LogiCORE IP Aurora 64B/66B v9.2

Product Guide

Vivado Design Suite

PG074 June 4, 2014

Table of Contents

Chapter 6: Test Bench

Appendix A: Verification, Compliance, and Interoperability

Appendix B: Migrating and Upgrading

Appendix C: Debugging

Appendix D: Generating a GT Wrapper File from the Transceiver Wizard

Appendix E: Additional Resources and Legal Notices

Introduction

The Xilinx® LogiCORE™ IP Aurora 64B/66B core is a scalable, lightweight, high data rate, link-layer protocol for high-speed serial communication. The protocol is open and can be implemented using Xilinx device technology.

The Vivado® Design Suite produces source code for Aurora 64B/66B cores. The cores can be simplex or full-duplex, and feature one of two simple user interfaces and optional flow control.

Features

- Aurora 64B/66B cores supported on the Vivado Design Suite
- General-purpose data channels with throughput range from 500 Mb/s to over 200 Gb/s
- Supports up to any consecutive 16 GTX transceivers
  or 16 Virtex®-7 FPGA GTH transceivers and 16 UltraScale™ device GTH transceivers.
- Aurora 64B/66B protocol specification v1.2 compliant (64B/66B encoding)
- Low resource cost with very low (3%) transmission overhead
- Easy-to-use AXI4-Stream (framing) or streaming interface and optional flow control
- Automatically initializes and maintains the channel
- Full-duplex or simplex operation
- 32-bit Cyclic Redundancy Check (CRC) for user data
- Supports RX polarity inversion
- Big Endian/Little Endian AXI4-Stream user interface

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family(1)	UltraScale architecture, Zynq®-7000 All Programmable SoC, Virtex-7(2), Kintex®-7(2)
Supported User Interfaces	AXI4-Stream
Resources(3)	See Table 2-1 and Table 2-2.
Provided with Core	
Design Files	RTL
Example Design	Verilog
Test Bench	Verilog
Constraints File	Xilinx Design Constraints (XDC)
Simulation Model	Not Provided
Supported S/W Driver	N/A
Tested Design Flows(4)	
Design Entry	Vivado Design Suite Vivado IP Integrator
Simulation	For supported simulators, see the Xilinx Design Tools: Release Notes Guide.
Synthesis	Vivado Synthesis
Support	
Provided by Xilinx @ www.xilinx.com/support	

Notes:

1. For a complete list of supported devices, see the Vivado IP catalog.
2. For more information, see *7 Series FPGAs Overview* (DS180) [Ref 1]. and *UltraScale Architecture and Product Overview* (DS890) [Ref 2]
3. For more complete performance data, see Performance, page 10.
4. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

Overview

This product guide describes the function and operation of the LogiCORE™ IP Aurora 64B/66B v9.2 core and provides information about designing, customizing, and implementing the core.

Aurora 64B/66B is a lightweight, serial communications protocol for multi-gigabit links (Figure 1-1). It is used to transfer data between devices using one or many GTX or GTH transceivers. Connections can be *full-duplex* (data in both directions) or *simplex* (data in either one of the directions).

The LogiCORE IP Aurora 64B/66B core supports the AMBA® protocol AXI4-Stream user interface. It implements the Aurora 64B/66B protocol using the high-speed serial GTX or GTH transceivers in applicable UltraScale™, Zynq®-7000, Virtex®-7 and Kintex®-7 devices. The core can use up to 16 consecutive device GTX or GTH transceivers running at any supported line rate to provide a low-cost, general-purpose, data channel with throughput from 500 Mb/s to over 200 Gb/s.

Aurora 64B/66B cores are verified for protocol compliance using an array of automated simulation tests.

Note: Version 9.2 of the Aurora 64B/66B core supports UltraScale, Zynq-7000, Virtex-7 and Kintex-7 devices.

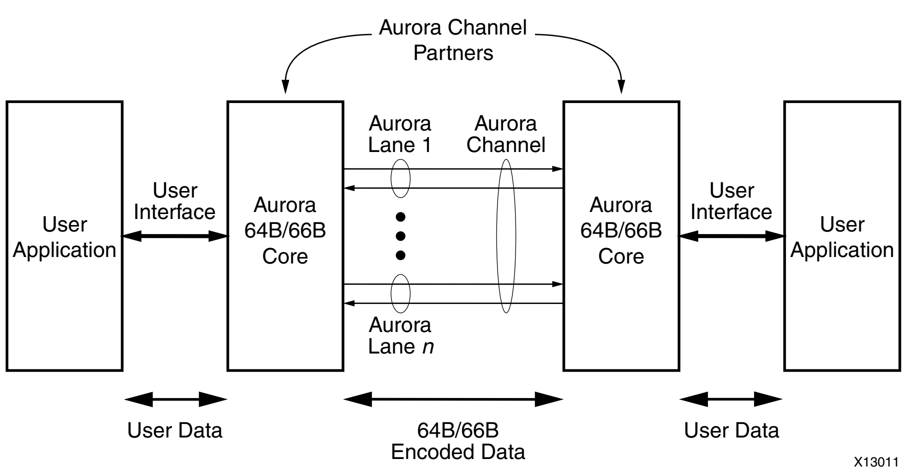


Figure 1-1: **Aurora 64B/66B Channel Overview**

Aurora 64B/66B cores automatically initialize a channel when they are connected to an Aurora 64B/66B channel partner. After initialization, applications can pass data across the channel as *frames* or *streams* of data. Aurora 64B/66B *frames* can be of any size, and can be interrupted any time by high priority requests. Gaps between valid data bytes are automatically filled with *idles* to maintain lock and prevent excessive electromagnetic interference. *Flow control* is optional in Aurora 64B/66B, and can be used to throttle the link partner transmit data rate, or to send brief, high-priority messages through the channel.

Streams are implemented in Aurora 64B/66B as a single, unending frame. Whenever data is not being transmitted, idles are transmitted to keep the link alive. Excessive bit errors, disconnections, or equipment failures cause the core to reset and attempt to initialize a new channel. The Aurora 64B/66B core can support a maximum of two symbols skew in the receive of a multi-lane channel. The Aurora 64B/66B protocol uses 64B/66B encoding. The 64B/66B encoding offers improved performance because of its very low (3%) transmission overhead, compared to 25% overhead for 8B/10B encoding.

RECOMMENDED: *Although the Aurora 64B/66B core is a fully-verified solution, the challenge associated with implementing a complete design varies depending on the configuration and functionality of the application. For best results, prior experience in building high-performance, pipelined FPGA designs using Xilinx implementation tools and user constraints files Xilinx® Design Constraints (XDC) is recommended.*

Read Status, Control, and the Transceiver Interface in Chapter 2 carefully. Consult the PCB design requirements information in the *UltraScale FPGAs GTH Transceivers User Guide* (UG576) [Ref 3] and *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476) [Ref 4]. Contact your local Xilinx representative for a closer review and estimation for your specific requirements.

Feature Summary

The LogiCORE IP Aurora 64B/66B core implements the Aurora 64B/66B protocol using the high-speed serial transceivers on the UltraScale, Zynq-7000, Virtex-7, and Kintex-7 devices. The core supports the AMBA® protocol AXI4-Stream user interface.

The Aurora 64B/66B core is based on the *Aurora 64B/66B Protocol Specification* (SP011) [Ref 5] and uses the high-speed serial GTX or GTH transceivers. The core is delivered as open-source code and supports Verilog design environments. Each core comes with an example design and supporting modules.

Applications

Aurora 64B/66B cores can be used in a wide variety of applications because of their low resource cost, scalable throughput, and flexible data interface. Examples of Aurora 64B/66B core applications include:

- **Chip-to-chip links:** Replacing parallel connections between chips with high-speed serial connections can significantly reduce the number of traces and layers required on a PCB. The Aurora 64B/66B core provides the logic needed to use GTX and GTH transceivers, with minimal FPGA resource cost.
- **Board-to-board and backplane links:** Aurora 64B/66B uses standard 64B/66B encoding, which is the preferred encoding scheme for 10-Gigabit Ethernet making it compatible with many existing hardware standards for cables and backplanes. Aurora 64B/66B can be scaled, both in line rate and channel width, to allow inexpensive legacy hardware to be used in new, high-performance systems.
- **Simplex connections (unidirectional):** In some applications there is no need for a high-speed back channel. The Aurora 64B/66B simplex protocol provides several ways to perform unidirectional channel initialization, making it possible to use the GTX and GTH transceivers when a back channel is not available, and to reduce costs due to unused full-duplex resources.
- **ASIC applications:** Aurora 64B/66B is not limited to FPGAs, and can be used to create scalable, high-performance links between programmable logic and high-performance ASICs. The simplicity of the Aurora 64B/66B protocol leads to low resource costs in ASICs as well as in FPGAs, and design resources like the Aurora 64B/66B bus functional model (BFM) with automated compliance testing make it easy to get an Aurora 64B/66B connection up and running. Contact Xilinx Sales or auroramkt@xilinx.com for information on licensing Aurora for ASIC applications.

Unsupported Features

There are no unsupported features in Aurora 64B/66B.

Licensing and Ordering Information

This Xilinx LogiCORE IP module is provided at no additional cost with the Xilinx Vivado® Design Suite under the terms of the Xilinx End User License. Information about this and other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information about pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

To use the Aurora 64B/66B core with an application specific integrated circuit (ASIC), a separate paid license agreement is required under the terms of the Xilinx Core License Agreement. Contact Aurora Marketing at auroramkt@xilinx.com for more information.

For more information, visit the Aurora 64B/66B product page.

Product Specification

Figure 2-1 shows a block diagram of the implementation of the Aurora 64B/66B core.

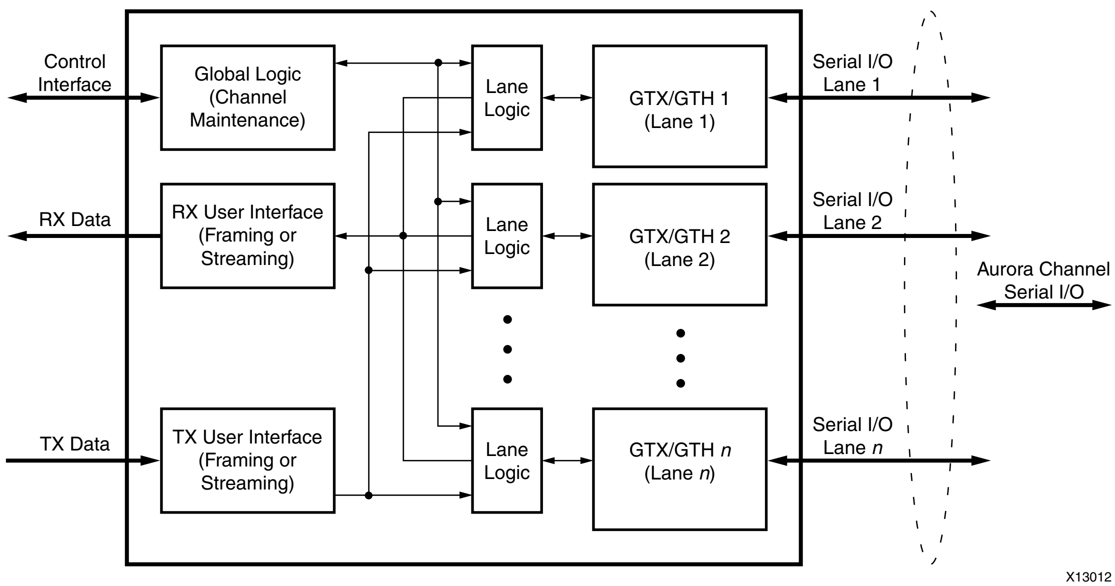


Figure 2-1: **Aurora 64B/66B Core Block Diagram**

The major functional modules of the Aurora 64B/66B core are:

- **Lane logic:** Each GTX and GTH transceiver is driven by an instance of the lane logic module, which initializes each individual GTX and GTH transceiver and handles the encoding and decoding of control characters and error detection.
- **Global logic:** The global logic module in the Aurora 64B/66B core performs the channel bonding for channel initialization. While the channel is operating, it keeps track of the Not Ready idle characters defined by the Aurora 64B/66B protocol and monitors all the lane logic modules for errors.
- **RX user interface:** The receive (RX) user interface moves data from the channel to the application. Streaming data is presented using a simple stream interface equipped with a data bus and *valid* and *ready* signals for flow control operation. Frames are presented using a standard AXI4-Stream interface. This module also performs flow control functions.

- **TX user interface:** The transmit (TX) user interface moves data from the application to the channel. A stream interface with valid and ready signals are used for streaming data. A standard AXI4-Stream interface is used for data frames. The module also performs flow control TX functions. The module has an interface for controlling clock compensation (the periodic transmission of special characters to prevent errors due to small clock frequency differences between connected Aurora 64B/66B cores). Normally, this interface is driven by a standard clock compensation manager module provided with the Aurora 64B/66B core, but it can be turned off, or driven by custom logic to accommodate special needs.

Standards

The Aurora 64B/66B core is compliant with the *Aurora 64B/66B Protocol Specification v1.2* (SP011) [Ref 5].

Performance

This section details the performance information for various core configurations.

Maximum Frequencies

The maximum frequency of the core operation is dependent on the line rates supported and the speed grade of the devices.

Latency

For a default single lane configuration, latency through an Aurora 64B/66B core is caused by pipeline delays through the protocol engine (PE) and through the GTX and GTH transceivers. The PE pipeline delay increases as the AXI4-Stream interface width increases. The GTX and GTH transceivers delays are fixed per the features of the GTX and GTH transceivers.

This section outlines method of measuring the latency for the Aurora 64B/66B core AXI4-Stream user interface in terms of `user_clk` cycles for UltraScale™, Zynq®-7000, Virtex®-7, and Kintex®-7 device GTX and GTH transceiver-based designs. For the purposes of illustrating latency, the Aurora 64B/66B modules are partitioned into GTX and GTH transceivers logic and protocol engine (PE) logic implemented in the FPGA logic.

Figure 2-2 illustrates the latency of the frame path.

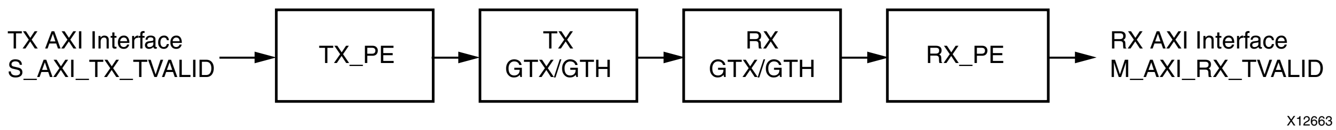


Figure 2-2: **Latency of the Frame Path**

Note: Figure 2-2 does not include the latency incurred due to the length of the serial connection between each side of the Aurora 64B/66B channel.

Maximum latency for designs using GTX or GTH transceivers from the first assertion on `s_axi_tx_tvalid` and `s_axi_tx_tready` to `m_axi_rx_tvalid` is approximately 53 `user_clk` cycles in simulation.

The pipeline delays are designed to maintain the clock speed.

Throughput

Aurora 64B/66B core throughput depends on the number of the transceivers and the target line rate of the transceivers selected. Throughput varies from 0.48 Gb/s to 203.3 Gb/s for a single-lane design to a 16-lane design, respectively. The throughput was calculated using 3% overhead of Aurora 64B/66B protocol encoding and 0.5 Gb/s to 13.1 Gb/s line rate range.

Resource Utilization

Table 2-1 through Table 2-2 show the number of look-up tables (LUTs) and flip-flops (FFs) used in selected Aurora 64B/66B *framing* and *streaming* modules in the Vivado® Design Suite implemented on a xc7vx485tffg1157-1 device. The Aurora 64B/66B core is also available in configurations not shown in the tables. The tables do not include the additional resource usage for flow controls. Resource utilization in the following tables do not include the additional resource usage for the example design modules, such as FRAME_GEN and FRAME_CHECK. Values provided are exact values for a given configuration. Values in the following tables are for the default configuration (3.125G) with support logic included.

Table 2-1: **Virtex-7 Family GTX Transceiver Resource Usage for Streaming**

Virtex-7 Family (GTX Transceiver)		Streaming		
		Duplex	Simplex	
Lanes	**Resource Type**	**Full-Duplex**	**TX-Only**	**RX-Only**
1	LUTs	549	315	377
	FFs	1359	476	957
2	LUTs	1044	467	686
	FFs	2379	761	1721
4	LUTs	1971	711	1452
	FFs	4347	1380	3139
8	LUTs	3610	1256	2805
	FFs	8219	2539	5973
16	LUTs	6656	1949	5496
	FFs	15966	4825	11641

Table 2-2: **Virtex-7 Family GTX Transceiver Resource Usage for Framing**

Virtex-7 Family (GTX Transceiver)		Framing		
		Duplex	Simplex	
Lanes	Resource Type	Full-Duplex	TX-Only	RX-Only
1	LUTs	873	315	597
	FFs	1398	499	975
2	LUTs	1475	471	1106
	FFs	2442	799	1748
4	LUTs	2628	764	2012
	FFs	4444	1425	3182
8	LUTs	4997	1566	3896
	FFs	8391	2623	6046
16	LUTs	9418	2874	7560
	FFs	16273	5018	11771

Note: UltraScale device utilization results are expected to be similar to 7 series devices.

Port Descriptions

The parameters used to generate each Aurora 64B/66B core determine the interfaces available (Figure 2-3) for that specific core. The Aurora 64B/66B cores have four to eight interfaces:

- User Interface, page 14
- User Flow Control Interface, page 17
- Native Flow Control Interface, page 20
- User K-Block Interface, page 20
- GTX and GTH Transceiver Interface, page 27
- Clock Interface, page 37
- DRP Interface, page 68
- Clock Compensation Interface, page 70

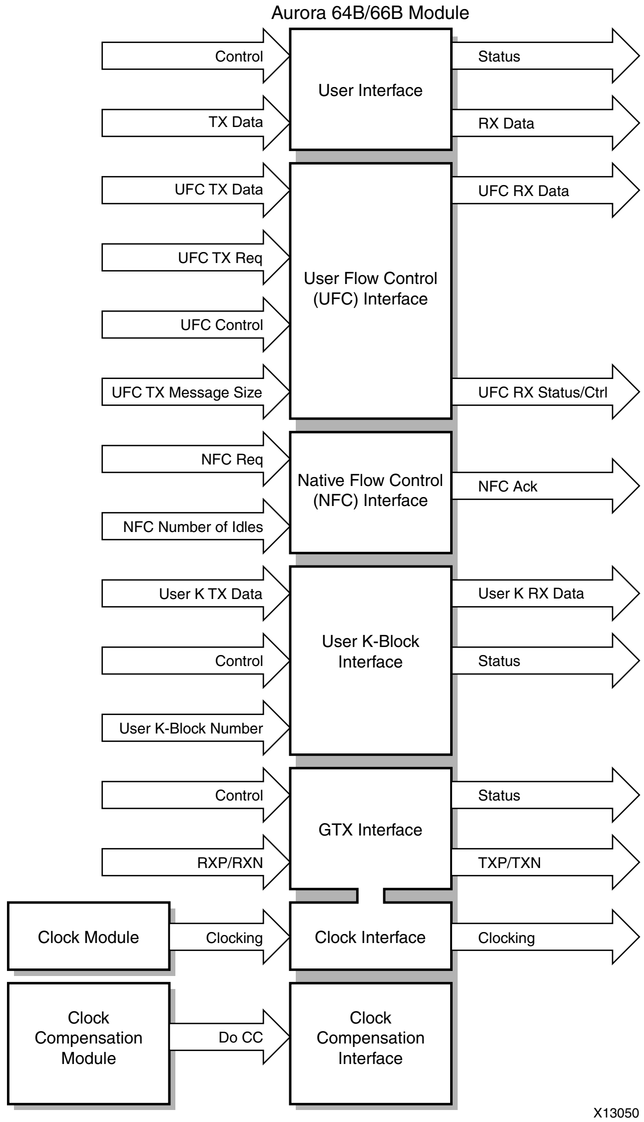


Figure 2-3: **Top-Level Interface**

User Interface

This interface includes all the ports needed to read and write *streaming* or *framed* data to and from the Aurora 64B/66B core. AXI4-Stream ports are used if the Aurora 64B/66B core is generated with a framing interface; for streaming modules, the interface consists of a simple set of data ports with data valid and ready ports. Full-duplex cores include ports for both transmit (TX) and receive (RX); simplex cores use only the ports they require in the direction they support. The width of the data ports in all interfaces depends on the number of GTX and GTH transceivers used by the core. CRC is computed on the data interface for every frame in the framing interface, if the CRC option is selected.

Framing Interface Ports (AXI4-Stream)

Table 2-3 lists the AXI4-Stream TX data ports and their descriptions. See Framing Interface, page 41 for more information. The core has an option to configure the AXI4-Stream User I/O as Little Endian from the Vivado® IDE. Default is Big Endian.

Table 2-3: **AXI4-Stream User I/O Ports (TX)**

Name	Direction	Description
s_axi_tx_tdata[0:(64*n*–1)] or s_axi_tx_tdata[(64*n*–1):0] [(1)]	Input	Outgoing data (Ascending bit order). `s_axi_tx_tdata[(64n-1):0]` is used when the **Little Endian Support** option is selected.
s_axi_tx_tready	Output	Asserted (active-High) during clock edges when signals from the source are accepted (if `s_axi_tx_tvalid` is also asserted). Deasserted (active-Low) on clock edges when signals from the source are ignored.
s_axi_tx_tlast	Input	Signals the end of the frame (active-High).
s_axi_tx_tkeep[0:(8*n*–1)] or s_axi_tx_tkeep[(8*n*–1):0] [(1)]	Input	Specifies the number of valid bytes in the last data beat (number of valid bytes = number of 1s in `tkeep`. `s_axi_tx_tkeep[(8n-1):0]` is used when the **Little Endian Support** option is selected. Example: `s_axi_tx_tkeep` = FF indicates 8 bytes are valid); valid only while `s_axi_tx_tlast` is asserted. The Aurora core supports continuous aligned stream and continuous unaligned stream of data and expects the data to be filled continuously from LSB to MSB. There cannot be invalid bytes interleaved with the valid `s_axi_tx_tdata` bus.
s_axi_tx_tvalid	Input	Asserted (active-High) when AXI4-Stream signals from the source are valid. Deasserted (active-Low) when AXI4-Stream control signals and/or data from the source should be ignored.

1. n is number of lanes.

Table 2-4 lists the AXI4-Stream RX data ports and their descriptions. See Framing Interface, page 41 for more information.

Table 2-4: **AXI4-Stream User I/O Ports (RX)**

Name	Direction	Description
m_axi_rx_tdata[0:(64*n*–1)] or m_axi_rx_tdata[(64*n*–1):0][(1)]	Output	Incoming frame data from channel partner (Ascending bit order). `m_axi_rx_tdata[(64n-1):0]` is used when the **Little Endian Support** option is selected.
m_axi_rx_tkeep[0:8*n*–1] or m_axi_rx_tkeep[8*n*–1:0][(1)]	Output	Specifies the number of valid bytes in the last data beat. Valid only when `m_axi_rx_tlast` is asserted. `m_axi_rx_tkeep[8n-1:0]` is used when the **Little Endian Support** option is selected.
m_axi_rx_tvalid	Output	Asserted (active-High) when data and control signals from an Aurora core are valid. Deasserted (Low) when data and/or control signals from an Aurora core should be ignored.
m_axi_rx_tlast	Output	Signals the end of the incoming frame (active-High, asserted for a single `user_clk` cycle).

1. n is number of lanes.

Streaming Ports

Table 2-5 lists the streaming TX data ports.

Table 2-5: **Streaming User I/O Ports (TX)**

Name	Direction	Description
s_axi_tx_tdata[0:(64*n*–1)] or s_axi_tx_tdata[(64*n*–1):0]	Input	Outgoing data (Ascending bit order). `s_axi_tx_tdata[(64n-1):0]` is used when the **Little Endian Support** option is selected.
s_axi_tx_tready	Output	Asserted (active-High) during clock edges when signals from the source are accepted (if `s_axi_tx_tvalid` is also asserted). Deasserted (active-Low) on clock edges when signals from the source are ignored.
s_axi_tx_tvalid	Input	Asserted (active-High) when AXI4-Stream signals from the source are valid. Deasserted (active-Low) when AXI4-Stream control signals and/or data from the source should be ignored.

Table 2-6 lists the streaming RX data ports. These ports are included on full-duplex and simplex RX framing cores. See Streaming Interface, page 49 for more information.

Table 2-6: **Streaming User I/O Ports (RX)**

Name	Direction	Description
m_axi_rx_tdata[0:(64*n*–1)] or m_axi_rx_tdata[(64*n*–1):0]	Output	Incoming data from channel partner (Ascending bit order). `m_axi_rx_tdata[(64n–1):0]` is used when the **Little Endian Support** option is selected.
m_axi_rx_tvalid	Output	Asserted (active-High) when data and control signals from an Aurora 64B/66B core are valid. Deasserted (active-Low) when data and/or control signals from an Aurora 64B/66B core should be ignored.

Notes:

High Priority request for any type of flow controls are:

- `TXDATAVALID` deassertion from the GT TX interface (1 cycle)
- CC transmission (6 cycles)

User Flow Control Interface

If the core is generated with User Flow Control (UFC) enabled, a UFC interface is created. The TX side of the UFC interface consists of a request, valid, and ready ports that are used to start a UFC message, and a port to specify the length of the message. You supply the message data to the UFC data port immediately after a UFC request, depending on valid and ready ports of the UFC interface; this in turn deasserts the ready port of the user data interface indicating that the core is no longer ready for normal data, thereby allowing UFC data to be written to the UFC data port.

The RX side of the UFC interface consists of a set of AXI4-Stream ports that allows the UFC message to be read as a frame. Full-duplex modules include both TX and RX UFC ports; simplex modules retain only the interface they need to send data in the direction they support. Table 2-7 describes the ports for the UFC interface. See User Flow Control, page 54 for more information.

Table 2-7: **UFC I/O Ports**

Name	Direction	Description
ufc_tx_req	Input	Asserted (active-High) to request a UFC message be sent to the channel partner. Requests are processed after a single cycle, unless another UFC message is in progress and not on its last cycle. After a request, the `s_axi_ufc_tx_tdata` bus is ready to send data within two cycles unless interrupted by a higher priority event.
ufc_tx_ms[0:7] or ufc_tx_ms[7:0]	Input	Specifies the number of bytes in the UFC message (the message size). The maximum UFC message size is 256. The value specified at `ufc_tx_ms` is one less than the actual amount of bytes transferred. For example, a value of 3 will transmit 4 bytes of data; and a value of 0 will transfer 1 byte. `ufc_tx_ms[7:0]` is used when the **Little Endian Support** option is selected.
s_axi_ufc_tx_tready	Output	Asserted (active-High) when an Aurora 64B/66B core is ready to read data from the `s_axi_ufc_tx_tdata` interface. This signal is asserted one clock cycle after `ufc_tx_req` is asserted and no high priority requests in progress. `s_axi_ufc_tx_tready` continues to be asserted while the core waits for data for the most recently requested UFC message. The signal is deasserted for CC, CB, and NFC requests, which are higher priority. While `s_axi_ufc_tx_tready` is asserted, `s_axi_tx_tready` is deasserted.
s_axi_ufc_tx_tdata[0:(64*n*–1)] or s_axi_ufc_tx_tdata[(64*n*–1):0]	Input	Input bus for UFC message data to the Aurora channel. Data is read from the bus into the channel only when both `s_axi_ufc_tx_tvalid` and `s_axi_ufc_tx_tready` are asserted on a positive `user_clk` edge. If the number of bytes in the message is not an integer multiple of the bytes in the bus, on the last cycle, only the bytes needed to finish the message starting from the left of the bus are used. `s_axi_ufc_tx_tdata[(64n-1):0]` is used when the **Little Endian Support** option is selected.
s_axi_ufc_tx_tvalid	Input	Asserted (active-High) when data on `s_axi_ufc_tx_tdata` is valid. If deasserted while `s_axi_ufc_tx_tready` is asserted, Idle blocks are inserted in the UFC message.
m_axi_ufc_rx_tdata[0:(64*n*–1)] or m_axi_ufc_rx_tdata[(64*n*–1):0][(1)]	Output	Incoming UFC message data from the channel partner. `m_axi_ufc_rx_tdata[(64n-1):0]` is used when the **Little Endian Support** option is selected.
m_axi_ufc_rx_tvalid	Output	Asserted (active-High) when the values on the `m_axi_ufc_rx_tdata` port is valid. When this signal is not asserted, all values on the `m_axi_ufc_rx_tdata` port should be ignored.
m_axi_ufc_rx_tlast	Output	Signals (active-High) the end of the incoming UFC message.

Table 2-7: **UFC I/O Ports *(Cont'd)***

Name	Direction	Description
m_axi_ufc_rx_tkeep[0:(8*n*–1)] or m_axi_ufc_rx_tkeep[(8*n*–1):0][(1)]	Output	Specifies the number of valid bytes of data presented on the `m_axi_ufc_rx_tdata` port on the last word of a UFC message. Valid only when `m_axi_ufc_rx_tlast` is asserted. Maximum size of UFC is 256 bytes. `m_axi_ufc_rx_tkeep[(8n–1):0]` is used when the **Little Endian Support** option is selected.

1. n is number of lanes.

Native Flow Control Interface

If the core is generated with native flow control (NFC) enabled, an NFC interface is created. This interface includes a request and an acknowledge port that are used to send NFC messages.

Note: NFC completion mode is not applicable to streaming designs.

See Native Flow Control, page 51 for more information.

Table 2-8 lists the ports for the NFC interface.

Table 2-8: **NFC I/O Ports**

Name	Direction	Description
s_axi_nfc_tx_tvalid	Input	Asserted (active-High) to request an NFC message be sent to the channel partner. Must be held until `s_axi_nfc_tx_tready` is asserted.
s_axi_nfc_tx_tready	Output	Asserted (active-High) when an Aurora 64B/66B core accepts an NFC request.
s_axi_nfc_tx_tdata[0:15] or s_axi_nfc_tx_tdata[15:0]	Input	Incoming NFC message data from the channel partner. `s_axi_nfc_tx_tdata[15:0]` is used when the **Little Endian Support** option is selected. See Native Flow Control for more information.

User K-Block Interface

If the core is generated with the User K-block feature enabled, a User K interface is created. User K-blocks are special single block codes that include control blocks that are not decoded by the Aurora interface, but are instead passed directly to the user application. These blocks can be used to implement application specific control functions. The TX side consists of valid and ready ports that are used to start a User K transmission along with the block number port to indicate which of the nine User K-blocks needs to be transmitted. The User K data is transmitted after the core provides a ready for the User K interface. It also indicates to the user interface that it is no longer ready for normal data, thereby allowing User K data to be written to the User K data port. The User K blocks are single block codes.

The receive side of the User K interface consists of an RX valid signal to indicate the reception of User K-block. Full-duplex modules include both TX and RX User K ports; simplex modules retain only the interface they need to send data in the direction they support.

Table 2-9 lists the ports for the User K-block interface. See User K-Block Interface, page 20 for more information.

Table 2-9: **User K-Block I/O Ports**

Name	Direction	Description
s_axi_user_k_tx_tdata[0:(64*n*–1)] or s_axi_user_k_tx_tdata[(64*n*–1):0][(1)]	Input	User K-block data is 64-bit aligned. `s_axi_user_k_tx_tdata[(64n-1):0]` is used when the **Little Endian Support** option is selected. Signal Mapping per lane: Default: `s_axi_user_k_tx_tdata=`{4'h0,`user_k_blk_no[0:4n-1], s_axi_user_k_tdata[0:56n-1]`}. Little Endian format: `s_axi_user_k_tx_tdata=`{`s_axi_user_k_tdata[56n-1:0]`,4'h0,`user_k_blk_no[4n-1:0]`}.
s_axi_user_k_tx_tvalid	Input	Asserted (active-High) when User K data on `s_axi_user_k_tx_tdata` port is valid.
s_axi_user_k_tx_tready	Output	Asserted (active-High) when the Aurora 64B/66B core is ready to read data from the `s_axi_user_k_tx_tdata` interface.
m_axi_rx_user_k_tvalid	Output	Asserted (active-High) when User K data on `m_axi_rx_user_k_tdata` port is valid.
m_axi_rx_user_k_tdata or m_axi_rx_user_k_tdata[(64n–1):0][(1)]	Output	Receive User K-blocks from the Aurora lane is 64-bit aligned. `m_axi_rx_user_k_tdata[(64n-1):0]` is used when the **Little Endian Support** option is selected. Signal Mapping per lane: Default: `m_axi_rx_user_k_tdata=`{4'h0,`rx_user_k_blk_no[0:4n-1]`,`m_axi_rx_user_k_tdata[0:56n-1]`}. Little Endian format: `m_axi_rx_user_k_tdata=`{`m_axi_rx_user_k_tdata[56n-1:0]`,4'h0,`rx_user_k_blk_no`[4n–1:0]}.

1. n is number of lanes.

Status and Control Ports

Table 2-10 describes the function of the status and control ports for full-duplex cores.

Table 2-10: **Status and Control Ports for Full-Duplex Cores**

Name	Direction	Description
channel_up	Output	Asserted (active-High) when Aurora channel initialization is complete and channel is ready to send/receive data.
lane_up[0:*m*–1][(1)]	Output	Asserted (active-High) for each lane upon successful lane initialization, with each bit representing one lane. The Aurora 64B/66B core can only receive data after all `lane_up` signals are asserted.
hard_err	Output	Hard error detected (active-High, asserted until Aurora 64B/66B core resets). See Table 2-21, page 61 for more details.

Table 2-10: **Status and Control Ports for Full-Duplex Cores *(Cont'd)***

Name	Direction	Description
loopback[2:0]	Input	See the *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476) [Ref 4] or *UltraScale Architecture GTH Transceivers User Guide* (UG576) [Ref 3] for details about loopback. See References in Appendix E.
power_down	Input	Drives Aurora 64B/66B core to reset (active-High).
reset	Input	Resets the Aurora 64B/66B core (active-High) is connected to top level through a debouncer. This port systematically resets all of the Aurora core logic. This signal is debounced using `user_clk` for at least 6 `user_clk` cycles. See Reset and Power Down in this product guide for more details.
soft_err	Output	Soft error detected in the incoming serial stream. See Table 2-21, page 61 for more details (active-High, asserted for a single `user_clk` period).
rxp[0:*m*-1]	Input	Positive differential serial data input pin.
rxn[0:*m*-1]	Input	Negative differential serial data input pin.
txp[0:*m*-1]	Output	Positive differential serial data output pin.
txn[0:*m*-1]	Output	Negative differential serial data output pin.
pma_init	Input	The `pma_init` (active-High) reset signal for the serial transceiver is connected to the top level through a debouncer. This port systematically resets all Physical Coding Sublayer (PCS) and Physical Medium Attachment (PMA) subcomponents of the transceiver. The signal is debounced using `init_clk_in` for at least 6 `init_clk` cycles. See the Reset section in the user guide of the related transceiver for more details.

Table 2-10: **Status and Control Ports for Full-Duplex Cores *(Cont'd)***

Name	Direction	Description
init_clk	Input	The init_clk signal is used to register and debounce the pma_init signal. The init_clk signal is used by the GT TX/RX Reset FSMs to initialize and execute the reset mode and handle MMCM reset for user_clk generation. The rate of init_clk is preferred to be in the range of 50 to 200 MHz. The default init_clk frequency set by the core is 50 MHz. You need to update this with respect to your system in the XDC. For designs with Zynq-7000 and 7 series devices: init_clk is constrained as 200 MHz in the example design xdc, <component name>_clocks.xdc and <component name>_ooc.xdc. In addition, the STABLE_CLOCK_PERIOD parameter is set as 5 ns to reflect the same in the <component name>_core file. The INIT_CLOCKPERIOD parameter in <component name>_TB is set to generate init_clk set as 5 ns. The init_clk frequency should be constrained between 50 MHz to 200 MHz. Any change in the init_clk period should be made in the example xdc, <component name>_clocks. xdc, <component name>_ooc. xdc, <component name>_core and <component name>_TB for proper operation of the IP core. When the core is generated with the **shared logic in core** option, the init_clk port becomes differential (init_clk_p, init_clk_n) For UltraScale architecture designs: The init_clk frequency should be equal to the TXUSERCLK frequency and the value should not exceed 200 MHz. The TXUSERCLK frequency depends on the line rate and the internal datapath width. Refer to the *UltraScale FPGAs GTH Transceivers User Guide* (UG576) [Ref 3] for more details. The Aurora 64B/66B core configures 32 bits as the internal datapath width for the GT. This init_clk is connected to DRPCLK of the DRP ports of GTHE3_CHANNEL as well. Any change in the init_clk period should be made in the example xdc file, <component name>_clocks.xdc, <component name>_ooc.xdc, and <component name>_TB for proper operation of the IP core.

Notes:

1. *m* is the number of GTX or GTH transceivers.

Table 2-11 describes the function of the status and control ports for simplex-TX cores.

Note: The Aurora 64B/66B Channel requires more time to accept any user-initiated request after assertion of the CHANNEL_UP signal. The time taken varies depending on the core configuration. The core however will not assert the respective TREADY for any of the interface until the channel is good for data transfer.

RECOMMENDED: *Monitor the assertion of the s_axi_tx_tready signal of the data interface before initiating any request through the Aurora channel. Any flow control request initiated before assertion of the s_axi_tx_tready signal will be not be processed by the core.*

Table 2-11: **Status and Control Ports for Simplex-TX or TX/RX Simplex Cores**

Name	Direction	Description
tx_channel_up	Output	Asserted (active-High) when Aurora channel initialization is complete and channel is ready to send data.
tx_lane_up[0:*m*–1][(1)]	Output	Asserted (active-High) for each lane upon successful lane initialization, with each bit representing one lane. The Aurora 64B/66B core can only transmit data after all `tx_lane_up` signals are asserted.
tx_hard_err	Output	Hard error detected (active-High, asserted until Aurora 64B/66B core resets). See Table 2-21, page 61 for more details.
power_down	Input	Drives Aurora 64B/66B core to reset (active-High).
reset	Input	Resets the Aurora 64B/66B core (active-High). This signal must be synchronous to `user_clk` and must be asserted for at least six `user_clk` cycles.
tx_soft_err	Output	Soft error detected in the transmit logic. See Table 2-21, page 61 for more details (active-High, asserted for a single `user_clk` period).
txp[0:*m*–1]	Output	Positive differential serial data output pin.
txn[0:*m*–1]	Output	Negative differential serial data output pin.
pma_init	Input	The `pma_init` (active-High) reset signal for the serial transceiver is connected to the top level through a debouncer. This port systematically resets all Physical Coding Sublayer (PCS) and Physical Medium Attachment (PMA) subcomponents of the transceiver. The signal is debounced using `init_clk_in` for at least 6 `init_clk` cycles. See the Reset section in the user guide of relevant transceiver for more details.

Table 2-11: **Status and Control Ports for Simplex-TX or TX/RX Simplex Cores *(Cont'd)***

Name	Direction	Description
init_clk	Input	The init_clk signal is used to register and debounce the pma_init signal. The init_clk signal is used by the GT TX Reset FSMs to initialize and execute the reset mode and handle MMCM reset for user_clk generation. The rate of init_clk is preferred to be in the range of 50 to 200 MHz. The default init_clk frequency set by the core is 50 MHz. You need to update this with respect to your system in the XDC. For designs with Zynq-7000 and 7 series devices: init_clk is constrained as 200 MHz in the example design xdc, <component name>_clocks.xdc and <component name>_ooc.xdc. In addition, the STABLE_CLOCK_PERIOD parameter is set as 5 ns to reflect the same in the <component name>_core file. The INIT_CLOCKPERIOD parameter in <component name>_TB is set to generate init_clk set as 5 ns. The init_clk frequency should be constrained between 50 MHz to 200 MHz. Any change in the init_clk period should be made in the example xdc, <component name>_clocks. xdc, <component name>_ooc. xdc, <component name>_core and <component name>_TB for proper operation of the IP core. When the core is generated with the **shared logic in core** option, the init_clk port becomes differential (init_clk_p, init_clk_n) For UltraScale architecture designs: The init_clk frequency should be equal to the TXUSERCLK frequency and the value should not exceed 200 MHz. The TXUSERCLK frequency depends on the line rate and the internal datapath width. Refer to the *UltraScale FPGAs GTH Transceivers User Guide* (UG576) [Ref 3] for more details. The Aurora 64B/66B core configures 32 bits as the internal datapath width for the GT. This init_clk is connected to DRPCLK of the DRP ports of GTHE3_CHANNEL as well. Any change in the init_clk period should be made in the example xdc file, <component name>_clocks.xdc, <component name>_ooc.xdc, and <component name>_TB for proper operation of the IP core.

Notes:
1. *m* is the number of GTX and GTH transceivers.

Table 2-12 describes the function of the status and control ports for simplex-RX cores. See Status and Control Ports, page 21 for more information.

Table 2-12: **Status and Control Ports for Simplex-RX or TX/RX Simplex Cores**

Name	Direction	Description
rx_channel_up	Output	Asserted (active-High) when Aurora channel initialization is complete and the channel is ready to receive data.
rx_lane_up[0:*m*–1](1)	Output	Asserted (active-High) for each lane upon successful lane initialization, with each bit representing one. The Aurora 64B/66B core can only receive data after all rx_lane_up signals are asserted.
rx_hard_err	Output	Hard error detected (active-High, asserted until Aurora 64B/66B core resets). See Table 2-21, page 61 for more details.

Table 2-12: **Status and Control Ports for Simplex-RX or TX/RX Simplex Cores *(Cont'd)***

Name	Direction	Description
power_down	Input	Drives Aurora 64B/66B core to reset (active-High).
reset	Input	Resets the Aurora 64B/66B core (active-High). This signal must be synchronous to user_clk and must be asserted for at least six user_clk cycles.
rx_soft_err	Output	Soft error detected in the receive logic. See Table 2-21, page 61 for more details. (active-High, asserted for a single user_clk period).
rxp[0:*m*–1]	Input	Positive differential serial data input pin.
rxn[0:*m*–1]	Input	Negative differential serial data input pin.
pma_init	Input	The pma_init (active-High) reset signal for the serial transceiver is connected to the top level through a debouncer. This port systematically resets all Physical Coding Sublayer (PCS) and Physical Medium Attachment (PMA) subcomponents of the transceiver. The signal is debounced using init_clk_in for at least 6 init_clk cycles.
init_clk	Input	The init_clk signal is used to register and debounce the pma_init signal. The init_clk signal is used by the GT TX Reset FSMs to initialize and execute the reset mode and handle MMCM reset for user_clk generation. The rate of init_clk is preferred to be in the range of 50 to 200 MHz. For designs with Zynq-7000 and 7 series devices: init_clk is constrained as 200 MHz in the example design xdc, <component name>_clocks.xdc and <component name>_ooc.xdc. In addition, the STABLE_CLOCK_PERIOD parameter is set as 5 ns to reflect the same in the <component name>_core file. The INIT_CLOCKPERIOD parameter in <component name>_TB is set to generate init_clk set as 5 ns. The init_clk frequency should be constrained between 50 MHz to 200 MHz. Any change in the init_clk period should be made in the example xdc, <component name>_clocks. xdc, <component name>_ooc. xdc, <component name>_core and <component name>_TB for proper operation of the IP core. When the core is generated with the **shared logic in core** option, the init_clk port becomes differential (init_clk_p, init_clk_n) For UltraScale architecture designs: The init_clk frequency should be equal to the TXUSERCLK frequency and the value should not exceed 200 MHz. The TXUSERCLK frequency depends on the line rate and the internal datapath width. Refer to the *UltraScale FPGAs GTH Transceivers User Guide* (UG576) [Ref 3] for more details. The Aurora 64B/66B core configures 32 bits as the internal datapath width for the GT. This init_clk is connected to DRPCLK of the DRP ports of GTHE3_CHANNEL as well. Any change in the init_clk period should be made in the example xdc file, <component name>_clocks.xdc, <component name>_ooc.xdc, and <component name>_TB for proper operation of the IP core.

Notes:
1. *m* is the number of GTX and GTH transceivers.

CAUTION! *The default init_clk frequency set by the core for 7 series devices is 50 MHz. You need to update this with respect to your system in the XDC.*

GTX and GTH Transceiver Interface

This interface includes the serial I/O ports of the GTX and GTH transceivers and the control and status ports of the Aurora 64B/66B core. This interface is your access to control functions such as reset, loopback, and power down. The DRP interface can be used to access or update the serial transceiver parameters and settings through the AXI4-Lite or Native DRP interface.

Table 2-13: **Transceiver DRP Ports**

Name	Direction	Description
rxp[0:*m*–1][1]	Input	Positive differential serial data input pin.
rxn[0:*m*–1]	Input	Negative differential serial data input pin.
txp[0:*m*–1]	Output	Positive differential serial data output pin.
txn[0:*m*–1]	Output	Negative differential serial data output pin.
loopback[2:0]	Input	Loopback port of the transceiver. See the related transceiver user guide for loopback test mode configurations
pma_init	Input	Asynchronous reset signal for the transceiver. See the related transceiver user guide for more information.
tx_lock	Output	Indicates incoming serial transceiver `refclk` is locked by the transceiver PLL. See the related transceiver user guide for more information.
7 Series and UltraScale FPGA Transceiver DRP Ports[2]		
drpaddr_in	Input	DRP address bus.
drp_clk_in	Input	DRP interface clock.
drpdi_in	Input	Data bus for writing configuration data from the FPGA logic resources to the transceiver
drpdo_out	Output	Data bus for reading configuration data from the transceiver to the FPGA logic resources.
drpen_in	Input	DRP enable signal.
drprdy_out	Output	Indicates operation is complete for write operations and data is valid for read operations.
drpwe_in	Input	DRP write enable.

1. m is the number of GTX and GTH transceivers
2. See the related transceiver user guide for more information on DRP operation
3. Transceiver debug ports will get enabled if you select the **Additional transceiver control and status ports** check box option in the Vivado IDE.

In Table 2-14 ports are visible only when the Transceiver Control **Additional transceiver control and status ports** option is selected through the dialog box while configuring the Aurora 64B/66B core. More details can be found at *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476) [Ref 4] and *UltraScale Architecture GTH Transceivers User Guide* (UG576) [Ref 3].

Table 2-14: **7 Series and Zynq-7000 Device Transceiver Debug Ports**

Name	Direction	Description
gt<lane>_eyescandataerror_out	Output	Asserts High for one `rec_clk` cycle when an (unmasked) error occurs while in the COUNT or ARMED state. Available for Duplex and RX-Only Simplex configuration. See the relevant transceiver user guide for more information.
gt<lane>_eyescanreset_in	Input	This port is driven High and then deasserted to start the EYESCAN reset process. Available for Duplex and RX-Only Simplex configuration. See the relevant transceiver user guide for more information.
gt<lane>_eyescantrigger_in	Input	Causes a trigger event. Available for Duplex and RX-Only Simplex configuration. See the relevant transceiver user guide for more information.
gt<lane>_rxcdrhold_in	Input	Hold the CDR control loop frozen. Available for Duplex and RX-Only Simplex configuration and applicable for Zynq-7000 and 7 series device GTX and GTH transceivers only. See the relevant transceiver user guide for more information
gt<lane>_rxlpmhfovrden_in	Input	OVRDEN RX LPM • `2'b00`: KH High frequency loop adapt • `2'b10`: Freeze current adapt value • `2'bx1`: Override KH value according to attribute RXLPM_HF_CFG Available for Duplex and RX-Only Simplex configuration and applicable for Zynq-7000 and 7 series device GTX and GTH transceivers only. See the relevant transceiver user guide for more information
gt<lane>_rxdfeagchold_in	Input	HOLD RX DFE • `2'b00`: Automatic gain control (AGC) loop adapt • `2'b10`: Freeze current AGC adapt value • `2'bx1`: Override AGC value according to attribute RX_DFE_GAIN_CFG Available for Duplex and RX-Only Simplex configuration and applicable for Zynq-7000 and 7 series device GTX and GTH transceivers only. See the relevant transceiver user guide for more information

Table 2-14: **7 Series and Zynq-7000 Device Transceiver Debug Ports *(Cont'd)***

Name	Direction	Description
gt<lane>_rxdfeagcovrden_in	Input	OVRDEN RX DFE • `2'b00`: Automatic gain control (AGC) loop adapt • `2'b10`: Freeze current AGC adapt value • `2'bx1`: Override AGC value according to attribute RX_DFE_GAIN_CFG Available for Duplex and RX-Only Simplex configuration and applicable for Zynq-7000 and 7 series device GTX and GTH transceivers only. See the relevant transceiver user guide for more information
gt<lane>_rxdfelfhold_in	Input	When set to `1'b1`, the current value of the low-frequency boost is held. When set to `1'b0`, the low-frequency boost is adapted. Available for Duplex and RX-Only Simplex configuration and applicable for 7 series device GTP transceivers only. See the relevant transceiver user guide for more information.
gt<lane>_rxdfelpmreset_in	Input	This port is driven High and then deasserted to start the DFE reset process. Available for Duplex and RX-Only Simplex configuration and applicable for Zynq-7000 and 7 series device GTX and GTH transceivers only. See the relevant transceiver user guide for more information.
gt<lane>_rxlpmlfklovrden_in	Input	OVRDEN RX LPM • `2'b00`: KL Low frequency loop adapt • `2'b10`: Freeze current adapt value • `2'bx1`: Override KL value according to attribute RXLPM_LF_CFG Available for Duplex and RX-Only Simplex configuration and applicable for Zynq-7000 and 7 series device GTX and GTH transceivers only. See the relevant transceiver user guide for more information.
gt<lane>_rxlpmen_in	Input	RX datapath •0: DFE •1: LPM Available for Duplex and RX-Only Simplex configuration and applicable for Zynq-7000 and 7 series device GTX and GTH transceivers only. See the relevant transceiver user guide for more information.

Table 2-14: **7 Series and Zynq-7000 Device Transceiver Debug Ports *(Cont'd)***

Name	Direction	Description
gt<lane>_rxmonitorout_out	Input	GTX transceiver: • RXDFEVP[6:0] = RXMONITOROUT[6:0] • RXDFEUT[6:0] = RXMONITOROUT[6:0] • RXDFEAGC[4:0] = RXMONITOROUT[4:0] GTH transceiver: • RXDFEVP[6:0] = RXMONITOROUT[6:0] • RXDFEUT[6:0] = RXMONITOROUT[6:0] • RXDFEAGC[3:0] = RXMONITOROUT[4:1] Available for Duplex and RX-Only Simplex configuration and applicable for Zynq-7000 and 7 series device GTX and GTH transceivers only. See the relevant transceiver user guide for more information.
gt<lane>_rxmonitorsel_in	Input	Select signal for rxmonitorout[6:0] • `2'b00`: Reserved • `2'b01`: Select AGC loop • `2'b10`: Select UT loop • `2'b11`: Select VP loop Available for Duplex and RX-Only Simplex configuration and applicable for Zynq-7000 and 7 series device GTX and GTH transceivers only. See the relevant transceiver user guide for more information.
gt<lane>_txpostcursor_in	Input	Transmitter post-cursor TX pre-emphasis control. Available for Duplex and TX-Only Simplex configuration. See the relevant transceiver user guide for more information.
gt<lane>_txdiffctrl_in	Input	Driver Swing Control. Available for Duplex and TX-Only Simplex configuration. See the relevant transceiver user guide for more information.
gt<lane>_txmaincursor_in	Input	Allows the main cursor coefficients to be directly set if the TX_MAINCURSOR_SEL attribute is set to `1'b1`. Available for Duplex and TX-Only Simplex configuration. See the relevant transceiver user guide for more information.
gt<lane>_txpolariry_in	Input	The txpolarity port is used to invert the polarity of outgoing data. • 0: Not inverted. TXP is positive, and TXN is negative. • 1: Inverted. TXP is negative, and TXN is positive. Available for Duplex and TX-Only Simplex configuration. See the relevant transceiver user guide for more information.

Table 2-14: **7 Series and Zynq-7000 Device Transceiver Debug Ports *(Cont'd)***

Name	Direction	Description
gt<lane>_txpmareset_in	Input	This port is used to reset the TX PMA. It is driven High and then deasserted to start the TX PMA reset process. Activating this port resets both the TX PMA and the TX PCS.
gt<lane>_txpcsreset_in	Input	This port is used to reset the TX PCS. It is driven High and then deasserted to start the PCS reset process. Activating this port only resets the TX PCS.
gt<lane>_txresetdone_out	Output	This active-High signal indicates the GTX/GTH transceiver TX has finished reset and is ready for use. This port is driven Low when `gttxreset` goes High and is not driven High until the GTX/GTH transceiver TX detects `txuserrdy` High.
gt<lane>_rxpmareset_in	Input	This port is driven High and then deasserted to start RX PMA reset process. Refer to the *7 Series FPGAs GTX/GTH Transceivers User Guide* [Ref 4] for more details
gt<lane>_rxpcsreset_in	Input	This port is driven High and then deasserted to start RX PMA reset process. Refer to the *7 Series FPGAs GTX/GTH Transceivers User Guide* [Ref 4] for more details. The `rxpcsreset` signal does not start the reset process until `rxuserrdy` is High.
gt<lane>_rxbufreset_in	Input	This port is driven High and then deasserted to start the RX elastic buffer reset process. Activating `rxbufreset` resets the RX elastic buffer only.
gt<lane>_rxresetdone_out	Output	When asserted, this active-High signal indicates the GTX/GTH transceiver RX has finished reset and is ready for use. This port is driven Low when `gtrxreset` is driven High. This signal is not driven High until `rxuserrdy` goes High.
gt<lane>_txbufstatus_out[1:0]	Output	`txbufstatus[1]`: TX buffer overflow or underflow status. When `txbufstatus[1]` is set High, it remains High until the TX buffer is reset. • 1: TX FIFO has overflow or underflow. • 0: No TX FIFO overflow or underflow error. `txbufstatus[0]`: TX buffer fullness. • 1: TX FIFO is at least half full. • 0: TX FIFO is less than half full.
gt<lane>_rxbufstatus_out[2:0]	Output	RX buffer status. • 000b: Nominal condition. • 001b: Number of bytes in the buffer are less than CLK_COR_MIN_LAT • 010b: Number of bytes in the buffer are greater than CLK_COR_MAX_LAT • 101b: RX elastic buffer underflow • 110b: RX elastic buffer overflow

Table 2-14: **7 Series and Zynq-7000 Device Transceiver Debug Ports *(Cont'd)***

Name	Direction	Description
gt<lane>_cplllock_out	Output	Active-High PLL frequency lock signal indicates that the PLL frequency is within predetermined tolerance. The transceiver and its clock outputs are not reliable until this condition is met.
gt_qplllock<quad>	Output	Active-High PLL frequency lock signal indicates that the PLL frequency is within predetermined tolerance. The transceiver and its clock outputs are not reliable until this condition is met.
gt<lane>_precursor_in	Input	Transmitter pre-cursor TX pre-emphasis control. Available for duplex and TX-Only simplex configuration. See the relevant transceiver user guide for more information.
gt<lane>_txprbsforceerr_in	Input	When this port is driven High, errors are forced in the PRBS transmitter. While this port is asserted, the output data pattern contains errors. When `txprbssel` is set to 000, this port does not affect TXDATA.
gt<lane>_txprbssel_in[2:0]	Input	Transmitter PRBS generator test pattern control. • 000: Standard operation mode (test pattern generation is off) • 001: PRBS-7 • 010: PRBS-15 • 011: PRBS-23 • 100: PRBS-31 • 101: PCI® Express compliance pattern. Only works with 20-bit and 40-bit modes • 110: Square wave with 2 UI (alternating 0s/1s) • 111: Square wave with 16 UI, 20 UI, 32 UI, or 40 UI period (based on data width)
gt<lane>_rxprbssel_in[2:0]	Input	Receiver PRBS checker test pattern control. Only these settings are valid: • 000: Standard operation mode. (PRBS check is off) • 001: PRBS-7 • 010: PRBS-15 • 011: PRBS- 23 • 100: PRBS-31 No checking is done for non-PRBS patterns. Single bit errors cause bursts of PRBS errors because the PRBS checker uses data from the current cycle to generate the next cycle expected data.
gt<lane>_rxprbserr_out	Output	This non-sticky status output indicates that PRBS errors have occurred.

Table 2-14: **7 Series and Zynq-7000 Device Transceiver Debug Ports *(Cont'd)***

Name	Direction	Description
gt<lane>_rxprbscntreset_in	Input	Resets the PRBS error counter.
GTX transceiver: gt<lane>_dmonitorout_out[7:0] GTH transceiver: gt<lane>_dmonitorout_out [14:0]	Output	Digital Monitor Output Bus

Notes:

1. <lane> takes values from 0 to AURORA_LANES.
2. For designs using UltraScale™devices, the prefixes of the optional transceiver debug ports for single-lane cores are changed from gt<lane> to gt, and the postfixes _in and _out are removed. For multi-lane cores, the prefixes of the optional transceiver debug ports gt(n) are aggregated into a single port.

Table 2-15: **UltraScale Architecture Transceiver Debug Ports** [(1)]

Name	Direction	Description
gt_cplllock	Output	This active-High PLL frequency lock signal indicates that the PLL frequency is within predetermined tolerance. The transceiver and its clock outputs are not reliable until this condition is met.
gt_dmonitorout	Output	Digital Monitor Output Bus Available for Duplex and RX-Only Simplex configuration.
gt_eyescandataerror	Output	Asserts High for one `REC_CLK` cycle when an (unmasked) error occurs while in the COUNT or ARMED state. Available for Duplex and RX-Only Simplex configuration
gt_eyescanreset	Input	This port is driven High and then deasserted to start the EYESCAN reset process. Available for Duplex and RX-Only Simplex configuration
gt_eyescantrigger	Input	Causes a trigger event. Available for Duplex and RX-Only Simplex configuration
gt_gtrxreset		This is assigned to `PMA_INIT` in gtx_wrapper
gt_gttxreset		This is assigned to `PMA_INIT` in gtx_wrapper
gt_loopback		Connected to `LOOPBACK` in gtx_wrapper
gt_rxbufreset	Input	This port is driven High and then deasserted to start the RX elastic buffer reset process. In either single mode or sequential mode, activating `RXBUFRESET` resets the RX elastic buffer only. Available for Duplex and RX-Only Simplex configuration

Table 2-15: **UltraScale Architecture Transceiver Debug Ports *(Cont'd)*[(1)]**

Name	Direction	Description
gt_rxbufstatus	Output	RX buffer status. • 000b: Nominal condition. • 001b: Number of bytes in the buffer are less than CLK_COR_MIN_LAT • 010b: Number of bytes in the buffer are greater than • CLK_COR_MAX_LAT • 101b: RX elastic buffer underflow • 110b: RX elastic buffer overflow Available for Duplex and RX-Only Simplex configuration
gt_rxcdrhold	Input	Hold the CDR control loop frozen. Available for Duplex and RX-Only Simplex configuration
gt_rxdfelpmreset	Input	This port is driven High and then deasserted to start the DFE reset process. Available for Duplex and RX-Only Simplex configuration
gt_rxlpmen	Input	RX datapath 0: DFE 1: LPM Available for Duplex and RX-Only Simplex configuration
gt_rxpcsreset	Input	This port is driven High and then deasserted to start the RX PCS reset process. Available for Duplex and RX-Only Simplex configuration.
gt_rxpmareset	Input	This port is driven High and then deasserted to start the RX PMA reset process. Available for Duplex and RX-Only Simplex configuration
gt_rxpmaresetdone	Output	This active-High signal indicates RX PMA reset is complete. This port is driven Low when `GTRXRESET` or `RXPMARESET` is asserted. Available for Duplex and RX-Only Simplex configuration.
gt_rxpolarity		Used internally
gt_rxprbscntreset	Input	Resets the PRBS error counter. Available for Duplex and RX-Only Simplex configuration.
gt_rxprbserr	Output	This non-sticky status output indicates that PRBS errors have occurred. Available for Duplex and RX-Only Simplex configuration.
gt_rxprbssel	Input	Connects to RXPRBSSEL on transceiver channel primitives. Available for Duplex and RX-Only Simplex configuration.
gt_rxrate	Input	Dynamic pins to automatically change effective PLL dividers in the GTH transceiver RX. These ports are used for PCI® Express and other standards. Available for Duplex and RX-Only Simplex configuration

Table 2-15: **UltraScale Architecture Transceiver Debug Ports *(Cont'd)*[(1)]**

Name	Direction	Description
gt_rxresetdone	Output	When asserted, this active-High signal indicates the GTH transceiver RX has finished reset and is ready for use. In sequential mode, this port is driven Low when `GTRXRESET` is driven High. This signal is not driven High until `RXUSERRDY` goes High. In single mode, this port is driven Low when any of the RX resets are asserted. This signal is not asserted until all RX resets are deasserted and RXUSERRDY is asserted. Available for Duplex and RX-Only Simplex configuration.
gt_txbufstatus	Output	TXBUFSTATUS provides the status for the TX Buffer or the TX asynchronous gearbox. When using the TX asynchronous gearbox, the port status is as follows. Bit 1: • 0: No TX asynchronous gearbox FIFO overflow. • 1: TX asynchronous gearbox FIFO overflow. Bit 0: • 0: No TX asynchronous gearbox FIFO underflow. • 1: TX asynchronous gearbox FIFO underflow. After the port is set High, it remains High until the TX asynchronous gearbox is reset. Available for Duplex and TX-Only Simplex configuration.
gt_txdiffctrl	Input	Driver Swing Control. The default is user specified. Available for Duplex and TX-Only Simplex configuration.
gt_txpcsreset	Input	This port is used to reset the TX PCS. It is driven High and then deasserted to start the PCS reset process. In sequential mode, activating this port only resets the TX PCS. Available for Duplex and TX-Only Simplex configuration.
gt_txpmareset	Input	This port is used to reset the TX PMA. It is driven High and then deasserted to start the TX PMA reset process. In sequential mode, activating this port resets both the TX PMA and the TX PCS. Available for Duplex and TX-Only Simplex configuration.
gt_txpolarity	Input	The `TXPOLARITY` port is used to invert the polarity of outgoing data. • 0: Not inverted. TXP is positive, and TXN is negative. • 1: Inverted. TXP is negative, and TXN is positive. Available for Duplex and TX-Only Simplex configuration.
gt_txpostcursor	Input	Transmitter post-cursor TX pre-emphasis control. The default is user specified. Available for Duplex and TX-Only Simplex configuration.

Table 2-15: **UltraScale Architecture Transceiver Debug Ports *(Cont'd)*[(1)]**

Name	Direction	Description
gt_txprbsforceerr	Input	When this port is driven High, errors are forced in the PRBS transmitter. While this port is asserted, the output data pattern contains errors. When TXPRBSSEL is set to 4'b0000, this port does not affect TXDATA. Available for Duplex and TX-Only Simplex configuration.
gt_txprbssel	Input	Transmitter PRBS generator test pattern control. 4'b0000: Standard operation mode (test pattern generation is off) • 4'b0001: PRBS-7 • 4'b0010: PRBS-9 • 4'b0011: PRBS-15 • 4'b0100: PRBS-23 • 4'b0101: PRBS-31 • 4'b1000: PCI Express compliance pattern. Only works with internal data width 20 bit and 40 bit modes • 4'b1001: Square wave with 2 UI (alternating 0s/1s) • 4'b1010: Square wave with 16 UI, 20 UI, 32 UI, or 40 UI period (based on internal data width) Available for Duplex and TX-Only Simplex configuration.
gt_txprecursor	Input	Transmitter pre-cursor TX pre-emphasis control. The default is user specified. Available for Duplex and TX-Only Simplex configuration.
gt_txresetdone	Output	This active-High signal indicates the GTH transceiver TX has finished reset and is ready for use. This port is driven Low when `GTTXRESET` goes High and is not driven High until the GTH transceiver TX detects `TXUSERRDY` High. Available for Duplex and TX-Only Simplex configuration.
gt_dmonitorout	Output	Digital Monitor Output Bus Available for Duplex and RX-Only Simplex configuration.
gt_qplllock	Output	Active-High PLL frequency lock signal indicates that the PLL frequency is within predetermined tolerance. The transceiver and its clock outputs are not reliable until this condition is met.

1. Refer to the *UltraScale FPGAs GTH Transceivers User Guide* (UG576) [Ref 3] for more information about these debug ports.

Clock Interface

IMPORTANT: *This interface is most critical for correct Aurora 64B/66B core operation. The clock interface has ports for the reference clocks that drive the GTX or GTH transceivers and ports for the parallel clocks that the Aurora 64B/66B core shares with application logic.*

Table 2-16 describes Aurora 64B/66B core clock ports. In GTX and GTH transceiver designs, the reference clock can be from GTXQ/GTHQ, which is a differential input clock for each GTX or GTH transceiver. The reference clock for a GTX or GTH transceiver is provided through the `clkin` port. For more details on the clock interface, see Clocking, page 75.

Table 2-16: **Clock Ports for a GTX or GTH based Aurora 64B/66B Core**

Name	Direction	Description
mmcm_not_locked	Input	For 7 series and Zynq-7000 devices: If a MMCM is used to generate clocks for the Aurora 64B/66B core, the `mmcm_not_locked` signal should be connected to the inverse of the PLL `locked` signal of the serial transceiver PLL. The clock modules provided with the Aurora 64B/66B core use the PLL for clock division. The `mmcm_not_locked` signal from the clock module should be connected to the `mmcm_not_locked` signal on the Aurora 64B/66B core. For UltraScale devices: `mmcm_not_locked` is connected to `gtwiz_userclk_tx_active_out` driven from `<=:USER_COMPONENT_NAME:>_ultrascale_tx_userclk` module. This is driven based on the clocking helper core status and signifies that the helper core is out of reset. This port should be asserted High for the clock module to generate the `user_clk` and `sync_clk` for the core. This ports functionality differs with that of 7 series device generated design. • 1: on this port means clocking helper core is active • 0: on this port means clocking helper core is not active and not ready for normal operation
user_clk	Input	Parallel clock shared by the Aurora 64B/66B core and the user application. The `user_clk` is the output of a BUFG whose input is derived from `tx_out_clk`. The clock generations are available in the `<component name>_clock_module` file. The `user_clk` goes as the `txusrclk2` input to the transceiver. See the related transceiver user guide/data sheet for rate related information.
tx_out_clk	Output	The GTX or GTH transceiver generates `tx_out_clk` from its reference clock based on its PLL speed setting. This clock should be buffered and used to generate the user clock for logic connected to the Aurora 64B/66B core.
sync_clk	Input	Parallel clock used by internal synchronization logic of the serial transceivers in the Aurora 64B/66B core. This clock is provided as the `txusrclk` to the transceiver interface. The `sync_clk` is double the rate of `user_clk`. See the related transceiver user guide/data sheet for rate related information.

Table 2-16: **Clock Ports for a GTX or GTH based Aurora 64B/66B Core *(Cont'd)***

Name	Direction	Description
gt_pll_lock	Output	Active-High, asserted when `tx_out_clk` is stable. When this signal is deasserted (Low), circuits using `tx_out_clk` should be held in reset.
gt_refclk	Input	The `gt_refclk` (clkp/clkn) port is a dedicated external clock generated from an oscillator. This clock is fed through a dedicated IBUFDS.

Detailed Functional Description

An Aurora 64B/66B core can be generated with either a framing or streaming user data interface. In addition, flow control options are available for designs with framing interfaces. See Flow Control.

The framing user interface complies with the *AXI4-Stream Protocol Specification* (*AMBA AXI4-Stream Protocol Specification*). It comprises the signals necessary for transmitting and receiving framed user data. The streaming interface allows you to send data without frame delimiters. It is simple to operate and uses fewer resources than framing

Top-Level Architecture

The Aurora 64B/66B top-level (block level) file instantiates the Aurora lane module, the TX and RX AXI4-Stream modules, the global logic module, and the wrapper for the GTX or GTH transceiver. This top-level wrapper file is instantiated in the example design file together with clock, reset circuit, and frame generator and checker modules.

Figure 2-4 shows the Aurora 64B/66B top level for a duplex configuration. The top-level file is the starting point for a user design.

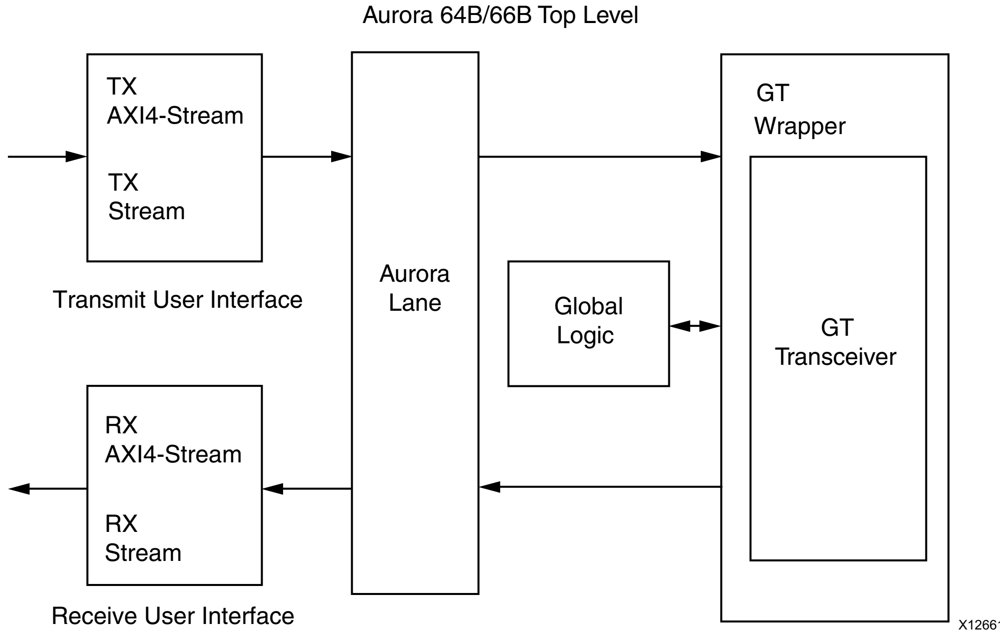


Figure 2-4: **Top-Level Architecture**

The following sections describe the streaming and framing interfaces in detail. User interface logic should be designed such that it complies with timing requirements of the respective interface as explained in the subsequent sections.

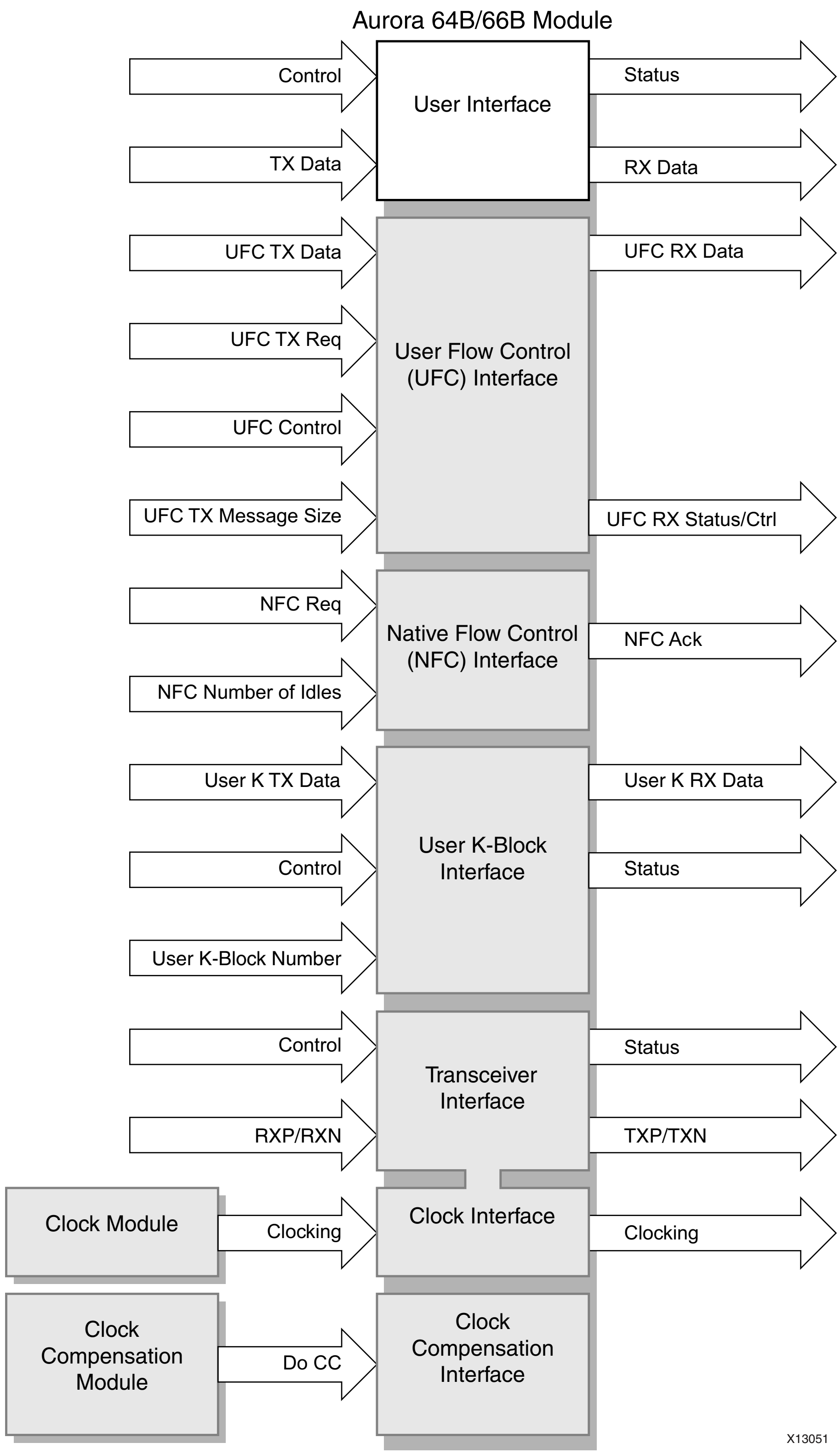


Figure 2-5: **Top-Level User Interface**

Note: The user interface signals vary depending upon the selections made when generating an Aurora 64B/66B core using the IP catalog.

Framing Interface

Figure 2-6 shows the framing user interface of the Aurora 64B/66B core, with AXI4-Stream compliant ports for TX and RX data. The core provides an option to configure the AXI4-Stream user I/O as little endian from the Vivado IDE. The default is big endian.

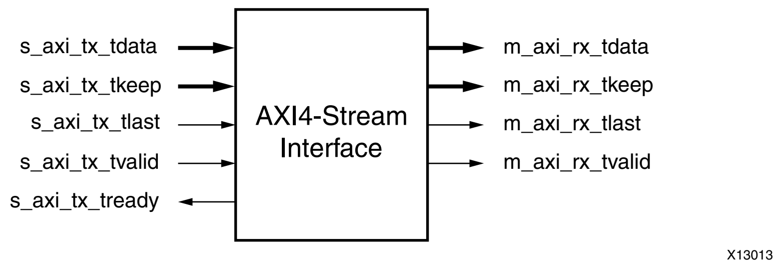


Figure 2-6: **Aurora 64B/66B Core Framing Interface (AXI4-Stream)**

Note: User interface widths will be **Big Endian** or **Little Endian** based on settings for the Aurora 64B/66B Vivado IDE

To transmit data, the user application should manipulate the control signals to cause the core to do the following:

- Take data from the user application on the `s_axi_tx_tdata` bus
- Indicates end of frame when `s_axi_tx_tlast` is asserted and stripes data across lanes in the Aurora Channel.
- User application can deassert `s_axi_tx_tvalid` to insert idle or pause cycles on the serial line

When the core receives data, it does the following:

- Detects and discards control bytes (idles, clock compensation)
- Asserts framing signals (`m_axi_rx_tlast`)
- Recovers data from the lanes
- Assembles data for presentation to the user application on the `m_axi_rx_tdata` bus along with valid number of bytes (`m_axi_rx_tkeep`) and `m_axi_rx_tvalid` is asserted during the `m_axi_rx_tlast` cycle

The AXI4-Stream user interface of Aurora 64B/66B cores uses ascending ordering. The cores transmit and receive the most significant bit of the least significant byte first. Figure 2-7 shows the organization of an *n*-byte example of the AXI4-Stream data interfaces of an Aurora 64B/66B core.

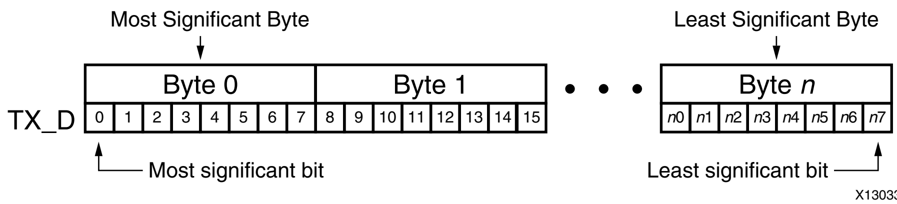


Figure 2-7: **AXI4-Stream Interface Bit Ordering**

Transmitting Data

AXI4-Stream is a synchronous interface. The Aurora 64B/66B core samples the data on the interface only on the positive edge of `user_clk`, and only on the cycles when both `s_axi_tx_tready` and `s_axi_tx_tvalid` are asserted (active-High).

When AXI4-Stream signals are sampled, they are only considered valid if `s_axi_tx_tvalid` and `s_axi_tx_tready` signals are asserted. The user application can deassert `s_axi_tx_tvalid` on any clock cycle; this causes the Aurora core to ignore the AXI4-Stream input for that cycle. If this occurs in the middle of a frame, idle symbols are sent through the Aurora channel, which eventually results idle cycles during the frame when it is received at the RX user interface.

AXI4-Stream data is only valid when it is framed. Data outside of a frame is ignored. To end a frame, assert `s_axi_tx_tlast` while the last word (or partial word) of data is on the `s_axi_tx_tdata` port. If the CRC option is selected, CRC is calculated and inserted into the data stream after the last data word. This re-calculates `s_axi_tx_tkeep` based on the number of valid CRC bytes and asserts `s_axi_tx_tlast` accordingly.

Data Strobe

AXI4-Stream allows the last word of a frame to be a partial word. This lets a frame contain any number of bytes, regardless of the word size. The `s_axi_tx_tkeep` bus is used to indicate the number of valid bytes in the final word of the frame. The bus is only used when `s_axi_tx_tlast` is asserted. TKEEP is the number of valid bytes in the `s_axi_tx_tdata` bus. TKEEP associates validity to a particular byte in the last data beat of a frame. If TKEEP is “0F” in the last beat of data with `s_axi_tx_tlast` asserted High, then 4 (LSB bytes) out of 8 bytes are valid and byte4 to byte7 are not valid. All 1s in the `s_axi_tx_tkeep` value indicate all bytes in the `s_axi_tx_tdata` port are valid. `s_axi_tx_tkeep` does not specify the position of the valid bytes, but is the number of valid bytes on the last beat of data with `s_axi_tx_tlast` asserted. Core expects `TKEEP` to be left aligned from LSB. See Appendix B, Migrating and Upgrading for limitations on the types of data stream supported by the core.

Aurora 64B/66B Frames

The TX submodule translates each user frame that it receives through the TX interface to an Aurora 64B/66B frame. The core starts an Aurora 64B/66B frame by sending a data block with the first word of data, and ends the frame by sending a separator block containing the last bytes of the frame. Idle blocks are inserted whenever data is not available. Blocks are eight bytes of scrambled data or control information with a two-bit control header (a total of 66 bits). All data in Aurora 64B/66B is sent as part of a data block or a separator block (a separator block consists of a count field, indicating how many bytes are valid in that particular block).

Table 2-17 shows a typical Aurora 64B/66B frame with an even number of data bytes.

Length

The user application controls the channel frame length by manipulating the `s_axi_tx_tvalid` and `s_axi_tx_tlast` signals. The Aurora 64B/66B core converts these to data blocks, idle blocks, and separator blocks, as shown in Table 2-17.

Table 2-17: **Typical Channel Frame**

Data Byte 0	Data Byte 1	Data Byte 2	Data Byte 3	. . .	Data Byte *n* –2	Data Byte *n* –1	Data Byte *n*
SEP (1E)	Count (4)	Data Byte 0	Data Byte 1	Data Byte 2	Data Byte 3	x	x

Example A: Simple Data Transfer

Figure 2-8 shows an example of a simple data transfer on a AXI4-Stream interface that is *n* bytes wide. In this case, the amount of data being sent is 3*n* bytes and so requires three data beats. `s_axi_tx_tready` is asserted, indicating that the AXI4-Stream interface is ready to transmit data. When the Aurora 64B/66B is not sending data, it sends idle blocks.

To begin the data transfer, the user application asserts `s_axi_tx_tvalid` and provides the first *n* bytes of the user frame. Because `s_axi_tx_tready` is already asserted, data transfer begins on the next clock edge. The data bytes are placed in data blocks and transferred through the Aurora channel.

To end the data transfer, the user application asserts `s_axi_tx_tlast`, `s_axi_tx_tvalid`, the last data bytes, and the appropriate value on the `s_axi_tx_tkeep` bus. In this example, `s_axi_tx_tkeep` is set to *FF* to indicate that all bytes are valid in the last data beat. The Aurora 64B/66B core sends the final word of data in data blocks, and must send an empty separator block on the next cycle to indicate the end of the frame. `s_axi_tx_tready` is reasserted on the next cycle so that more data transfers can continue. As long as there is no new data, the Aurora 64B/66B core sends idles.

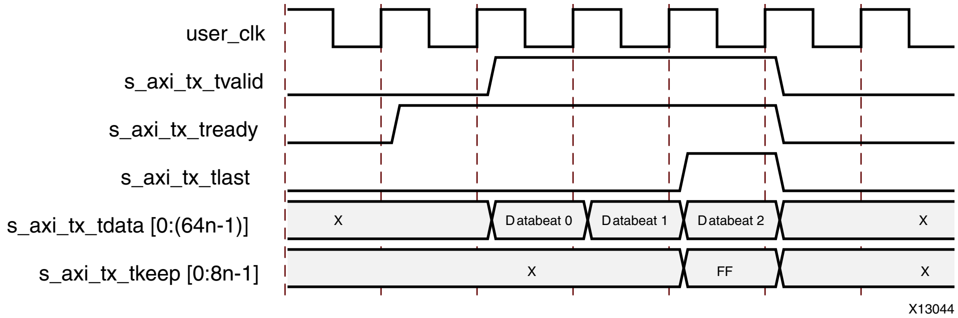


Figure 2-8: **Simple Data Transfer**

Example B: Data Transfer with Pause

Figure 2-9 shows how the user application can pause data transmission during a frame transfer. In this example, the user application is sending 3*n* bytes of data, and pauses the data flow after the first *n* bytes. After the first data word, the user application deasserts `s_axi_tx_tvalid`, causing the TX Aurora 64B/66B core to ignore all data on the bus and transmit idle blocks instead. The pause continues until `s_axi_tx_tvalid` is deasserted.

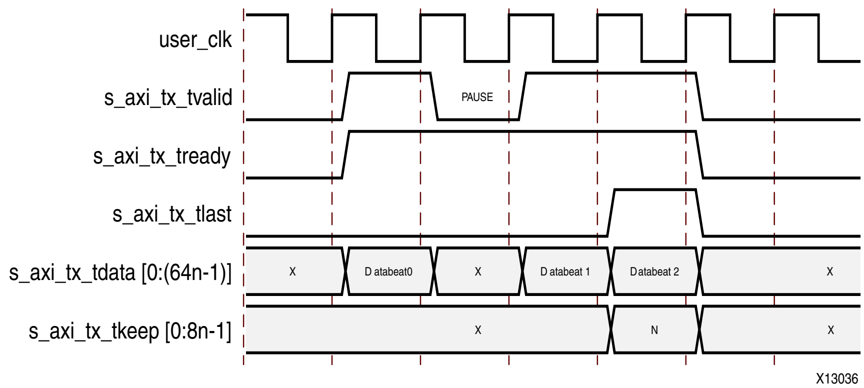
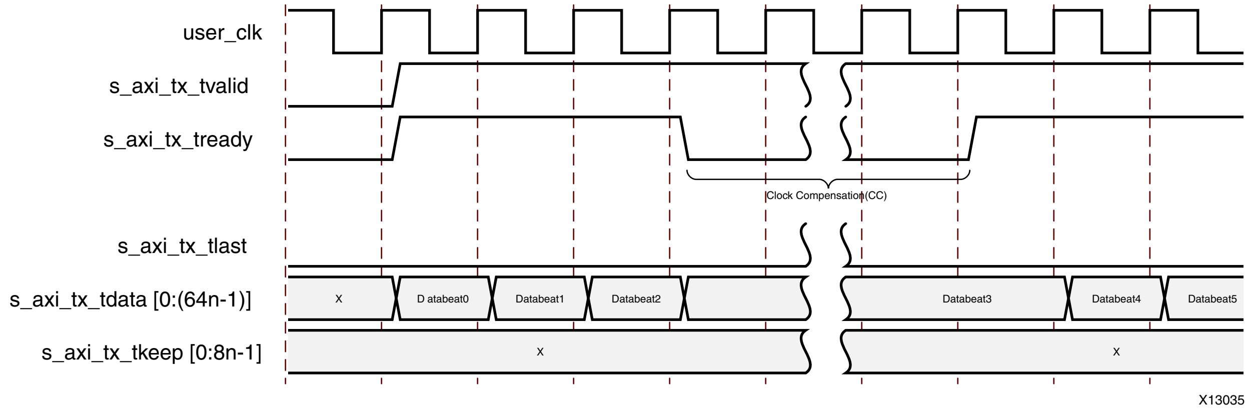


Figure 2-9: **Data Transfer with Pause**

Example C: Data Transfer with Clock Compensation

The Aurora 64B/66B core automatically interrupts data transmission when it sends clock compensation sequences. The clock compensation sequence imposes three cycles of PAUSE every 10,000 cycles.

Figure 2-10 shows how the Aurora 64B/66B core pauses data transmission during the clock compensation sequence.



Notes:

1. When clock compensation is used, uninterrupted data transmission is not possible. See Clock Compensation Interface, page 70 for more information about when clock compensation is required.

Figure 2-10: **Data Transfer Paused by Clock Compensation**

TX Interface Example

This section illustrates a simple example of an interface between a transmit FIFO and the AXI4-Stream interface of an Aurora 64B/66B core.

To review, to transmit data, the user application asserts `s_axi_tx_tvalid`, `s_axi_tx_tready` indicates that the data on the `s_axi_tx_tdata` bus is transmitted on the next rising edge of the clock, assuming `s_axi_tx_tvalid` remains asserted.

Figure 2-11 is a diagram of a typical connection between an Aurora 64B/66B core and the data source (in this example, a FIFO), including the simple logic needed to generate, `s_axi_tx_tvalid` and `s_axi_tx_tlast` from typical FIFO buffer status signals. While `reset` is FALSE, the example application waits for a FIFO to fill, then generates the `s_axi_tx_tvalid` signal. These signals cause the Aurora 64B/66B core to start reading the FIFO by asserting the `s_axi_tx_tready` signal.

The Aurora 64B/66B core encapsulates the FIFO data and transmits it until the FIFO is empty. Now the example application tells the Aurora 64B/66B core to end the transmission using the `s_axi_tx_tlast` signal.

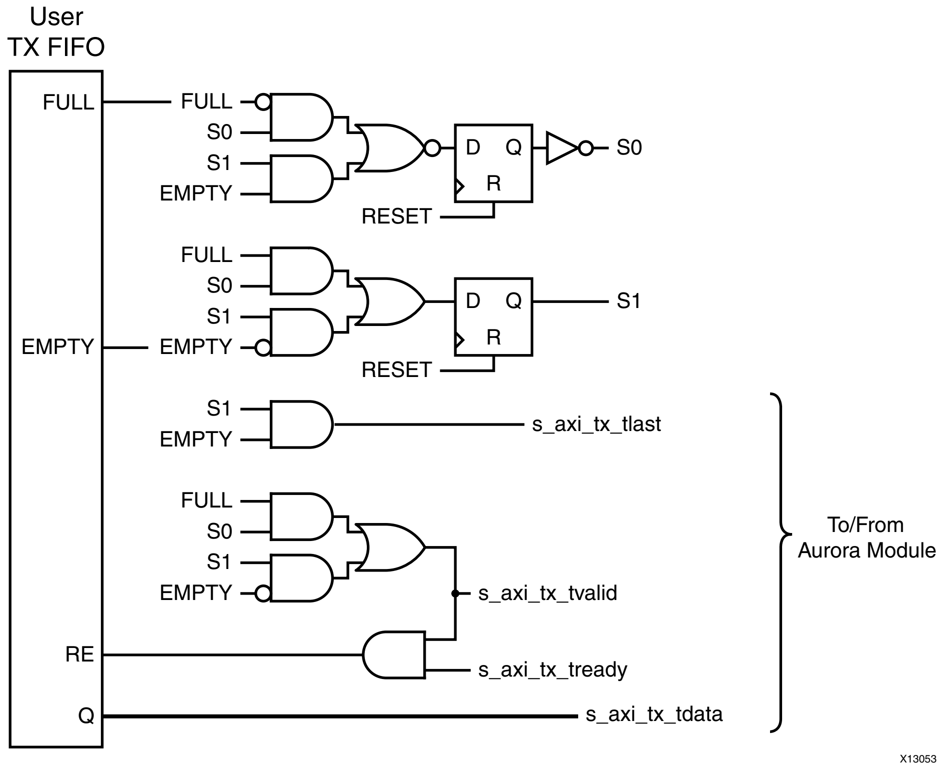


Figure 2-11: **Transmitting Data**

Receiving Data

When the Aurora 64B/66B core receives an Aurora 64B/66B frame, it presents it to the user application through the RX AXI4-Stream interface after discarding the control information, idle blocks, and clock compensation blocks.

The Aurora 64B/66B core has no built-in buffer for user data. As a result, there is no `m_axi_rx_tready` signal on the RX AXI4-Stream interface. The only way for the user application to control the flow of data from an Aurora channel is to use one of the core optional flow control features. In most cases, a FIFO should be added to the RX datapath to ensure no data is lost while flow control messages are in transit.

The Aurora 64B/66B core asserts the `m_axi_rx_tvalid` signal when the signals on its RX AXI4-Stream interface are valid. Applications should ignore any values on the RX AXI4-Stream ports sampled while `m_axi_rx_tvalid` is deasserted (active-Low).

The `m_axi_rx_tvalid` signal is asserted concurrently with the first word of each frame from the Aurora 64B/66B core, The `m_axi_rx_tlast` signal is asserted concurrently with the last word or partial word of each frame. The `m_axi_rx_tkeep` port indicates the number of valid bytes in the final word of each frame. It uses the same byte indication procedure as `s_axi_tx_tkeep` and indicates all bytes valid (all 1s) when `m_axi_rx_tkeep` is not asserted and specifies exact number of bytes valid when `m_axi_rx_tkeep` is asserted (active-High).

If the CRC option is selected, the received data stream is computed for the expected CRC value. This block re-calculates the m_axi_rx_tkeep value and asserts m_axi_rx_tlast correspondingly.

The Aurora 64B/66B core can deassert m_axi_rx_tvalid anytime, even during a frame.

Example A: Data Reception with Pause shows the reception of a typical Aurora 64B/66B frame.

Example A: Data Reception with Pause

Figure 2-12 shows an example of 3*n* bytes of received data interrupted by a pause. Data is presented on the m_axi_rx_tdata bus. When the first *n* bytes are placed on the bus, the m_axi_rx_tvalid output is asserted to indicate that data is ready for the user application. On the clock cycle following the first data beat, the core deasserts m_axi_rx_tvalid, indicating to the user application that there is a pause in the data flow.

After the pause, the core asserts m_axi_rx_tvalid and continues to assemble the remaining data on the m_axi_rx_tdata bus. At the end of the frame, the core asserts m_axi_rx_tlast. The core also computes the value of m_axi_rx_tkeep bus and presents it to the user application based on the total number of valid bytes in the final word of the frame.

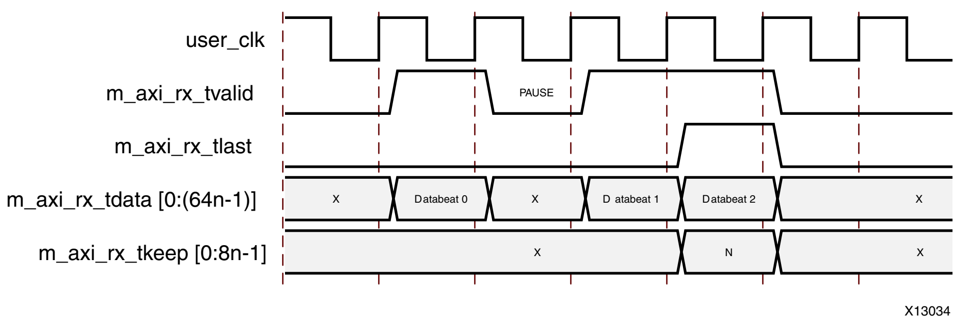


Figure 2-12: **Data Reception with Pause**

RX Interface Example

The RX AXI4-Stream interface of an Aurora 64B/66B core can be implemented with a simple FIFO. To receive data, the FIFO monitors the m_axi_rx_tvalid signal. When valid data is present on the m_axi_rx_tdata port, m_axi_rx_tvalid is asserted. Because m_axi_rx_tvalid is connected to the FIFO WE port, the data and framing signals are written to the FIFO.

Framing Efficiency

There are two factors that affect framing efficiency in the Aurora 64B/66B core:

- Size of the frame
- Data invalid request from gear box that occurs after every 32 user_clk cycles

The clock compensation (CC) sequence, which uses three `user_clk` cycles on every lane every 10,000 `user_clk` cycles, consumes about 0.03% of the total channel bandwidth.

The gear box in GTX and GTH transceivers requires periodic pause to account for the clock divider ratio and 64B/66B encoding. This appears as a back pressure in the AXI4-Stream interface and user data needs to be stopped for one cycle after every 32 cycles (Figure 2-13). The User Interface has the `s_axi_tx_tready` signal from the Aurora core being deasserted (active-Low) for one cycle once every 32 cycles. The pause cycle is used to compensate the Gearbox for the 64B/66B encoding.

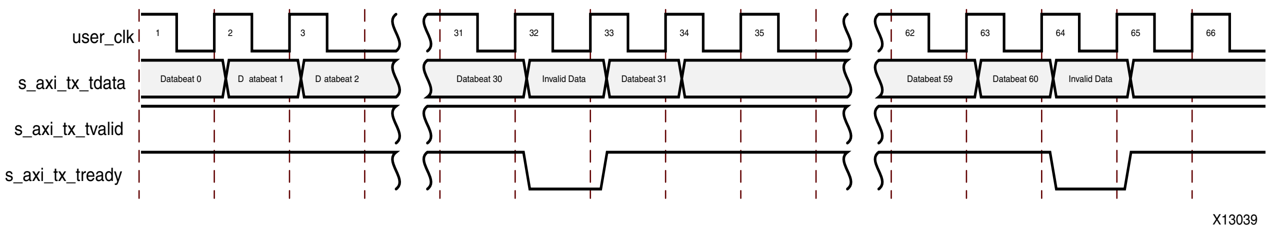


Figure 2-13: **Framing Efficiency**

For more information on gear box pause in GTX and GTH transceivers, see the *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476) [Ref 4] or *UltraScale FPGAs GTH Transceivers User Guide* (UG576) [Ref 3].

The Aurora 64B/66B core implements the Strict Aligned option of the Aurora 64B/66B protocol. No data blocks are placed after Idle blocks or SEP blocks on a given cycle. The restriction of not placing data blocks after SEP blocks reduces framing efficiency in a multilane Aurora 64B/66B core.

Table 2-18 is an example calculated after including overhead for clock compensation. It shows the efficiency for a single-lane channel and illustrates that the efficiency increases as frame length increases.

Table 2-18: **Efficiency Example**

User Data Bytes	Framing Efficiency %
100	96.12
1,000	99.18
10,000	99.89

Table 2-19 shows the overhead in single-lane channel when transmitting 256 bytes of frame data. The resulting data unit is 264 bytes long due to the SEP block used to end the frame. This results in 3.03% overhead in the transmitter. In addition, clock compensation blocks must be transmitted for three cycles every 10,000 cycles, resulting in an additional 0.03% overhead in the transmitter.

Table 2-19: **Typical Overhead for Transmitting 256 Data Bytes**

Lane	Clock	Function
[D0:D7]	1	Channel frame data
[D8:D15]	2	Channel frame data
. . .		
[D248:D255]	32	Channel frame data
Control block	33	SEP0 block

Streaming Interface

Figure 2-14 shows an example of an Aurora 64B/66B core configured with a streaming user interface.

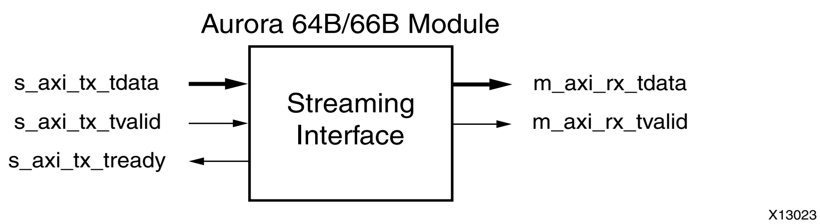


Figure 2-14: **Aurora 64B/66B Core Streaming User Interface**

Note: Width of `s_axi_tx_tdata` and `m_axi_rx_tdata` depends on **Little Endian** or **Big Endian** support from the Vivado IDE.

Transmitting and Receiving Data

The streaming interface allows the Aurora channel to be used as a pipe. Words written into the TX side of the channel are delivered, in order after some latency, to the RX side. After initialization, the channel is always available for writing, except when the `do_cc` signal is asserted to send clock compensation sequences. Applications transmit data through the `s_axi_tx_tdata` port, and use the `s_axi_tx_tvalid` port to indicate when the data is valid (asserted active-High). The streaming Aurora interface expects data to be filled for the entire `s_axi_tx_tdata` port width (integral multiple of eight bytes). The Aurora 64B/66B core deasserts `s_axi_tx_tready` (active-Low) when the channel is not ready to receive data. Otherwise, `s_axi_tx_tready` remains asserted.

When `s_axi_tx_tvalid` is deasserted, gaps are created between words. These gaps are preserved, except when clock compensation sequences are being transmitted. Clock compensation sequences are replicated or deleted by the CC logic to make up for frequency differences between the two sides of the Aurora channel. As a result, gaps created when `DO_CC` is asserted can shrink and grow. For details on the `do_cc` signal, see Clock Compensation Interface, page 70.

When data arrives at the RX side of the Aurora channel it is presented on the m_axi_rx_tdata bus and m_axi_rx_tvalid is asserted. The data must be read immediately or it will be lost. If this is unacceptable, a buffer must be connected to the RX interface to hold the data until it can be used.

Figure 2-15 shows a typical example of a streaming data transfer. The example begins with neither of the ready signals asserted, indicating that both the user logic and the Aurora 64B/66B core are not ready to transfer data. During the next clock cycle, the Aurora 64B/66B core indicates that it is ready to transfer data by asserting s_axi_tx_tready. One cycle later, the user logic indicates that it is ready to transfer data by asserting the s_axi_tx_tvalid signal and placing data on the s_axi_tx_tdata bus. Because both ready signals are now asserted, data D0 is transferred from the user logic to the Aurora 64B/66B core. Data D1 is transferred on the following clock cycle.

In this example, the Aurora 64B/66B core deasserts its ready signal, s_axi_tx_tready, and no data is transferred until the next clock cycle when, once again, the s_axi_tx_tready signal is asserted. Then the user application deasserts s_axi_tx_tvalid on the next clock cycle, and no data is transferred until both ready signals are asserted.

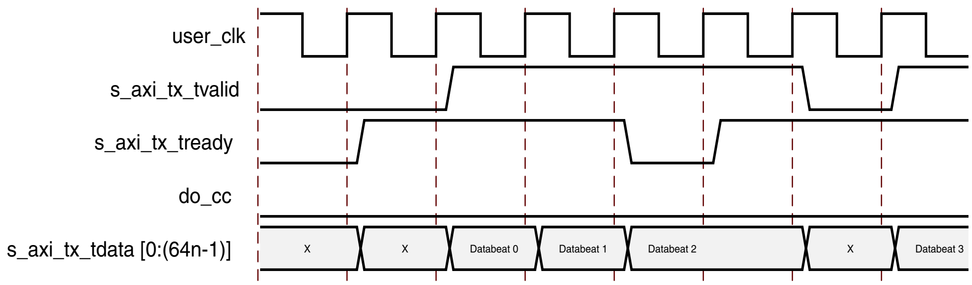


Figure 2-15: **Typical Streaming Data Transfer**

Figure 2-16 shows a typical example of streaming data reception.

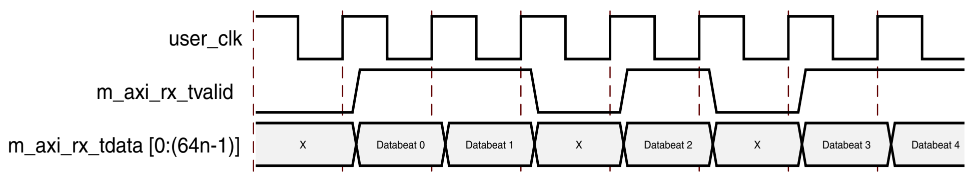


Figure 2-16: **Typical Streaming Data Reception**

Flow Control

This section explains how to use Aurora flow control. Two optional flow control interfaces are available. *Native flow control* (NFC) is used for regulating the data transmission rate at the receiving end of a full-duplex channel. *User flow control* (UFC) is used to accommodate high-priority messages for control operations.

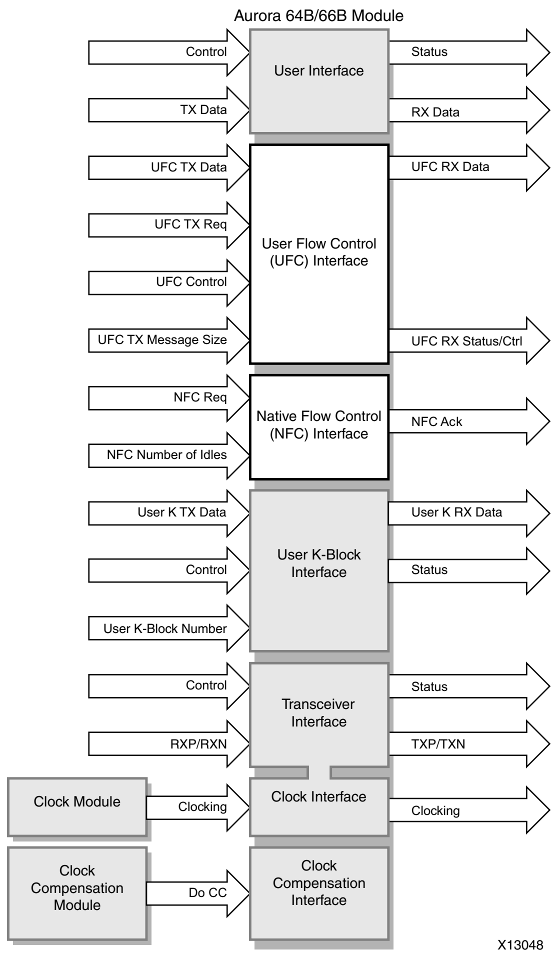


Figure 2-17: **Top-Level Flow Control**

Native Flow Control

The Aurora 64B/66B protocol includes native flow control (NFC) to allow receivers to control the rate at which data is sent to them by specifying a number of cycles that the channel partner cannot send data. The data flow can even be turned off completely by requesting that the transmitter temporarily send only idles (XOFF). NFC is typically used to prevent FIFO overflow conditions. For detailed explanation of NFC operation, see the *Aurora 64B/66B Protocol Specification v1.2* (SP011) [Ref 5].

Figure 2-18 and Figure 2-19 shows the NFC message format in default (Big Endian) mode and in Little Endian mode.

0:6 (don't care)	7 (nfc_xoff)	8:15 (data)

Figure 2-18: **NFC Message Format in Default Mode**

15:9 (don't care)	8 (nfc_xoff)	7:0 (data)

Figure 2-19: **NFC Message Format in Little Endian Mode**

NFC Message in Default Mode

To send an NFC message to a channel partner, the user application asserts `s_axi_nfc_tx_tvalid` and writes an 8-bit Pause count to `s_axi_nfc_tx_tdata[8:15]`. The pause code indicates the minimum number of cycles the channel partner must wait after receiving an NFC message before it can resume sending data. The number of `user_clk` cycles without data is equal to `s_axi_nfc_tx_tdata` + 1.

The signal `s_axi_nfc_tx_tdata[7]` indicates NFC_XOFF. Assert to send an NFC_XOFF message, requesting that the channel partner stop sending data until it receives a non-XOFF NFC message or reset. When a request is transmitted with PAUSE and XOFF both set to 0, NFC is set to XON mode. To turn off XOFF mode, a XON message (all 0s) should be transmitted; after reception of this XON request, any new NFC request will be honored by the core. The user application must hold `s_axi_nfc_tx_tvalid`, `s_axi_nfc_tx_tdata[8:15]`, and `s_axi_nfc_tx_tdata[7](nfc_xoff)` (if used) until `s_axi_nfc_tx_tready` is asserted on a positive `user_clk` edge, indicating the Aurora 64B/66B core will transmit the NFC message.

Aurora 64B/66B cores cannot transmit data while sending NFC messages. `s_axi_tx_tready` is always deasserted on the cycle following an `s_axi_nfc_tx_tready` assertion. NFC Completion mode is available only for the framing Aurora 64B/66B interface.

Example A: Transmitting an NFC Message

Figure 2-20 shows an example of the transmit timing when the user application sends an NFC message to a channel partner using a AXI4-Stream interface.

Note: Signal `s_axi_tx_tready` is deasserted for one cycle to create the gap in the data flow in which the NFC message is placed.

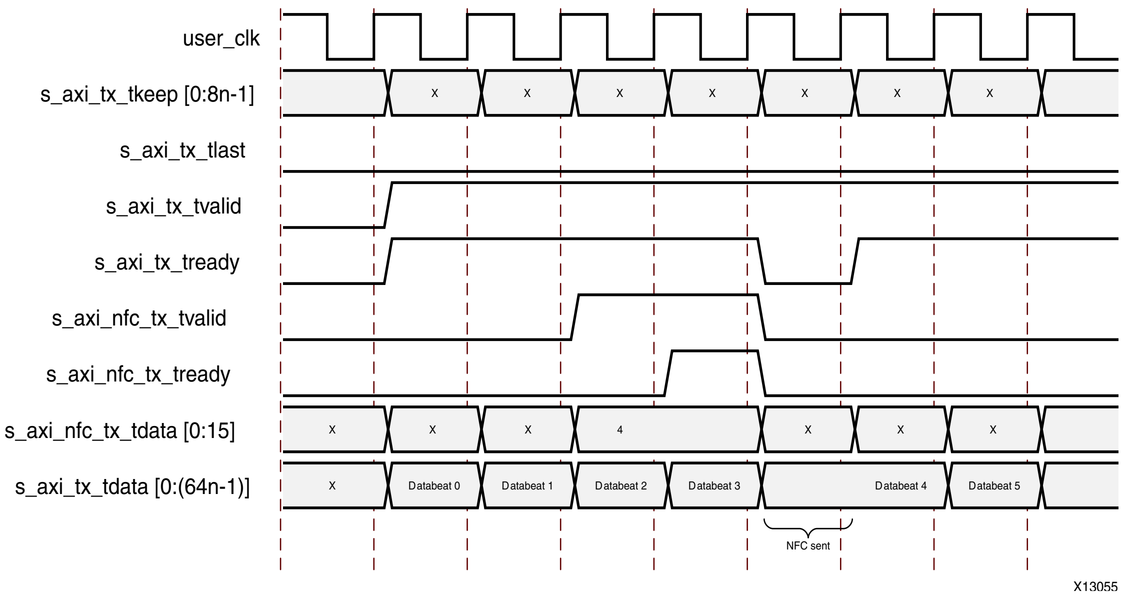


Figure 2-20: **Transmitting an NFC Message**

Example B: Receiving a Message with NFC Idles Inserted

Figure 2-21 shows an example of the signals on the TX user interface when an NFC message is received. In this case, the NFC message sends the number `8'b01`, requesting two cycles without data transmission. The core deasserts `s_axi_tx_tready` on the user interface to prevent data transmission for two cycles. In this example, the core is operating in Immediate NFC mode. Aurora 64B/66B cores can also operate in completion mode, where NFC Idles are only inserted before the first data bytes of a new frame. If a completion mode core receives an NFC message while it is transmitting a frame, it finishes transmitting the frame before deasserting `s_axi_tx_tready` to insert idles.

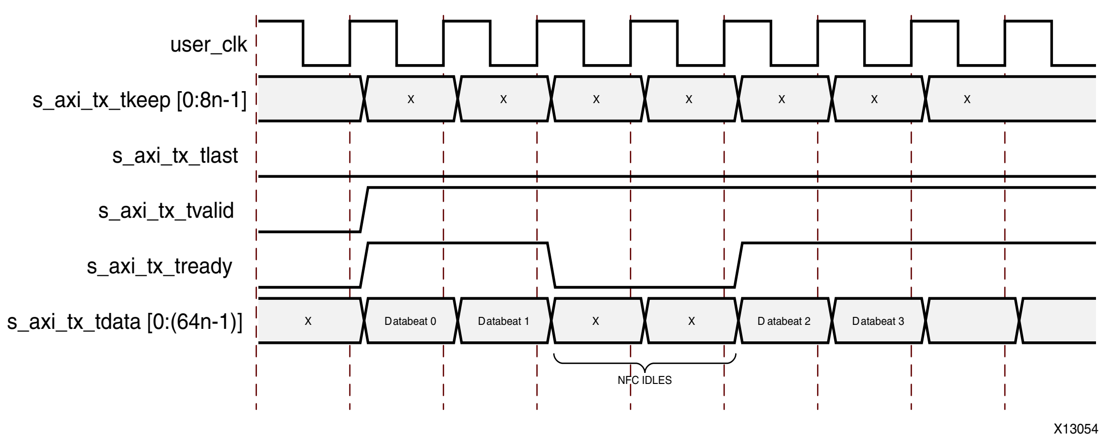


Figure 2-21: **Transmitting a Message with NFC Idles Inserted**

User Flow Control

The Aurora 64B/66B protocol includes user flow control (UFC) to allow channel partners to send control information using a separate in-band channel. The user application can send short UFC messages to the channel partner of the core without waiting for the end of a frame in progress. The UFC message shares the channel with regular frame data, but has a higher priority than frame data. UFC messages are interruptible by high-priority control blocks such as CC/NR/CB/NFC blocks.

Transmitting UFC Messages

UFC messages can carry from 1 to 256 data bytes. The user application specifies the length of the message by driving the number of bytes required minus one on the `ufc_tx_ms` port. For example, a value of 3 will transmit 4 bytes of data; and a value of 0 will transfer 1 byte.

To send a UFC message, the user application asserts `ufc_tx_req` while driving the `ufc_tx_ms` port with the desired SIZE code for a single cycle. After a request, a new request cannot be made until `s_axi_ufc_tx_tready` is asserted for the final cycle of the previous request. The data for the UFC message must be placed on the `s_axi_ufc_tx_tdata` port and the `s_axi_ufc_tx_tvalid` signal must be asserted whenever the bus contains valid message data.

The core deasserts `s_axi_tx_tready` while sending UFC data, and keeps `s_axi_ufc_tx_tready` asserted until it has enough data to complete the message that was requested. If `s_axi_ufc_tx_tvalid` is deasserted during a UFC message, Idles are sent in the channel, `s_axi_tx_tready` remains deasserted, and `s_axi_ufc_tx_tready` remains asserted. If a CC request, CB request, or NFC request is made to the core, `s_axi_ufc_tx_tready` is deasserted while the requested operation is performed, because CC, CB, and NFC have higher priority.

Example A: Transmitting a Single-Cycle UFC Message

The procedure for transmitting a single cycle UFC message is shown in Figure 2-22. In this case a 4-byte message is being sent on an 8-byte interface.

Note: Signals `s_axi_tx_tready` and `s_axi_ufc_tx_tready` are deasserted for a cycle before the core accepts message data: this cycle is used to send the UFC header.

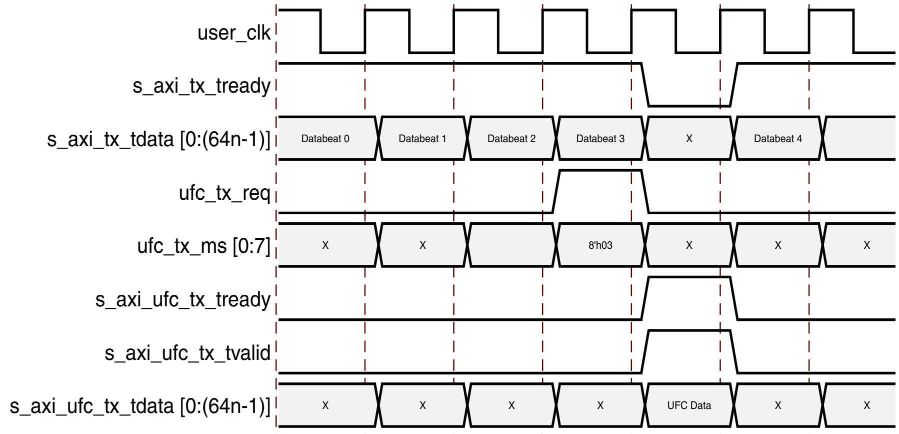


Figure 2-22: **Transmitting a Single-Cycle UFC Message**

Example B: Transmitting a Multicycle UFC Message

The procedure for transmitting a two-cycle UFC message is shown in Figure 2-23. In this case the user application is sending a 16-byte message using an 8-byte interface.

The `s_axi_ufc_tx_tready` signal is asserted for two cycles to transmit UFC data.

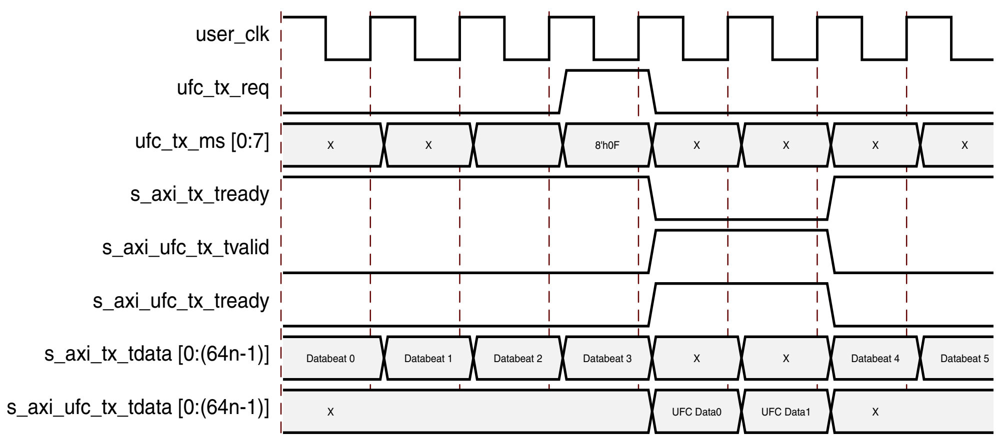


Figure 2-23: **Transmitting a Multi-Cycle UFC Message**

Receiving User Flow Control Messages

When the Aurora 64B/66B core receives a UFC message, it passes the data from the message to the user application through a dedicated UFC AXI4-Stream interface. The data is presented on the `m_axi_ufc_rx_tdata` port; assertion of `m_axi_ufc_rx_tvalid` indicates the start of the message data and `m_axi_ufc_rx_tlast` indicates the end. `m_axi_ufc_rx_tkeep` is used to show the number of valid bytes on `m_axi_ufc_rx_tdata` during the last cycle of the message (for example, while m_axi_ufc_rx_tlast is asserted). Signals on the ufc_rx AXI4-Stream interface are only valid when m_axi_ufc_rx_tvalid is asserted.

Example C: Receiving a Single-Cycle UFC Message

Figure 2-24 shows an Aurora 64B/66B core with an 8-byte data interface receiving a 4-byte UFC message. The core presents this data to the user application by asserting m_axi_ufc_rx_tvalid and m_axi_ufc_rx_tlast to indicate a single cycle frame. The m_axi_ufc_rx_tkeep bus is set to 4, indicating only the four most significant bytes of the interface are valid.

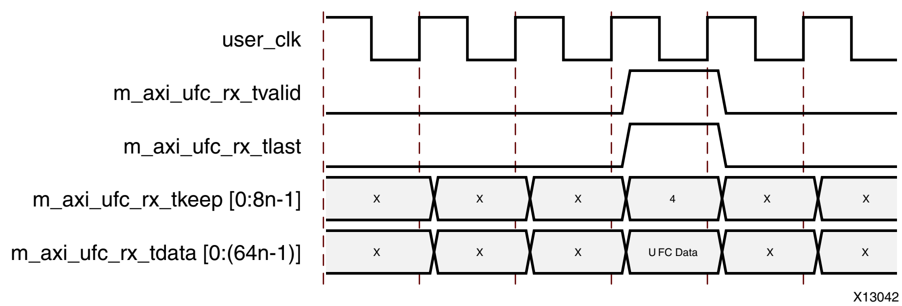


Figure 2-24: **Receiving a Single-Cycle UFC Message**

Example D: Receiving a Multicycle UFC Message

Figure 2-25 shows an Aurora 64B/66B core with an 8-byte interface receiving a 15-byte message.

Note: The resulting frame is two cycles long, with m_axi_ufc_rx_tkeep set to 7 on the second cycle indicating that all seven bytes of the data are valid.

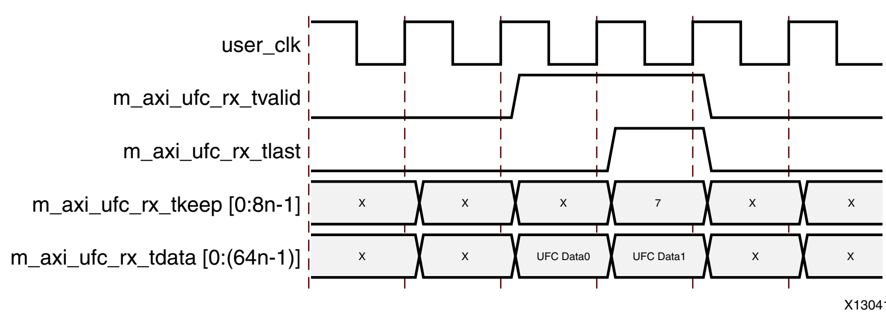


Figure 2-25: **Receiving a Multi-Cycle UFC Message**

User K-Block Interface

This section describes short single block data transmission and reception.

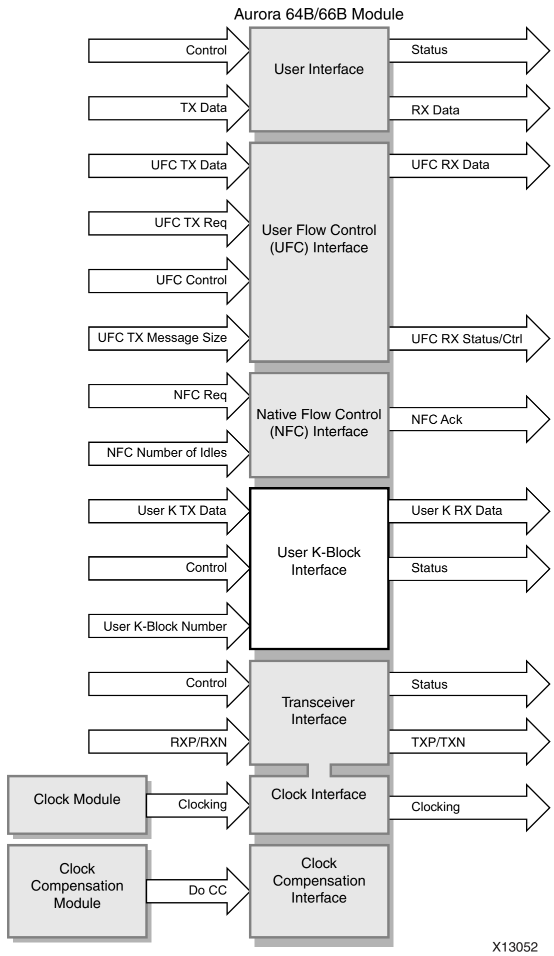


Figure 2-26: **Top-Level User K-Block Interface**

User K-blocks are special single block codes which include control blocks that are not decoded by the Aurora interface, but are instead passed directly to the user application. These blocks can be used to implement application-specific control functions. There are nine available User K-blocks (Table 2-20). Their priority is lower than UFC but higher than user data.

Table 2-20: **Valid Block Type Field (BTF) Values for User K-Block**

User K-Block Name	User K-Block BTF
User K-Block 0	0xD2
User K-Block 1	0x99
User K-Block 2	0x55
User K-Block 3	0xB4
User K-Block 4	0xCC
User K-Block 5	0x66
User K-Block 6	0x33
User K-Block 7	0x4B
User K-Block 8	0x87

The User K-block is not differentiated for streaming or framing designs. Each block code of User K is eight bytes wide and is encoded with a User K BTF, which is indicated by the user application in `s_axi_user_k_tx_tdata` as User K Block No. The User K-block is a single block code and is always delineated by User K Block No. You should provide the User K Block No as specified in Table 2-9, page 21. It can have only seven bytes of `s_axi_user_k_tdata`.

Figure 2-27 and Figure 2-28 shows the User K format in default (Big Endian) mode and in Little Endian mode.

0:3 (zeros)	4:7 (userk)	8:63 (data)

Figure 2-27: **User K format in Default Mode**

63:8 (data)	7:4 (zeros)	3:0 (userk)

Figure 2-28: **User K format in Little Endian Mode**

Transmitting User K-Blocks

The s_axi_user_k_tx_tready signal is asserted by Aurora and is prioritized by CC, CB, NFC, and UFC. After placing s_axi_user_k_tx_tdata and along with User K Block No and s_axi_user_k_tx_tvalid is asserted, the user application can change s_axi_user_k_tx_tdata if required when s_axi_user_k_tx_tready is asserted (Figure 2-29). This enables the Aurora core to select appropriate User K BTF among the nine User K-blocks. The data available during assertion of s_axi_user_k_tx_tready is always serviced.

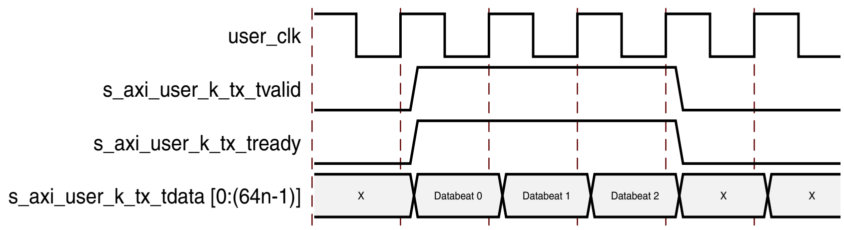


Figure 2-29: **Transmitting User K Data and User K-Block Number**

Receiving User K-Blocks

The receive BTF is decoded and the block number for the corresponding BTF is passed on to the user application as such (Figure 2-30). The user application can validate the m_axi_rx_user_k_tdata available on the bus when m_axi_rx_user_k_tvalid is asserted.

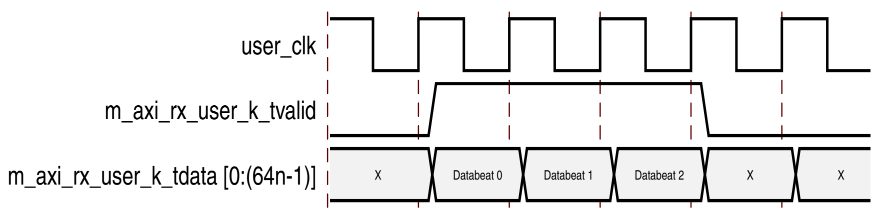


Figure 2-30: **Receiving User K Data and User K-Block Number**

Status, Control, and the Transceiver Interface

The status and control ports of the Aurora 64B/66B core allow user applications to monitor the Aurora channel and use built-in features of the serial transceiver interface. This section provides diagrams and port descriptions for the Aurora 64B/66B core status and control interface, along with the GTX and GTH serial I/O interface.

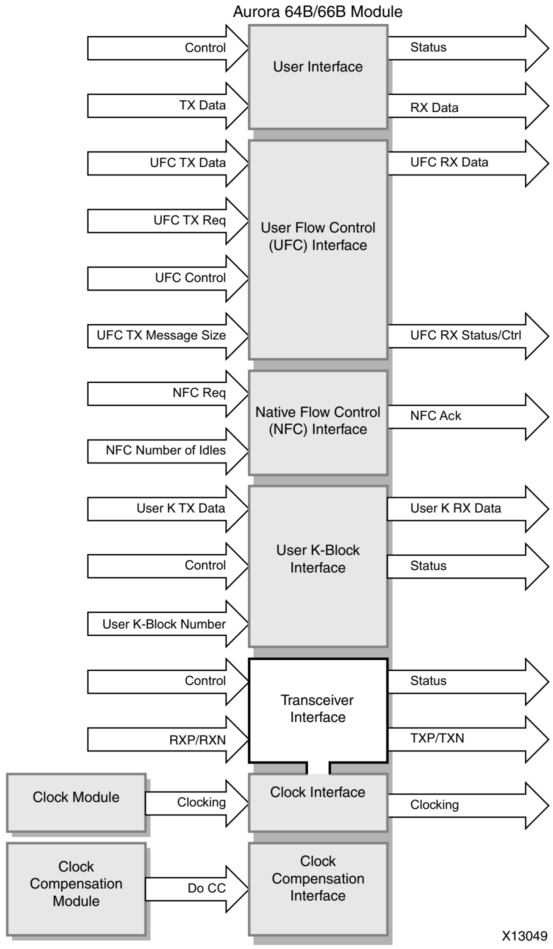


Figure 2-31: **Top-Level GTX Interface**

Status and Control Ports

Aurora 64B/66B cores are full-duplex/simplex, and provide a TX and an RX Aurora channel connection. The Aurora 64B/66B core does not require any sideband signals for simplex mode of operation. Figure 2-32 shows the status and control interface for an Aurora 64B/66B core.

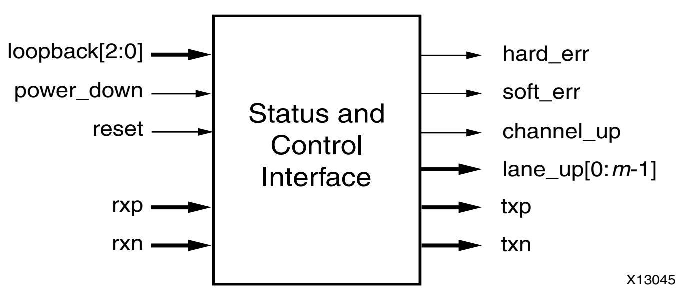


Figure 2-32: **Status and Control Interface for the Aurora 64B/66B Core**

Error Signals in Aurora 64B/66B Cores

Equipment problems and channel noise can cause errors during Aurora channel operation. The 64B/66B encoding allows the Aurora 64B/66B core to detect some bit errors that occur in the channel. The core reports these errors by asserting the `soft_err` signal on every cycle they are detected.

The core also monitors each high-speed serial GTX and GTH transceiver for hardware errors such as buffer overflow and loss of lock. The core reports hardware errors by asserting the `hard_err` signal. Catastrophic hardware errors can also manifest themselves as burst of soft errors. The core uses the Block Sync algorithm described in the *Aurora 64B/66B Protocol Specification v1.2* (SP011) [Ref 5] to determine whether to treat a burst of soft errors as a hard error.

Whenever a hard error is detected, the Aurora 64B/66B core automatically resets itself and attempts to re-initialize. In most cases, this allows the Aurora channel to be reestablished as soon as the hardware issue that caused the hard error is resolved. Soft errors do not lead to a reset unless enough of them occur in a short period of time to trigger the block sync state machine.

Table 2-21: **Error Signals in Full Duplex Cores**

Signal	Description
hard_err/ tx_hard_err/ rx_hard_err	**TX Overflow/Underflow**: The elastic buffer for TX data overflows or underflows. This can occur when the user clock and the reference clock sources are not running at the same frequency. **RX Overflow/Underflow**: The clock correction and channel bonding FIFO for RX data overflows or underflows. This can occur when the clock source frequencies for the two channel partners are not within ±100 ppm.
soft_err/ tx_soft_err/ rx_soft_err	**Soft Errors**: There are too many soft errors within a short period of time. The block sync state machine used for alignment automatically attempts to realign if too many invalid sync headers are detected. Soft Errors will not be transformed into Hard Errors. **Invalid SYNC Header**: The 2-bit header on the 64-bit block was not a valid control or data header. **Invalid BTF**: A control block was received with an unrecognized value in the block type field (BTF). This is usually the result of a bit error.

Initialization

Aurora 64B/66B cores initialize automatically after power-up, reset, or hard error. Aurora 64B/66B core modules on each side of the channel perform the Aurora initialization procedure until the channel is ready for use. The `lane_up` bus indicates which lanes in the channel have finished the lane initialization portion of the initialization procedure. This signal can be used to help debug equipment problems in a multi-lane channel. `channel_up` is asserted only after the core completes the entire initialization procedure.

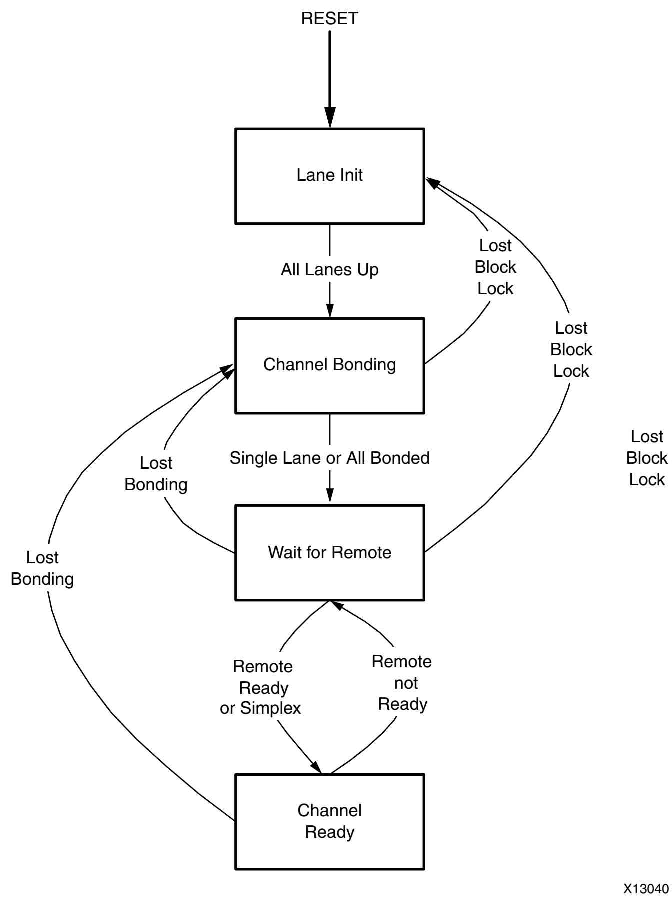


Figure 2-33: **Initialization Overview**

Aurora 64B/66B cores can receive data before `channel_up` is asserted. Only the `m_axi_rx_tvalid` signal on the user interface should be used to qualify incoming data. `channel_up` can be inverted and used to reset modules that drive the TX side of a full-duplex channel, because no transmission can occur until after `channel_up`. If user application modules need to be reset before data reception, one of the `lane_up` signals can be inverted and used. Data cannot be received until after all the `lane_up` signals are asserted.

Aurora Simplex Operation

Simplex Aurora 64B/66B cores do not have any sideband connection and use timers to declare that the partner is out of initialization and is ready for data transfer.

Simplex TX/RX cores are that which have both transmit and receive portions of the GT configured to operate independently. However, the Simplex TX/RX cores have reset and `pma_init` common between the transmit and receive path of the core.

The user application can modify the timer value based on the channel requirement. For Simplex links, it is expected that `rx_channel_up` is asserted before `tx_channel_up` is asserted. This will ensure that Simplex RX is ready to receive before Simplex TX is operational.

TX Lane Up is asserted based on a 24-bit counter to account for Block Lock and CDR lock times of the Simplex RX link. Depending on deassertion time delta between TX/RX RESET or PMA_INITs, the SIMPLEX_TIMER_VALUE parameter in Simplex TX has to be adjusted to meet the preceding criteria. The SIMPLEX_TIMER_VALUE parameter can be updated in <user_component_name>_core.v.

- If `tx_reset` is deasserted after `rx_reset`, the default value of 12 bits is sufficient for the link to be operational.
- If `tx_reset` is deasserted before `rx_reset`, the SIMPLEX_TIMER_VALUE parameter in Simplex TX has to accommodate the delay in the reset deassertion time.

Reset and Power Down

Reset

The `reset` signals on the control and status interface are used to set the Aurora 64B/66B core to a known starting state. Resetting the core stops any channels that are currently operating; after reset, the core attempts to initialize a new channel. When Reset on Aurora channel partner1 is asserted, channel partner2 will also lose lock. Channel Partner2 will regain lock when partner1 is out of reset and transmits valid patterns.

On full-duplex modules, the `reset` signal resets both the TX and RX sides of the channel when asserted on the positive edge of `user_clk`. Simplex Aurora cores have respective reset ports. Asserting `pma_init` will reset the entire serial transceiver which will eventually reset Aurora core also.

Reset Sequencing

Following is the recommended reset sequence for the Aurora 64B/66B core at the example design level. See Figure 2-34.

1. Assert reset. Wait for a minimum of 128 `user_clk` clock cycle times.
2. Assert `pma_init`. Keep `pma_init` and `reset` asserted for at least one second; this ensures that there is no transmission of CC characters, making sure the remote agent will detect a hot plug event. See Hot-Plug Logic in Chapter 3.
3. Deassert `pma_init`.
4. Deassert `reset`.

Notes:

1. The preceding reset sequence is implemented in `<user_component_name>_exdes.v` for reference.
2. For Simplex use cases, TX reset has to be asserted first then followed by RX reset. This ensures that whenever TX reset is asserted, Simplex-TX will send NA idle characters. The Channel partner (Simplex-RX) will receive these characters and the link will shut down gracefully.
3. For TX/RX Simplex cores, assertion of `reset` and `pma_init` input ports will reset both the TX and RX portions of the core and GT respectively. The `reset` and `pma_init` connection is similar to that of duplex cores.

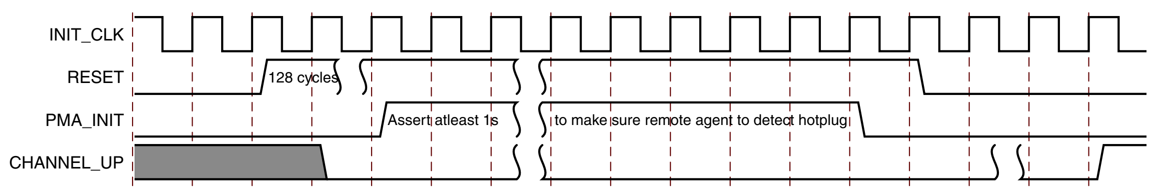


Figure 2-34: **Reset Sequencing**

pma_init Staging

The top level `pma_init` input at the example design level is delayed for 128 cycles (`pma_init_stage`). This signal is pulse stretched for a 24-bit counter time (`pma_init_assertion`). An aggregated signal from above is provided to the core as the `pma_init` input. This is to make sure that `pma_init` assertion to the core will result in `reset` assertion to the entire core also.

Inside the `<user_component_name>_support_reset_logic.v`, the debouncer logic (reset_debounce_r) will remain in the reset state until the `gt_reset_in` signal (`pma_init_assertion`) signal is High. This ensures that there is an internally generated reset whenever the top level `pma_init` is asserted.

Figure 2-35 shows the behavior.

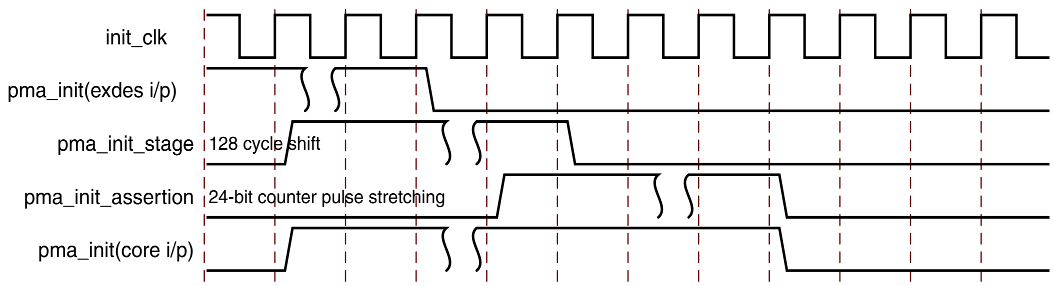


Figure 2-35: **pma_init Staging**

Assertion of `pma_init` to the core will result in hot-plug reset assertion in the channel partner core. The reset sequence after hot-plug reset assertion is shown in Figure 2-36.

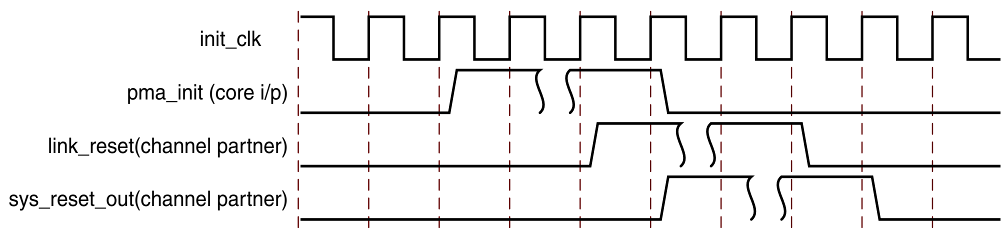


Figure 2-36: **pma_init to Remote System Reset**

Reset Flow

The top level `reset` input at the example design level is debounced and connected to the core (`reset_pb`). This signal is aggregated along with the serial transceiver reset status and the hot-plug reset from the core in the core reset logic to generate a reset to the core (`sys_reset_out`). This signal is expected to be connected to the core `reset` input. Figure 2-37 shows the behavior.

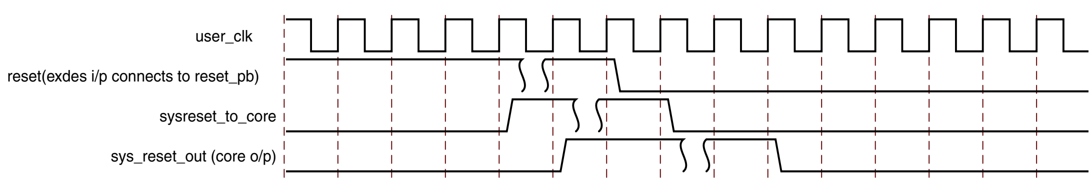


Figure 2-37: **Reset Flow**

Note:

1. `reset_pb` and `reset` at the input of the core should not be tied together, to account for the preceding requirement.
2. `sys_reset_out` should be used to drive the reset input to the core, along with additional system specific resets, if any.

Power Down

When power_down is asserted, only the Aurora 64B/66B core logic will be in reset. This does not turn off the GTX or GTH transceivers used in the design.

Timing

Figure 2-38 shows the timing for the reset signal. In a quiet environment, t_{CU} is generally less than 500 clocks; In a noisy environment, t_{CU} can be much longer.

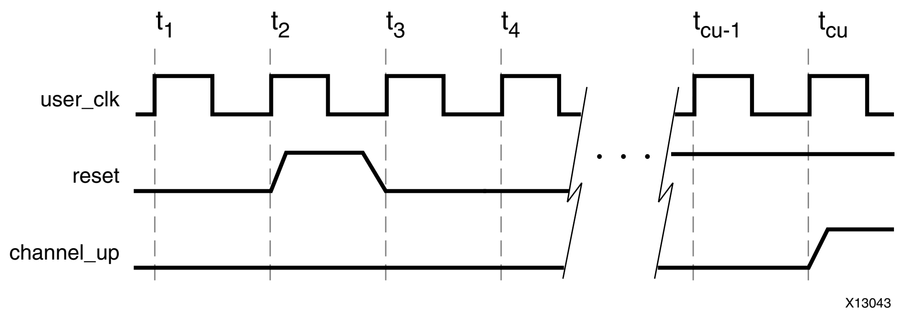


Figure 2-38: **Reset and Power Down Timing**

Reset Use Cases

Use Case 1: reset assertion in duplex core

The reset assertion in the duplex core should be a minimum of 128 user_clk cycles. In effect to this, channel_up will be deasserted as shown in the Figure 2-39.

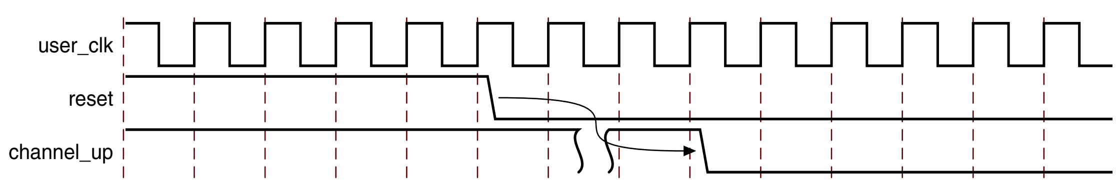


Figure 2-39: **Assertion of reset in the Duplex Core**

Use Case 2: PMA_INIT assertion in duplex core

Figure 2-40 shows the pma_init assertion in the duplex core and should be a minimum of 128 init_clk cycles. As a result, user_clk will be stopped after a few clock cycles because there is no txoutclk from the transceiver and channel_up will be deasserted.

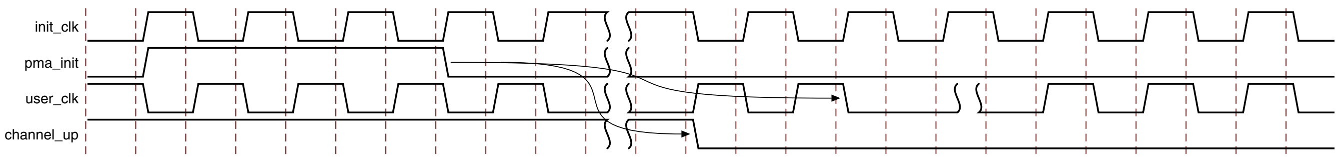


Figure 2-40: **pma_init Assertion in the Duplex Core**

Use Case 3: Assertion of reset in the Simplex Core

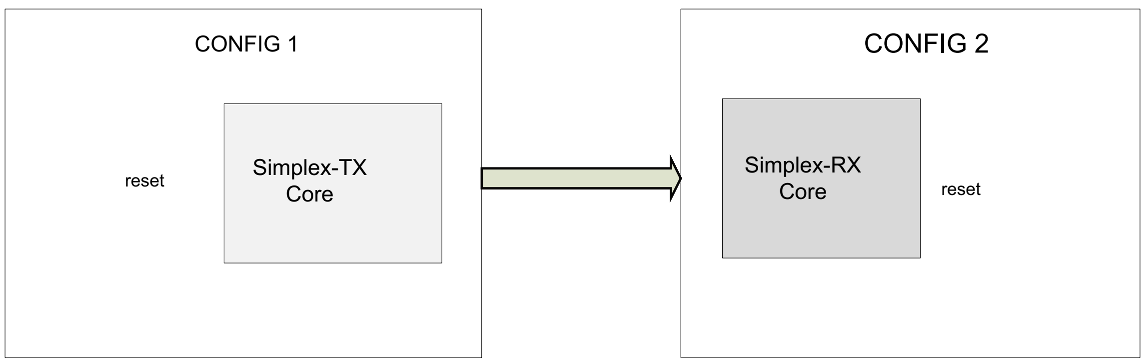


Figure 2-41: **System with Simplex Cores**

Figure 2-41 shows the Simplex-TX core and Simplex-RX core connected in a system. CONFIG1 and CONFIG2 can be in same or multiple device(s).

Following is the recommended procedure of TX cores reset and RX cores reset assertion in the Simplex core.

1. The signal RX cores reset is asserted for 128 user_clk cycles followed by reset on the RX Simplex core asserted for 128 user_clk cycles.
2. tx_channel_up and rx_channel_up are deasserted after a minimum of five user_clk clock cycles.
3. The signal reset in the RX Simplex core is deasserted (or) released before reset in the TX Simplex core is deasserted. This will ensure that transceiver in the Simplex-RX core will have sufficient transitions for CDR lock before the Simplex-TX core achieves TX_CHANNELUP.

4. `rx_channel_up` is asserted before `tx_channel_up` assertion. This condition must be satisfied by Simplex-RX core and simplex timer parameters (SIMPLEX_TIMER_VALUE) in Simplex-TX core needs to be adjusted to meet this criteria. The SIMPLEX_TIMER_VALUE parameter can be updated in `<user_component_name>_core.v`.
5. `tx_channel_up` is asserted after Simplex-TX core completes the Aurora protocol channel initialization sequence transmission for configured time. Assertion of `tx_channel_up` last will ensure that the Simplex-TX core will transmit an Aurora initialization sequence when Simplex-RX core is ready.

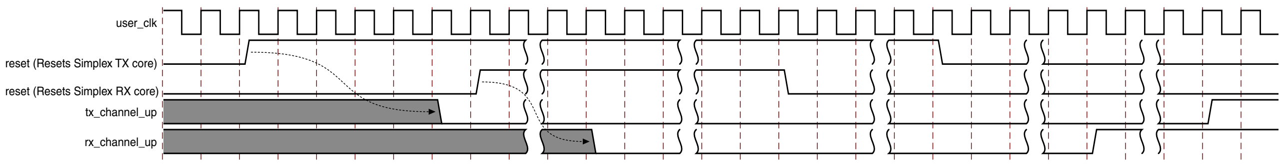


Figure 2-42: **Reset Assertion in Simplex Cores**

6. In the case of TX/RX Simplex cores, the reset sequence in duplex cores for `reset` and `pma_init` assertions can be followed. However, the SIMPLEX_TIMER_VALUE needs to be tuned based on the use model of the core.

DRP Interface

The DRP interface controls or monitors the status of the transceiver block. The user application can access or update the serial transceiver settings by writing/reading the values through the DRP ports. The Native interface provides the native transceiver DRP interface. The AXI4-Lite interface can also be selected to access the DRP ports through it.

Table 2-22: **AXI4-Lite Signal Definitions**

Name	Direction	Description
s_axi_awaddr	Input	AXI4-Lite Write address for DRP
s_axi_awvalid	Input	Write address valid
s_axi_awready	Output	Write address ready
s_axi_araddr	Input	Read address
s_axi_arvalid	Input	Read address valid
s_axi_arready	Output	Read address ready
s_axi_wdata	Input	Write data
s_axi_wvalid	Input	Write valid
s_axi_wready	Output	Write ready
s_axi_bvalid	Output	Write response valid
s_axi_rdata	Output	Read data
s_axi_rvalid	Output	Read valid

Table 2-22: **AXI4-Lite Signal Definitions *(Cont'd)***

Name	Direction	Description
s_axi_rready	Input	Read ready
s_axi_bready	Input	Write response ready

Note: The core expects the user AXI4-Lite interface to be ready to take the data when the DRP read operation is initiated.

Table 2-23: **DRP Port Signal Definitions**

Port	Direction	Clock Domain	Description
drpaddr[8:0]	Input	DRPCLK	DRP address bus
drpclk	Input	N/A	DRP interface clock
drpen	Input	DRPCLK	DRP enable signal 0: No read or write operation performed 1: Enables a read or write operation For write operations, `drpwe` and `drpen` should be driven High for one `drpclk` cycle only. See Figure 2-31 for correct operation.
drpdi[15:0]	Input	DRPCLK	Data bus for writing configuration data from the FPGA logic resources to the transceiver.
drprdy	Output	DRPCLK	Indicates operation is complete for write operations and data is valid for read operations.
drpdo[15:0]	Output	DRPCLK	Data bus for reading configuration data from the GTX or GTH transceiver to the FPGA logic resources.
drpwe	Input	DRPCLK	DRP write enable 0: Read operation when `drpen` is 1. 1: Write operation when `drpen` is 1. For write operations, `drpwe` and `drpen` should be driven High for one `drpclk` cycle only.

Note: For UltraScale devices the DRP port names starts with `gt<lane>_drp*`. where lane = number of lanes.

The DRP interface will assert `drpen` when the Write Address or Read Address channel from the AXI4-Lite interface is active with the respective Valid/Ready signals asserted. The `drpwe` signal for write operation is enabled when the Write Data channel from the AXI4-Lite interface is active. When the Read Data channel from AXI4-Lite is enabled, `drpdo` will have the data requested for the address specified through `drpaddr`.

Clock Compensation Interface

This interface is included in modules that transmit data, and is used to manage clock compensation. Whenever the do_cc port is driven High, the core stops the flow of data and flow control messages, then sends clock compensation sequences. Each Aurora 64B/66B core is accompanied by a clock compensation management module that is used to drive the clock compensation interface in accordance with the *Aurora 64B/66B Protocol Specification v1.2* (SP011) [Ref 5]. When the same physical clock is used on both sides of the channel and hot-plug logic is disabled, do_cc should be tied Low. However it is highly recommended to have CC logic enabled for reliable operation of the link.

All Aurora 64B/66B cores include a clock compensation interface for controlling the transmission of clock compensation sequences. Table 2-24 describes the function of the clock compensation interface ports.

Table 2-24: **Clock Compensation I/O Ports**

Name	Direction	Description
do_cc	Input	The Aurora 64B/66B core sends CC sequences on all lanes on every clock cycle when this signal is asserted. Connects to the do_cc output on the CC module.

Chapter 3

Designing with the Core

This chapter includes guidelines and additional information to make designing with the core easier.

General Design Guidelines

All Aurora 64B/66B core implementations require careful attention to system performance requirements. Pipelining, logic mappings, placement constraints and logic duplications are all methods that help boost system performance.

Keep It Registered

To simplify timing and increase system performance in an FPGA design, keep all inputs and outputs registered between the user application and the core. This means that all inputs and outputs from user application should come from or connect to a flip-flop. While registering signals might not be possible for all paths, it simplifies timing analysis and makes it easier for the Xilinx tools to place-and-route the design.

Recognize Timing Critical Signals

The XDC file provided with the example design for the core identifies the critical signals and the timing constraints that should be applied.

Use Supported Design Flows

The core is delivered as Verilog source code. The example implementation scripts provided currently use XST as synthesis tool for the example design that is delivered with the core. Other synthesis tools can be used.

Make Only Allowed Modifications

The Aurora 64B/66B core is not user modifiable. Any modifications might have adverse effects on the system timings and protocol compliance. Supported user configurations of the Aurora 64B/66B core can only be made by selecting options from the IP catalog.

Shared Logic

Up to version 8.1 of the core, the RTL hierarchy for the core was fixed. This resulted in some difficulties because shareable clocking and reset logic needed to be extracted from the core example design for use with a single instance or multiple instances of the core.

Shared logic is a new feature that provides a more flexible architecture that works both as a standalone core and as a part of a larger design with one or more core instances. This minimizes the amount of HDL modifications required, but at the same time retains the flexibility to address more use cases.

The new level of hierarchy is called <user_component_name>_support. Figure 3-1 and Figure 3-2 show two hierarchies where the shared logic block is contained either in the core or in the example design. In these figures, <user_component_name> is the name of the generated core. The difference between the two hierarchies is the boundary of the core. It is controlled using the **Shared Logic** option in the Vivado® IDE.

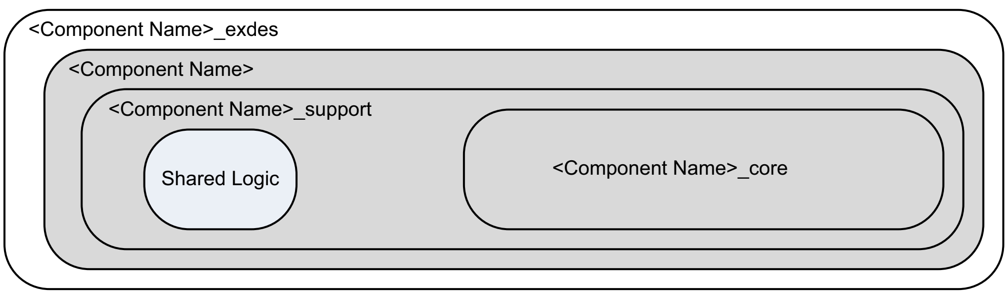


Figure 3-1: **Shared Logic Included in Core (highlighted in gray is the xci top)**

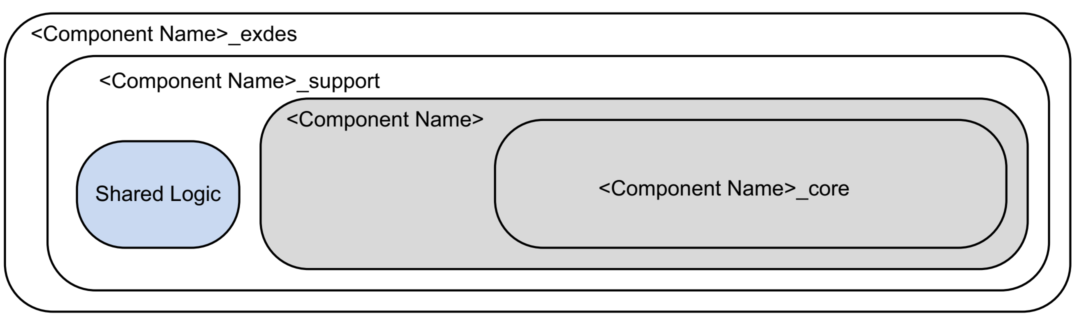


Figure 3-2: **Shared Logic Included in Example Design (highlighted in gray is the xci top)**

The contents of the shared logic depend upon the physical interface and the target device. Shared logic will contain instance(s) of the GT differential buffer, support reset logic and instantiation of `<=: USER_COMPONENT_NAME:>_CLOCK_MODULE`. In addition to these blocks, shared logic will also contain an instance of transceiver common. The transceiver common is instantiated based on the selected transceiver type (GTX or GTH). Support reset logic contains the de-bouncer logic for the reset and `gt_reset` ports.

Table 3-1 provides the details about the port changes due to **Shared Logic** option.

Table 3-1: **Port Changes Due to Shared Logic Option**

NAME	Direction	Description	Remarks
gt_refclk1_p gt_refclk1_n	Input	Differential Transceiver Reference Clock 1	Enabled when **Shared Logic in Core** is selected.
gt_refclk2_p gt_refclk2_n	Input	Differential Transceiver Reference Clock 2	Enabled when **Shared Logic in Core** is selected and more than one reference clock is required.
refclk1_in	Input	Single Ended Transceiver Reference Clock 1	Enabled when **Shared Logic in Example Design** is selected.
refclk2_in	Input	Single Ended Transceiver Reference Clock 2	Enabled when **Shared Logic in Example Design** is selected and more than one reference clock is required.
user_clk_out	Output	User Clock output	Enabled when **Shared Logic in Core** is selected
init_clk_out	output	INIT Clock output	Enabled when **Shared Logic in Core** is selected. Available only for 7 series devices.
sync_clk	Input	Sync clock input from the support logic	Enabled when **Shared Logic in Example Design** is selected
sync_clk_out	Output	Sync clock output to be used by the support logic	Enabled when **Shared Logic in Core** is selected.
reset_pb	Input	Push Button Reset, the top level `reset` input at the Example Design Level, This is required in the core as the Support Reset logic is now inside the core	
gt_reset_out	Output	Output of de-bouncer for gt_reset	Enabled when **Shared Logic in Core** is selected.
gt_refclk1_out	Output	Single Ended Transceiver Reference clock	Enabled when **Shared Logic in Core** is selected.
gt_refclk2_out	Output	Single Ended Transceiver Reference clock	Enabled when **Shared Logic in Core** is selected.
mmcm_not_locked_out	Output	The `mmcm_not_locked` signal from the clock module.	

Table 3-1: **Port Changes Due to Shared Logic Option *(Cont'd)***

NAME	Direction	Description	Remarks
gt_rxcdrovrden_in	Input	RXCDR Override used to configure GT in loopback mode	
gt_qpllclk_quad<quad>_in gt_qpllrefclk_quad<quad>_in	Input	Clock inputs generated by GTXE2_COMMON/ GTHE2_COMMON/ GTHE3_COMMON	<quad> refers to the active transceiver quad and starts from 1 to 12. Enabled when **Shared Logic in Example Design** is selected. Applicable for GTX or GTH transceiver designs. These ports are enabled for each quad that you select in the Vivado IDE during core configuration in the Vivado Design Suite.
gt_qpllclk_quad<quad>_out gt_qpllrefclk_quad<quad>_out	Output	Clock outputs generated by GTXE2_COMMON/ GTHE2_COMMON/ GTHE3_COMMON	<quad> refers to the active transceiver quad and starts from 1 to 12. Enabled when **Shared Logic in Core** is selected. Applicable for GTX or GTH transceiver designs. These ports are enabled for each quad that you select in the Vivado IDE during core configuration in the Vivado Design Suite.
gt_to_common_qpllreset_out	Output	QPLL common reset out to be used by the slave shared logic	Enabled when **Shared Logic in Example Design** is selected and when QPLL is being used.
gt_qplllock_quad<quad>_in gt_qpllrefclklost_quad<quad>_in	Input	QPLL lock and refclock lost signal inputs from the master shared logic	Enabled when **Shared Logic in Example Design** is selected and when QPLL is being used. <quad> refers to the active transceiver quad and starts from 1 to 12
gt_qplllock_quad<quad>_out gt_qpllrefclklost_quad<quad>_out	Output	QPLL lock and refclock lost signal outputs to the slave shared logic	Enabled when **Shared Logic in Core** is selected and when QPLL is being used. <quad> refers to the active transceiver quad and starts from 1 to 12,
init_clk_p init_clk_n	Input	Differential Free running system/board clock	Enabled when **Shared Logic in Core** is selected. Available only for 7 series devices.
sys_reset_out	Output	Output system reset to be used by the logic in the example design level	
init_clk	Input	Free running system/board clock	Available only for 7 series devices

Clocking

Good clocking is critical for the correct operation of the UltraScale™, Zynq®-7000, Virtex®-7, and Kintex®-7 device Aurora 64B/66B core. The core requires a low-jitter reference clock to drive the high-speed TX clock and clock recovery circuits in the GTX or GTH transceiver. It also requires at least one frequency-locked parallel clock for synchronous operation with the user application.

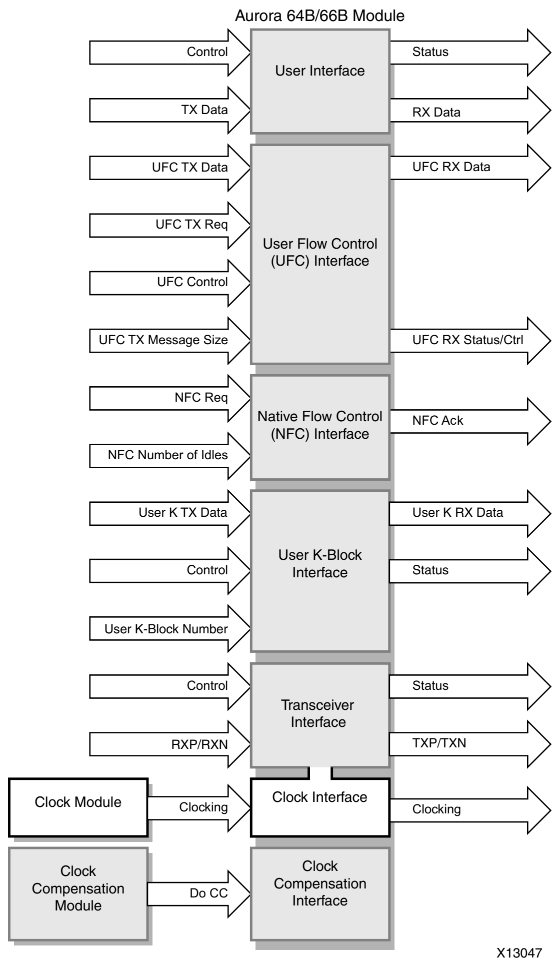


Figure 3-3: **Top-Level Clocking**

Each Aurora 64B/66B core is generated in the `example_project` directory that includes a design called `aurora_example`. This design instantiates the Aurora 64B/66B core that was generated and demonstrates a working clock configuration for the core. First-time users should examine the aurora example design and use it as a template when connecting the clock interface.

Clock Interface and Clocking

Aurora 64B/66B Clocking Architecture

Figure 3-4 shows the clocking architecture in the Aurora 64B/66B core for Zynq-7000, Virtex-7, and Kintex-7 device GTX or GTH transceivers.

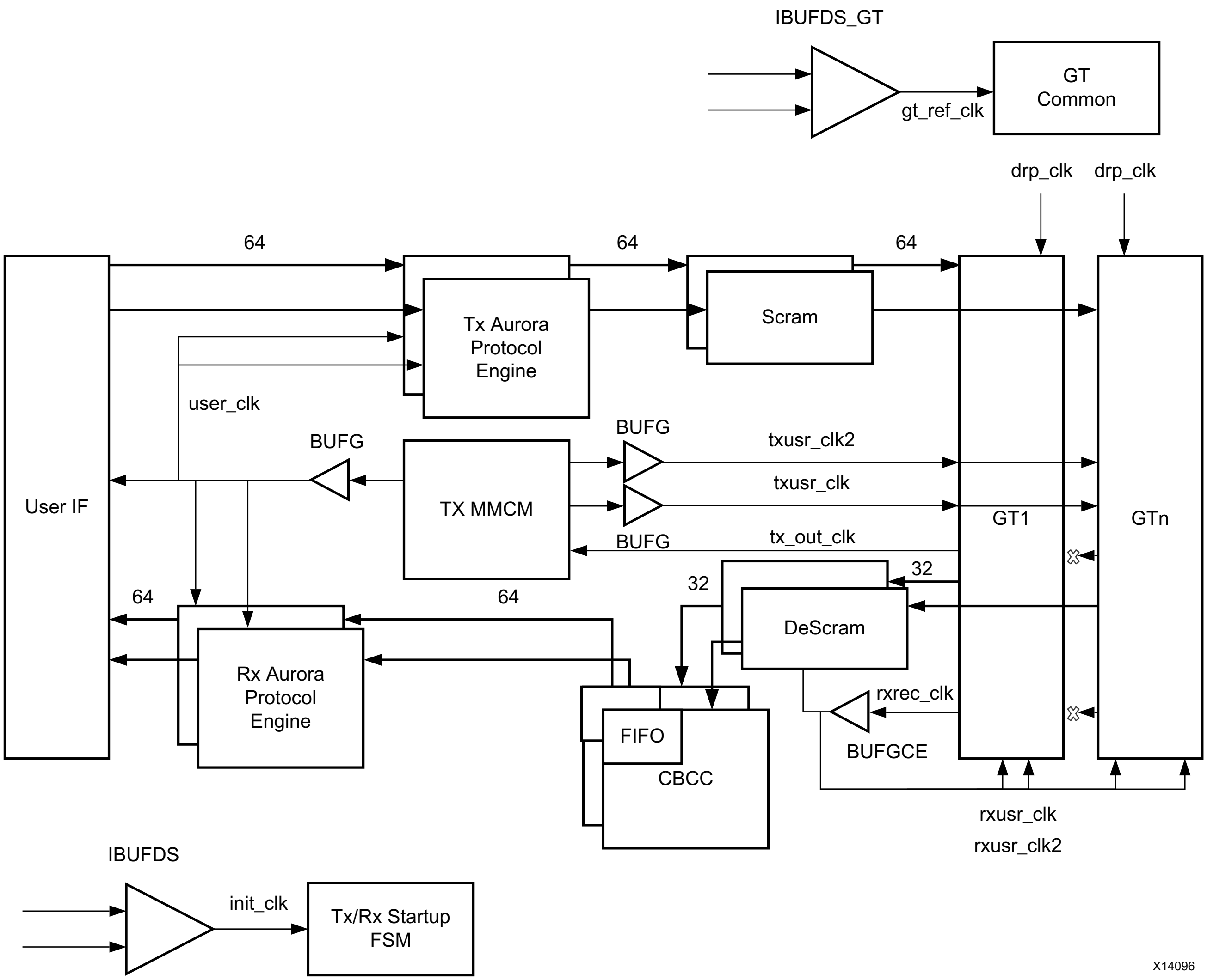


Figure 3-4: **Aurora 64B/66B Clocking for Zynq-7000, Virtex-7, and Kintex-7 Device GTX or GTH Transceivers**

Connecting user_clk, sync_clk, and tx_out_clk

The Aurora 64B/66B cores use three phase-locked parallel clocks. The first is `user_clk`, which synchronizes all signals between the core and the user application. All logic touching the core must be driven by `user_clk`, which in turn must be the output of a global clock buffer (BUFG).

The `user_clk` signal is used to drive the `txusrclk2` port of the serial transceiver. The `tx_out_clk` is selected such that the data rate of the parallel side of the module matches the data rate of the serial side of the module, taking into account 64B/66B encoding and decoding.

The third phase-locked parallel clock is `sync_clk`. This clock must also come from a BUFG and is used to drive `txusrclk` port of the serial transceiver. It is also connected to the Aurora 64B/66B core to drive the internal synchronization logic of the serial transceiver.

To make it easier to use the two parallel clocks, a clock module is provided in a subdirectory called `clock_module` under `example_design/support` or under `src` based on shared logic settings. The ports for this module are described in Table 2-16, page 37. If the clock module is used, the `mmcm_not_locked` signal should be connected to the `mmcm_not_locked` output of the clock module; `tx_out_clk` should connect to the clock module `clk` port, and `pll_lock` should connect to the clock module `pll_not_locked` port. If the clock module is not used, connect the `mmcm_not_locked` signal to the inverse of the `locked` signal from any PLL used to generate either of the parallel clocks, and use the `pll_lock` signal to hold the PLLs in reset during stabilization if `tx_out_clk` is used as the PLL source clock. The `txusrclk` could be unreliable during assertion of `pma_init`; hence, the core will use a stable clock (`init_clk`) for MMCM synchronization. Using a stable clock to sample adds more robustness to the link.

If MMCM is used to generate a stable clock (`init_clk`), `pma_init` needs to be applied to the Aurora core until MMCM lock is established. This ensures that the core remains in a known state before a stable clock is available for the core.

Usage of BUFG in the Aurora 64B/66B Core

The Aurora 64B/66B core uses four BUFGs for a given core configuration using GTX or GTH transceivers. Aurora 64B/66B is an eight-byte-aligned protocol, and the datapath from the user interface is 8-bytes aligned. For GTX or GTH transceivers, the core configures the transmit path as eight bytes and the receive path as four bytes.

The CB/CC logic is internal to the core, which is primarily based on the received recovered clock from the serial transceiver. The BUFG usage is constant for any core configuration and does not increase with any core feature.

Reference Clocks for FPGA Designs

Aurora 64B/66B cores require low-jitter reference clocks for generating and recovering high-speed serial clocks in the GTX and GTH transceivers. Each reference clock can be set to the reference clock input ports: `gtxq/gthq`. Reference clocks should be driven with high-quality clock sources whenever possible to decrease jitter and prevent bit errors. DCMs should never be used to drive reference clocks, because they introduce too much jitter.

For multi-lane designs in Zynq-7000, Virtex-7, and Kintex-7 devices, the Aurora 64B/66B wizard allows selecting clocks one Quad above and one Quad below the selected Quad per north-south clocking criteria. A second reference clock source can be selected if the quad selection exceeds the 3-Quad boundary. For details on north-south clocking, see the *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476) [Ref 4].

For UltraScale devices, the Xilinx implementation tools make necessary adjustments to the north-south routing and the pin swapping necessary to the GTHE3 transceiver clock inputs to route clocks from one quad to another, when required.

The maximum number of GTH transceivers that can be sourced by a single clock pin pair is 20.

IMPORTANT: *The following rules must be observed when sharing a reference clock to ensure that jitter margins for high-speed designs are met: The number of GTH transceiver quads above the sourcing quad must not exceed two. The number of GTX or GTH transceiver quads below the sourcing quad must not exceed two.*

Clock Compensation

The clock compensation feature allows up to ± 100 ppm difference in the reference clock frequencies used on each side of an Aurora channel. This feature is used in systems where a separate reference clock source is used for each device connected by the channel, and where the same `user_clk` is used for transmitting and receiving data.

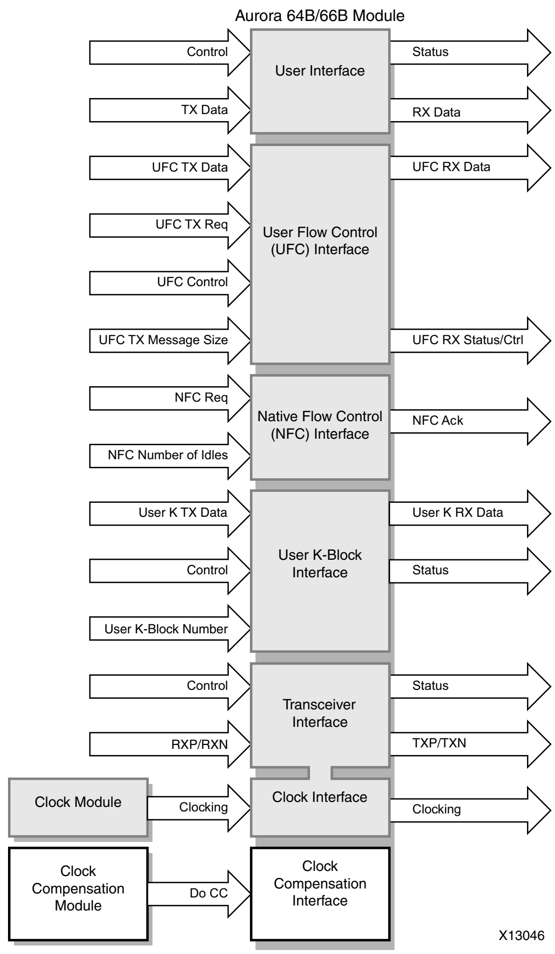


Figure 3-5: **Top-Level Clock Compensation Interface**

The Aurora 64B/66B core clock compensation interface enables full control over the core clock compensation features. A standard clock compensation module is generated with the Aurora 64B/66B core to provide Aurora-compliant clock compensation for systems using separate reference clock sources; users with special clock compensation requirements can drive the interface with custom logic. If the same reference clock source is used for both sides of the channel, the interface can be tied to ground to disable clock compensation.

Figure 3-6 and Figure 3-7 are waveform diagrams showing how the `do_cc` signal works.

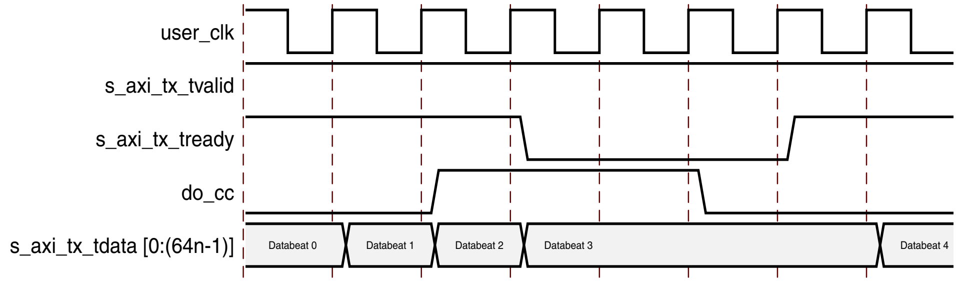


Figure 3-6: **Streaming Data with Clock Compensation Inserted**

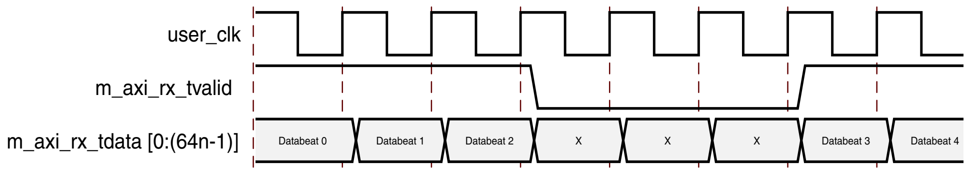


Figure 3-7: **Data Reception Interrupted by Clock Compensation**

The Aurora protocol specifies a clock compensation mechanism that allows up to ± 100 ppm difference between reference clocks on each side of an Aurora channel. To perform Aurora-compliant clock compensation, `do_cc` must be asserted for three `user_clk` cycles every 10,000 cycles. While `do_cc` is asserted, `s_axi_tx_tready` is deasserted on the TX user interface while the channel is being used to transmit clock compensation sequences.

A standard clock compensation module is generated along with each Aurora 64B/66B core from the Vivado® design tools, in the `cc_manager` subdirectory under `example_design`. It automatically generates pulses to create Aurora compliant clock compensation sequences on the `do_cc` port. This module should always be connected to the clock compensation port on the Aurora module, except in special cases. Table 3-2 shows the port description for the standard CC module.

Table 3-2: **Standard CC I/O Port**

Name	Direction	Description
do_cc	Output	Connect this port to the `do_cc` input of the Aurora 64B/66B core.
channel_up	Input	Connect this port to the `channel_up` output of a full-duplex core, or to the `tx_channel_up` output of a TX-only simplex port.

Clock compensation is not needed when both sides of the Aurora channel are being driven by the same clock (see Figure 3-7, page 80) because the reference clock frequencies on both sides of the module are locked. In this case, `do_cc` should be tied to ground.

Other special cases when the standard clock compensation module is not appropriate are possible. The do_cc port can be used to send clock compensation sequences at any time, for any duration to meet the needs of specific channels. The most common use of this feature is scheduling clock compensation events to occur outside of frames, or at specific times during a stream to avoid interrupting data flow.

IMPORTANT: *In general, customizing the clock compensation logic is not recommended, and when it is attempted, it should be performed with careful analysis, testing, and consideration of these guidelines:*

- Clock compensation sequences should last at least three user_clk cycles to ensure they are recognized by all receivers.
- Be sure the duration and period selected are sufficient to correct for the maximum difference between the frequencies of the clocks that will be used.
- Do not perform multiple clock compensation sequences within eight cycles of one another.
- Clock Compensation should not be disabled when hot-plug logic is enabled.

Core Features

This section describes the following features of the Aurora 64B/66B core.

- CRC
- Using Vivado Lab Tools
- Hot-Plug Logic
- Little Endian Support

CRC

A 32-bit CRC, implemented for framing user data interface, is available in the <component name>_crc_top.v module. The crc_valid and crc_pass_fail_n signals indicate the result of a received CRC with a transmitted CRC (see Table 3-3).

Table 3-3: **CRC Module Ports**

Port Name	Direction	Description
crc_valid	Output	Active-High signal that samples the crc_pass_fail_n signal.
crc_pass_fail_n	Output	The crc_pass_fail_n signal is asserted High when the received CRC matches the transmitted CRC. This signal is not asserted if the received CRC is not equal to the transmitted CRC. The crc_pass_fail_n signal should always be sampled with the crc_valid signal.

Using Vivado Lab Tools

The ILA and VIO cores aid in debugging and validating the design in the board and are provided with the Aurora 64B/66B core. The Aurora 64B/66B core connects the relevant signals to the VIO to facilitate easier bring-up or debug of the design. Select the Vivado lab tools option from the core Vivado Integrated Design Environment (IDE) (see Figure 4-1, page 86) to include it as a part of the example design.

Cores generated with Vivado lab tools enabled will have three VIO interfaces and one ILA interface.

- vio1_inst – contains core Lane Up, Channel Up, Data Error count, Soft Error count, Channel Up transition count along with System Reset, GT Reset and Loopback ports
- vio2_inst – contains status of reset quality counters
- vio3_inst – contains test pass/fail status for repeat reset test

Hot-Plug Logic

Hot-plug logic in Aurora 64B/66B designs is based on the received clock compensation characters. Reception of clock compensation characters at the RX interface of Aurora infers that the communication channel is active and not broken. If clock compensation characters are not received in a predetermined time, the hot-plug logic resets the core and the transceiver. The clock compensation module must be used for Aurora 64B/66B designs.

To disable hot-plug logic, set the ENABLE_HOTPLUG parameter to 0 in the `<component name>_cbcc_gtx_6466.v` module. With hot-plug logic disabled, the core does not get repeatedly reset when looking for clock compensation characters in duplex and any valid BTF characters for Simplex RX in the received data.

IMPORTANT: *It is highly recommended to keep hot plug logic enabled for predictable operation of the link.*

Following is the description of the hot-plug sequence.

1. Requirements: Before replacing the card or powering down a specific system or reprogramming the bit file, it is required to assert `reset` before doing hot plug so that the remote agent channel goes down gracefully and gets ready when you remove and plug in the link.
2. How it works: When `reset` is asserted at least for 128 cycles before doing hot plug, this will generate enough NA_IDLES for the remote link to deassert Channel Up without any errors.
3. Limitations: If the preceding sequence is not followed, it is possible that SOFT/DATA errors will be observed and the link will not have a graceful shutdown.

Little Endian Support

The Aurora 64B/66B IP core supports the user interfaces in big endian format by default. It also supports little endian format to enable it to connect to AXI4-Stream compliant IP designs seamlessly. Select the **Little Endian Support** in the Vivado IDE to select little endian format. It applies to the User Data, UFC, NFC and User K interfaces. Refer to the relevant interface for changes in ports.

Design Flow Steps

This chapter describes customizing and generating the core, constraining the core, and the simulation, synthesis and implementation steps that are specific to this IP core. More detailed information about the standard Vivado® design flows in the IP Integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 6]
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 7]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 8]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 9]

If you are customizing and generating the core in the Vivado IP Integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 6] for detailed information. The IP Integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values change, see the description of the parameter in this chapter. To view the parameter value, run the `validate_bd_design` command in the Tcl console.

Customizing and Generating the Core

This section includes information on using Vivado Design Suite to customize and generate the LogiCORE™ IP Aurora 64B/66B core.

Note: This core provides basic support for IP Integrator, but no parameter propagation is supported.

Vivado Integrated Design Environment

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

1. Select the IP from the IP catalog (**IP Catalog -> Communication & Networking -> Serial Interfaces ->Aurora 64B66B**).
2. Double-click the selected IP or select the Customize IP command from the toolbar or right-click menu.

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 7] and the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 8].

The Aurora 64B/66B core can be customized to suit a wide variety of requirements using the IP catalog. This chapter details the available customization parameters and how these parameters are specified within the IP catalog interface.

Using the IP Catalog

The Aurora 64B/66B IP catalog displays when you select the Aurora 64B/66B core in the Vivado IP catalog. Figure 4-1, page 86 and Figure 4-2, page 87 show features that are described in corresponding sections.

IP Catalog

Figure 4-1 and Figure 4-2 show the catalog. The left side displays a representative block diagram of the Aurora 64B/66B core as currently configured. The right side consists of user-configurable parameters. Details on the customizing options are provided in the following subsections, starting with Component Name, page 87.

Note: Figures in this chapter are illustrations of the Vivado IDE. This layout might vary from the current version.

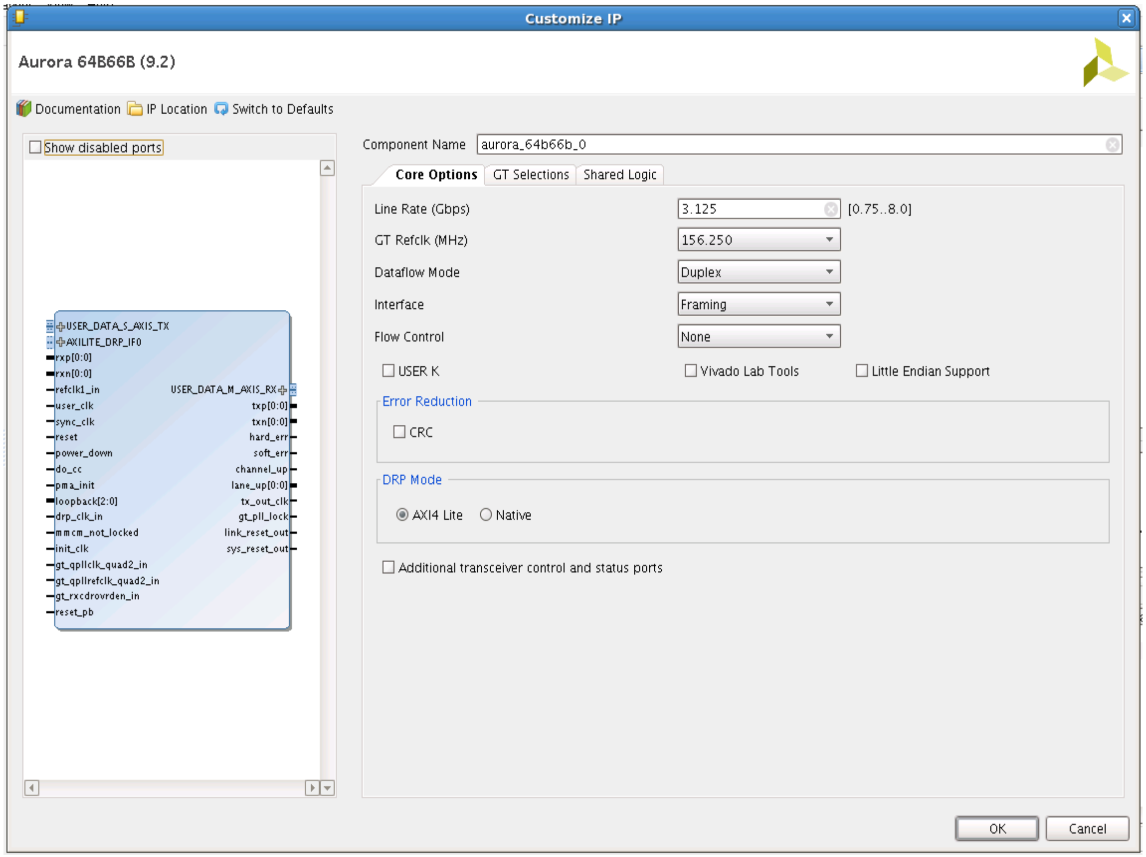


Figure 4-1: **Aurora 64B/66B IP Catalog Page 1 for 7 Series FPGAs**

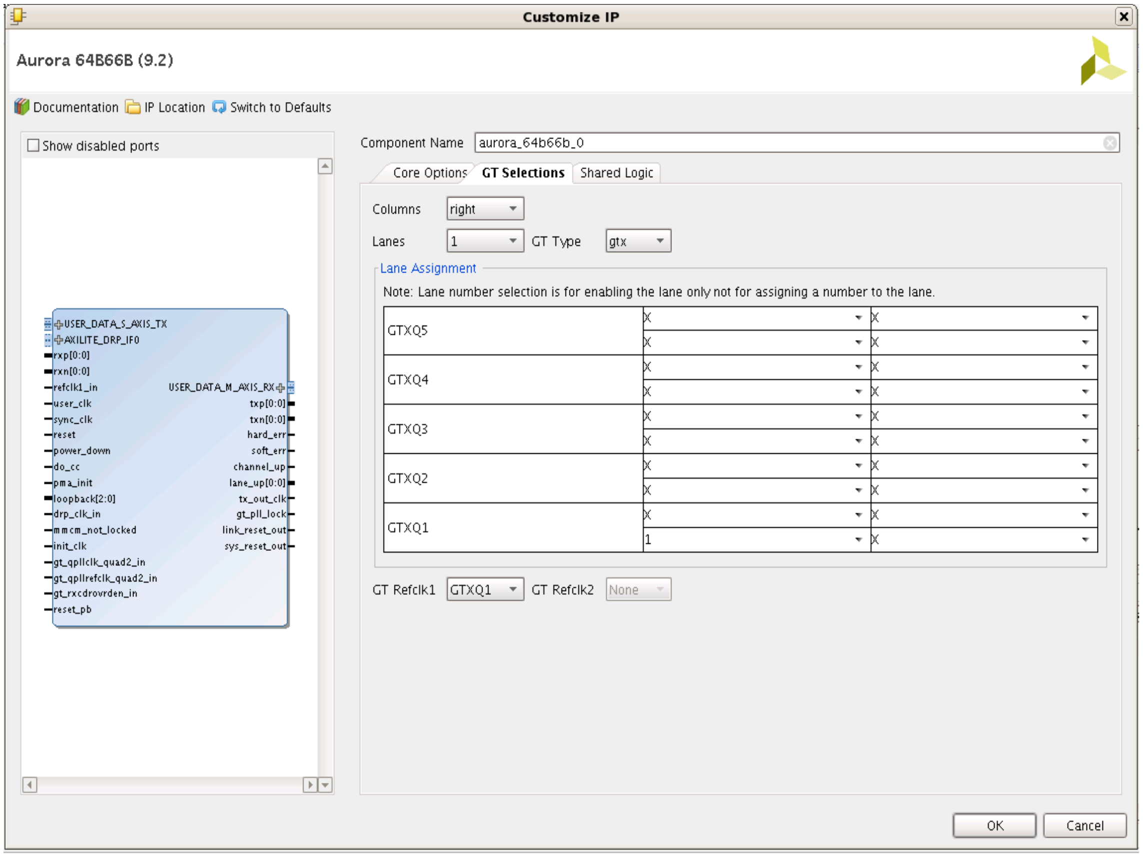


Figure 4-2: **Aurora 64B/66B IP Catalog Page 2 for 7 Series FPGAs**

Component Name

Enter the top-level name for the core in this text box. Illegal names are highlighted in red until they are corrected. All files for the generated core are placed in a subdirectory using this name. The top-level module for the core also use this name.

Default: aurora_64b66b_0

Line Rate

Enter a floating-point value in gigabits per second. The value entered must be within the valid range shown. This determines the unencoded bit rate at which data is transferred over the serial link.

Default: 3.125 Gb/s for GTX transceivers and Virtex®-7 FPGA GTH transceivers

GT Reference Clock Frequency

Select a reference clock frequency from the drop-down list. Reference clock frequencies are given in Megahertz, and depend on the line rate selected. For best results, select the highest rate that can be practically applied to the reference clock input of the target device.

Default: 156.25 MHz

Data Flow Mode

Select the options for the direction of the channel that the Aurora 64B/66B core supports. Simplex Aurora 64B/66B cores have a single, unidirectional serial port that connects to a complementary simplex Aurora 64B/66B core. Two options are provided as RX-only simplex or TX-only simplex. These options select the direction of the channel that the Aurora 64B/66B core supports.

Duplex – Aurora 64B/66B cores have both TX and the corresponding RX on the other side for communication.

Default: Duplex

Interface

Select the type of datapath interface used for the core. Select **Framing** to use a complete AXI4-Stream interface that allows encapsulation of data frames of any length. Select **Streaming** to use a simple word-based interface with a data valid signal to stream data through the Aurora channel.

Default: Framing

Flow Control

Select the required option to add flow control to the core. *User* flow control (UFC) allows applications to send each other brief, high-priority messages through the Aurora channel. *Native* flow control (NFC) allows full-duplex receivers to regulate the rate of the data sent to them. Immediate mode allows idle codes to be inserted within data frames while completion mode only inserts idle codes between complete data frames.

Available options are:

- None
- UFC only
- Immediate Mode – NFC
- Completion Mode – NFC
- UFC + Immediate Mode – NFC
- UFC + Completion Mode – NFC

For the streaming interface, only immediate mode is available. For the framing interface, both immediate and completion modes are available.

Default: None

User K

Select to add User K interface to the core. User K-blocks are special single-block codes passed directly to the user application. These blocks are used to implement application-specific control functions.

Default: Unchecked

CRC

Select the option to insert CRC32 in the data stream.

Default: Unchecked

Little Endian Support

Select to change all of the interface(s) to little endian format. See Little Endian Support in Chapter 3 for more information, By default the core uses Big Endian format.

Default: Unchecked

DRP

Select the required interface to control or monitor the transceiver interface using the Dynamic Reconfiguration Port (DRP).

Available options are:

- Native
- AXI4_Lite

Default: Native

Columns

Select appropriate GT column from the drop-down list.

Default: left

Lanes

Select the number of lanes (GTX and GTH transceivers) to be used in the core. The valid range depends on the target device selected.

Default: 1

GT_TYPE

Select the type of serial transceiver from the drop-down list. This option is applicable only for Virtex-7 XT devices. For other devices, the drop-down box is not visible.

Available options are:

- GTX
- V7GTH

Default: gtx

Lane Assignment

See the diagram in the information area in Figure 4-2. Each numbered row represents a serial transceiver tile and each active box represents an available GTX or GTH transceiver. For each Aurora lane in the core, starting with Lane 1, select a GTX or GTH transceiver and place the lane by selecting its number in the GTX or GTH placement box.

- "X" in the drop-down menu means that lane is not selected.
- "<1 - 16>" selected from the drop-down menu means that particular lane is selected. It does not assign that number to the physical lane.

RECOMMENDED: *Always select consecutive/physically adjacent lanes for a multi-GT design.*

Note: The Aurora core implements the transceiver placement in a predefined way. The core generates transceiver placement (LOC) constraints in ascending fashion. Move the cursor in the Vivado IDE to see the transceiver being selected in the 7 series and Zynq®-7000 family-based design. The manner in which numbers are entered in the lane selection will not change the transceiver LOC or core implementation in any way. The Lane Assignment is not available for UltraScale™ architecture-based designs. It is strongly recommended that lane selection should be continuous for timing closure.

GT REFCLK1 and GT REFCLK2

Select reference clock sources for the GTX and GTH transceiver tiles from the drop-down list in this section.

Default: GT REFCLK Source 1: GTXQn/ GTHQn; GT REFCLK Source 2: None;

Note: *n* depends on the serial transceiver (GTX or GTH) position.

Vivado Lab Tools

Select to add Vivado lab tools to the Aurora 64B/66B core. (See Using Vivado Lab Tools, page 82.) This option provides a debugging interface that shows the core status signals.

Default: Unchecked

Shared Logic

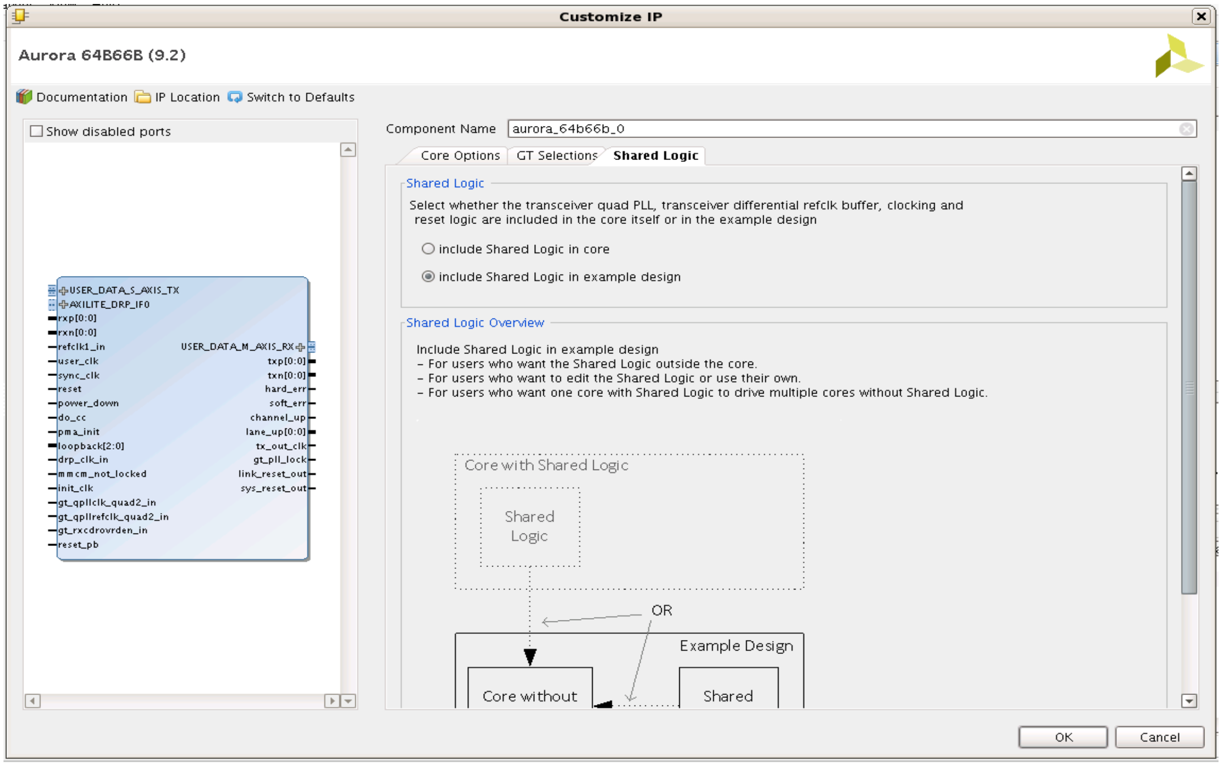


Figure 4-3: **Shared Logic for 7 Series FPGAs**

Select to include transceiver common PLL and its logic in the IP core or in the example design.

Available options:

- include shared logic in core
- include shared logic in example design

Default: include shared logic in example design

Additional Transceiver Control and Status Ports

Select to include transceiver control and status ports to core top level

Default: Unchecked

OK

Click **OK** to generate the core. (See Generating the Core, page 86.) The modules for the Aurora 64B/66B core are written to the IP catalog tool project directory using the same name as the top level of the core.

User Parameters

Table 4-1 (7 Series devices) and Table 4-2 (UltraScale™ architecture-based devices) show the relationship between the GUI fields in the Vivado IDE and the User Parameters in XCI files (which can be viewed in the Tcl console). Use the information in the tables for batch-driven Tcl flows to set GUI parameters and generate the Aurora 64B/66B core.

Table 4-1: **7 Series[1] GUI Parameter to User Parameter Mapping**

GUI Parameter/Value	User Parameter/Value	Default Value
Core Options		
Line Rate (Gbps)	C_LINE_RATE	3.125
GT Refclk (MHz)	C_REFCLK_FREQUENCY	156.250
Dataflow Mode	Dataflow_Config	Duplex
Interface	Interface_Mode	Framing
Flow Control	Flow_Mode	None
User K	C_USER_K	false
Vivado Lab Tools	C_USE_CHIPSCOPE	false
Little Endian Support	C_USE_BYTESWAP	false
Error Reduction		
CRC	CRC_MODE	NONE
DRP Mode		
AXI4 Lite (default mode)	drp_mode	AXI4_LITE
Native		
Additional transceiver control and status ports	TransceiverControl	false
GT Selections[2]		
Columns	C_COLUMN_USED	right[3]
Lanes	C_AURORA_LANES	1
GT Type	C_GT_TYPE	gtx[4]
Lane Assignment[5][6]		
Select transceiver to include GTXE2_CHANNEL_X1Y4 in your design [7]	C_GT_LOC_5 [8]	1
Select transceiver to include GTXE2_CHANNEL_X1Y5 in your design	C_GT_LOC_6	X
Select transceiver to include GTXE2_CHANNEL_X1Y5 in your design	C_GT_LOC_7	X
Select transceiver to include GTXE2_CHANNEL_X1Y7 in your design	C_GT_LOC_8	X
Select transceiver to include GTXE2_CHANNEL_X1Y8 in your design	C_GT_LOC_9	X

Table 4-1: **7 Series[1] GUI Parameter to User Parameter Mapping *(Cont'd)***

GUI Parameter/Value	User Parameter/Value	Default Value
Lane Assignment (cont'd)		
Select transceiver to include GTXE2_CHANNEL_X1Y9 in your design	C_GT_LOC_10	X
Select transceiver to include GTXE2_CHANNEL_X1Y10 in your design	C_GT_LOC_11	X
Select transceiver to include GTXE2_CHANNEL_X1Y11 in your design	C_GT_LOC_12	X
Select transceiver to include GTXE2_CHANNEL_X1Y12 in your design	C_GT_LOC_13	X
Select transceiver to include GTXE2_CHANNEL_X1Y13 in your design	C_GT_LOC_14	X
Select transceiver to include GTXE2_CHANNEL_X1Y14 in your design	C_GT_LOC_15	X
Select transceiver to include GTXE2_CHANNEL_X1Y15 in your design	C_GT_LOC_16	X
Select transceiver to include GTXE2_CHANNEL_X1Y16 in your design	C_GT_LOC_17	X
Select transceiver to include GTXE2_CHANNEL_X1Y17 in your design	C_GT_LOC_18	X
Select transceiver to include GTXE2_CHANNEL_X1Y18 in your design	C_GT_LOC_19	X
Select transceiver to include GTXE2_CHANNEL_X1Y19 in your design	C_GT_LOC_20	X
GT Refclk (MHz)		
GT Refclk1	C_GT_CLOCK_1	GTXQ1
GT Refclk2	C_GT_CLOCK_2	None
Shared Logic		
Include Shared Logic in core	SupportLevel[9]	0
Include Shared Logic in example design (default mode)		

Notes:
1. The values in this table reflect the default device (xc7vx485tffg1157-1).
2. X0Y0 GT selection is based on column.
3. If a device has GTs on both sides, left is the default value.
4. If the device has GTX transceivers, gtx is default value. If it has GTH transceivers, v7gth is the default value.
5. Lane number selection is for enabling the lane only and not for assigning numbers to the lane.
6. Lane selection is applicable only for 7 Series FPGAs and not for UltraScale devices.
7. In the default device, GT starts from GTXE2_CHANNEL_X1Y4. Otherewise, it starts from GTXE2_CHANNEL_X0Y0.
8. C_GT_LOC_i where, i varies from 1 to 48. By default, the lowest i C_GT_LOC_i is assigned.
9. If Shared Logic in Core option is selected, SupportLevel is 1.

Table 4-2: **UltraScale GUI Parameter to User Parameter Mapping**

GUI Parameter/Value	User Parameter/Value	Default Value
Core Options		
Physical Layer		
Line Rate (Gbps)	C_LINE_RATE	10.3125
Lanes	C_AURORA_LANES	1
GT Type	C_GT_TYPE	gth
GT Refclk (MHz)	C_REFCLK_FREQUENCY	156.250
Link Layer		
Dataflow Mode	Dataflow_Config	Duplex
Interface	Interface_Mode	Framing
Flow Control	Flow_Mode	None
User K	C_USER_K	false
CRC	CRC_MODE	NONE
Little Endian Support	C_USE_BYTESWAP	false
Debug and Control		
DRP Mode		
AXI4 Lite (default mode)	drp_mode	AXI4_LITE
Native		
Additional transceiver control and status ports	TransceiverControl	false
Vivado Lab Tools	C_USE_CHIPSCOPE	false
Shared Logic		
Include Shared Logic in core	SupportLevel[1]	0
Include Shared Logic in example design (default mode)		

Notes:
1. If Shared Logic in Core option is selected, SupportLevel is 1.

Core Customization Options for UltraScale Architecture Specific Designs

The section describes core customization options for UltraScale architecture-specific designs provided through the Vivado IDE.

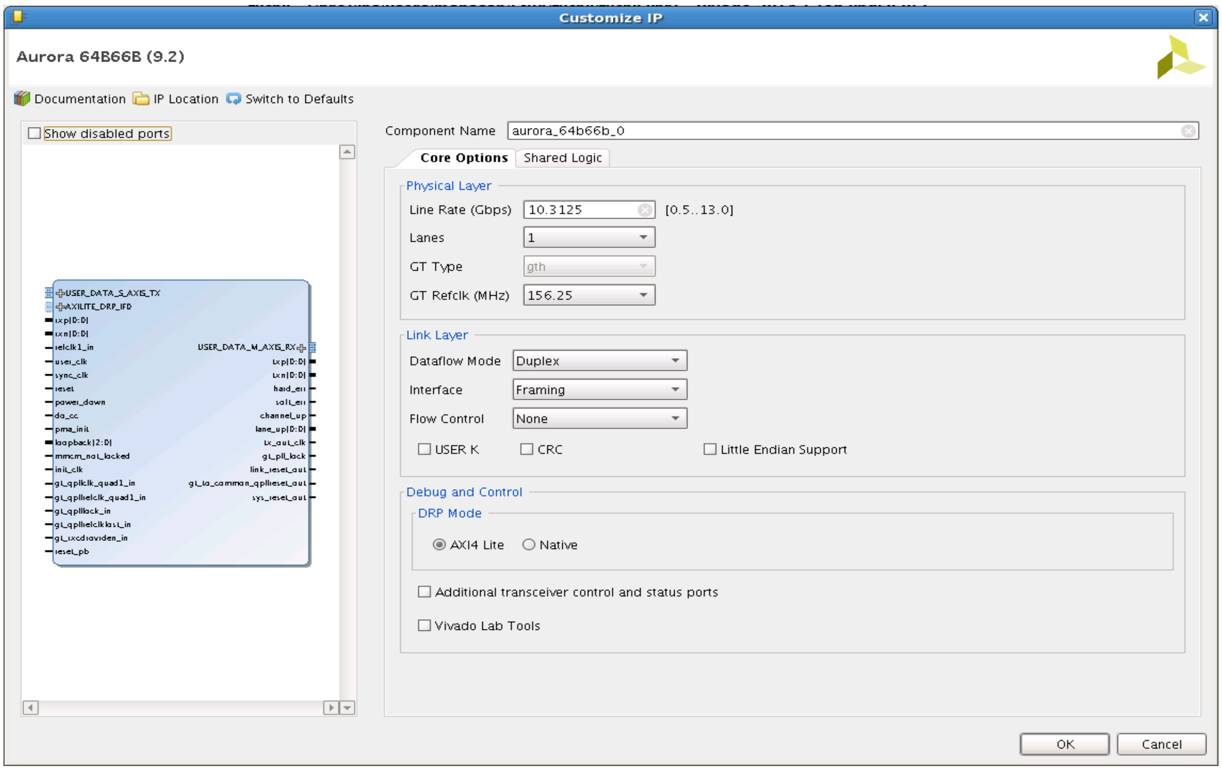


Figure 4-4: **Aurora 64B/66B IP Catalog Page 1 for UltraScale Devices**

Figure 4-4 shows the Vivado IDE of the Aurora 64B/66B core when targeted for UltraScale devices. In this mode, the GT configurations are set through the **Customize IP** window. The core supports line rate from 0.5 Gb/s to 13.0 Gb/s. The configurable parameters for the GT are **Line Rate**, **Lanes**, and **GT Refclk**. Based upon the line rate, the choice of the GT reference clock will be auto-updated. Based upon the line rate, the range of GT Refclk values will be made available for configuration. Based upon the user configuration, the parameter list is generated in the XCI file. This XCI file is used as a basis for further Aurora 64B/66B and GT Wizard configurations.

UltraScale Device GT Implementation

UltraScale device GT implementation support is through the dynamic configuration call, which is known as the hierarchical design methodology flow. See the *7 Series GTZ Transceiver User Guide* (UG478) [Ref 10]. The user configurations related to the GT are passed at the time of Aurora core configuration through the Vivado IDE. Refer to the *UltraScale FPGAs Transceivers Wizard Product Guide* (PG182) [Ref 11] for more information about the usage of the UltraScale device GT Wizard. In the Aurora 64B/66B core design, the UltraScale device GT Wizard is referred through sub core reference calls. With the updates in the UltraScale architecture, the GT Wizard own submodules, for example, reset controller, and data width sizing, are designed to reside in the GT Wizard itself, while the transmit and receive user clocking module helper cores are designed to always reside outside of the GT Wizard. The GT common location is based upon the lane configuration speed and the targeted UltraScale device. For speeds above 8.0 Gb/s, the GT common resides outside of the GT Wizard. The GT common will be part of the Aurora 64B/66B example design when the core is configured in the non-shared mode, while it will be part of the core when the core is configured in the shared mode through the Vivado IDE options. The Aurora 64B/66B core configures CPLL for line rates from 0.5 Gb/s to 8.0 Gb/s and QPLL1 for line rates from 8.1 Gb/s to 13.0 Gb/s.

UltraScale Device GT Channel Instance in the Core

The GT parameters like lane speed, number of lanes, reference clock, choice of CPLL/QPLL1 (based upon the choice of lane rate) and GT locations are automatically passed to the GT Wizard through the hierarchical IP flow. Based upon these parameters, the GT configuration is completed and the GT instance is generated in the Aurora 64B/66B core. As mentioned previously the GT Wizard contains the reset controller and user data-width sizing module in the GT Wizard.

UltraScale Device GT Clocking Structure in the Core

The main clocking module of the core generates the user clock, sync clock and initialization clock. The sync clock and user clock are the reference clocks to the GT channel interface as well as in the core logic. The core will always instantiate the transmitter user clocking module, while inclusion of the receive clocking module is based upon the core configuration. When the Aurora 64B/66B core is configured in shared mode, the clocking module becomes part of the core and its ports are available as output ports for sharing. In case of non-shared mode configuration, the clocking module will be part of the example design and the core will have these ports as input ports.

UltraScale Device GT Common Instance in the Core

The GT common from the UltraScale architecture GT Wizard is part of the shared logic of the Aurora 64B/66B core. This is applicable only when the lane speeds above 8.0 Gb/s are chosen. Based upon the number of lanes, the core will auto-insert the number of GT common quads. Each GT quad provides reference clocks up to four GT channels. The core provides the interface for clock, reset and lock signals to the GT common module. By default, the core provides the consecutive GT locations.

The clock module provides the reference clock to the GT common, while GT common provides the clock, reference clock, clock lock and reference clock lost signals for each GT channel located in each quad.

When the Aurora 64B/66B core is configured in shared mode, the GT common becomes part of the core and its ports are available as output ports. In the case of non-shared mode configuration (speed above 8 Gb/s), the GT common will be part of the Aurora 64B/66B example design and the core will have these ports as input ports.

When lane speeds below 8 Gb/s are chosen, the GT common resides in the GT Wizard IP core and only its ports are available at the core periphery in shared mode. In non-shared mode, these ports are internally contained within the core.

Note: For all the speeds less than 8 Gb/s, the GT common will not be part of actual shared or non-shared mode of the core.

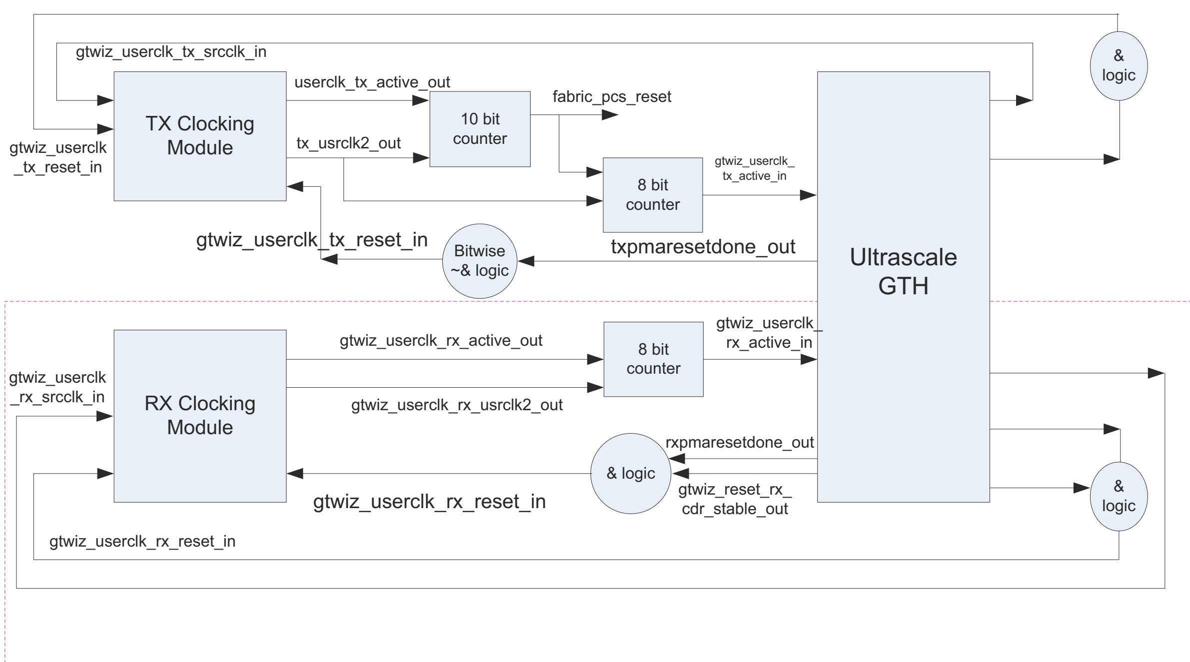


Figure 4-5: **Reset Sequence Logic implementation (Representation Purpose Only)**

The extended reset active signals ensure the faithful reset sequencing between the GT channel and GT common.

GT Channel Locations

In UltraScale architecture GT implementation for the Aurora 64B/66B core, it is expected that the GT locations are consecutive. Based upon the number of lanes selected and the targeted UltraScale device, the core provides consecutive GT channel locations which are by default set by the GT Wizard. In QPLL1 (line rate above 8.0 Gb/s) based designs, the GT common becomes part of the shared or non-shared logic choice for the core. The connection between the GT common and the GT channel is based upon number of lanes.

RECOMMENDED: *Do not alter any default locations, unless otherwise absolutely needed after the design is generated, else the design functionality cannot not be guaranteed. In the case when the line rate is less than 8.0 Gb/s is chosen, the CPLL becomes part of GT Wizard hierarchical core.*

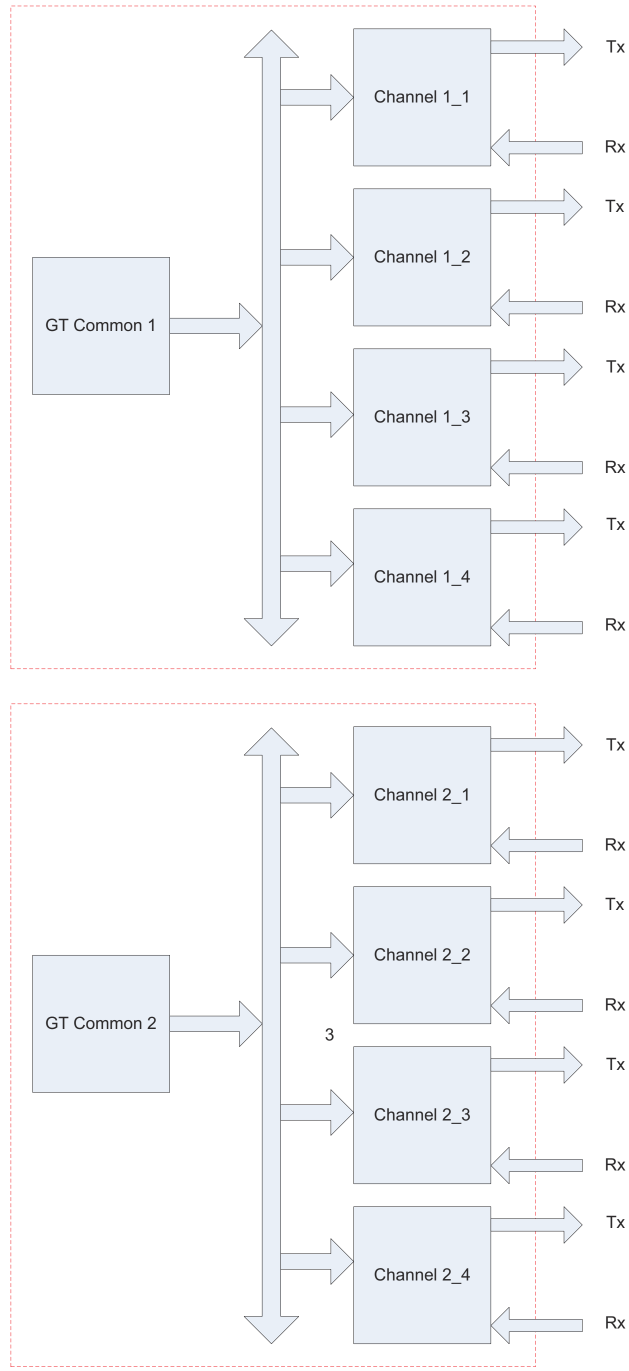


Figure 4-6: **Pictorial Representation of the UltraScale Architecture GT Common Interface with GT Channels (Aurora 64B/66B Configuration: 9 Gb/s and 8 lanes)**

How GT Locations are Mapped By the Aurora 64B/66B Core

Through the Aurora 64B/66B core Vivado IDE, you can configure the Line Rate, number of Lanes and Data Flow mode and so forth. Internally, these inputs are communicated back to the GT Wizard through the hierarchical IP calling mechanism. For a selected UltraScale device, for both the CPLL or QPLL1 based designs, the GT Wizard provides the correct information of the range of the reference clock and the default GT locations available for the device. Refer to Figure 4-6 to see how GT Channels are interfaced with each other. These default locations are available in the XDC provided by the UltraScale device GT Wizard instance. You can refer to these locations for any further updates. It is recommended not to alter these locations. However, based upon your design requirements, you can choose different channel locations. The selected GT channel location should be consecutive and assigned in such a way that it uses a minimum number of quads. For example, if you want to configure two Aurora designs each of three lanes, then it is required that these two Aurora designs be located in two different quads. Each quad group consists of one GT common and four GT channels and other logic. See the *UltraScale FPGAs GTH Transceivers User Guide* (UG576) [Ref 3] for detailed information about how the quad structure is organized. In the present implementation of the Aurora 64B/66B core for UltraScale devices, each GT common provides a clock, reference clock, clock lock and reference clock lost signal interface up to four GT channels located in the same quad. Based upon the number of channels needed, the core infers another quad and provides the correct interface to the GT channels allocated for that particular quad.

In the case of CPLL based implementation, where the line rate is chosen between 0.5 Gb/s to 8.0 Gb/s, the CPLL resides in the GT Wizard core instance and provides all the internal connections by default.

Output Generation

The customized Aurora 64B/66B core is delivered as a set of HDL source modules in Verilog. These files are arranged in a predetermined directory structure under the project directory name provided to the IP catalog when the project is created as shown in this section.

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 7].

Constraining the Core

This section contains information about constraining the core in the Vivado Design Suite.

Device, Package, and Speed Grade Selections

Not Applicable

Clock Frequencies

Aurora 64B/66B example design clock constraints can be grouped into the following three categories:

- GT reference clock constraint

  The Aurora 64B/66B core uses one minimum reference clock and two maximum reference clocks for the design. The number of GT reference clocks is derived based on transceiver selection (that is, lane assignment in the second page of the Vivado IDE). The GT REFCLK value selected in the first page of the Vivado IDE is used to constrain the GT reference clock. The `create_clock` XDC command is used to constrain GT reference clocks.

- CORECLK clock constraint

  CORECLKs are the clock based on which the core functions. CORECLKS such as USER_CLK and SYNC_CLK are derived out of TXOUTCLK generated by the GT transceiver based on the applied reference clock and the divider settings of the GT transceiver. The Aurora 64B/66B core calculates the USER_CLK/SYNC_CLK frequency based on the line rate and GT interface width. The `create_clock` XDC command is used to constrain all CORECLKs.

- INIT_CLK constraint

  The Aurora 64B/66B example design uses a debounce circuit to sample PMA_INIT asynchronously clocked by the `init_clk` clock. The `create_clock` XDC command is used to constrain the `init_clk` clock.

RECOMMENDED: *It is recommended to have the system clock frequency lower than the GT reference clock frequency and in the range of 50 to 200 MHz for 7 series and Zynq devices. For UltraScale devices, the recommended range is 6.25 MHz to line_rate/64 or 200 MHz whichever is less.*

Notes

- The default `init_clk` frequency for 7 series FPGAs set by the core is 50 MHz. Update this with respect to your system in the XDC file and STABLE_CLOCK_PERIOD in the `<user_component_name>_core.v` file.
- For CPLL-based UltraScale architecture-based designs, if the `init_clk` frequency is other than line_rate/64, update the C_FREERUN_FREQUENCY parameter with the exact frequency in the `<user_component_name>_gt/synth/<user_component_name>_gt.v` file inside the ip folder.

False Paths

The False Path constraint is defined on the first stage of the flip-flop of the CDC module.

Example Design

The generated example design with support logic in the example design is a 10.3125 Gb/s line rate and a 156.25 MHz reference clock. The XDC file generated for the XC7K325T-FFG900–2 device on the KC724 board follows:

```
<user_component_name>_exdes.xdc

####################### CLOCK CONSTRAINTS #####################
##User Clock Constraint: the value is selected based on the line rate of the module
  create_clock -name TS_user_clk_i -period 6.206 [get_pins
<user_component_name>_block_i/clock_module_i/user_clk_net_i/O]

##SYNC Clock Constraint
  create_clock -name TS_sync_clk_i -period 3.103 [get_pins
<user_component_name>_block_i/clock_module_i/sync_clock_net_i/O]

##Reference clock constraint for GTX
  create_clock -name GTXQ0_left_i -period 6.400 [get_ports GTXQ0_P]
  create_clock -name GTXQ0_left_i -period 6.400 [get_ports GTXQ0_N]

##INIT_CLK board Clock Constraint
create_clock -name TS_INIT_CLK -period 20 [get_ports INIT_CLK_P]
create_clock -name TS_INIT_CLK -period 20 [get_ports INIT_CLK_N]

##False path constraint to the first D input pin of the synchronizer stages
set_false_path -to [get_pins -hier *<user_component_name>_cdc_to*/D]

##PIN LOCATION CONSTRAINTS
set_property LOC C25 [get_ports INIT_CLK_P]
set_property LOC B25 [get_ports INIT_CLK_N]
set_property LOC G19 [get_ports RESET]
set_property LOC K18 [get_ports PMA_INIT]
set_property LOC A20 [get_ports CHANNEL_UP]
set_property LOC A17 [get_ports LANE_UP]

################################ GT CLOCK Locations   ##############
##Differential SMA Clock Connection
set_property LOC R8 [get_ports GTXQ0_P]
set_property LOC R7 [get_ports GTXQ0_N]

set_property LOC GTXE2_CHANNEL_X0Y0 [get_cells
<user_component_name>_block_i/<user_component_name>_i/inst/<user_component_name>_wr
apper_i/<user_component_name>_multi_gt_i/<user_component_name>_GTX_INST/gtxe2_i]
```

The preceding example XDC is for reference only. This XDC is created automatically when the core is generated from the Vivado design tools.

Clock Management

Not Applicable

Clock Placement

Not Applicable

Banking

Not Applicable

Transceiver Placement

The `set_property` XDC command is used to constrain the GT transceiver location. This is provided as a tooltip on the second page of the Vivado IDE. A sample XDC is provided for reference.

I/O Standard and Placement

The positive differential clock input pin (ends with _P) and negative differential clock input pin (ends with _N) are used as the GT reference clock. The `set_property` XDC command is used to constrain the GT reference clock pins.

Simulation

This section contains information about simulating in the Vivado Design Suite. For details, see the *Vivado Design Suite User Guide - Logic Simulation* (UG900) [Ref 9].

Aurora IP core delivers the demonstration test bench for the example design. Simulation status is reported through messages. The TEST COMPLETED SUCCESSFULLY message signifies the completion of the example design simulation.

Note: The `Reached max. simulation time limit` message means that simulation was not successful See Appendix C, Debugging for more information.

Simulating the Duplex core is a single-step process after generating the example design. Simplex core simulation requires partner generation. The partner core is generated automatically and the synthesized netlist is available under the simulation file set when clicking **Open IP Example Design**. Due to the synthesizing of the partner core, opening an example design of a Simplex core takes more time than the Duplex example design generation.

Simulation speed up:

The C_EXAMPLE_SIMULATION parameter has been introduced to speed up post synthesis/implementation netlist functional simulations.

1. If core generation is through batch mode, include this command, `set c_example_simulation true` as part of the core generation.
2. Run the Tcl command to speed up simulation. The generated core with the preceding command is *only* for simulation.
3. If core generation is through the Vivado IDE, change the EXAMPLE_SIMULATION parameter to 1 in the generated RTL in the following files `<USER_COMPONENT_NAME>_exdes.v` and `<USER_COMPONENT_NAME>_core.v` to speed up simulation.

Synthesis and Implementation

This section contains information about synthesis and implementation in the Vivado Design Suite.

For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 7].

Implementation

The quick start example consists of the following components:

Overview

- An instance of the Aurora 64B/66B core generated using the default parameters
    - Full-duplex with a single GTX or GTH transceiver
    - AXI4-Stream interface
- A demonstration test bench to simulate two instances of the example design

The Aurora 64B/66B example design has been tested with the Vivado Design Suite for synthesis and Mentor Graphics Questa® SIM for simulation.

Generating the Core

To generate an Aurora 64B/66B core with default values using the Vivado design tools:

1. Start the Vivado design tools from a required directory. For help starting and using the Vivado design tools, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 7].
2. Choose **Create New Project New > Project > Next.**
3. Type the new project name and enter the project location.
4. Select **Project Type** as RTL Project and click **Next**.
5. Select the part as xc7vx485tffg1157–1.
6. After creating the project, click **IP catalog** in the **Project Manager** panel.
7. Locate the Aurora 64B/66B v9.2 core in the IP catalog taxonomy tree under: / Communication_&_Networking/Serial_Interfaces.
8. Double-click the core.
9. Click **OK**.

Implementing the Example Design

The example design needs to be generated from the IP core.

1. Right-click the generated IP. Click **Open Example Design** on the menu displayed for the right-click operation. This action opens an example design for the generated IP core.
2. Click **Run Implementation** to run the synthesis followed by implementation. Additionally you can also generate a bitstream by clicking **Generate Bitstream**.

Note: You need to specify LOC and IO standards in XDC for all input and output ports of the design.

For details about synthesis and implementation, see “Synthesizing IP” and “Implementing IP” in the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 7].

Detailed Example Design

This chapter contains information about the example design provided in the Vivado® Design Suite.

Directory and File Contents

See Output Generation, page 100 for the directory structure and file contents of the example design.

Quick Start Example Design

The quick start instructions provide a step-by-step procedure for generating an Aurora 64B/66B core, implementing the core in hardware using the accompanying example design, and simulating the core with the provided demonstration test bench (`demo_tb`). For detailed information about the example design provided with the Aurora 64B/66B core, see Detailed Example Design.

The quick start example design consists of these components:

- An instance of the Aurora 64B/66B core generated using the default parameters
  - Full-duplex with a single GTX transceiver
  - AXI4-Stream user interface
- A top-level example design (`<component name>_exdes`) with an XDC file to configure the core for simple data transfer operation
- A demonstration test bench to simulate two instances of the example design

Detailed Example Design

Each Aurora 64B/66B core includes an example design (`<component name>_exdes`) that uses the core in a simple data transfer system. For more details about the `example_design` directory, see Output Generation in Chapter 4.

The example design consists of two main components:

- Frame generator (FRAME_GEN) connected to the TX interface
- Frame checked (FRAME_CHECK) connected to the RX user interface

Figure 5-1 shows a block diagram of the example design for a full-duplex core. Table 5-1, page 108 describes the ports of the example design.

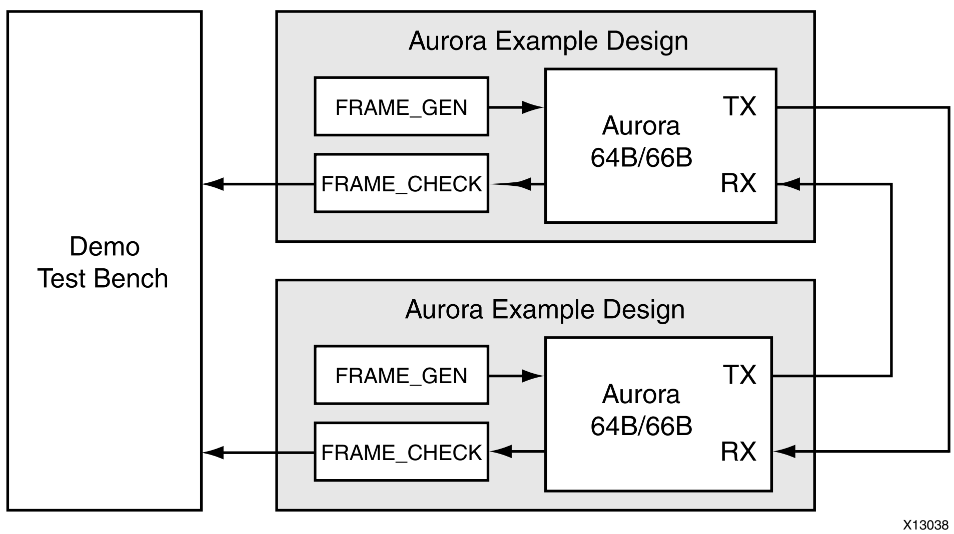


Figure 5-1: **Example Design**

The example design uses all the interfaces of the core. There are separate AXI4-Stream interfaces for optional flow control. Simplex cores without a TX or RX interface have no FRAME_GEN or FRAME_CHECK block, respectively. The frame generator produces a random stream of data for cores with a streaming/framing interface.

The design can also be used as a reference for connecting the trickier interfaces on the Aurora 64B/66B core, such as the clocking interface.

When using the example design on a board, be sure to edit the `<component name>_exdes` file in the `example_design` subdirectory to supply the correct pins and clock constraints. Table 5-1 describes the ports available in the example design.

Table 5-1: **Example Design I/O Ports**

Port	Direction	Description
rxn[0:*m*–1]	Input	Negative differential serial data input pin.
rxp[0:*m*–1]	Input	Positive differential serial data input pin.
txn[0:*m*–1]	Output	Negative differential serial data output pin.
txp[0:*m*–1]	Output	Positive differential serial data output pin.
reset	Input	Reset signal for the example design. The active-High reset is debounced using a `user_clk` signal generated from the reference clock input.
<reference clock(s)>	Input	The reference clocks for the Aurora 64B/66B core are brought to the top level of the example design. See Clock Interface and Clocking in Chapter 3 for details about the reference clocks.
<core error signals>	Output	The error signals from the Aurora 64B/66B core Status and Control interface are brought to the top level of the example design and registered. See Status, Control, and the Transceiver Interface in Chapter 2 for details.
<core channel up signals>	Output	The channel up status signals for the core are brought to the top level of the example design and registered. See Status, Control, and the Transceiver Interface in Chapter 2 for details.
<core lane up signals>	Output	The lane up status signals for the core are brought to the top level of the example design and registered. Cores have a lane up signal for each GTX and GTH transceiver they use. See Status, Control, and the Transceiver Interface in Chapter 2 for details.
pma_init	Input	The reset signal for the PCS and PMA modules in the GTX and GTH transceivers is connected to the top level through a debouncer. The signal is debounced using the `init_clk`. See the Reset section in the *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476) [Ref 4] for further details on GT RESET.
init_clk_p/ init_clk_n	Input	The `init_clk` signal is used to register and debounce the PMA_INIT signal. The `init_clk` signal must not come from a GTX or GTH transceiver, and should be set to a slow rate, preferably slower than the reference clock. The `init_clk` signal is single-ended for UltraScale™ devices.
data_err_count[0:7]	Output	Count of the number of frame data words received by the FRAME_CHECK that did not match the expected value.
ufc_err	Output	Asserted (active-High) when UFC data words received by the FRAME_CHECK that did not match the expected value.
user_k_err	Output	Asserted (active-High) when User K data words received by the FRAME_CHECK that did not match the expected value.

FRAME_GEN

Framing TX Data Interface

To transmit the user data, the FRAME_GEN user data state machine manipulates control signals to do the following:

- After the Aurora interface is out of RESET and reaches CHANNEL_UP state, pseudo-random data is generated using the user data linear feedback shift register (LFSR) and connected to `s_axi_tx_tdata` bus.
- Generates the `s_axi_tx_tlast` for the current frame based on two counters. An 8-bit counter is used to determine the size of the frame and another 8-bit counter to keep track of number of user data bytes sent. Frame size counter is initialized and incremented by one for every frame.
- The `s_axi_tx_tkeep` bus is connected to lower bits of user data LFSR to generate SEP and SEP7 conditions.
- The `s_axi_tx_tvalid` signal is asserted according to AXI4-Stream protocol specification.
- User data state machine state transitions are controlled by `s_axi_tx_tready` provided by the Aurora AXI4-Stream interface.
- Various kinds of frame traffic are generated including single cycle frame.

Figure 5-2 shows the FRAME_GEN framing user interface of the Aurora 64B/66B core, with AXI4-Stream compliant ports for TX data.

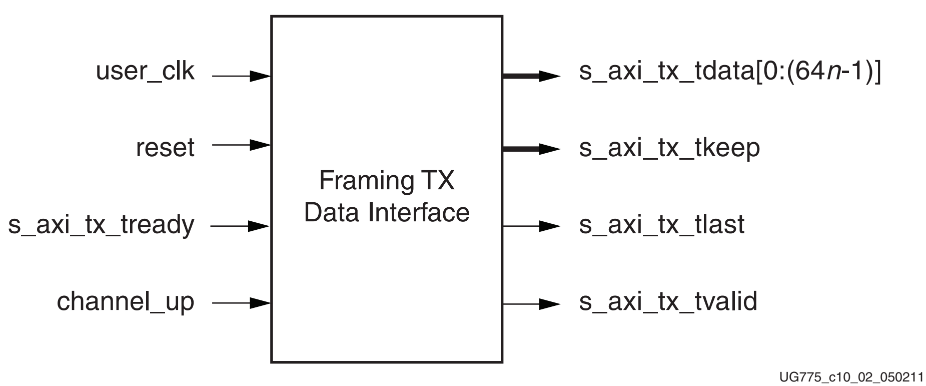


Figure 5-2: **Aurora 64B/66B Core Framing TX Data Interface (FRAME_GEN)**

Table 5-2 lists the FRAME_GEN framing TX data ports and their descriptions.

Table 5-2: **FRAME_GEN Framing User I/O Ports (TX)**

Name	Direction	Description
s_axi_tx_tdata[0:(64*n*–1)]	Output	User frame data. Width is 64**n* where *n* is the number of lanes.
s_axi_tx_tkeep[0:*n*–1)]	Output	Specifies the number of valid bytes in the last data beat; Valid only while `s_axi_tx_tlast` is asserted High.
s_axi_tx_tvalid	Output	Asserted (active-High) when AXI4-Stream signals from the source are valid. Deasserted (Low) when AXI4-Stream control signals and/or data from the source should be ignored.
s_axi_tx_tlast	Output	Signals the end of the frame data (active-High).
s_axi_tx_tready	Input	Asserted (active-High) during clock edges when signals from the source are accepted (if `s_axi_tx_tvalid` is also asserted). Deasserted (Low) on clock edges when signals from the source are ignored.
channel_up	Input	Asserted when Aurora channel initialization is complete and channel is ready to send data.
user_clk	Input	Parallel clock shared by the Aurora 64B/66B core and the user application.
reset	Input	Resets the Aurora core (active-High).

Streaming TX Data Interface

Streaming TX data interface is similar to framing TX data interface without framing delimiters, `s_axi_tx_tlast`, and `s_axi_tx_tkeep`. To transmit the user data, the FRAME_GEN user data state machine manipulates control signals to do the following:

- After the Aurora interface is out of RESET and reaches CHANNEL_UP state, pseudo-random data is generated using LFSR and connected to `s_axi_tx_tdata` bus.
- LFSR generates new data for every assertion of `s_axi_tx_tready`.
- The `s_axi_tx_tvalid` signal is always asserted.

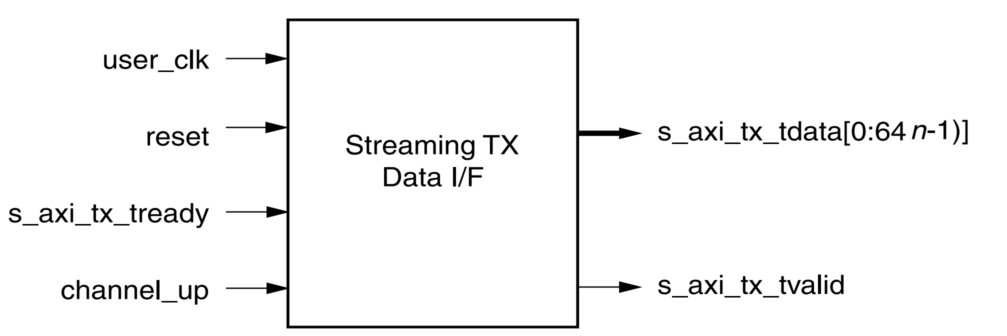


Figure 5-3: **Aurora 64B/66B Core Streaming TX Data Interface (FRAME_GEN)**

Table 5-3 lists the FRAME_GEN streaming TX data ports and their descriptions.

Table 5-3: **FRAME_GEN Streaming User I/O Ports (TX)**

Name	Direction	Description
s_axi_tx_tdata[0:(64*n*–1)]	Output	Outgoing frame data. Width is 64**n* where *n* is the number of lanes.
s_axi_tx_tvalid	Output	Asserted (active-High) when AXI4-Stream signals from the source are valid. Deasserted (Low) when AXI4-Stream control signals and/or data from the source should be ignored.
s_axi_tx_tready	Input	Asserted (active-High) during clock edges when signals from the source are accepted (if `s_axi_tx_tvalid` is also asserted). Deasserted (Low) on clock edges when signals from the source are ignored.
channel_up	Input	Asserted when Aurora channel initialization is complete and channel is ready to send data.
user_clk	Input	Parallel clock shared by the Aurora 64B/66B core and the user application.
reset	Input	Resets the Aurora core (active-High).

UFC TX Interface

To transmit the UFC data, the FRAME_GEN UFC state machine manipulates control signals to do the following:

- Asserts `ufc_tx_req` after CHANNEL_UP indication from the Aurora TX interface.
- `ufc_tx_ms` is also transmitted along with `ufc_tx_req`. The `ufc_tx_ms` signal transmits zero initially for the first UFC frame and is incremented by one for the following UFC frames until it reaches 255 (maximum value).
- The `s_axi_ufc_tx_tvalid` signal is asserted after placing the `ufc_tx_req`.
- The `s_axi_ufc_tx_tdata` signal is transmitted after receiving `s_axi_ufc_tx_tready` from the Aurora TX interface.
- UFC frame transmission frequency is controlled by the UFC_IFG parameter

Figure 5-4 shows the FRAME_GEN UFC TX interface of the Aurora 64B/66B core, with AXI4-Stream compliant ports for UFC TX data.

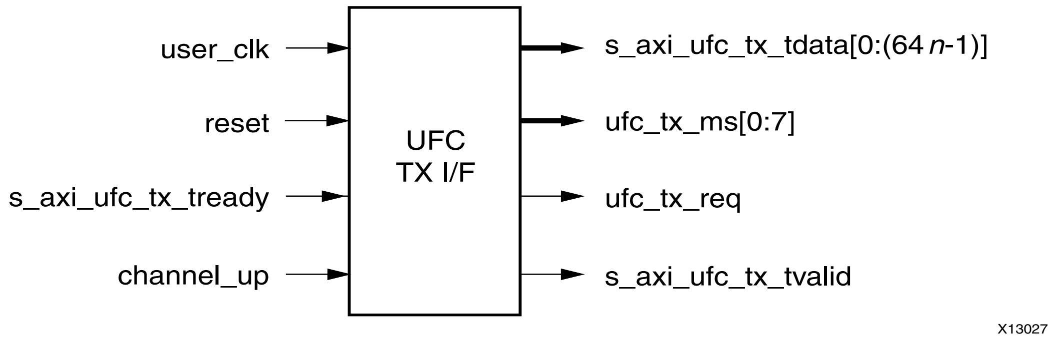


Figure 5-4: **Aurora 64B/66B Core UFC TX Interface (FRAME_GEN)**

Table 5-4 lists the FRAME_GEN UFC TX data ports and their descriptions.

Table 5-4: **FRAME_GEN UFC User I/O Ports (TX)**

Name	Direction	Description
ufc_tx_req	Output	Asserted to request a UFC message to be sent to the channel partner (active-High). Requests are processed after a single cycle, unless another UFC message is in progress and not on its last cycle. After a request, the `s_axi_ufc_tx_tdata` bus is ready to send data within two cycles unless interrupted by a higher priority event.
ufc_tx_ms[0:7]	Output	Specifies the number of bytes in the UFC message (the Message Size). The max UFC Message Size is 256. The value specified at `ufc_tx_ms` is one less than the actual amount of bytes transferred. For example, a value of 3 will transmit 4 bytes of data.
s_axi_ufc_tx_tdata [0:(64*n*–1)]	Output	Output bus for UFC message data to the Aurora channel. Data is read from the bus into the channel only when both `s_axi_ufc_tx_tvalid` and `s_axi_ufc_tx_tready` are asserted on a positive `user_clk` edge. If the number of bytes in the message is not an integer multiple of the bytes in the bus, on the last cycle, only the bytes needed to finish the message starting from the left of the bus are used.
s_axi_ufc_tx_tvalid	Output	Assert (active-High) when data on `s_axi_ufc_tx_tdata` is valid. If deasserted while `s_axi_ufc_tx_tready` is asserted, Idle blocks are inserted in the UFC message.
s_axi_ufc_tx_tready	Input	Asserted (active-high) when an aurora 64B/66B core is ready to read data from the `s_axi_ufc_tx_tdata` interface. this signal is asserted one clock cycle after `ufc_tx_req` is asserted and no high priority requests in progress. `s_axi_ufc_tx_tready` continues to be asserted while the core waits for data for the most recently requested ufc message. the signal is deasserted for cc and nfc requests, which are higher priority. while `s_axi_ufc_tx_tready` is asserted, `s_axi_tx_tready` is deasserted.
channel_up	Input	Asserted when Aurora channel initialization is complete and channel is ready to send data.

Table 5-4: **FRAME_GEN UFC User I/O Ports (TX) *(Cont'd)***

Name	Direction	Description
user_clk	Input	Parallel clock shared by the Aurora 64B/66B core and the user application.
reset	Input	Resets the Aurora core (active-High).

NFC TX Interface

To transmit the NFC frame, the FRAME_GEN NFC state machine manipulates control signals to do the following:

- NFC state machine waits until TX user data transmission and enters into NFC XON mode.
- The `s_axi_nfc_tx_tdata` value is transmitted along with `s_axi_nfc_tx_tvalid`.
- After predefined period of time, NFC state machine enters into NFC XOFF mode.
- NFC state transitions are governed by `s_axi_nfc_tx_tready`.
- NFC frame transmission frequency is controlled by NFC_IFG parameter.

Figure 5-5 shows the FRAME_GEN NFC TX interface of the Aurora 64B/66B core, with AXI4-Stream compliant ports for NFC TX data.

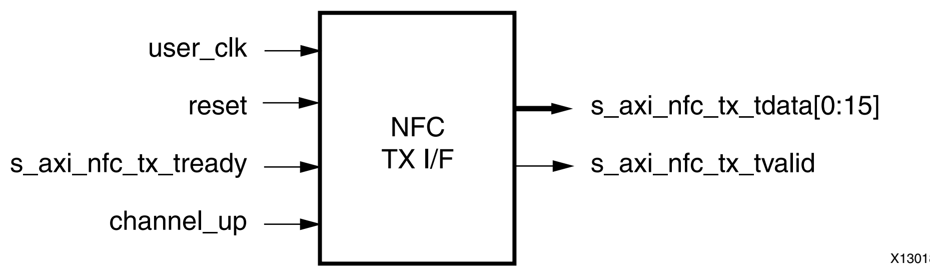


Figure 5-5: **Aurora 64B/66B Core NFC TX Interface (FRAME_GEN)**

Table 5-5 lists the FRAME_GEN NFC TX data ports and their descriptions.

Table 5-5: **FRAME_GEN NFC User I/O Ports (TX)**

Name	Direction	Description
s_axi_nfc_tx_tvalid	Output	Asserted to request an NFC message to be sent to the channel partner (active-High). Must be held until `s_axi_nfc_tx_tready` is asserted.
s_axi_nfc_tx_tdata [0:15]	Output	Indicates how many `user_clk` cycles the channel partner must wait before it can send data when it receives the NFC message. Must be held until `s_axi_nfc_tx_tready` is asserted. The number of `user_clk` cycles without data is equal to `s_axi_nfc_tx_tdata[8:15]` + 1. `s_axi_nfc_tx_tdata[7]` (active-High) is mapped to `nfc_xoff`, which requests the channel partner to stop sending data until it receives a non-XOFF NFC message or is reset. Signal Mapping: `s_axi_nfc_tx_tdata` = {7'h0, NFC XOFF bit, NFC Data}
s_axi_nfc_tx_tready	Input	Asserted when an Aurora core accepts an NFC request (active-High).
channel_up	Input	Asserted when Aurora channel initialization is complete and channel is ready to send data.
user_clk	Input	Parallel clock shared by the Aurora 64B/66B core and the user application.
reset	Input	Resets the Aurora core (active-High).

User K TX Interface

To transmit the User K data, FRAME_GEN manipulates control signals to do the following:

- The `s_axi_user_k_tx_tvalid` signal is asserted after User K inter-frame gap.
- Pre-defined User K data is transmitted along with User K Block No. User K Block No is set as zero for the first User K-block and is incremented by one for the following User K-blocks until it reaches 8.
- User K transmission frequency is controlled by USER_K_IFG parameter.

Figure 5-6 shows the FRAME_GEN User K TX interface of the Aurora 64B/66B core, with AXI4-Stream compliant ports for User K TX data.

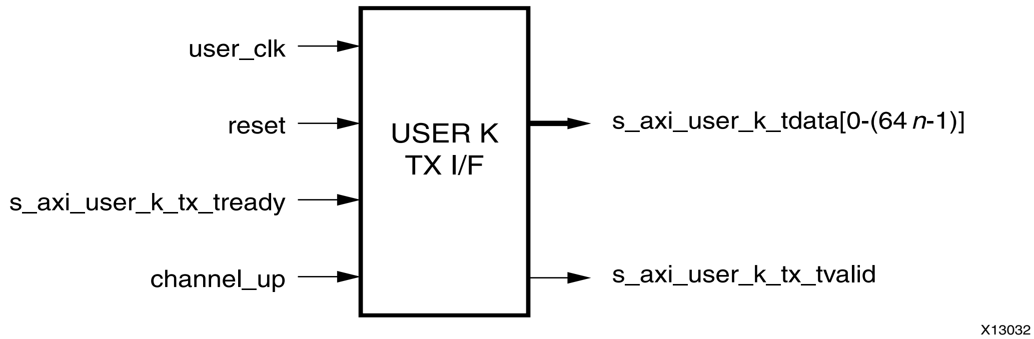


Figure 5-6: **Aurora 64B/66B Core User K TX Interface (FRAME_GEN)**

Table 5-6 lists the FRAME_GEN User K TX data ports and their descriptions.

Table 5-6: **FRAME_GEN User K User I/O Ports (TX)**

Name	Direction	Description
s_axi_user_k_tdata [0: (*n**64–1)]	Output	User K-block data. `s_axi_user_k_tx_tdata` = {4'h0, USER K BLOCK NO, USER K DATA[0:56n–1]}
s_axi_user_k_tx_tvalid	Output	Asserted (active-High) when User K data on `s_axi_user_k_tdata` port is valid.
s_axi_user_k_tx_tready	Input	Asserted (active-High) when the Aurora 64B/66B core is ready to read data from the `s_axi_user_k_tx_tdata` interface.
channel_up	Input	Asserted (active-High) when Aurora channel initialization is complete and channel is ready to send data.
user_clk	Input	Parallel clock shared by the Aurora 64B/66B core and the user application.
reset	Input	Resets the Aurora core (active-High).

FRAME_CHECK

Framing RX Data Interface

The expected frame RX data is computed by LFSR. The received user data is validated by checking against following AXI4-Stream protocol rules:

Start the frame when `m_axi_rx_tvalid` is asserted

1. The `m_axi_rx_tkeep` bus is valid during `m_axi_rx_tlast` assertion.
2. The `m_axi_rx_tvalid` signal should be asserted during comparison of expected to actual data:

Incoming RX data through `m_axi_rx_tdata` port is registered and compared with calculated RX data internal to FRAME_CHECK. If the incoming RX data does not match with expected RX data, an 8-bit counter is incremented. This error counter is indicated to the user application through `data_err_count` port. The Error counter freezes counting when it reaches 255.

Note: The counter can be cleared by applying reset.

Figure 5-7 shows the FRAME_CHECK framing user interface of the Aurora 64B/66B core, with AXI4-Stream compliant ports for RX data.

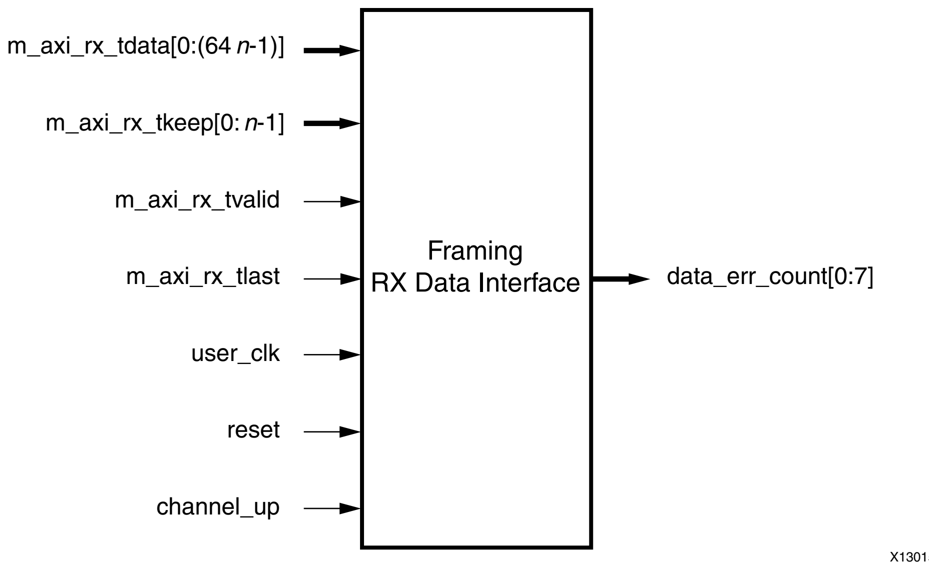


Figure 5-7: **Aurora 64B/66B Core Framing RX Data Interface (FRAME_CHECK)**

Table 5-7 lists the FRAME_CHECK framing RX data ports and their descriptions.

Table 5-7: **FRAME_CHECK Framing User I/O Ports (RX)**

Name	Direction	Description
m_axi_rx_tdata[0:(64*n*–1)]	Input	Incoming frame data from channel partner (Ascending bit order).
m_axi_rx_tkeep[0:*n*–1]	Input	Specifies the number of valid bytes in the last data beat. Valid only when `m_axi_rx_tlast` is asserted.
m_axi_rx_tvalid	Input	Asserted (active-High) when data and control signals from an Aurora core are valid. Deasserted (Low) when data and/or control signals from an Aurora core should be ignored.
m_axi_rx_tlast	Input	Signals the end of the incoming frame (active-High, asserted for a single `user_clk` cycle).
data_err_count[0:7]	Output	Count of the number of RX frame data words received by the frame checker that did not match the expected value.
channel_up	Input	Asserted (active-High) when Aurora channel initialization is complete and channel is ready to send data.

Table 5-7: **FRAME_CHECK Framing User I/O Ports (RX)** ***(Cont'd)***

Name	Direction	Description
user_clk	Input	Parallel clock shared by the Aurora 64B/66B core and the user application.
reset	Input	Resets the Aurora core (active-High).

Streaming RX Data Interface

- In streaming mode, the incoming RX data is compared against calculated RX data.
- The RX data is compared only when `m_axi_rx_tvalid` is asserted.

Figure 5-8 shows the FRAME_CHECK streaming user interface of the Aurora 64B/66B core ports for RX data.

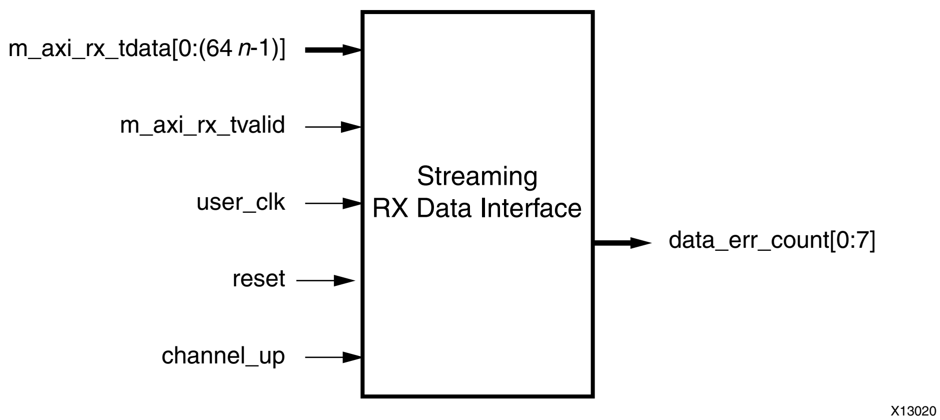


Figure 5-8: **Aurora 64B/66B Core Streaming RX Data Interface (FRAME_CHECK)**

Table 5-8 lists the FRAME_CHECK streaming RX data ports and their descriptions.

Table 5-8: **FRAME_CHECK Streaming User I/O Ports (RX)**

Name	Direction	Description
m_axi_rx_tdata[0:(64*n*–1)]	Input	Incoming frame data from channel partner (ascending bit order).
m_axi_rx_tvalid	Input	Asserted (active-High) when data and control signals from an Aurora core are valid. Deasserted (Low) when data and/or control signals from an Aurora core should be ignored.
data_err_count[0:7]	Output	Count of the number of RX data words received by the frame checker that did not match the expected value.
channel_up	Input	Asserted (active-High) when Aurora channel initialization is complete and channel is ready to send data.
user_clk	Input	Parallel clock shared by the Aurora 64B/66B core and the user application.
reset	Input	Resets the Aurora core (active-High).

UFC RX Interface

- Expected UFC RX data is computed by LFSR.
- Error checking and counter logic is similar to that of Framing RX Data Interface.
- If the incoming `m_axi_ufc_rx_tdata` does not match with expected RX UFC data, an 8-bit error counter is incremented.
- The error counter is indicated to the user application through the `ufc_err_count` port.

Figure 5-9 shows the FRAME_CHECK UFC RX interface of the Aurora 64B/66B core, with AXI4-Stream compliant ports for UFC RX data.

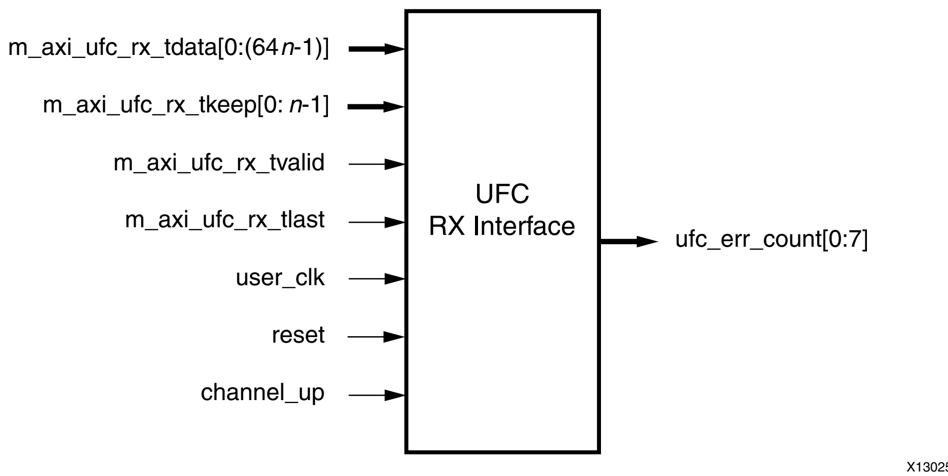


Figure 5-9: **Aurora 64B/66B Core UFC RX Interface (FRAME_CHECK)**

Table 5-9 lists the FRAME_CHECK UFC RX data ports and their descriptions.

Table 5-9: **FRAME_CHECK UFC User I/O Ports (RX)**

Name	Direction	Description
m_axi_ufc_rx_tdata [0: (64*n*–1)]	Input	Incoming UFC message data from the channel partner.
m_axi_ufc_rx_tkeep [0: *n*–1]	Input	Specifies the number of valid bytes of data presented on the `m_axi_ufc_rx_tdata` port on the last word of a UFC message. Valid only when `m_axi_ufc_rx_tlast` is asserted. *n* = 256 bytes maximum.
m_axi_ufc_rx_tvalid	Input	Asserted (active-High) when the values on the `m_axi_ufc_rx_tdata` port is valid. When this signal is not asserted, all values on the `m_axi_ufc_rx_tdata` port should be ignored.
m_axi_ufc_rx_tlast	Input	Signals the end of the incoming UFC message.
ufc_err_count[0:7]	Output	Count of the number of RX UFC data words received by the frame checker that did not match the expected value.
channel_up	Input	Asserted (active-High) when Aurora channel initialization is complete and channel is ready to send data.
user_clk	Input	Parallel clock shared by the Aurora 64B/66B core and the user application.
reset	Input	Resets the Aurora core (active-High).

User K RX Interface

- The `m_axi_rx_user_k_tvalid` is asserted during comparison of expected to actual User K data
- Incoming `m_axi_rx_user_k_tdata` is compared against predefined User K data.
- 8-bit `user_k_err_count` is incremented if the comparison fails.
- The error counter is indicated to the user application through `user_k_err_count` port.

Figure 5-10 shows the FRAME_CHECK User K RX interface of the Aurora 64B/66B core, with AXI4-Stream compliant ports for User K RX data.

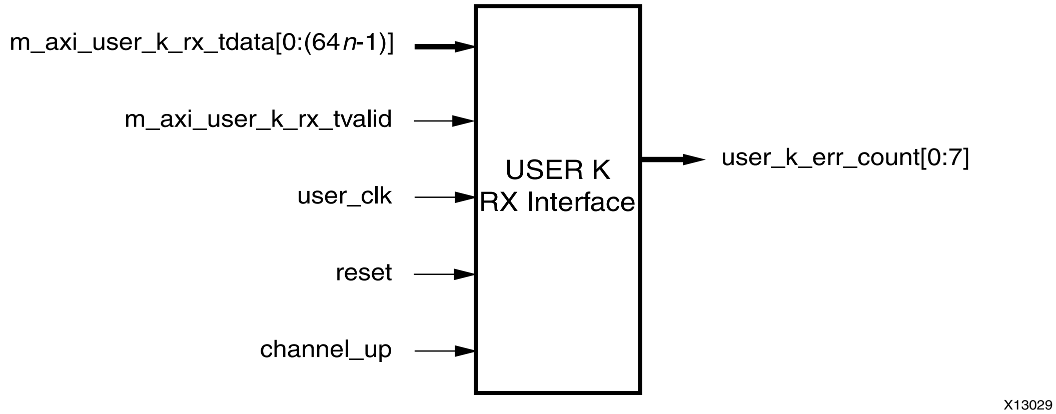


Figure 5-10: **Aurora 64B/66B Core User K RX Interface (FRAME_CHECK)**

Table 5-10 lists the FRAME_CHECK User K RX data ports and their descriptions.

Table 5-10: **FRAME_CHECK User K User I/O Ports (RX)**

Name	Direction	Description
m_axi_rx_user_k_tvalid	Input	Asserted (active-High) when User K data on `m_axi_rx_user_k_tdata` port is valid.
m_axi_rx_user_k_tdata[0:(64*n*–1)]	Input	Receive User K-blocks from the Aurora lane. Signal Mapping per lane: `m_axi_rx_user_k_tdata`={4'h0, User K Block No, User K Data}
user_k_err_count[0:7]	Output	Count of the number of RX User K data words received by the frame checker that did not match the expected value.
channel_up	Input	Asserted when Aurora channel initialization is complete and channel is ready to send data.
user_clk	Input	Parallel clock shared by the Aurora 64B/66B core and the user application.
reset	Input	Resets the Aurora core (active-High).

The Aurora 64B/66B example design has been tested with XST for synthesis and Mentor Graphics Questa® SIM for simulation.

Implementing the Example Design

The example design needs to be generated from the IP core. To do that, right-click the generated IP. Click **Open Example Design** on the menu displayed for the right-click operation. This action opens an example design for the generated IP core. You can click **Run Implementation** to run the Synthesis followed by implementation. Additionally you can also generate a bitstream by clicking **Generate Bitstream**.

Note: You need to specify LOC and IO standards in XDC for all input and output ports of the design.

Hardware Reset FSM in the Example Design

The Aurora 64B/66B v9.2 core example design incorporates a hardware reset FSM to perform repeated resets and monitoring robustness of the link. This FSM also contains an option to set different time periods between reset assertions. Also continuous `channel_up` and `link_reset` transition counters are monitored and the test status is reported through VIO.

The following signals are added in to the default ILA and VIOs for probing the link:

i_ila

- `tx_d_i[0:15]`: TX Data from the LocalLink Frame Gen module
- `rx_d_i[0:15]`: RX Data to the LocalLink Frame check module
- `data_err_count_o`: 8-bit Data error count value, it is expected to be 'd0 in normal operations
- `lane_up_vio_usrclk`: `lane_up` signal
- `channel_up_i`: `channel_up` signal
- `soft_err_i`: Soft error monitor
- `hard_err_i`: Hard error monitor

vio1_inst:

- `sysreset_from_vio_i`: `reset` input to example design
- `gtreset_from_vio_i`: `pma_init` to example design
- `vio_probe_in2`: Quality counters for Link status
- `rx_cdrovrden_i`: Used while enabling loopback mode
- `loopback_i`: Used while enabling loopback mode

vio2_inst:

- `reset_quality_cntrs`: Used to reset all the quality counters in the example design
- `reset_test_fsm_from_vio`: Used to reset the hardware reset test FSM
- `reset_test_enable_from_vio`: Used to enable/start the repeat reset test from the vio ports on the hardware.
- `iteraion_cnt_sel_from_vio`: Number of repeat reset iterations to be initiated. This is a 4-bit encoded value for a fixed number of iterations that can be seen in the example design when Vivado lab tools are enabled.
- `lnk_reset_in_initclk`: Input probe to monitor the assertion of `link_reset`

- soft_err_in_initclk: Input probe to monitor the soft_err status
- chan_up_transcnt_20bit_i [15:8]: Number of channel_up transaction counts; this can be used to monitor the number of reset iterations that have been completed.

  Note:

  a. chan_up_transcnt_20bit_i is probed only [15:8] bits; hence, this probe will take some time to update the status.

  b. If you want to change the number of reset iterations, it can be done through modifying the respective value for iteraion_cnt_sel_from_vio and correspondingly select chan_up_transcnt_20bit_i for probing the status.

vio3_inst:

- test_passed_r: Test pass status is asserted after the respective iteration count if resets are done successfully.
- test_failed_r: Test fail status is asserted if there is either a lack of channel_up or some data errors have occurred.
- lnkrst_cnt_20bit_vio_i: Probe to monitor the number of times the link_reset is asserted.
- reset_test_fsm_chk_time_sel: 3-bit encoded value probe to select the hardware reset_fsm check time for channel_up assertions after reset is deasserted.

Hardware FSM Operation:

In the example design (<user_component_name>_exdes.v), a new hardware initiated repeat reset FSM has been added to test the robustness of the link when subject to repeat reset. The FSM consists of IDLE, ASSERT_RST, DASSERT_RST, WAIT, WAIT1, CHECK, FAIL and DONE states.

1. In IDLE state, test_passed_r indicating reset test passed, test_failed_r indicating reset test fail, and timer_r providing iteration count of reset will default to 0.
2. When the reset_test_enable_from_vio signal from vio is asserted, the hardware FSM traverses to the ASSERT_RST state where the pma_init is asserted for a pre-defined time (28-bit count time).
3. This pma_init assertion ensures that there is a hot plug detected by the link partner. Then the hardware FSM traverses to the DEASSERT_RST state where the pma_init is deasserted and the timer is pre-loaded with a default value that can be configured using the reset_test_fsm_chk_time_sel vio signal.
4. Then the FSM moves to the WAIT state until the selected time is expired. In this state, all the checks like data errors and occurrences of soft errors are checked and it makes sure that the channel-up is asserted High and has not toggled more than once for this iteration of pma_init.

5. If this condition is not met, the FSM moves to fail state and the repeat reset run is stopped, else the FSM moves to WAIT1 state where few data packets are transmitted and received.
6. In the next state, the CHECK state, the channel-up transitions are checked again. If there is not more than one transition, the FSM returns to the IDLE state until the requested iterations are completed. This ensures that the link is robust and recovers reliably across multiple repeat resets of the link.

Test Bench

The Aurora core delivers a demonstration test bench for the example design. This chapter describes the Aurora test bench and its functionality. The test bench consist of the following modules:

- Device Under Test (DUT)
- Clock and reset generator
- Status monitor

The Aurora test bench components can change based on the selected Aurora core configurations, but the basic functionality remains the same for all of the core configurations.

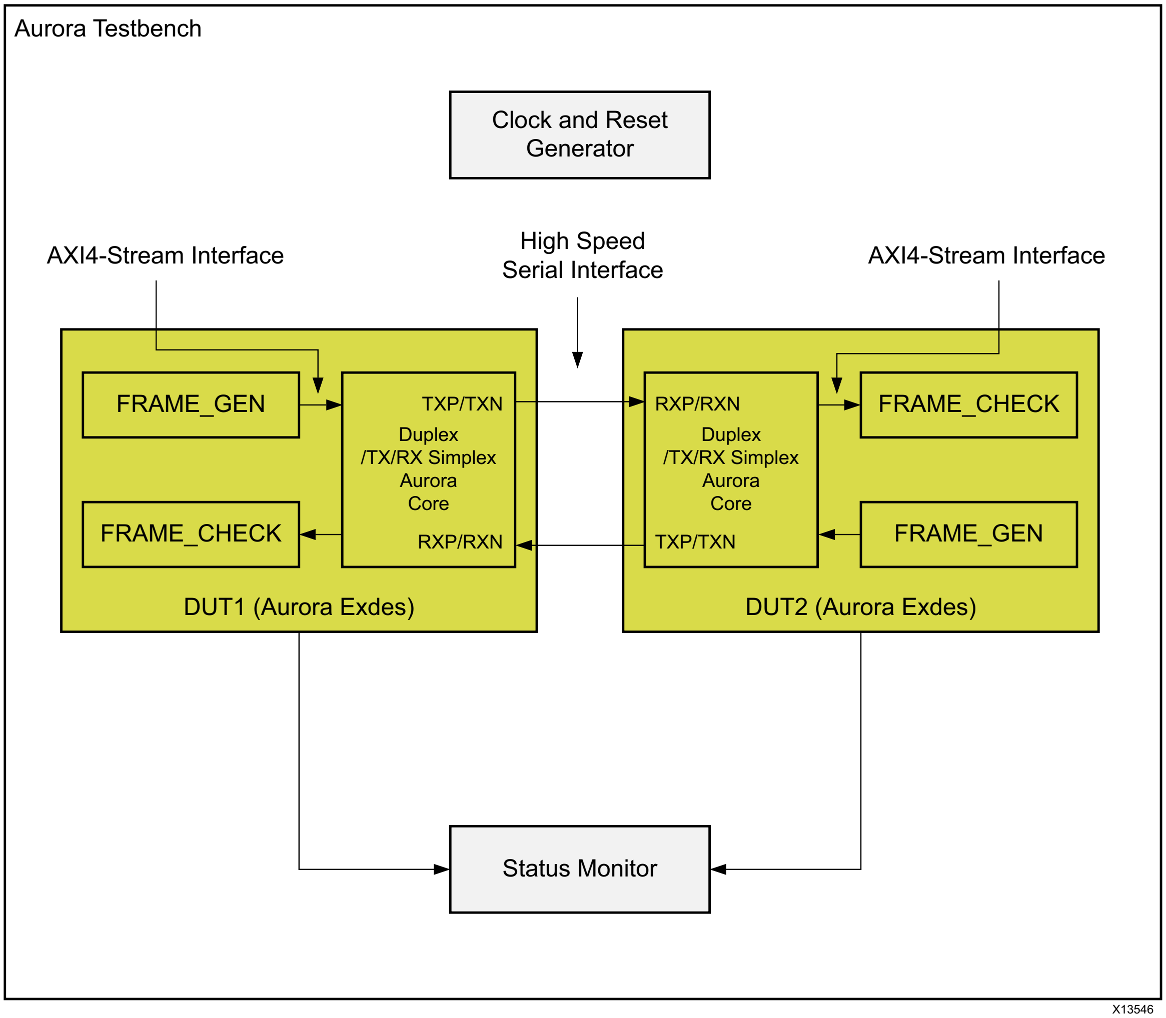


Figure 6-1: **Aurora Test Bench for Duplex Configuration**

The Aurora test bench environment connects the Aurora Duplex /TX/RX Simplex core in loopback using a high-speed serial interface. Figure 6-1 shows the Aurora test bench for the Duplex /TX/RX Simplex configuration.

The test bench looks for the state of the channel, then the integrity of the user data, UFC data and User-K for a predetermined simulation time. The `channel_up` assertion message indicates that link training and channel bonding (in case of multi-lane designs) are successful. The counter is maintained in the FRAME_CHECK module to track the reception of the erroneous data. The test bench flags an error when erroneous data is received.

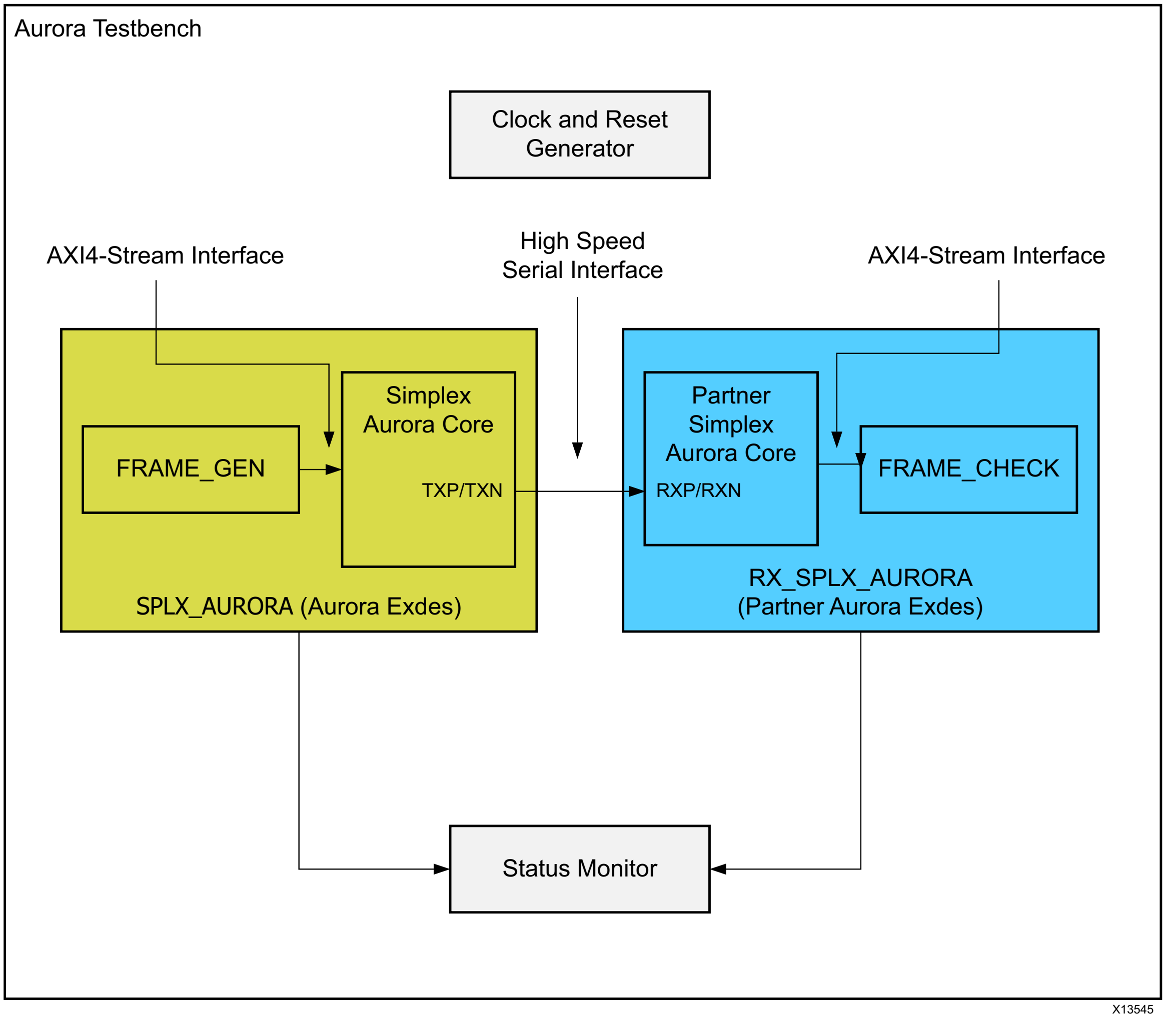


Figure 6-2: **Aurora Test Bench for Simplex Configuration**

The Aurora test bench environment connects the Aurora Simplex core to the partner Simplex Aurora core using the high-speed serial interface. Figure 6-2 shows the Aurora test bench for the Simplex configuration where DUT1 is configured as TX-only Simplex and DUT2 is configured as RX-only Simplex.

The test bench looks for the state of the transmitter channel and receiver channel and then the integrity of the user data for a predetermined simulation time. The `tx_channel_up` and `rx_channel_up` assertion messages indicate that link training and channel bonding (in case of multi-lane designs) are successful.

Appendix A

Verification, Compliance, and Interoperability

This appendix provides details about how this IP core was tested for compliance.

Aurora 64B/66B cores are verified for protocol compliance using an array of automated hardware and simulation tests. The core comes with an example design implemented using a linear feedback shift register (LFSR) for understanding and verification of the core features.

The Aurora 64B/66B core is verified using the Aurora 64B/66B Bus Functional Model (BFM) and proprietary custom test benches. The Aurora 64B/66B BFM verifies the protocol compliance along with interface level checks and error scenarios. An automated test system runs a series of simulation tests on the most widely used set of design configurations chosen at random. Aurora 64B/66B cores are also tested in hardware for functionality, performance, and reliability using Xilinx® GTX transceiver demonstration boards. Aurora verification test suites for all possible modules are continuously being updated to increase test coverage across the range of possible parameters for each individual module.

The boards used for hardware testing of the Aurora 64B/66B core are KC724, KC705, VC7203, and ZC723. A series of test scenarios are validated using these platforms.

To achieve interoperability among different versions of Aurora 64B/66B cores for 7 series FPGA GT transceivers, a new user-level parameter has been introduced in v9.2 version of the core. This parameter must be set to inter-operate the core as shown in Table A-1.

Table A-1: **Aurora 64B/66B Interoperability**

V9_2 Interoperability with V8_1 of Aurora 64B/66B		
V9_2\V8_1	**V8_1 GTX**	**V8_1 GTH**
V9_2 GTX	√	√
V9_2 GTH	√	√
V9_2 Interoperability with V7_3 of Aurora 64B/66B		
V9_2\V7_3	**V7_3 GTX**	**V7_3 GTH**
V9_2 GTX	√	x
V9_2 GTH	√	x

To handle backward compatibility with earlier core versions, two parameters, BACKWARD_COMP_MODE1 and BACKWARD_COMP_MODE2, are included in the <user_component_name>_core.v module.

BACKWARD_COMP_MODE1 /BACKWARD_COMP_MODE2

- Default value is set to 0. This will ensure interoperability between v9.2 core and v9.1 core and between v9.2 core and v9.0 core.
- Set both these parameters to 1 to make the v9.2 core interoperate with the v8.1 core or with the v7.3 core.

Migrating and Upgrading

This appendix contains information about migrating a design from ISE® to the Vivado® Design Suite, upgrading to a more recent version of the IP core, and migrating legacy (LocalLink based) Aurora Cores to the AXI4-Stream Aurora Core.

For customers upgrading in the Vivado Design Suite, important details (where applicable) about any port changes and other impact to user logic are included.

Device Migration

If you are migrating from a 7 series GTX or GTH device to an UltraScale™ GTH device, the prefixes of the optional transceiver debug ports for single-lane cores are changed from "`gt0`", "`gt1`" to "`gt`", and the suffix "`_in`" and "`_out`" are dropped. For multi-lane cores, the prefixes of the optional transceiver debug ports gt(n) are aggregated into a single port. For example: `gt0_gtrxreset` and `gt1_gtrxreset` now become `gt_gtrxreset [1:0]`. This is true for all ports, with the exception of the DRP buses which follow the convention of `gt(n)_drpxyz`.

It is important to update your design to use the new transceiver debug port names. For more information about migration to UltraScale devices, see the *UltraScale Architecture Migration Methodology Guide* (UG1026) [Ref 12].

Migrating to the Vivado Design Suite

For information about migrating to the Vivado Design Suite, *see the ISE to Vivado Design Suite Migration Guide* (UG911) [Ref 13].

Upgrading in the Vivado Design Suite

This section provides information about any changes to the user logic or port designations that take place when you upgrade to a more current version of this IP core in the Vivado Design Suite.

In the latest revision of the core, there have been several changes which make the core pin-incompatible with the previous version (s). These changes were required as part of the general one-off hierarchical changes to enhance the customer experience and are not likely to occur again.

Shared Logic

As part of the hierarchical changes to the core, it is now possible to have the core itself include all of the logic which can be shared between multiple cores, which was previously exposed in the example design for the core.

RECOMMENDED: *If you are updating a previous version to a new one with shared logic, there is no simple upgrade path and it is recommended to consult the Shared Logic sections of this document for more guidance.*

Updates from v9.1 Core

Table B-1 explains the new ports that are added in v9.2 of Aurora 64B/66B core and provides guidance on the impact of these port additions on the existing v9.1-based designs.

Table B-1: **New Ports added for Aurora 64B/66B in 2014.1**

New Ports	Direction	Reason for Adding
gt_refclk1_out gt_refclk2_out	Output (master)	In shared logic designs, the master (shared logic in core) which has the differential GT input will instantiate the IBUFDS and will pass on the single-ended `refclk` to the GT. However, the slave (shared logic in the example design) will be expecting a single-ended `refclk` input which is not readily available from the master in the V9.1 version. This might result in an additional differential GT `refclk` and an IBUFDS instantiated outside and gives it as an input to the slave or manually bring out the `gt_refclk[1,2]` from the master; hence in v9.2 these two new output ports are added.
gt_reset_out	Output (master)	Provides `gt_reset_out` that connects to the slave `pma_init` input port to ensure the proper GT reset sequencing of master and slave designs
mmcm_not_locked_out	Output (master)	The slave design has the `mmcm_not_locked` port as an input, which is being used in the TX startup FSM. The master design will have an instance of MMCM and will drive `mmcm_not_locked` as output. The Aurora 64B/66B v9.1 core did not have this output port from the core.
s_axi_bready s_axi_bready_lane[1..15]	Input (DRP mode is AXI4-Lite)	To ensure AXI4-Lite compatibility, this new port has been added when the DRP mode is AXI4-Lite. To ensure a smooth upgrade, this new input is tied to a default value of 1 and need not be driven by user logic.

When IP is upgraded, critical warnings occur due to these port additions. All these critical warnings can be safely ignored if the design does not use the functionality provided with the new ports.

Updates from v9.0 Core

- In TX Startup FSM, the counting mechanism for `mmcm_lock_count` used to be done on `txuserclk`; this was a limitation as this was a recovered clock, now using `stable_clock` for the MMCM Lock synchronization.
- RX datapath is now made 32-bit until the CBCC module, Width conversion logic and `clk_en` generation is avoided; this is handled in the CBCC module before writing data in to FIFO.
- Lane skew tolerance enhancement; now able to tolerate more lane to lane skew.
- Logic to detect Polarity inversion and to invert polarity while lane init is enabled.
- Internally the core generates `tx_channel_up` for Aurora TX logic and `rx_channel_up` for Aurora RX logic; this ensures that RX logic will be active and ready to receive before TX logic. `rx_channel_up` is given out as `channel_up`.
- Common reset and controls across all lanes.
- Increased the RX CDR lock time from 50 KUI to 37 MUI as suggested by the transceiver user guide.
- Increased the Block sync header max count from 64 to 60K to increase the robustness of the link.
- Allowed transmission of more idle characters to add more robustness to link during channel initialization.
- Removed the reset to scrambler and made it free running to achieve faster CDR lock; the default pattern sent by scrambler is the scrambled value of NA idle character.
- Updated GTH transceiver QPLL attributes - See AR 56332.
- Shared logic and optional transceiver control and status debug ports are added.
- Updated synchronizers for clock domain crossing to reduce "Mean Time Between Failures" (MTBF) from meta-stability, currently using a common synchronizer module, and applying false path constraints only for the first stage of the flops.
- Added support for Cadence IES and Synopsys VCS simulators.
- Added Vivado lab tools support for debug.
- Added quality counters in the example design to increase the test quality.
- Added hardware reset state machine in example design to perform repeat reset testing.

Migrating Legacy (LocalLink based) Aurora Cores to the AXI4-Stream Aurora Core

Prerequisites

- Vivado design tools build containing the Aurora 64B/66B v9.x core supporting the AXI4-Stream protocol
- Familiarity with the Aurora directory structure
- Familiarity with running the Aurora example design
- Basic knowledge of the AXI4-Stream and LocalLink protocols
- Latest product guide (PG074) of the core with the AXI4-Stream updates
- Legacy documents: *LogiCORE IP Aurora 64B/66B v4.2 Data Sheet* (DS528) [Ref 14], *LogiCORE IP Aurora 64B/66B v4.1 Getting Started Guide* (UG238) [Ref 15], and *LogiCORE IP Aurora 64B/66B v4.2 User Guide* (UG237) [Ref 16] for reference.
- Migration guide (this Appendix)

Overview of Major Changes

The major change to the core is the addition of the AXI4-Stream interface:

- The user interface is modified from the legacy LocalLink (LL) to AXI4-Stream.
- All AXI4-Stream signals are active-High, whereas LocalLink signals are active-Low.
- The user interface in the example design and design top file is AXI4-Stream.

- A new shim module is introduced in the AXI4-Stream Aurora core to convert AXI4-Stream signals to LL and LL back to AXI4-Stream,
  - The AXI4-Stream to LL shim on the transmit converts all AXI4-Stream signals to LL.
  - The shim deals with active-High to active-Low conversion of signals between AXI4-Stream and LocalLink.
  - Generation of SOF_N and REM bits mapping are handled by the shim.
  - The LL to AXI4-Stream shim on the receive converts all LL signals to AXI4-Stream.
- Each interface (PDU, UFC, and NFC) has a separate AXI4-Stream to LL and LL to AXI4-Stream shim instantiated from the design top file.
- Frame generator and checker have respective LL to AXI4-Stream and AXI4-Stream to LL shim instantiated in the Aurora example design to interface with the generated AXI4-Stream design.

Block Diagrams

Figure B-1 shows an example Aurora design using the legacy LocalLink interface. Figure B-2 shows an example Aurora design using the AXI4-Stream interface.

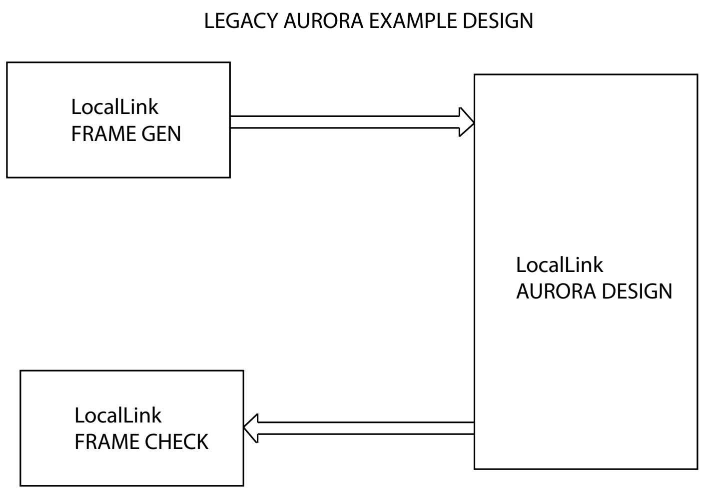


Figure B-1: **Legacy Aurora Example Design**

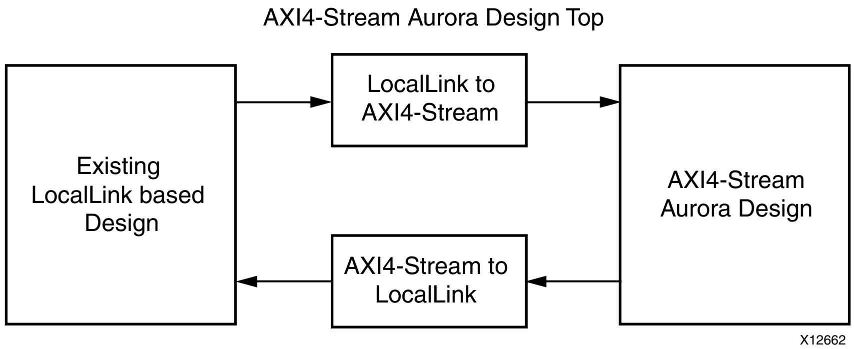


Figure B-2: **AXI4-Stream Aurora Example Design**

Signal Changes

Table B-2: **Interface Changes**

LocalLink Name	AXI4-S Name	Difference
TX_D	s_axi_tx_tdata	Name change only
TX_REM	s_axi_tx_tkeep	Name change. For functional differences, see Table 2-3, page 15
TX_SOF_N		Generated Internally
TX_EOF_N	s_axi_tx_tlast	Name change; Polarity
TX_SRC_RDY_N	s_axi_tx_tvalid	Name change; Polarity
TX_DST_RDY_N	s_axi_tx_tready	Name change; Polarity
UFC_TX_REQ_N	ufc_tx_req	Name change; Polarity
UFC_TX_MS	ufc_tx_ms	No Change
UFC_TX_D	s_axi_ufc_tx_tdata	Name change only
UFC_TX_SRC_RDY_N	s_axi_ufc_tx_tvalid	Name change; Polarity
UFC_TX_DST_RDY_N	s_axi_ufc_tx_tready	Name change; Polarity
NFC_TX_REQ_N	s_axi_nfc_tx_tvalid	Name change; Polarity
NFC_TX_ACK_N	s_axi_nfc_tx_tready	Name change; Polarity
NFC_PAUSE	s_axi_nfc_tx_tdata	Name change. For signal mapping, see Table 2-8, page 20
NFC_XOFF		
USER_K_DATA	s_axi_user_k_tdata	Name change. For signal mapping, see Table 2-9, page 21
USER_K_BLK_NO		
USER_K_TX_SRC_RDY_N	s_axi_user_k_tx_tvalid	Name change; Polarity
USER_K_TX_DST_RDY_N	s_axi_user_k_tx_tready	Name change; Polarity
RX_D	m_axi_rx_tdata	Name change only
RX_REM	m_axi_rx_tkeep	Name change. For functional difference, see Table 2-3, page 15
RX_SOF_N		Removed

Table B-2: **Interface Changes *(Cont'd)***

LocalLink Name	AXI4-S Name	Difference
RX_EOF_N	m_axi_rx_tlast	Name change; Polarity
RX_SRC_RDY_N	m_axi_rx_tvalid	Name change; Polarity
UFC_RX_DATA	m_axi_ufc_rx_tdata	Name change only
UFC_RX_REM	m_axi_ufc_rx_tkeep	Name change For functional difference, see Table 2-7, page 18
UFC_RX_SOF_N		Removed
UFC_RX_EOF_N	m_axi_ufc_rx_tlast	Name change; Polarity
UFC_RX_SRC_RDY_N	m_axi_ufc_rx_tvalid	Name change; Polarity
RX_USER_K_DATA RX_USER_K_BLK_NO	m_axi_rx_user_k_tdata	Name change For functional difference, see Table 2-9, page 21
RX_USER_K_SRC_RDY_N	m_axi_rx_user_k_tvalid	Name change; Polarity

Migration Steps

Generate an AXI4-Stream Aurora core from the Vivado design tools.

Simulate the Core

1. Run the `vsim -do simulate_mti.do` file from the `/simulation/functional` directory.
2. Questa® SIM launches and compiles the modules.
3. The `wave_mti.do` file loads automatically and populates AXI4-Stream signals.
4. Allow the simulation to run. This might take some time.
   a. Initially lane up is asserted.
   b. Channel up is then asserted and the data transfer begins.
   c. Data transfer from all flow control interfaces now begins.
   d. Frame checker continuously checks the received data and reports for any data mismatch.
5. A 'TEST PASS' or 'TEST FAIL' status is printed on the Questa SIM console providing the status of the test.

Implement the Core

1. Run **`./implement.sh`** (for Linux) from the `/implement` directory.
2. The implement script compiles the core and runs through the Vivado design tool and generates a bit file and netlist for the core.

Integrate to an Existing LocalLink-based Aurora Design

1. The Aurora core provides a lightweight 'shim' to interface to any existing LL based interface. The shims are delivered along with the core from the aurora_64b66b_v8_0 version of the core.
2. See Figure B-2, page 135 for the emulation of a LL Aurora core from a AXI4-Stream Aurora core.
3. Two shims `<user_component_name>_ll_to_axi.v` and `<user_component_name>_axi_to_ll.v` are provided in the `src` directory of the AXI4-Stream Aurora core.
4. Instantiate both the shims along with `<user_component_name>.v` in the existing LL based design top.
5. Connect the shim and AXI4-Stream Aurora design as shown in Figure B-2, page 135.

6. The latest AXI4-Stream Aurora core can be plugged into any existing LL design environment.

Vivado IDE Changes

Figure B-3 shows the AXI4-Stream signals in the IP Symbol diagram.

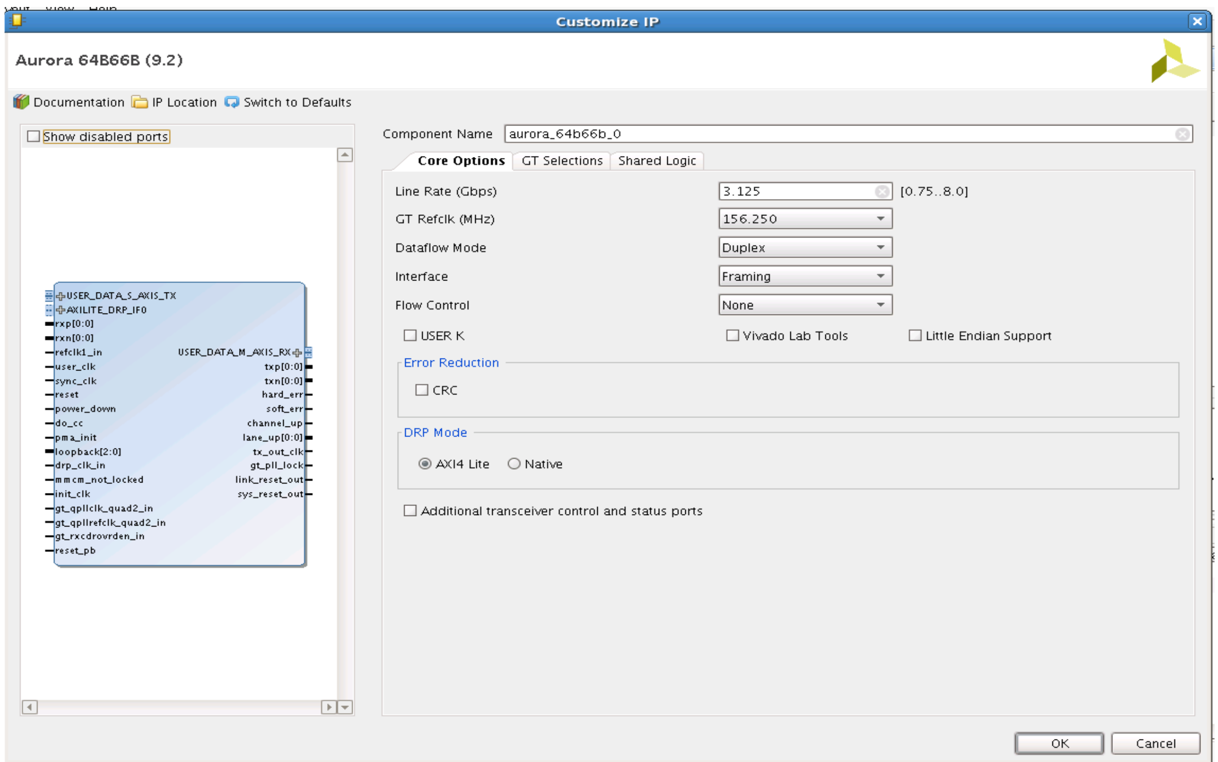


Figure B-3: **AXI4-Stream Signals**

Limitations

This section outlines the limitations of the Aurora 64B/66B core for AXI4-Stream support.

IMPORTANT: *Be aware of the following limitations while interfacing the Aurora 64B/66B core with the AXI4-Stream compliant interface core.*

Limitation 1:

The AXI4-Stream specification supports four types of data stream:

- Byte stream
- Continuous aligned stream
- Continuous unaligned stream
- Sparse stream

The Aurora 64B/66B core supports only continuous aligned stream and continuous unaligned stream. The position bytes are valid only at the end of packet.

Limitation 2:

The AXI4-Stream protocol supports transfer with zero data at the end of packet, but the Aurora 64B/66B core expects at least one byte should be valid at the end of packet.

Appendix C

Debugging

This appendix includes details about resources available on the Xilinx Support website and debugging tools.

Finding Help on Xilinx.com

To help in the design and debug process when using the Aurora 64B/66B core, the Xilinx Support web page (www.xilinx.com/support) contains key resources such as product documentation, release notes, answer records, information about known issues, and links for obtaining further product support. Also see the Aurora home page.

Documentation

This product guide is the main document associated with the Aurora 64B/66B core. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page (www.xilinx.com/support) or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator from the Design Tools tab on the Downloads page (www.xilinx.com/download). For more information about this tool and the features available, open the online help after installation.

Answer Records

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records for this core can be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use proper keywords such as

- Product name
- Tool message(s)
- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

To use the Answers Database Search:

1. Navigate to www.xilinx.com/support. The Answers Database Search is located at the top of this web page.
2. Enter keywords in the provided search field and select Search.
   - Examples of searchable keywords are product names, error messages, or a generic summary of the issue encountered.
   - To see all answer records directly related to the Aurora 64B/66B core, search for the phrase "Aurora 64B66B"

Master Answer Record for the Aurora 64B/66B Core

AR: 54368

Xilinx provides premier technical support for customers encountering issues that require additional assistance.

Contacting Technical Support

Xilinx provides technical support at www.xilinx.com/support for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation, or if changes are made to any section of the design labeled DO NOT MODIFY.

To contact Xilinx Technical Support:

1. Navigate to www.xilinx.com/support.
2. Open a WebCase by selecting the WebCase link located under Additional Resources.

When opening a WebCase, include:

- Target FPGA including package and speed grade.
- All applicable Xilinx Design Tools and simulator software versions.
- The XCI file created during Aurora 64B/66B core generation
- Additional files based on the specific issue might also be required. See the relevant sections in this debug guide for guidelines about which file(s) to include with the WebCase.

Note: Access to WebCase is not available in all cases. Log in to the WebCase tool to see your specific support options.

Debug Tools

There are many tools available to address Aurora 64B/66B core design issues. It is important to know which tools are useful for debugging various situations.

Vivado Lab Tools

Vivado® lab tools insert logic analyzer and virtual I/O cores directly into your design. Vivado lab tools also allow you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed. This feature in the Vivado IDE is used for logic debugging and validation of a design running in Xilinx devices.

The Vivado logic analyzer is used with the logic debug IP cores, including:

- ILA 3.0 (and later versions)
- VIO 3.0 (and later versions)

See the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 17].

Reference Boards

Various Xilinx development boards support the Aurora 64B/66B core. These boards can be used to prototype designs and establish that the core can communicate with the system.

- 7 series FPGA evaluation boards
  - KC705
  - KC724
  - VC7203
  - ZC723

Simulation Debug

Lanes and Channel do not come up in simulation

- The quickest way to debug these problems is to view the signals from one of the GTX or GTH transceiver instances that are not working.
- Make sure that the reference clock and user clocks are all toggling.

  Note: Only one of the reference clocks should be toggling, The rest will be tied Low.
- Check to see that `recclk` and `txoutclk` are toggling. If they are not toggling, you might have to wait longer for the PMA to finish locking. You should typically wait about 6–9 µs for lane up and channel up. You might need to wait longer for simplex/ 7 series FPGA designs.
- Make sure that `txn` and `txp` are toggling. If they are not, make sure you have waited long enough (see the previous bullet) and make sure you are not driving the `tx` signal with another signal.
- Check in the <user_component_name>_support module whether the `pll/mmcm_not_locked` signal and the `reset` signals are on your design. If these are being held active, your Aurora module will not be able to initialize.
- Be sure you do not have the `power_down` signal asserted.
- Make sure the `txn` and `txp` signals from each GTX or GTH transceiver are connected to the appropriate `rxn` and `rxp` signals from the corresponding GTX or GTH transceiver on the other side of the channel
- You will need to instantiate the "glbl" module and use it to drive the `power_up` reset at the beginning of the simulation to simulate the reset that occurs after configuration. You should hold this reset for a few cycles. The following code can be used an example:

```
//Simulate the global reset that occurs after configuration at the beginning
//of the simulation.
assign glbl.GSR = gsr_r;
assign glbl.GTS = gts_r;

initial
    begin
        gts_r = 1'b0;
        gsr_r = 1'b1;
        #(16*CLOCKPERIOD_1);
        gsr_r = 1'b0;
    end
```

- If you are using a multilane channel, make sure all the GTs on each side of the channel are connected in the correct order.

Channel comes up in simulation but S_AXI_TX_TREADY is never asserted (never goes High)

- If your module includes flow control but you are not using it, make sure the request signals are not currently driven High. `s_axi_nfc_tx_tvalid` and `ufc_tx_req` are active-High: if they are High, `s_axi_tx_tready` will stay Low because the channel will be allocated for flow control.
- Make sure `do_cc` is not being driven High continuously. Whenever `do_cc` is High on a positive clock edge, the channel is used to send clock correction characters, so `s_axi_tx_tready` is deasserted.
- If your module includes USER K Blocks but you are not using it, make sure the `s_axi_user_k_tx_tvalid` is not driven High. If it is High, `s_axi_tx_tready` will stay Low as channel will be allocated for USER K Blocks.
- If you have NFC enabled, make sure the design on the other side of the channel did not send an NFC XOFF message. This will cut off the channel for normal data until the other side sends an NFC XON message to turn the flow on again. See ug775.pdf for more details.

Bytes and words are being lost as they travel through the Aurora channel

- If you are using the AXI4-Stream interface, make sure you are writing data correctly. The most common mistake is to assume words are written without looking at `s_axi_tx_tready`. Also remember that the `s_axi_tx_tkeep` signal must be used to indicate which bytes are valid when `s_axi_tx_tlast` is asserted.
- Make sure you are reading correctly from the RX interface. Data and framing signals are only valid when `m_axi_rx_tvalid` is asserted.

Problems while compiling the design

Make sure you include all the files from the src directory when compiling.

Next Step

Open a support case to have the appropriate Xilinx expert assist with the issue.

To create a technical support case in WebCase, see the Xilinx website at: www.xilinx.com/support/clearexpress/websupport.htm

Items to include when opening a case:

- Detailed description of the issue
- Results of the steps listed previously
- Attach a VCD or WLF dump of the observation

Hardware Debug

As the transceiver is the critical building block in aurora core, debugging and ensuring proper operation of the transceiver is extremely important. Figure C-1 shows the steps involved for debugging transceiver related issues.

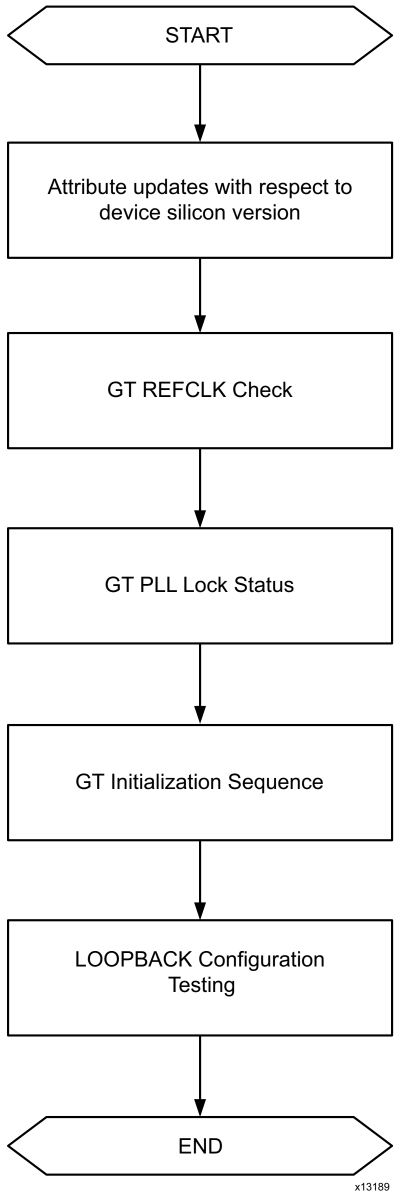


Figure C-1: **Transceiver Debug Flow Chart**

1. Attribute updates with respect to the device silicon version transceiver attributes must match with the silicon version of the device being used in the board. Apply all the applicable workarounds and Answer Records given for the relevant silicon version.
2. GT REFCLK Check

   A low jitter differential clock must be provided to the transceiver reference clock. Connecting the onboard differential clock to the transceiver will narrow down the issue to the external clock generation and/or external clock cables connected to transceiver.
3. GT PLL Lock Check

   Transceiver locks into the incoming GT REFCLK and asserts the `plllock` signal. This signal is available as the `tx_lock` signal in Aurora example design. Make sure that the GT PLL attributes are set correctly and that the transceiver generates `txoutclk` with expected frequency for the given line rate and datapath width options. It must be noted that the Aurora core uses Channel PLL/Quad PLL (CPLL/QPLL) in the generated core for GTX or GTH transceivers.
4. GT Initialization Sequence

   The Aurora core uses the sequential mode as the reset mode and all of the transceiver components are being reset sequentially one after another. `txresetdone` and `rxresetdone` signals are asserted at the end of the transceiver initialization. In general, `rxresetdone` assertion will take longer time compare to `txresetdone` assertion. Make sure, `gt_reset` signal pulse width duration matches with respective transceiver guideline. `txresetdone` and `rxresetdone` signals are available in the Aurora example design to monitor.
5. LOOPBACK Configuration Testing

   Loopback modes are specialized configurations of transceiver datapath. The `loopback` port at Aurora example design will control the loopback modes. Four loopback modes are available and refer respective transceiver UG for guidelines and more information. Figure C-2 illustrates a loopback test configuration with four different loopback modes.

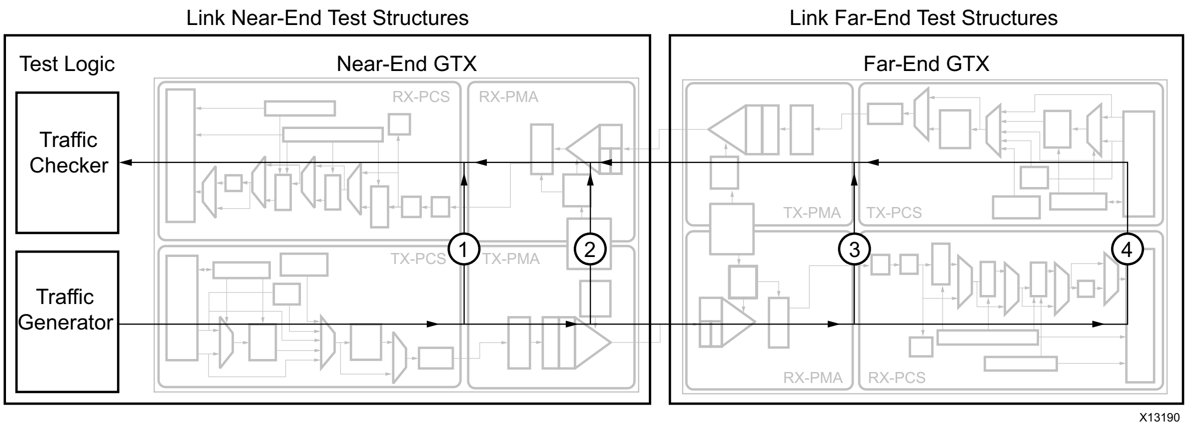


Figure C-2: **Loopback Testing Overview**

Design Bring-Up on Evaluation Board

Aurora Validation on the KC705 Board

Setup Requirements:

- Software: Vivado Design Suite
- Hardware Components required:
    - Kintex-7 FPGA KC705 Evaluation Kit Base Board
    - Two KC705 boards with power adapters

Validation and core generation steps:

1. Open the Vivado Design Suite and create a new project with a part number, typically, xc7k325tffg900-2 (or alternately you can select the boards option and in it the Kintex-7 KC705 Evaluation platform) and select **Finish**.
2. In the Project Manager window of Vivado, select IP catalog, and search for Aurora 64b66b in Communication & Networking => Serial Interfaces.
3. While customizing the Aurora 64B/66B core, in the tab **Core Option**, check the **Vivado Lab Tools** option. Then in the tab GT Selections, select GTXE2_X0Y8 in GTXQ2.
4. Generate and Open IP Example Design for the project.

5. Open <user_component_name>_exdes.xdc and make sure that pin locations of all the ports of the Aurora core are proper. As `hard_err`, `soft_err` and `data_err_count` are not being used in the evaluation board setup, you can add the following line in this file:

   set_property BITSTREAM.General.UnconstrainedPins {Allow} [current_design]

6. Save the file.

Table C-1: **Pin Locations**

Pin Name	Location onboard	Remarks
init_clk_p	AD12	
init_clk_n	AD11	
reset	AG5	
pma_init	AC6	
lane_up	A8	
channel_up	AA8	
hard_err		Not LOC constrained
soft_err		Not LOC constrained
data_err_count		Not LOC constrained
refclk_p	J8	GTXQ2_P
refclk_n	J7	GTXQ2_N

7. Run Synthesis, Implementation and generate the bitstream.
8. The procedure to connect the boards follows:
   a. `txp` from board 1 should be connected to `rxp` in board 2 and `txn` from board 1 should be connected to `rxn` in board 2.
   b. Similarly, `txp` from board 2 should be connected to `rxp` in board 1 and `txn` from board 2 should be connected to `rxn` in board 1.
9. Program the boards with the bit files, and with the help of ila/vio you can monitor `lane_up`, `channel_up`, `data_err_count`.

Interface Debug

If data is not being transmitted or received for the AXI4-Stream Interfaces, check the following conditions:

- If transmit `s_axi_tx_tready` is stuck Low following the `s_axi_tx_tvalid` input being asserted, the core cannot send data.
- If the receive `s_axi_tx_tvalid` is stuck Low, the core is not receiving data.
- Check that the `user_clk` inputs are connected and toggling.
- Check that the AXI4-Stream waveforms are being followed. See. Figure 2-8.
- Check core configuration.
- Add appropriate core specific checks.

Generating a GT Wrapper File from the Transceiver Wizard

The transceiver attributes play a vital role in the functionality of the Aurora 64B/66B core. Use the latest transceiver wizard to generate the transceiver wrapper file.

RECOMMENDED: *Xilinx strongly recommends that you update the transceiver wrapper file in the Design Suite tool releases when the transceiver wizard has been updated but the Aurora core has not.*

This appendix provides instructions to generate the transceiver wrapper files:

Use these steps to generate the transceiver wrapper file using the 7 series FPGAs Transceivers Wizard:

1. Using the IP catalog, run the latest version of the 7 series FPGAs Transceivers Wizard. Make sure the Component Name of the transceiver wizard matches the Component Name of the Aurora 64B/66B core.
2. Select the protocol template: Aurora 64B/66B.
3. Change the Line Rate in both TX and RX based on the application requirement.
4. Select the Reference Clock from the drop-down box menu in both TX and RX based on the application requirement.
5. Select transceiver(s) and the clock source(s) based on the application requirement.
6. On Page 3, select External Data Width of RX to be 32 Bits and Internal Data Width to be 32 bits. Ensure Tx is configured with 64-bit external data width and 32-bit internal data width.
7. Keep all other settings as default.
8. Generate the core.
9. Replace the `<component name>_gtx.v` file in the `example_design/gt/directory` available in the Aurora 64B/66B core with the generated `<component name>_gt.v` file generated from the 7 series FPGAs Transceivers Wizard.

The transceiver settings for the Aurora 64B/66B core are up to date now.

Note: The UltraScale™ architecture Aurora 64B/66B IP core uses the hierarchical core calling method to call the UltraScale device GTWizard IP core. In this way, all the transceiver attributes, parameters, and required workarounds are up to date. Manual editing of the UltraScale device transceiver files are not required in most of the cases.

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

For a glossary of technical terms used in Xilinx documentation, see the Xilinx Glossary.

References

These documents provide supplemental material useful with this product guide:

1. *7 Series FPGAs Overview* (DS180)
2. *UltraScale Architecture and Product Overview* (DS890)
3. *UltraScale Architecture GTH Transceivers User Guide* (UG576)
4. *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476)
5. *Aurora 64B/66B Protocol Specification v1.2* (SP011)
6. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
7. *Vivado Design Suite User Guide: Designing with IP* (UG896)
8. *Vivado Design Suite User Guide: Getting Started* (UG910)
9. *Vivado Design Suite User Guide - Logic Simulation* (UG900)
10. *7 Series GTZ Transceiver User Guide* (UG478)
11. *UltraScale FPGAs Transceivers Wizard Product Guide* (PG182)
12. *UltraScale Architecture Migration Methodology Guide* (UG1026)
13. *ISE to Vivado Design Suite Migration Guide* (UG911)
14. *LogiCORE IP Aurora 64B/66B v4.2 Data Sheet* (DS528)
15. *LogiCORE IP Aurora 64B/66B v4.1 Getting Started Guide* (UG238)
16. *LogiCORE IP Aurora 64B/66B v4.2 User Guide* (UG237)

17. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
18. *Vivado AXI Reference Guide* (UG1037)
19. *Virtex-7 FPGAs Data Sheet: DC and Switching Characteristics* (DS183)
20. *Kintex-7 FPGAs Data Sheet: DC and Switching Characteristics* (DS182)
21. *Synthesis and Simulation Design Guide* (UG626)
22. *ARM® AMBA® 4 AXI4-Stream Protocol v1.0 Specification* (ARM IHI 0051A)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
06/04/2014	9.2 (Rev 1)	• Added User Parameter information.
04/02/2014	9.2	• Added C_EXAMPLE_SIMULATION parameter for post synthesis/implementation simulation speedup. • Added support for UltraScale™ devices. • Enhanced support for IP Integrator. • Added Little endian support for data and flow control interfaces as non-default Vivado® IDE selectable option. • Provided interoperability guidance. • Resolved functional issue seen with specific frame lengths in certain scenarios.
12/18/2013	9.1	• Added default information to `init_clk_p, initclk_n`, and INIT_CLK description. • Updated reset sequencing steps and waveform. • Added information about `pma_init` staging. • Updated screen captures. • Added sequence of steps describing hardware FSM reset
10/02/2013	9.0	• Added new chapters: Simulation, Test Bench and Synthesis and Implementation. • Added shared logic and transceiver debug features. • Updated directory and file structure. • Changed signal and port names to lowercase. • Added Zynq®-7000 device support. • Updated RX datapath architecture. • Updated Aurora Simplex Operation description. • Updated Figure 3-2 and screen captures in Chapter 4. • Updated Hot-Plug Logic description. • Added IP Integrator support. • Updated XDC file for the example design. • Added design bring-up on evaluation board information.

Date	Version	Revision
06/19/2013	8.1	• Revision number advanced to 8.1 to align with core version number. • Updated for 2013.2 release and core version 8.1. • Fixed a NFC transmit failure scenario when Clock Correction is transmitted in conjunction with the second NFC request. NFC state machine is updated to handle such scenarios.
03/20/2013	2.0	• Updated for 2013.1 release and core version 8.0. • Removed all ISE® design tools and Virtex®-6 related device information. • Added Reset waveforms • Updated debug guide with core and transceiver debug details • Created lowercase ports for Verilog • Added Simplex TX/RX support • Enhanced protocol to increase Channel Init time • Included TXSTARTUPFSM and RXSTARTUPFSM modules to control GT reset sequence
12/18/2012	1.0.1	• Updated for 14.4 and 2012.4 release. • Added TKEEP description • Updated Debugging appendix.
10/16/2012	1.0	Initial Xilinx release as a product guide. This document replaces UG775, *LogiCORE IP Aurora 64B/66B User Guide* and DS815, *LogiCORE IP Aurora 64B/66B Data Sheet*. • Added section explaining constraining of the core. • Added section explaining core debugging.
06/04/2014		

Please Read: Important Legal Notices