Scrambling is recommended for data rates greater than 3,125 Mbps, and is optional for lower data rates (622 to 3,125 Mbps).
De-scramble	Turn to to descramble the data. 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.

Parameter	Description
Broadcast mode	Turn on to use broadcast mode. This parameter applies only to the transmitter. If you enable this parameter, you configure the IP core to use a single shared transmitter and multiple receivers in the master device.
Enable frequency offset tolerance	This 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 turn on this option, select an offset tolerance of ±100 or ±300 parts per million (ppm).
Link Layer	
Data type	Select whether to format the data as a stream or in packets. If you select **Streaming**, all link layer basic parameters, including data and priority ports, and buffering are disabled (grayed out). Streaming mode does not include link-layer functions.
Packet type	Select whether to send your packets as priority packets, data packets, or both.
Enable flow control	The SerialLite II IP core provides this parameter as an optional means of exerting backpressure on a data source when data consumption is too slow. Turn on this parameter to ensure that the receive FIFO buffers do not overflow. **Note:** 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.
Pause quantum time	Activation of flow control causes a pause in transmission. Specify a pause duration from to 8 to 2,040 columns.
Threshold	You must set the **Threshold** parameter 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
Refresh period	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 period must be less than the pause quantum time. The packet is retransmitted if the FIFO buffer is still breached.

Parameter	Description
Retry-on-error	This parameter improves the bit error rate of your data. • **On**: Logic is created to acknowledge segments and retransmit segments when errors occur. Eight transmit segment buffers are created. • **Off**: Logic is not created to acknowledge segments. This is the default setting. If you turn off this parameter, no segment acknowledgments are generated or expected, and all segments are transmitted without any acknowledgments from the receiver. This parameter is only available for priority packets.
Timeout	Set the time out value for the segment to be acknowledged. The time-out value is based primarily on the round trip latency—from the time a packet is sent to when the acknowledge signal 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. • Do not to set the time out to be too long so the system does not have to wait too long for link errors to resolve. • Do not set the time out to be too short because then the system always times out and the link never remains up.
Segment size	This parameter is only applicable when the **Retry-on-error** parameter is turned on. This parameter settings range from 8 to 2,048 bytes in 2^n increments, and the default value is 256 bytes. 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. This buffering is required to support the **Retry-on-error** option, which is only allowed for priority packets. If a packet is larger than a segment size, a full segment must be queued before it can be transmitted. This queuing 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.
Buffer size (Transmitter and Receiver)	Specify a FIFO buffer size value for the transmitter and receiver.

Parameter	Description
Enable CRC for priority/data packets (Transmitter and Receiver)	If your transmitter or receiver requires cyclic redundancy code (CRC) checking, turn on the **Enable CRC** option for your chosen packet type. • **On**: CRC logic is created. CRC usage is specified independently for each port. • **Off**: CRC logic is not created. CRC usage is specified independently for each port. This is the default CRC setting.
CRC Type	Select 16 bits or 32 bits for the CRC type. • **16 bits**: Generates a two-byte CRC. Adequate for packets of around 1 KBytes or smaller. This is the default algorithm when CRC is enabled. • **32 bits**: 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.
Configure Transceiver (only applicable for Arria II GX and Stratix IV devices)	
Specify VOD control setting	Select the Voltage Output Differential (VOD) control setting value. **Note:** This parameter is disabled when the number of lanes in the transmit direction is equal to zero.
Specify pre-emphasis control setting	Select pre-emphasis control setting value. For Stratix IV devices, the pre-emphasis control values supported are 0,1,2,3,4, and 5. • 0 = Pre-emphasis option is turned off • 1 = Maximum negative value • 2 = Medium negative value • 3 = Special value in which only the first post-tap is set (set to the maximum), while the other taps are off • 4 = Medium positive value • 5 = Maximum positive value For Arria II GX devices, the pre-emphasis setting cannot be changed. This parameter is set to 0 by default. It is disabled when the number of lanes in the transmit direction is equal to zero.

Parameter	Description
Bandwidth mode (Transmitter and Receiver)	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. Select **low** or **high** bandwidth mode for the transmitter and **low**, **medium**, or **high** bandwidth mode for the receiver. • 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. • 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. If the number of lanes in the transmit or receive direction is equal to zero, the bandwidth mode for that direction is disabled. **Note:** This parameter is not applicable for Arria II GX devices.
Transmitter buffer power (VCCH)	This setting is used to calculate the VOD from the buffer power supply and the transmitter termination to derive the proper VOD range. • Arria II GX devices = 1.5 V • Stratix IV devices = 1.4 V or 1.5 V
Specify equalizer control setting	Select the equalizer control setting value. 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.

Parameter	Description
Starting channel number	To reconfigure the functionality settings, select a starting channel number. The range for the dynamic reconfiguration starting channel number setting is 0 to 380. 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. **Note:** This parameter is not applicable for Arria II GX devices.

Related Information

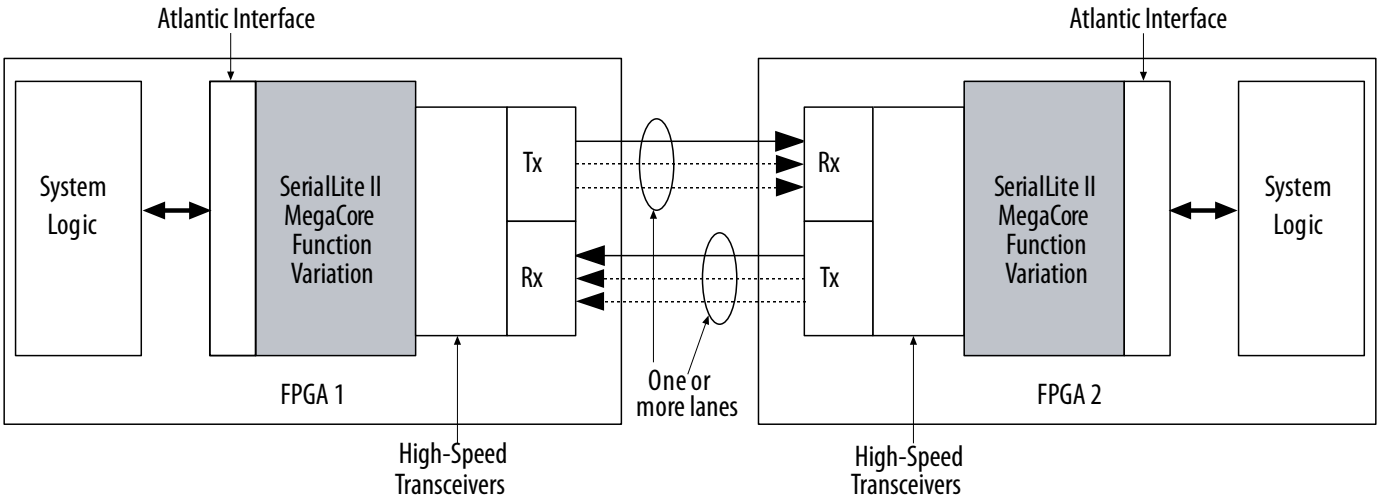
Parameterize the IP Core on page 2-1

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.

Figure 2-1: Complete SerialLite II Link



Data Rate

The data rate range varies based on the device and the transfer size (TSIZE).

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.

Table 2-3: 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

The data rates for an individual Arria II GX device are limited to the respective speed grades,

Table 2-4: 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

Reference Clock Frequency

Valid values for reference clock frequency change with the data rate but the reference input clock frequency must be within 50 MHz and 622 MHz. Range of supported reference clock frequency is dependent on device. Please refer to the device datasheet for range of supported frequency for every device.

The general formula to determine frequency:

```
Frequency = p×Data Rate/(2×m)
```

where p = 1 or 2, and m = 4, 5, 8, 10, 16, 20, or 25

Condition for frequency to be valid:

```
(50×p) < Frequency < 622
```

Port Type

The **Port Type** parameter offers three options: bidirectional, transmitter only, and receiver only.

- 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.
- 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.
- 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.

The following diagrams illustrate the symmetric and asymmetric modes.

Note: A full line indicates a mandatory lane, and a dotted line indicates an optional lane.

Figure 2-2: Symmetric Mode Block Diagram

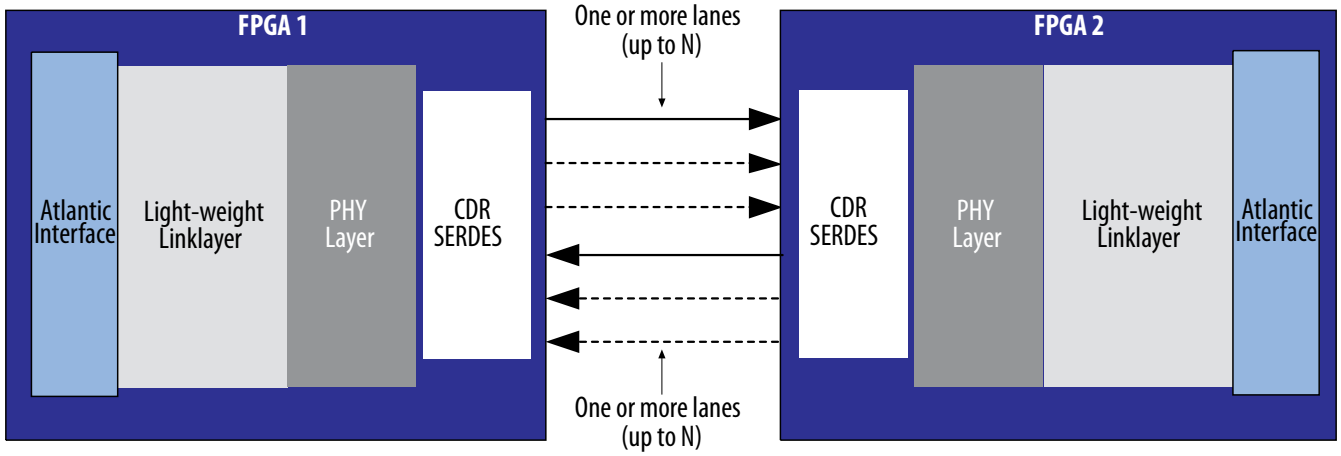


Figure 2-3: Streaming Symmetric Mode Block Diagram

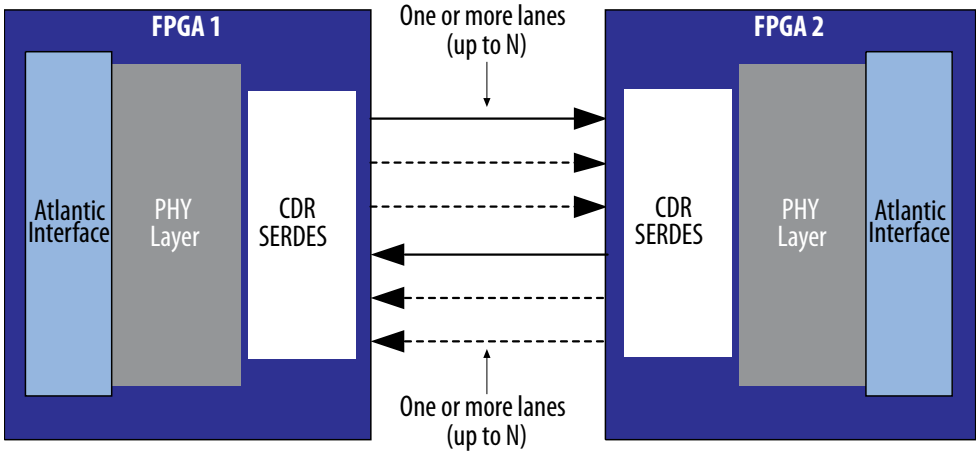


Figure 2-4: Simplex Mode Block Diagram

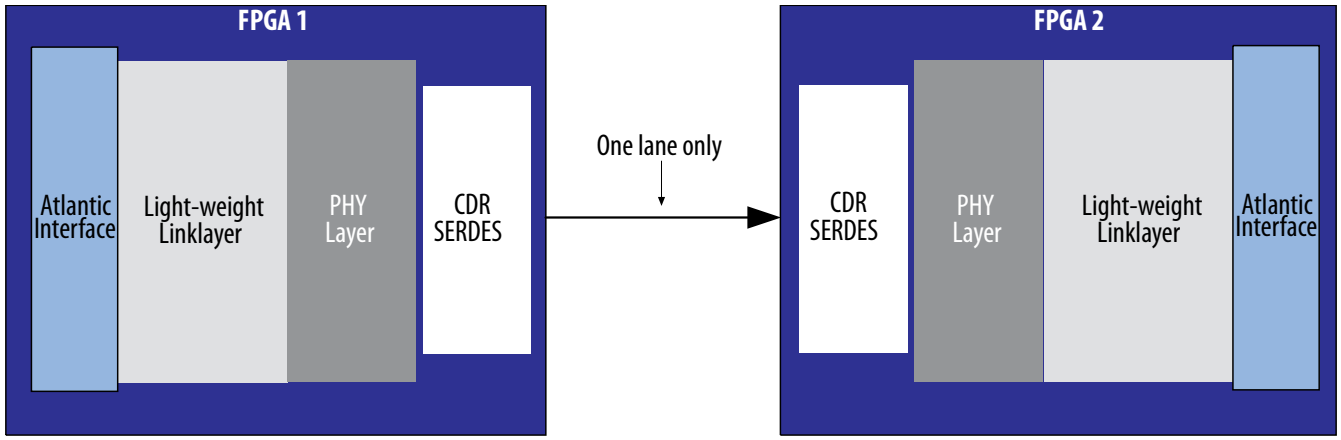


Figure 2-5: Streaming Simplex Mode Block Diagram

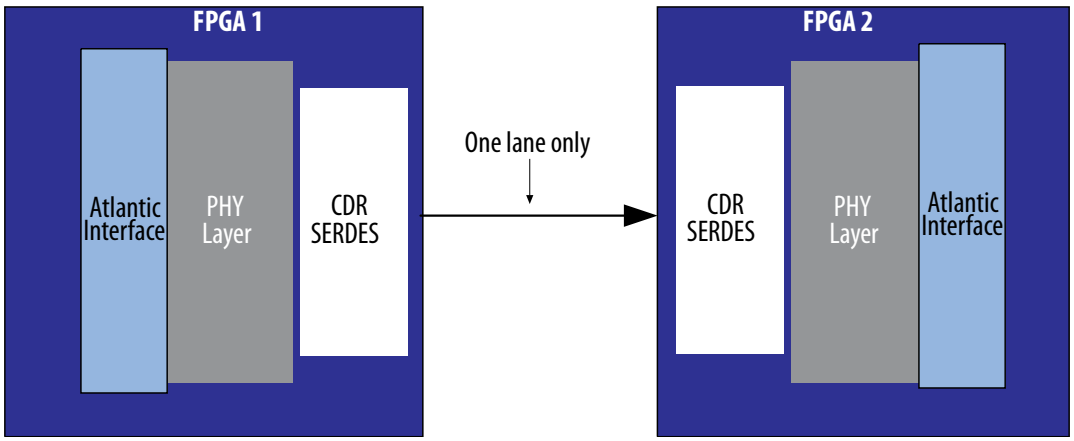


Figure 2-6: Asymmetric Mode Block Diagram

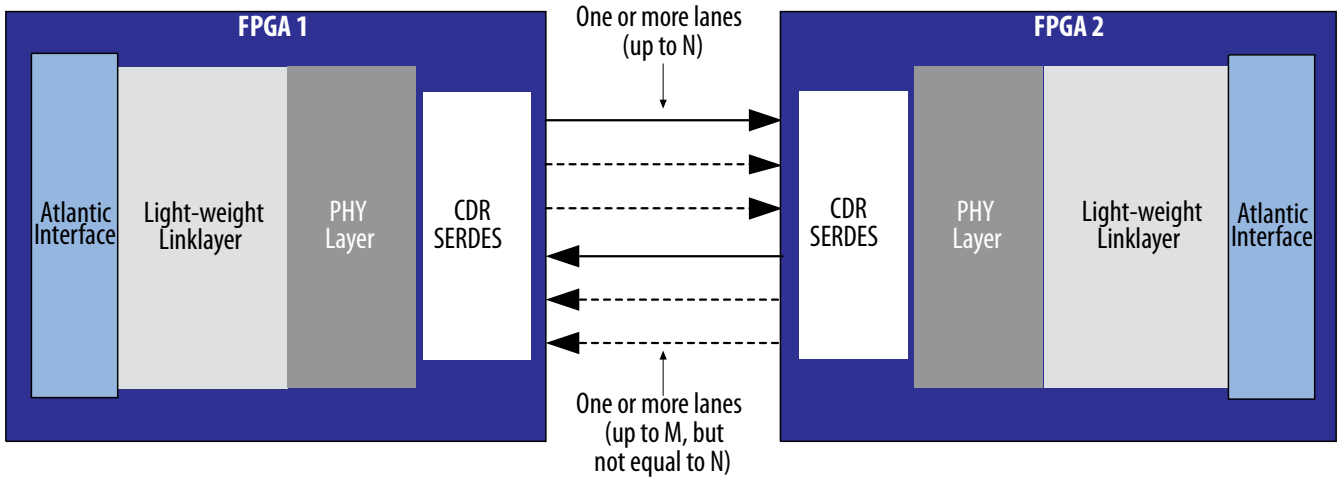
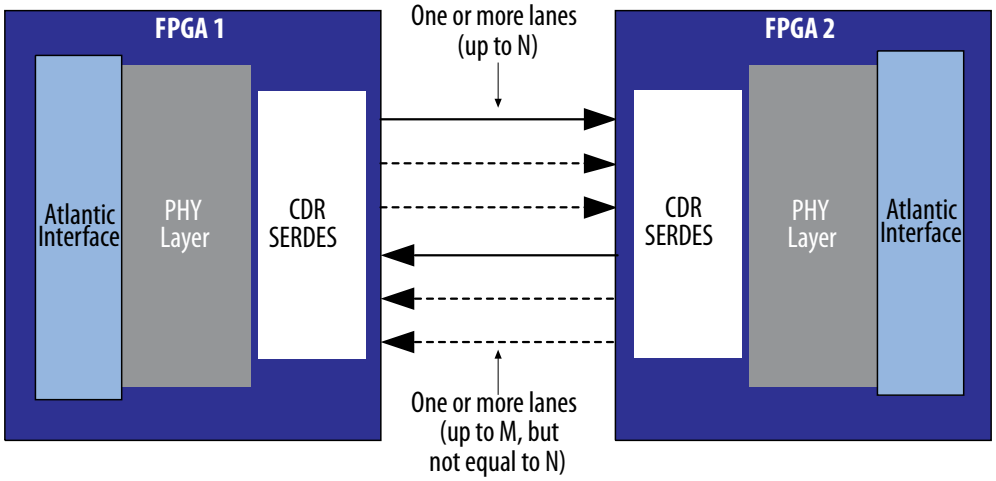


Figure 2-7: Streaming Asymmetric Mode Block Diagram



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.

The receiver on the far end must synchronize itself to incoming data streams. To do so, the receiver uses the self-synchronizing LSM, a light-weight implementation that is especially useful when data is streaming. Because there is no handshaking or exchange of status information between the receiver and transmitter, the **Self Synchronized Link Up** 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.

The **Self Synchronized Link Up** 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 in one of these two conditions:

- When the adjacent receiver has locked—if this status information can be made available.
- 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 or Custom PHY IP core completes the power-on reset cycle. The self-synchronizing link-up state machine does not look at the incoming stream until the transceiver reset is complete.

Note: The Arria II GX and Stratix IV devices use the ALTGX IP core. The later devices use the Custom PHY IP core.

For example, the following procedure shows the transceiver reset sequence in a transceiver device:

1. Wait for the pll_locked signal (stat_tc_pll_locked) to be asserted, which happens when the PLL in the ALTGX or Custom PHY IP core locks to the reference clock (trefclk). 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 or Custom PHY 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.

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

You should characterize the timing of the signals in the transceiver reset sequence 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 SerialLite II 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. When 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.

Note: The self-synchronizing LSM also locks onto the clock compensation sequence.

For Arria II GX and Stratix IV devices, you can turn on the **Enable frequency offset tolerance** option to allow the receiver to automatically relock if the link goes down. Therefore, the transmitter is not required to assert ctrl_tc_force_train to retrain the link (which may be impossible in a

unidirectional link because the transmitter does not necessarily detect that the receiver has lost the link).

For Arria V, Cyclone V, and Stratix V devices, you have to expose and integrate all the related signals from the transceiver.

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 0×FFFF XORed with the lane number (LN).

Broadcast Mode

Broadcast mode allows the SerialLite II IP core to use a single shared transmitter and multiple receivers in the master device.

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 2-8: Broadcast Mode Block Diagram

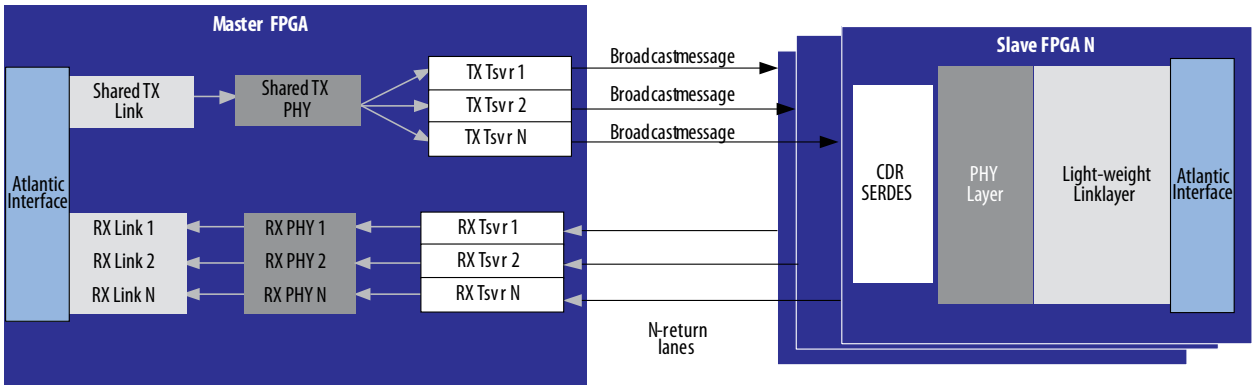
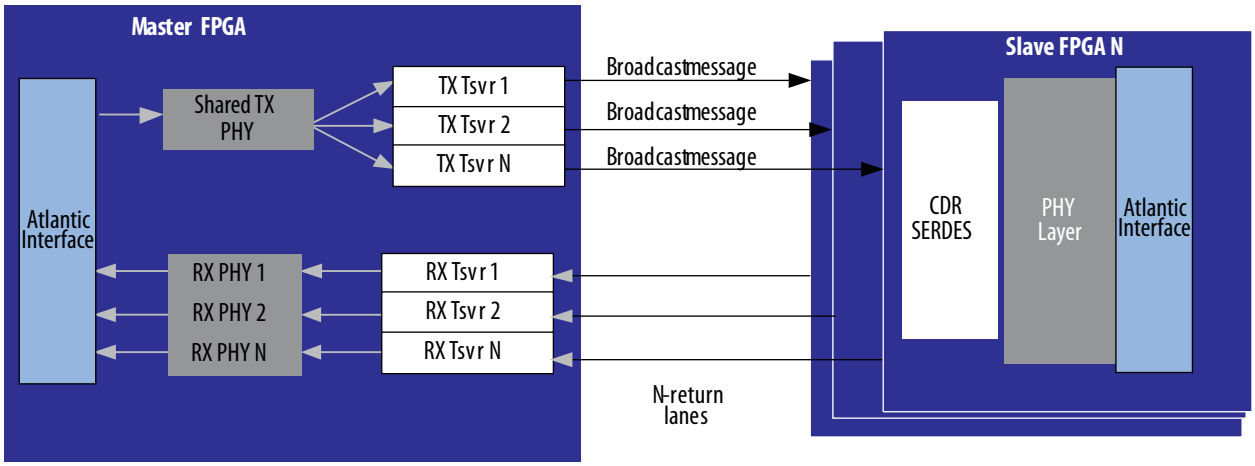


Figure 2-9: Streaming Broadcast Mode Block Diagram



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.
- 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

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:

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

Data Type

The regular data port allows data to be formatted as a stream or in packets.

Table 2-5: Data Type

Data Type	Description
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.
Streaming	• 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. • When the system link up is complete, your logic should provide data continuously. The SerialLite II IP core does not encapsulate streaming data. .

Packet Type

You can send your packets as priority packets or data packets.

Table 2-6: Differences between Data Packets and Priority Packets

Data Packets	Priority Packets (Retry-on-Error Enabled)	Priority Packets (Retry-on-Error Disabled)
• 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.	• 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. • 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).	• A store-and-forward data flow is implemented for priority packet segments. • Priority packets are broken into segment-sized 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. • 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.

Retry-on-Error

The retry-on-error mechanism improves the bit error rate of your data.

The retry-on-error parameter provides for segments with errors to be retransmitted, so that only good segments are delivered to the Atlantic receive interface. When you turn on the **Retry-on-error** parameter, all segments sent by the transmitter are acknowledged.

- `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.

The IP core can hold up to seven segments waiting for acknowledgement 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 in this sequence:

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.
   - Do 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.
   - Do not set the time out to be too short because then the system always times out and the link never remains up.

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 acknowledgments from the receiver.

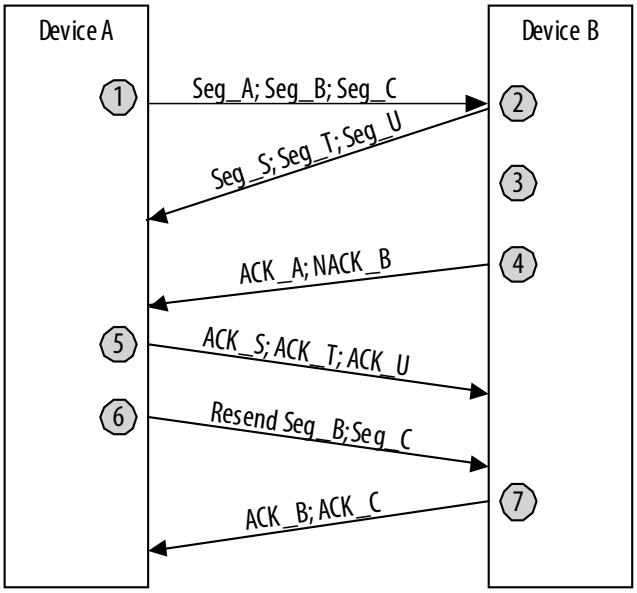
Table 2-7: Retry-on-Error Responses

This table summarizes the response to various transmission errors.

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 resend

Figure 2-10: Retry-On-Error Example

This figure shows an example of the retry-on-error operation.



Notes:
(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.

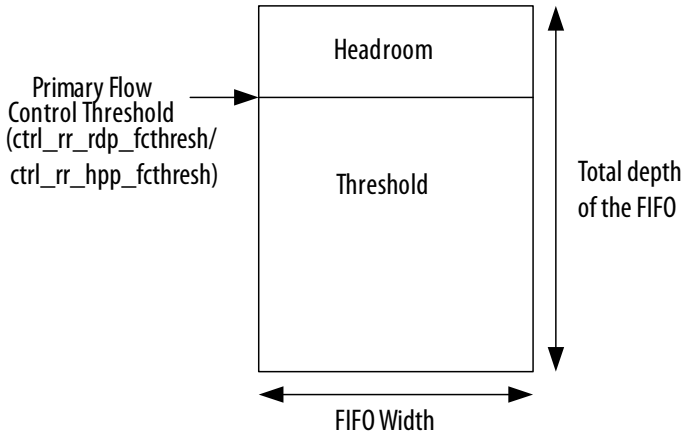
Flow Control Operation

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 PAUSE instruction causes the transmitter to cease transmission for specified pause duration. When the pause duration expires, the transmission resumes.

When flow control is used, the FIFO buffer is structured as two sections, threshold and headroom.

Figure 2-11: 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.

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

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

- Set `FIFO SIZE` by selecting a value in the **Buffer Size** (**Receiver**) option.
- Set `TSIZE` by selecting a number in **Transfer Size** option.
- Set `RX_NUMBER_LANES` by selecting a value for **Number of lanes** (**Receiver Settings**)

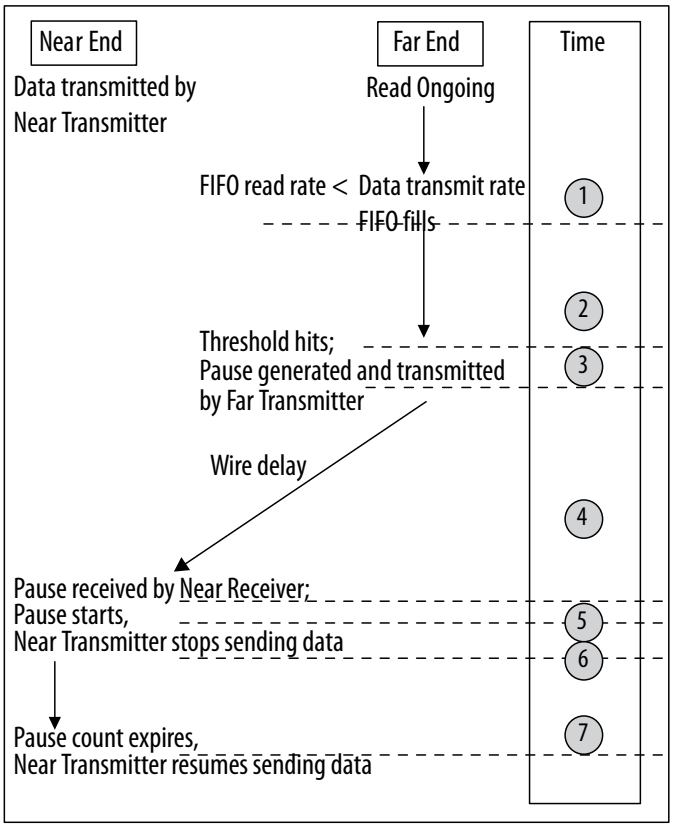
If in this example, you select a high-priority `FIFO SIZE` of 1,024 B, and a `TSIZE` of 2 in a four-lane SerialLite II configuration:

```
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.

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.

Figure 2-12: Flow Control Operation Example



Notes:

(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).

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** in the parameter editor.
- 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).

Selecting the Proper Threshold Value

To determine FIFO threshold size, you need to factor in the flow control internal latency.

Table 2-8: SerialLite II Flow Control Internal Latency

This table defines the specification value for flow control internal latency. Use this information to determine the minimum FIFO threshold size avoiding starvation during the flow control.

Internal Latency	Latency Value (cycles)	Description
`tlate_fc_transmit`	24	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.
`t_wd`	This value depends on the data rate and trace lengths in the application.	Wire delay between the devices.
`tlate_fc_receive`	23 + deskew cycles	Latency that occurs in the duration when the flow control link management packet reaches the transceiver pins until the IP core processes the request. • Deskew cycles = 0 for single lane configuration • Deskew cycles = worst case lane to lane skew in the transceiver

Internal Latency	Latency Value (cycles)	Description
tlate_stop_data	• Regular data: 41 • Priority data: 41 + seg_TX + seg_RX	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. **Note:** seg_TX and seg_RX are taken into account only for priority packets with retry-on-error feature. If a priority packet with retry-on-error feature is in transfer, flow control begins immediately after the current segment of the priority packet has been sent.] seg_TX = [segment size/(TSIZE* TX_NUMBER_LANES)] seg_RX = [segment size/(TSIZE* RX_NUMBER_LANES)]

To calculate latency numbers in terms of time units, multiply the latency values by the tx_coreclock clock period.

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

Note: 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, set threshold value based on this formula:

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

Selecting the Proper Pause Duration

You can specify the duration of the transmission pause in terms of columns.

In elements, this value is 8/TSIZE to 2,040/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 may not know that at design time, so you must estimate a reasonable value. The effect of a TSIZE-2, 120-element pause (240 columns on the GUI) causes more delay than needed. However, an 80-element delay (160 columns on the GUI) results in the pause being renewed after 80 elements, for a total 160 elements of delay, even longer than the 120-element pause.

Selecting the Proper Refresh Value

The flow control refresh period determines the number of columns before a flow control packet can be retransmitted.

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

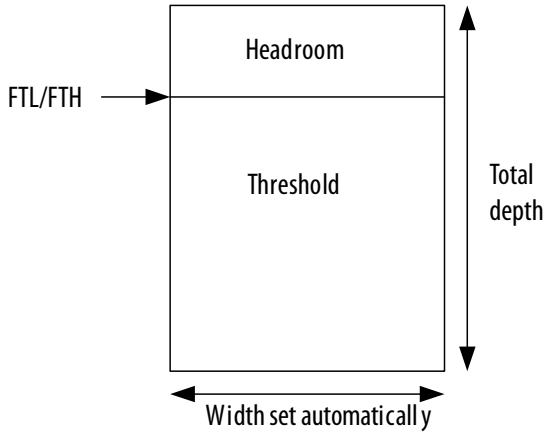
The SerialLite II IP core automatically sets the width of the receive FIFO buffers at TSIZE bytes per lane.

- 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 size of the FIFO buffers are affected by these factors:

- 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 eop_dav and a high threshold), the FIFO must be big enough to hold a full packet at minimum.
- Wire delay and bit rate—The wire propagation delay and the bit rate change the wire latency, which must be accommodated if flow control is used.

Figure 2-13: 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.

Note: 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_rxhpp_eopdav signals. The end-of-packet feature determines whether the dav remains high: if the signal is asserted, and there is an end-of- packet beneath the FTL threshed, the dav signal remains high until the end-of-packet is read out of the FIFO 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_rxhpp_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_rxhpp_fth or ctl_rxrdp_fth, or dav = 1, it means that there 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_rxhpp_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_txhpp_fth) value for transmitter variations controls when the txrdp_dav/txhpp_dav signals are asserted and deasserted for the write side of the FIFO buffer, respectively. The txrdp_dav signal indicates when there is room available to write new data into the FIFO buffer, and is asserted when the fill level of the FIFO is less than the FTH setting, and deasserted when the fill level of the FIFO is greater than the FTH.

For example, if FTH is five, and the fill level is four, the txrdp_dav/txhpp_dav signal is high, indicating that the user can write data into the FIFO. If the fill level for this example is six, the txrdp_dav/txhpp_dav signal is low, indicating that the user should stop writing data into the FIFO. ctl_txhpp_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_txhpp_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_txhpp_fth = 250 element units.

TSIZE = 2, and one FIFO element = 2 bytes

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

Note: You can set any value below 256 element units for ctl_txhpp_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.

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.

Transceiver Configuration

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.

Note: The **Configure Transceiver** option is available only for Arria II GX and Stratix IV devices. If you select Arria V, Cyclone V, or Stratix V as the target device family, you are required to instantiate the Custom PHY IP core as the hard transceiver. For Intel Stratix 10 and Arria 10 devices, contact your local Altera representative or file a Service Request (SR).

Related Information

IP Core Configuration for Intel Stratix 10, Arria 10, Arria V, Cyclone V, and Stratix V Devices on page 3-18

Voltage Output Differential (VOD) Control Settings

The Stratix IV transceivers allow you to set the VOD 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 2-9: VOD Control Settings

This table shows how the VOD value you chose in the SerialLite II parameter editor corresponds to the mV value. The VOD value is 0 by default.

VOD Value (Per Lane)	100 Ω (mV) Stratix IV
0	200
1	400
2	600
3	700
4	800
5	900
6	1,000
7	1,200

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.

Transmitter Buffer Power (VCCH)

You can set your preferred VCCH value to the transmit and receive pins.

1. In the Quartus Prime software, 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`).

Transceiver Reconfiguration Block

When you use an Arria II GX, Arria V, Intel Stratix 10, Arria 10, Cyclone V, Stratix IV, or a Stratix V device, you can instantiate a transceiver reconfiguration block.

The transceiver reconfiguration block dynamically changes the following physical media attachment (PMA) settings:

- Pre-emphasis
- Equalization
- VOD
- Offset cancellation

Note: 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 Quartus-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).

Note: For Intel Stratix 10, Arria 10, Arria II GX, and Stratix IV devices, you must instantiate the transceiver reconfiguration block on the devices, 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 Arria V, Cyclone V, and Stratix V devices, you need to include a dynamic reconfiguration block for the offset cancellation to occur.

ALTGX Support Signals

The ALTGX support signals are only present on variants that use the Arria II GX and Stratix IV integrated PHY.

The ALTGX support signals 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.

Table 2-10: ALTGX Support Signals

Signal	Direction	Description
`cal_blk_clk`	Input	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 *Arria II GX Device Handbook* or the *Stratix IV Device Handbook*.
`reconfig_clk`	Input	The `reconfig_clk` input signal is the ALTGX dynamic reconfiguration clock. This signal must be connected as described in the *Arria II GX Device Handbook* or the *Stratix IV Device Handbook* if the ALTGX dynamic reconfiguration block is used. Otherwise, this signal must be set to `1'b0`.
`reconfig_togxb`	Input	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 *Arria II GX Device Handbook* or the *Stratix IV Device Handbook* 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.

Signal	Direction	Description
reconfig_fromgxb	Output	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 cancellation support). This signal identifies the transceiver channel whose settings are being transmitted to the dynamic reconfiguration. This signal must be connected as described in the *Arria II GX Device Handbook* or the *Stratix IV Device Handbook* if the external dynamic reconfiguration block is used. Otherwise, leave this signal unconnected. For Arria II GX and Stratix IV devices, you must use the dynamic reconfiguration block because they require offset cancellation.
gxb_powerdown	Input	gxb_powerdown resets and powers down all circuits in the transceiver block. This signal does not affect the refclk buffers and reference clock lines. All the gxb_powerdown input signals of cores placed in the same quad should be tied together. The gxb_powerdown signal should be tied low or should remain asserted for at least 2 ms whenever it is asserted.

Related Information

SerialLite II Implementation in Stratix V Devices
You can use the SerialLite II implementation in Stratix V devices as a guideline to connect the transceiver signals to the core for Arria V, Cyclone V, and Stratix V devices.

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

Table 2-11: Error Summary

This table summarizes the causes and results of the SerialLite II error types.

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

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.

Table 2-12: Factors Comprising Speed

Factors	Description
Feature selection	These features impact speed more significantly: • 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. Your system may require some of these features, 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.
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 refer to the *Command-Line Scripting* and the *Design Space Explorer* chapters in the *Quartus Prime Handbook*.
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 the *Quartus Prime Handbook* for more information on limiting fanout.

Factors	Description
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 the *Quartus Prime Handbook* for more information on floorplanning.

Related Information

Quartus Prime Standard Edition Handbook Volume 2: Design Implementation and Optimization
Provides more information about design optimization.

Minimizing Logic Utilization

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

Table 2-13: Features Affecting Logic Usage

Features	Description
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.

Table 2-14: Features Affecting Memory Usage

Features	Description
Lane count	The lane count 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).

Features	Description
Receive FIFO buffer size	You can minimize memory usage by not adding significant amounts of margin to the minimum specified sizes.

SerialLite II IP Core Functional Description

3

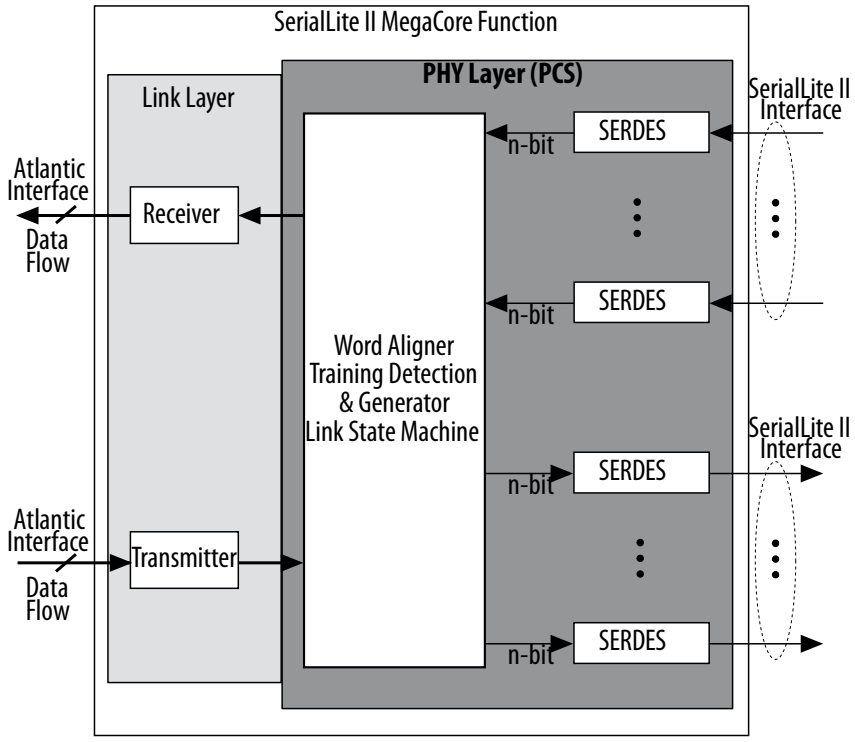
2019.01.09

UG-0705 Subscribe Send Feedback

The SerialLite II IP core consists of parameterized logic and a parameterized testbench.

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, and 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.



Related Information

SerialLite II IP Core Testbench on page 4-1

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.

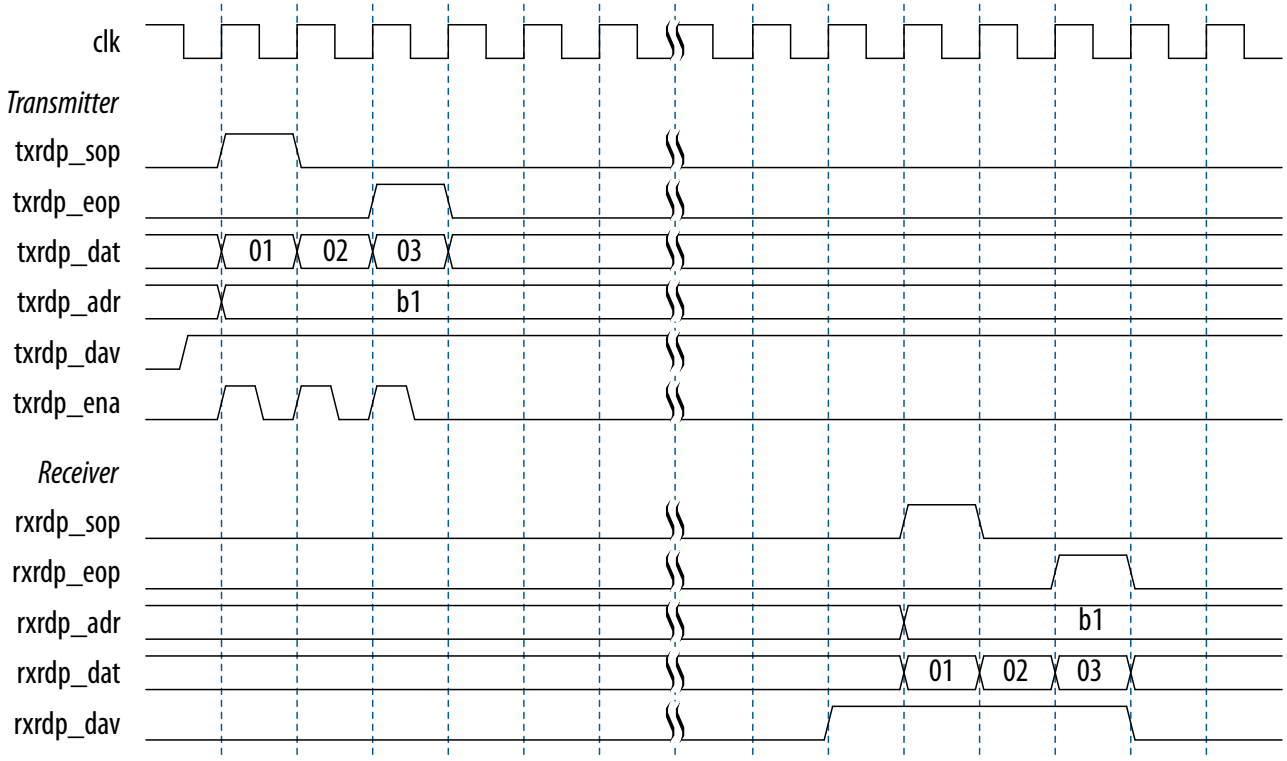
The SerialLite II IP core allows you to create one or two data ports: one for regular data and one for priority data.

- Each port has a full Atlantic interface.
- 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 IP core is an Atlantic interface master. The logic that drives data into the SerialLite II IP core or receives data from the SerialLite II IP core is referred as the system logic.

Figure 3-1: Transmitting and Receiving SerialLite II Data Packets

This timing diagram shows how the data packets are transmitted and received through the Atlantic interface.



On the transmitter side:

- The IP core sends the user input data packets to the Atlantic interface when 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:

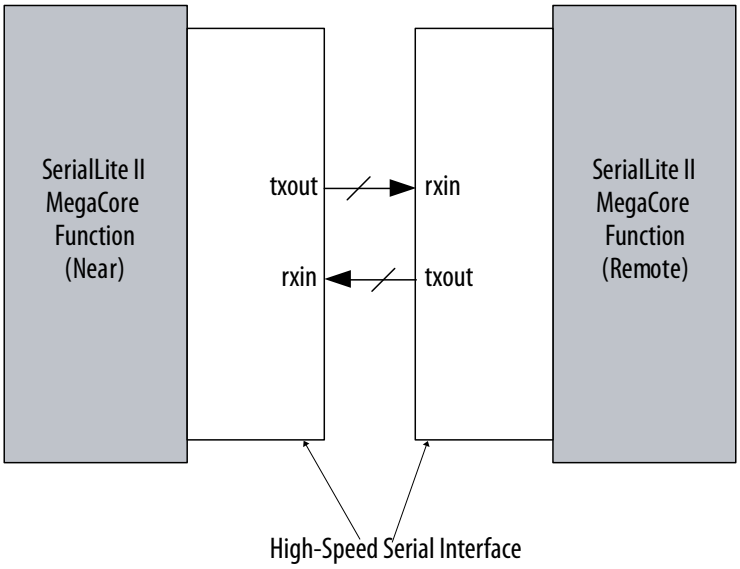
- The IP core transmits the data packets through the HSSI link and the data packets go through another SerialLite II IP core.
- In the other SerialLite II IP core, the same reverse processes are done in the SerialLite II data link layer and physical layer to strip off the framing and return the raw data 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.

Note: 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.

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.



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:

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

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

```
Core clock frequency = 3,125/(2×10) = 156.25 MHz
```

Aggregate Bandwidth

The bit rate specifies the rate of data transmission on a single lane.

In a multi-lane configuration, the total available bandwidth is the single-lane bit rate multiplied by the number of lanes. 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 × 10 (due to 8B/10B encoding). Thus for 4 lanes, the data bus size is equal to 40 bits (4×10 =40).

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

- For a 32-byte packet, given a link rate of 800 Mbps × 4 = 3.2 Gbps, the transfer equals:
  - data bits: 256
  - bits sent: 360
  - 256/360 × 3.2 = 2.276 Gbps
- For a 64-byte packet, given a link rate of 800 Mbps × 4 = 3.2 Gbps, the transfer equals:
  - data bits: 512
  - bits sent: 680
  - 512/680 × 3.2 = 2.409 Gbps
- For a 128-byte packet, given a link rate of 800 Mbps × 4 = 3.2 Gbps, the transfer equals:
  - data bits: 1, 024
  - bits sent: 1, 320
  - 1,024/1, 320 × 3.2 = 2.482 Gbps

External Clock Modes

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

- A synchronous configuration is used for a link where both ends are on the same board or on two boards driven by the same system clock.
- An asynchronous configuration is used when the two ends of the link are on different boards, each having its own independent clock source.

Figure 3-2: Synchronous Mode

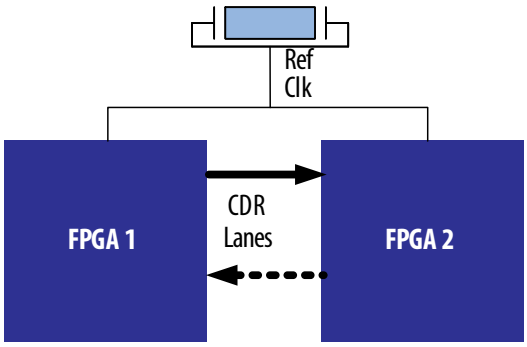
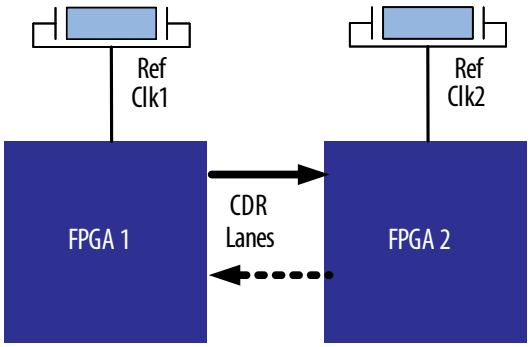


Figure 3-3: Asynchronous Mode



Internal Clocking Configurations

For Arria V, Intel Stratix 10, Arria 10, 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, the IP core generates a Tcl script ((<variation name>_ `constraints.tcl`). These settings are automatically written to your project directory when you run the generated Tcl script.

SerialLite II Deskew Support

You can use the SerialLite II deskew information to ensure trace length differences do not exceed the timing budget.

Table 3-1: SerialLite II Deskew Tolerance

The values include worst case lane–to–lane skew in the transceivers. To calculate in terms of time units, multiply the value in this table by the `tx_coreclock` clock period.

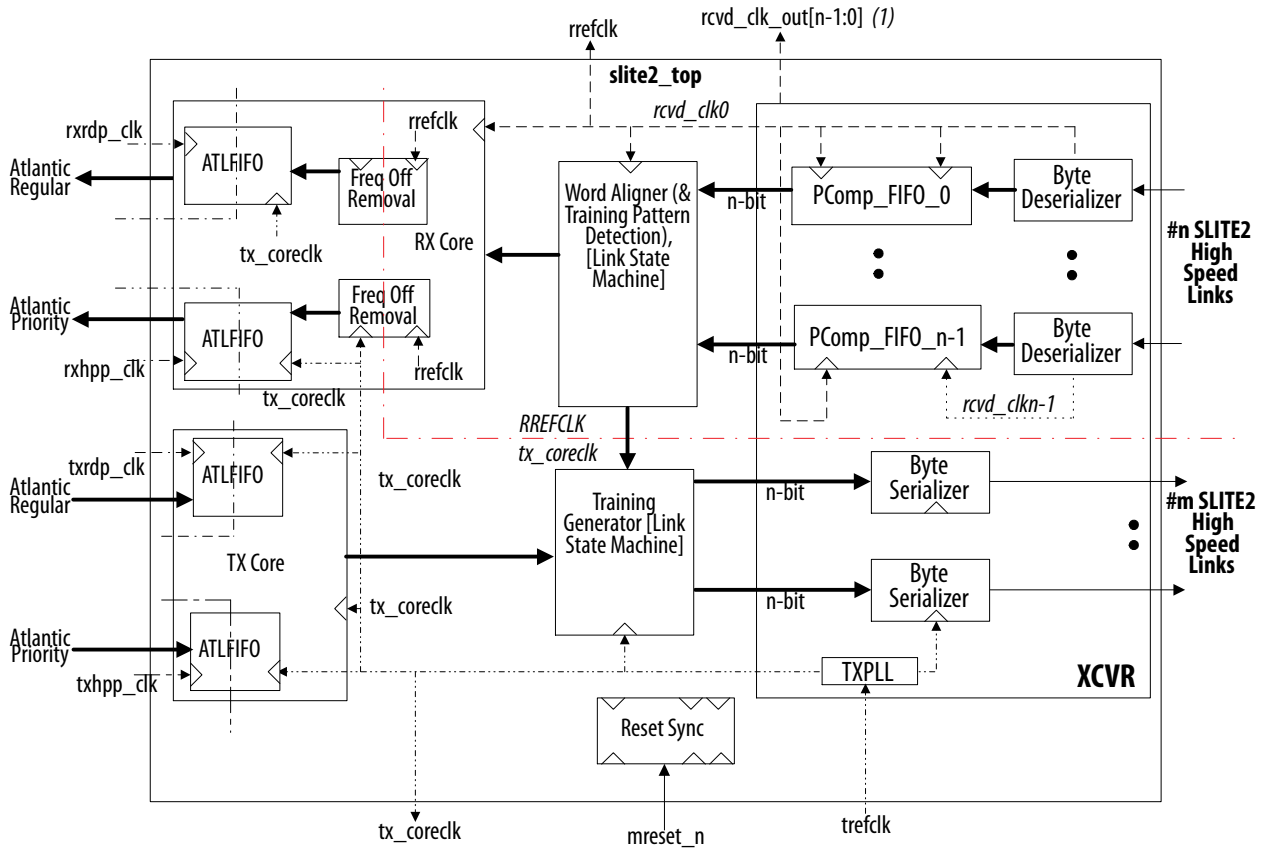
Transfer Size	Max Deskew (Cycles)
1	14
2	6
4	2

SerialLite II Clocking Structure

The SerialLite II IP core clock structures vary based on the configuration parameters.

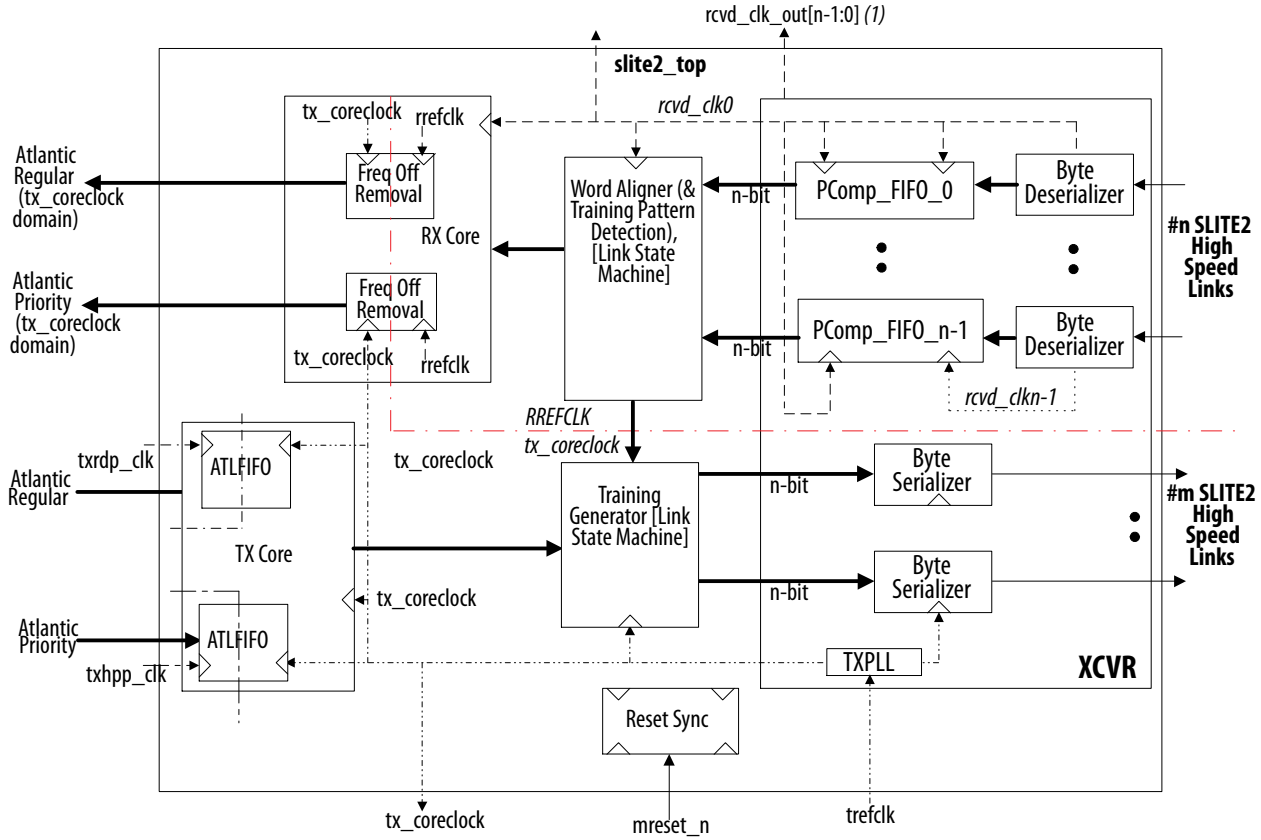
The following diagrams show the SerialLite II IP core clock structures, which vary based on the configuration parameters.

Figure 3-4: Full-Featured Clock Structure



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

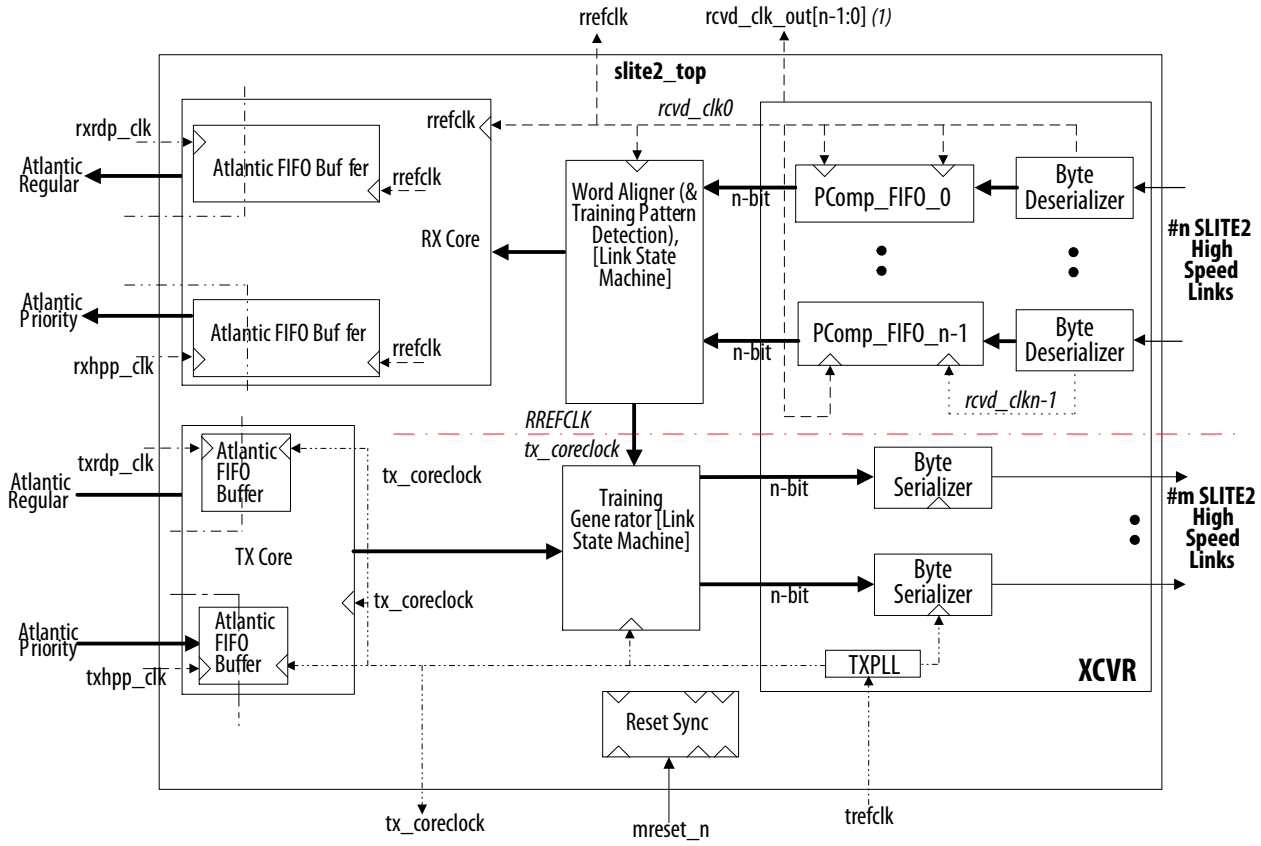
Figure 3-5: No Receiver FIFO Buffers Clock Structure



Note:

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

Figure 3-6: Full-Featured No Frequency Offset Clock Structure



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

Figure 3-7: No Receiver FIFO Buffers No Frequency Offset Clock Structure

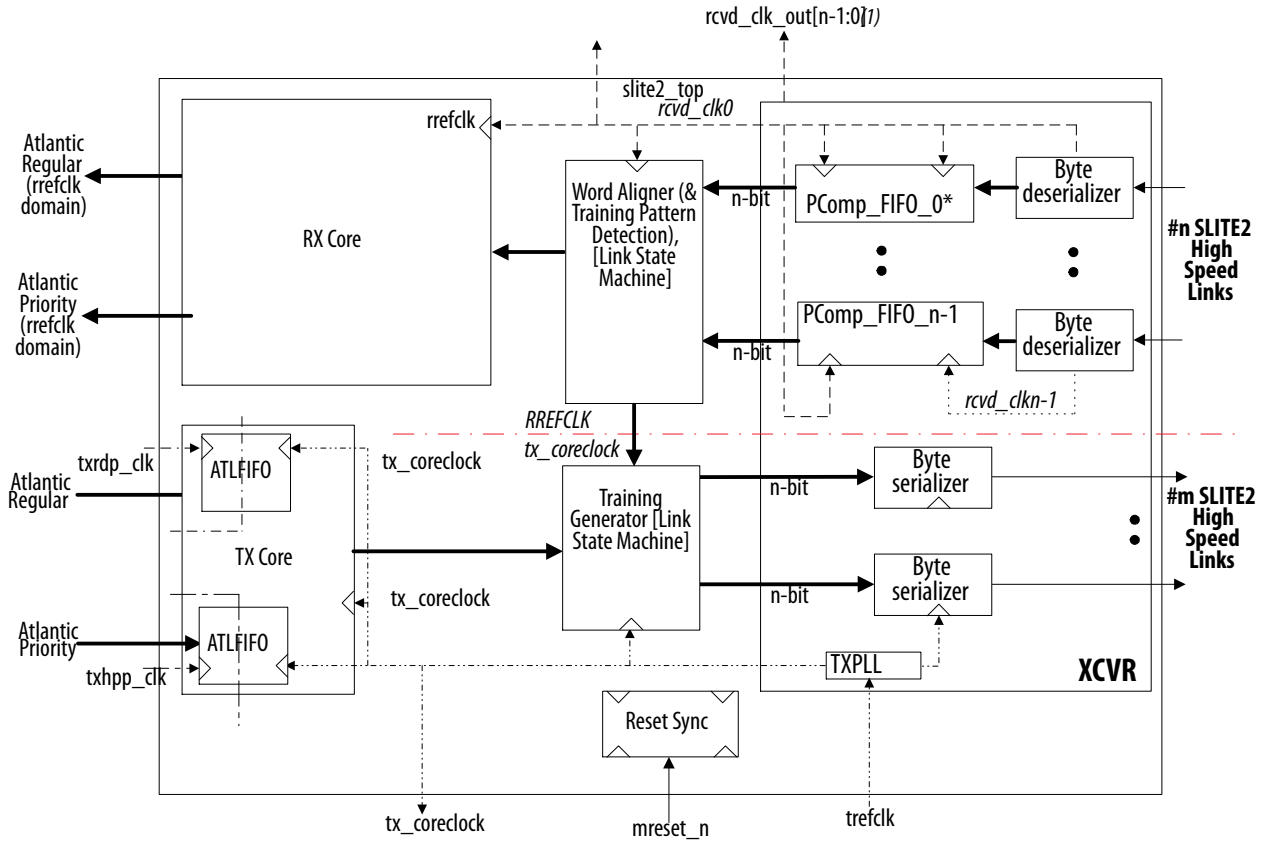
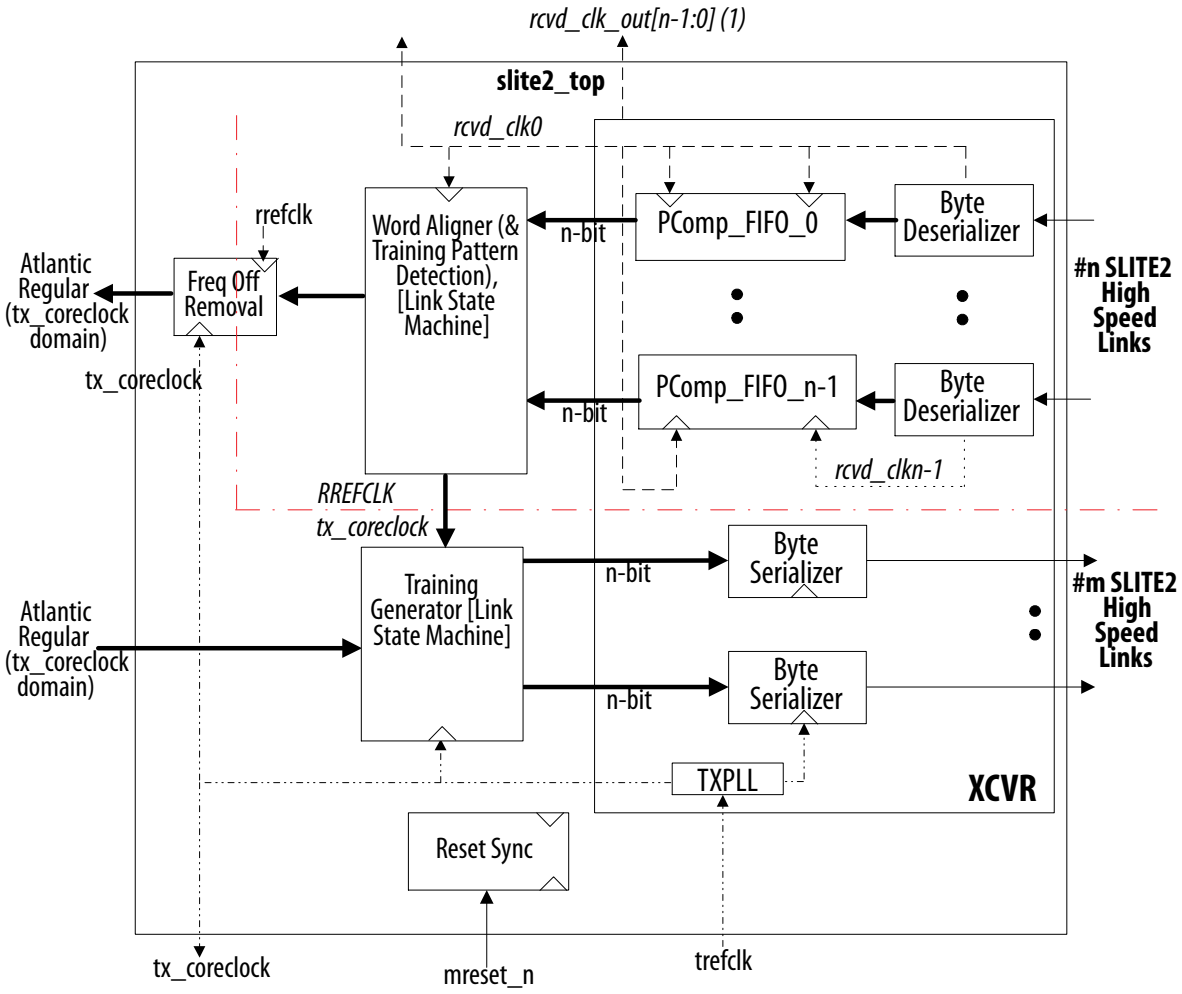


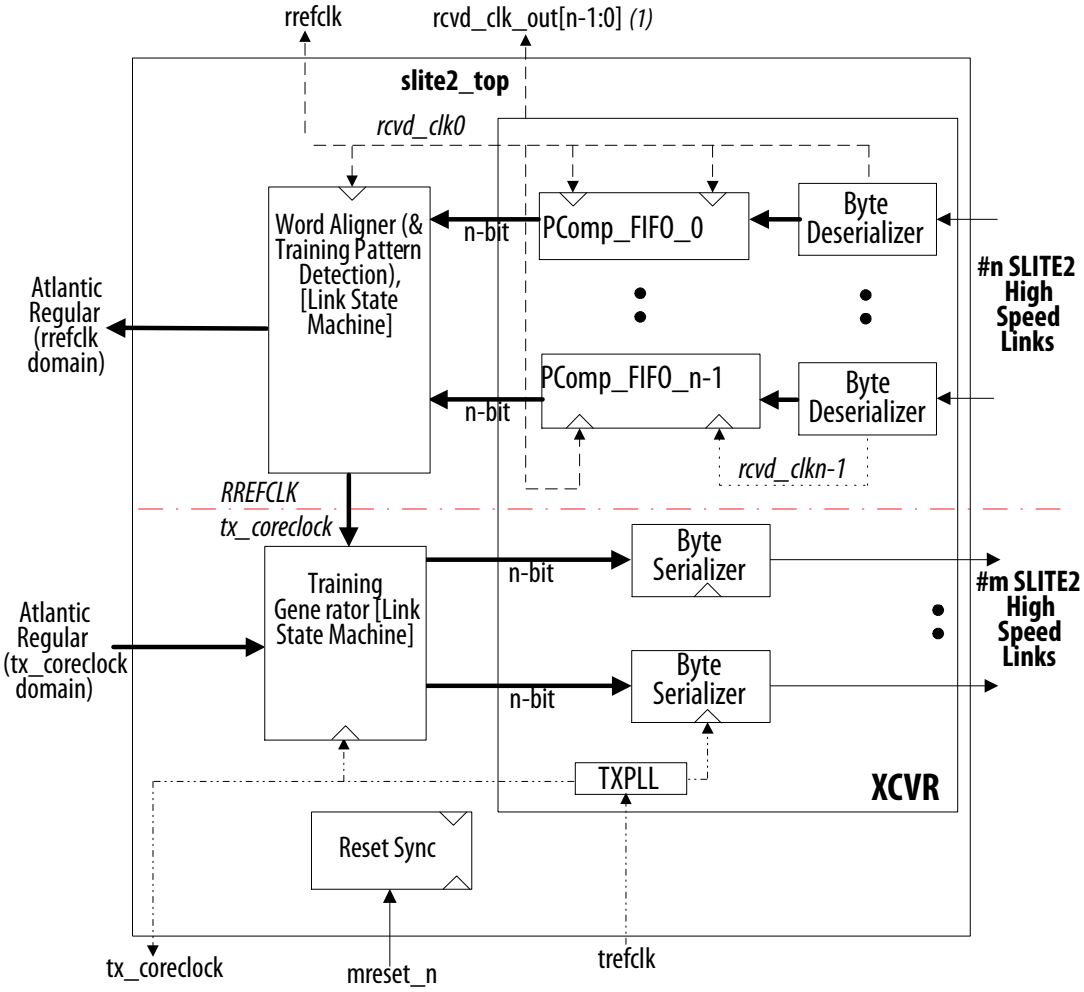
Figure 3-8: Streaming Full-Featured Clock Structure



Note:

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

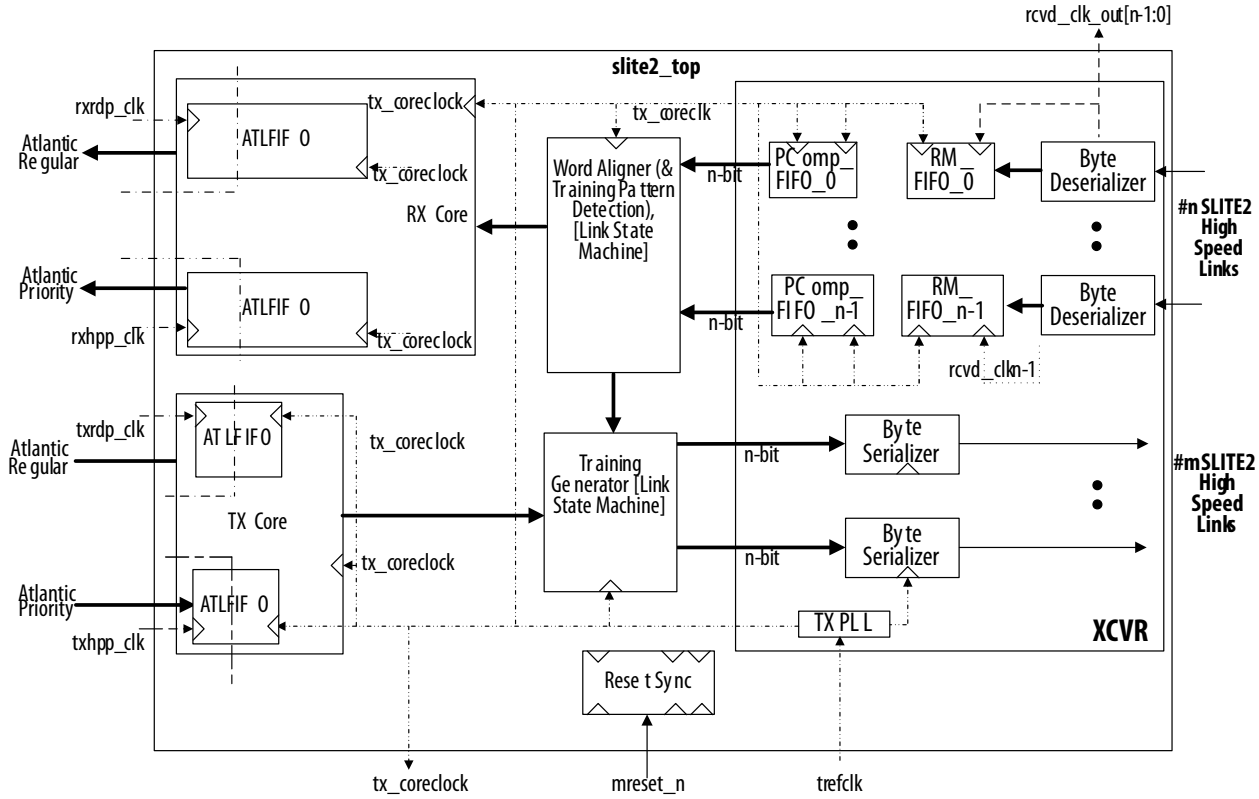
Figure 3-9: Streaming No Frequency Offset Clock Structure



Note:

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

Figure 3-10: Full Featured Clock Structure for 5G Symmetrical With TSIZE = 2



For Intel Stratix 10, Arria 10, 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_coreclkin`) 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_coreclkin`) of all RX channels (PHY IP).

SerialLite II Pin-Out Diagrams

Your SerialLite II IP core design always contains a PHY layer, based on the device you select.

The link layer portions is present if you set the **Data Type** option to **Packets**. The inclusion of receiver and transmitter components is determined by the **Port Type** option that you select: **Bidirectional**, **Transmitter only**, or **Receiver only**.

The following figures show some examples of pin-out diagrams.

Figure 3-11: Arria II GX/Stratix IV PHY Layer

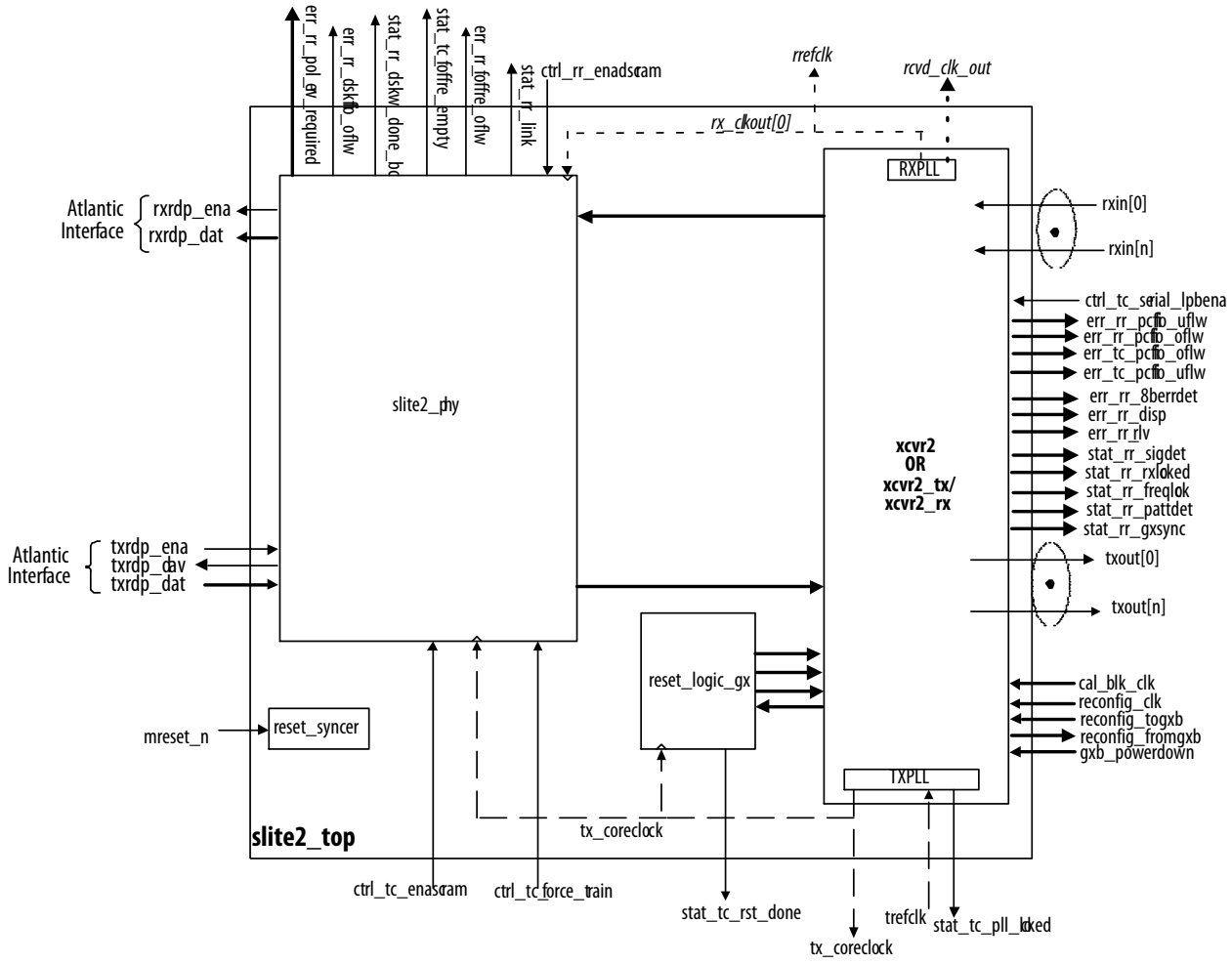
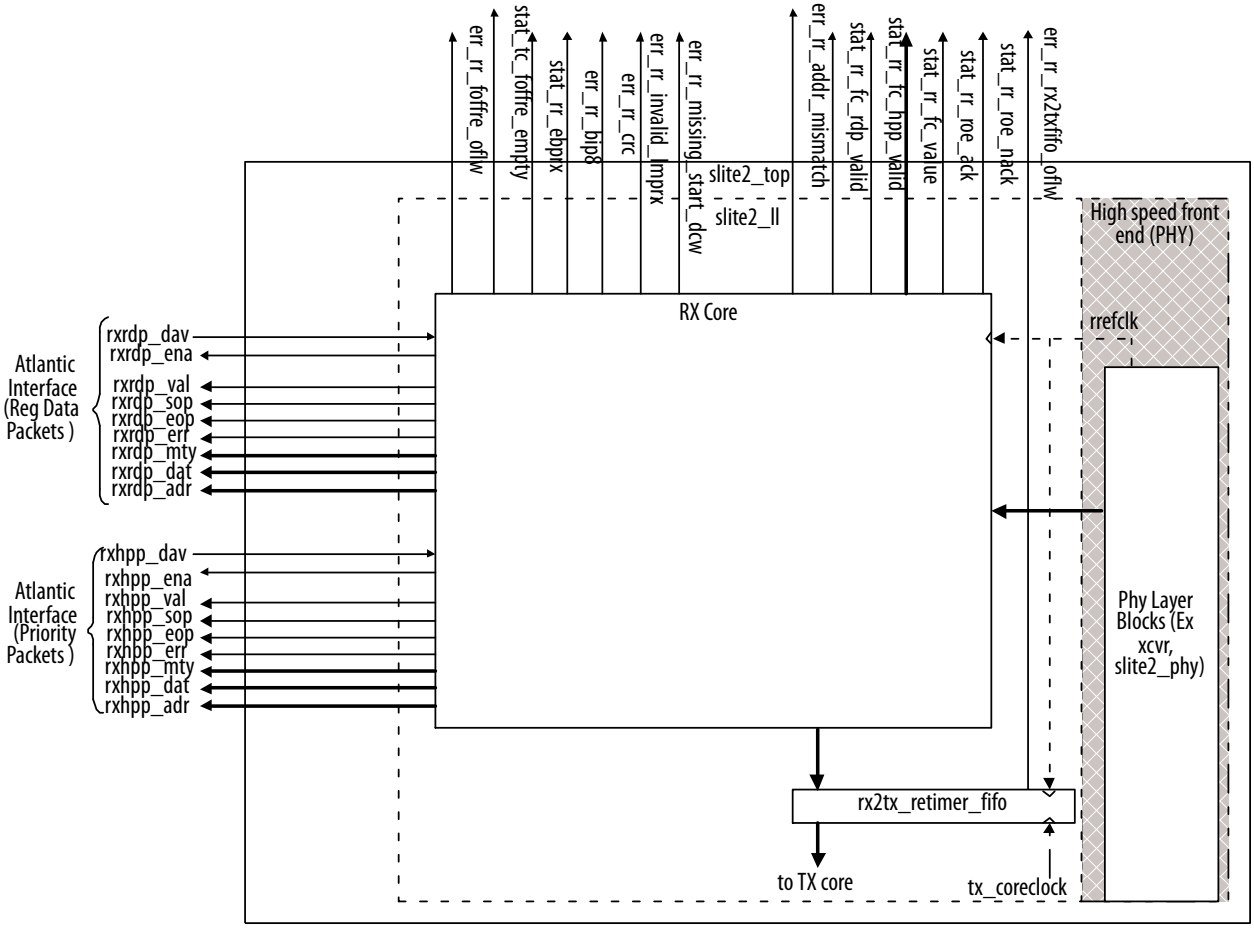


Figure 3-12: Receiver Layer With No FIFO



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

Figure 3-13: Receiver Link Layer With FIFO

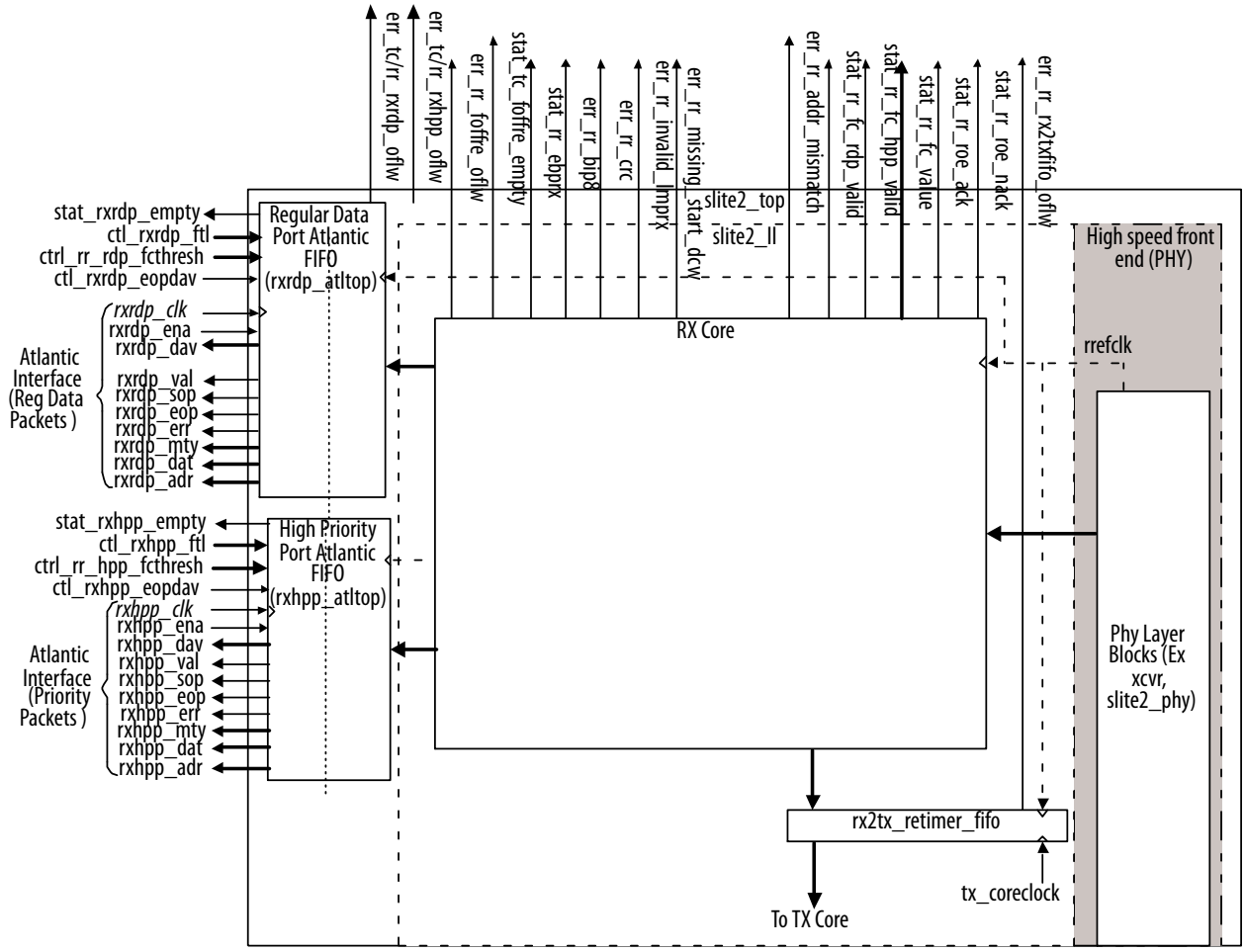
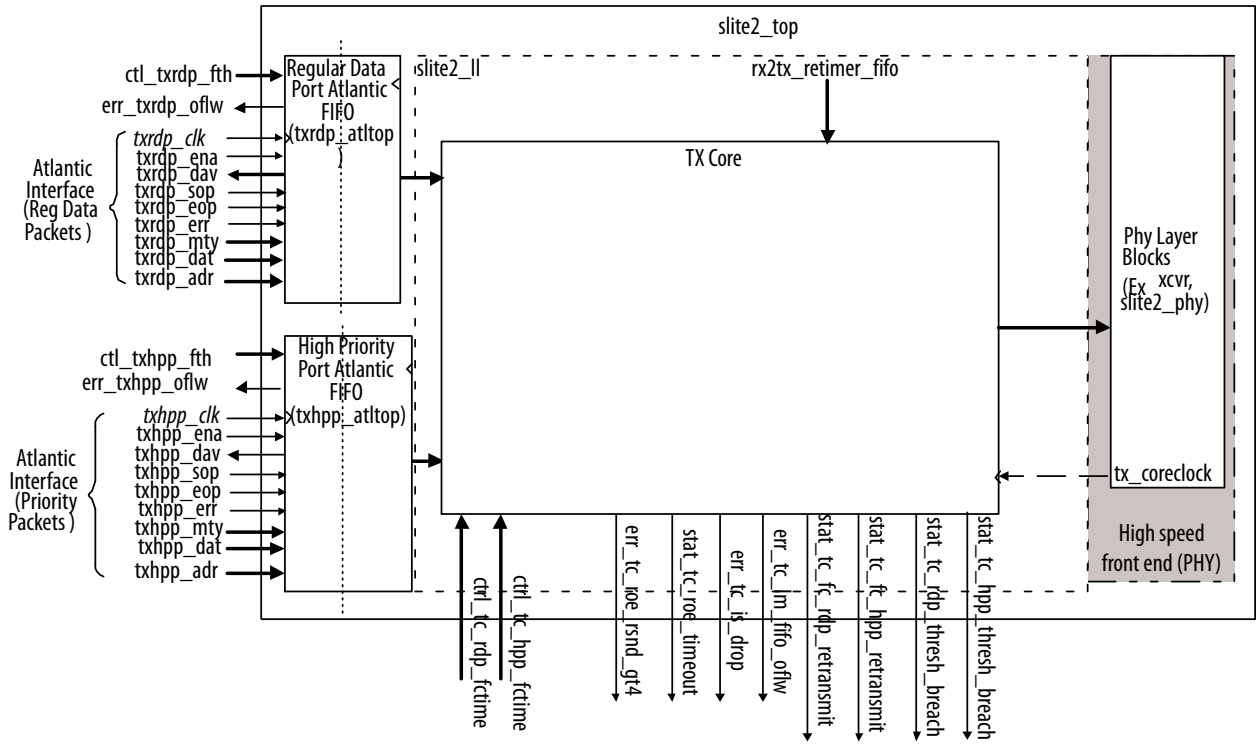


Figure 3-14: Transmitter Link Layer



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.

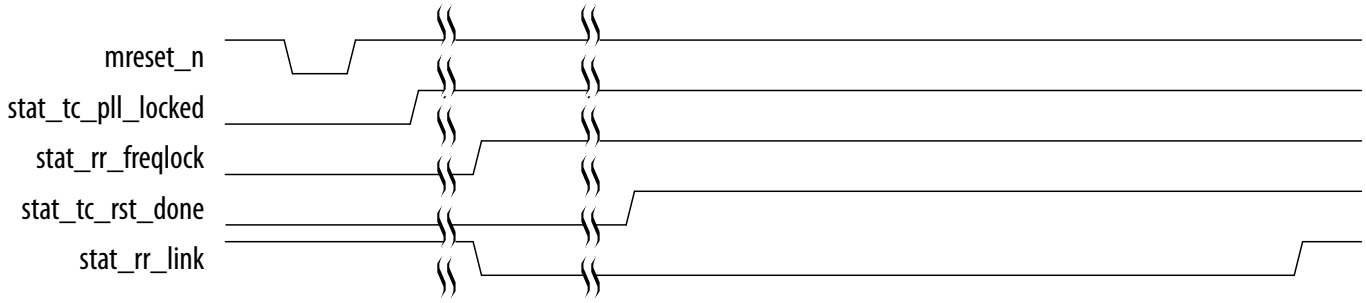
Note: This initialization and restart sequence is only applicable to Arria II GX and Stratix IV devices. For the later devices, refer to the respective *Device Handbooks*.

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 3-15: Initialization

This figure shows what happens when you initialize the SerialLite II IP core.



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 additional guidelines to create a working design.

- If you use the Tcl constraints to make assignments for the MegaCore functions, you must edit the Tcl script associated with each generated SerialLite II MegaCore function to update the 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: 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. Also 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.

IP Core Configuration for Intel Stratix 10, Arria 10, Arria V, Cyclone V, and Stratix V Devices

The supported features for the SerialLite II IP core in Intel Stratix 10, Arria 10, Arria V, Cyclone V, and Stratix V devices are the same with the Stratix IV GX devices, except for the hard transceiver features. Because there is no hard transceiver in this configuration, you need to instantiate the Custom PHY IP core and integrate both cores in your design.

Table 3-2: 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 10/ Arria V GZ/ Arria V GX/ Stratix V	Data Rate (Mbps) for Cyclone V
32 (TSIZE = 4)	• Word alignment mode: Manual (2)/ Automatic synchronization state machine (3) • Word alignment pattern: 10'h17c	3,126 to 6,375 (or higher for Intel Stratix 10 and Arria 10 devices)	3,126 to 5,000
	8B/10B encoder/decoder		
16 (TSIZE = 2)	• Word alignment mode: Automatic synchronization state machine • Word alignment pattern: 10'h17c	1,000 to 5,000	1,000 to 3,750
	8B/10B encoder/decoder		
8 (TSIZE = 1)	• Word alignment mode: Automatic synchronization state machine • Word alignment pattern: 10'h17c	622 to 2,500	622 to 1,875
	8B/10B encoder/decode		

Design Consideration

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

(2) Assert the `rx_enapatternalign` register in Custom PHY through the Avalon-MM interface to trigger another alignment when synchronization is lost.
(3) Applicable only for Intel Stratix 10, Arria 10, Arria V GZ, and Stratix V devices.

Table 3-3: Design Consideration

Considerations	Description
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. **Note:** The SISTER IP core is a SerialLite II IP core with parameters derived from the DUT parameters.
Simulation Support	The Quartus Prime 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_modelsim.tcl`) to enable the simulation to work in the ModelSim simulator.

Related Information

Testbench Specifications on page 4-2

Parameter Settings For SerialLite II and Custom PHY IP Cores

The parameters associated with the transceiver configuration in the SerialLite II IP core are disabled because there is no hard transceiver in this configuration. Other parameters for the SerialLite II IP core remains the same and are enabled.

Refer to **SerialLite II Parameter Settings** on page 2-6 for a more detailed description of the parameters.

Table 3-4: Custom PHY IP Core Settings

This table lists 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.

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

Option	Description	Setting
rx_ready output port	Indicates that the Custom PHY IP core is ready to receive data.	Required
Enable TX Bitslip	Provides control for bitslip functionality.	Off
Create rx_coreclkin port	Provides transceiver clock output to the rx_coreclk signal in the SerialLite II IP core. For Intel Stratix 10, Arria 10, Arria V, Cyclone V, and Stratix V designs with more than 1 channel, connect transceiver PHY rx_clkout(0) to rx_coreclkin (N-1:0).	Required
Create tx_coreclkin 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_coreclkin (N-1:0).	Required
Create rx_recovered_clk port	Provides a recovered clock output for the transceiver.	Off
Create ports	Provide the following ports: • `rx_is_lockedtoref` • `rx_is_lockedtodata` • `rx_signaldetect` • `tx_forceelecidle`	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 word aligner status ports	Provide the following ports: • `rx_syncstatus` • `rx_patterndetect` • `rx_bitslipboundaryselectout`	Required
Enable run length violation checking	Enables run length violation check to the `err_rr_rlv` signal in the SerialLite II IP core. **Note:** The `err_rr_rlv` signal is no longer exposed at the top level in the SerialLite II IP core for Intel Stratix 10, Arria 10, Arria V, Cyclone V, and Stratix V devices. Enable and monitor this signal from the transceiver.	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

Option	Description	Setting
Enable 8B/10B encoder/decoder	Provide the following ports: • `rx_runningdisp` • `rx_datak` (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 8B/10B status ports	Provide the following status ports for the 8B/10B encoder/decoder operation: • `rx_errdetect` • `rx_disperr` (provides running disparity status to the `err_rr_disp` signal in the SerialLite II IP core)	Required
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

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 3-5: New Interface Signals

Note: Some transceiver signals are removed due to the exclusion of hard transceiver in this configuration.

Signal	Direction	Width	Description
`rx_parallel_data_out`	Input	(Datapath width) × (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) × (Number of transmitter channels	Data output for the hard transmitter.
`tx_ctrlenable`	Output	(Number of control bits) × (Number of transmitter channels)	Control signal to indicate the control word in the `tx_parallel_data_in` signal.
`tx_coreclk`	Input	1	Clock input from the hard transmitter.

Signal	Direction	Width	Description
rx_ctrldetect	Output	(Number of control bits) × (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) × (Number of receiver channels)	Pattern detect output for the hard transceiver.
err_rr_disp	Input	(Number of control bits) × (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.
err_rr_8berrdet	Input	(Number of control bits) × (Number of receiver channels)	Shows 8B/10B errors from the transceiver.

SerialLite II 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 3-6: High-Speed Serial Interface Signals

Signal	Direction	Clock Domain	Description
rxin [*n*-1] *n* = RX number of lanes	Output	–	SerialLite II differential receive data bus. Bus carries packets, cells, or in-band control words. **Note:** This signal is removed in configurations targeted for Intel Stratix 10, Arria 10, Arria V, Cyclone V, and Stratix V devices due to the exclusion of hard transceivers.
txout[*m*-1] *m* = TX Number of lanes	Output	–	SerialLite II differential transmit data bus. Bus carries packets, cells, or in-band control words. **Note:** This signal is removed in configurations targeted for Intel Stratix 10, Arria 10, Arria V, Cyclone V, and Stratix V devices due to the exclusion of hard transceivers.

Signal	Direction	Clock Domain	Description
rrefclk	Output	rrefclk	Receive core output PLL-derived clock. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed. For example, rrefclk0 is the signal from SerialLite II receiver block 0.
trefclk	Input	trefclk	Reference clock used to drive the transmitter PLL. The PLL is used to generate the transmit core clock (tx_coreclock). **Note:** This signal is removed in configurations targeted for Intel Stratix 10, Arria 10, Arria V, Cyclone V, and Stratix V devices due to the exclusion of hard transceivers.
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.
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 Intel Stratix 10, Arria 10, 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 trefclk.
stat_rr_link	Output	rrefclk	Link Status. When high, the link is enabled. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed. For example, stat_rr_link0 is the signal from SerialLite II receiver block 0.

Table 3-7: Transceiver Signals

rxnl is the receive number of lanes;

txnl is the transmit number of lanes;

srx is the transfer size × the receive number of lanes.

Signal	Direction	Clock Domain	Description
`ctrl_tc_serial_lpbena`	Input	`tx_coreclock`	Serial Loopback (`TXOUT` internally connected to `RXIN`). Tie signal to `1'b0` to not use loopback and tie to `1'b1` to use serial loopback.
`rcvd_clk_out` [*rxnl*-1:0]	Output	–	Per lane recovered clock. **Note:** Not applicable for Arria V, Cyclone V, and Stratix V devices.
`err_rr_8berrdet` [*srx*-1:0]	Output/ Input	`rrefclk`	8B/10B error detection signal. **Note:** Output port for Arria II GX and Stratix IV devices; input port for Arria V, Cyclone V, and Stratix V devices.
`err_rr_disp` [*srx*-1:0]	Output/ Input	`rrefclk`	Disparity error detection signal. **Note:** Output port for Arria II GX and Stratix IV devices; input port for Arria V, Cyclone V, and Stratix V devices.
`err_rr_pcfifo_uflw` [*rxnl*-1:0]	Output	`rrefclk`	Interface/phase compensation FIFO buffer underflow signal (Arria II GX and Stratix IV devices only).
`err_rr_pcfifo_oflw` [*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. (Arria II GX and Stratix IV devices only)
`err_tc_pcfifo_uflw` [*txnl*-1:0]	Output	`tx_coreclock`	Interface/phase compensation FIFO buffer underflow signal. **Note:** This signal is removed in configurations targeted for Arria V and Stratix V devices due to the exclusion of hard transceivers.

Signal	Direction	Clock Domain	Description
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) **Note:** This signal is removed in configurations targeted for Arria V and Stratix V devices due to the exclusion of hard transceivers.
stat_rr_gxsync[*srx*-1:0]	Output	rrefclk	Gives the status of the pattern detector and word aligner. **Note:** This signal is removed in configurations targeted for Arria V and Stratix V devices due to the exclusion of hard transceivers. If needed, you can use the rx_syncstatus signal generated when you create optional word aligner status port during the Custom PHY generation.
stat_rr_rxlocked [*rxnl*-1:0]	Output	rrefclk	Receiver PLL locked signal. Indicates whether or not the receiver PLL is phase locked to the CRU reference clock. When the PLL locks to data, which happens some time after the transceiver's rx_freqlocked signal is asserted high, this signal has little meaning because it only indicates lock to the reference clock. This signal is active high for Arria II GX and Stratix IV devices. **Note:** This signal is removed in configurations targeted for Arria V and Stratix V devices due to the exclusion of hard transceivers. If needed, you can use the rx_is_lockedtoref signal generated when you create additional ports during the Custom PHY instantiation.

Signal	Direction	Clock Domain	Description
`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. **Note:** This signal is removed in configurations targeted for Intel Stratix 10, Arria 10, Arria V, Cyclone V, and Stratix V devices due to the exclusion of hard transceivers. If needed, you can use the `rx_is_lockedtodata` signal generated when you create additional ports during the Custom PHY instantiation.
`stat_rr_pattdet` [*srx*-1:0]	Output/ Input	`rrefclk`	Pattern detection signal **Note:** Output port for Arria II GX and Stratix IV devices; input port for Intel Stratix 10, Arria 10, Arria V, Cyclone V, and Stratix V devices.
`reconfig_fromgxb ]` Arria II GX/Stratix IV GX: [*recon_quad*`*17-1:0]`	Output	`reconfig_clk`	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 cancellation. **Note:** `recon_quad` is the total number of Quads being used. 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. **Note:** This signal is removed in configurations targeted for Intel Stratix 10, Arria 10, Arria V, Cyclone V, and Stratix V devices due to the exclusion of hard transceivers.

Signal	Direction	Clock Domain	Description
reconfig_togxb Arria II GX/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 cancellation. **Note:** This signal is removed in configurations targeted for Intel Stratix 10, Arria 10, Arria V, Cyclone V, and Stratix V devices due to the exclusion of hard transceivers.
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 cancellation. **Note:** This signal is removed in configurations targeted for Intel Stratix 10, Arria 10, Arria V, Cyclone V, and Stratix V devices due to the exclusion of hard transceivers.
cal_blk_clk	Input	–	Calibration clock for the termination resistor calibration block. The frequency range of cal_blk_clk is 10 to 125 MHz. **Note:** This signal is removed in configurations targeted for Intel Stratix 10, Arria 10, Arria V, Cyclone V, and Stratix V devices due to the exclusion of hard transceivers.
gxb_powerdown	Input	–	Transceiver block reset and power down. This signal resets and powers down all circuits in the transceiver block. This does not affect the refclk buffers and reference clock lines. All the gxb_powerdown input signals of cores placed in the same transceiver block should be tied together. The gxb_powerdown signal should be tied low or should remain asserted for at least 2 ms whenever it is asserted. **Note:** This signal is removed in configurations targeted for Intel Stratix 10, Arria 10, Arria V, Cyclone V, and Stratix V devices due to the exclusion of hard transceivers.

Table 3-8: Atlantic Interface Signals

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

Note: 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.

Signal	Direction	Clock Domain	Description
`rxrdp_clk`	Input	–	Atlantic receive regular data port clock. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.
`txrdp_clk`	Input	–	Atlantic transmit regular data port clock.
`rxhpp_clk`	Input	–	Atlantic receive high priority port clock. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.
`txhpp_clk`	Input	–	Atlantic transmit high priority port clock.
`rxrdp_ena`	Input	`rxrdp_clk`	Enable signal on the Atlantic interface. Indicates that the data is to be read on the next clock cycle. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.
`rxrdp_dav`	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. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.
`rxrdp_dav`	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. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.
`rxrdp_val`	Output	`rxrdp_clk`	The output data is valid. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.

Signal	Direction	Clock Domain	Description
rxrdp_sop	Output	rxrdp_clk	Start of packet indicator on the Atlantic interface. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.
rxrdp_eop	Output	rxrdp_clk	End of packet indicator on the Atlantic interface. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.
rxrdp_err	Output	rxrdp_clk	Error indicator on the Atlantic Interface. This signal is not necessarily held high until rxrdp_eop is asserted. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.
rxrdp_mty[*m*-1:0]	Output	rxrdp_clk	Number of empty bytes in the data word. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed. **Note:** *d* is the empty value, which is log2 (data width).
rxrdp_dat[*d*-1:0]	Output	rxrdp_clk	User data bits. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed. **Note:** *m* is the data width, which is 8 × transfer size × the RX number of lanes.
rxrdp_adr[7:0]	Output	rxrdp_clk	User-defined packet ID. Only valid with rxrdp_sop. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.
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.

Signal	Direction	Clock Domain	Description
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]	Input	txrdp_clk	Number of empty bytes in the data word. **Note:** *tm* is the empty value, which is log2 (data width).
txrdp_dat[*td*-1:0]	Input	txrdp_clk	User data bits. **Note:** *td* is the data width, which is 8 × transfer size × the TX number of lanes.
txrdp_adr[7:0]	Input	txrdp_clk	User-defined packet ID.
rxhpp_ena	Input	rxhpp_clk	Enable signal on the Atlantic interface. Indicates that the data is to be read on the next clock cycle. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.
rxhpp_dav	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. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.
rxhpp_dav	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. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.
rxhpp_val	Output	rxhpp_clk	The output data is valid. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.
rxhpp_sop	Output	rxhpp_clk	Start of packet indicator on the Atlantic interface. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.

Signal	Direction	Clock Domain	Description
rxhpp_eop	Output	rxhpp_clk	End of packet indicator on the Atlantic interface. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.
rxhpp_err	Output	rxhpp_clk	Error indicator on the Atlantic Interface. This signal is not necessarily held high until rxhpp_eop is asserted. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.
rxhpp_mty [*m*-1:0]	Output	rxhpp_clk	Number of empty bytes in the data word. **Note:** In broadcast mode, these signals will have the corresponding receiver function number post-fixed. **Note:** *m* is the empty value, which is log2 (data width).
rxhpp_dat [*d*-1:0]	Output	rxhpp_clk	User data bits. **Note:** In broadcast mode, these signals will have the corresponding receiver function number post-fixed. **Note:** *d* is the data width, which is 8 × transfer size × the RX number of lanes.
rxhpp_adr[3:0]	Output	rxhpp_clk	User-defined packet ID. Only valid with rxhpp_sop. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.
txhpp_ena	Input	txhpp_clk	Enable signal on the Atlantic interface. Indicates that the data is valid.
txhpp_dav	Output	txhpp_clk	Indicates that the input FIFO buffer is not full.
txhpp_sop	Input	txhpp_clk	Start of packet indicator on the Atlantic interface.
txhpp_eop	Input	txhpp_clk	End of packet indicator on the Atlantic interface.
txhpp_err	Input	txhpp_clk	Error indicator on the Atlantic interface.

Signal	Direction	Clock Domain	Description
txhpp_mty [*tm*-1:0]	Input	txhpp_clk	Number of empty bytes in the data word. **Note:** *tm* is the empty value, which is log2 (data width).
txhpp_dat [*td*-1:0]	Input	txhpp_clk	User data bits. **Note:** *td* is the data width, which is 8 × transfer size × the TX number of lanes.
txhpp_adr[3:0]	Input	txhpp_clk	User-defined packet ID.

Table 3-9: Atlantic Interface Signals for Streaming Mode

Signal	Direction	Clock Domain	Description
rxrdp_dat [*d*-1:0]	Output	rrefclk	Received user data bits. **Note:** *d* is = FIFO SIZE / (TSIZE * RX Number of lanes). **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.
rxrdp_ena	Output	rrefclk	Enable signal on the Atlantic interface. Indicates that the data is valid on the current clock cycle. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.
txrdp_dat [*td*-1:0]	Input	tx_coreclock	User data bits to be transmitted. **Note:** *td* is = FIFO SIZE / (TSIZE * TX Number of lanes).
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 **Enable frequency offset tolerance** is not turned on, this signal will always be high when the link is up.

Table 3-10: Protocol Processor's Error, Status and Control Signals

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 **Enable frequency offset tolerance** is turned off (regular data port).
err_rr_rxhpp_oflw	Output	rrefclk	Indicates that the Atlantic FIFO buffer has overflowed and data has been lost when **Enable frequency offset tolerance** is turned off (priority data port).
err_tc_rxrdp_oflw	Input	tx_coreclock	Indicates that the Atlantic FIFO buffer has overflowed and data has been lost when **Enable frequency offset tolerance** is turned on (regular data port).
err_tc_rxhpp_oflw	Input	tx_coreclock	Indicates that the Atlantic FIFO buffer has overflowed and data has been lost when **Enable frequency offset tolerance** is turned on (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		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		rxdrp_clk	Indicates that the internal Atlantic FIFO buffer is empty, and the read request is ignored. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.
stat_rxhpp_empty		rxhpp_clk	Indicates that the internal Atlantic FIFO buffer is empty, and the read request is ignored. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.

Signal	Direction	Clock Domain	Description
ctl_rxhpp_ftl [*n*-1:0])		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. **Note:** *n* is = FIFO SIZE / (TSIZE * RX Number of lanes).
ctl_rxrdp_ftl [*n*-1:0]		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. **Note:** *n* is = FIFO SIZE / (TSIZE * RX Number of lanes).
ctl_rxhpp_eopdav		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. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.
ctl_rxrdp_eopdav		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. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.
ctl_txhpp_fth [*tn*-1:0]		txhpp_clk	Transmit high priority port FIFO buffer threshold high dav control. **Note:** *tn* is = FIFO SIZE / (TSIZE * TX Number of lanes).
ctl_txrdp_fth [*tn*-1:0]		txrdp_clk	Transmit regular data port FIFO buffer threshold high dav control. **Note:** *tn* is = FIFO SIZE / (TSIZE * TX Number of lanes).

Table 3-11: Troubleshooting Signals

These signals do not necessarily need to be connected to external logic. In general, they are for diagnostic purposes. Some signals are only available in certain configurations.

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. **Note:** Not applicable for Intel Stratix 10, Arria 10, Arria V, Cyclone V, and Stratix V devices because the transceiver is generated independently. For these devices, you can find out if the transceiver is ready by polling on the `tx_ready` or `rx_ready` signals from the Custom PHY IP core.
`err_rr_foffre_oflw`	Output	`rrefclk`	Indicates that frequency offset tolerance FIFO buffer has overflowed. The link restarts. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.
`stat_tc_foffre_empty`	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. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.
`stat_rr_ebprx`	Output	`rrefclk`	Indicates that an end of bad packet character was received. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.
`err_rr_bip8`	Output	`rrefclk`	Indicates that a BIP-8 error was detected in the received link management packet. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.

Signal	Direction	Clock Domain	Description
err_rr_crc	Output	rrefclk	Indicates that a CRC error was detected in the received segment/packet. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.
err_rr_fcrx_bne	Output	rrefclk	Indicates that a flow control link management packet was received, but flow control is not enabled. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.
err_rr_roerx_bne	Output	rrefclk	Indicates that a retry-on-error link management packet was received, but **Retry-on-error** parameter is not enabled. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.
err_rr_invalid_lmprx	Output	rrefclk	Indicates that an invalid link management packet was received. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.
err_rr_missing_start_dcw	Output	rrefclk	Indicates that data byte(s) received, but a start of data control word (DCW) is missing. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.
err_addr_mismatch	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. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.

Signal	Direction	Clock Domain	Description
err_rr_pol_rev_required	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. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.
err_rr_dskfifo_oflw	Output	rrefclk	Indicates that deskew FIFO buffer has overflowed. Link restarts. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.
stat_rr_dskw_done_bc	Output	rrefclk	Indicates that a bad column was received after successful deskew completion. Link is restarted. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.
stat_tc_rdp_thresh_breach	Output	rrefclk	Indicates that the receiver regular data port FIFO buffer is breached, transmit flow control link management packet. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.
stat_tc_hpp_thresh_breach	Output	tx_coreclock	Indicates that the receiver priority data port FIFO buffer is breached, transmit flow control link management packet. **Note:** In broadcast mode, this signal will have the corresponding receiver function number post-fixed.
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.

Signal	Direction	Clock Domain	Description
stat_tc_roe_timeout	Output	tx_coreclock	**Retry-on-error** only: Indicates that the transmitter 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.

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.

SerialLite II IP Core Testbench

4

2019.01.09

UG-0705

The SerialLite II IP core testbench helps you to verify your design implementation.

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 you can use it with any Verilog HDL or VHDL simulator. You can run the testbench in the standard edition (SE) or the Altera edition (AE) of the ModelSim software.

- 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.

Note: For Intel Stratix 10, Arria 10, Arria V, Cyclone V, and Stratix V configurations, you are required to edit the dynamically generated testbench to include the Custom PHY IP core instantiation. You also need to update the generated <*variant_name*>`_run_modelsim.tcl` to include the Custom PHY transceiver files.

Related Information

IP Core Configuration for Intel Stratix 10, Arria 10, Arria V, Cyclone V, and Stratix V Devices on page 3-18

Testbench Files

The Quartus Prime software generates the testbench files when you create a SerialLite IP core variation.

The Verilog HDL demonstration testbench and associated scripts are generated automatically when you create a SerialLite II IP core variation.

The VHDL demonstration testbench and the scripts to run it are generated when you create a SerialLite II 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`

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

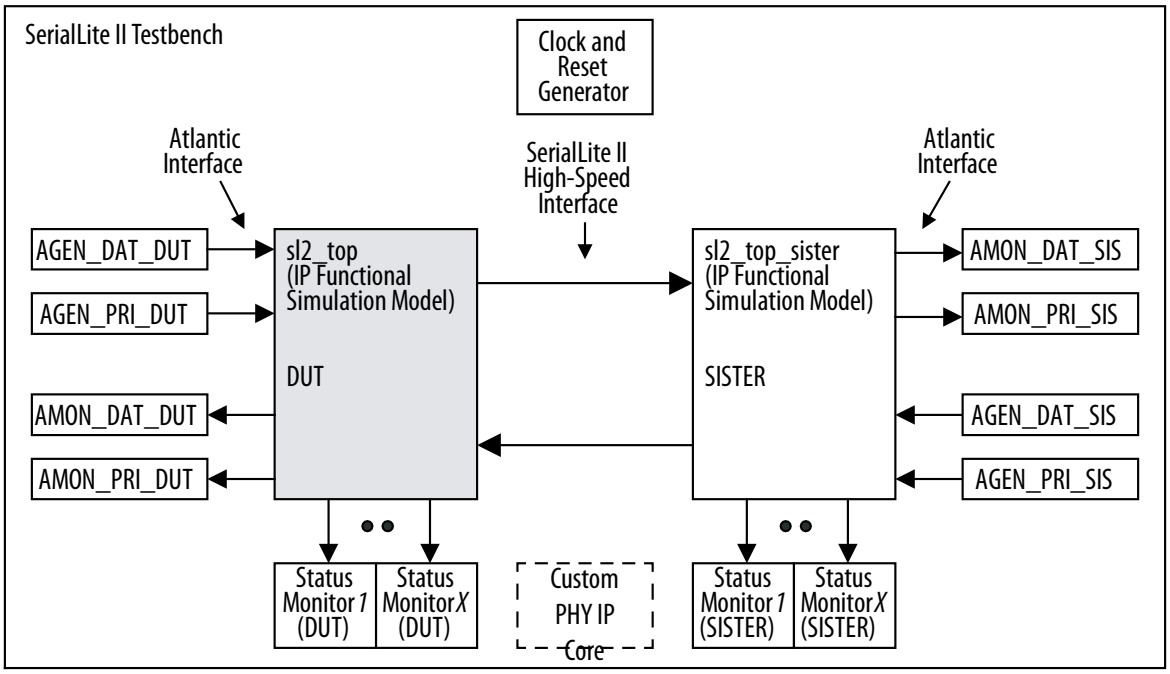
Testbench Specifications

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.

Figure 4-1: SerialLite II Testbench Environment (Non-Broadcast)



Notes:
_DAT = Regular Data Port
_PRI = High Priority Port
_DUT = Refers to the DUT side
_SIS = Refers to the SISTER side

Note: 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. However, 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.

The testbench environment (tb) 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.

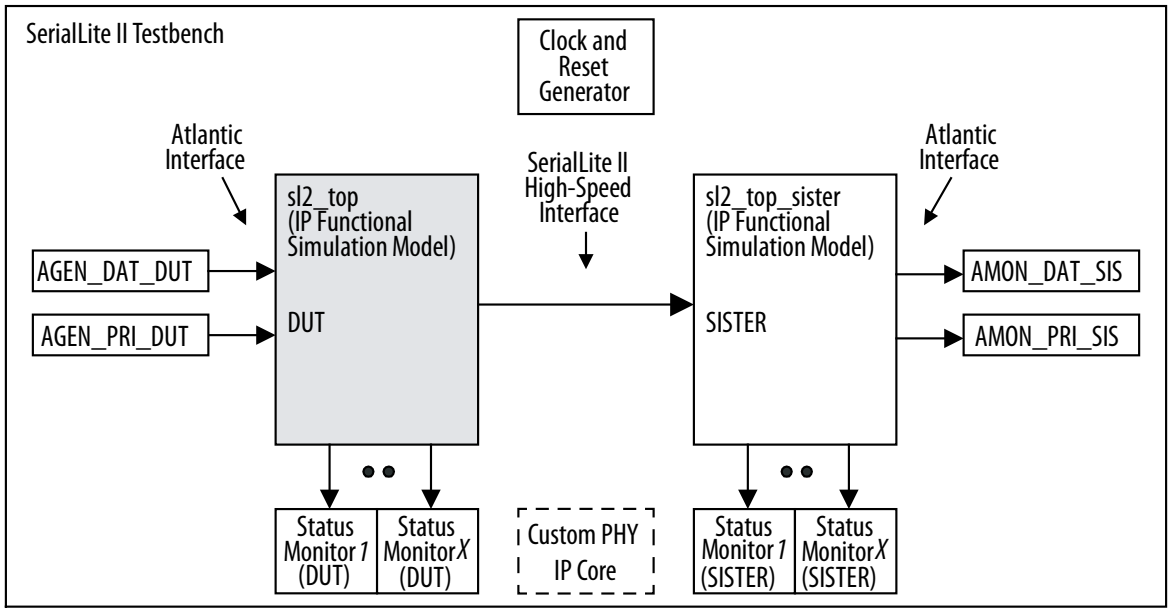
For a broadcast DUT, there are multiple SISTER instantiations. Pin monitor utilities monitor the SerialLite II status and error pins of the DUT and SISTER(s).

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

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 4-2: SerialLite II Testbench Environment (Single Mode–Transmitter Only, Verilog HDL Only, Non-Broadcast)

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

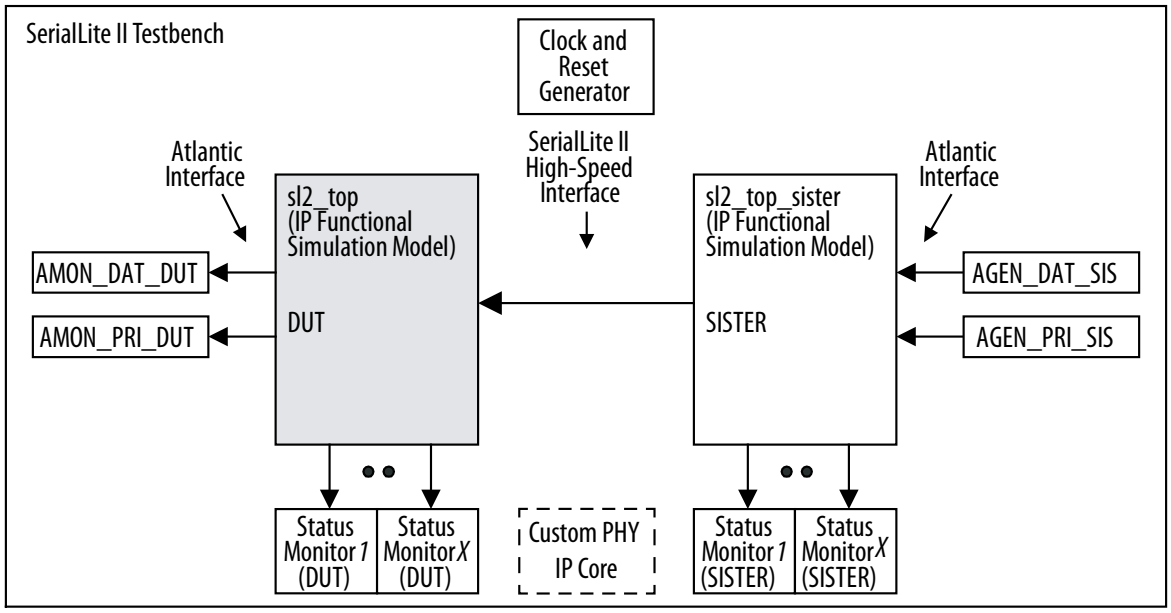


Notes:
_DAT = Regular Data Port
_PRI = High Priority Port
_DUT = Refers to the DUT side
_SIS = Refers to the SISTER side

Note: The DUT and the SISTER IP cores may have different parameters; depending on the DUT parameters, and some components may be missing.

Figure 4-3: SerialLite II Testbench Environment (Single Mode–Receiver Only, Verilog HDL Only, Non-Broadcast)

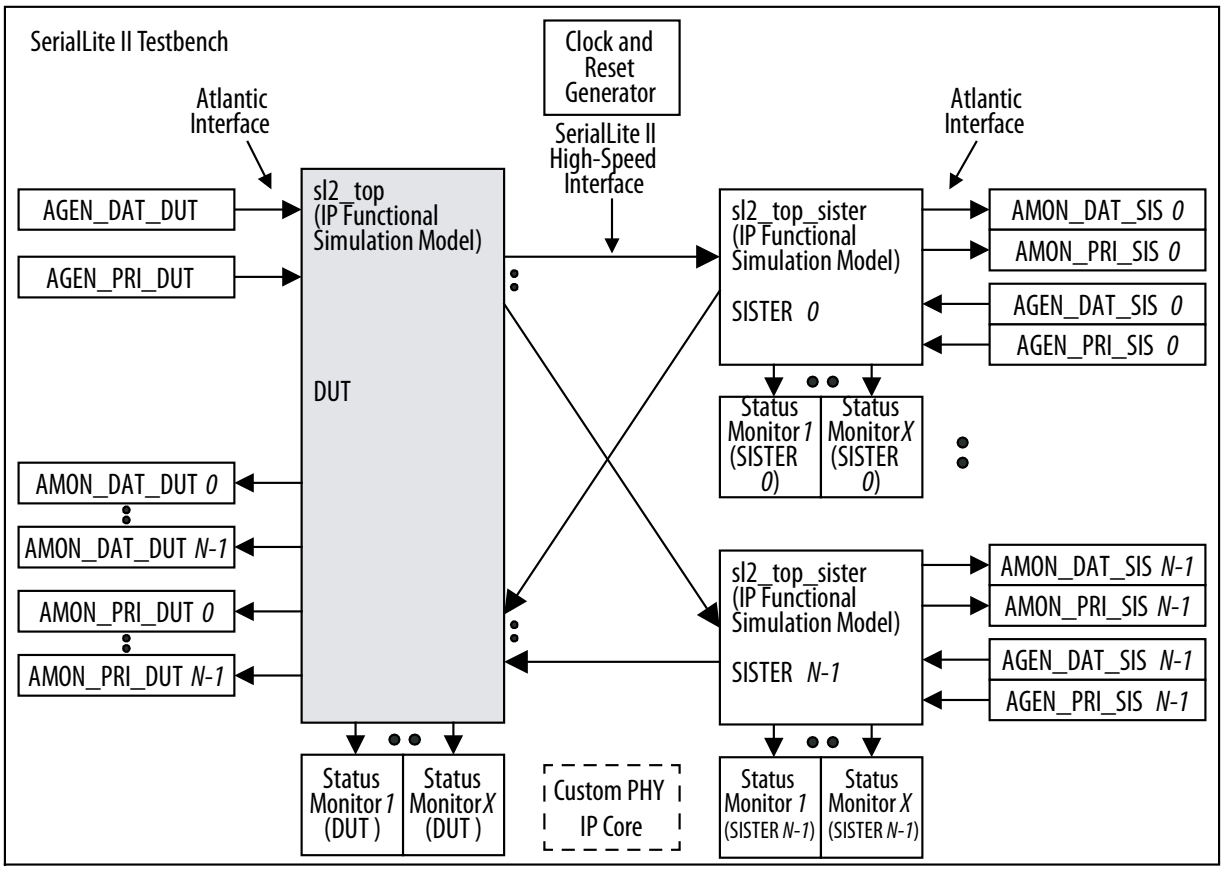
This figure shows the testbench environment for a SerialLite II single mode–receiver only, non-broadcast mode IP core. The SISTER model contains a transmitter.



Notes:
_DAT = Regular Data Port
_PRI = High Priority Port
_DUT = Refers to the DUT side
_SIS = Refers to the SISTER side

Figure 4-4: SerialLite II Testbench Environment, Verilog HDL Only (Standard Broadcast Mode)

This figure shows the testbench environment for a SerialLite II standard broadcast mode IP core with multiple SISTER instances that have one receive and transmit port.



Notes:
_DAT = Regular Data Port
_PRI = High Priority Port
_DUT = Refers to the DUT side
_SIS = Refers to the SISTER side

Simulation Flow

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

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.

When all monitors receive the last packet, the testbench ends.

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 Prime 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.

Note: If you choose Intel Stratix 10, Arria 10, Arria V, Cyclone V, or Stratix V as the target device family, you must add a list of the Custom PHY IP core simulation files into the command line Tcl file.

Simulation Pass and Fail Conditions

To understand the simulation, you need to know 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.

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`)

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. If more time is needed for a particularly long run, you can increase the WATCHTIME value.

- For Verilog HDL: Change the already defined WATCHTIME inside the testbench main section `` `define WATCHTIME 100,000,000 ``.
- For VHDL: edit the <*variation_name*>`_tb.vhd` to change the constant `WATCHTIME: time: = 100000000 ns`.

In Verilog HDL, instead of increasing gthe WATCHTIM, you could 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 the 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

The components of the SerialLite testbench each has it own functions.

Table 4-1: Testbench Components Description

DUT	The Verilog HDL or VHDL IP functional simulation model of the device under test (DUT)
SISTER	A Verilog HDL or VHDL IP functional simulation model used to test the DUT. When the DUT is asymmetric (for example, the number of receiving lanes is different than the number of transmitting lanes), is configured in single mode (receiver or transmitter only), or is configured in broadcast mode, the SISTER parameters may not match the DUT parameters, or multiple SISTER MegaCore functions may need to be instantiated.
AGEN	This testbench includes separate versions of the AGEN module for Verilog HDL and VHDL.
AMON	This testbench includes separate versions of the AMON module for Verilog HDL and VHDL.
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.
Clock and Reset Generator	The DUT and the SISTER use a common clock, with the frequency set by the MegaWizard Plug-In Manager. 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).
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 MegaWizard Plug-In Manager.

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 MegaCore function. It also supports the streaming mode if the data port is configured as such.
- The first byte of each generated packet is a sequential identifier (id) that seeds the LFSR. Every time the send_packet task is called, the agen id is incremented by one. The module operates in one of two modes: data port or priority port. When in priority port mode, the Atlantic dav signal is ignored for all but the first transfer of a packet. There can be multiple agen instantiations (for data and priority port, DUT and SISTER), depending on the DUT's chosen parameters.

Table 4-2: AGEN Tasks

AGEN Tasks	Description
`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.
`ipg(min[31:0],max[31:0])`	If the `gap` task is called, successive packets are separated by a random number of idle cycles.
`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`.
`verbose(bit_value)`	This task enables or disables the display of AGEN verbose messages.
`corrupt_sop`	This 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`	This 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.

Table 4-3: send_packet Task Field Description

The table below describes the `send_packet` task fields.

Field Location in Task	Field	Valid Values	Description
1	`addr`	• 0 to 0xFF (data) • 0 to 0xF (priority)	Set to 0.

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

Table 4-4: gap Task Field Description

The table below describes the gap task fields.

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

Table 4-5: iptg Task Field Description

The table below describes the iptg task fields.

Field Location in Task	Field	Valid Values	Description
1	prob	• 0 to 0xFF (data) • 0 to 0xF (priority)	Set to 0.
2	min	0 to 0xFFFF_FFFF (cycles)	The min field sets the minimum value, in Atlantic clock cycles, for a random gap between AGEN write transactions.
3	max	0 to 0xFFFF_FFFF (cycles)	The max field sets the maximum value, in Atlantic clock cycles, for a random gap between AGEN write transactions. A max field greater than or equal to the min field is required. When max==min, no gap occurs.

Table 4-6: verbose Task Field Description

The table below describes the verbose task fields.

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).

Table 4-7: AGEN Parameters

The Quartus Prime software sets the AGEN parameters based on the selected configuration. The parameters are fixed for a given SerialLite II configuration.

AGEN Parameter	Description

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";`

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

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. When 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.

The 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.

Table 4-8: AMON Tasks

AGEN Tasks	Description
data_checking(bit_value))	This task enables or disables the data checking.
id_checking(bit_value)	This task enables or 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.
mp_checking(bit_value)	This task enables or disables the missing SOP and EOP checking.
gap(prob[31:0],min[31:0],max[31:0])p	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.
verbose (bit_value)	This task enables or disables the display of verbose messages.

Table 4-9: data_checking Task Field Description

The table below describes the data_checking task fields.

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.

Table 4-10: id_checking Task Field Description

The table below describes the id_checking task fields.

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.

Table 4-11: wait_all_packets Task Field Description

The table below describes the wait_all_packets task fields.

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.

Table 4-12: mp_checking Task Field Description

The table below describes the mp_checking task fields.

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.

Table 4-13: read_transaction_gap Task Field Description

The table below describes the read_transaction_gap task fields.

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.

Table 4-14: verbose Task Field Description

The table below describes the verbose task fields.

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).

Table 4-15: AMON Parameters

The Quartus Prime software sets the AMON parameters based on the selected configuration. The parameters are fixed for a given SerialLite II configuration.

AMON Parameter	Description

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 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 AGEN.
	`defparam amon_dat_dut.PORT_NAME = "AMON_DAT_DUT";`
	`defparam amon_pri_sis.PORT_NAME = "AMON_PRI_SIS";`

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

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

The simulation includes status pin monitors (`pin_mon`) 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.

Table 4-16: Pin Monitor Tasks (Verilog HDL)

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

Clock and Reset Generator

The DUT and the SISTER use a common clock, with the frequency set by the Quartus Prime software.

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).

Note: 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 `Time=0`.

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

```
SERIALLITE2_TB_RESET_START

ERIALLITE2_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 Intel Stratix 10, Arria 10, Arria V, Cyclone V, and Stratix V configurations.

You are required to separately instantiate the Custom PHY IP core using the Quartus Prime software.

Example Testbench – Verilog HDL

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

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

```
//SERIALLITE2_TB_MAIN_START

//SERIALLITE2_TB_MAIN_END
```

Note: This example testbench may not match your testbench exactly.

Table 4-17: Example of a Demonstration Testbench

The table 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.

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 *tc_start/tc_end* pairs in the testbench, otherwise the testbench is declared INCOMPLETE.
`err_limit = 0;`	Sets expectation for the number of errors.
`tc_start(`TBID);`	Testcase start.
`wait (reset_n == 1);`	Waiting for the reset to complete; the reset is asserted in a separate initial block.
`// initialize packet` `counters`	
`pkt_cnt_dat_dut = packets_` `to_send;`	Sets the number of packets to be sent to the regular data port of the DUT 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.

Main Section	Comments
`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.
`fork`	Launch multiple send packet loops in parallel.

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

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

Main Section	Comments
`tc_end(`TBID);` `exit;` `end`	All loops must finish (receive all packets) before exiting.
`endmodule`	End of test case. Main initial block end.
`//SERIALLITE2_TB_MAIN_END`	End of testbench main section.

SerialLite II IP Core User Guide Archives

A

2019.01.09

UG-0705 Subscribe Send Feedback

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

IP Core Version	User Guide
14.0	SerialLite II IP Core User Guide
13.1	SerialLite II IP Core User Guide

Revision History for SerialLite II IP Core User Guide

B

2019.01.09

UG-0705 Subscribe Send Feedback

Date	Version	Changes
January 2019	2019.01.09	Added note in the *Reference Clock Frequency* topic to clarify that supported reference clock frequency is dependent on device. Users should refer to each device datasheet for the supported reference clock frequency range.
October 2016	2016.10.28	• Added indirect support for Intel Stratix 10 devices. Contact an Altera representative or file a Service Request (SR) to use the Intel Stratix 10 devices.
May 2016	2016.05.02	• Edited aggregate bandwidth information—the input data bus into the processor portion is 36 bits wide (32 bits of raw data and 8 bits of control information). • Added indirect support for Arria 10 devices. Contact an Altera representative or file a Service Request (SR) to use the Arria 10 devices. • Updated transceiver information for Arria 10, Arria V, Cyclone V, and Stratix V devices. You need to generate the Custom PHY IP core for these devices. • Changed instances of *Quartus II* to *Quartus Prime*. • Added links to archived versions of the *SerialLite II IP Core User Guide*.
July 2014	2014.07.09	• 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.
January 2014	13.1	• 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.

Date	Version	Changes
July 2012	12.0	• Added Arria V and Stratix V device support. • Added information about MegaCore configuration for Arria V and Stratix V devices.
February 2011	10.1	• 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	• 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	• Added additional ALT2GXB parameters - V_{CCH}, Reference Input Clk Frequency, and Reconfiguration Channel Number. • Added Stratix IV device support.
October 2007	7.2 (Beta)	• Added `gxb_powerdown` signal to ease merging of multiple SerialLite II cores into the same transceiver block. • Added VHDL testbench for some configurations. • Moved the ALT2GXB megafunction instantiation into a separate file to ease post-generation changes.