

7 Series FPGAs Transceivers Wizard v3.6

LogiCORE IP Product Guide

Vivado Design Suite

PG168 May 19, 2022

Xilinx is creating an environment where employees, customers, and partners feel welcome and included. To that end, we're removing non-inclusive language from our products and related collateral. We've launched an internal initiative to remove language that could exclude people or reinforce historical biases, including terms embedded in our software and IPs. You may still find examples of non-inclusive language in our older products as we work to make these changes and align with evolving industry standards. Follow this [link](#) for more information.



Table of Contents

IP Facts

Chapter 1: Overview

About the Wizard	5
Feature Summary	9
Applications	11
Unsupported Features	12
Licensing and Ordering	12

Chapter 2: Product Specification

Performance	13
Port Descriptions	14

Chapter 3: Designing with the Core

General Design Guidelines	33
Clocking	35
Resets	38
Reset Sequence Modules for GTH and GTP Transceivers	40

Chapter 4: Design Flow Steps

Customizing and Generating the Core	41
Constraining the Core	73
Simulation	78
Synthesis and Implementation	78

Chapter 5: Example Design

Functional Simulation Using the Vivado Design Tools	79
Implementing Using the Vivado Design Tools	80
Timing Simulation Using the Vivado Design Tools	80
Using Vivado Design Suite Debug Feature with the Wizard	81
Directory and File Contents	82
Output Generation	83
Directory and File Contents	84
Example Design Description for GTX, GTH, and GTP Transceivers	88

Reset Sequence Modules for GTH and GTP Transceivers	93
Example Design Description for GTZ Transceivers.	93
Known Limitations of the GTZ Wizard	101
Known Limitations of the Wizard	101

Appendix A: Verification, Compliance, and Interoperability

Simulation	103
Hardware Testing	103

Appendix B: Upgrading

Migrating to the Vivado Design Suite.	104
Upgrading in the Vivado Design Suite	104

Appendix C: Debugging

Finding Help on Xilinx.com	114
Debug Tools	115
Wizard Validation	116
Simulation Debug.	125
Next Step	128
Hardware Debug	128
Loopback Limitations.	132
GT Debug Using IBERT	132
Debugging Using Serial I/O Analyzer	132
Debugging Using Embedded BERT	133
7 Series GT Wizard Hardware Validation on the KC705 Evaluation Board	133

Appendix D: Additional Resources and Legal Notices

Xilinx Resources	151
Documentation Navigator and Design Hubs	151
References	151
Revision History	152
Please Read: Important Legal Notices	153

Introduction

The 7 series FPGAs Transceivers Wizard LogiCORE™ IP automates the task of creating HDL wrappers to configure Xilinx® 7 series FPGA on-chip transceivers. The wizard customization Vivado® Integrated Design Environment (IDE) allows you to configure one or more high-speed serial transceivers using either pre-defined templates supporting popular industry standards, or from scratch to support a wide variety of custom protocols.

IMPORTANT: Download the most up-to-date IP update before using the Wizard.

Features

- Creates customized HDL wrappers to configure high-speed serial transceivers in 7 series FPGAs.
- Automatically configures analog settings.
- Predefined templates are provided for Aurora 8B/10B, Aurora 64B/66B, CEI-6G, DisplayPort, Interlaken, Open Base Station Architecture Initiative (OBSAI), OC192, OC48, SRIO, 10GBASE-R, Common Packet Radio Interface (CPRI), Gigabit Ethernet, 10 Gb Attachment Unit Interface (XAUI), RXAUI, and XLAUI, OTU3, 10GH Small Form-factor Pluggable Plus (SFP+), Optical Transport Network OTU3, V-by-One, SDI, and others as well as custom protocol using start from scratch.

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family ⁽¹⁾	Artix®-7, Kintex®-7, and Virtex®-7 FPGAs, and Zynq SoCs
Supported User Interfaces	Not Applicable
Resources	
Provided with Core	
Design Files	RTL
Example Design	Verilog and VHDL (Only Verilog is supported for GTZ transceivers)
Test Bench	Verilog and VHDL (Only Verilog is supported for GTZ transceivers)
Constraints File	XDC
Simulation Model	None
Supported S/W Driver ⁽²⁾	Not Applicable
Tested Design Flows⁽²⁾	
Design Entry	Vivado Design Suite
Simulation	For supported simulators, see the Xilinx Design Tools: Release Notes Guide .
Synthesis	Vivado Synthesis.
Support	
Release Notes and Known Issues	Master Answer Records: 54691
All Vivado IP Change Logs	Master Vivado IP Change Logs: 72775
Xilinx Support web page	

Notes:

1. For a complete list of supported devices, see the Vivado IP catalog.
2. For the supported versions of the tools, see the [Xilinx Design Tools: Release Notes Guide](#).

Overview

The 7 series FPGAs Transceivers Wizard (Wizard) can be used to configure one or more Virtex®-7, Kintex®-7, Artix®-7, and Zynq®-7000 device transceivers. Start from scratch, or use an industry-standard template to configure 7 series FPGA transceiver cores. The Wizard generates a custom wrapper for the transceivers with all inputs given through the transceiver wizard Vivado® IDE. In addition, the wizard generates an example design, test bench, and scripts to observe the transceivers operating under simulation and in hardware.

About the Wizard

The 7 series FPGAs Transceiver Wizard automates the task of creating HDL wrappers to configure the high-speed serial transceivers in Artix-7, Kintex-7, and Virtex-7 FPGAs.

The menu-driven interface allows you to configure one or more transceivers using predefined templates for popular industry standards, or by using custom templates, to support a wide variety of custom protocols. The Wizard produces a wrapper, an example design, and a test bench for rapid integration and verification of the serial interface with your custom function.

The Wizard produces a wrapper that instantiates one or more properly configured transceivers for custom applications ([Figure 1-1](#)).

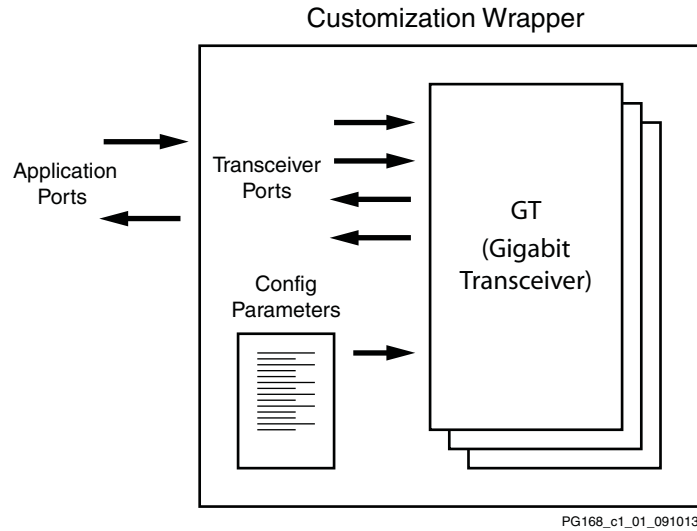


Figure 1-1: Transceiver Wizard Wrapper

The Wizard can be accessed from the Vivado Design Suite.

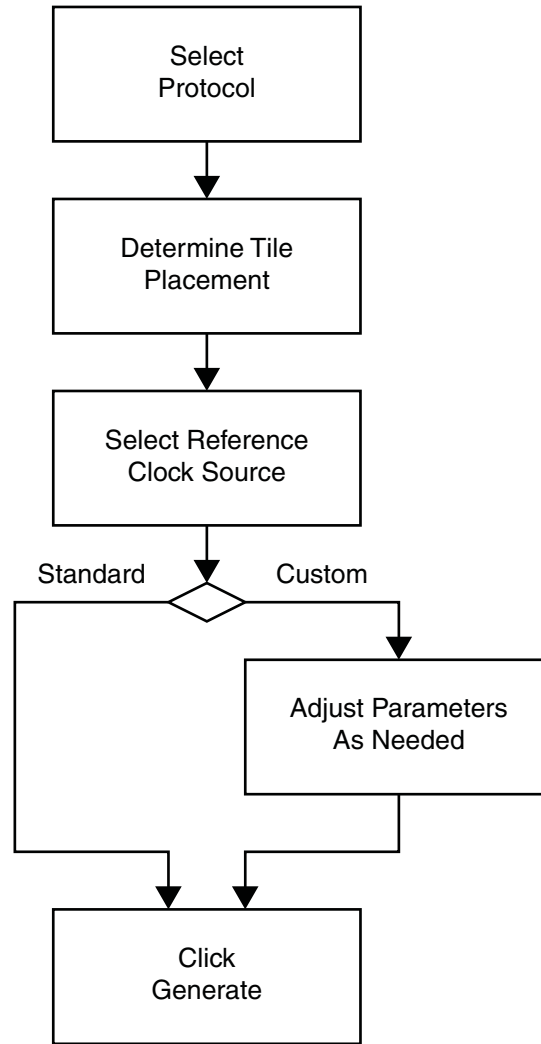
For the latest information on this wizard, see the [Architecture Wizards product information page](#).

For documentation, see the [7 series FPGAs Transceivers Wizard page](#).

Functional Overview

Figure 1-2 shows the steps required to configure transceivers using the Wizard. Start the Vivado IP catalog, select the 7 series FPGAs Transceivers Wizard, then follow the chart to configure the transceivers and generate a wrapper that includes the accompanying example design.

- To use an existing template with no changes, click **Generate**.
- To modify a standard template or start from scratch, proceed through the Wizard and adjust the settings as needed.



PG168_c1_02_091013

Figure 1-2: Wizard Configuration Steps

Structure of the Transceiver Wrapper, Example Design, and Test Bench

Figure 1-3 shows the relationship of the transceiver wrapper, example design, and test bench files generated by the Wizard. For details, see [Example Design Description for GTX, GTH, and GTP Transceivers](#), page 88.

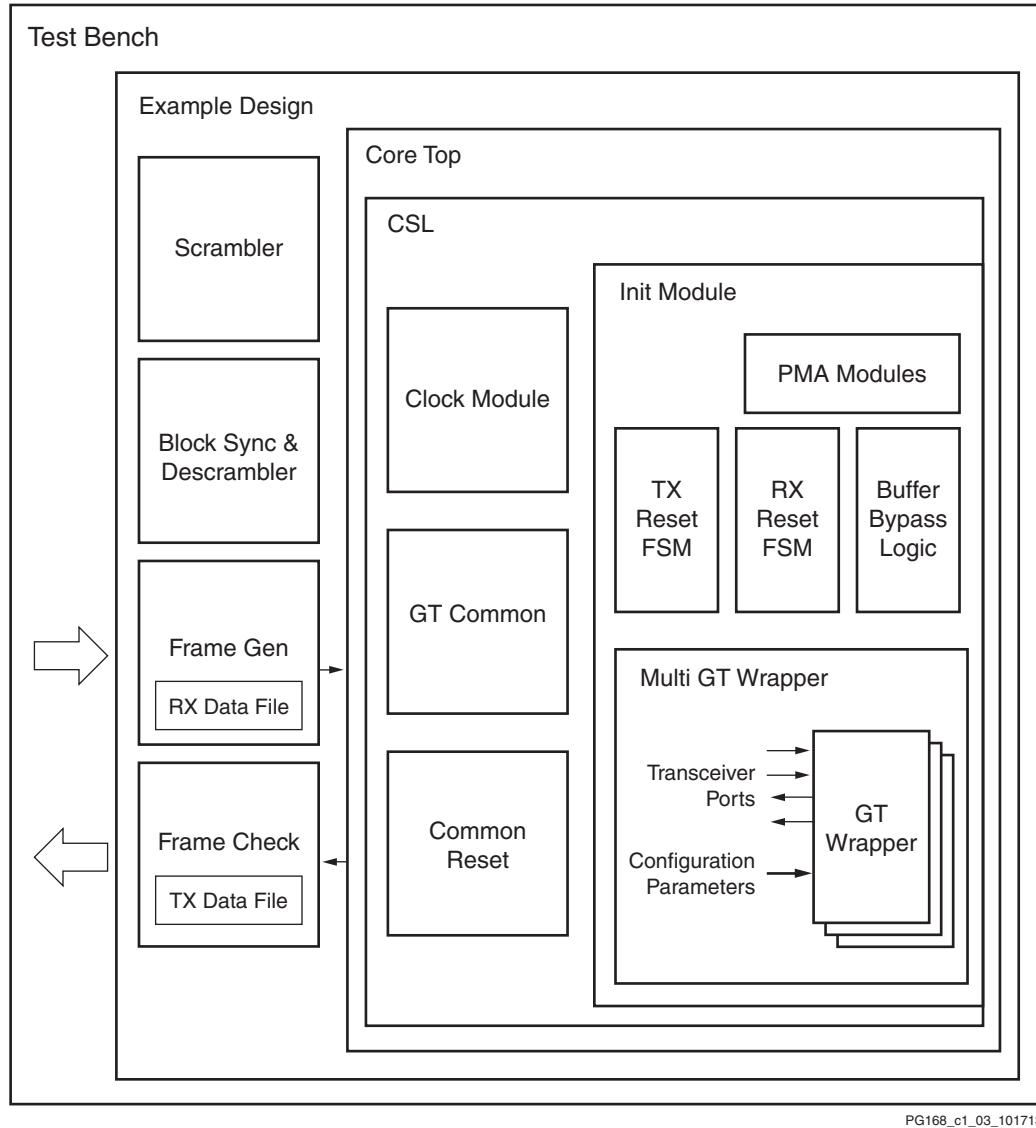


Figure 1-3: Structure of the Transceiver Wrapper, Example Design, and Test Bench

The following files are generated by the Wizard to illustrate the components needed to simulate the configured transceiver:

- Transceiver wrapper, which includes:
 - Specific serial transceiver configuration parameters set using the Wizard.
 - Transceiver primitive selected using the Wizard.
- Example design demonstrating the modules required to simulate the wrapper. These include:
 - FRAME_GEN module: Generates a user-definable data stream for simulation analysis.

- FRAME_CHECK module: Tests for correct transmission of data stream for simulation analysis.
- Test bench: Top-level test bench demonstrating how to stimulate the design.

Feature Summary

The Wizard has these features:

- Creates customized HDL wrappers to configure transceivers in the Kintex-7 and Virtex-7 FPGAs:
 - Predefined templates automate transceiver configuration for industry standard protocols.

GTX transceivers support:

- Common Packet Radio Interface (CPRI): 0.6, 1.2, 2.4, 3.072, 4.9, 6.144, and 9.83 Gb/s
- OC-48: 2.488 Gb/s
- OC-192: 9.956 Gb/s
- Gigabit Ethernet: 1.25 Gb/s
- Aurora 64B/66B: 12.5 Gb/s
- Aurora 8B/10B: 6.6 Gb/s
- DisplayPort: 1.620, 2.7, 5.4 Gb/s
- 10GBASE-R: 10.3125 Gb/s
- Interlaken: 4.25, 5.0, 6.25 Gb/s
- Open Base Station Architecture Initiative (OBSAI): 3.072 Gb/s
- OBSAI: 6.144 Gb/s
- 10 Gb Attachment Unit (XAUI): 3.125 Gb/s
- 10 Gb Reduced Attachment Unit (RXAUI): 6.25 Gb/s
- Serial ATA (SATA): 6.0
- Serial RapidIO Gen1: 1.25, 2.5, 3.125 Gb/s
- Serial RapidIO Gen2: 1.25, 2.5, 3.125, 5.0, 6.25 Gb/s
- JESD204: 3.0, 6.0 Gb/s
- 100 Gb Attachment Unit Interface (CAUI): 10.3125 Gb/s
- 10GBASE-KR: 10.3125 Gb/s

- Common Electrical Interface (CEI) 6G-SR: 4.976–6.375 Gb/s
- 40 Gb Attachment Unit Interface (XLAUI): 10.3125 Gb/s
- Quad Serial Gigabit Media Independent Interface (QSGMII): 5 Gb/s
- High-Definition Serial Digital Interface (HD-SDI)/3 Gb/s Serial Digital Interface (3G-SDI): 1.485/2.97 Gb/s

GTH transceivers support:

- CEI 6G-SR: 4.976–6.375 Gb/s
- Interlaken: 6.25 Gb/s
- 10GBASE-KR: 10.3125 Gb/s
- 10GBASE-R: 10.3125 Gb/s
- XLAUI: 10.3125 Gb/s
- CEI-11: 9.956–11.1 Gb/s
- CAUI: 10.3125 Gb/s
- OTU4: 11.18, 12.5, 13.1 Gb/s
- CPRI: 0.6, 1.2, 2.4, 3.072, 4.9, 6.144, and 9.83 Gb/s
- Gigabit Ethernet: 1.25 Gb/s
- OC-48: 2.48832 Gb/s
- OC-192: 9.956 Gb/s
- DisplayPort: 1.620, 2.7, 5.4 Gb/s
- JESD204
- Optical-channel Transport Lane (OTL) 3.4: 10.7546 Gb/s
- QSGMII: 5 Gb/s
- RXAUI: 6.25 Gb/s
- XAUI: 3.125 Gb/s
- Aurora 64B/66B: 12.5 Gb/s
- Aurora 8B/10B: 6.6 Gb/s
- Serial RapidIO Gen2: 1.25, 2.5, 3.125, 5.0 Gb/s

GTP transceivers support:

- CEI 6G-SR: 4.976–6.375 Gb/s
- Aurora 64B/66B: 6.6 Gb/s
- Aurora 8B/10B: 6.6 Gb/s
- DisplayPort: 1.620, 2.7, 5.4 Gb/s
- JESD204
- CPRI: 0.6, 1.2, 2.4, 3.072, 4.9, 6.144 Gb/s
- SDI/HD-SDI/3G-SDI: 0.27/1.485/2.97 Gb/s
- V-by-One®: 2.97/3.7125/1.485 Gb/s
- Gigabit Ethernet: 1.25 Gb/s
- QSGMII: 5 Gb/s
- RXAUI: 6.25 Gb/s
- XAUI: 3.125 Gb/s
- SATA: 6.0
- Serial RapidIO Gen1: 1.25, 2.5, 3.125 Gb/s
- Serial RapidIO Gen2: 1.25, 2.5, 3.125, 5.0, 6.25 Gb/s

GTZ transceivers support:

- CAUI4: 27.78125 Gb/s
- Custom protocols can be specified using the **Start from Scratch** option in the Vivado IDE.
- Automatically configures transceiver analog settings
- Supports 64B/66B, 64B/67B, and 8B/10B encoding/decoding. RXCDR_CFG calculations in modes other than 8B/10B ending/decoding mode will send scrambled or PRBS patterns.
- Includes an example design with a companion test bench as well as implementation and simulation scripts

Applications

The Transceiver Wrapper generated by the core can be interfaced with any of the protocol specific IP mentioned in [Feature Summary](#).

Unsupported Features

The Wizard can be used to generate designs with asymmetrical data widths (internal and external) on TX and RX but functional/timing simulation of the same is not supported. The Wizard does not enable users to select the transceivers from both columns (if available in a device). The Wizard generates only Verilog wrappers for GTZ transceivers.

Licensing and Ordering

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](#).

Product Specification

The 7 series FPGAs transceivers are power-efficient transceivers, supporting line rates of 6.6 Gb/s for GTP transceivers, 12.5 Gb/s for GTX transceivers, and 13.1 Gb/s for GTH transceivers. The GTX/GTH transceiver is highly configurable and tightly integrated with the programmable logic resources of the FPGA. For each of these line rates, you can select a custom value based on your requirements, or you can choose from pre-provided industry standard protocols (for example, CPRI, Gigabit Ethernet, or XAUI). Specify the number of serial transceivers for each line rate that is programmed with these settings. Because usage of the Quad PLL (QPLL) is recommended for line rates above 6.5 Gb/s, you can select QPLL/CPLL for each line rate falling in the range 0.6 Gb/s to 6.5 Gb/s.

Performance

The wrapper generated by the Wizard can be configured for high performance depending on the selection of the protocol standard.

Maximum Frequencies

For more details about frequencies, see the appropriate FPGA data sheet:

- *Virtex-7 FPGAs Data Sheet: DC and Switching Characteristics* (DS183) [\[Ref 1\]](#)
- *Kintex-7 FPGAs Data Sheet: DC and Switching Characteristics* (DS182) [\[Ref 2\]](#)
- *Artix-7 FPGAs Data Sheet: DC and Switching Characteristics* (DS181) [\[Ref 5\]](#)
- *Zynq-7000 SoC Data Sheet: DC and Switching Characteristics* (DS191) [\[Ref 6\]](#)

Port Descriptions

Table 2-1 describes the input and output ports provided by the 7 series FPGAs transceivers circuit. Some ports are optional, and those are optionally selected based upon the protocol selection. The availability of the ports is controlled by user-selected parameters. For example, the Aurora 64B/66B protocol template does not have a TXINHIBIT port, but the CPRI protocol template includes a TXINHIBIT optional port when generating through the Wizard. Any port that is not exposed is appropriately tied off.

Table 2-1: Port List

Port	Direction	Clock Domain	Description
GTREFCLK_PAD_N_IN/ GTREFCLK_PAD_P_IN	Input	-	External differential clock input pin pair for the reference clock of the 7 series FPGA transceiver Quad.
SYSCLK_IN	Input	-	System clock is used to drive the FPGA logic in the example design. When the DRP interface is enabled, DRP_CLK_IN is connected to SYSCLK_IN in the example design. This clock needs to be constrained in the Xilinx Design Constraints (XDC).
DRP_CLK_IN_P/ DRP_CLK_IN_N	Input	-	External differential clock input pin pair for the DRP interface clock. This clock needs to be constrained in the XDC. See the 7 series data sheets for more information.
TRACK_DATA_OUT	Output	rxusrclk	Indicates that valid data is received on the RX side. It is a level signal synchronous to RXUSRCLK2.
RXN_IN/RXP_IN	Input	RX serial clk	RXP and RXN are the differential input pairs for each of the receivers in the 7 series FPGA transceiver Quad.
TXN_OUT/TXP_OUT	Output	TX serial clk	TXP and TXN are the differential output pairs for each of the transmitters in the 7 series FPGA transceiver Quad.

Table 2-2 describes the optional ports for GTZ transceivers.

Table 2-2: Optional Ports for GTZ Transceivers

Options	Direction	Clock Domain	Description
TXFIBRESET	Input	Async	Brings out the TXFIBRESET ports to the example design from which you can control the RESET of the FIB portion of the GTZ transceiver.
RXFIBRESET	Input	Asyc	Brings out the RXFIBRESET ports to the example design from which you can control the RESET of the FIB portion of the GTZ transceiver.

Table 2-2: Optional Ports for GTZ Transceivers (Cont'd)

Options	Direction	Clock Domain	Description
TXFIFOSTATUS	Output	txusrclk[0..7] based on TXUSRCLK_SEL_LANEn setting	This brings out the TXFIFOSTATUS port to the example design allowing you to learn the FIFO status.
RXFIFOSTATUS	Output	rxusrclk[0..7] based on RXUSRCLK_SEL_LANEn setting	This brings out the RXFIFOSTATUS port to the example design allowing you to learn the FIFO status.
TXRATESEL	Input	Async	Brings the TXRATESEL ports out onto the example top level. These ports are used to control the TX PLL divider ratios.
RXRATESEL	Input	Async	Brings the RXRATESEL ports out onto the example top level. These ports are used to control the RX PLL divider ratios.
TXPOLARITY	Input	Async	Brings out the TXPOLARITY port to the example design.
RXPOLARITY	Input	Async	Brings out the RXPOLARITY port to the example design.
TXEN	Input	Async	Brings out the TXEN port to the example design.
RXEN	Input	Async	Brings out the RXEN port to the example design.
TXOUTPUTEN	Input	Async	Brings out the TXOUTPUTEN port to the example design.
TXATTNCTRL	Input	Async	Brings out the TXATTNCTRL port to the example design.
TXEQPOSTCTRL	Input	Async	Brings out the TXEQPOSTCTRL port to the example design.
TXEQPRECTRL	Input	Async	Brings out the TXEQPRECTRL port to the example design.
TXSLEWCTRL	Input	txusrclk[0..7] based on TXUSRCLK_SEL_LANEn setting	Brings out the TXSLEWCTRL port to the example design.
RXBITSLIP	Input	rxusrclk[0..7] based on RXUSRCLK_SEL_LANEn setting	Brings out the RXBITSLIP port onto the example design. This port can be used to slip data in raw mode.

Table 2-2: Optional Ports for GTZ Transceivers (Cont'd)

Options	Direction	Clock Domain	Description
RXSIGNALOK	Output	rxusrclk[0..7] based on RXUSRCLK _SEL _LANEn setting	Brings out the RXSIGNALOK port onto the example design.
CORECNTL	Input	Async	Brings out the CORECNTL ports onto the example design.
REFSEL	Input	Async	Brings out the REFSEL ports onto the example design.
PLLRECALEN	Input	Async	Brings out the PLLRECALEN ports onto the example design.
RXPRBS	Input	Async	Brings out all the RXPRBS related ports onto the example design.
TXPRBS	Input	Async	Brings out all the TXPRBS related ports onto the example design.
LOOPBACK	Input	Async	Brings out all the LOOPBACK control ports onto the example design.

Table 2-3 describes the 8B/10B optional ports.

Table 2-3: 8B/10B Optional Ports

Option		Direction	Clock Domain	Description
TX	TXBYPASS8B10B	Input	txusrclk	2-bit wide port disables 8B/10B encoder on a per-byte basis. High-order bit affects high-order byte of datapath.
	TXCHARDISPMODE	Input	txusrclk	2-bit wide ports control disparity of outgoing 8B/10B data. High-order bit affects high-order byte of datapath.
	TXCHARDISPVAL	Input	txusrclk	
RX	RXCHARISCOMMA	Output	txusrclk	2-bit wide port flags valid 8B/10B comma characters as they are encountered. High-order bit corresponds to high-order byte of datapath.
	RXCHARISK	Output	txusrclk	2-bit wide port flags valid 8B/10B K characters as they are encountered. High-order bit corresponds to high-order byte of datapath.

Note: Options not used by the XAUI example are shaded.

Table 2-4 shows the optional ports available for latency and clocking.

Table 2-4: Optional Ports

Option	Direction	Clock Domain	Description
TXPCSRESET	Input	Async	Active-High reset signal for the transmitter physical coding sublayer (PCS) logic.
TXBUFSTATUS	Output	txusrclk2	2-bit signal monitors the status of the TX elastic buffer. This option is not available when the TX buffer is bypassed.
TXRATE	Input	txusrclk2	Transmit rate change port.
RXPCSRESET	Input	Async	Active-High reset signal for the receiver PCS logic.
RXBUFSTATUS	Output	rxusrclk2	Indicates condition of the RX elastic buffer. Option is not available when the RX buffer is bypassed.
RXBUFRESET	Input	Async	Active-High reset signal for the RX elastic buffer logic. This option is not available when the RX buffer is bypassed.
RXRATE	Input	rxusrclk2	Receive rate change port.
QPLLPD	Input	Async	Visible only when GTX or GTH transceiver is selected. Powerdown port for QPLL.
CPLLPD	Input	Async	Visible only when GTX or GTH transceiver is selected. Powerdown port for channel PLL (CPLL).
PLLOPD	Input	Async	Visible only when GTP transceiver is selected. Powerdown port for PLL0.

Table 2-4: Optional Ports (Cont'd)

Option	Direction	Clock Domain	Description
PLL1PD	Input	Async	Visible only when GTP transceiver is selected. Powerdown port for PLL1.

Note: Options not used by the XAUI example are shaded.

Table 2-5 shows transceiver core debug ports that enable debug and control of the core for users wanting to drop the 7 series FPGAs Transceivers Wizard core into their designs.

Table 2-5: Transceiver Control and Status Interface

Port Names	XAUI	RXAUI	QSGMII	GigE PCS/PMA ⁽¹⁾	DisplayPort	SRIO Gen2	Aurora 8B10B	Aurora 64B66B	JESD204	OBSAI	CPRI	10GBASE-R	10GBASE-KR
TXCHARDISPMODE	No	No	Yes	Yes	Yes	No	Yes	No	No	N/A	No	No	No
TXCHARDISPVAL	No	No	Yes	Yes	Yes	No	Yes	No	No	N/A	No	No	No
RXCHARISCOMMA	Yes	Yes	Yes	Yes	No	Yes	Yes	No	Yes	N/A	Yes	No	No
RXCHARISK	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	N/A	Yes	No	No
RXSTARTOFSEQ	No	No	No	No	No	No	No	No	No	N/A	No	No	No
TXPCSRESET	No	No	No	No	Yes	Yes	No	No	No	N/A	No	No	No
TXPMARESET	No	No	No	No	No	No	No	No	No	N/A	No	No	No
TXBUFSTATUS	No	No	Yes	Yes	Yes	Yes	Yes	No	No	N/A	No	Yes	Yes
RXPCSRESET	No	No	No	No	Yes	Yes	No	No	No	N/A	No	No	No
RXPMARESET	No	No	No	No	Yes	No	No	No	No	N/A	No	No	No
RXBUFSTATUS	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	N/A	Yes	No	No
RXBUFRESET	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	N/A	Yes	No	No
RXSLIDE	No	No	No	No	No	No	No	No	No	N/A	No	No	No
RXBYTEISALIGN	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	N/A	Yes	No	No
RXBYTEREALIGN	Yes	Yes	Yes	Yes	No	Yes	Yes	No	Yes	N/A	Yes	No	No
RXCOMMADET	Yes	Yes	Yes	Yes	No	Yes	Yes	No	Yes	N/A	Yes	No	No
TXPOLARITY	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes	N/A	Yes	Yes	Yes
TXINHIBIT	No	No	No	No	Yes	Yes	No	No	No	N/A	No	No	No
TXDIFFCTRL	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	N/A	Yes	Yes	No
TXPOSTCURSOR	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	N/A	Yes	Yes	No
TXPRECURSOR	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	N/A	Yes	Yes	No
TXMAINCURSOR	No	No	No	No	No	No	Yes	Yes	No	N/A	No	No	No
TXQPISENN	No	No	No	No	No	No	No	No	No	N/A	No	No	No
TXQPISENP	No	No	No	No	No	No	No	No	No	N/A	No	No	No
TXQPIBIASEN	No	No	No	No	No	No	No	No	No	N/A	No	No	No
TXQPIWEAKPUP	No	No	No	No	No	No	No	No	No	N/A	No	No	No
TXQPISTRONGPDOWN	No	No	No	No	No	No	No	No	No	N/A	No	No	No
RXPOLARITY	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes	N/A	Yes	Yes	Yes
RXDFELPMRESET	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes	N/A	Yes	No	No
RXDFEAGCOVRDEN	No	No	Yes	Yes	No	No	No	Yes	No	N/A	No	No	No
RXLPMLFKLOVRDEN	No	No	No	No	No	No	No	Yes	No	N/A	No	No	No

Table 2-5: Transceiver Control and Status Interface (Cont'd)

Port Names	XAUI	RXAUI	QSGMII	GigE PCS/PMA ⁽¹⁾	DisplayPort	SRIO Gen2	Aurora 8B10B	Aurora 64B66B	JESD204	OBSAI	CPRI	10GBASE-R	10GBASE-KR
RXLPMHFOVRDEN	No	No	No	No	No	No	No	Yes	No	N/A	No	No	No
RXLPMHFHOLD (GTP)	Yes	Yes	Yes	Yes	Yes	No	Yes	No	Yes	N/A	Yes	No	No
RXLPMLFHOLD (GTP)	Yes	Yes	Yes	Yes	Yes	No	Yes	No	Yes	N/A	Yes	No	No
RXQPIEN	No	No	No	No	No	No	No	No	No	N/A	No	No	No
RXQPISENN	No	No	No	No	No	No	No	No	No	N/A	No	No	No
RXQPISENP	No	No	No	No	No	No	No	No	No	N/A	No	No	No
RXLPMEN	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	N/A	Yes	No	No
TXPRBSSEL	Yes	Yes	Yes	Yes	Yes	No	No	No	Yes	N/A	Yes	No	No
TXPRBSFORCEERR	Yes	Yes	Yes	Yes	Yes	No	No	No	Yes	N/A	Yes	Yes	Yes
RXPRBS_LOOPBACK			Yes	Yes	No	No	No	No	No	N/A	No	No	No
RXPRBSCNTRESET	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	N/A	Yes	No	No
RXPRBSERR	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	N/A	Yes	No	No
RXPRBSSEL	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	N/A	Yes	No	No
LOOPBACK	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	N/A	Yes	No	No
COMWAKEDET	No	No	No	No	No	No	No	No	No	N/A	No	No	No
TXDETECTRX	No	No	No	No	No	No	No	No	No	N/A	No	No	No
RXSTATUS	No	No	No	No	No	No	No	No	Yes	N/A	Yes	No	No
TXCOMMIT	No	No	No	No	No	No	No	No	No	N/A	No	No	No
TXELECIDLE	Yes	Yes	Yes	Yes	No	Yes	No	No	No	N/A	No	No	No
RXVALID	No	No	No	No	No	No	No	No	No	N/A	No	No	No
TXCOMSAS	No	No	No	No	No	No	No	No	No	N/A	No	No	No
PHYSTATUS	No	No	No	No	No	No	No	No	No	N/A	No	No	No
COMINITDET	No	No	No	No	No	No	No	No	No	N/A	No	No	No
TXCOMWAKE	No	No	No	No	No	No	No	No	No	N/A	No	No	No
COMSASDET	No	No	No	No	No	No	No	No	No	N/A	No	No	No
COMFINISH	No	No	No	No	No	No	No	No	No	N/A	No	No	No
TXPD	Yes	Yes	Yes	Yes	Yes	No	Yes	No	Yes	N/A	Yes	No	No
RXPD	Yes	Yes	Yes	Yes	Yes	No	Yes	No	Yes	N/A	Yes	No	No
TXRESETDONE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	N/A	Yes	No	No
RXRESETDONE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	N/A	Yes	No	No
DRPADDR	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	N/A	Yes	No	No
DRPEN	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	N/A	Yes	No	No
DRPDI	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	N/A	Yes	No	No
DRPWE	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	N/A	Yes	No	No
DRPRDY	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	N/A	Yes	No	No
DRPDO	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	N/A	Yes	No	No
RXDISPERR	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	N/A	Yes	No	No
RXNOTINTABLE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	N/A	Yes	No	No
EYESCANDATAERROR	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	N/A	Yes	Yes	Yes
EYESCANRESET	Yes	Yes	Yes	Yes	No	No	No	No	Yes	N/A	Yes	Yes	Yes

Table 2-5: Transceiver Control and Status Interface (Cont'd)

Port Names	XAU1	RXAU1	QSGMII	GigE PCS/PMA ⁽¹⁾	DisplayPort	SRIO Gen2	Aurora 8B10B	Aurora 64B66B	JESD204	OBSAI	CPRI	10GBASE-R	10GBASE-KR
EYESCANTRIGGER	Yes	Yes	Yes	Yes	No	No	No	No	Yes	N/A	Yes	Yes	Yes
RXMONITOROUT	Yes	Yes	Yes	Yes	No	No	No	No	Yes	N/A	Yes	No	No
RXMONITORSSEL	Yes	Yes	Yes	Yes	No	No	No	No	Yes	N/A	Yes	No	No
RXRATE	Yes	Yes	Yes	No	No	No	No	No	No			Yes	Yes
RXCDRHOLD	No	No	No	No	No	No	No	No				Yes	Yes

Notes:

- Changes applicable to both protocol templates: gigabit_ethernet_CC and gigabit_ethernet_noCC.

Table 2-6 shows transceiver debug ports that are available in all protocol templates for GTX and GTH transceivers.

Table 2-6: GTX and GTH Transceiver Debug Ports

Name	Direction	Clock Domain	Width	Description
DRP				
drpclk_in	Input	drp_clk_in	1 × Num. channels	Connects to DRPCLK on transceiver channel
drpaddr_in	Input	drp_clk_in	drp_addr_width x Num. channels	Connects to DRPADDR on transceiver channel primitives. See 7-series Architecture GTX/GTH Transceivers User Guide(UG476) for more information on drp_addr_width.
drpdi_in	Input	drp_clk_in	16 × Num. channels	Connects to DRPDI on transceiver channel primitives
drpen_in	Input	drp_clk_in	1 × Num. channels	Connects to DRPEN on transceiver channel primitives
drpwe_in	Input	drp_clk_in	1 × Num. channels	Connects to DRPWE on transceiver channel primitives
drpdo_out	Output	drp_clk_in	1 × Num. channels	Connects to DRPDO on transceiver channel primitives
drprdy_out	Output	drp_clk_in	1 × Num. channels	Connects to DRPRDY on transceiver channel primitives
TX reset and Initialization				

Table 2-6: GTX and GTH Transceiver Debug Ports (Cont'd)

Name	Direction	Clock Domain	Width	Description
gt_gttxreset_in	Input	sysclk_in	1 × Num. channels	Connects to GTTXRESET on transceiver channel primitives
gt_txpmareset_in	Input	Async	1 × Num. channels	Resets the TX PMA. Driven High, then deasserted to start the TX PMA reset process. Activating this port resets both the TX PMA and the TX PCS.
gt_txpcsreset_in	Input	Async	1 × Num. channels	Resets the TX PCS. Driven High, then deasserted to start the PCS reset process. Activating this port only resets the TX PCS.
gt_txuserddy_in	Input	Async	1 × Num. channels	Connects to TXUSERRDY on transceiver channel primitives
gt_txresetdone_out	Output	txusrclk2	1 × Num. channels	Indicates the GTX/GTH transceiver TX has finished reset and is ready for use. Driven Low when gttxreset goes High and not driven High until the GTX/GTH transceiver TX detects txuserddy High.
RX reset and Initialization				
gt_gtrxreset_in	Input	sysclk_in	1 × Num. channels	Connects to GTRXRESET on transceiver channel primitives
gt_rxpmareset_in	Input	Async	1 × Num. channels	Driven High, then deasserted to start the RX PMA reset process.
gt_rxpcsreset_in	Input	Async	1 × Num. channels	Driven High, then deasserted to start the RX PMA reset process. The rxpcsreset signal does not start the reset process until rxuserddy is High.

Table 2-6: GTX and GTH Transceiver Debug Ports (Cont'd)

Name	Direction	Clock Domain	Width	Description
gt_rxbufreset_in	Input	Async	1 × Num. channels	Driven High, then deasserted to start the RX elastic buffer reset process. In either single or sequential mode, activating rxbufreset resets the RX elastic buffer only.
gt_rxuserddy_in	Input	Async	1 × Num. channels	Connects to RXUSERRDY on transceiver channel primitives
gt_rxpmaresetdone_out	Output	Async	1 × Num. channels	Indicates that the RX PMA reset is complete. Driven Low when GTRXRESET or RXPMARESET is asserted. Available for duplex and RX-only simplex configurations. Available only with GTH transceivers.
gt_rxresetdone_out	Output	rxusrclk2	1 × Num. channels	When asserted, indicates the GTX/GTH transceiver RX has finished reset and is ready for use. Driven Low when gtrxreset is driven High. Not driven High until rxuserddy goes High.
gt_txbufstatus_out	Output	txusrclk2	2 × Num. channels	txbufstatus[1] : TX buffer overflow or underflow status. Whentxbufstatus[1] is set High, the signal 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.

Table 2-6: GTX and GTH Transceiver Debug Ports (Cont'd)

Name	Direction	Clock Domain	Width	Description
gt_rxbufstatus_output	Output	rxusrclk2	3 × Num. channels	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.
gt_txphaligndone_output	Output	Async	1 × Num. channels	Connects to TXPHALIGNDONE on transceiver channel primitives
gt_txphinitdone_output	Output	Async	1 × Num. channels	Connects to TXPHINITDONE on transceiver channel primitives
gt_txdlysresetdone_output	Output	Async	1 × Num. channels	Connects to TXDLYSRESETDONE on transceiver channel primitives
gt_rxphaligndone_output	Output	Async	1 × Num. channels	Connects to RXPHALIGNDONE of transceiver channel primitives
gt_rxdlysresetdone_output	Output	Async	1 × Num. channels	Connects to RXDLYSRESETDONE of transceiver channel primitives
gt_rxsyncdone_output	Output	Async	1 × Num. channels	Connects to RXSYNCDONE of transceiver channel primitives
cplllock_output	Output	Async	1 × Num. channels	Active-High PLL frequency lock signal indicating that PLL frequency is within the predetermined tolerance. The transceiver and its clock outputs are not reliable until this condition is met.

Table 2-6: GTX and GTH Transceiver Debug Ports (Cont'd)

Name	Direction	Clock Domain	Width	Description
qplllock_out	Output	Async	1 × Num. channels	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.
Signal Integrity and Functionality				
gt_eyescantrigger_in	Input	rxusrclk2	1 × Num. channels	Connects to EYESCANTRIGGER on transceiver channel primitives
gt_eyescanreset_in	Input	Async	1 × Num. channels	Connects to EYESCANRESET on transceiver channel primitives
gt_eyescandataerror_out	Output	Async	1 × Num. channels	Connects to EYESCANDATAERROR on transceiver channel primitives
gt_loopback_in	Input	Async	3 × Num. channels	Connects to LOOPBACK on transceiver channel primitives
gt_rxpolarity_in	Input	rxusrclk2	1 × Num. channels	Connects to RXPOLARITY on transceiver channel primitives
gt_txpolarity_in	Input	txusrclk2	1 × Num. channels	Connects to TXPOLARITY on transceiver channel primitives
gt_rxdfelprmreset_in	Input	Async	1 × Num. channels	Connects to RXDFELPMRESET on transceiver channel primitives
gt_rxlpmen_in	Input	rxusrclk2	1 × Num. channels	Connects to RXLPMEN on transceiver channel primitives
gt_txprecursor_in	Input	Async	5 × Num. channels	Connects to TXPRECURSOR on transceiver channel primitives
gt_txpostcursor_in	Input	Async	5 × Num. channels	Connects to TXPOSTCURSOR on transceiver channel primitives

Table 2-6: GTX and GTH Transceiver Debug Ports (Cont'd)

Name	Direction	Clock Domain	Width	Description
gt_txdiffctrl_in	Input	txusrclk2	4 × Num. channels	Connects to TXDIFFCTRL on transceiver channel primitives
gt_txprbsforceerr_in	Input	txusrclk2	1 × Num. channels	Connects to TXPRBSFORCEERR on transceiver channel primitives
gt_txprbsssel_in	Input	txusrclk	4 × Num. channels	Connects to TXPRBSSEL on transceiver channel primitives
gt_rxprbsssel_in	Input	rxusrclk2	4 × Num. channels	Connects to RXPRBSSEL on transceiver channel primitives
gt_rxprbserr_out	Output	rxusrclk2	1 × Num. channels	Connects to RXPRBSERR on transceiver channel primitives
gt_rxprbscntrese t_in	Input	rxusrclk2	1 × Num. channels	Connects to RXPRBSCNTRESET on transceiver channel primitives
gt_rxcdrhold_in	Input	Async	1 × Num. channels	Connects to RXCDRHOLD on transceiver channel primitives
gt_dmonitor_out_out	Output	Async	17 × Num. channels	Connects to DMONITOROUT on transceiver channel primitives
gt_rxdisperr_out	Output	rxusrclk2	1 × Num. channels	Indicates the corresponding byte shown on rxdata has a disparity error. The rxdisperr pin of the transceiver is connected to this port.

Notes:

1. Each port is prefixed with gt<lane>_ where lane is from 0 to Num. channels -1

Table 2-7 shows transceiver debug ports that are available in all protocol templates for GTP transceivers.

Table 2-7: GTP Transceiver Debug Ports

Name	Direction	Clock Domain	Width	Description
DRP				
drpclk_in	Input	drp_clk_in	1 × Num. channels	Connects to DRPCLK on transceiver channel
drpaddr_in	Input	drp_clk_in	drp_addr_width × Num. channels	Connects to DRPADDR on transceiver channel primitives. See 7-series Architecture GTX/ GTH Transceivers User Guide(UG476) for more information on drp_addr_width.
drpdi_in	Input	drp_clk_in	16 × Num. channels	Connects to DRPDI on transceiver channel primitives
drpen_in	Input	drp_clk_in	1 × Num. channels	Connects to DRPEN on transceiver channel primitives
drpwe_in	Input	drp_clk_in	1 × Num. channels	Connects to DRPWE on transceiver channel primitives
drpdo_out	Output	drp_clk_in	1 × Num. channels	Connects to DRPDO on transceiver channel primitives
drprdy_out	Output	drp_clk_in	1 × Num. channels	Connects to DRPRDY on transceiver channel primitives
TX reset and Initialization				
gt_gttxreset_in	Input	sysclk_in	1 × Num. channels	Connects to GTTXRESET on transceiver channel primitives
gt_txpmareset_in	Input	Async	1 × Num. channels	Resets the TX PMA. Driven High, then deasserted to start the TX PMA reset process. Activating this port resets both the TX PMA and the TX PCS.
gt_txpcsreset_in	Input	Async	1 × Num. channels	Resets the TX PCS. Driven High, then deasserted to start the PCS reset process. Activating this port only resets the TX PCS.

Table 2-7: GTP Transceiver Debug Ports (Cont'd)

Name	Direction	Clock Domain	Width	Description
gt_txuserrdy_in	Input	Async	1 × Num. channels	Connects to TXUSERRDY on transceiver channel primitives
gt_txresetdone_out	Output	txusrclk2_in	1 × Num. channels	Indicates the GTX/GTH transceiver TX has finished reset and is ready for use. Driven Low when gtxxreset goes High and not driven High until the GTX/GTH transceiver TX detects txuserrdy High.
RX reset and Initialization				
gt_gtrxreset_in	Input	sysclk_in	1 × Num. channels	Connects to GTRXRESET on transceiver channel primitives.
gt_rxpmareset_in	Input	Async	1 × Num. channels	Driven High, then deasserted to start the RX PMA reset process.
gt_rxpcsreset_in	Input	Async	1 × Num. channels	Driven High, then deasserted to start the RX PMA reset process. The rxpcsreset signal does not start the reset process until rxuserrdy is High.
gt_rxbufreset_in	Input	Async	1 × Num. channels	Driven High, then deasserted to start the RX elastic buffer reset process. In either single or sequential mode, activating rxbufreset resets the RX elastic buffer only.
gt_rxuserrdy_in	Input	Async	1 × Num. channels	Connects to RXUSERRDY on transceiver channel primitives

Table 2-7: GTP Transceiver Debug Ports (Cont'd)

Name	Direction	Clock Domain	Width	Description
gt_rxpmares etdo ne_out	Output	-	1 × Num. channels	Indicates that the RX PMA reset is complete. Driven Low when GTRXRESET or RXPMARESET is asserted. Available for duplex and RX-only simplex configurations. Available only with GTH transceivers.
gt_rxresetdo ne_out	Output	rxusrclk2	1 × Num. channels	When asserted, indicates the GTX/GTH transceiver RX has finished reset and is ready for use. Driven Low when gtrxreset is driven High. Not driven High until rxuserdy goes High.
Clocking				
gt_txbufstat us_o ut	Output	txusrclk2	2 × Num. channels	txbufstatus[1] : TX buffer overflow or underflow status. When txbufstatus[1] is set High, the signal 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.

Table 2-7: GTP Transceiver Debug Ports (Cont'd)

Name	Direction	Clock Domain	Width	Description
gt_rxbufstatus_out	Output	rxusrclk2	3 × Num. channels	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.
gt_txphaligndone_out	Output	Async	1 × Num. channels	Connects to TXPHALIGNDONE on transceiver channel primitives
gt_txphinitdone_out	Output	Async	1 × Num. channels	Connects to TXPHINITDONE on transceiver channel primitives
gt_txdlysresetdone_out	Output	Async	1 × Num. channels	Connects to TXDLYSRESETDONE on transceiver channel primitives
gt_rxphaligndone_out	Output	Async	1 × Num. channels	Connects to RXPHALIGNDONE of transceiver channel primitives
gt_rxdlysresetdone_out	Output	Async	1 × Num. channels	Connects to RXDLYSRESETDONE of transceiver channel primitives
gt_rxsyncdone_out	Output	Async	1 × Num. channels	Connects to RXSYNCDONE of transceiver channel primitives
pll0lock_out	Output	Async	1 × Num. channels	PLL0LOCK and PLL1LOCK of the 7 series FPGA GTP transceiver COMMON block. Available shared logic is in core.
pll1lock_out	Output	Async	1 × Num. channels	PLL0LOCK and PLL1LOCK of the 7 series FPGA GTP transceiver COMMON block. Available shared logic is in core.
Signal Integrity and Functionality				

Table 2-7: GTP Transceiver Debug Ports (Cont'd)

Name	Direction	Clock Domain	Width	Description
gt_eyescantrigger_in	Input	rxusrclk2	1 × Num. channels	Connects to EYESCANTRIGGER on transceiver channel primitives
gt_eyescanreset_in	Input	Async	1 × Num. channels	Connects to EYESCANRESET on transceiver channel primitives
gt_eyescandataerror_out	Output	Async	1 × Num. channels	Connects to EYESCANDATAERROR on transceiver channel primitives
gt_loopback_in	Input	Async	3 × Num. channels	Connects to LOOPBACK on transceiver channel primitives
gt_rxpolarity_in	Input	rxusrclk2	1 × Num. channels	Connects to RXPOLARITY on transceiver channel primitives
gt_txpolarity_in	Input	txusrclk2	1 × Num. channels	Connects to TXPOLARITY on transceiver channel primitives
gt_rxlpmreset_in	Input	Async	1 × Num. channels	Resets the LPM circuitry.
gt_rxlpmhfhold_in	Input	RXUSRCLK2	1 × Num. channels	When set to 1'b1, the current value of the high-frequency boost is held. When set to 1'b0, the high-frequency boost is adapted.
gt_rxlpmhfhold_in	Input	RXUSRCLK2	1 × Num. channels	When set to 1'b1, the high-frequency boost is controlled by the RXLPM_HF_CFG attribute. When set to 1'b0, the high-frequency boost is controlled by the rxlpmhfhold signal.
gt_rxlpmfhold_in	Input	Async	1 × Num. channels	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.

Table 2-7: GTP Transceiver Debug Ports (Cont'd)

Name	Direction	Clock Domain	Width	Description
gt_txprecursor_in	Input	Async	5 × Num. channels	Connects to TXPRECURSOR on transceiver channel primitives
gt_txpostcursor_in	Input	Async	5 × Num. channels	Connects to TXPOSTCURSOR on transceiver channel primitives
gt_txdiffctrl_in	Input	txusrclk2	4 × Num. channels	Connects to TXDIFFCTRL on transceiver channel primitives
gt_txprbsforce_err_in	Input	txusrclk2	1 × Num. channels	Connects to TXPRBSFORCEERR on transceiver channel primitives
gt_txprbsssel_in	Input	txusrclk	4 × Num. channels	Connects to TXPRBSSEL on transceiver channel primitives
gt_rxprbsssel_in	Input	rxusrclk2	4 × Num. channels	Connects to RXPRBSSEL on transceiver channel primitives
gt_rxprbserr_out	Output	rxusrclk2	1 × Num. channels	Connects to RXPRBSERR on transceiver channel primitives
gt_rxprbscnt_reset_in	Input	rxusrclk2	1 × Num. channels	Connects to RXPRBSCNTRESET on transceiver channel primitives
gt_rxcdrhold_in	Input	Async	1 × Num. channels	Connects to RXCDRHOLD on transceiver channel primitives
gt_dmonitor_out_out	Output	Async	1 × Num. channels	Connects to DMONITOROUT on transceiver channel primitives
gt_rxdisperr_out	Output	rxusrclk2	1 × Num. channels	Indicates the corresponding byte shown on rxdata has a disparity error. The rxdisperr pin of the transceiver is connected to this port.

Table 2-7: GTP Transceiver Debug Ports (Cont'd)

Name	Direction	Clock Domain	Width	Description
gt_rxnotintable_out	Output	rxusrclk2	1 × Num. channels	Indicates the corresponding byte shown on rxdata was not a valid character in the 8B/10B table. rxNotintable pin of the transceiver is connected to this port.
gt_rxcomma det_out	Output	rxusrclk2	1 × Num. channels	This signal is asserted when the comma alignment block detects a comma. The assertion occurs several cycles before the comma is available at the FPGA RX interface. 0: Comma not detected 1: Comma detected

Notes:

1. Each port is prefixed with gt<lane>_ where lane is from 0 to Num. channels -1.

Designing with the Core

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

General Design Guidelines

- Line rate selection should be compatible with the device and speed grade chosen in the project settings.
- The Start Up finite state machines (FSMs) given in the example design are mainly to demonstrate a proper reset initialization sequence. You need to fine-tune the counter settings in the FSMs per the design requirements.
- The design needs to be constrained per the constraints generated by the Wizard. You need to add additional constraints based on the board requirements.
- Every version of the Wizard is associated with a specific silicon version for GTX, GTH, GTP, and GTZ transceivers. Follow [AR 46048](#) to ensure the appropriate version is chosen.

Serial Transceiver Location

Based on the total number of serial transceivers selected, provide the specific location of each serial transceiver that you intend to use. The region shown in the panel indicates the location of serial transceivers in the tile. This demarcation of the region is based on the physical placement of serial transceivers with respect to the median of BUFGs available for each device. Refer to [Constraining the Core](#) for more information.

Use the Example Design as a Starting Point

Each instance of 7 series FPGA transceiver core that is created is delivered with an example design that can be simulated and implemented in FPGAs. This design can be used as a starting point for your own design or can be used to troubleshoot the user application, if necessary.

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 the user application should come from, or connect to a flip-flop. Registering signals might not be possible for all paths, but doing so 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 or VHDL source code. The example implementation scripts provided currently use the Vivado® synthesis tool for the example design that is delivered with the core. Other synthesis tools can also be used.

Make Only Allowed Modifications

The 7 series FPGA transceiver core is not user-modifiable. Any modifications might have adverse effects on the system timings and protocol compliance. Supported user configurations of the 7 series FPGA transceiver core can only be made by selecting options from the Vivado IDE.

Clocking

The clocks mentioned in [Table 3-1](#) are to be driven for proper operation of the example design.

Table 3-1: Clock Signals

Clock	Description
GTREFCLK_PAD_N_IN/ GTREFCLK_PAD_P_IN	External differential clock input pin pair for the reference clock of the 7 series FPGA transceiver Quad.
DRP_CLK_IN_P/ DRP_CLK_IN_N	External differential clock input pin pair for the DRP interface clock. This clock needs to be constrained in the XDC. See the 7 series data sheets for more information.
SYSCLK	Free running system/board clock is used to drive the FPGA logic in the example design. When the DRP interface is enabled, DRP_CLK is connected to SYSCLK in the example design. This clock needs to be constrained in the XDC.
PLLLOCKDETCLK	<p>Stable reference clock for the detection of the feedback and reference clock signals to the PLL. The input reference clock to the PLL or any output clock generated from the PLL (for example, TXOUTCLK) must not be used to drive this clock.</p> <p>This clock is required only when using the PLLFBCLKLOST and PLLREFCLKLOST ports. It does not affect the PLL lock detection, reset, and powerdown functions.</p> <p>PLL indicates the CPLL/QPLL for GTX/GTH transceivers and PLL0/PLL1 for GTP transceivers. In the example design, PLLLOCKDETCLK is connected to DRPCLK or SYSCLK. PLLREFCLKLOST is used in the TX/RX StartUp FSM modules.</p>

Generation of TX/RXUSRCLK and TX/RXUSRCLK2

The Wizard generates the TXUSRCLK/TXUSRCLK2 as follows:

- TXUSRCLK and TXUSRCLK2 are always generated using the TXOUTCLK, which is the output of the transceiver.
- As given in the *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476) [Ref 7], the source for the TXOUTCLK can be the reference clock or the PMA clock.

Table 3-2: TXOUTCLK Configuration Setting

TX Buffer Bypassed	Source for TXOUTCLK
Yes	Set to Reference Clock . You cannot change the selection in the Vivado IDE.
No	You can select Reference Clock or PMA Clock on Page 2 of the Vivado IDE.

The Wizard generates the RXUSRCLK/RXUSRCLK2 as follows:

- RXUSRCLK and RXUSRCLK2 can be generated using TXOUTCLK or RXOUTCLK, which are the outputs of the transceiver.
- As given in the *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG 476) [Ref 7], the source for the RXOUTCLK can be the reference clock or the recovered clock.

Table 3-3: RXOUTCLK Configuration Setting

RX Buffer Bypassed	Source for RXUSRCLK	Source for RXOUTCLK
Yes	Set to RXOUTCLK. You cannot change the selection in the Vivado IDE.	Set to Recovered Clock . You cannot change the selection in the Vivado IDE.
No	You can select TXOUTCLK or RXOUTCLK on Page 2 of the Vivado IDE.	You can select Reference Clock or Recovered Clock on Page 2 of the Vivado IDE.

Instantiation of MMCM/BUFG

The TX/RXOUTCLK are routed to the FPGA logic through a mixed-mode clock manager (MMCM) or BUFG. An MMCM is needed when the TXUSRCLK does not match TXUSRCLK2 and the RXUSRCLK does not match RXUSRCLK2. Table 3-4 and Table 3-5 capture various scenarios to instantiate an MMCM.

Table 3-4: MMCM for TX Buffer Bypass

TX External Data Width	TX Internal Data Width	TX Buffer Bypassed	Source for TXOUTCLK	TXUSRCLK2	MMCM
16	16/20	No	TXOUTCLKPMA	TXUSRCLK	No
32	16/20	No	TXOUTCLKPMA	TXUSRCLK/2	Yes
32	32/40	No	TXOUTCLKPMA	TXUSRCLK	No

Table 3-4: MMCM for TX Buffer Bypass (Cont'd)

TX External Data Width	TX Internal Data Width	TX Buffer Bypassed	Source for TXOUTCLK	TXUSRCLK2	MMCM
64	32/40	No	TXOUTCLKPMA	TXUSRCLK/2	Yes
20	20	No	TXOUTCLKPMA	TXUSRCLK	No
40	20	No	TXOUTCLKPMA	TXUSRCLK/2	Yes
40	40	No	TXOUTCLKPMA	TXUSRCLK	No
80	40	No	TXOUTCLKPMA	TXUSRCLK/2	Yes
16	16/20	Yes/No	Ref Clk If Refclk = TXUSRCLK2 then Wizard chooses TXPLLREFCLK_DIV1 as TXOUTCLK Else Wizard chooses TXPLLREFCLK_DIV2 As TXOUTCLK You should choose the appropriate REFCLK to meet the preceding criteria.	TXUSRCLK	No
32	16/20	Yes/No		TXUSRCLK/2	Yes
32	32/40	Yes/No		TXUSRCLK	No
64	32/40	Yes/No		TXUSRCLK/2	Yes
20	20	Yes/No		TXUSRCLK	No
40	20	Yes/No		TXUSRCLK/2	Yes
40	40	Yes/No		TXUSRCLK	No
80	40	Yes/No		TXUSRCLK/2	Yes

Table 3-5: MMCM for RX Buffer Bypass

RX External Data width	RX Internal Data Width	RX Buffer Bypassed	Source for RXOUTCLK	RXUSRCLK2	MMCM
16	16/20	No	RXOUTCLKPMA/TXOUTCLK/REFCLK	RXUSRCLK	No
32	16/20	No	RXOUTCLKPMA/TXOUTCLK/REFCLK	RXUSRCLK/2	Yes
32	32/40	No	RXOUTCLKPMA/TXOUTCLK/REFCLK	RXUSRCLK	No
64	32/40	No	RXOUTCLKPMA/TXOUTCLK/REFCLK	RXUSRCLK/2	Yes
20	20	No	RXOUTCLKPMA/TXOUTCLK/REFCLK	RXUSRCLK	No
40	20	No	RXOUTCLKPMA/TXOUTCLK/REFCLK	RXUSRCLK/2	Yes
40	40	No	RXOUTCLKPMA/TXOUTCLK/REFCLK	RXUSRCLK	No
80	40	No	RXOUTCLKPMA/TXOUTCLK/REFCLK	RXUSRCLK/2	Yes

Table 3-5: MMCM for RX Buffer Bypass (Cont'd)

RX External Data width	RX Internal Data Width	RX Buffer Bypassed	Source for RXOUTCLK	RXUSRCLK2	MMCM
16	16/20	Yes	RXOUTCLKPMA	RXUSRCLK	No
32	16/20	Yes		RXUSRCLK/2	Yes
32	32/40	Yes		RXUSRCLK	No
64	32/40	Yes		RXUSRCLK/2	Yes
20	20	Yes		RXUSRCLK	No
40	20	Yes		RXUSRCLK/2	Yes
40	40	Yes		RXUSRCLK	No
80	40	Yes		RXUSRCLK/2	Yes

Resets

Reset Finite State Machine

The intent for the reset FSM included in the example design is to provide:

- An application example of an initialization FSM meeting the requirements described in the *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476) [Ref 7].
- An example of an initialization FSM that addresses industry challenges to initialize the serial transceivers in scenarios such as post FPGA-configuration and RX data interruption or replacement (such as cable plug or unplug).

The reset FSM is constantly being improved to provide a robust initialization and reset scheme. The FSM is to demonstrate the right methodology and should not be mistaken as a specification.

The tx_startup_fsm is illustrated on the left side of [Figure 3-1](#).

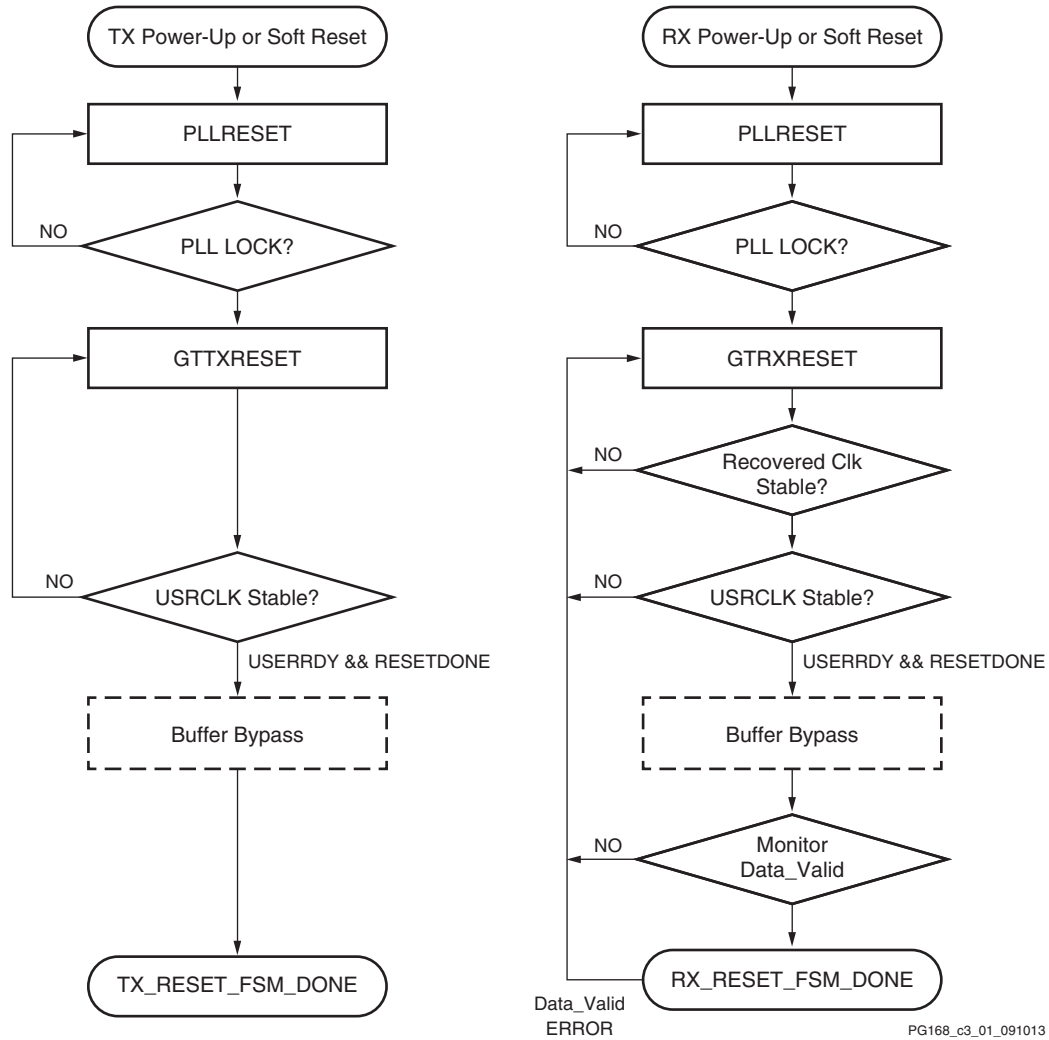


Figure 3-1: Diagram of Simplified FSM

The CPLL/QPLL lock is monitored along with the TXUSRCLK stability prior to TXRESETDONE. Buffer bypass logic for phase alignment is implemented if the TX buffer is disabled.

The rx_startup_fsm is illustrated on the right side of Figure 3-1. The C/QPLL lock, recovered clock stability, and RXUSRCLK are examined prior to RXRESETDONE followed by the buffer bypass logic for phase alignment. The FSM stays at the state that monitors data validity, which can be an 8B/10B error, frame sync error, or CRC from the user design until a user-defined error occurs.

These are some assumptions and notes for the Example Reset FSM:

- A stable REFCLK is assumed to be present at all times.
- All resets are assumed to be in sequential mode. The reset FSMs run on SYSCLK, which is the same as the DRPCLK. If the SYSCLK and DRPCLK are not the same in the user design, care should be taken to add the appropriate synchronizers.

- The example design engaged additional gates available such as PLLREFCLKLOST and Wait-time. Wait-time in use should not be regarded as the specification.
- RECCLK_STABLE is used as an indicator of RXOUTCLK (recovered clock) stability within a configured PPM offset from the reference clock (default 5,000 ppm). The appropriate T_{DLOCK} is used (see the *Virtex-7 FPGAs Data Sheet: DC and Switching Characteristics* (DS183) [Ref 1]).
- The example design defaults FRAME_CHECKER as data_valid. You can identify your data valid indicator and provide the necessary hysteresis on this signal to avoid a false indication of data_valid. Data valid is only used as an indicator to monitor the RX link. You need to customize the data valid indicator based on your system design.
- You have the liberty to modify or re-invent an FSM to meet your specific system requirements while adhering to the guidelines in the *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476) [Ref 7].

There might be other restrictions in Answer Records or Errata.

Reset Sequence Modules for GTH and GTP Transceivers

GTH and GTP transceivers need additional reset sequencing for production silicon. The reset sequencers for GTRXRESET, RXPMARESET, and RXRATE are instantiated in the `<component_name>_gt.v[hd]` module. These modules need exclusive access to the DRP interface of the transceiver. The DRP_BUSY_OUT signal that is synchronous to the DRPCLK indicates that you cannot access the DRP interface. You should implement a DRP arbitration scheme to ensure smooth operation of the various reset sequencers. In the current design, the GTRXRESET sequencer takes highest priority.

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 and 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) [\[Ref 3\]](#)
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [\[Ref 9\]](#)
- *Vivado Design Suite User Guide: Getting Started* (UG910) [\[Ref 10\]](#)
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [\[Ref 4\]](#)

Customizing and Generating the Core

This section includes information about using Xilinx tools to customize and generate the core in the Vivado Design Suite.

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) [\[Ref 3\]](#) for detailed information. 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.

The 7 series FPGA transceiver 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.

The 7 series FPGA transceiver core can be found in **FPGA Features and Design > IO Interface** in the Vivado IP Catalog.

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

1. Select the IP from the IP catalog.

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 9] and the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 10].

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

Figure 4-1 shows the 7 series FPGA transceiver Wizard Customize IP dialog boxes with customizing information.

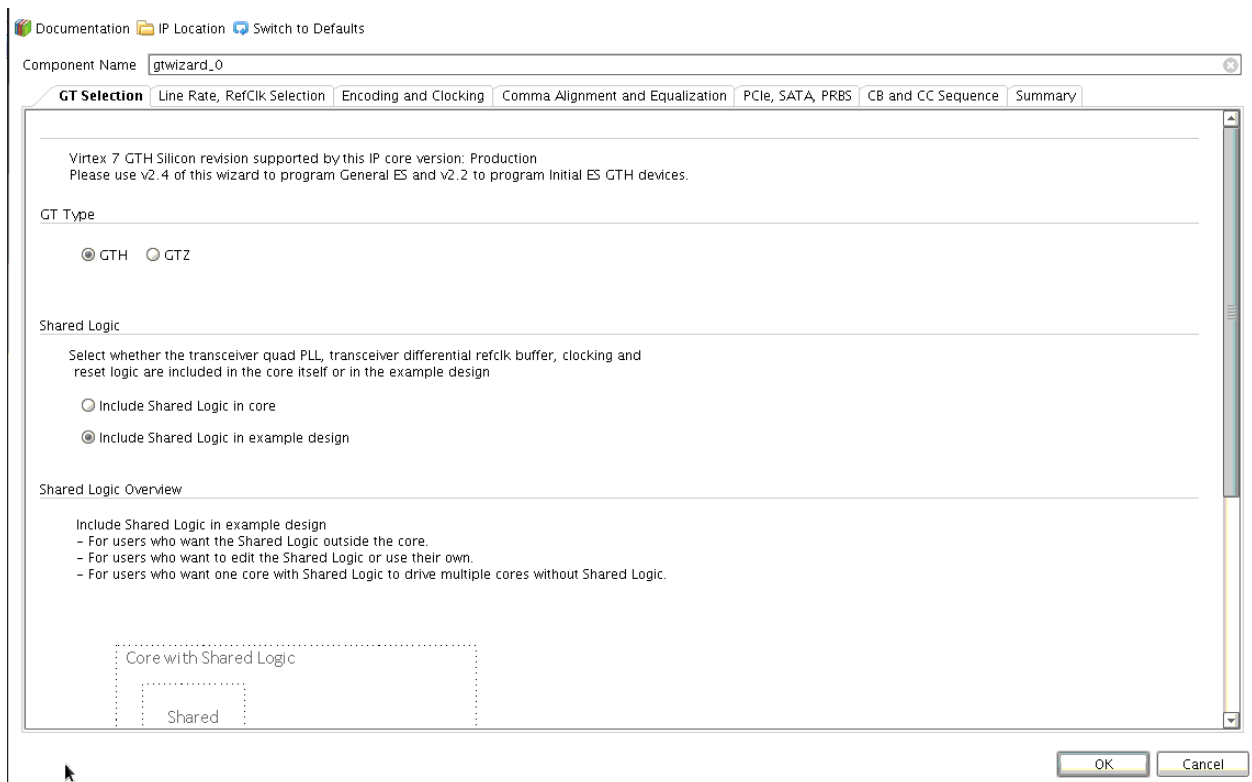


Figure 4-1: 7 Series FPGAs Transceiver Wizard

Component Name

The Component Name field can consist of any combination of alphanumeric characters including the underscore symbol. 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 uses this name.

Example Design—XAUI Configuration

The example design covered in this section is a wrapper that configures a group of transceivers for use in a XAUI application. Guidelines are also given for incorporating the wrapper in a design and for the expected behavior in operation. For detailed information, see [Chapter 5, Example Design](#).

The XAUI example design consists of these components:

- A single transceiver wrapper implementing a four-lane XAUI port using four transceivers.
- A demonstration test bench to drive the example design in simulation.
- An example design providing clock signals and connecting an instance of the XAUI wrapper with modules to drive and monitor the wrapper in hardware, including optional Vivado Design Suite debug feature tools support.
- Scripts to synthesize and simulate the example design.

The Wizard example design has been tested with Vivado Design Suite 2014.3 for synthesis and QuestaSim 10.2a for simulation.

[Figure 4-2](#) shows a block diagram of the default XAUI example design.

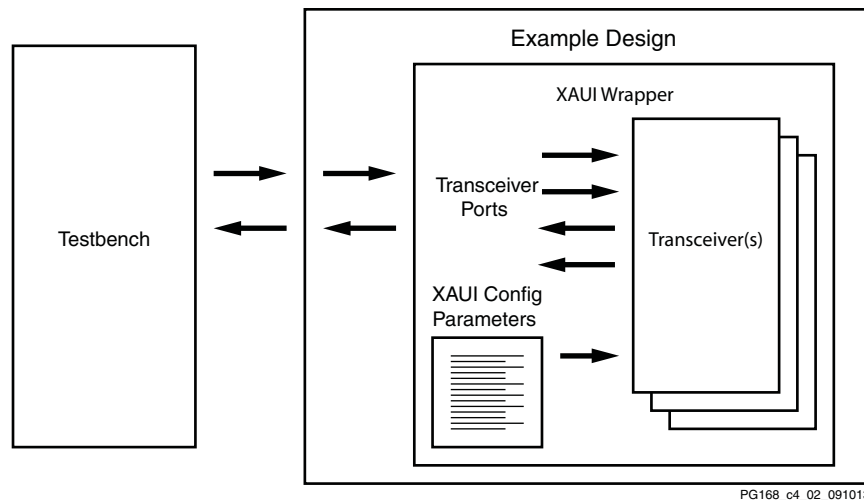


Figure 4-2: Example Design and Test Bench—XAUI Configuration

Setting the Project Options

Set the project options using these steps:

1. Click **Part** in the option tree.
2. Select **Virtex7** from the Family list.

3. Select a device from the Device list that supports transceivers.
4. Select an appropriate package from the Package list. This example uses the XC7V1500T device (see Figure 4-3 or Figure 4-4). For an example design using GTZ transceivers, select the XC7VH580T device.

Note: If an unsupported silicon family is selected, the Wizard remains light gray in the taxonomy tree and cannot be customized. Only devices containing 7 series FPGAs transceivers are supported by the Wizard. See *7 Series FPGAs Overview (DS180)* [Ref 11] for a list of devices containing the 7 series FPGAs transceivers.

5. Click **Generation** in the option tree and select either Verilog or VHDL as the output language. For GTZ transceivers, select only Verilog. VHDL is not supported.
6. Click **OK**.

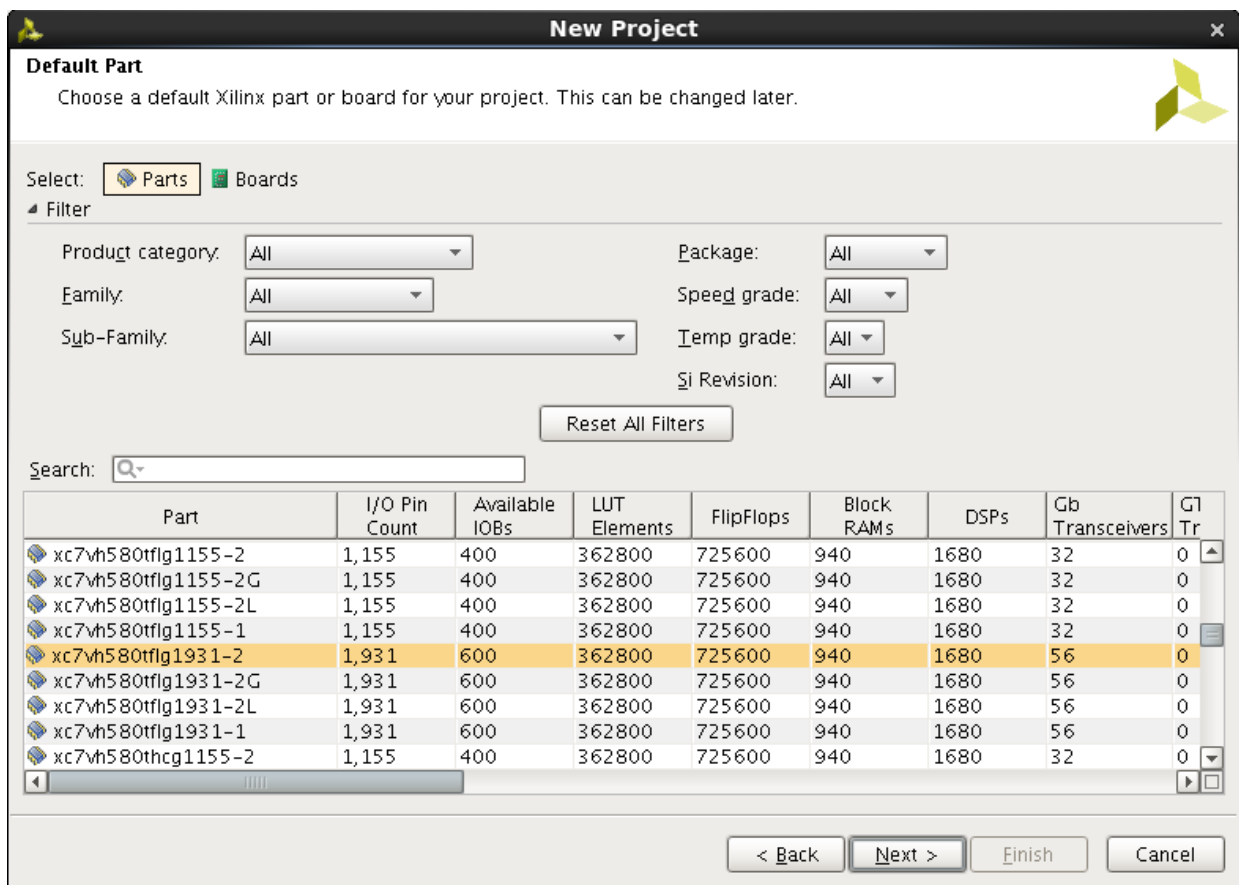


Figure 4-3: Target Architecture Setting

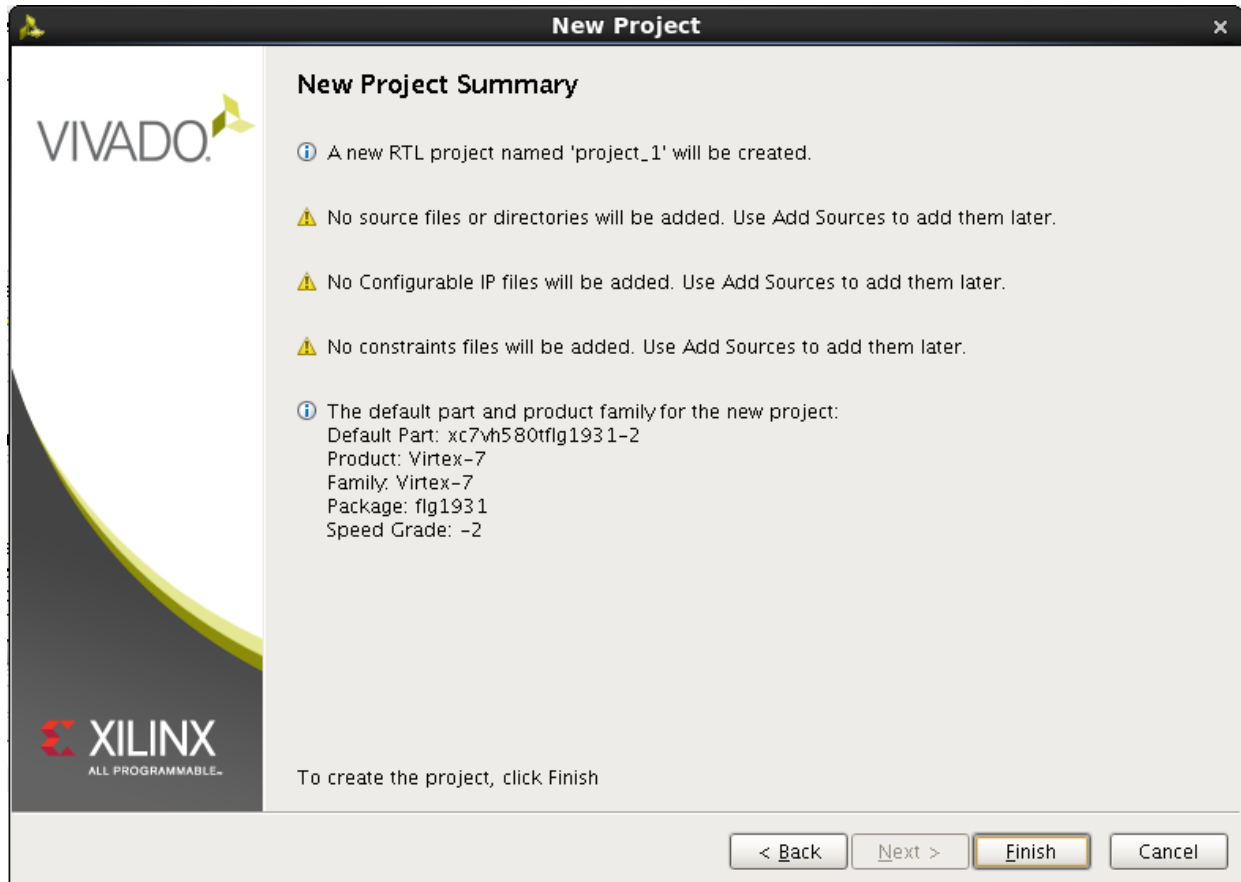


Figure 4-4: Target Architecture Setting (Vivado Design Suite)

Configuring and Generating the Wrapper

This section provides instructions for generating an example transceiver wrapper using the default values. The wrapper, associated example design, and supporting files are generated in the project directory. For additional details about the example design files and directories, see [Chapter 5, Example Design](#).

1. Open a project by selecting **File > Open Project** or create a new project by selecting **File > New Project** in the Vivado Design Suite.
2. Open the IP catalog and select **FPGA Features and Design > IO Interfaces** in the View by Function pane.
3. Double-click **7 Series FPGAs Transceiver Wizard** to bring up the 7 series FPGA Transceiver Customize IP dialog box.

GTZ Transceivers

Octal Selection, Channel Selection, Line Rate, and REFCLK

Page 1 of the Wizard (Figure 4-5) allows you to select the component name and determine the line rate and reference clock frequency. In addition, this page specifies a protocol template.

1. In the Component Name field, enter a name for the Wizard instance. This example uses the name `caui4_wrapper`.
2. In the GT_Type field, select **GTZ**. The type of transceiver depends on the device chosen in Project Options.
3. Select one or both of the octals GTZE2_OCTAL0 or GTZE2_OCTAL1. The number of available octals depends on the target device and package.

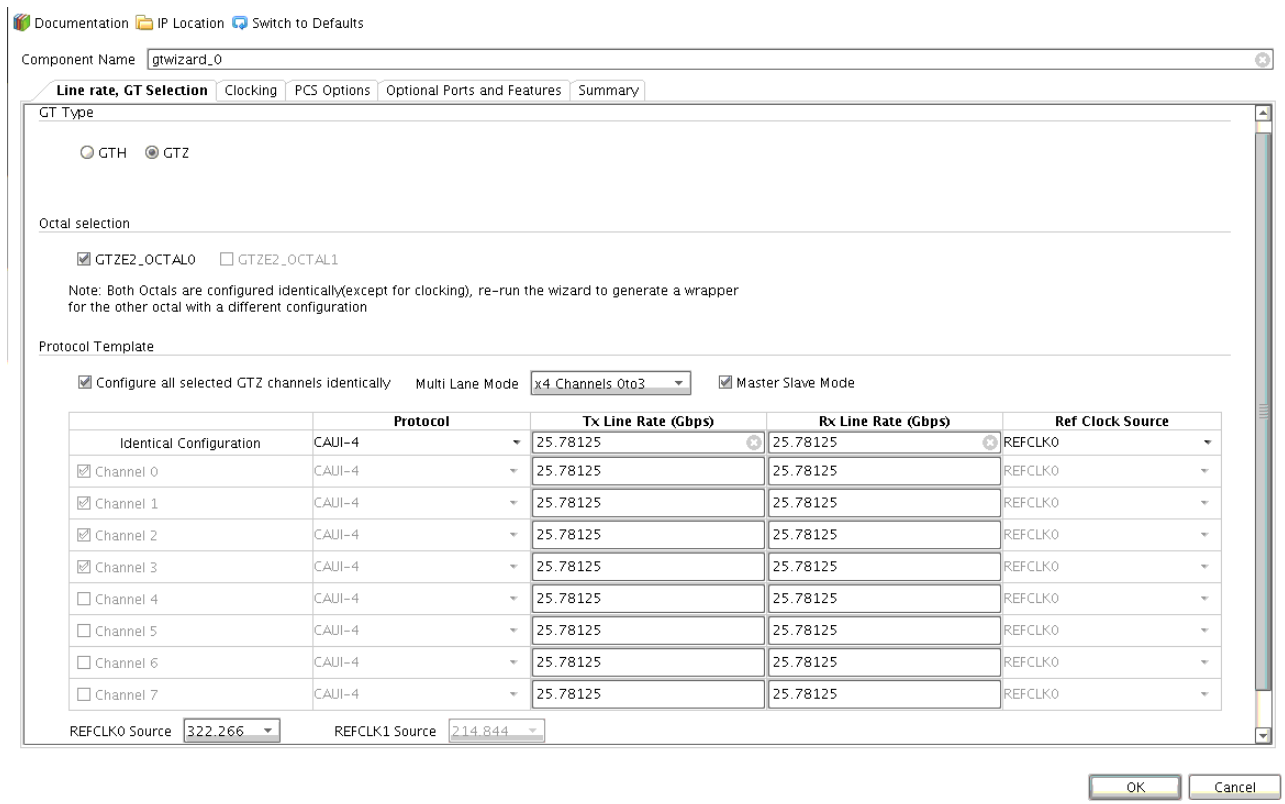


Figure 4-5: Line Rates and Transceiver Selection—GTZ Page 1

4. Select the multichannel mode if you wish to set the octal in one of the multichannel modes. Selecting a mode here will automatically select the channels appropriately also. If using one of the multichannel modes, use the Master Slave mode to enable one of the channels as the master and the other as slaves.

5. The GTZ wizard supports both identical configuration and non-identical configuration of channels within an octal. To configure identically, select the **Configure all selected GTZ channels identically** checkbox.
6. Select **Start from scratch** if you wish to manually set all parameters. Select one of the available protocols from the list to begin designing with a predefined protocol template. The CAUI4 example uses the CAUI 4 protocol template. The CAUI 4 protocol template uses four channels.
7. Use [Table 4-1](#) to determine the line rate and reference clock settings. The line rate allowed is dependent on the speed grade of the device:
 - For -2G speed grade: 9.8 Gb/s – 14.025 Gb/s and 19.6 Gb/s – 28.05 Gb/s
 - For other speed grades: 9.8 Gb/s – 12.890625 Gb/s and 19.6 Gb/s – 25.78125 Gb/s

Table 4-1: Line Rate and REFCLK Settings—GTZ Transceivers

Options	Description
Line Rate	Set to the desired target line rate in Gb/s. Should be the same for both TX and RX. The CAUI4 example uses 25.78125 Gb/s.
Reference Clock Source	Select either REFCLK0 or REFCLK1. The CAUI4 example uses REFCLK0.
REFCLK0 Source	Select from the list of the optimal reference clock frequency to be provided by the application. The CAUI4 example uses 322.266 MHz.
REFCLK1 Source	Select from the list of the optimal reference clock frequency to be provided by the application. The CAUI4 example uses 322.266 MHz.

Clocking

The **Clocking** tab of the Wizard ([Figure 4-6](#)) allows you to select the clocking for the octal and the enabled channels within an octal. You can verify all the settings and selections made on this tab graphically with the image shown below the settings.

1. Select the source for TXOUTCLK0 and TXOUTCLK1 of the octal as per [Table 4-2](#).
2. Select the source for RXOUTCLK<0-3> of the octal as per [Table 4-2](#).

Table 4-2: TX/RXOUTCLKs of the Octal

Options	Description
TXOUTCLK<0-1>	These can be sourced from the TXOUTCLKs from any of the individual channels (only channels enabled on page 1). This selection applies identically for all the octals enabled on page 1.
RXOUTCLK<0-3>	These can be sourced from the RXOUTCLKs from any of the individual channels that were selected on page 1). This selection applies identically for all the octals on page 1.

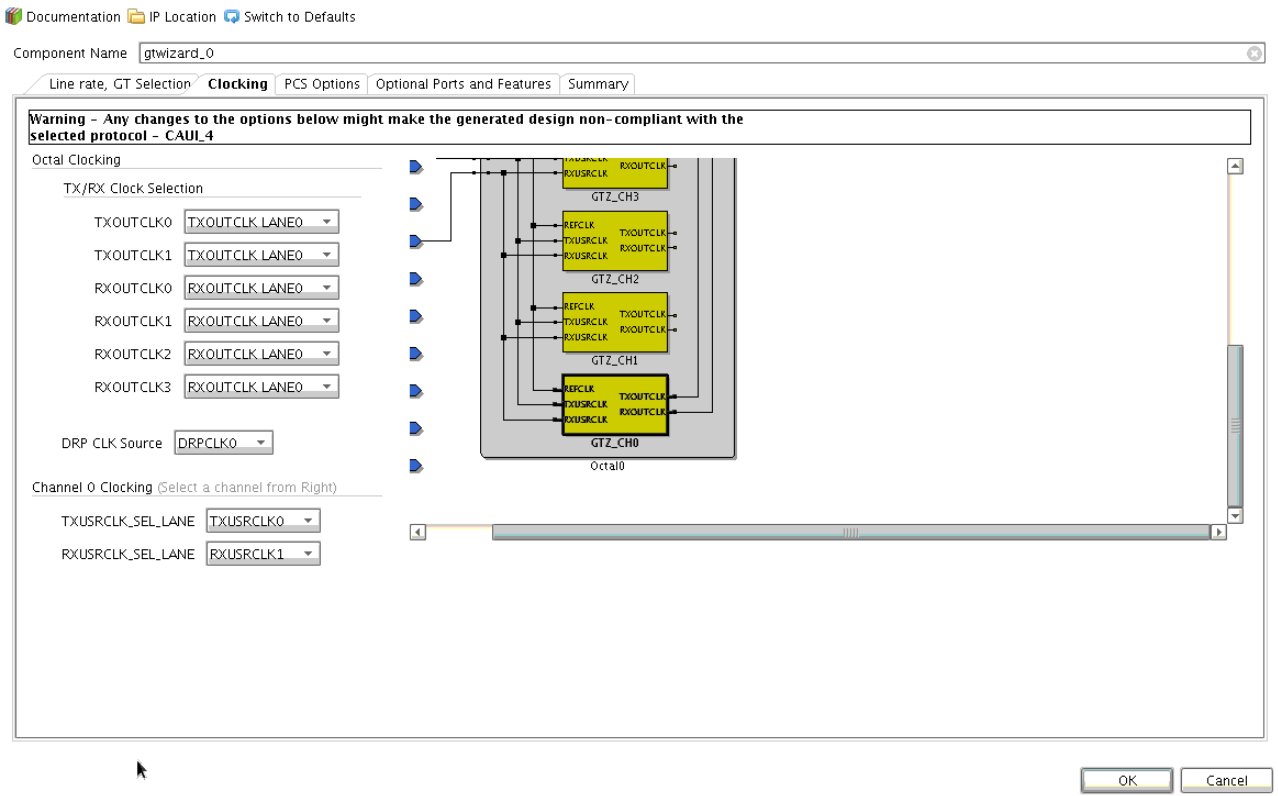


Figure 4-6: Octal and Channel Clocking—GTZ Page 2

3. Select the source for the user clocks TX/RXUSRCLK<0-7> (Table 4-3). The USRCLK numbering shown here in the Vivado IDE is relative to OCTAL0. These USRCLKs are mapped internally (by the Wizard) to octal1 as follows:

Table 4-3: TX/RXUSRCLKs of the Octal

Options	Description
TXUSRCLK<0-7>	The available sources will be the TXOUTCLKs of the active octals selected in page 1.
RXUSRCLK<0-7>	The available sources will be the RXOUTCLKs of the active octals selected in page 1.

- TX/RXUSRCLK0 – TX/RXUSRCLK4
- TX/RXUSRCLK1 – TX/RXUSRCLK5
- TX/RXUSRCLK2 – TX/RXUSRCLK6
- TX/RXUSRCLK3 – TX/RXUSRCLK7
- TX/RXUSRCLK4 – TX/RXUSRCLK0
- TX/RXUSRCLK5 – TX/RXUSRCLK1
- TX/RXUSRCLK6 – TX/RXUSRCLK2
- TX/RXUSRCLK7 – TX/RXUSRCLK3

4. Channel clocking: First select the channel number for which you wish to configure the clocking.
5. In the TX/RXOUTCLK source for the channel selected above, the only supported sources are TX/RX FIFO CLKs.
6. In the TX/RXUSRCLK LANE sel, select the USRCLK that you wish to source for this channel. For channels of octal1, the USRCLK selection made here will be automatically mapped by the wizard as shown in [step 3](#).
7. Select the source for the DRPCLK.

Table 4-4: Channel Clocking

Options	Description
TXOUTCLK source	Select the TXOUTCLK source for each channel. Only the TX FIFO CLK is supported. Selecting the TX FIFO CLK selects TXOUTCLKPMA_DIV4 (see the <i>7 Series FPGAs GTZ Transceivers User Guide</i> (UG478) [Ref 12] for more information).
RXOUTCLK source	Select the RXOUTCLK source for each channel. Only the RX FIFO CLK is supported. Selecting the RX FIFO CLK selects RXOUTCLKPMA_DIV4 (see the <i>7 Series FPGAs GTZ Transceivers User Guide</i> (UG478) [Ref 12] for more information).
TXUSRCLK lane sel	Select one among the available eight TX user clocks.
RXUSRCLK lane sel	Select one among the available eight RX user clocks.

PCS Modes

The third page of the Wizard ([Figure 4-7](#)) allows you to select the data width and PCS mode options.

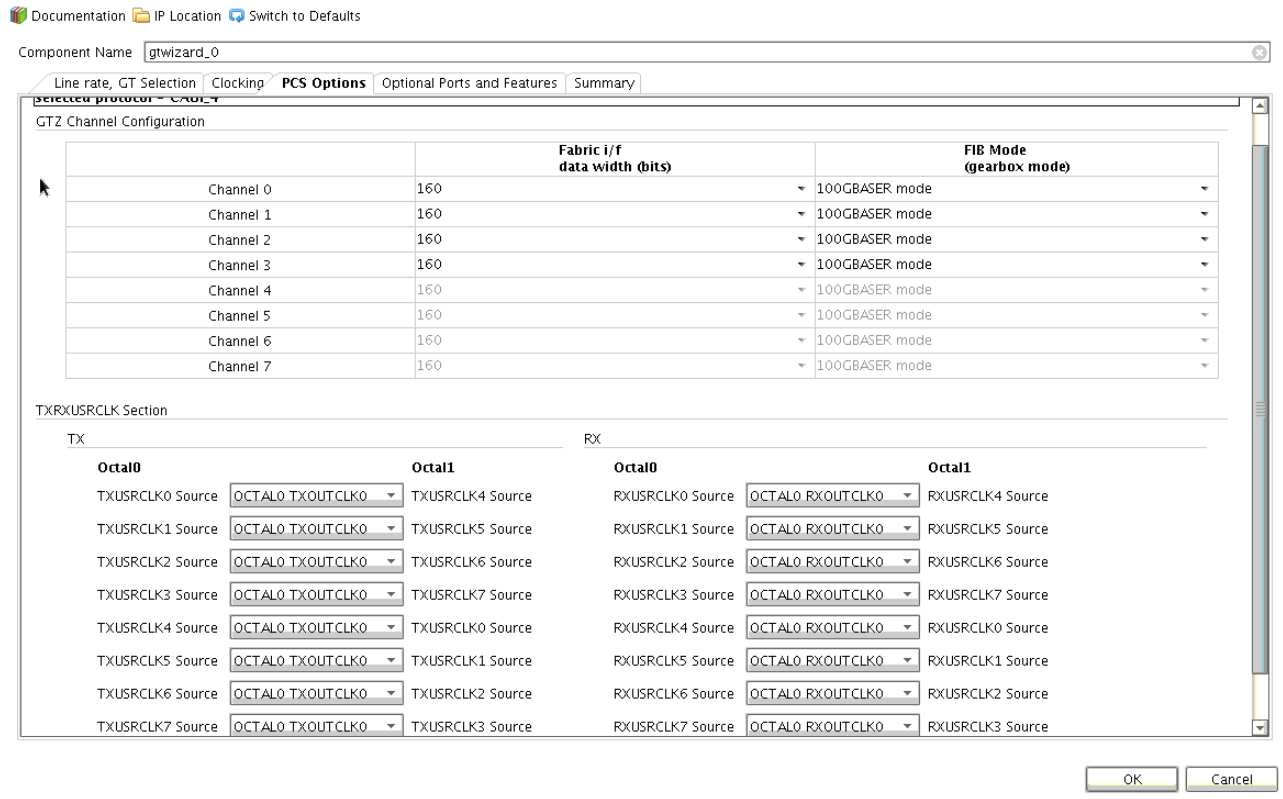


Figure 4-7: PCS Options—GTZ Page 3

1. Select the data width for each of the channels enabled on **Line Rate, GT Selection** tab. The options shown here are dependent on the line rates entered on page 1. For 28.05, 27.95255, and 25.78125 Gb/s, only 160-bit mode is applicable. For 14.025 and 10.3125 Gb/s, both 160- and 80-bit modes are applicable. The example for the CAUI4 template uses a 160-bit width.
2. Select the FIB mode options here (Table 4-5). This is dependent on the data width selected above. For a data width of 160 bits, only 100GBSAER mode is applicable. For 80 bits, both 100GBASER and 64B/66B modes are applicable. The CAUI4 protocol uses 100GBASER here.

Table 4-5: FIB Options

Options	Description
Fabric i/f data width	Select the interface data width.
FIB mode (gearbox mode)	Select the gearbox mode.

Optional Ports

Page 4 of the Wizard (Figure 4-8) allows you to select the optional ports to bring out to the example top and multi-transceiver wrapper.

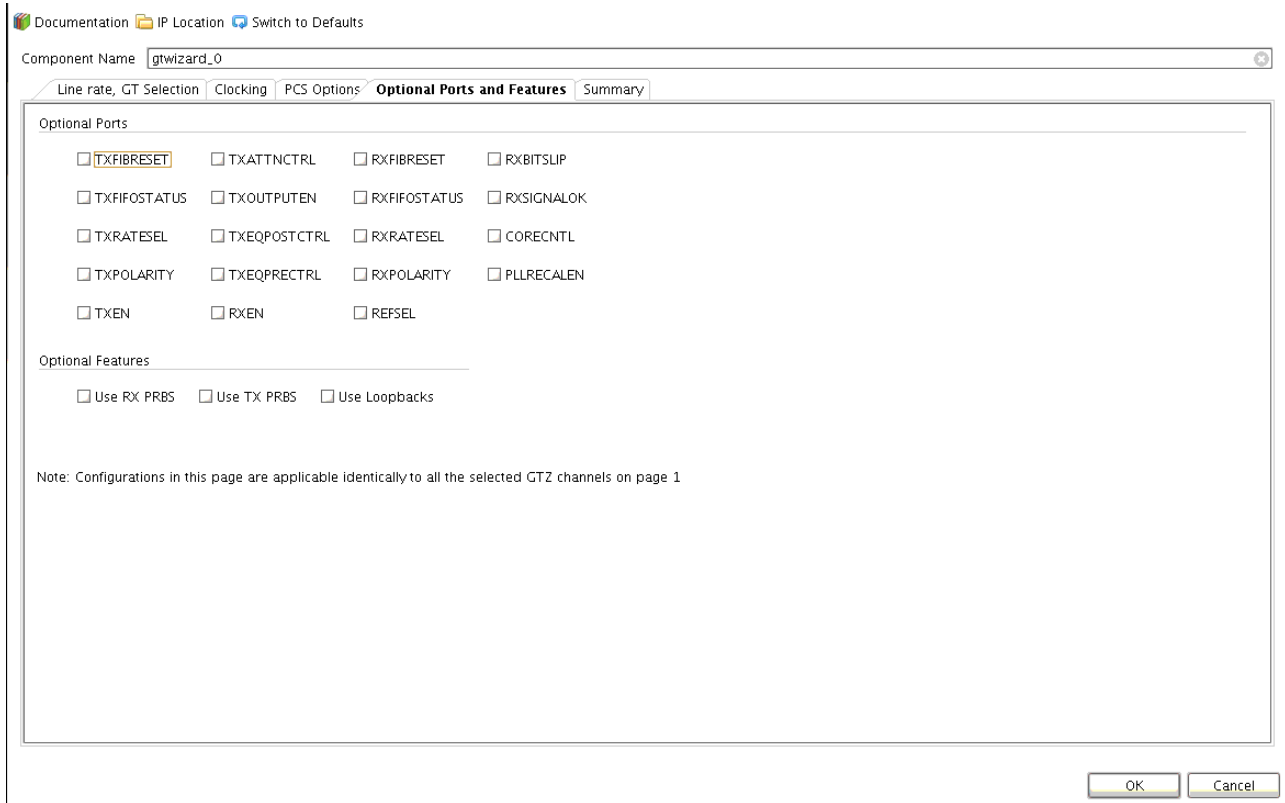


Figure 4-8: Optional Ports—GTZ Page 4

Table 4-6: Optional Ports for GTZ Transceivers

Options	Description
TXFIBRESET	Brings out the TXFIBRESET ports to the example design from which you can control the RESET of the FIB portion of the GTZ transceiver.
RXFIBRESET	Brings out the RXFIBRESET ports to the example design from which you can control the RESET of the FIB portion of the GTZ transceiver.
TXFIFOSTATUS	Brings out the TXFIFOSTATUS port to the example design allowing you to learn the FIFO status.
RXFIFOSTATUS	Brings out the RXFIFOSTATUS port to the example design allowing you to learn the FIFO status.
TXRATESEL	Brings the TXRATESEL ports out onto the example top level. These ports are used to control the TX PLL divider ratios.
RXRATESEL	Brings the RXRATESEL ports out onto the example top level. These ports are used to control the RX PLL divider ratios.
TXPOLARITY	Brings out the TXPOLARITY port to the example design.
RXPOLARITY	Brings out the RXPOLARITY port to the example design.
TXEN	Brings out the TXEN port to the example design.
RXEN	Brings out the RXEN port to the example design.

Table 4-6: Optional Ports for GTZ Transceivers

Options	Description
TXOUTPUTEN	Brings out the TXOUTPUTEN port to the example design.
TXATTNCTRL	Brings out the TXATTNCTRL port to the example design.
TXEQPOSTCTRL	Brings out the TXEQPOSTCTRL port to the example design.
TXEQPRECTRL	Brings out the TXEQPRECTRL port to the example design.
TXSLEWCTRL	Brings out the TXSLEWCTRL port to the example design.
RXBITSLIP	Brings out the RXBITSLIP port onto the example design. This port can be used to slip data in raw mode.
RXSIGNALOK	Brings out the RXSIGNALOK port onto the example design.
CORECNTL	Brings out the CORECNTL ports onto the example design.
REFSEL	Brings out the REFSEL ports onto the example design.
PLLRECALEN	Brings out the PLLRECALEN ports onto the example design.
RXPRBS	Brings out all the RXPRBS related ports onto the example design.
TXPRBS	Brings out all the TXPRBS related ports onto the example design.
LOOPBACK	Brings out all the LOOPBACK control ports onto the example design.

Summary

Page 5 of the Wizard (Figure 4-9) provides a summary of the selected configuration parameters. After reviewing the settings, click **Generate** to exit and generate the wrapper.

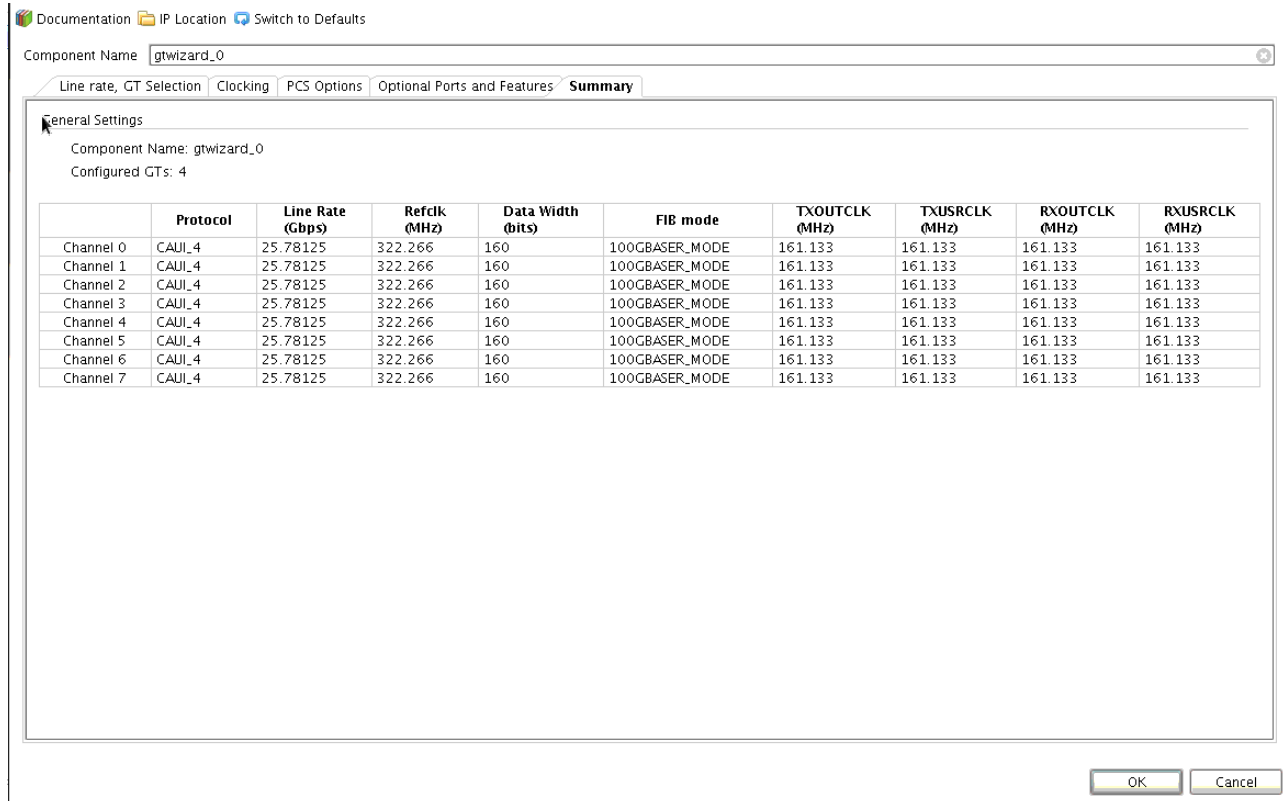


Figure 4-9: Summary Page—GTZ Page 5

GTX, GTH, and GTP Transceivers

GT Type Selection

Page 1 of the Wizard (Figure 4-10) allows you to select the component name and determine the line rate and reference clock frequency. In addition, this page specifies the option to include shared logic in the core or in the example design.

1. In the Component Name field, enter a name for the Wizard instance. This example uses the name **xaui_wrapper**.
2. In the GT_Type field, select **GTX**, **GTH**, or **GTP**. The type of transceiver depends on the device chosen in Project Options.
3. Including the shared logic option helps Wizard users to choose the shared logic resource at the core level or in the example design.

Select one of the available protocols from the list to begin designing with a predefined protocol template. The XAUI example uses the XAUI protocol template.

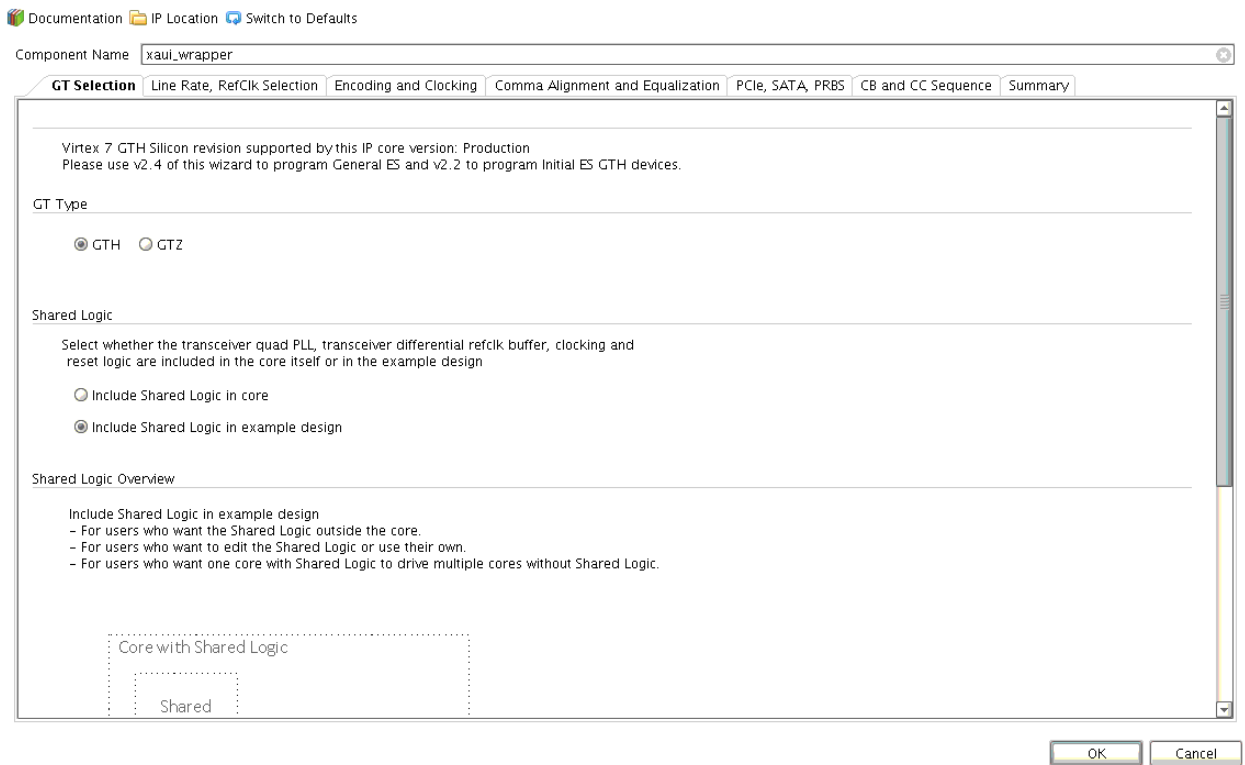


Figure 4-10: GT Transceiver Selection—Page 1

Line Rate, Transceiver Selection, and Clocking

Page 2 of the Wizard (Figure 4-11) allows you to select the transceiver location and clocking. The number of available transceivers appearing on this page depends on the selected target device and package. The XAUI example design uses four transceivers.

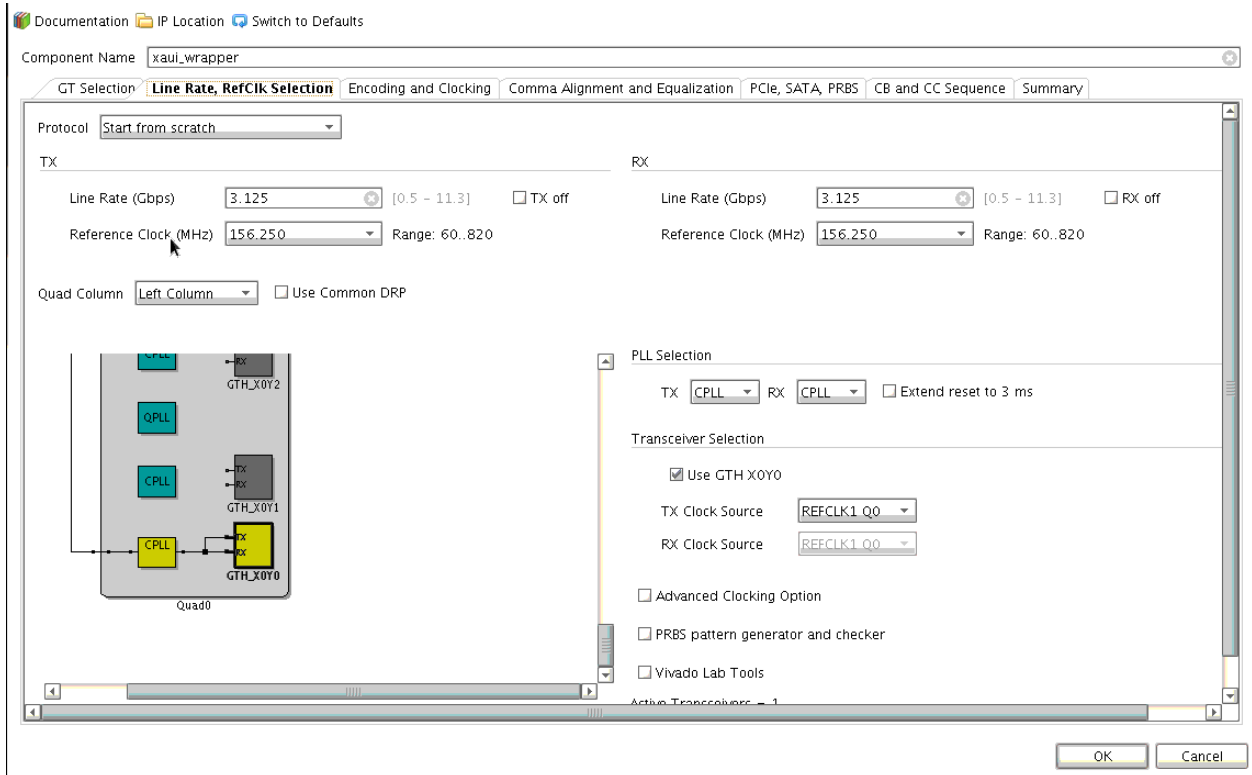


Figure 4-11: Transceiver Selection and Clocking—Page 2

Use Table 4-7 and Table 4-8 to determine the line rate and reference clock settings.

Table 4-7: TX Settings

Options	Description
Line Rate	Set to the desired target line rate in Gb/s. Can be independent of the receive line rate. The XAUI example uses 3.125 Gb/s.
Reference Clock	Select from the list the optimal reference clock frequency to be provided by the application. The XAUI example uses 156.25 MHz.
TX off	Selecting this option disables the TX path of the transceiver. The transceiver will act as a receiver only. The XAUI example design requires both TX and RX functionality.

Note: Options not used by the XAUI example are shaded.

Table 4-8: RX Settings

Options	Description
Line Rate	Set to the desired target line rate in Gb/s. The XAUI example uses 3.125 Gb/s.
Reference Clock	Select from the list the optimal reference clock frequency to be provided by the application. The XAUI example uses 156.25 MHz.

Table 4-8: RX Settings (Cont'd)

Options	Description
RX off	Selecting this option disables the RX path of the transceiver. The transceiver will act as a transmitter only. The XAUI example design requires both TX and RX functionality.

Note: Options not used by the XAUI example are shaded.

Use [Tables 4-9 through 4-12](#) to determine the optional ports settings available on this page.

Table 4-9: Additional Options

Option	Description
Use Common DRP	Select this option to have the dynamic reconfiguration port signals of the COMMON block available to the application.
Advanced Clocking Option	Use this check box to bring out all possible reference clock ports to the generated wrapper. Used for dynamic clock switching.
Vivado Design Suite Debug Feature	Use this check box to bring out the ILA and VIO cores in IP for hardware debugging and control. For more details on debugging using the ILA and VIO cores in hardware, refer to 7 Series GT Wizard Hardware Validation on the KC705 Evaluation Board, page 133 .

Table 4-10: Select the Transceiver and the Reference Clocks

Option	Description
GT	Select the individual transceivers by location to be used in the target design. The XAUI example requires four transceivers.
TX Clock Source	Determines the source for the reference clock signal provided to each selected transceiver (see Table 4-12). Two differential clock signal input pin pairs, labeled REFCLK0 and REFCLK1 are provided for every four transceivers. The groups are labeled Q0 through Q4 starting at the bottom of the transceiver column. Each transceiver has access to the local signal group and one or two adjacent groups depending upon the transceiver position. The XAUI example uses the REFCLK0 signal from the group local to the four selected transceivers (REFCLK0 Q0 option).
RX Clock Source	

Table 4-11: PLL Selection

Option	Description
QPLL	GTX and GTH transceivers: Use the Quad PLL when all four transceivers of the Quad are used to save power. Quad PLL is shared across four transceivers of a Quad.
CPLL	GTX and GTH transceivers: Use the Channel PLL based on the line rate supported by the selected transceiver.
PLL0	GTP transceivers only: PLL0 is shared across four transceivers of a Quad.
PLL1	GTP transceivers only: PLL1 is shared across four transceivers of a Quad.

Table 4-12: Reference Clock Source Options

Option	Description
REFCLK0/1 Q0	Reference clock local to transceivers Y0-Y3.
REFCLK0/1 Q1	Reference clock local to transceivers Y4-Y7.
REFCLK0/1 Q2	Reference clock local to transceivers Y8-Y11.
REFCLK0/1 Q3	Reference clock local to transceivers Y12-Y15.
REFCLK0/1 Q4	Reference clock local to transceivers Y16-Y19.
REFCLK0/1 Q5	Reference clock local to transceivers Y20-Y23.
REFCLK0/1 Q6	Reference clock local to transceivers Y24-Y27.
REFCLK0/1 Q7	Reference clock local to transceivers Y28-Y31.
REFCLK0/1 Q8	Reference clock local to transceivers Y32-Y35.

CPLL/QPLL/PLL0/PLL1REFCLKSEL has been always tied to 001 for REFCLK0, and 010 for REFCLK1 irrespective of whether the clock is from the North or South Quad. If you plan to use a reference clock from either North or South Quad, you must following the guidelines for changing the reference clock, which is found in the respective transceiver user guides for the device.

Encoding and Optional Ports

Page 3 of the Wizard ([Figure 4-12](#)) allows you to select encoding and 8B/10B optional ports. [Tables 4-13](#) through [4-20](#) list the available options.

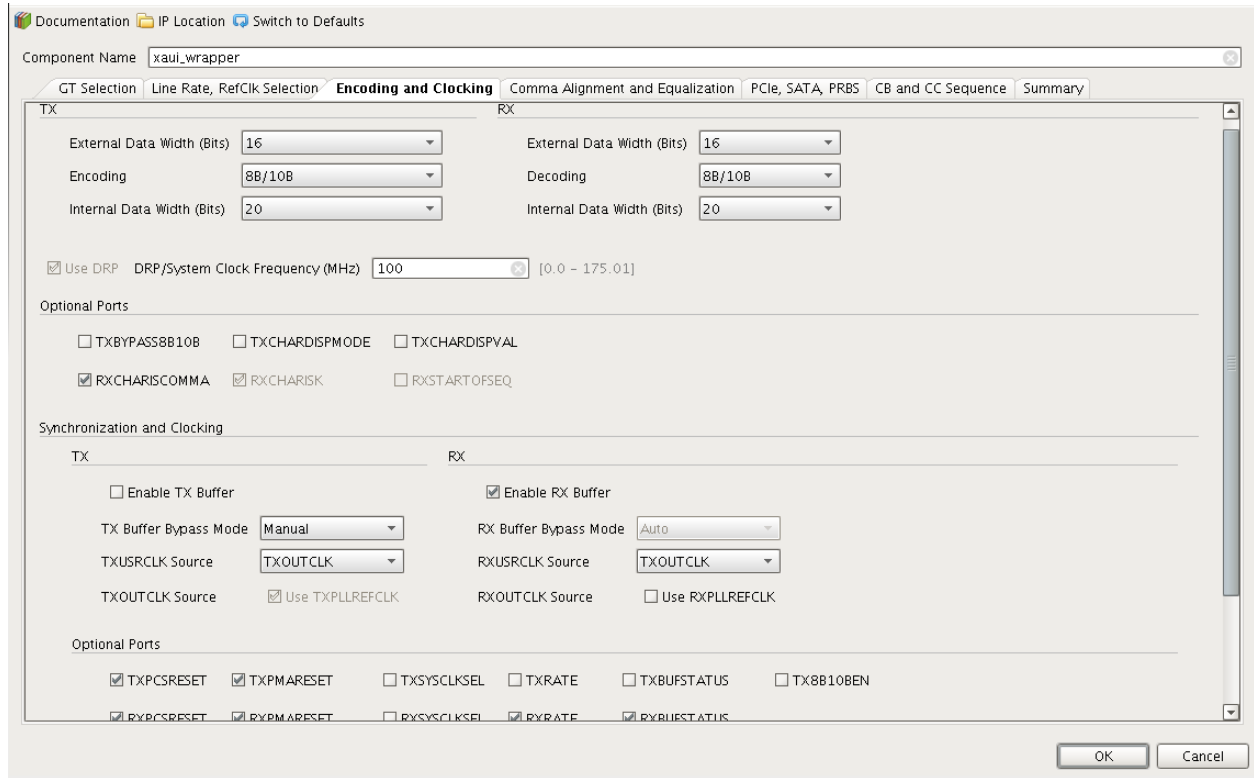


Figure 4-12: Encoding and Optional Ports—Page 3

Table 4-13: TX Settings

Options		Description
External Data Width	16	Sets the transmitter application interface data width to two 8-bit bytes.
	20	The transmitter application interface datapath width is set to 20 bits.
	32	Sets the transmitter application interface data width to two 8-bit bytes.
	40	Sets the transmitter application interface data width to 40 bits.
	64	Sets the transmitter application interface datapath width to eight 8-bit bytes (64 bits).
	80	Sets the transmitter application interface data width to 80 bits.

Table 4-13: TX Settings (Cont'd)

Options		Description
Encoding	8B/10B	Data stream is passed to an internal 8B/10B encoder prior to transmission.
	64B/ 66B_with_Ext_Seq_Ctr	Data stream is passed through the 64B/66B gearbox and scrambler. Sequence counter for the gearbox is implemented in the example design.
	64B/ 66B_with_Int_Seq_Ctr	GTX transceivers only: Data stream is passed through the 64B/66B gearbox and scrambler. Sequence counter for the gearbox is implemented within the transceiver.
	64B/ 67B_with_Ext_Seq_Ctr	Data stream is passed through the 64B/67B gearbox and scrambler. Sequence counter for the gearbox is implemented in the example design.
	64B/ 67B_with_Int_Seq_Ctr	GTX transceivers only: Data stream is passed through the 64B/67B gearbox and scrambler. Sequence counter for the gearbox is implemented within the transceiver.
Internal Data Width	16	Selects the internal data width as 16.
	20	Selects the internal data width as 20.
	32	Selects the internal data width as 32.
	40	Selects the internal data width as 40.

Notes:

- Options not used by the XAUI example are shaded.

Table 4-14: RX Settings

Options		Description
External Data Width	16	Sets the receiver application interface data width to two 8-bit bytes.
	20	Sets the receiver application interface data width to 20 bits.
	32	Sets the receiver application interface datapath width to four 8-bit bytes (32 bits).
	40	Sets the receiver application interface data width to 40 bits.
	64	Sets the receiver application interface datapath width to eight 8-bit bytes (64 bits).
	80	Sets the receiver application interface data width to 80 bits.
Decoding	8B/10B	Data stream is passed to an internal 8B/10B decoder.
	64B/66B	Data stream is passed through the 64B/66B gearbox and de-scrambler.
	64B/67B	Data stream is passed through the 64B/67B gearbox and de-scrambler.

Table 4-14: RX Settings (Cont'd)

Options		Description
Internal Data Width	16	Selects the internal data width as 16.
	20	Selects the internal data width as 20.
	32	Selects the internal data width as 32.
	40	Selects the internal data width as 40.

Notes:

- Options not used by the XAUI example are shaded.
- RX settings should be the same as TX settings.

Table 4-15: DRP

Option	Description
Use DRP	Select this option to have the dynamic reconfiguration port signals of the CHANNEL block available to the application

The TX PCS/PMA Phase Alignment setting controls whether the TX buffer is enabled or bypassed. See the *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476) [Ref 7] for details on this setting. The RX PCS/PMA alignment setting controls whether the RX phase alignment circuit is enabled.

Table 4-16 shows the optional ports for 8B/10B.

Table 4-16: 8B/10B Optional Ports

Option		Description
TX	TXBYPASS8B10B	2-bit wide port disables 8B/10B encoder on a per-byte basis. High-order bit affects high-order byte of datapath.
	TXCHARDISPMODE	2-bit wide ports control disparity of outgoing 8B/10B data. High-order bit affects high-order byte of datapath.
	TXCHARDISPVAL	
RX	RXCHARISCOMMA	2-bit wide port flags valid 8B/10B comma characters as they are encountered. High-order bit corresponds to high-order byte of datapath.
	RXCHARISK	2-bit wide port flags valid 8B/10B K characters as they are encountered. High-order bit corresponds to high-order byte of datapath.

Notes:

- Options not used by the XAUI example are shaded.

Table 4-17 shows the TX and RX buffer bypass options.

Table 4-17: TX and RX Buffer Bypass Options

Option		Description
TX	Enable TX Buffer	If the Enable TX Buffer checkbox is checked, the TX buffer in the transceiver is enabled. This buffer can be bypassed for low, deterministic latency.
TX	TX Buffer Bypass Mode	This option is only if the TX buffer is bypassed. It is mandatory to use the manual mode. See the <i>7 Series FPGAs GTX/GTH Transceivers User Guide</i> (UG476) [Ref 7] for details.
RX	Enable RX Buffer	If the Enable RX Buffer checkbox is checked, the RX elastic buffer in the transceiver is enabled. This buffer can be bypassed for low, deterministic latency.
RX	RX Buffer Bypass Mode	This option is visible only if the RX buffer is bypassed. Auto mode is the recommended setting. To use manual mode, see the <i>7 Series FPGAs GTX/GTH Transceivers User Guide</i> (UG476) [Ref 7].

Table 4-18 details the TXUSRCLK and RXUSRCLK source signal options.

Table 4-18: TXUSRCLK and RXUSRCLK Source

Option		Description
TX	TXOUTCLK	TXUSRCLK is driven by TXOUTCLK.
RX	TXOUTCLK	RXUSRCLK is driven by TXOUTCLK. This option is not available if the RX buffer is bypassed. For RX buffer bypass mode, RXOUTCLK is used to source RXUSRCLK.

Table 4-19 details the TXOUTCLK and RXOUTCLK source signal options.

Table 4-19: TXOUTCLK and RXOUTCLK Source

Option		Description
TX	Use TXPLLREFCLK	If the check box Use TXPLLREFCLK is checked, TXOUTCLK ⁽¹⁾ is generated from the input reference clock; otherwise, the Wizard selects the appropriate source for TXOUTCLK.
RX	Use RXPLLREFCLK	If the check box Use RXPLLREFCLK is checked, RXOUTCLK ⁽¹⁾ is generated from the input reference clock; otherwise, the Wizard selects the appropriate source for RXOUTCLK.

Notes:

1. See *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476) [Ref 7] for more information on TXOUTCLK and RXOUTCLK control.

Table 4-20 shows the optional ports available for latency and clocking.

Table 4-20: Optional Ports

Option	Description
TXPCSRESET	Active-High reset signal for the transmitter PCS logic.
TXBUFSTATUS	2-bit signal monitors the status of the TX elastic buffer. This option is not available when the TX buffer is bypassed.
TXRATE	Transmit rate change port.

Table 4-20: Optional Ports (Cont'd)

Option	Description
RXPCSRESET	Active-High reset signal for the receiver PCS logic.
RXBUFSTATUS	Indicates condition of the RX elastic buffer. Option is not available when the RX buffer is bypassed.
RXBUFRESET	Active-High reset signal for the RX elastic buffer logic. This option is not available when the RX buffer is bypassed.
RXRATE	Receive rate change port.
QPLLPD	Visible only when GTX or GTH transceiver is selected. Powerdown port for QPLL.
CPLLPD	Visible only when GTX or GTH transceiver is selected. Powerdown port for CPLL.
PLL0PD	Visible only when GTP transceiver is selected. Powerdown port for PLL0.
PLL1PD	Visible only when GTP transceiver is selected. Powerdown port for PLL1.
TXSYSCLKSEL	Selects the reference clock source to drive the TX datapath.
RXSYSCLKSEL	Selects the reference clock source to drive the RX datapath.
TXPMARESET	Active-High reset signal for the transmitter PMA logic.
RXPMARESET	Active-High reset signal for the receiver PMA logic.
TX8B10BEN	TX8B10BEN is set High to enable the 8B/10B encoder.
RXCDRHOLD	Holds the CDR control loop frozen.
SIGVALIDCLK	Visible for GTH or GTP transceiver. Clock for OOB circuit.
CLKRSVD	Visible for GTX transceiver. Clock for OOB circuit.
TXPIPPMEN	Enables the TX phase interpolator PPM control block.
TXPIPPMOVRDEN	Enables direct control of TXPI.
TXPIPPMPD	Power downs the TX interpolator PPM control block.
TXPIPPMSTEPSIZE	Specifies the amount to increment or decrement PI code.

Notes:

- Options not used by the XAUI example are shaded.

Alignment, Termination, and Equalization

Page 4 of the Wizard ([Figure 4-13](#)) allows you to set comma characters and control receive equalization and terminal voltage.

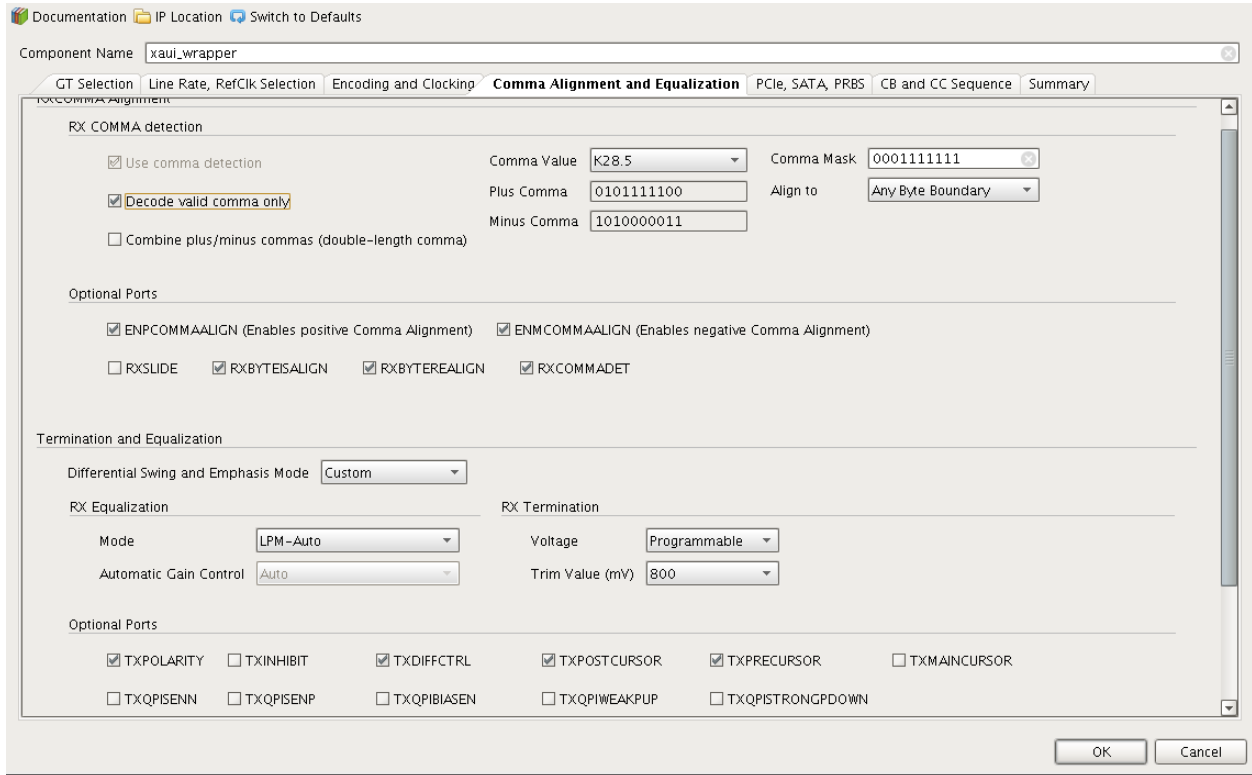


Figure 4-13: Comma Alignment and Equalization—Page 4

Table 4-21 shows the receive comma alignment settings.

Table 4-21: Comma Detection

Option	Description
Use Comma Detection	Enables receive comma detection. Used to identify comma characters and SONET framing characters in the data stream.
Decode Valid Comma Only	When receive comma detection is enabled, limits the detection to specific defined comma characters.
Comma Value	Select one of the standard comma patterns or User Defined to enter a custom pattern. The XAUI example uses K28.5.
Plus Comma	10-bit binary pattern representing the positive-disparity comma character to match. The right-most bit of the pattern is the first bit to arrive serially. The XAUI example uses 0101111100 (K28.5).
Minus Comma	10-bit binary pattern representing the negative-disparity comma character to match. The right-most bit of the pattern is the first bit to arrive serially. The XAUI example uses 1010000011 (K28.5).

Table 4-21: Comma Detection (Cont'd)

Option		Description
Comma Mask		10-bit binary pattern representing the mask for the comma match patterns. A 1 bit indicates the corresponding bit in the comma patterns is to be matched. A 0 bit indicates <i>don't care</i> for the corresponding bit in the comma patterns. The XAUI example matches the lower seven bits (K28.5).
Align to...	Any Byte Boundary	When a comma is detected, the data stream is aligned using the comma pattern to the nearest byte boundary.
	Two Byte Boundary	When a comma is detected, the data stream is aligned using the comma pattern to the 2-byte boundary.
	Four Byte Boundary	When a comma is detected, the data stream is aligned using the comma pattern to the 4-byte boundary.
Optional Ports	ENPCOMMAALIGN	Active-High signal that enables the byte boundary alignment process when the plus comma pattern is detected.
	ENMCOMMAALIGN	Active-High signal that enables the byte boundary alignment process when the minus comma pattern is detected.
	RXSLIDE	Active-High signal that causes the byte alignment to be adjusted by one bit with each assertion. Takes precedence over normal comma alignment.
	RXBYTEISALIGNED	Active-High signal indicating that the parallel data stream is aligned to byte boundaries.
	RXBYTEREALIGN	Active-High signal indicating that byte alignment has changed with a recent comma detection. Note that data errors can occur with this condition.
	RXCOMMADET	Active-High signal indicating the comma alignment logic has detected a comma pattern in the data stream.

Notes:

- Options not used by the XAUI example are shaded.

Table 4-22 details the pre-emphasis and differential swing settings.

Table 4-22: Pre-emphasis and Differential Swing

Option	Description
Differential Swing and Emphasis Mode	Specifies the transmitter pre-cursor pre-emphasis mode setting. Selecting Custom mode enables user driven settings for differential swing and pre-emphasis level. The XAUI example uses the Custom mode to dynamically set the pre-emphasis level. See the <i>7 Series FPGAs GTX/GTH Transceivers User Guide (UG476)</i> [Ref 7] for details.

Table 4-23 describes the RX equalization settings.

Table 4-23: RX Equalization

Option	Description
Equalization Mode	Sets the equalization mode in the receiver. See the <i>7 Series FPGAs GTX/GTH Transceivers User Guide</i> (UG476) [Ref 7] for details on the decision feedback equalizer. The XAUI example uses DFE-Auto mode.
Automatic Gain Control	Sets the automatic gain control of the receiver. The value can be set to Auto or Manual .
Use RX Equalizer CTLE3 Adaptation Logic (DFE mode only)	Applicable only to GTX transceivers. If checked, the CTLE3 adaptation logic is instantiated in the example design. For more information, see Chapter 5, Example Design .

Notes:

- Options not used by the XAUI example are shaded.

[Table 4-24](#) describes the RX termination settings.

Table 4-24: RX Termination

Option	Description
Termination Voltage	Selecting GND grounds the internal termination network. Selecting Floating isolates the network. Selecting AVTT applies an internal voltage reference source to the termination network. Select the Programmable option for Termination Voltage to select RX termination voltage from a drop-down menu. The XAUI example uses the GND setting.

[Table 4-25](#) lists the optional ports available on this page.

Table 4-25: Optional Ports

Option	Description
TXPOLARITY	Active-High signal to invert the polarity of the transmitter output.
TXINHIBIT	Active-High signal forces transmitter output to steady state.
RXPOLARITY	Active-High signal inverts the polarity of the receive data signal.
TXQPIBIASEN	Active-High signal to enable QPI bias.
TXQPIWEAKUP	Active-High signal transmit for QPI.
RXDFAAGCOVRDEN	Active-High signal for DFE AGC over-ride.
TXPOSTCURSOR	TXPOSTCURSOR port.
TXPRECURSOR	TXPRECURSOR port.
TXQPISENN	Transmit QPI port (negative polarity).
RXDFEMONITOROUT	Receive DFE monitor port.
RXLPMHFOVRDEN	Receive low pass override enable port.
TXQPISENP	Transmit QPI port (positive polarity).

Table 4-25: Optional Ports (Cont'd)

Option	Description
RXDFEMONITORSEL	Receive DFE monitor select port.
RXLPLMKLOVRDEN	Receive low pass override enable port.
TXQPISTRONGPDOWN	Transmit QPI power down port.
RXDFELPMRESET	Resets the receive DFE/LPM block.
TXDIFFCTRL	Transmit driver swing control.
RXQPISENN	Sense output that registers a 1 or 0 on the MGTRXN pin.
RXQPISENP	Sense output that registers a 1 or 0 on the MGTRXP pin.
RXQPIEN	Disables the RX termination for the QPI protocol.

Notes:

- Options not used by the XAUI example are shaded.

PCI Express, SATA, OOB, PRBS, Channel Bonding, and Clock Correction Selection

Page 5 of the Wizard ([Figure 4-14](#)) allows you to configure the receiver for PCI Express and Serial ATA (SATA) features. In addition, configuration options for the RX out-of-band (OOB) signal, pseudo-random bitstream sequence (PRBS) detector, and channel bonding and clock correction settings are provided.

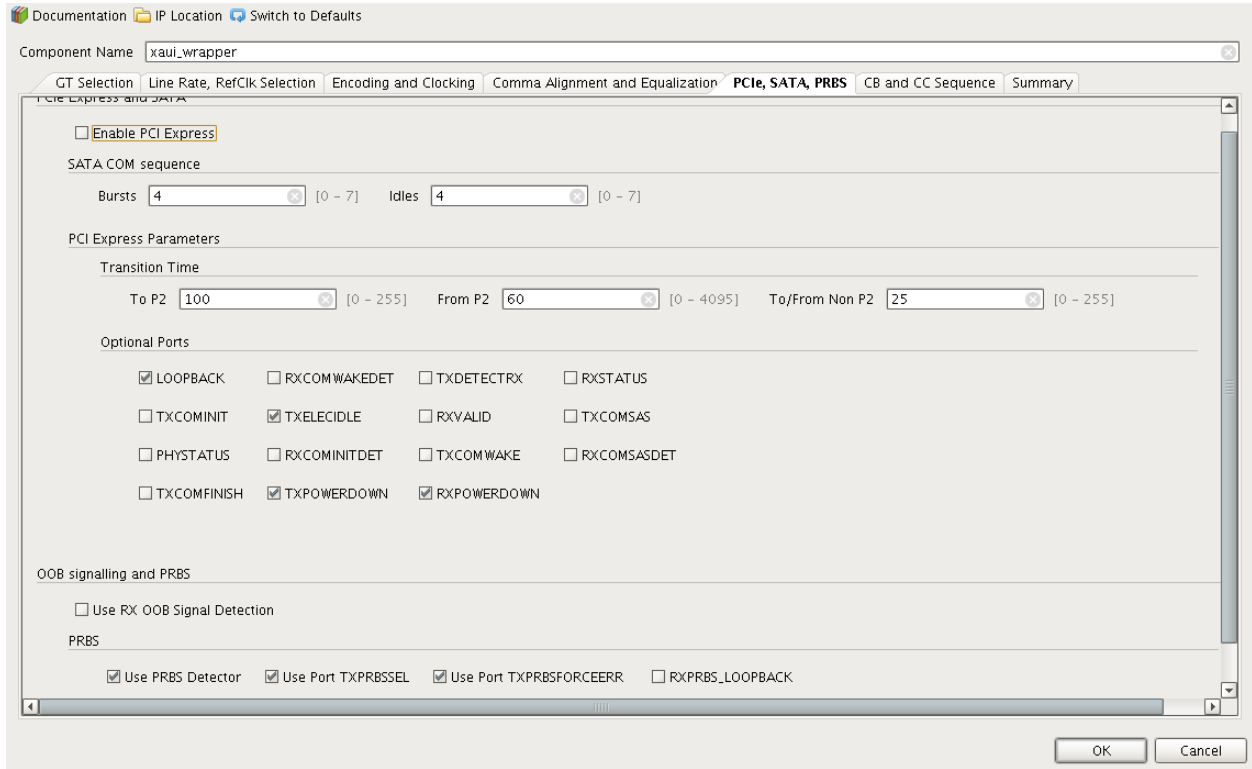


Figure 4-14: PCIe, SATA, OOB, PRBS, Channel Bonding, and Clock Correction Selection—Page 5

Table 4-26 details the receiver SATA configuration options.

Table 4-26: Receiver Serial ATA Options

Options		Description
Enable PCI Express		Selecting this option enables certain operations specific to PCI Express, including enabling options for PCI Express powerdown modes and PCIe® channel bonding. This option should be activated whenever the transceiver is used for PCI Express.
SATA COM Sequence	Bursts	Integer value between 0 and 7 indicating the number of burst sequences to declare a COM match. This value defaults to 4, which is the burst count specified in the SATA specification for COMINIT, COMRESET, and COMWAKE.
	Idles	Integer value between 0 and 7 indicating the number of idle sequences to declare a COM match. Each idle is an OOB signal with a length that matches COMINIT/COMRESET or COMWAKE.

Notes:

- Options not used by the XAUI example are shaded.

Table 4-27 details the receiver PCI Express configuration options.

Table 4-27: PCI Express and SATA Parameters

Option		Description
Transition Time	To P2	Integer value between 0 and 65,535. Sets a counter to determine the transition time to the P2 power state for PCI Express. See the <i>7 Series FPGAs GTX/GTH Transceivers User Guide</i> (UG476) [Ref 7] for details on determining the time value for each count. The XAU1 example does not require this feature and uses the default setting of 100.
	From P2	Integer value between 0 and 65,535. Sets a counter to determine the transition time from the P2 power state for PCI Express. See the <i>7 Series FPGAs GTX/GTH Transceivers User Guide</i> (UG476) [Ref 7] for details on determining the time value for each count. The XAU1 example does not require this feature and uses the default setting of 60.
	To/From non-P2	Integer value between 0 and 65,535. Sets a counter to determine the transition time to or from power states other than P2 for PCI Express. See the <i>7 Series FPGAs GTX/GTH Transceivers User Guide</i> (UG476) [Ref 7] for details on determining the time value for each count. The XAU1 example does not require this feature and uses the default setting of 25.

Table 4-27: PCI Express and SATA Parameters (Cont'd)

Option	Description	
Optional Ports	LOOPBACK	3-bit signal to enable the various data loopback modes for testing.
	RXSTATUS	3-bit receiver status signal. The encoding of this signal is dependent on the setting of RXSTATUS encoding format.
	RXVALID	Active-High, PCI Express RX OOB/beacon signal. Indicates symbol lock and valid data on RXDATA and RXCHARISK[3:0].
	COMINITDET	Active-High initialization detection signal.
	COMSASDET	Active-High detection signal for SATA.
	COMWAKEDET	Active-High wake up detection signal.
	TXCOMINIT	Transmit initialization port.
	TXCOMSAS	OOB signal.
	TXCOMWAKE	OOB signal.
	COMFINISH	Completion of OOB.
	TXDETECTRX	PIPE interface for PCI Express specification-compliant control signal. Activates the PCI Express receiver detection feature. Function depends on the state of TXPOWERDOWN, RXPOWERDOWN, TXELECIDLE, TXCHARDISPMODE, and TXCHARDISPVAL. This port is not available if RXSTATUS encoding format is set to SATA.
	TXELECIDLE	Drives the transmitter to an electrical idle state (no differential voltage). In PCI Express mode this option is used for electrical idle modes. Function depends on the state of TXPOWERDOWN, RXPOWERDOWN, TXELECIDLE, TXCHARDISPMODE, and TXCHARDISPVAL.
	PHYSTATUS	PCI Express receive detect support signal. Indicates completion of several PHY functions.
	TXPOWERDOWN	Powerdown port for the transmitter.
RXPOWERDOWN	Powerdown port for the receiver.	

Notes:

- Options not used by the XAUI example are shaded.

Table 4-28 shows the OOB signal detection options.

Table 4-28: OOB Signal Detection

Option	Description
Use RX OOB Signal Detection	Enables the internal OOB signal detector. OOB signal detection is used for PCIe and SATA.

Table 4-29 details the PRBS settings.

Table 4-29: PRBS Detector

Option	Description
Use PRBS Detector	Enables the internal PRBS detector. This feature can be used by an application to implement a built-in self test.
Use Port TXPRBSSEL	Selects the PRBS Transmission control port.
Use Port TXPRBSFORCEERR	Enables the PRBS force error control port. This port controls the insertion of errors into the bitstream.
RXPRBSERR_LOOPBACK	Select this option to loopback RXPRBSERR bit to TXPRBSFORCEERR of the same transceiver.

Notes:

- Options not used by the XAUI example are shaded.

Table 4-30 shows the channel bonding options.

Table 4-30: Channel Bonding Setup

Option	Description
Use Channel Bonding	Enables receiver channel bonding logic using unique character sequences. When recognized, these sequences allow for adding or deleting characters in the receive buffer to byte-align multiple data transceivers.
Sequence Length	Select from the drop-down list the number of characters in the unique channel bonding sequence. The XAUI example uses 1.
Sequence Max Skew	Select from the drop-down list the maximum skew in characters that can be handled by channel bonding. Must always be less than half the minimum distance between channel bonding sequences. The XAUI example uses 7.
Use Two Channel Bonding Sequences	Activates the optional second channel bonding sequence. Detection of either sequence triggers channel bonding.

Notes:

- Options not used by the XAUI example are shaded.

Table 4-31 shows the clock correction options.

Table 4-31: Clock Correction Setup

Option	Description
Use Clock Correction	Enables receiver clock correction logic using unique character sequences. When recognized, these sequences allow for adding or deleting characters in the receive buffer to prevent buffer underflow/overflow due to small differences in the transmit/receive clock frequencies.
Sequence Length	Select from the drop-down list the number of characters (subsequences) in the unique clock correction sequence. The XAUI example uses 1.

Table 4-31: Clock Correction Setup (Cont'd)

Option	Description
PPM Offset	Indicates the PPM offset between the transmit and receive clocks.
Periodicity of the CC Sequence	Indicates the interval at which CC sequences are inserted in the data stream.
Use Two Clock Correction Sequences	Activates the optional second clock correction sequence. Detection of either sequence triggers clock correction.

Notes:

- Options not used by the XAUI example are shaded.

Channel Bonding and Clock Correction Sequence

Page 6 of the Wizard (Figure 4-15) allow you to define the channel bonding sequence(s). Table 4-32 describes the sequence definition settings and Table 4-31, page 70 describes the clock setup settings.

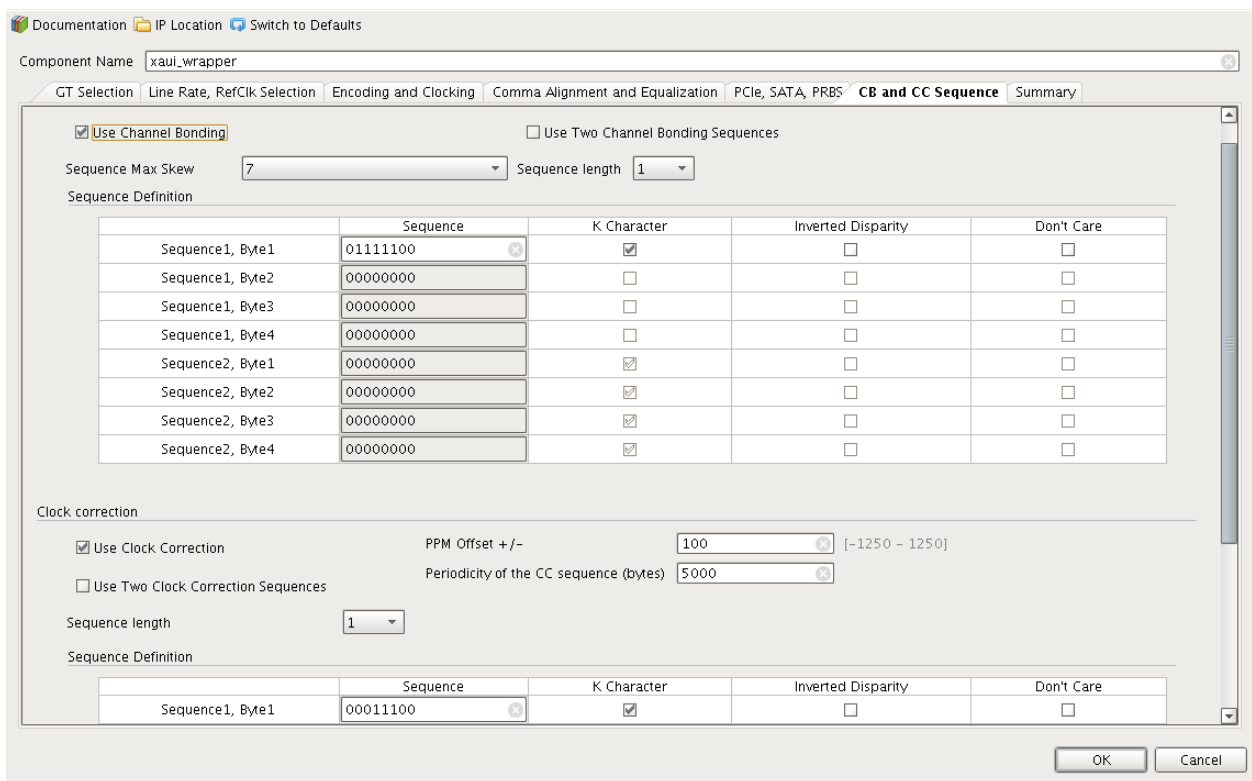


Figure 4-15: Channel Bonding and Clock Correction Sequence—Page 6

Table 4-32: Channel Bonding and Clock Correction Sequences

Option	Description
Byte (Symbol)	Set each symbol to match the pattern the protocol requires. The XAUI sequence length is 8 bits. 01111100 is used for channel bonding. 00011100 is used for clock correction. The other symbols are disabled because the sequence length is set to 1.
K Character	This option is available when 8B/10B decoding is selected. When checked, as is the case for XAUI, the symbol is an 8B/10B K character.
Inverted Disparity	Some protocols with 8B/10B decoding use symbols with deliberately inverted disparity. This option should be checked when such symbols are expected in the sequence.
Don't Care	Multiple-byte sequences can have wild card symbols by checking this option. Unused bytes in the sequence automatically have this option set.

Notes:

- Options not used by the XAUI example are shaded.

Summary

Page 7 of the Wizard (Figure 4-16) provides a summary of the selected configuration parameters. After reviewing the settings, click **Generate** to exit and generate the wrapper.

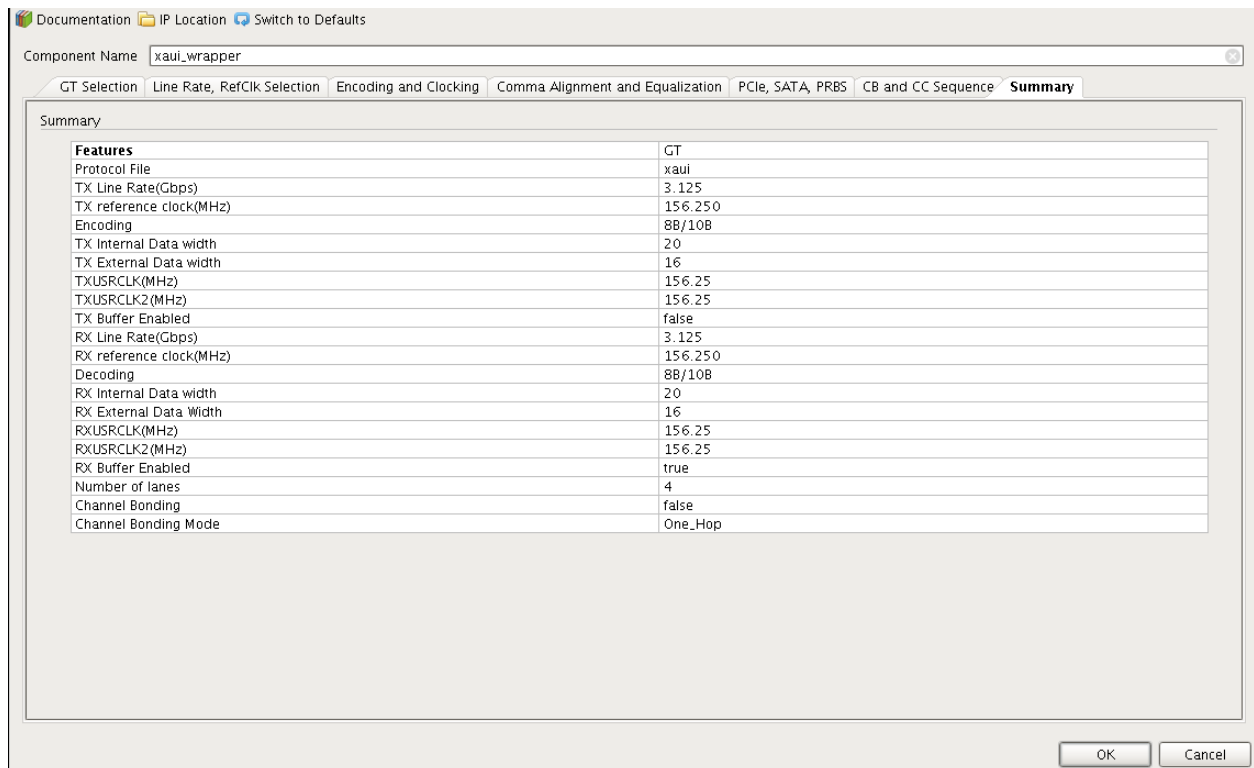


Figure 4-16: Summary—Page 6

Constraining the Core

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

Required Constraints

The 7 series FPGAs Transceivers Wizard core is generated with its own timing constraints based on the choices you made when customizing the core using the Wizard. The reference location constraints provided as part of example design setup and the user can modify the constraints according to their use case.

Device, Package, and Speed Grade Selections

Device support is detailed in the release notes for this core.

Clock Frequencies

7 Series FPGA Transceiver Core Reference Clock Constraint

The number of reference clocks is derived based on transceiver selection. The Reference Clock (MHz) value selected in the second tab of the Vivado IDE is used to constrain the required reference clock. The **create_clock** XDC command is used to constrain all necessary required reference clocks.

False Paths

The system clock and user clock are not related to one another. No phase relationship exists between these two clocks. The two clock domains need to be set as false paths. The **set_false_path** XDC command is used to constrain the false paths.

Example Design

The generated example design is a 10.3125 Gb/s line rate and a 156.25 MHz reference clock for protocol 10-GBASE-R. The XDC file generated for the XC7VX1140TFLG1926-2L device follows:

```
##### Clock Constraints #####

##### GT reference clock constraints #####
# create_clock -name q0_clk1_refclk_i -period 6.4 q0_clk1_refclk_i

#create_clock -name GT0_GTREFCLK0_IN -period 6.4 [get_pins -hier -filter
{name=~*gt0_xaui_wrapper_i*gthe2_i*GTREFCLK0}]
```

```
#create_clock -name GT1_GTREFCLK0_IN -period 6.4 [get_pins -hier -filter
{name=~*gt1_xaui_wrapper_i*gthe2_i*GTREFCLK0}]
#create_clock -name GT2_GTREFCLK0_IN -period 6.4 [get_pins -hier -filter
{name=~*gt2_xaui_wrapper_i*gthe2_i*GTREFCLK0}]
#create_clock -name GT3_GTREFCLK0_IN -period 6.4 [get_pins -hier -filter
{name=~*gt3_xaui_wrapper_i*gthe2_i*GTREFCLK0}]
create_clock -name GT0_GTREFCLK0_COMMON_IN -period 6.4 [get_pins -hier -filter
{name=~*common0_i*gthe2_common_i*GTREFCLK0}]

create_clock -name drpclk_in_i -period 10.0 [get_pins -hier -filter
{name=~*xaui_wrapper_support_i*gt_usrclk_source*DRP_CLK_BUFG*I}]
#
# # DRP Clock Constraint
# create_clock -name drpclk_in_i -period 10.0 [get_pins -hier -filter
{name=~*xaui_wrapper_support_i*gt_usrclk_source*DRPCLK_OUT}]
# ]
#

# User Clock Constraints
# create_clock -name gt0_txusrclk_i -period 6.4 [get_pins -hier -filter
{name=~*xaui_wrapper_support_i*xaui_wrapper_init_i*gt0_txusrclk_in}]
# create_clock -name gt0_txusrclk2_i -period 6.4 [get_pins -hier -filter
{name=~*xaui_wrapper_support_i*xaui_wrapper_init_i*gt0_txusrclk2_in}]
# create_clock -name gt0_rxusrclk_i -period 6.4 [get_pins -hier -filter
{name=~*xaui_wrapper_support_i*xaui_wrapper_init_i*gt0_rxusrclk_in}]
# create_clock -name gt0_rxusrclk2_i -period 6.4 [get_pins -hier -filter
{name=~*xaui_wrapper_support_i*xaui_wrapper_init_i*gt0_rxusrclk2_in}]
# create_clock -name gt1_txusrclk_i -period 6.4 [get_pins -hier -filter
{name=~*xaui_wrapper_support_i*xaui_wrapper_init_i*gt1_txusrclk_in}]
# create_clock -name gt1_txusrclk2_i -period 6.4 [get_pins -hier -filter
{name=~*xaui_wrapper_support_i*xaui_wrapper_init_i*gt1_txusrclk2_in}]
# create_clock -name gt1_rxusrclk_i -period 6.4 [get_pins -hier -filter
{name=~*xaui_wrapper_support_i*xaui_wrapper_init_i*gt1_rxusrclk_in}]
# create_clock -name gt1_rxusrclk2_i -period 6.4 [get_pins -hier -filter
{name=~*xaui_wrapper_support_i*xaui_wrapper_init_i*gt1_rxusrclk2_in}]
# create_clock -name gt2_txusrclk_i -period 6.4 [get_pins -hier -filter
{name=~*xaui_wrapper_support_i*xaui_wrapper_init_i*gt2_txusrclk_in}]
# create_clock -name gt2_txusrclk2_i -period 6.4 [get_pins -hier -filter
{name=~*xaui_wrapper_support_i*xaui_wrapper_init_i*gt2_txusrclk2_in}]
# create_clock -name gt2_rxusrclk_i -period 6.4 [get_pins -hier -filter
{name=~*xaui_wrapper_support_i*xaui_wrapper_init_i*gt2_rxusrclk_in}]
# create_clock -name gt2_rxusrclk2_i -period 6.4 [get_pins -hier -filter
{name=~*xaui_wrapper_support_i*xaui_wrapper_init_i*gt2_rxusrclk2_in}]
# create_clock -name gt3_txusrclk_i -period 6.4 [get_pins -hier -filter
{name=~*xaui_wrapper_support_i*xaui_wrapper_init_i*gt3_txusrclk_in}]
# create_clock -name gt3_txusrclk2_i -period 6.4 [get_pins -hier -filter
{name=~*xaui_wrapper_support_i*xaui_wrapper_init_i*gt3_txusrclk2_in}]
# create_clock -name gt3_rxusrclk_i -period 6.4 [get_pins -hier -filter
{name=~*xaui_wrapper_support_i*xaui_wrapper_init_i*gt3_rxusrclk_in}]
# create_clock -name gt3_rxusrclk2_i -period 6.4 [get_pins -hier -filter
{name=~*xaui_wrapper_support_i*xaui_wrapper_init_i*gt3_rxusrclk2_in}]

##### RefClk Location constraints #####
set_property LOC AA30 [get_ports Q0_CLK1_GTREFCLK_PAD_N_IN ]
set_property LOC AA29 [get_ports Q0_CLK1_GTREFCLK_PAD_P_IN ]

## LOC constrain for DRP_CLK_P/N
## set_property LOC C25 [get_ports DRP_CLK_IN_P]
## set_property LOC B25 [get_ports DRP_CLK_IN_N]
```

```
##### mgt wrapper constraints #####

##----- Set placement for gt0_gth_wrapper_i/GTHE2_CHANNEL -----
set_property LOC GTHE2_CHANNEL_X0Y0 [get_cells xau_i_wrapper_support_i/xau_i_wrapper_init_i/
inst/xau_i_wrapper_i/gt0_xau_i_wrapper_i/gthe2_i]
##----- Set placement for gt1_gth_wrapper_i/GTHE2_CHANNEL -----
set_property LOC GTHE2_CHANNEL_X0Y1 [get_cells xau_i_wrapper_support_i/xau_i_wrapper_init_i/
inst/xau_i_wrapper_i/gt1_xau_i_wrapper_i/gthe2_i]
##----- Set placement for gt2_gth_wrapper_i/GTHE2_CHANNEL -----
set_property LOC GTHE2_CHANNEL_X0Y2 [get_cells xau_i_wrapper_support_i/xau_i_wrapper_init_i/
inst/xau_i_wrapper_i/gt2_xau_i_wrapper_i/gthe2_i]
##----- Set placement for gt3_gth_wrapper_i/GTHE2_CHANNEL -----
set_property LOC GTHE2_CHANNEL_X0Y3 [get_cells xau_i_wrapper_support_i/xau_i_wrapper_init_i/
inst/xau_i_wrapper_i/gt3_xau_i_wrapper_i/gthe2_i]

##----- Set ASYNC_REG for flop which have async input -----
##set_property ASYNC_REG TRUE [get_cells -hier -filter
{name=~*gt0_frame_gen*system_reset_r_reg}]
##set_property ASYNC_REG TRUE [get_cells -hier -filter
{name=~*gt0_frame_check*system_reset_r_reg}]
##set_property ASYNC_REG TRUE [get_cells -hier -filter
{name=~*gt1_frame_gen*system_reset_r_reg}]
##set_property ASYNC_REG TRUE [get_cells -hier -filter
{name=~*gt1_frame_check*system_reset_r_reg}]
##set_property ASYNC_REG TRUE [get_cells -hier -filter
{name=~*gt2_frame_gen*system_reset_r_reg}]
##set_property ASYNC_REG TRUE [get_cells -hier -filter
{name=~*gt2_frame_check*system_reset_r_reg}]
##set_property ASYNC_REG TRUE [get_cells -hier -filter
{name=~*gt3_frame_gen*system_reset_r_reg}]
##set_property ASYNC_REG TRUE [get_cells -hier -filter
{name=~*gt3_frame_check*system_reset_r_reg}]

##----- Set False Path from one clock to other -----
# set_false_path -from [get_clocks "gt0_txusrclk2_i"] -to [get_clocks
-include_generated_clocks "gt0_txusrclk_i"]
# set_false_path -from [get_clocks "gt0_txusrclk_i"] -to [get_clocks
-include_generated_clocks "gt0_txusrclk2_i"]
#
# set_false_path -from [get_clocks "gt0_rxusrclk2_i"] -to [get_clocks
-include_generated_clocks "gt0_rxusrclk_i"]
# set_false_path -from [get_clocks "gt0_rxusrclk_i"] -to [get_clocks
-include_generated_clocks "gt0_rxusrclk2_i"]
# set_false_path -from [get_clocks "drpclk_in_i"] -to [get_clocks -include_generated_clocks
"gt0_txusrclk_i"]
# set_false_path -from [get_clocks -include_generated_clocks "gt0_txusrclk_i"] -to
[get_clocks "drpclk_in_i"]
# set_false_path -from [get_clocks "drpclk_in_i"] -to [get_clocks -include_generated_clocks
"gt0_txusrclk2_i"]
# set_false_path -from [get_clocks -include_generated_clocks "gt0_txusrclk2_i"] -to
[get_clocks "drpclk_in_i"]
#
# set_false_path -from [get_clocks "drpclk_in_i"] -to [get_clocks -include_generated_clocks
"gt0_rxusrclk_i"]
# set_false_path -from [get_clocks -include_generated_clocks "gt0_rxusrclk_i"] -to
[get_clocks "drpclk_in_i"]
# set_false_path -from [get_clocks "drpclk_in_i"] -to [get_clocks -include_generated_clocks
"gt0_rxusrclk2_i"]
```

```
# set_false_path -from [get_clocks -include_generated_clocks "gt0_rxusrclk2_i"] -to
[get_clocks "drpclk_in_i"]

# set_false_path -from [get_clocks "gt1_txusrclk2_i"] -to [get_clocks
-include_generated_clocks "gt1_txusrclk_i"]
# set_false_path -from [get_clocks "gt1_txusrclk_i"] -to [get_clocks
-include_generated_clocks "gt1_txusrclk2_i"]
#
# set_false_path -from [get_clocks "gt1_rxusrclk2_i"] -to [get_clocks
-include_generated_clocks "gt1_rxusrclk_i"]
# set_false_path -from [get_clocks "gt1_rxusrclk_i"] -to [get_clocks
-include_generated_clocks "gt1_rxusrclk2_i"]
# set_false_path -from [get_clocks "drpclk_in_i"] -to [get_clocks -include_generated_clocks
"gt1_txusrclk_i"]
# set_false_path -from [get_clocks -include_generated_clocks "gt1_txusrclk_i"] -to
[get_clocks "drpclk_in_i"]
# set_false_path -from [get_clocks "drpclk_in_i"] -to [get_clocks -include_generated_clocks
"gt1_txusrclk2_i"]
# set_false_path -from [get_clocks -include_generated_clocks "gt1_txusrclk2_i"] -to
[get_clocks "drpclk_in_i"]
#
# set_false_path -from [get_clocks "drpclk_in_i"] -to [get_clocks -include_generated_clocks
"gt1_rxusrclk_i"]
# set_false_path -from [get_clocks -include_generated_clocks "gt1_rxusrclk_i"] -to
[get_clocks "drpclk_in_i"]
# set_false_path -from [get_clocks "drpclk_in_i"] -to [get_clocks -include_generated_clocks
"gt1_rxusrclk2_i"]
# set_false_path -from [get_clocks -include_generated_clocks "gt1_rxusrclk2_i"] -to
[get_clocks "drpclk_in_i"]

# set_false_path -from [get_clocks "gt2_txusrclk2_i"] -to [get_clocks
-include_generated_clocks "gt2_txusrclk_i"]
# set_false_path -from [get_clocks "gt2_txusrclk_i"] -to [get_clocks
-include_generated_clocks "gt2_txusrclk2_i"]
#
# set_false_path -from [get_clocks "gt2_rxusrclk2_i"] -to [get_clocks
-include_generated_clocks "gt2_rxusrclk_i"]
# set_false_path -from [get_clocks "gt2_rxusrclk_i"] -to [get_clocks
-include_generated_clocks "gt2_rxusrclk2_i"]
# set_false_path -from [get_clocks "drpclk_in_i"] -to [get_clocks -include_generated_clocks
"gt2_txusrclk_i"]
# set_false_path -from [get_clocks -include_generated_clocks "gt2_txusrclk_i"] -to
[get_clocks "drpclk_in_i"]
# set_false_path -from [get_clocks "drpclk_in_i"] -to [get_clocks -include_generated_clocks
"gt2_txusrclk2_i"]
# set_false_path -from [get_clocks -include_generated_clocks "gt2_txusrclk2_i"] -to
[get_clocks "drpclk_in_i"]
#
# set_false_path -from [get_clocks "drpclk_in_i"] -to [get_clocks -include_generated_clocks
"gt2_rxusrclk_i"]
# set_false_path -from [get_clocks -include_generated_clocks "gt2_rxusrclk_i"] -to
[get_clocks "drpclk_in_i"]
# set_false_path -from [get_clocks "drpclk_in_i"] -to [get_clocks -include_generated_clocks
"gt2_rxusrclk2_i"]
# set_false_path -from [get_clocks -include_generated_clocks "gt2_rxusrclk2_i"] -to
[get_clocks "drpclk_in_i"]

# set_false_path -from [get_clocks "gt3_txusrclk2_i"] -to [get_clocks
-include_generated_clocks "gt3_txusrclk_i"]
```

```

# set_false_path -from [get_clocks "gt3_txusrclk_i"] -to [get_clocks
-include_generated_clocks "gt3_txusrclk2_i"]
#
# set_false_path -from [get_clocks "gt3_rxusrclk2_i"] -to [get_clocks
-include_generated_clocks "gt3_rxusrclk_i"]
# set_false_path -from [get_clocks "gt3_rxusrclk_i"] -to [get_clocks
-include_generated_clocks "gt3_rxusrclk2_i"]
# set_false_path -from [get_clocks "drpclk_in_i"] -to [get_clocks -include_generated_clocks
"gt3_txusrclk_i"]
# set_false_path -from [get_clocks -include_generated_clocks "gt3_txusrclk_i"] -to
[get_clocks "drpclk_in_i"]
# set_false_path -from [get_clocks "drpclk_in_i"] -to [get_clocks -include_generated_clocks
"gt3_txusrclk2_i"]
# set_false_path -from [get_clocks -include_generated_clocks "gt3_txusrclk2_i"] -to
[get_clocks "drpclk_in_i"]
#
# set_false_path -from [get_clocks "drpclk_in_i"] -to [get_clocks -include_generated_clocks
"gt3_rxusrclk_i"]
# set_false_path -from [get_clocks -include_generated_clocks "gt3_rxusrclk_i"] -to
[get_clocks "drpclk_in_i"]
# set_false_path -from [get_clocks "drpclk_in_i"] -to [get_clocks -include_generated_clocks
"gt3_rxusrclk2_i"]
# set_false_path -from [get_clocks -include_generated_clocks "gt3_rxusrclk2_i"] -to
[get_clocks "drpclk_in_i"]

```

Clock Management

There are no specific clock management constraints for this core.

Clock Placement

There are no specific clock placement constraints for this core.

Banking

There are no specific banking constraints for this core.

Transceiver Placement

The `set_property` XDC command is used to constrain the 7 series FPGA transceiver location. This is provided as a tool tip 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 7 series FPGA transceiver reference clock. The `set_property` XDC command is used to constrain the transceiver reference clock pins.

Simulation

For comprehensive information about Vivado simulation components, as well as information about using supported third-party tools, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 4].



IMPORTANT: For cores targeting 7 series devices, UNIFAST libraries are not supported. Xilinx IP is tested and qualified with UNISIM libraries only.

Synthesis and Implementation

For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 9].

Example Design

This chapter introduces the example design that is included with the 7 series FPGAs transceiver wrappers. The example design demonstrates how to use the wrappers and demonstrates some of the key features of the transceivers.

Functional Simulation Using the Vivado Design Tools

The 7 series FPGAs Transceivers Wizard provides a quick way to simulate and observe the behavior of the wrapper using the provided example design and script files.

To simulate simplex designs, the `SIMPLEX_PARTNER` environment variable should be set to the path of the complementary core generated to test the simplex design. For example, if a design is generated with `RX OFF`, a simplex partner design with `RX enabled` is needed to simulate the device under test (DUT). The `SIMPLEX_PARTNER` environment variable should be set to the path of the `RX enabled` design. The name of the simplex partner should be the same as the name of the DUT with a prefix of `tx` or `rx` as applicable. In the current example, the name of the simplex partner design would be prefixed with `rx`.

Using QuestaSim

Prior to simulating the wrapper with QuestaSim, the functional (gate-level) simulation models must be generated. All source files in the following directories must be compiled to a single library as shown in [Table 5-1](#). See the *Synthesis and Simulation Design Guide* (UG626) [\[Ref 13\]](#) for instructions on how to compile simulation libraries.

Table 5-1: Required QuestaSim Simulation Libraries

HDL	Library	Source Directories
Verilog	UNISIMS_VER	<Xilinx dir>/verilog/src/unisims <Xilinx dir>/secureip/mti
VHDL	UNISIM	<Xilinx dir>/vhdl/src/unisims/primitive <Xilinx dir>/secureip/mti

Implementing Using the Vivado Design Tools

When all of the parameters are set as desired, clicking **Generate** creates a directory structure under the provided Component Name. Wrapper generation proceeds and the generated output populates the appropriate subdirectories.

The directory structure for the XAUI example is provided in [Chapter 4, Customizing and Generating the Core](#).

After wrapper generation is complete, the results can be tested in hardware. The provided example design incorporates the wrapper and additional blocks allowing the wrapper to be driven and monitored in hardware. The generated output also includes several scripts to assist in running the software.

From the command prompt, navigate to the project directory and type the following:

For Windows:

```
> cd xau_i_wrapper\implement
> implement.bat
```

For Linux:

```
% cd xau_i_wrapper/implement
% implement.sh
```

Note: Substitute *Component Name* string for `xau_i_wrapper`.

These commands execute a script that synthesizes, builds, maps, places, and routes the example design and produces a bitmap file. The resulting files are placed in the `implement/results` directory.

Timing Simulation Using the Vivado Design Tools

The Wizard provides a script to observe the behavior of the example design during timing simulations. Timing simulations are not supported for the GTZ wizard.

Using QuestaSim

Prior to performing the timing simulation with QuestaSim, the generated design should pass through implementation. All source files in the following directories must be compiled to a single library, as shown in [Table 5-2](#). See the *Synthesis and Simulation Design Guide* (UG626) [\[Ref 13\]](#) for instructions on how to compile simulation libraries.

Table 5-2: Required QuestaSim Timing Simulation Libraries

HDL	Library	Source Directories
Verilog	SIMPRIMS_VER	<Xilinx dir>/verilog/src/simprims <Xilinx dir>/secureip/mti
VHDL	SIMPRIM	<Xilinx dir>/vhdl/src/simprims/primitive <Xilinx dir>/secureip/mti

The Wizard provides a command line script for use within QuestaSim. To run a VHDL or Verilog QuestaSim simulation of the wrapper, use these instructions:

1. Launch the QuestaSim simulator and set the current directory to:

```
<project_directory>/<component_name>/simulation/timing
```

2. Set the MTI_LIBS variable:

```
questasim> setenv MTI_LIBS <path to compiled libraries>
```

3. Launch the simulation script:

```
questasim> do simulate_mti.do
```

The QuestaSim script compiles and simulates the routed netlist of the example design and test bench.

Using Vivado Design Suite Debug Feature with the Wizard

The Vivado Design Suite debug feature ILA and VIO cores aid in debugging and validating the design in the board. To assist with debugging, these cores are provided with the 7 series FPGAs Transceivers Wizard wrapper, which is enabled by setting USE_CHIPSCOPE to 1 in the <component_name>_exdes.v[hd] file.

Directory and File Contents

The 7 series FPGAs Transceivers Wizard directories and their associated files are defined in the following sections.

<project directory>

The <project directory> contains all project files for the Vivado design tools.

Table 5-3: Project Directory

Name	Description
<component_name>.v [hd]	Main transceiver wrapper. Instantiates individual transceiver wrappers. For use in the target design.
<component_name>.[veo vho]	Transceiver wrapper files instantiation templates. Includes templates for the transceiver wrapper module and the IBUFDS module.
<component_name>.xci	Log file from the Vivado design tools describing which options were used to generate the transceiver wrapper. An XCI file is generated by the Vivado design tools for each Wizard wrapper that it creates in the current project directory. An XCI file can also be used as an input to the Vivado design tools.
<component_name>_gt.v [hd]	Individual transceiver wrapper to be instantiated in the main transceiver wrapper. Instantiates the selected transceivers with settings for the selected protocol.
<component_name>_octal0.v	Specific to GTZ transceivers. Individual transceiver wrapper to be instantiated in the main transceiver wrapper. Instantiates the selected transceivers with settings for the selected protocol.
<component_name>_octal1.v	Specific to GTZ transceivers. Individual transceiver wrapper to be instantiated in the main transceiver wrapper. Instantiates the selected transceivers with settings for the selected protocol.

<component name>/<component name>.data

The doc directory contains the PDF documentation provided with the Wizard.

Table 5-4: Doc Directory

Name	Description
<project_dir>/<component_name>/doc	
pg168-gtwizard.pdf	7 Series FPGAs Transceivers Wizard LogiCORE IP User Guide

<component name>/<component name>.srcs

The src directory contains the source code files that are mandatory for certain cores.

Table 5-5: src Directory

Name	Description
<project_dir>/<component_name>/src	
gtwizard_v3_5_beachfront.v	This file should be used to interface the user logic with the GTZ transceiver.

<component name>/<component name>_example design







The example design directory contains the example design files provided with the Wizard wrapper.






Output Generation

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 9].

This section provides detailed information about the files and the directory structure generated by the Xilinx Vivado design tools.

The customized 7 series FPGA transceiver core is delivered as a set of HDL source modules in the language selected in the IP catalog project with supporting files. 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 mentioned below.

-  <component name>_example
 - Contains Vivado project and log files
 -  <component name>_example.src\sources_1
 - IP core and example design files
 -  constrs_1\imports\example_design
 - Example design constraint file
 -  sim_1\imports\simulation
 - Simulation test bench file
 -  source_1/ip/<component name>
 - IP Core source files directory
 -  <component name>
 - IP core Verilog/VHDL source files

-  <component name>\example_design\
 Verilog/VHDL files for example design
-  <component name>
 Vivado Design Suite debug feature files
-  <imports>\<component name>
 -  <component name>\example_design\
 Verilog/VHDL files for example design
-  <example_design>\<support>
 Shared logic resource files

Directory and File Contents

The 7 series FPGAs Transceivers Wizard directories and their associated files are defined in the following sections.

<project directory>\example_project

This is the top-level directory for the example design. The project directory contains the Vivado design tools project file and log file.

Table 5-6: <component name>_example Directory

Name	Description
<component name>_example.xpr	Vivado project file
vivado.log	Vivado project log file
vivado.jou	Vivado project journal file

<component name>_example.src\sources_1

This directory contains example design and IP core source files.

constrs_1\imports\example_design

The directory contains the example design constraint file.

Table 5-7: **constrs_1 Directory**

Name	Description
<component name>_exdes.xdc	7 series FPGA Transceiver core example design XDC constraints

sim_1\imports\simulation

The directory contains the example design test bench file.

Table 5-8: **simulation Directory**

Name	Description
<component name>_tb.v [hd]	Test bench file for example design

source_1\ip\<component name>

This directory contains IP core source files.

<component name>\

The directory contains the IP core source files.

Table 5-9: **IP Core Source Files**

Name	Description
<component_name>.v [hd]	Main transceiver wrapper. Instantiates individual transceiver wrappers. For use in the target design.
<component_name>.[veo vho]	Transceiver wrapper files instantiation templates. Includes templates for the transceiver wrapper module and the IBUFDS module.
<component_name>.xci	Log file from the Vivado tools describing which options were used to generate the transceiver wrapper. An XCI file is generated by the Vivado tools for each Wizard wrapper that it creates in the current project directory. An XCI file can also be used as an input to the Vivado tools.
<component_name>_gt.v [hd]	Individual transceiver wrapper to be instantiated in the main transceiver wrapper. Instantiates the selected transceivers with settings for the selected protocol.
<component_name>_octal0.v	Specific to GTZ transceivers. Individual transceiver wrapper to be instantiated in the main transceiver wrapper. Instantiates the selected transceivers with settings for the selected protocol.

Table 5-9: IP Core Source Files (Cont'd)

Name	Description
<component_name>_octall.v	Specific to GTZ transceivers. Individual transceiver wrapper to be instantiated in the main transceiver wrapper. Instantiates the selected transceivers with settings for the selected protocol.
gtwizard_v3_4_beachfront.v	This file should be used to interface the user logic with the GTZ transceiver.

This directory also contains XCI files.

Table 5-10: XCI files

Name	Description
ila.xci vio.xci	7 series FPGA NGC files for the debug cores compatible with the Vivado Design Suite debug feature.

ip**<component name>**\example_design**<component name>**

Verilog/VHDL files for example design.

Table 5-11: ip**<component name>**\example_design**<component name>**

Name	Description
tx_startup_fsm.v [hd]	Reset module for transmitter. This file is not generated for GTZ transceivers.
rx_startup_fsm.v [hd]	Reset module for receiver. This file is not generated for GTZ transceivers.

imports**<component name>**

This directory contains the example design top module and reset logic module.

Table 5-12: **<component name>**\example_design Directory

Name	Description
gt_frame_check.v [hd]	Frame check logic to be instantiated in the example design.
gt_frame_gen.v [hd]	Frame generator logic to be instantiated in the example design.
<component_name>_exdes.v [hd]	Top-level example design. Contains transceiver, reset logic, and instantiations for frame generator and frame check logic. Also contains Vivado Design Suite debug feature Pro module instantiations.

Table 5-12: <component name>\example_design Directory (Cont'd)

Name	Description
gt_rom_init_tx.dat	Block RAM initialization pattern for gt_frame_gen module. The pattern is user modifiable.
gt_rom_init_rx.dat	Block RAM initialization pattern for gt_frame_check module. The pattern is user modifiable.

<example_design>\<support>

This directory contains shared logic resource files.

Table 5-13: example_design\support Directory

Name	Description
<component_name>_support.v[hd]	Core support layer of the 7 series FPGA transceiver design.
<component_name>_gt_usrclk_source.v[hd]	Transceiver user clock module that generates clocking signals for transceivers and the user logic.
<component_name>_common.v[hd]	This file instantiates the GTXE2_COMMON or GTXE2_COMMON primitive, which is shared across respective multiple transceiver cores.
<component_name>_common_reset.v[hd]	This file generates the reset circuitry that is used as a core reset for the 7 series FPGAs Transceivers Wizard core.

<component name>_example.srcs/sim_1/imports/simulation

The simulation directory contains the simulation test bench file provided with the Wizard wrapper.

Table 5-14: Simulation Directory

Name	Description
<project_dir>/<component name>_example.srcs/sim_1/imports/simulation	
<component_name>_tb.v [hd]	Test bench to perform functional simulation of the provided example design. See Functional Simulation Using the Vivado Design Tools, page 79 .
<component_name>_tb_imp.v [hd]	Test bench to perform timing simulation of the provided example design. See Timing Simulation Using the Vivado Design Tools, page 80 . This file is not generated for GTZ transceivers.

Example Design Description for GTX, GTH, and GTP Transceivers

The example design that is delivered with the wrappers helps designers understand how to use the wrappers and transceivers in a design. The example design is shown in Figure 5-1.

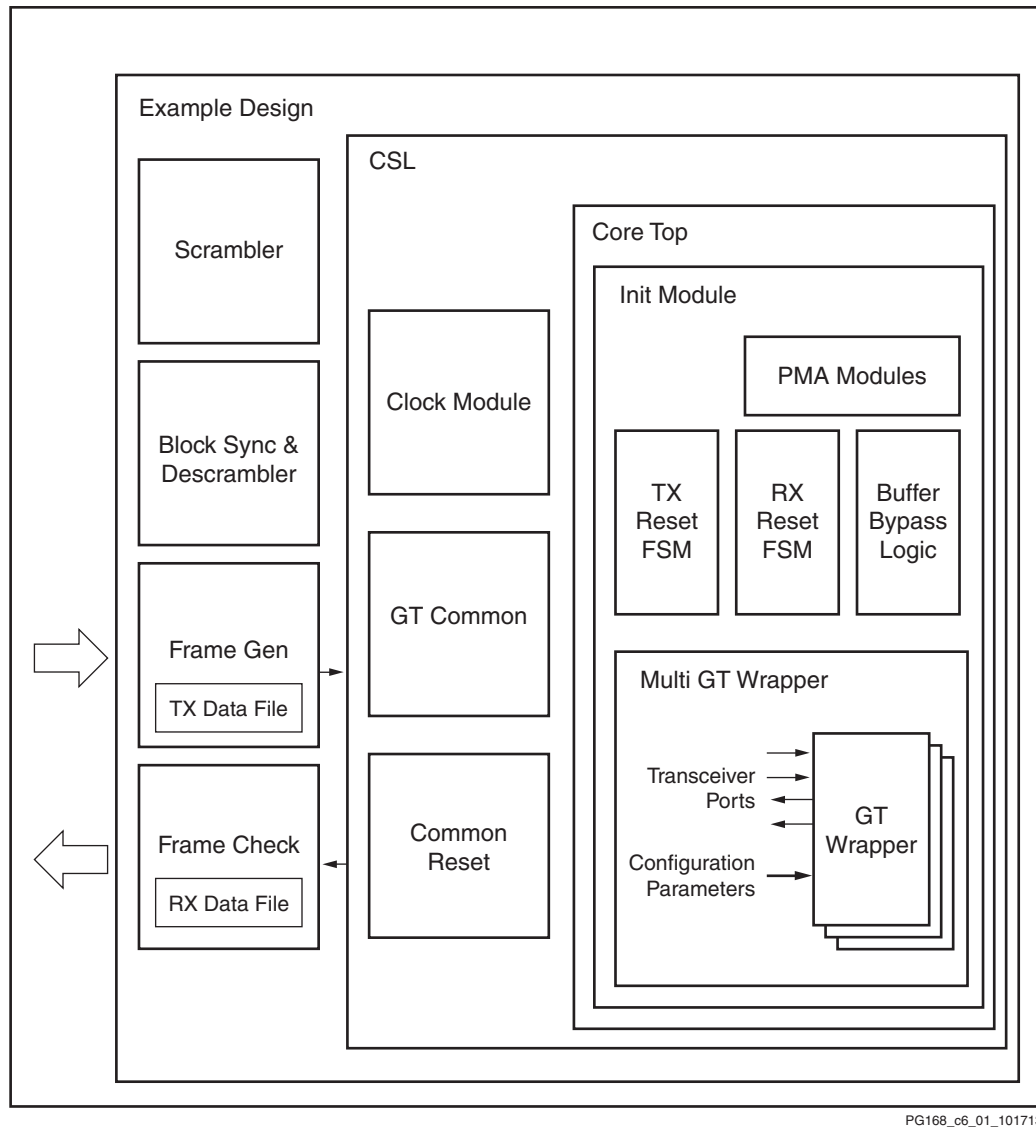


Figure 5-1: Diagram of Example Design and Test Bench

The example design connects a frame generator and a frame checker to the wrapper. The frame generator transmits an incrementing counting pattern while the frame checker monitors the received data for correctness. The frame generator counting pattern is stored in the block RAM. This pattern can be easily modified by altering the parameters in the `gt_rom_init_tx.dat` and `gt_rom_init_rx.dat` files. The frame checker contains the

same pattern in the block RAM and compares it with the received data. An error counter in the frame checker keeps a track of how many errors have occurred.

If comma alignment is enabled, the comma character will be placed within the counting pattern. Similarly, if channel bonding is enabled, the channel bonding sequence would be interspersed within the counting pattern.

The frame check works by first scanning the received data for the `START_OF_PACKET_CHAR`. In 8B/10B designs, this is the comma alignment character. After the `START_OF_PACKET_CHAR` has been found, the received data will continuously be compared to the counting pattern stored in the block RAM at each `RXUSRCLK2` cycle. After comparison has begun, if the received data ever fails to match the data in the block RAM, checking of receive data will immediately stop, an error counter will be incremented and the frame checker will return to searching for the `START_OF_PACKET_CHAR`.

The example design also demonstrates how to properly connect clocks to transceiver ports `TXUSRCLK`, `TXUSRCLK2`, `RXUSRCLK` and `RXUSRCLK2`. Properly configured clock module wrappers are also provided if they are required to generate user clocks for the instantiated transceivers. The logic for scrambler, descrambler, and block synchronization is instantiated in the example design for 64B/66B and 64B/67B encoding.

The example design can be synthesized and implemented using the Vivado design tools and then observed in hardware using the Vivado Design Suite debug feature. RX output ports such as `RXDATA` can be observed on the Vivado Design Suite debug feature ILA core while input ports can be controlled from the Vivado Design Suite debug feature VIO core.

For the example design to work properly in simulation, both the transmit and receive side need to be configured with the same encoding and datapath width in the GUI. In addition, the example design contains the initialization module, which consists of two independent finite state machines (`tx_startup_fsm` and `rx_startup_fsm`) and the `recclk_monitor` block.

Reset Finite State Machine

The intent for the reset FSM included in the example design is to provide:

- An application example of an initialization FSM meeting the requirements described in *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476) [Ref 7].
- An example of an initialization FSM that addresses industry challenges to initialize the GT transceivers in scenarios such as post FPGA-configuration and RX data interruption or replacement (such as cable plug or unplug).

The reset FSM is constantly being improvised to provide a robust initialization and reset scheme. The FSM is to demonstrate the right methodology and should not be mistaken as a specification.

The `tx_startup_fsm` is illustrated on the left side of [Figure 5-2](#).

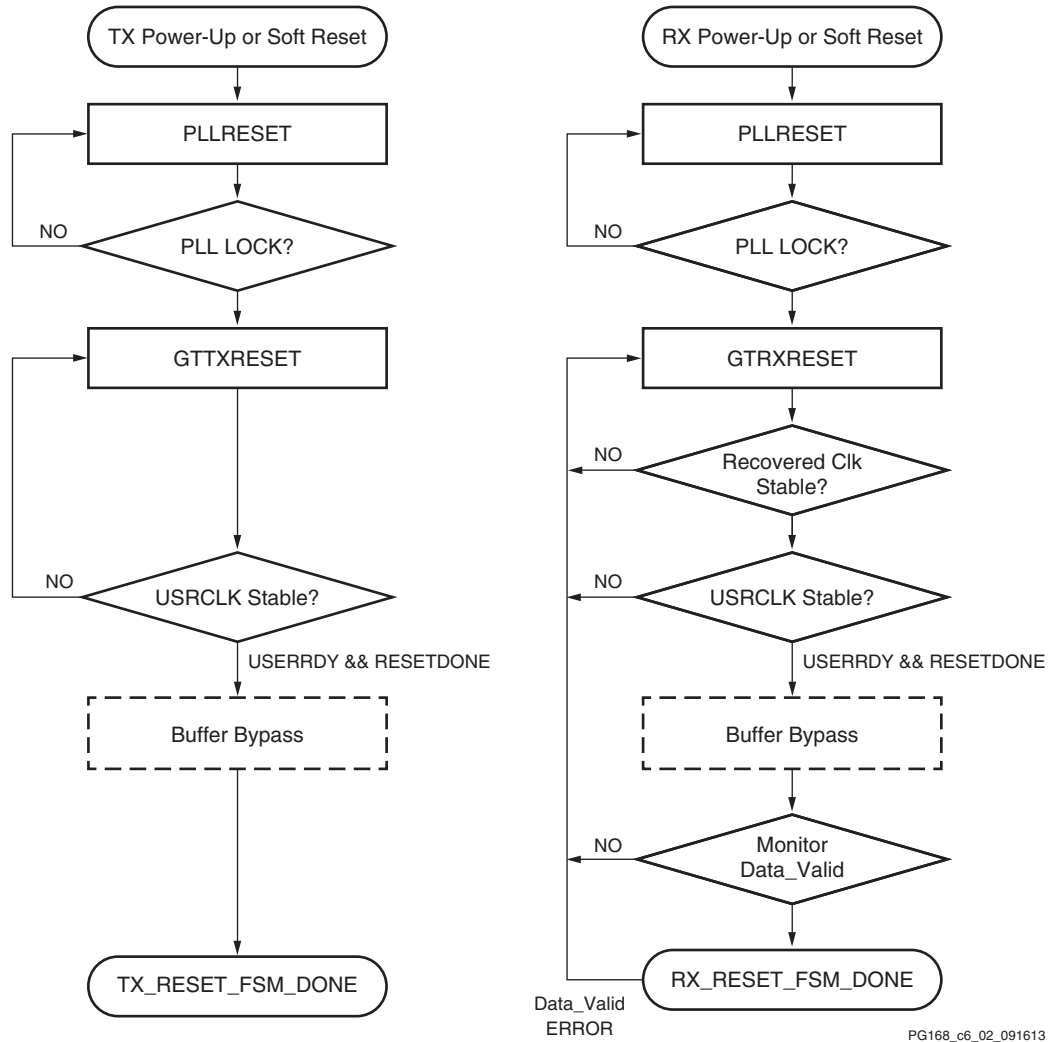


Figure 5-2: Diagram of Simplified FSM

Note: If RX and TX are combined in a core, only the TX FSM will reset the PLL. If RX-Only Core is generated, then RX FSM will reset the PLL.

The C/QPLL lock is monitored along with TXUSRCLK stability prior to TXRESETDONE. Buffer bypass logic for phase alignment is implemented if the TX buffer is disabled.

The rx_startup_fsm is also illustrated on the right side of Figure 5-2. The C/QPLL lock, recovered clock stability, and RXUSRCLK are examined prior to RXRESETDONE followed by the buffer bypass logic for phase alignment. The FSM finally stays at the state that monitors data validity, which can be an 8B/10B error, frame sync error, or CRC from the user design until a user-defined error occurs.

The rx_startup_fsm can reset the RX side of the transceiver if data_valid is lost between receives (possibly due to cable pull up). This can be enabled by setting the port DONT_RESET_ON_DATA_ERROR. If DONT_RESET_ON_DATA_ERROR = 1'b0, the FSM auto resets if an error is detected.

Table 5-15: Port Descriptions

S.No	Port Name	Clock Domain	Description
1	SOFT_RESET_TX_IN	sysclk_in	Active High reset to reset the TX Startup FSM and to start the TX initialization.
2	SOFT_RESET_RX_IN	sysclk_in	Active High reset to reset the RX Startup FSM and to start the RX initialization.
3	GT<n>_DATA_VALID_IN	sysclk_in	Indicates that data is valid on the RX side.
4	GT<n>_TX_FSM_RESET_DONE_OUT	sysclk_in	Indicates TX initialization is complete.
5	GT<n>_RX_FSM_RESET_DONE_OUT	sysclk_in	Indicates RX initialization is complete.
6	DON'T_RESET_ON_DATA_ERROR_IN	Asynchronous	Used to auto-reset the RX side if an error is detected. If DONT_RESET_ON_DATA_ERROR = 1'b0, the FSM auto-resets when an error is detected.

These are some assumptions and notes for the Example Reset FSM:

- A stable REFCLK is assumed to be present at all times.
- All resets are assumed to be in Sequential mode. The reset FSMs run on SYSCLK, which is the same as the DRPCLK. If the SYSCLK and DRPCLK are not the same in the user design, care should be taken to add the appropriate synchronizers.
- The example design engaged additional gates available such as PLLREFCLKLOST and Wait-time. Wait-time in use should not be regarded as the specification.
- RECCLK_STABLE is used as an indicator of RXOUTCLK (recovered clock) stability within a configured PPM offset from the reference clock (default 5000 ppm). The appropriate T_{DLOCK} is used. See the *Virtex-7 FPGAs Data Sheet: DC and Switching Characteristics* (DS183) [Ref 1].
- The example design defaults FRAME_CHECKER as data_valid. You can identify your data valid indicator and provide the necessary hysteresis on this signal to avoid a false indication of data_valid. Data valid is only used as an indicator to monitor the RX link. You need to customize the data valid indicator based on your system design.
- You can modify or re-invent an FSM to meet your specific system requirements while adhering to the guidelines in the *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476) [Ref 7].
- There might be other restrictions in answer records or errata.

CTLE3 Adaptation Modules for GTX Transceivers

CTLE AGC Comp – CTLE3 Adaptation

This block compensates for the fact that the 7 series FPGA GTX transceiver does not have adaptive logic for tuning CTLE3. The purpose of the block is to adjust CTLE3 gain to keep AGC from railing. The block uses channel DRP to set CTLE3 values and the RXMONITOR ports to observe AGC. This block is activated whenever you assert a GTRXRESET, RXPMARESET, or RXDFELPMRESET (actual operation begins at the deassertion of any of these reset signals).

Note: You do not have control of the DRP interface as long as the ADAPT_DONE signal coming out of these blocks is not High.

Example Design Hierarchy

The hierarchy for the design used in this example is:

```

EXAMPLE_TB
|__XAU_WRAPPER_EXDES
|   |__XAU_WRAPPER_INIT
|       |__TX_STARTUP_FSM           (1 per transceiver)
|       |__RX_STARTUP_FSM           (1 per transceiver)
|       |__RECCLK_MONITOR
|       |__XAU_WRAPPER
|           |__XAU_WRAPPER_GT       (1 per transceiver)
|           |__XAU_WRAPPER_GT_FRAME_GEN (1 per transceiver)
|           |__XAU_WRAPPER_GT_FRAME_CHECK
|           |__XAU_WRAPPER_GT_USRCLK_SOURCE
|           |__XAU_WRAPPER_CLOCK_MODULE

```

Reset Sequence Modules for GTH and GTP Transceivers

GTH and GTP transceivers need additional reset sequencing for production silicon. The reset sequencers for `GTRXRESET`, `RXPARESET`, and `RXRATE` are instantiated in the `<component_name>_gt.v[hd]` module. These modules need exclusive access to the DRP interface of the transceiver. The `DRP_BUSY_OUT` signal that is synchronous to the `DRPCLK` indicates that you cannot access the DRP interface. You should implement a DRP arbitration scheme to ensure smooth operation of the various reset sequencers. In the current design, the `GTRXRESET` sequencer takes highest priority.

Example Design Description for GTZ Transceivers

The example design that is delivered with the wrappers helps designers understand how to use the wrappers and transceivers in a design.

The example design connects a frame generator and a frame checker to the wrapper. The frame generator transmits an incrementing counting pattern while the frame checker monitors the received data for correctness. The frame generator counting pattern is stored in the block RAM. This pattern can be easily modified by altering the parameters in the `gt_rom_init_tx.dat` and `gt_rom_init_rx.dat` files. The frame checker contains the same pattern in the block RAM and compares it with the received data. An error counter in the frame checker keeps track of how many errors have occurred.

The example design also demonstrates how to generate the USRCLKs out of the OUTCLKs coming out of an octal. Also note the connections of USRCLKs for octal0 and octal1. Properly configured clock module wrappers are also provided if they are required to generate user clocks for the instantiated transceivers. The logic for scrambler, descrambler, and block synchronization is instantiated in the example design for 64B/66B. The `frame_gen_top` module also shows how to drive the txsequence counter and how to pause the data and header controls based on the sequence counter values. The USRCLK source module shows how and when an MMCM is required to generate a USRCLK.

The example design can be synthesized using XST or the Vivado tools and implemented with the Vivado tools. It can be simulated using VCS or Cadence Incisive Unified Simulator (IUS).

Reset and CTLE Tuning

The example design also demonstrates how to reset GTZ and enable the CTLE tuning for a given channel in an octal. It is essential to enable CTLE tuning to receive clean data on the serial interface of the GTZ transceiver in internal PMA loopback. The `<Component_name>_pll_gtx_tx_rx_ctle.v` module demonstrates an implementation of the procedure documented in the *7 Series FPGAs GTZ Transceivers User Guide* (UG478) [Ref 12]. This module is instantiated once for each octal instantiation because it uses the DRP interface associated with an octal to make enables and monitor the CTLE tuning.

The interface for this module is shown in [Table 5-16](#).

Table 5-16: CTLE Tuning Module Ports

Ports	Direction	Description
FIBRESET	OUTPUT	Resets the RX FIB part of every GTZ in an octal.
FSM_DONE	OUTPUT	Indicates that TX and RX are out of reset and GTZ is ready to use.
RD	OUTPUT	The DRP interface port to be connected to the corresponding signal on the octal.
WR	OUTPUT	The DRP interface port to be connected to the corresponding signal on the octal.
DRPADDR_O	OUTPUT	The DRP interface port to be connected to the corresponding signal on the octal.
DATA	OUTPUT	The DRP interface port to be connected to the corresponding signal on the octal.
TXEN	OUTPUT	Should be connected to TXEN of each GTZ. It Enable the TX SerDes for the lane.
RXEN	OUTPUT	Should be connected to RXEN of each GTZ. It Enable the RX SerDes for the lane.
GTZTXRESET	OUTPUT	Resets both FIB and SerDes on TX side.
GTZRXRESET	OUTPUT	Resets both FIB and SerDes on RX side.

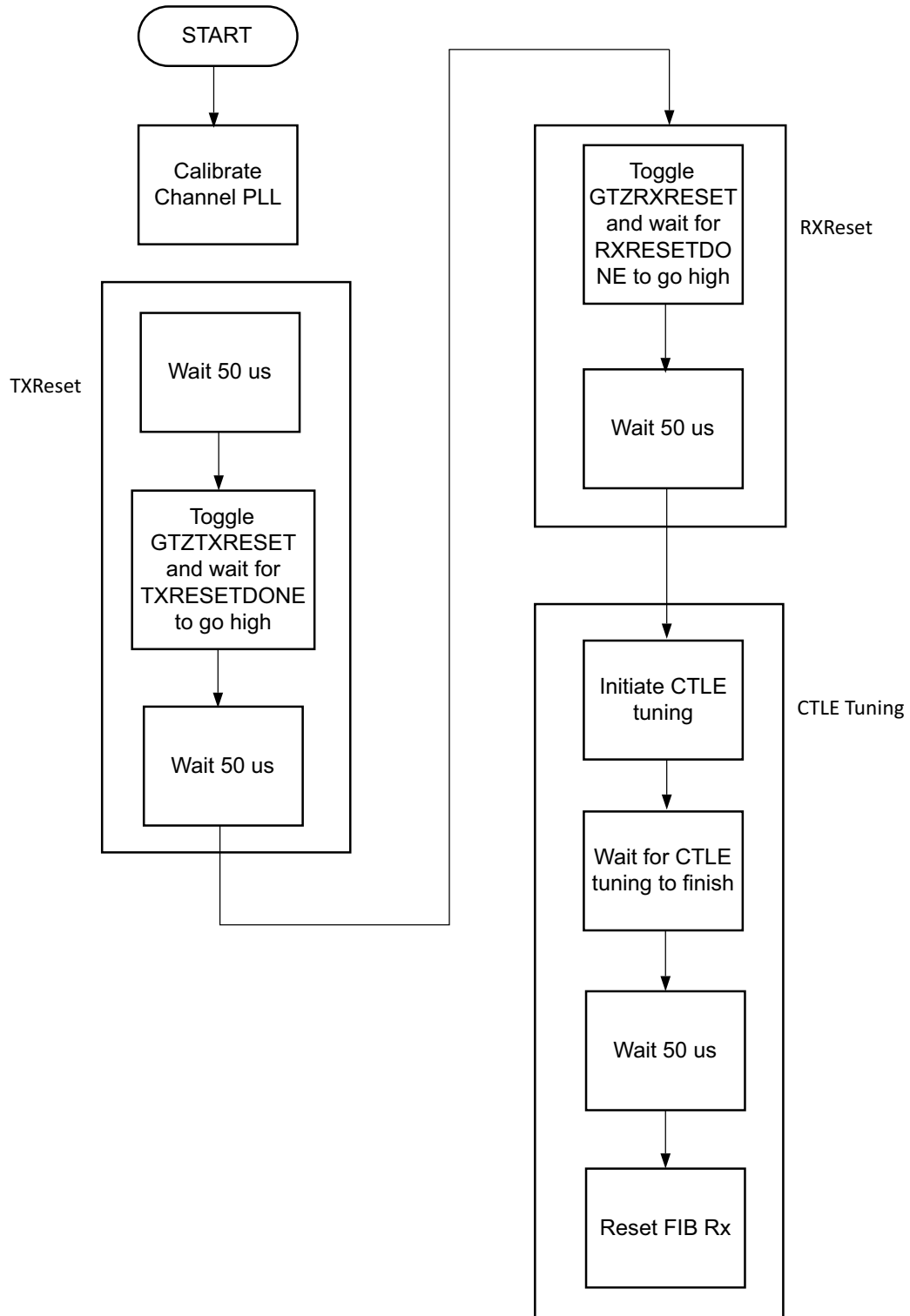
Table 5-16: (Cont'd)CTLE Tuning Module Ports

Ports	Direction	Description
PLLRECAL	OUTPUT	Recalibrates the PLL if either TXEN or RXEN is asserted
CLK	INPUT	Clk to the module.
RESET	INPUT	Resets the entire state machine.
TXRESETDONE	INPUT	Indicates that TX has come out of reset and is ready to use.
RXRESETDONE	INPUT	Indicates that RX has come out of reset and is ready to use.
TXRDY	INPUT	Indicates that TX PLL is locked.
RXRDY	INPUT	Indicates that RX PLL is locked.
TUNING_ADDR1	INPUT	The address for checking the status of CTLE tuning.
TUNING_ADDR2	INPUT	The address for setting the CTLE tuning mode and enabling it.
DRPRDY	INPUT	The DRP interface port to be connected to the corresponding signal on the octal.
DRPDO	INPUT	The DRP interface port to be connected to the corresponding signal on the octal.
REDO_CTLE_TUNING	INPUT	When enabled, the tuning enable FSM re-enables the CTLE tuning.
DISABLE_TUNING	INPUT	Disables tuning for that lane.
REDO_RXRESET_CTLE_TUNING	INPUT	When enabled, the tuning enable FSM re-enables the RXRESET and also CTLE tuning.
REDO_PLL_RESET	INPUT	When enabled, only PLL reset is performed.

The `<component_name>_pll_gtz_tx_rx_ctle.v` module performs the necessary reset sequence for GTZ and also CTLE tuning. The `FSM_DONE` signal indicates that the transceiver is out of reset and CTLE tuning is done.

To tune individual lanes, you can enable the `REDO_CTLE_TUNING` port for the particular lane. In the `<Component_name>_init.v` file, change the `oct<octal>_lane_select_i` signal to the particular lane.

Figure 5-3 and Figure 5-4 show the Reset and CTLE tuning enable state machine.



X15001-091015

Figure 5-3: Reset and CTLE Tuning Enable State Machine

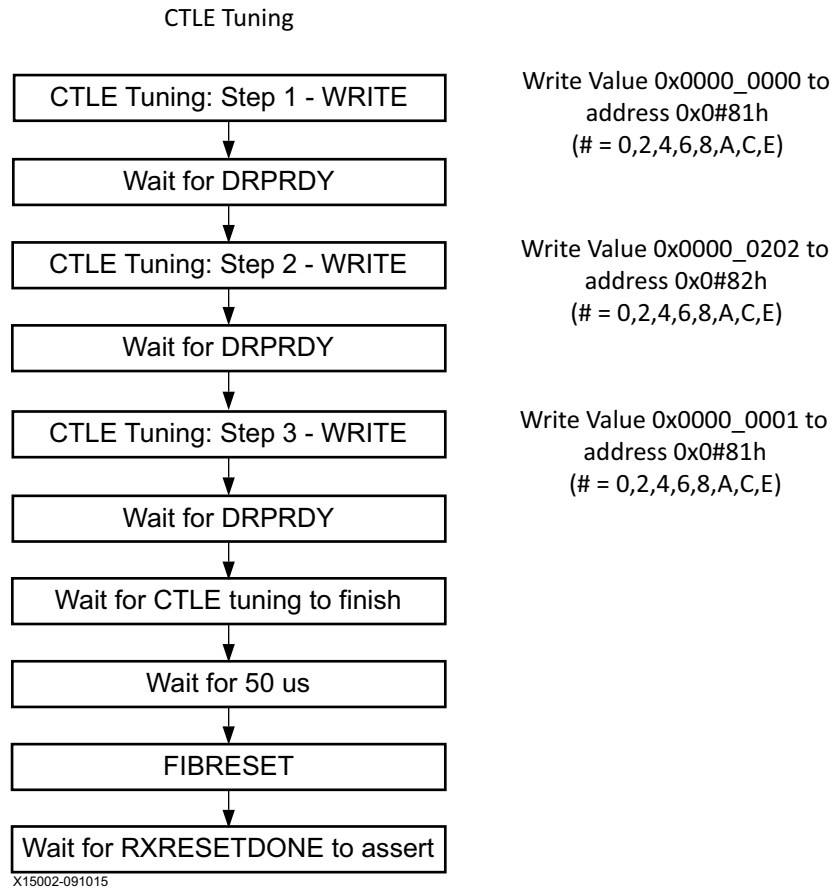


Figure 5-4: Continued Reset and CTLE Tuning Enable State Machine

Beachfront Module

The beachfront module must be used along with every instantiation of the GTZE2_OCTAL primitive. This module guarantees the timing of all the synchronous signals across the GTZ transceiver and FPGA logic interface by routing them through pre-locked flip-flops and LUTs. Each input and output signal of the GTZE2_OCTAL passes through this module. Each of the synchronous signals are passed through flip-flops and LUTs that are pre-LOCed.

In addition, the beachfront also has the capability to instantiate the BUFG_LB primitives to generate USERCLKs from OUTCLKs. This functionality is controlled by the parameters listed in Table 5-17. Apart from these parameters, all other parameters seen on the beachfront are the same as the ones that are used on the GTZE2_OCTAL primitive. For more information, see the *7 Series FPGAs GTZ Transceivers User Guide (UG478)* [Ref 12].

Table 5-17: TXUSRCLK Attributes on Beachfront

Parameter	Description
TXUSRCLK_SEL_MUX0	<ul style="list-style-type: none"> TXUSRCLK0: Selects the incoming (from FPGA logic) TXUSRCLK0 port to pass on to the GTZE2_OCTAL primitive and also use the same to clock the ports in beachfront. You should take the TXOUTCLK0 output from the beachfront and use an MMCM or a BUFG_LB to generate the user clock. TXCORECLK0: Instantiates a BUFG_LB primitive whose input is TXOUTCLK0 and output is TXCORECLK0. The output is passed on to the GTZE2_OCTAL parameter as the TXUSRCLK0, and the same is also used internally to clock beachfront flip-flops. You should use TXCORECLK0 as TXUSRCLK0 in FPGA logic in this case.
TXUSRCLK_SEL_MUX1	<ul style="list-style-type: none"> TXUSRCLK1: Selects the incoming (from FPGA logic) TXUSRCLK1 port to pass on to the GTZE2_OCTAL primitive and also use the same to clock the ports in beachfront. You should take the TXOUTCLK1 output from the beachfront and use an MMCM or a BUFG_LB to generate the user clock. TXCORECLK1: Instantiates a BUFG_LB primitive whose input is TXOUTCLK1 and output is TXCORECLK1. The output is passed on to the GTZE2_OCTAL parameter as the TXUSRCLK1, and the same is also used internally to clock beachfront flip-flops. You should use the TXCORECLK1 as TXUSRCLK1 in the FPGA logic in this case.

Table 5-18 shows the RXUSRCLK attributes on the beachfront module.

Table 5-18: RXUSRCLK Attributes on Beachfront

Parameter	Description
RXUSRCLK_SEL_MUX0	<ul style="list-style-type: none"> RXUSRCLK0: Selects the incoming (from FPGA logic) RXUSRCLK0 port to pass on to the GTZE2_OCTAL primitive and also use the same to clock the ports in beachfront. You should take the RXOUTCLK0 output from the beachfront and use an MMCM or a BUFG_LB to generate the user clock. RXCORECLK0: Instantiates a BUFG_LB primitive whose input is RXOUTCLK0 and output is RXCORECLK0. The output is passed on to the GTZE2_OCTAL parameter as the RXUSRCLK0, and the same is also used internally to clock beachfront flip-flops. You should use the RXCORECLK0 as RXUSRCLK0 in FPGA logic in this case.
RXUSRCLK_SEL_MUX1	<ul style="list-style-type: none"> RXUSRCLK1: Selects the incoming (from FPGA logic) RXUSRCLK1 port to pass on to the GTZE2_OCTAL primitive and also use the same to clock the ports in beachfront. You should take the RXOUTCLK1 output from the beachfront and use an MMCM or a BUFG_LB to generate the user clock. RXCORECLK1: Instantiates a BUFG_LB primitive whose input is RXOUTCLK1 and output is RXCORECLK1. The output is passed on to the GTZE2_OCTAL parameter as the RXUSRCLK1, and the same is also used internally to clock beachfront flip-flops. You should use the RXCORECLK1 as RXUSRCLK1 in FPGA logic in this case.

Table 5-18: **RXUSRCLK Attributes on Beachfront (Cont'd)**

Parameter	Description
RXUSRCLK_SEL_MUX2	<ul style="list-style-type: none"> • RXUSRCLK2: Selects the incoming (from FPGA logic) RXUSRCLK2 port to pass on to the GTZE2_OCTAL primitive and also use the same to clock the ports in beachfront. You should take the RXOUTCLK2 output from the beachfront and use an MMCM or a BUFG_LB to generate the user clock. • RXCORECLK2: Instantiates a BUFG_LB primitive whose input is RXOUTCLK1 and output is RXCORECLK2. The output is passed on to the GTZE2_OCTAL parameter as the RXUSRCLK2, and the same is also used internally to clock beachfront flip-flops. You should use the RXCORECLK2 as RXUSRCLK2 in FPGA logic in this case.
RXUSRCLK_SEL_MUX3	<ul style="list-style-type: none"> • RXUSRCLK3: Selects the incoming (from FPGA logic) RXUSRCLK3 port to pass on to the GTZE2_OCTAL primitive and also use the same to clock the ports in beachfront. You should take the RXOUTCLK3 output from the beachfront and use an MMCM or a BUFG_LB to generate the user clock. • RXCORECLK3: Instantiates a BUFG_LB primitive whose input is RXOUTCLK3 and output is RXCORECLK3. The output is passed on to the GTZE2_OCTAL parameter as the RXUSRCLK3, and the same is also used internally to clock beachfront flip-flops. You are supposed to use the RXCORECLK3 as RXUSRCLK3 in FPGA logic in this case.

Input ports to the beachfront from the FPGA logic are prefixed with B2M_. Output ports from the beachfront going to the FPGA logic are prefixed with M2B_. Thus, you should always use either the M2B_ or B2M_ ports of the beachfront to connect to the GTZE2_OCTAL. Apart from the ports listed in Table 5-19, all other ports on the beachfront interface are the same as the ones seen in the GTZE2_OCTAL primitive. For more information, see the *7 Series FPGAs GTZ Transceivers User Guide (UG478)* [Ref 12].

Table 5-19: **TX/RXCORECLK Ports on Beachfront**

Port	Direction	Description
TXCORECLK0	Output	This port should be used as TXUSRCLK0 if the parameter TXUSRCLK_SEL_MUX0 is set to TXCORECLK0; else this port should be ignored.
TXCORECLK1	Output	This port should be used as TXUSRCLK1 if the parameter TXUSRCLK_SEL_MUX1 is set to TXCORECLK1; else this port should be ignored.
TXCORECLK0	Output	This port should be used as TXUSRCLK1 if the parameter TXUSRCLK_SEL_MUX1 is set to TXCORECLK1; else this port should be ignored.
RXCORECLK0	Output	This port should be used as RXUSRCLK0 if the parameter RXUSRCLK_SEL_MUX0 is set to RXCORECLK0; else this port should be ignored.
RXCORECLK1	Output	This port should be used as RXUSRCLK1 if the parameter RXUSRCLK_SEL_MUX1 is set to RXCORECLK1; else this port should be ignored.

Table 5-19: TX/RXCORECLK Ports on Beachfront

Port	Direction	Description
RXCORECLK2	Output	This port should be used as RXUSRCLK2 if the parameter RXUSRCLK_SEL_MUX2 is set to RXCORECLK2; else this port should be ignored.
RXCORECLK3	Output	This port should be used as RXUSRCLK3 if the parameter RXUSRCLK_SEL_MUX3 is set to RXCORECLK3; else this port should be ignored.

Dynamic Phase Deskew

The example design generated by the CAUI4 protocol template of the Wizard also includes a dynamic phase deskew logic within it to ensure the lane-to-lane skew is minimized. This logic is found in the `gt_usrclk_source.v` module generated in the `example_design` folder. This logic, with the help of the MMCM, adjusts the phase of the RXFIFO read clock to get the maximum margin on the right side without losing too much on the left margin.

- You need to set the M, D, and O values of the MMCM to get the required VCO frequency and resolution. The resolution is $1/56 F_{VCO}$.
- The MMCM uses the dynamic phase variation feature. An FSM controls the phase shift of USRCLK using the phase shift interface.
- Shifting starts whenever RXRESETDONE is asserted from all channels. The Low-to-High transition of the RXRESETDONE signal triggers the adjustment again.
- You should assert the RXFIBRESET once after the MMCM is locked. This ensures that the RX FIFO is reset after the phase adjustments are done on the read clock.
- Each channel has different left and right margins, but it is guaranteed to be above $(3.1 + 1.5)$ when operated at 25.8 Gb/s.
- Whenever the link goes down and is later re-established using a new training sequence, it is possible to get a different write clock phase. This training sequence is always followed by an RXFIFO reset to bring the FIFO to a normal operating condition. In that case, RXRESETDONE to the deskew logic toggles and re-initiates the phase adjustment.

Multi-Lane Mode

The Wizard enables multi-lane mode and puts the transceiver in master-slave mode by default whenever the CAUI4 protocol template is selected. This is essential to reduce the lane-to-lane deskew. For more information on how to use the multi-lane mode, see the *7 Series FPGAs GTZ Transceivers User Guide (UG478)* [Ref 12].

Using The Example Design in Hardware

Follow these steps to use the example design in hardware.

1. Set EXAMPLE_SIMULATION to 0.
2. Bring out the `soft_reset` signal to a pushbutton and `track_data` to LEDs.

Example Design Hierarchy

The hierarchy for the design used in this example is:

```
EXAMPLE_TB
|___CAUI4_WRAPPER_EXDES
|   |___CAUI4_INIT
|   |   |___CAUI4_WRAPPER
|   |   |   |___CAUI4_WRAPPER_OCTAL0 (1 per octal)
|   |   |   |___GTWIZARD_V3_4_BEACHFRONT (1 per octal)
|   |   |___CAUI4_WRAPPER_CTLE_TUNING (1 per octal)
|   |   |___CAUI4_WRAPPER_RX_STARTUP_FSM (1 per channel)
|   |___CAUI4_WRAPPER_GT_FRAME_GEN_TOP (1 per channel)
|   |   |___CAUI4_WRAPPER_GT_FRAME_GEN
|   |   |___CAUI4_WRAPPER_SCRAMBLER (5 per channel)
|   |___CAUI4_WRAPPER_GT_FRAME_CHECK_TOP (1 per channel)
|   |   |___CAUI4_WRAPPER_GT_FRAME_CHECK (5 per channel)
|   |   |___CAUI4_WRAPPER_DESCRAMBLER (5 per channel)
|   |   |___CAUI4_WRAPPER_BLOCK_SYNC_SM (5 per channel)
|   |___CAUI4_WRAPPER_GT_USRCLK_SOURCE (contains MMCM and dynamic phase deskew)
```

Known Limitations of the GTZ Wizard

- Dynamic phase deskew is necessary to reduce the lane-to-lane deskew for multi-lane protocols like CAUI4 operating at greater than 25G per line. You are required to reuse the code as shown in the `<component_name>_gt_usrclk_source.v` module to achieve the same.
- Even though `RXFIBRESET` is listed as an optional port, this is always brought out to the top level of the example design to enable users to control the RX FIFO reset for operations like CTLE tuning and dynamic phase deskew logic.

Known Limitations of the Wizard

- The 7 series FPGAs Transceivers Wizard core wrapper can be generated with asymmetrical data width from the Wizard but simulation support for asymmetrical data width is not supported for the core.
- Generation of the 7 series FPGAs Transceivers Wizard core with different encoding and decoding styles for RX and TX is removed from the Wizard.

- The simulation for Artix-7 device designs and some Virtex-7 device GTH designs takes more time to complete because the silicon work-around modules take more time to finish.

The 20-UI Square Wave figure in *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476) [Ref 7] shows the process of TX phase and delay alignment. This sequence is not followed by the exdes generated by the 7 series FPGAs Transceivers Wizard in simulation. Instead, the S_TXPHINIT line asserts at the same time as the M_TXPHINIT signal. Also contrary to the figure, the M_TXPHINITDONE signal only asserts when all of the other TXPHINITDONE signals have asserted. The slave TXPHINITDONE signals all assert at different times. There is a disconnect between the figure in *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476) and the actual operation of the Wizard example design. The 7 series FPGAs Transceivers Wizard example design simulation is correct.

Verification, Compliance, and Interoperability

The 7 series FPGAs Transceivers Wizard core is verified for protocol compliance using an array of automated hardware and simulation tests. The core comes with an example design implemented based on the selected protocol file and its respective configuration from the wizard for understanding and verification of the core features.

Simulation

An automated test system runs a series of simulation tests on the most widely used set of design configurations chosen at random. The transceiver cores are also tested in hardware for functionality, performance, and reliability using Xilinx GTX transceiver demonstration boards. Transceiver verification test suites for all possible modules are continuously being updated to increase the test coverage across the range of possible parameters for each individual module.

Hardware Testing

The boards used for verification are:

- KC705
- KC724
- VC7203

Upgrading

This appendix contains information about migrating a design from ISE® to the Vivado® Design Suite, and for upgrading to a more recent version of the IP 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.

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 14](#)].

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.

Prerequisites

- Vivado design tools build containing the 7 series FPGAs Transceivers Wizard v3.4.
- Familiarity with the 7 series FPGAs Transceivers Wizard directory structure.
- Familiarity with running the GT Wizard example design.
- The latest product guide (PG168) for the core.
- Migration guide (this appendix).

Overview of Major Changes

The major change to the core is the introduction of a new Core Support Level (CSL) which can be controlled by an option in the GUI to use shared logic at the core level or in the generated example design as shown in [Figure 4-1](#).

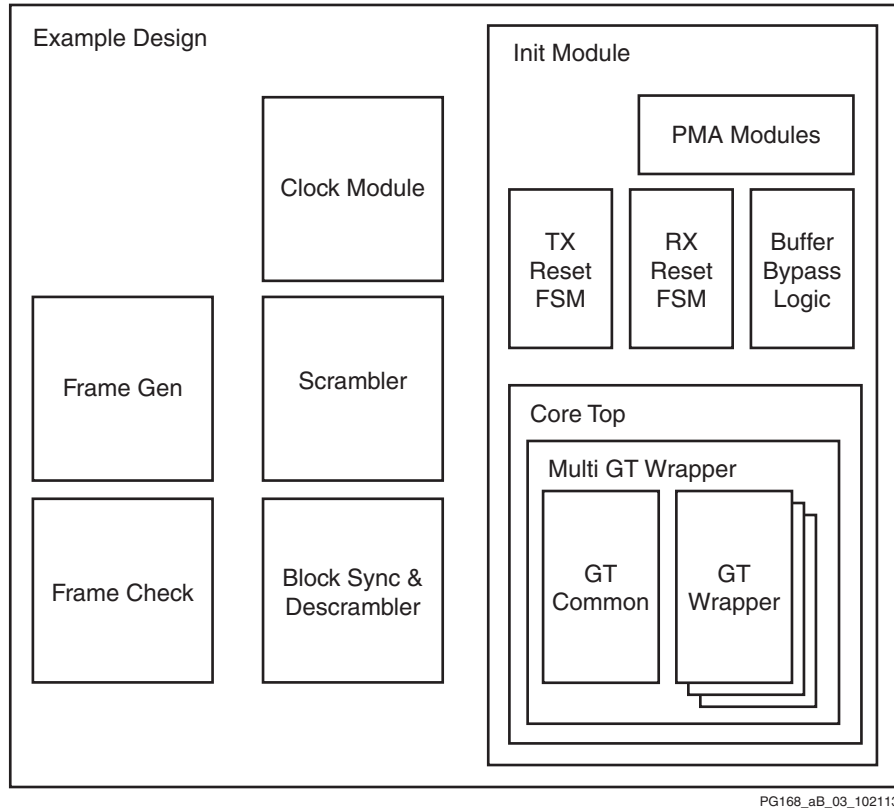
When the 7 series FPGAs Transceivers Wizard core is generated with the **include Shared Logic in core** option selected, the wrapper and RTL files for the core support level are available in the directory `/<project_name>/<component_name>.srcs/sources_1/ip/<component_name>/`.

When the 7 series FPGAs Transceivers Wizard core is generated with the **include Shared Logic in example design** option selected, the wrapper and RTL file for the core support level is available in the directory `/<project_name>/<component_name>_example/<component_name>_example.srcs/sources_1/imports/example_design/support`.

Also, the core support level wrapper file includes instantiations of `<component_name>_GT_USRCLK_SOURCE.v[hd]`, `<component_name>_common.v[hd]`, `<component_name>_common_reset.v[hd]`, and `<component_name>_init.v[hd]`. For GTX/GTH transceivers, QPLL can be shared between four different transceivers from the same Quad. A separate wrapper file (`<component_name>_common.v[hd]`) for GTXE2_COMMON/GTHE2_COMMON primitives gets generated and can be used for generating a QPLL reference clock for transceivers of the same Quad. For GTP transceivers, PLL0/PLL1 can be used across the four Quads, thus a separate wrapper file (`<component_name>_common.v[hd]`) for GTPE2_COMMON primitives gets generated.

Block Diagram

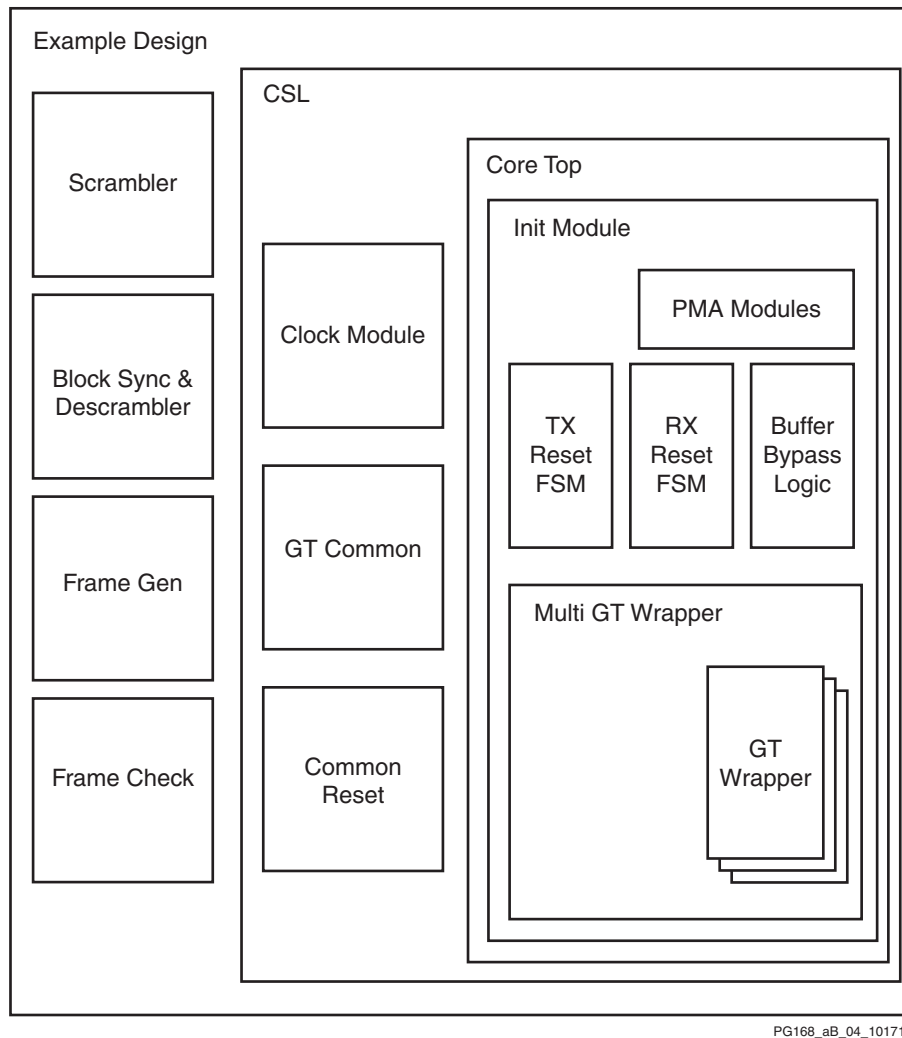
Figure B-1 shows a 7 series FPGAs Transceivers Wizard example design using the legacy core.



PG168_aB_03_102113

Figure B-1: Legacy Example Design

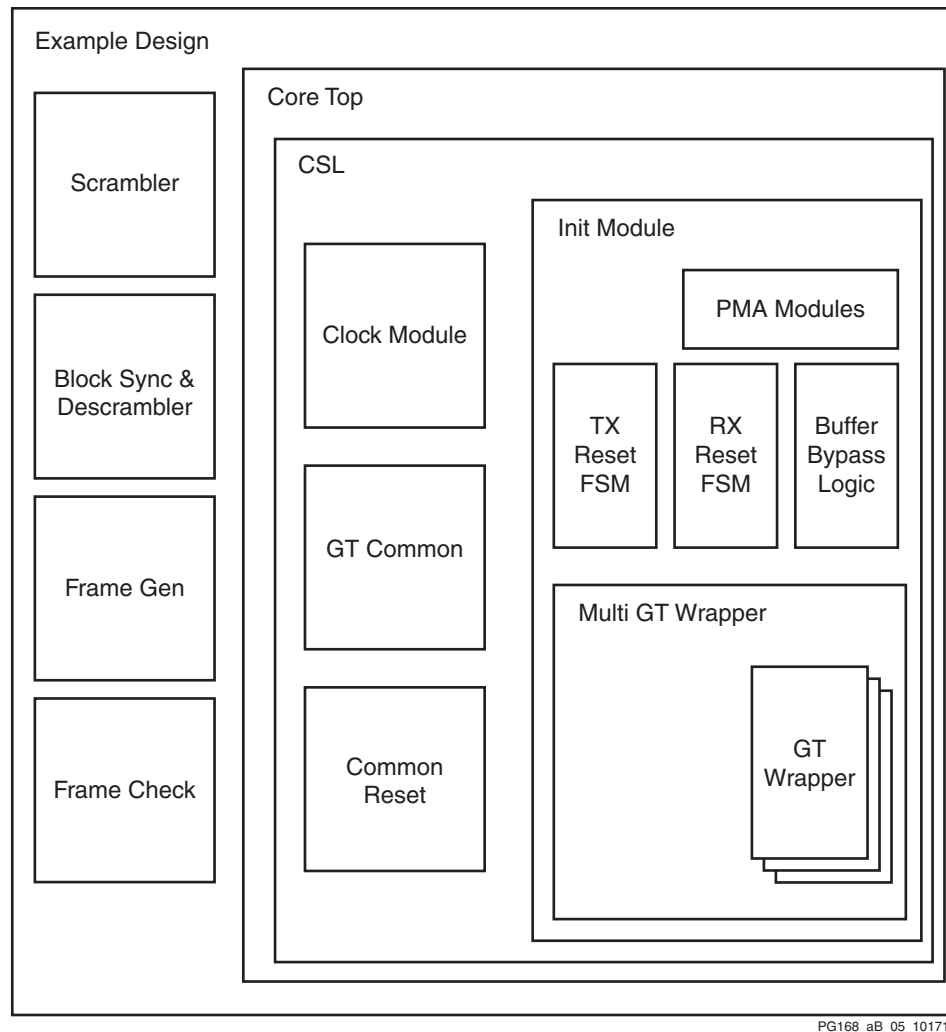
Figure B-2 is the 7 series FPGAs Transceivers Wizard v3.4 example design generated with the **include Shared Logic in example design** option selected in the GUI.



PG168_aB_04_101713

Figure B-2: Shared Logic in Example Design

Figure B-3 is the 7 series FPGAs Transceivers Wizard v3.4 example design generated with the **include Shared Logic in core** option selected in the GUI.



PG168_aB_05_101713

Figure B-3: Shared Logic in Core

Shared Logic Use Models with Multiple Instances

1. Generate one 7 series FPGA transceiver core with the **include Shared Logic in core** option (referred to as the master core throughout this document), and generate one more 7 series FPGA transceiver core with the **include Shared Logic in example design** option (referred to as the slave core throughout this document). The common module can be shared across four transceivers in a Quad. The cores can be connected together with the "shareable" outputs from the master connected to the slave cores. Name the master core `gtwizard_master` and the slave core `gtwizard_slave`.
2. Generate the example design for the master core `gtwizard_master`. This produces an example design with a pattern generator module and a checker module.

3. Generate the example design for the slave cores. This produces an example design containing a core support layer of the slave core `gtwizard_slave`.
4. Open the `<component_name>_support.v` file of the slave cores and copy the `<component_name>.v` of the slave cores instance and paste it twice into the core support level of the master core `gtwizard_master`. Close the slave example project.
5. You will now need to further edit the master example design and core support level as follows:
 - a. Bring all the ports of the slave core instances to the core support level of the master core except `txusrclk`, `txusrclk2`, `rxusrclk`, `rxusrclk2`, `drpclk`, `qplloutrefclk`, `qplloutclk` (for GTX/GTH transceivers) and `pll0outrefclk`, `pll0outclk`, `pll1outrefclk`, `pll1outclk` (for GTP transceivers) and `soft_reset`.
 - b. Connect `txusrclk`, `txusrclk2`, `rxusrclk`, `rxusrclk2`, `drpclk`, `qplloutrefclk`, and `qplloutclk` `soft_reset` of slave core instances to the same source as the master core instance.
 - c. Replicate the pattern generator and pattern checker instance in the example design of the master core and connect the signals of the slave cores that are brought to the core support level.
 - d. Create unique signals for the DRP interfaces on the slave cores and connect them in the same way as is done for the master core.

Suggestion: use `s1_drp_*` for slave 1 and `s2_drp_*` for slave 2

- e. Create unique signals for the `tx/rxresetdone_out` ports for the slaves and AND all slave `txresetdone_out` together with the master `txresetdone_out` output from the master. Similarly, AND all slave `rxresetdone_out` together with the master `rxresetdone_out` output from the master. A unique signal should be created for the master `resetdone_out` ports `m_txresetdone_out` and `m_rxresetdone_out` to feed out to the example design `resetdone_out` port.

Verilog example:

```
assign txresetdone_out = m_txresetdone_out && s1_txresetdone_out &&
s2_txresetdone_out;
```

```
assign rxresetdone_out = m_rxresetdone_out && s1_rxresetdone_out &&
s2_rxresetdone_out;
```

- f. In the module declaration for the example design, add new ports `TXN_S1`, `TXP_S1`, `RXN_S1`, `RXP_S1`, `TXN_S2`, `TXP_S2`, `RXN_S2`, and `RXP_S2` for the slave designs and connect them to the `TXP_OUT`, `TXN_OUT`, `RXP_IN`, and `RXN_IN` ports of the corresponding slave cores instances in the core support level.

Constraints

Alter the top-level XDC LOC to the GT locations. To get the paths for the GT locations, first elaborate the design then enter this Tcl command in the Vivado Tcl console:

```
get_cells -hierarchical -filter {NAME=~*gtxe2_i*}
```

This command is for a device containing GTXE2s and can be modified for gtp2_i or gthe2_i as appropriate.

Signal Changes

At the individual transceiver wrapper level:

Extra transceiver debug ports required for the protocol support have been brought out. For details on the extra ports, refer to [Table 2-5, page 18](#).

At the multi-transceiver wrapper level:

Extra transceiver debug ports required for the protocol support have been brought out as mentioned above, as well as extra ports from the GTXE2/GTHE2/GTPE2_COMMON modules that are required to be connected to the GTXE2/GTHE2/GTPE2_CHANNEL.

For GTX/GTH transceivers:

- qplloutclk
- qplloutrefclk

For GTP transceivers:

- pll0outclk
- pll0outrefclk
- pll1outclk
- pll1outrefclk

At the Init wrapper level:

Extra transceiver debug ports required for the protocol support have been brought out, as well as extra ports from the GTXE2/GTHE2/GTPE2_COMMON modules that are required to be connected to the GTXE2/GTHE2/GTPE2_CHANNEL as mentioned above and the PLL lock signals that will be used by the TX/RX start up FSMs.

For GTX/GTH transceivers:

- qplllock

For GTP transceivers:

- pll0lock
- pll1lock

Migration Steps

Generate the 7 series FPGAs Transceivers Wizard v3.4 from the Vivado tools 2014.3 IP catalog as described in [Chapter 4, Design Flow Steps](#).



RECOMMENDED: *Since the introduction of CSL and moving the common module from the multi-transceiver level, legacy 7 series FPGAs Transceivers Wizard IP users have to make some changes in their design depending on the wrapper file they are using. These changes are listed below.*

- Using individual transceiver wrapper files:

The user has to update the new ports added for transceiver debug in their design. For details on the new ports, refer to [Table 2-5, page 18](#).

- Using multi-transceiver wrapper files:

The user has to generate the wizard with the "shared logic in core" option, then manually instantiate the generated common file (`<component_name>_common.v[hd]`) in the design and update the new transceiver debug ports along with these ports:

For GTX/GTH transceivers: `qplloutclk`, `qplloutrefclk`, `qplllock`

For GTP transceivers: `pll0[1]outclk`, `pll0[1]outrefclk`, `pll0[1]lock`

- c) Using the Init wrapper file:

The user has to generate the wizard with the "shared logic in core" option, then manually instantiate the generated common file (`<component_name>_common.v[hd]`) in the design and update the new transceiver debug ports along with these ports:

For GTX/GTH transceivers: `qplloutclk`, `qplloutrefclk`, `qplllock`

For GTP transceivers: `pll0[1]outclk`, `pll0[1]outrefclk`, `pll0[1]lock`

Vivado Design Suite Debug Feature Option

The Vivado Design Suite debug feature option provided in the GUI allows 7 series FPGAs Transceivers Wizard users to include or exclude ILA and VIO debug cores in the IP while generating the core. [Figure B-4](#) shows new options added to the GUI for debugging and monitoring the core. For more information on how to debug and monitor in the 7 series FPGAs Transceivers Wizard core, refer to [Appendix C, Debugging](#).

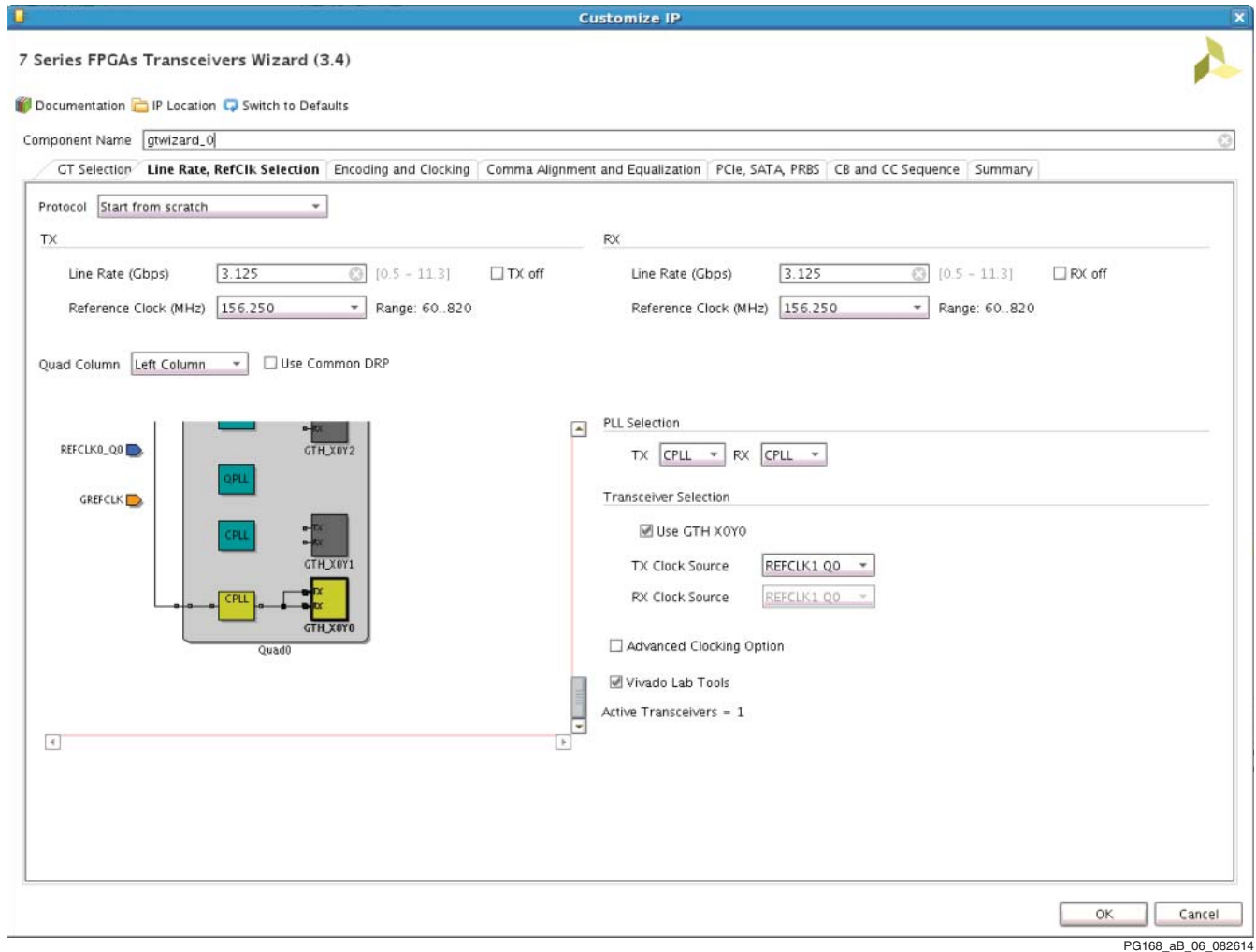


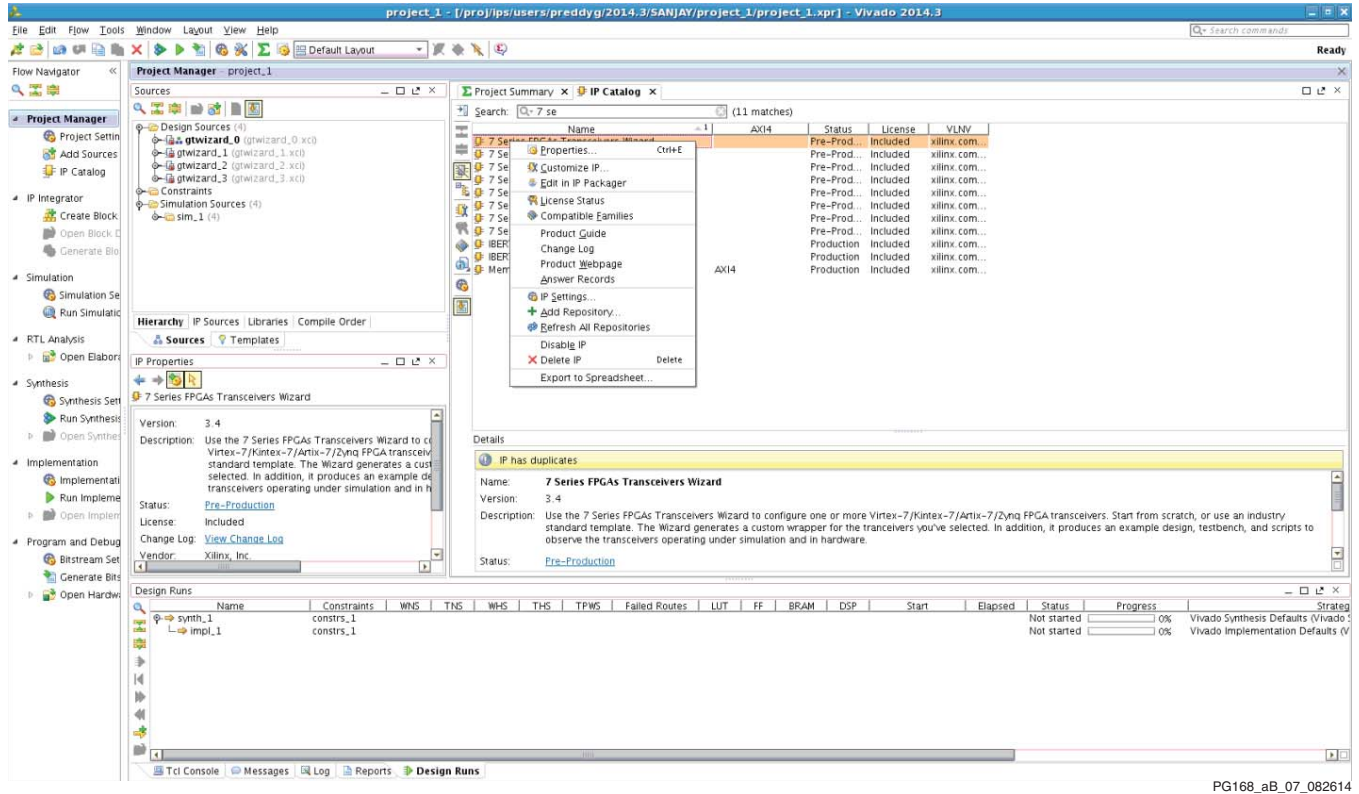
Figure B-4: Vivado Design Suite Debug Feature

Transceiver Core Debug

To provide additional control and debug visibility of the transceivers based on the protocol template selected from the GUI, the 7 series FPGAs Transceivers Wizard core is provided with debug ports required for transceiver core debug that are useful when debugging transceiver links. For more information on transceiver debug ports, refer to [Table 2-5, page 18](#). These control and debug ports allow Wizard users to fine tune the transceiver’s TX driver and RX equalization, in addition to other functions such as the ability to send/detect PRBS patterns. These ports allow you to tune the transceiver to your particular link.

Change Log Information

For more information on all current changes with the 7 series FPGAs Transceivers Wizard, refer to the change log information from the IP catalog as shown in [Figure B-5](#).



PG168_aB_07_082614

Figure B-5: Change Log

Parameter Changes

No changes.

Port Changes

No changes.

Other Changes

No changes.

Debugging

This appendix provides 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 7 series FPGAs Transceivers Wizard core, the [Xilinx Support web page](#) 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 [High Speed Serial](#) home page.

Documentation

This product guide is the main document associated with the 7 series FPGAs Transceivers Wizard 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](#) or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator on the [Downloads page](#). For more information about this tool and the features available, open the online help after installation.

Solution Centers

For support specific to the 7 series FPGAs Transceivers Wizard core, see the [Xilinx High-Speed Serial I/O Solution Center](#).

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 also 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.

Master Answer Record for the 7 series FPGAs Transceivers Wizard core

[AR: 54691](#)

Technical Support

Xilinx provides technical support in the [Xilinx Support web page](#) for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support if you do any of the following:

- Implement the solution in devices that are not defined in the documentation.
- Customize the solution beyond that allowed in the product documentation.
- Change any section of the design labeled DO NOT MODIFY.

To contact Xilinx Technical Support, navigate to the [Xilinx Support web page](#).

Debug Tools

There are many tools available to address 7 series FPGAs Transceivers Wizard core design issues.

Vivado Design Suite Debug Feature



RECOMMENDED: For debugging, Xilinx strongly recommends setting the synthesis option for `-flatten_hierarchy` to **none** or **rebuilt**. Enable the filtering according to hierarchy to preserve netlist hierarchy and easing the process of signal location.

Vivado® Design Suite debug feature inserts logic analyzer and virtual I/O cores directly into your design. The debug feature allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed. This feature represents the functionality in the Vivado IDE that is used for logic debugging and validation of a design running in Xilinx devices in hardware.

The Vivado logic analyzer is used to interact with the logic debug LogiCORE 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* [Ref 15].

Reference Boards

Various Xilinx development boards support the 7 series FPGAs Transceivers Wizard 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

Wizard Validation

This section provides an overview of the hardware setup for the validation of the Wizard, steps for initial bring up, scope of validation, and plan of validation. The purpose of this section is to provide an approach to start building a validation plan along with a hardware setup method to begin testing different features and combinations.

Wizard Validation Scope

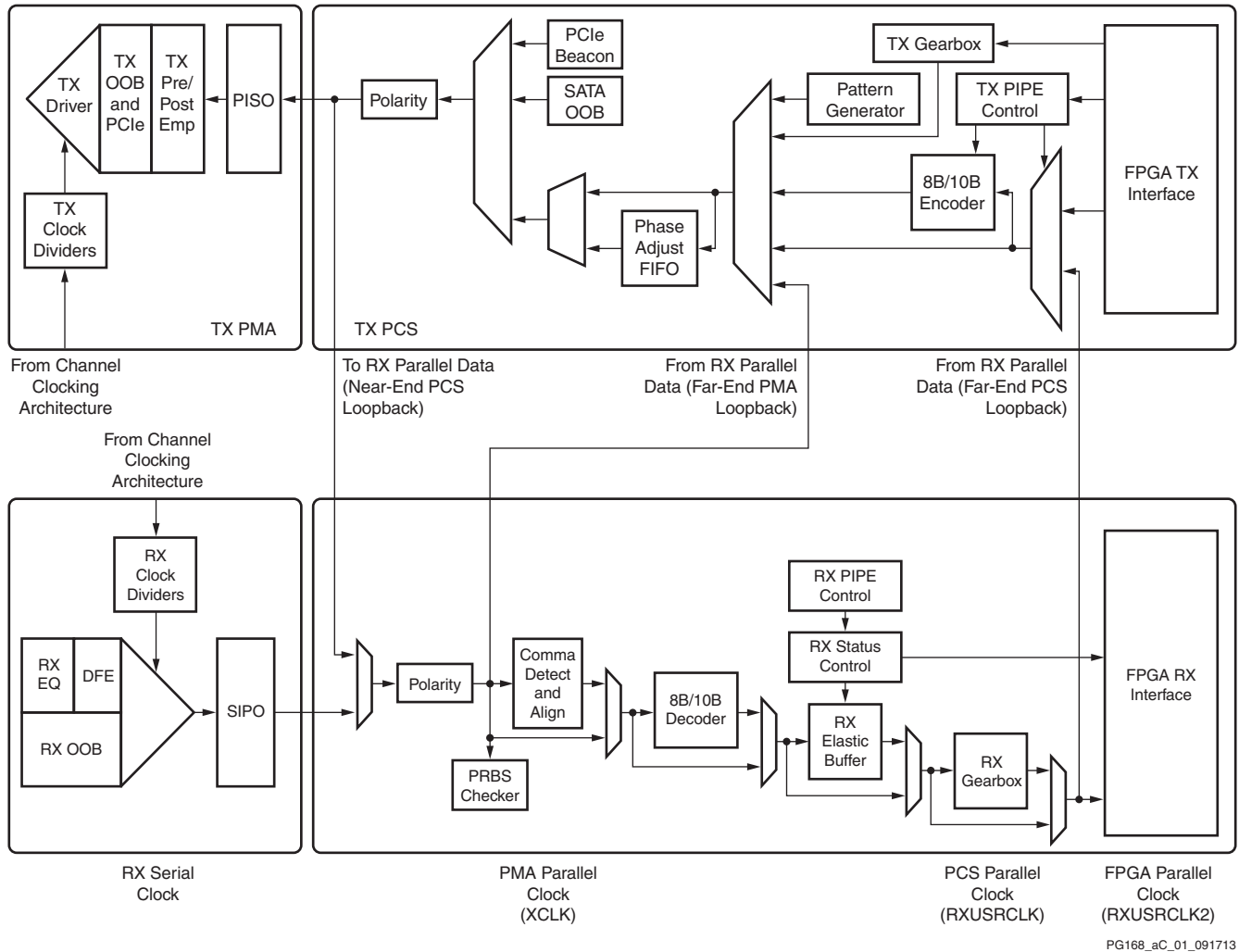


Figure C-1: Block Diagram of the 7 Series FPGA GTX Transceiver Channel

As shown in [Figure C-1](#), a GTX transceiver channel in the 7 series FPGA has two main blocks: transmitter and receiver. The PMA of each channel is configurable through attributes and DRP configuration whereas different submodules of PCS are configurable through ports and attributes. The validation of the Wizard will mostly focus on:

- [Datapath Testing](#)
- [Different PLL Testing](#)
- [Loopback Mode Testing](#)
- [PRBS Mode Testing](#)
- [TX/RX Buffer Testing](#)
- [REFCLK Testing](#)
- [Powerdown Testing](#)
- [TX/RX Rate-change Testing](#)
- [TX/RX Polarity Testing](#)
- [TX/RX Electrical Idle Testing](#)
- [Encoding and Decoding Testing](#)
- [Comma Detection Testing](#)
- [Comma Alignment Testing](#)
- [TX/RX OOB Testing](#)
- [TX/RX PCIe Beacon Testing](#)
- [PCIe Receiver Detection Testing](#)
- [Channel-bond Testing](#)
- [Clock-correction Testing](#)
- [TX/RX USERCLK Testing](#)
- [Reset Testing](#)

As shown in [Figure C-4](#), two FPGAs on different boards are connected through a BullsEye cable. This setup replicates a real scenario of how the SerDes usually talk to each other. Using this board-to-board setup, datapath testing can be done in five different ways as discussed in [Loopback Configuration Testing, page 130](#).

Vivado Design Suite debug feature is used to debug and trigger a special case to test using the same bit image. For example, to configure a specific loopback mode you can use the VIO core and pass a required value to configure a particular case. Similarly, the Vivado Design Suite debug feature helps to test different features by enabling and disabling them using both ILA and VIO cores.

Datapath Testing

The normal datapath testing can be classified in terms of internal and external data width. [Table C-1](#) shows the different possible combinations of data width.

Table C-1: TX/RX Internal and External Data Width Combination

TX Internal Data Width	TX External Data Width	RX Internal Data Width	RX External Data Width
16	16	16	16
16	32	16	32
20	20	20	20
20	40	20	40
32	32	32	32
32	64	32	64
40	40	40	40
40	80	40	80

Different PLL Testing

This tests the different possible VCO ranges of the CPLL and QPLL w.r.t. divider setting as shown in [Table C-2](#).

Table C-2: TX/RX Line Rate w.r.t. Different PLL Ranges

PLL Type	Output Divider	Line Rate Range (Gb/s)
CPLL line rate	1	3.2–6.6
	2	1.6–3.3
	4	0.8–1.85
	8	0.5–0.825
	16	NA
QPLL range1 line rate	1	5.93–8
	5	2.965–4
	4	1.4825–2
	8	0.74125–1
	16	NA
QPLL range2 line rate	1	9.8–12.5
	2	4.6–6.25
	4	2.45–3.125
	8	1.225–1.5625
	16	0.6125–0.78125

Loopback Mode Testing

This tests four combination of loopback on a board-to-board setup and those are;

- Near-End PCS loopback > LOOPBACK (001)
- Near-End PMA loopback > LOOPBACK (010)
- Far-End PMA loopback > LOOPBACK (100)
- Far-End PCS loopback > LOOPBACK (110)

PRBS Mode Testing

This tests the following mode combinations:

- PRBS-7 (On 8B/10B encoding, all possible internal data widths) > PRBSSEL (001)
- PRBS-15 (On 8B/10B encoding, all possible internal data widths) > PRBSSEL (010)
- PRBS-23 (none for 8B/10B encoding) > PRBSSEL (011)
- PRBS-31 (none for 8B/10B encoding) > PRBSSEL (100)
- PCIe compliance pattern (only 20-bit and 40-bit internal data widths) > PRBSSEL (101)
- Square wave of 2 UI > PRBSSEL (110)
- Square wave of n UI (Where "n" is internal data width) > PRBSSEL (111)

TX/RX Buffer Testing

In the TX path, a TX FIFO is available to control the data flow between the PCS and PMA clock domain. Similarly, the elastic buffer is available in the RX path. In the RX path, the datapath is tested with the buffer and in bypass mode.

REFCLK Testing

This tests all possible divider settings, namely, 1, 2, 4, 8, and 16. This also tests the REFCLK selection from +1/-1 neighbor Quad.

Powerdown Testing

This tests the following powerdown features:

- TX powerdown (TXPD) in PCIe and SATA mode
- RX powerdown (RXPDP) in PCIe and SATA mode
- CPLL powerdown (CPLLDP)
- QPLL powerdown (QPLLDP)

- TX phase and delay aligner powerdown (TXPHDLYPD)
- RX phase and delay aligner powerdown (RXPHDLYPD)

TX/RX Rate-change Testing

This tests all the possible static and dynamic rates on both the TX and RX side as shown in [Table C-3](#).

Table C-3: TX/RX Possible Rate w.r.t. Static and Dynamic Divider (D) Settings

TXRATE	RXRATE
000 (use TXOUT_DIV)	000 (use RXOUT_DIV)
001 (/1)	001 (/1)
010 (/2)	010 (/2)
011 (/4)	011 (/4)
100 (/8)	100 (/8)
101 (/16)	101 (/16)

TX/RX Polarity Testing

This tests the polarity inversion on the TX and RX serial line depending on the TXPOLARITY and RXPOLARITY control signals.

TX/RX Electrical Idle Testing

This tests the electrical idle condition of the TX and RX, respectively. In the transmitter when the TX serial lines are in common-mode voltage, the transmitter is a TX electrical mode, and the same occurs in the receiver. In PCIe mode, this testing requires a beacon transmitter and receiver.

Encoding and Decoding Testing

This tests all different encoding and decoding types, as listed in [Table C-4](#).

Table C-4: Different Combinations of Encoding and Decoding Scheme

Encoding Type	Decoding Type
8B/10B	10B/8B
None	None
None_MSB_First	None_MSB_First
64B/66B	66B/64B
64B/67B	67B/64B

Comma Detection Testing

This tests to decode valid commas as per the combinations shown in [Table C-5](#).

Table C-5: Comma Detection Type w.r.t. RX Data Width and Type of Decoding

RX External Data Width	Decoding Type	Comma Detection Type
20/40/80	10B8B	K28.5/K28.1
20/40/80	None/None-MSB-First	User defined

Comma Alignment Testing

This tests all possible comma alignment combinations for 10B8B and None and None_MS_B_first decoding w.r.t. RX data width, as shown in [Table C-6](#).

Table C-6: Comma Alignment Combination w.r.t. RX Data Width and Type of Decoding

Type of Decoding	RX Internal Data Width	RX Byte Alignment
	20	Any byte boundary/two byte boundary/none
10B8B	40	Any byte boundary/two byte boundary/four byte boundary/none
None	20	Any byte boundary/two byte boundary/none
	40	Any byte boundary/two byte boundary/four byte boundary/none
None_MS_B_first	20	Any byte boundary/two byte boundary/none
	40	Any byte boundary/two byte boundary/four byte boundary/none
66B64B	Any	None
67B64B	Any	None

TX/RX OOB Testing

There are three possible OOBs available in the SATA/SAS protocol, as shown in [Table C-7](#).

Table C-7: TX/RX OOB with All Possible Power and Rate States

TX OOB Type	TX POWERDOWN	TX Rate	RX OOB Type	RX POWERDOWN	RX Rate
COMINIT	PARTIAL SLUMBER NORMAL	GEN1 GEN2 GEN3	COMINIT	PARTIAL SLUMBER NORMAL	GEN1 GEN2 GEN3
COMWAKE	PARTIAL SLUMBER NORMAL	GEN1 GEN2 GEN3	COMWAKE	PARTIAL SLUMBER NORMAL	GEN1 GEN2 GEN3
COMSAS	PARTIAL SLUMBER NORMAL	GEN1 GEN2 GEN3	COMSAS	PARTIAL SLUMBER NORMAL	GEN1 GEN2 GEN3

TX/RX PCIe Beacon Testing

To transmit a beacon in PCIe mode, the requirement is that power-mode should be in P2 and `txelecidle` should be active-Low. Similarly, to detect a beacon in the RX path, the condition is that power-mode should be in P2, and when a beacon is detected the `rxelecidle` status will be asserted Low (Table C-8).

Table C-8: TX/RX PCIe Beacon Condition

PCIe Powermode	PCIe TXELECIDLE	PCIe Powermode	PCIe RXELECIDLE
P2	Active-Low	P2	Active-Low

PCIe Receiver Detection Testing

This tests the receiver detection in PCIe mode (Table C-9).

Table C-9: PCIe Receiver Detection Condition

PCIe Powerdown	PCIe TXDETECTRX	RXSTATUS	PHYSTATUS
P1	Active-High	3'h3	Active-High pulse

Channel-bond Testing

This tests all possible combinations of channel-bond in all decoding schemes (Table C-10).

Table C-10: Combinations of Channel-bond

Decoding Type	CB Sequence Number	Sequence Length	Sequence Maximum Skew	CB Sequence
10B8B decoding	1 or 2	1 or 2 or 4	1 to 14	User defined
None	1 or 2	1 or 2 or 4	1 to 14	User defined
None_MSB_first	1 or 2	1 or 2 or 4	1 to 14	User defined
66B64B	NA	NA	NA	NA
67B64B	NA	NA	NA	NA

Clock-correction Testing

This tests all possible combinations of clock-correction in all decoding schemes (Table C-11).

Table C-11: Combinations of Clock-correction

Decoding type	CB Sequence No	Sequence Length	CC Sequence	Periodicity
10B8B decoding	1 or 2	1 or 2 or 4	User defined	User defined
None	1 or 2	1 or 2 or 4	User defined	User defined
None_MSB_first	1 or 2	1 or 2 or 4	User defined	User defined

Table C-11: Combinations of Clock-correction (Cont'd)

Decoding type	CB Sequence No	Sequence Length	CC Sequence	Periodicity
66B64B	NA	NA	NA	NA
67B64B	NA	NA	NA	NA

TX/RX USERCLK Testing

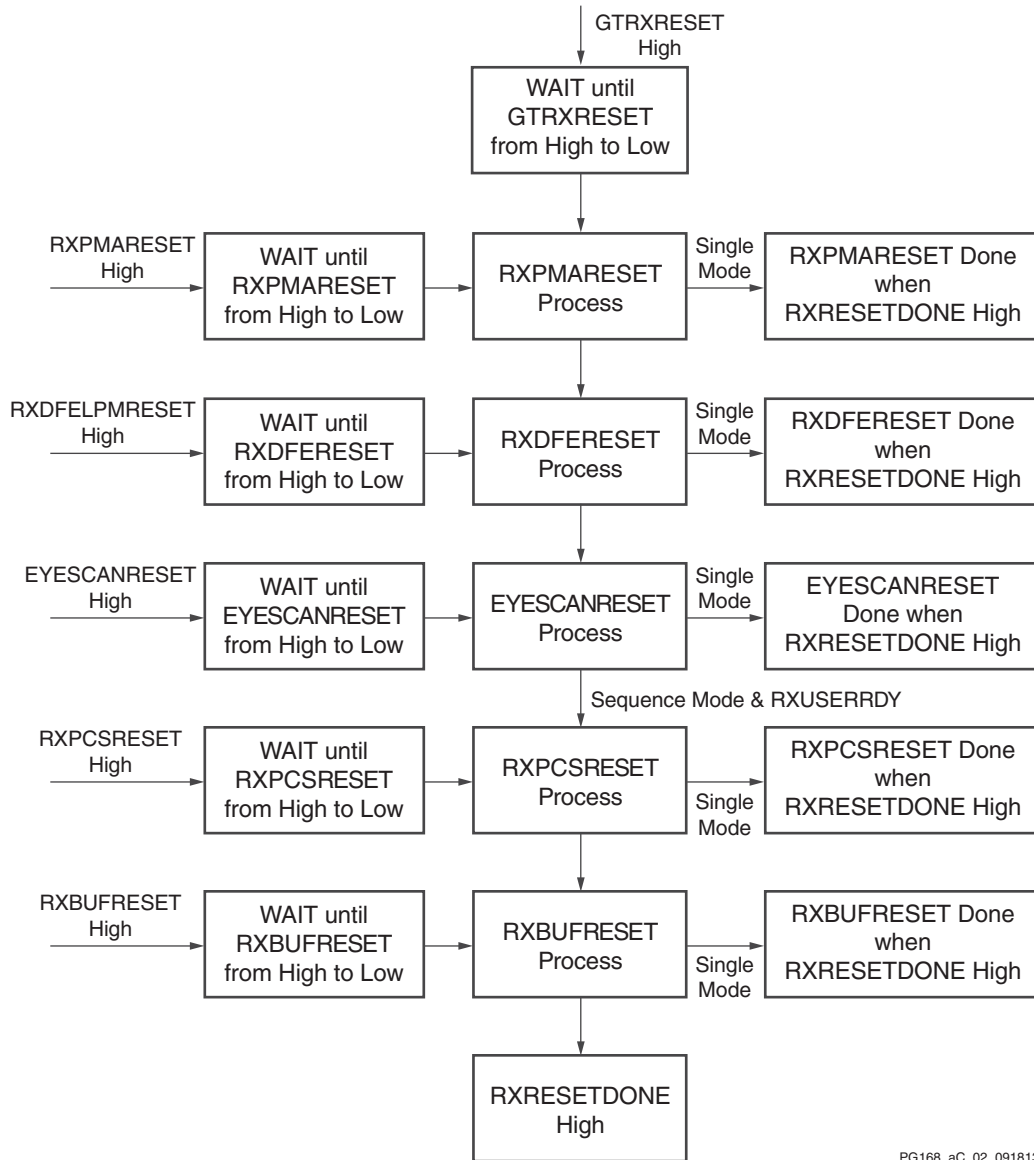
This tests all possible combinations of the TX and RX user clock source (Table C-12).

Table C-12: Combinations of User Clock Source

TXUSRCLK Source	RXUSRCLK Source
TXOUTCLK	TXOUTCLK
TXOUTCLK	RXOUTCLK
RXOUTCLK	TXOUTCLK
RXOUTCLK	RXOUTCLK

Reset Testing

In the TX path, the resets TXPMARESET and TXPCSRESET are tested. Similarly, in the RX path, the resets RXPMARESET, RXPCSRESET, RXDFELPMRESET, EYESCANRESET, RXBUFRESET, and RXOORESET are tested, as shown in Figure C-2.



PG168_aC_02_091813

Figure C-2: Reset Sequence in RX Path

Simulation Debug

The 7 series FPGAs Transceivers Wizard core example design has specific prerequisites that the simulation environment and the test bench must fulfill. These are described in the following sections. See the latest version of the *Synthesis and Simulation Design Guide* (UG626) [Ref 13] for more information on simulator dependency on hardware description language (HDL).

The prerequisites for the simulation environment are:

- A simulator with support for SecureIP models.
- A mixed-language simulator for VHDL simulation.
- An installed GTX/GTH/GTP transceiver SecureIP model.
- The ability to run COMPXLIB, which compiles the simulation libraries (for example, UNISIM, SIMPRIMS) in the correct order.
- The correct setup of the simulator for SecureIP use (initialization file, environment variables).
- The correct simulator resolution (Verilog).

If the channel does not come up in simulation:

- Ensure that `DRP_CLK` and `SYSCLK` are driven correctly.
- The quickest way to debug these problems is to view the signals from one of the serial transceiver instances that are not working.
- Make sure that the serial transceiver 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` from the serial transceiver wrapper are toggling. If not toggling, you might have to wait longer for the PMA to finish locking. You should typically wait about 6 to 9 microseconds for the channel to come up. You might need to wait longer for simplex designs and 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 the `GTTXRESET`, `GTRXRESET`, `CPLLRESET`, `QPLLRESET`, `PLLORESET`, and `PLL1RESET` signals in your design. If these are being held active, your 7 series transceiver module will not be able to initialize.
- Check if the `CPLLLOCK`, `QPLLLOCK`, `TXRESETDONE`, and `RXRESETDONE` signals are asserted. If they are not, look into the state machines in the Init module.
- Be sure you do not have the `POWER_DOWN`, `TXPD`, and `RXPD` signals asserted.
- Make sure the `TXN` and `TXP` signals from each transceiver are connected to the appropriate `RXN` and `RXP` signals from the corresponding transceiver on the other side of the channel.
- If you are using a multi-lane channel, make sure all the serial transceivers on each side of the channel are connected in the correct order.
- If you are simulating Verilog, you 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 as 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 the channel comes up in simulation but not in hardware:

- Make sure the REFCLK frequency is exactly the same as the input in the 7 series FPGA transceiver core-generated Wizard.
- Make sure that the DRP clock is constrained properly in the XDC. See the 7 series data sheets for more information.
- When the DRP interface is enabled, DRP_CLK is connected to SYSCLK in the example design. Thus, make sure that SYSCLK is also constrained properly in the XDC when it is enabled.
- If REFCLK is driven from a synthesizer, make sure the synthesizer (PLL/MMCM_NOT_LOCKED) is stable (locked).
- Make sure the cable connection from TXP/TXN to RXP/RXN is proper.
- Make sure that the transceiver locations are properly set in XDC as chosen from the Wizard.
- If there are RXNOTINTABLE errors observed from the serial transceiver, validate the link using IBERT. Make sure there is no BER in the channel. Use the sweep test in the IBERT tool and use the same serial transceiver attributes that provide zero BER in IBERT.
- Make sure all the signals at the transceiver interface are toggling correctly. All necessary signals needed to debug should be captured in the Vivado Design Suite debug feature if you are debugging in hardware.

Problems while compiling the design:

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

Next Step

If the debug suggestions listed previously do not resolve the issue, 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

To discuss possible solutions, use the Xilinx User Community: forums.xilinx.com/xlnx/

Hardware Debug

The 7 series FPGAs Transceivers Wizard core has an option to use the Vivado Design Suite debug feature in the example design. Debugging and ensuring proper operation of the transceiver is extremely important in any protocol that uses the 7 series FPGAs Transceivers Wizard core. The 7 series FPGAs Transceivers Wizard core example design has a VIO core instantiated and connected with important status and control signals for validating the design in board.

To assist with debugging, these VIO cores are provided with the 7 series FPGAs Transceivers Wizard wrapper, which is enabled by setting EXAMPLE_USE_CHIPSCOPE to 1 in the `<component_name>exdes.v[hd]` file. [Figure C-3](#) shows the steps involved in debugging transceiver related issues.

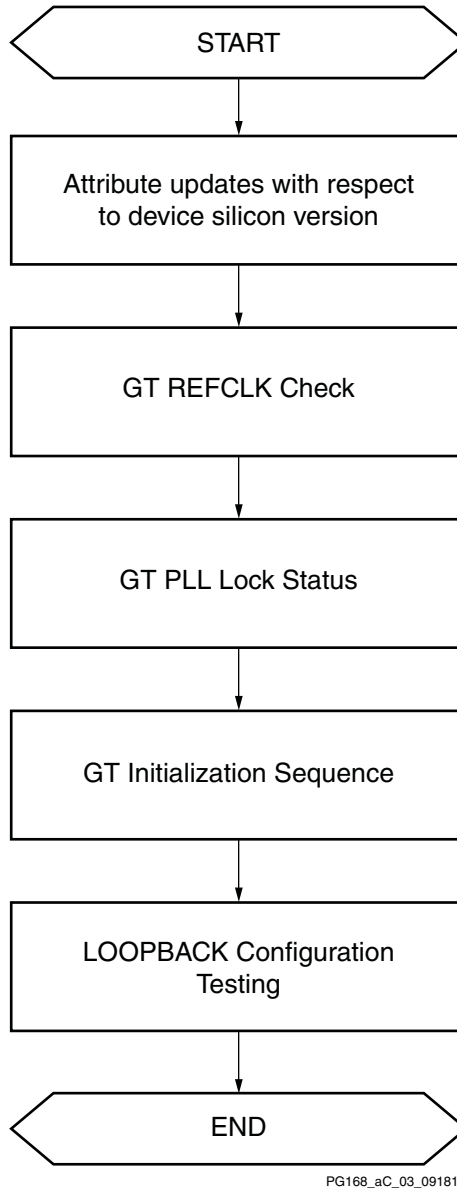


Figure C-3: Transceiver Debug Flow Chart

- 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.
- The EXAMPLE_SIMULATION parameter must be set to 0 in the top-level 7 series FPGAs Transceivers Wizard wrapper to enable hardware debugging.
- 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 the transceiver.

- GT PLL Lock Check

The transceiver locks into the incoming GT REFCLK and asserts the PLLLOCK signal. This signal is available as the PLLLOCK_OUT signal in the transceiver core example design. Make sure that the GT PLL attributes are set correctly and that the transceiver generates TXOUTCLK and RXOUTCLK with the expected frequency for the given line rate and datapath width options.

- GT Reset

In the 7 series FPGAs Transceivers Wizard core, RX resets can operate in two different modes: sequential mode and single mode. The TX reset can operate only in sequential mode. Reset modes have no impact on CPLL/QPLL resets. The TXRESETDONE and RXRESETDONE signals are asserted at the end of the transceiver initialization. In general, RXRESETDONE assertion takes a longer time compared to TXRESETDONE assertion. Make sure the GT_RESET signal pulse width duration matches with the respective transceiver guidelines. The TXRESETDONE and RXRESETDONE signals are available in the 7 series FPGAs Transceivers Wizard core example design to monitor.

- GT Initialization Sequence

The 7 series FPGA transceiver must be initialized after device power-up and configuration before it can be used. See [AR 43482](#) for details on the initialization requirements. The TX and RX datapaths must be initialized only after the associated PLL is locked. The transceiver TX and RX initialization comprises two steps:

- Initializing the associated PLL driving TX/RX
- Initializing the TX and RX datapaths (PMA + PCS)

- Loopback Configuration Testing

Loopback modes are specialized configurations of the transceiver datapath where the traffic stream is folded back to the source. The LOOPBACK port in the 7 series FPGAs Transceivers Wizard core example design will transmit a specific traffic pattern and then compare to check for errors and control the loopback modes. Loopback test modes fall into two broad categories:

- Near-end loopback modes loop transmit data back in the transceiver closest to the traffic generator. Near-end loopback modes are:
 - Near-end PCS loopback
 - Near-end PMA loopback
- Far-end loopback modes loop received data back in the transceiver at the far end of the link. Far-end loopback modes are:

- Far-end PMA loopback
- Far-end PCS loopback

Loopback testing can be used either during development or in deployed equipment for fault isolation. The traffic patterns used can be either application traffic patterns or specialized pseudo-random bit sequences. The loopback operations are controlled by the LOOPBACK [2 : 0] ports:

- 000: Normal operation
- 001: Near-end PCS loopback
- 010: Near-end PMA loopback
- 011: Reserved
- 100: Far-end PMA loopback
- 101: Reserved
- 110: Far-end PCS loopback
- 111: Reserved

Four loopback modes are available. See the respective transceiver user guides for guidelines and more information. [Figure C-4](#) illustrates a loopback test configuration with four different loopback modes.

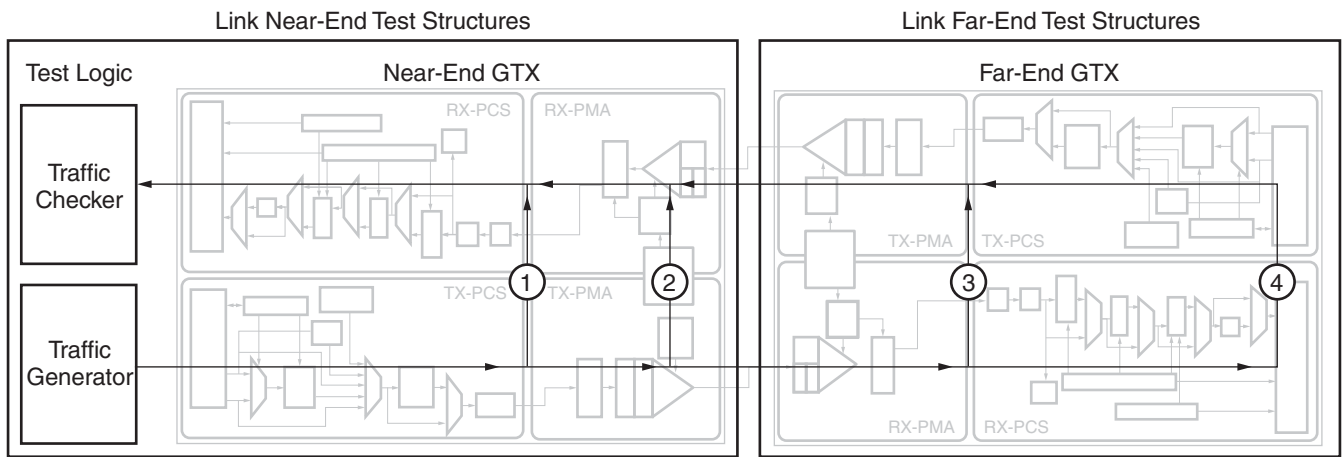


Figure C-4: Loopback Test Modes

Loopback Limitations

- When using near-end PMA loopback, do not drive the RX pins of the transceivers.
 - When using far-end PMA loopback for one transceiver, the other transceivers in the same Quad are not affected.
-

GT Debug Using IBERT

Integrated Bit Error Ratio Tester (IBERT) core for the 7 series FPGA transceivers is designed for evaluating and monitoring the transceivers, and is supported by the Vivado design tools available in the IP catalog under Debug & Verification. Because communication logic is included in IBERT, it allows the design to be run-time accessible through JTAG, through which transceiver debug can be done on hardware.

IBERT design has a pattern generator and a pattern checker that sends a generated pattern through the transmitter and accepts data through the receiver and checks it against internally generated patterns. The following condition qualifies the link:

- Link Up: When the checker receives five consecutive cycles of data with no errors, the `LINK` signal is asserted in IBERT.
- Link Down: If the `LINK` signal is asserted and the checker receives five consecutive cycles with data errors, the `LINK` signal is deasserted.

If you see the Link Down issue in IBERT, use the Vivado Design Suite debug feature IBERT design to validate the serial transceiver link. The debug feature IBERT design also allows you to optimize serial transceiver link parameters during run-time.

Debugging Using Serial I/O Analyzer

You can change 7 series FPGA transceiver ports and attributes. The DRP interface logic allows the run-time software to monitor and change any attribute of the transceivers and the corresponding CPLL/QPLL. Readable and writable registers are also included that are connected to the various ports of the transceiver. All are accessible at run-time using the Vivado serial I/O analyzer to debug DRP- and port-related issues.

Debugging Using Embedded BERT

In 7 series FPGA transceivers, there is an in-built PRBS function generator (pattern generator) on the transmit PCS, and PRBS function checker (pattern checker) on the receiver PCS that can be used for the Bit Error Rate Test (BERT). PRBS 7-bit, PRBS 15-bit, PRBS 23-bit, and PRBS 31-bit are available in the 7 series transceivers. These can be controlled using the `TXPRBSSEL[2:0]` port as:

- 000: No PRBS pattern generation
- 001: Enables 27-1 PRBS checker
- 010: Enables 215-1 PRBS checker
- 011: Enables 223-1 PRBS checker
- 100: Enables 231-1 PRBS checker
- 101: PCIe technology compliance pattern generation
- 110: Generation of square wave with 2 UI period
- 111: Generation of square wave with 16/20/32/40 UI period (depending on data width)

Errors in the bitstream can be forced based on the value driven on `TXPRBSFORCEERR`.

- 0: No errors are forced in the PRBS transmitter
- 1: Errors are forced in the PRBS transmitter. The output data pattern contains errors. When `TXPRBSSEL` is set to 000, this port does not affect `TXDATA`.

The `RXPRBSCNTRESET` port can be used to reset the error counter in the transceiver and `RXPRBSEERR` can be monitored to identify when an error occurred. The `RX_PRBS_ERR_CNT` attribute can be read out through the DRP to validate the PRBS error counter value.

7 Series GT Wizard Hardware Validation on the KC705 Evaluation Board

The Kintex-7 FPGA KC705 evaluation kit provides a comprehensive, high-performance development and demonstration platform using the Kintex-7 family for high-bandwidth and high-performance applications in multiple market segments. For more information about this evaluation board, see the *Kintex-7 FPGA KC705 Evaluation Kit* (UG883) [Ref 16].

Setup Requirements

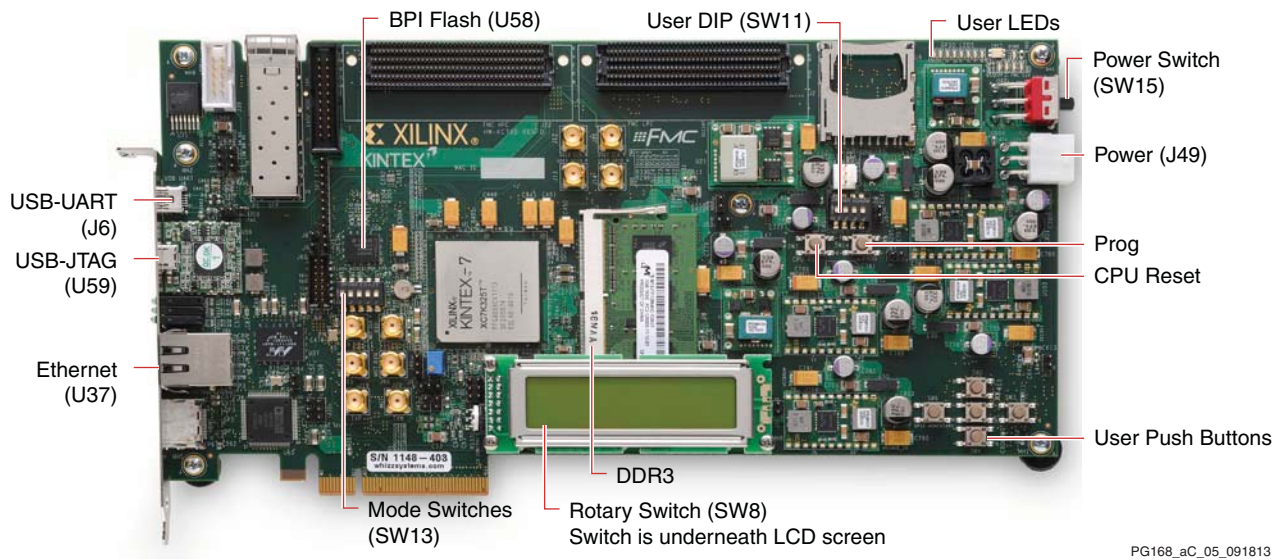
Before you start this tutorial, make sure you understand the hardware and software components needed to perform the steps. The following subsections list the requirements.

Software

Vivado Design Suite 2012.3

Hardware

Kintex-7 FPGA KC705 evaluation kit base board (Figure C-5).



PG168_aC_05_091813

Figure C-5: KC705 Evaluation Kit

GTX Transceiver Validation

As discussed in Chapter 4, Design Flow Steps, customize the core per the requirements, generate the core, and follow these steps:

1. In the Project Manager window of the Vivado IP catalog, right-click the core and select **Open IP Example Design**. A new Vivado IDE opens with the core example design files.
2. Open <component_name>_exdes.v[hd] and change EXAMPLE_USE_CHIPSCOPE from 0 to 1 to enable debugging using the Vivado Design Suite debug feature cores.
3. Open <component_name>_0.v[hd] and change EXAMPLE_SIMULATION value from 1 to 0 as shown below:

```
.EXAMPLE_SIMULATION (0),
```

4. Open <component_name>_exdes.xdc and make sure that the GTREFCLK_PAN_N_IN and GTREFCLK_PAD_P_IN location is proper.

Example:

```
set_property LOC J7 [get_ports Q2_CLK0_GTREFCLK_PAD_N_IN ]
set_property LOC J8 [get_ports Q2_CLK0_GTREFCLK_PAD_P_IN ]
```

5. Make sure that the GTXE2_CHANNEL location is set properly in `<component_name>_exdes.xdc`.

Example:

```
set_property LOC GTXE2_CHANNEL_X0Y8 [get_cells
gtwiz_eou_test2_support_i/gtwiz_eou_test2_init_i/inst/gtwiz_eou_test2_i/
gt0_gtwiz_eou_test2_i/gtxe2_i]
```

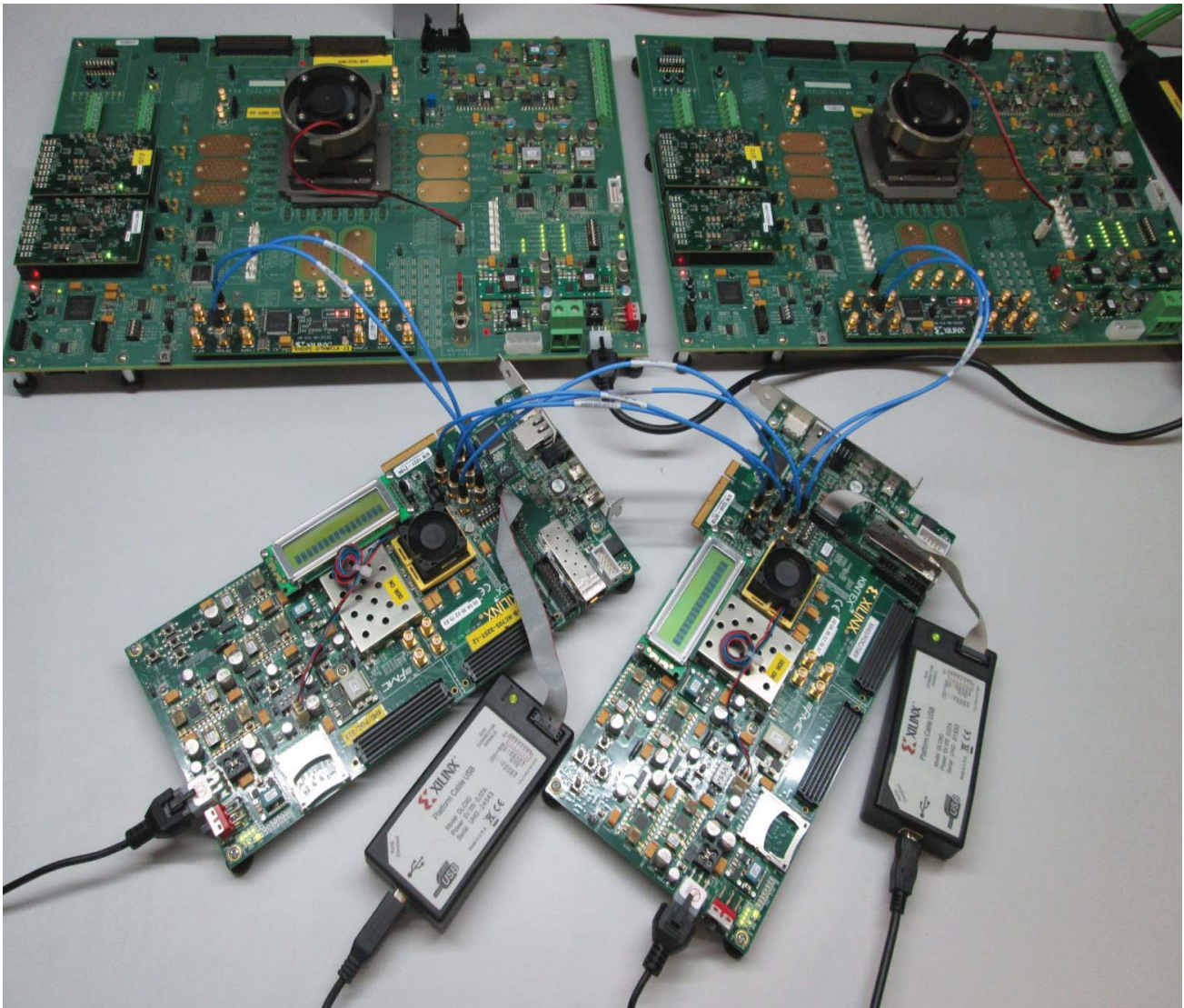
6. Click **Run Synthesis** and after synthesis is completed, click **Open Synthesized Design**. The tool runs for some time to synthesize and open the synthesized netlist.
7. When the synthesized design is opened, enter **write_debug_probes** `<ltx_file_name>` in the Tcl console and press **Enter**. This command creates `<ltx_file_name>.ltx` under the `<component_name>_example` folder.
8. Click **Generate Bitstream** which generates `routed.bit` in the `<component_name>_example` folder. If you see any errors while generating the BIT file, click **Open Implemented Design**. If implementation is already done, the operation opens the implemented design. If not, the tool runs for some time to implement and open the implemented design.
9. When the implemented design is opened, enter the following comments in the Tcl console to generate the `routed.bit` bitstream:

```
set_property SEVERITY {Warning} [get_drc_checks NSTD-1]; set_property SEVERITY
{Warning} [get_drc_checks UCIO-1]
```

```
write_bitstream -bitgen_options {-g UnconstrainedPins:Allow} -file routed.bit -force
```

10. Ensure that the board setup is arranged according to these steps:
 - 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.

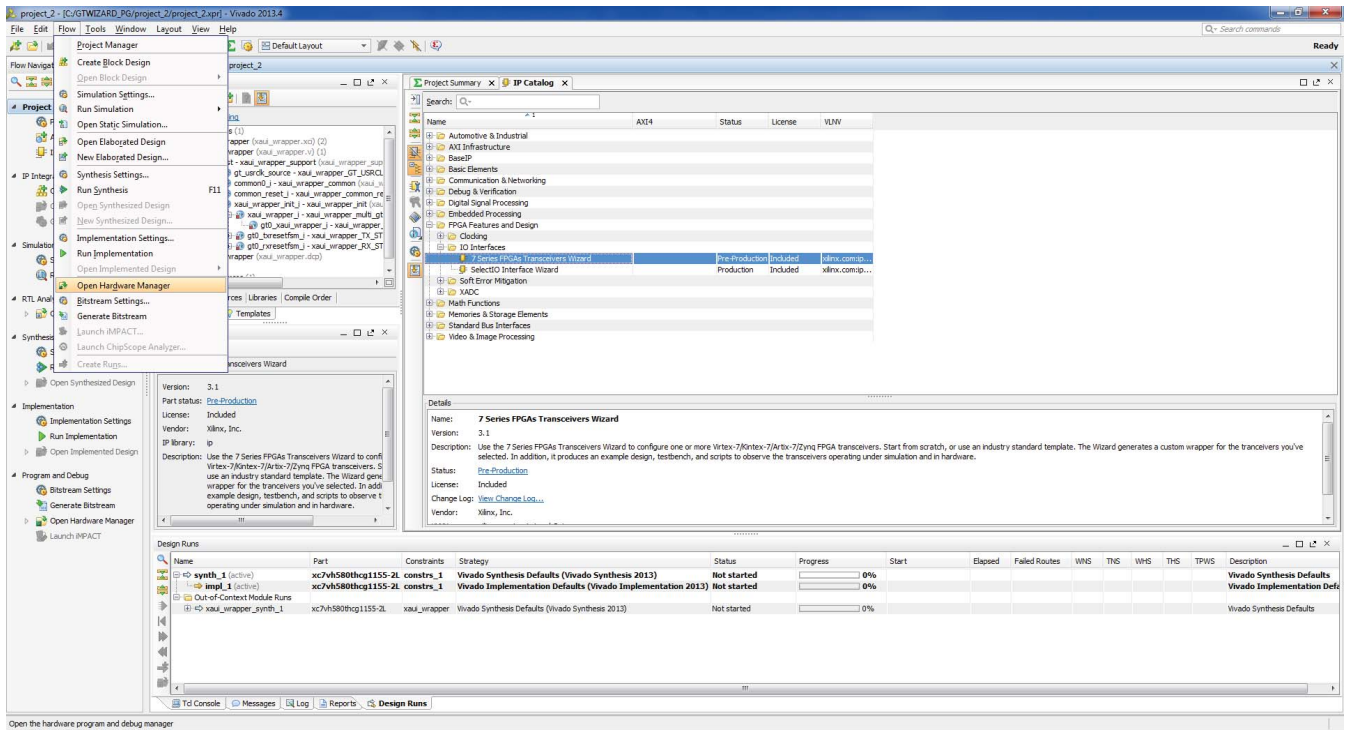
- c. The reference clock to each KC705 board should be fed from a different source, as shown in [Figure C-6](#).



PG168_aC_06_090613

Figure C-6: Hardware Setup

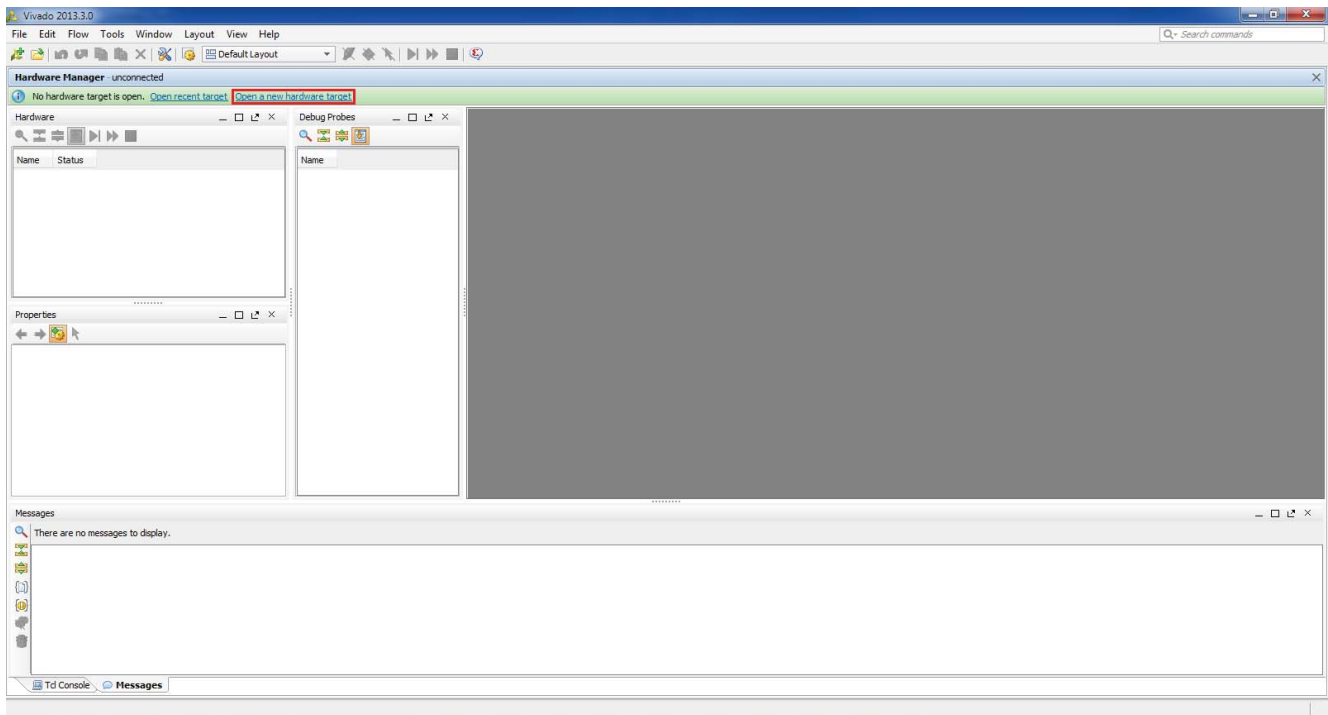
11. When the bit file is generated, click **File > Open Hardware Session**, as shown in Figure C-7.



PG168_aC_07_111913

Figure C-7: Open Hardware Session

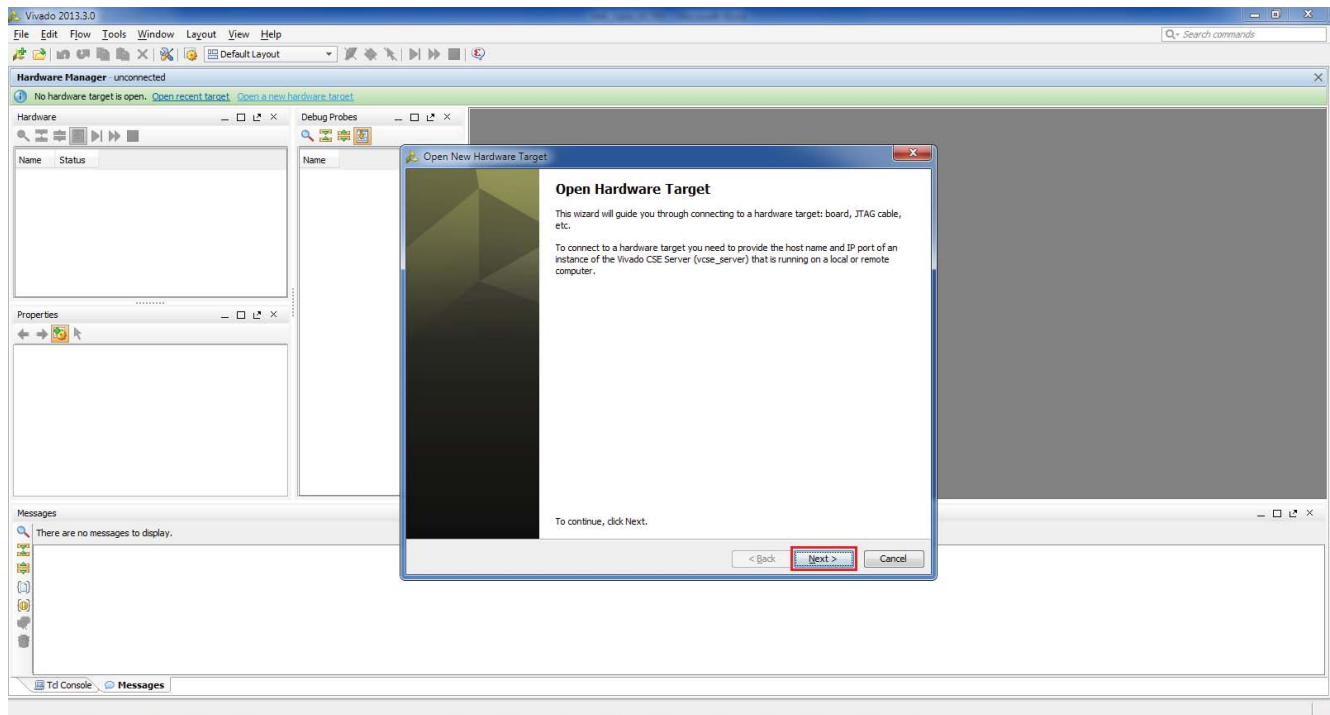
12. A new window appears, as shown in [Figure C-8](#). Click **Open a new hardware target**.



PG168_aC_08_090613

Figure C-8: Open a New Hardware Target

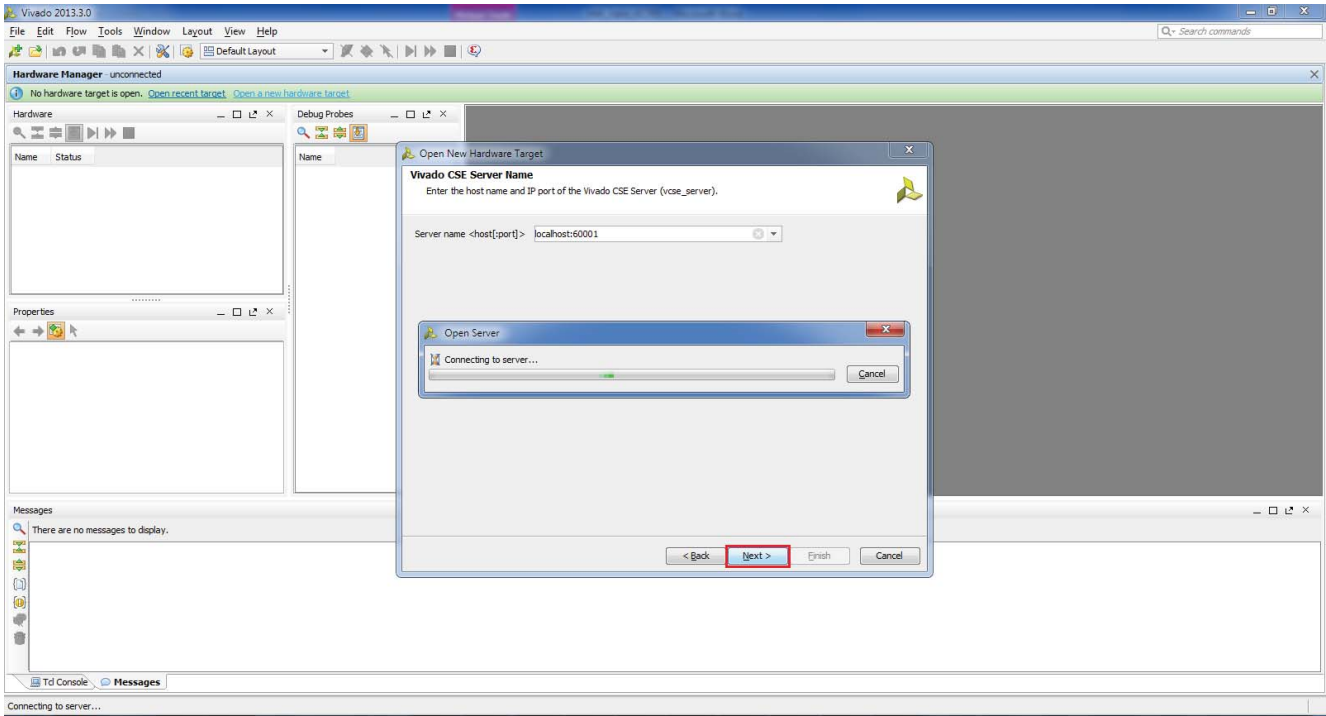
13. The **Open New Hardware Target** window opens, as shown in [Figure C-9](#). Click **Next**.



PG168_aC_09_090613

Figure C-9: **Open Hardware Target**

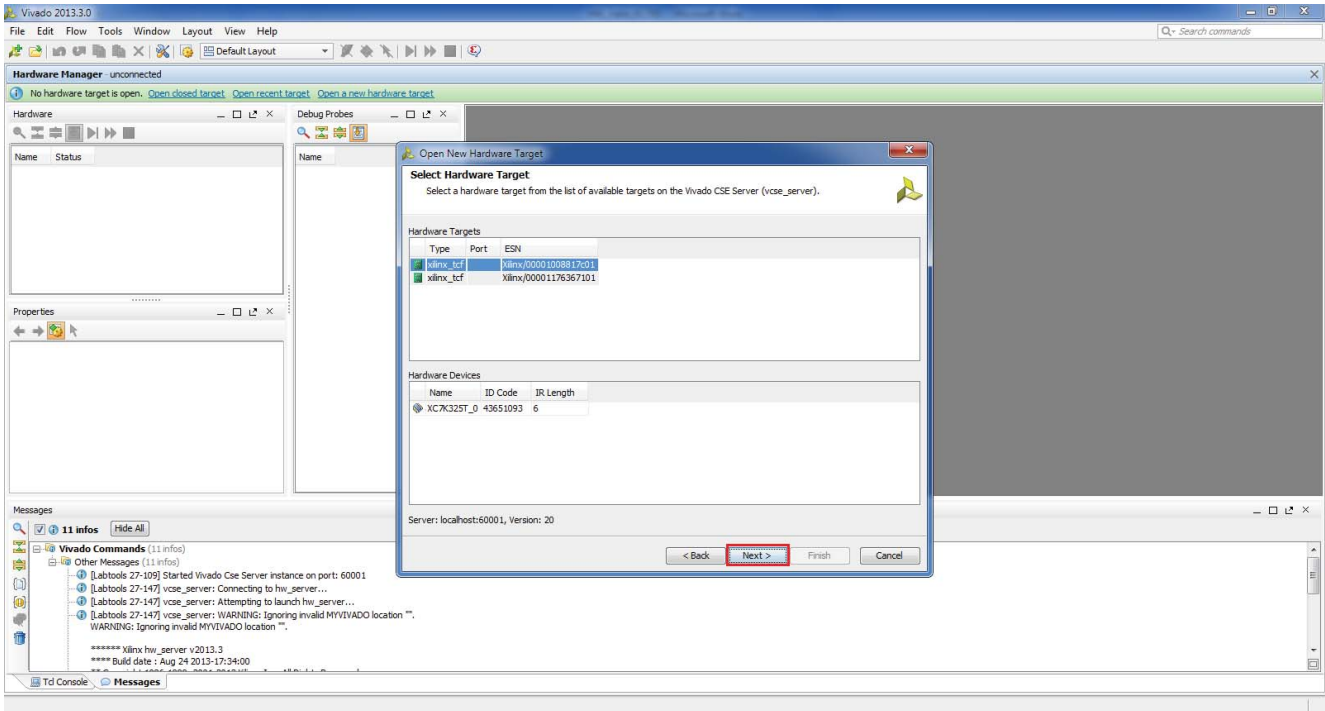
- Enter the server name to which the KC705 board is connected in **Vivado CSE Server Name**, as shown in [Figure C-10](#), and click **Next**. The Vivado IDE automatically connects to the server and detects all boards connected to the server.



PG168_aC_10_090613

Figure C-10: Vivado CSE Server Name

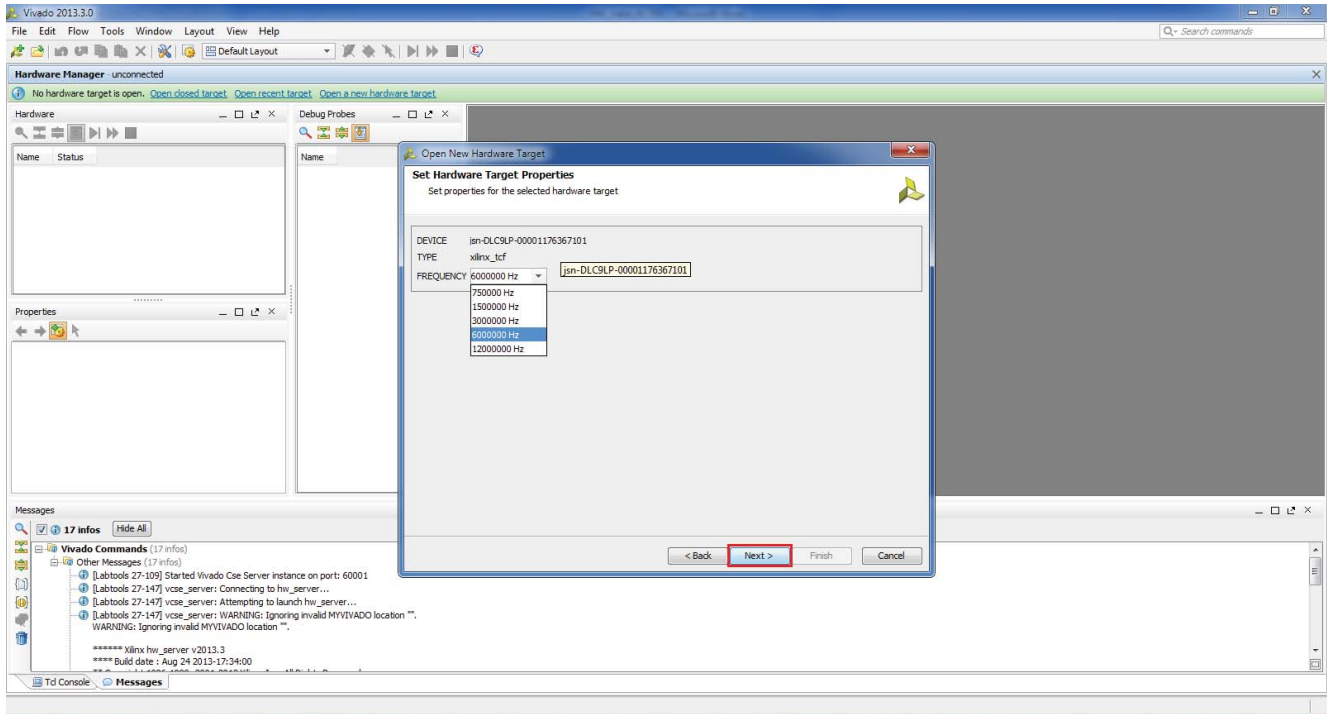
- The **Select Hardware Target** window opens with all targeted boards connected to the server, as shown in Figure C-11. Select the board that you want to program and click **Next** to set the targeted hardware JTAG properties.



PG168_aC_11_090613

Figure C-11: Select Hardware Target

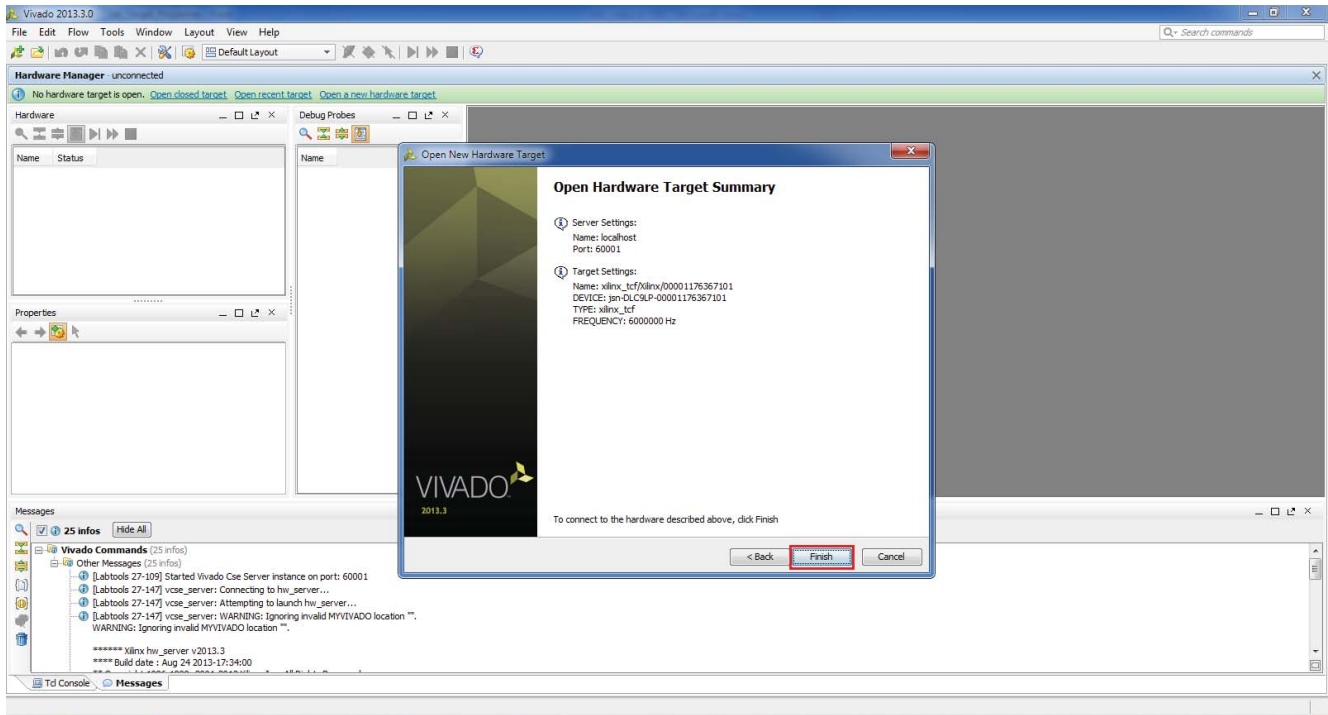
16. **Set Hardware Target Properties** allows you to choose any JTAG clock speed from the drop-down menu to program the targeted hardware. Choose the desired clock frequency and click **Next**.



PG168_aC_12_090613

Figure C-12: Set Hardware Target Properties

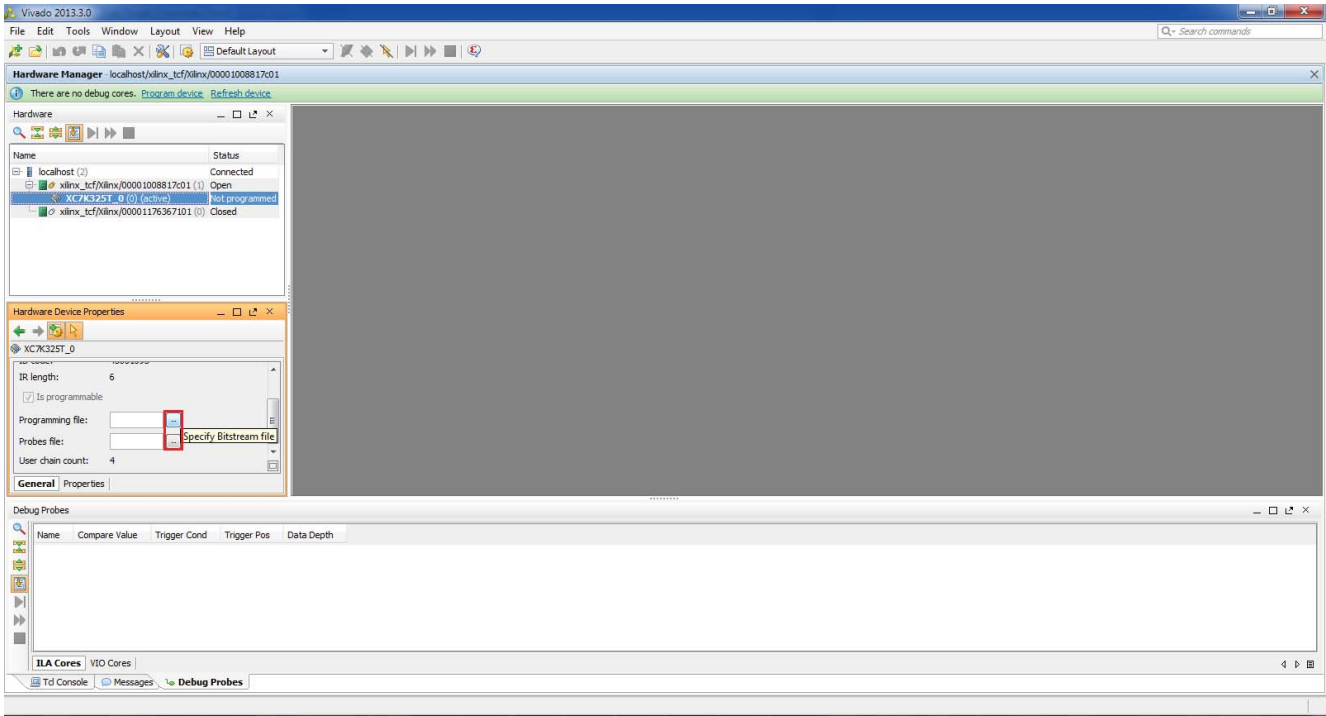
17. Review the targeted hardware summary and click **Finish**, as shown in [Figure C-13](#).



PG168_aC_13_090613

Figure C-13: Open Hardware Targeted Summary

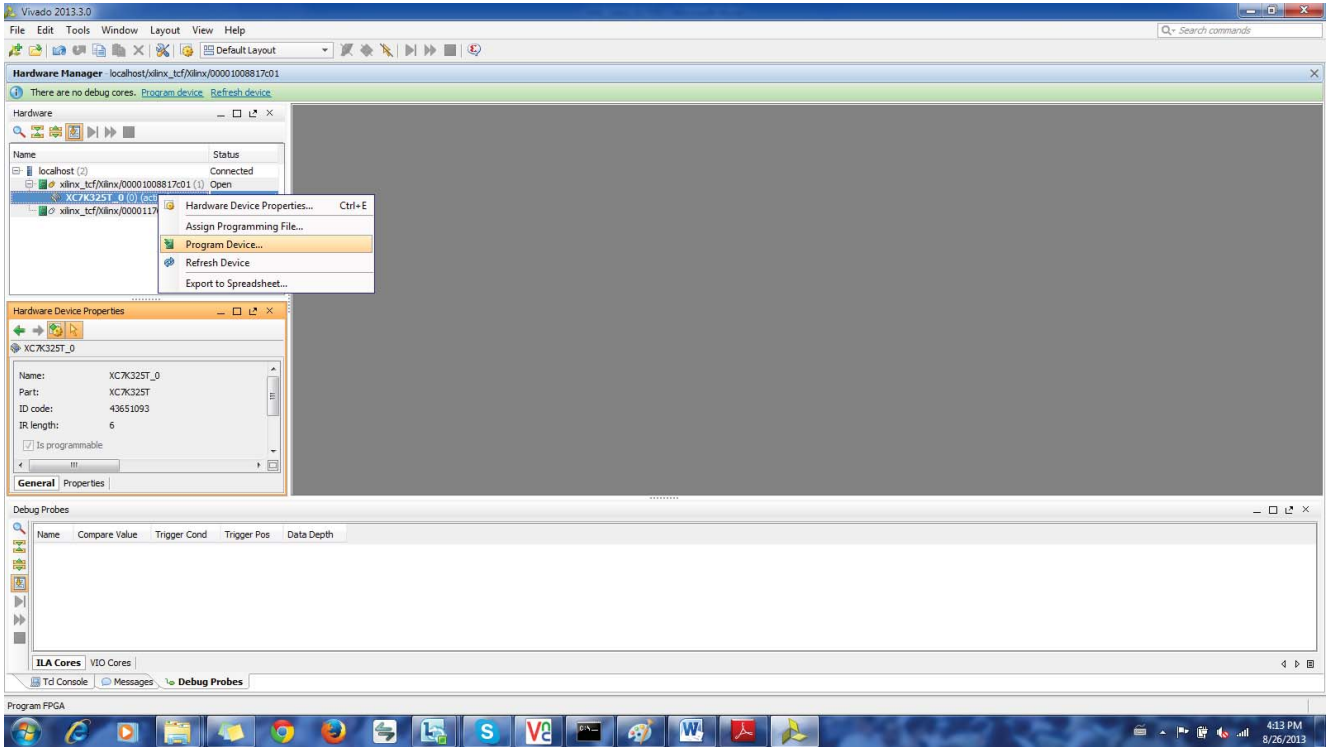
- Browse to and specify the bitstream file (<rouded>.bit) location in the programming file, and probe file (<probfile.ltx>) location in the Probes file as shown in Figure C-14.



PG168_aC_14_090613

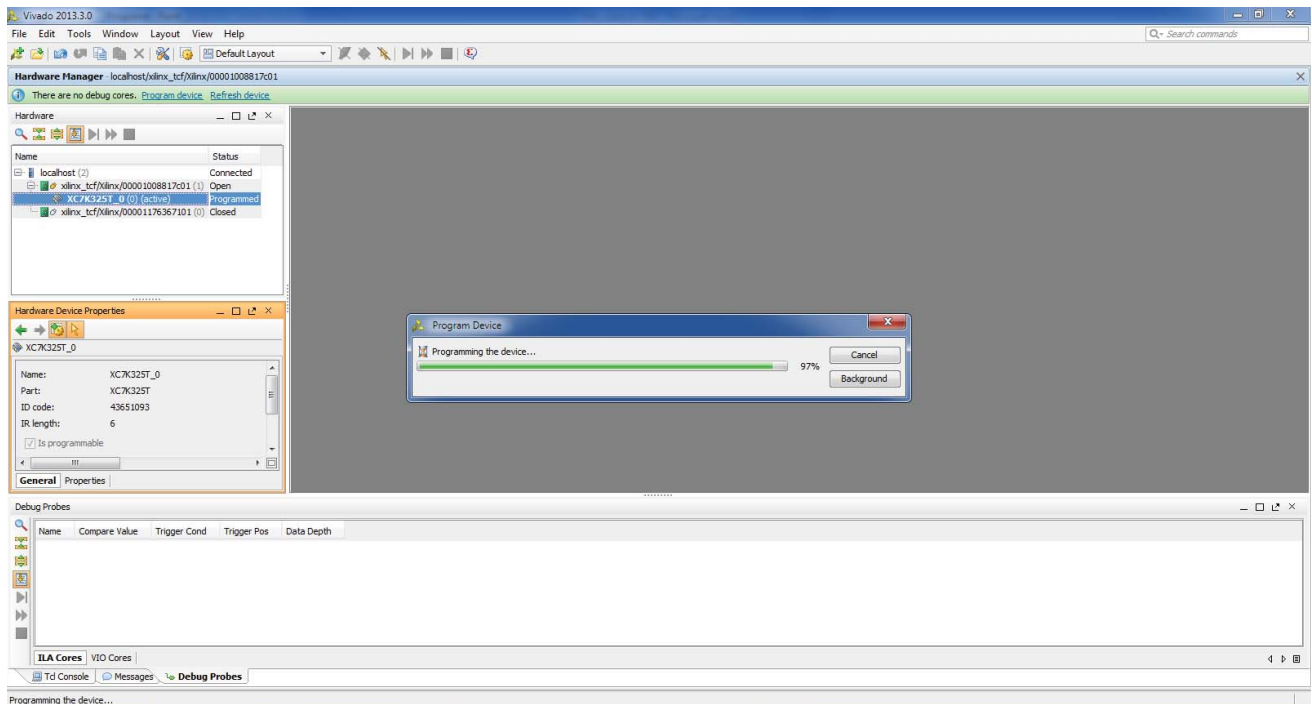
Figure C-14: Hardware Device Properties

- Right-click the device and select **Program Device** as shown in Figure C-15 and Figure C-16. Make sure the bit file location is correct, and click **OK**.



PG168_aC_15_090613

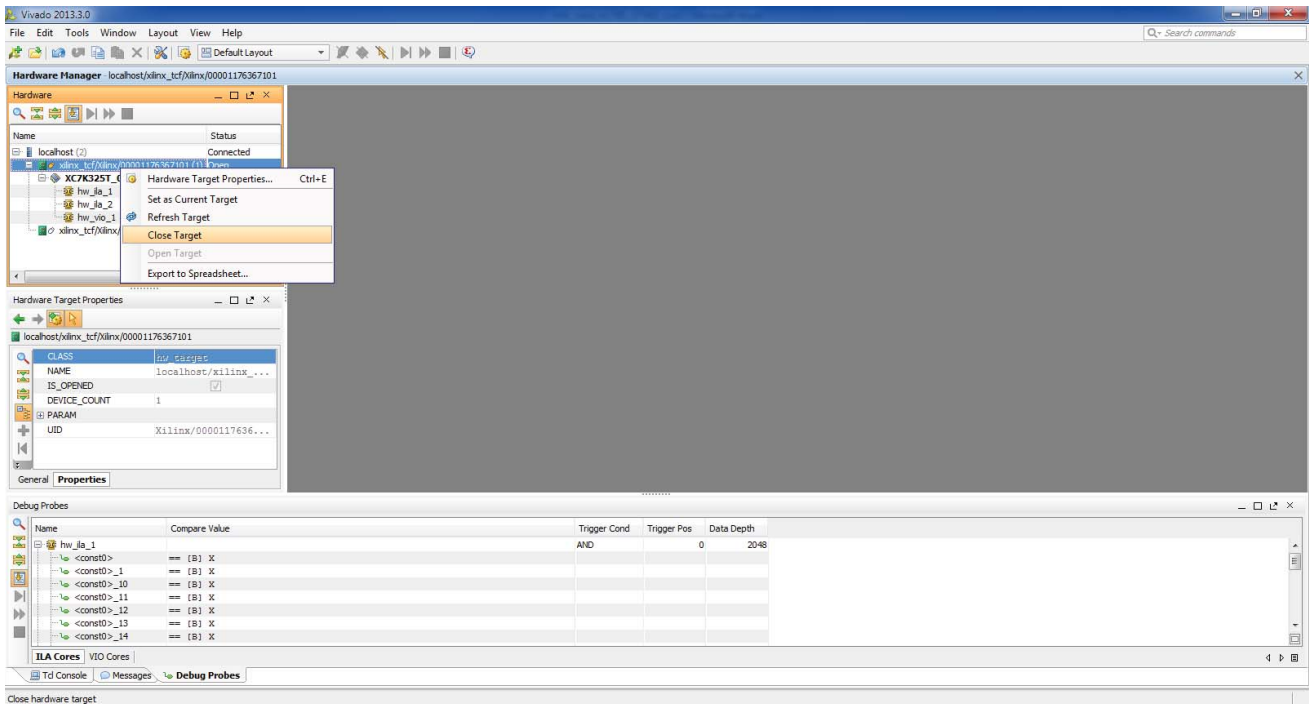
Figure C-15: Program Device (1)



PG168_aC_16_090613

Figure C-16: Program Device (2)

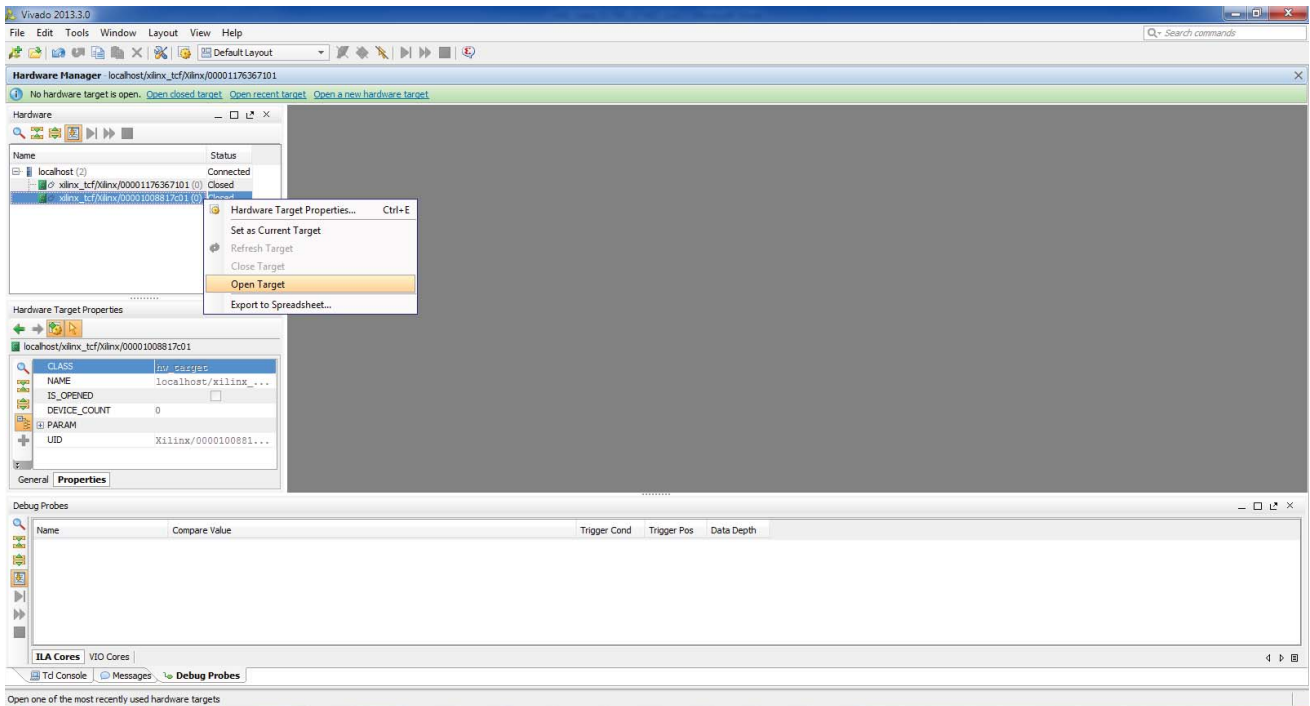
- When programming is finished on board 1, right-click the programmed device and select **Close Target**, as shown in Figure C-17.



PG168_aC_17_090613

Figure C-17: Close Target

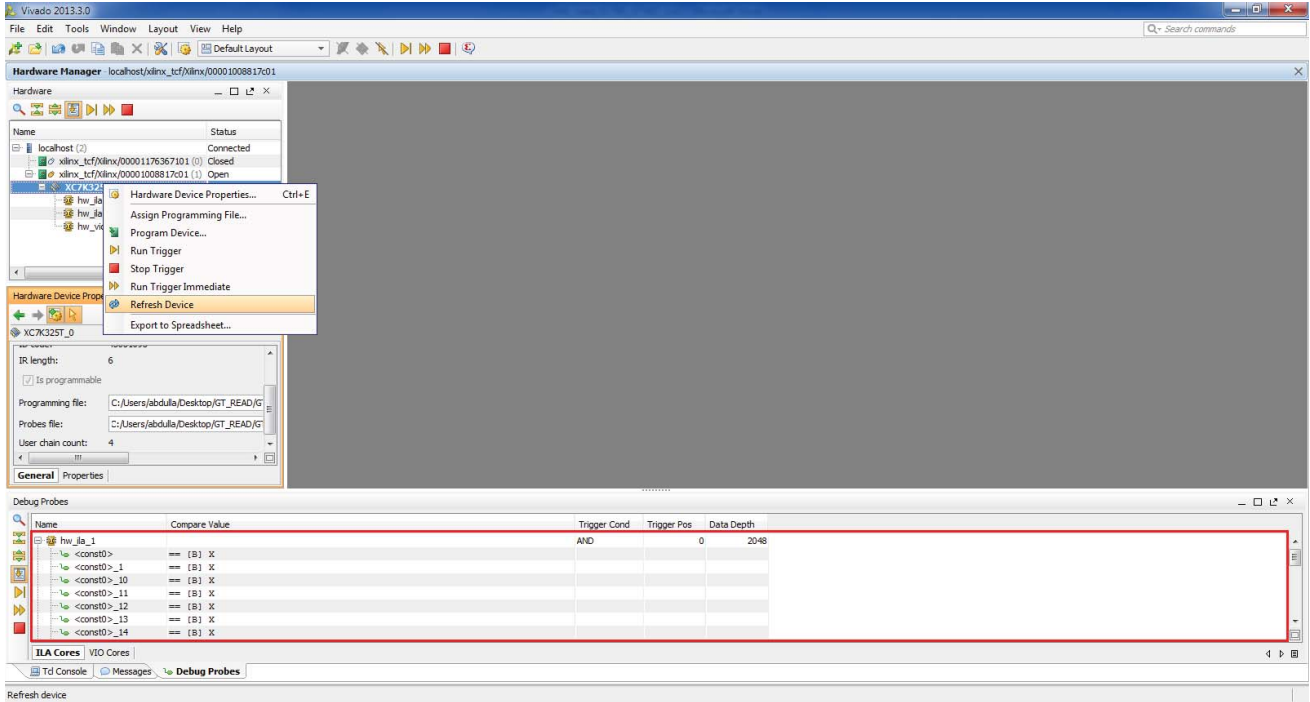
- To program board 2, select the second device, right-click the device, and select **Open Target**, as shown in [Figure C-18](#).



PG168_aC_18_090613

Figure C-18: Open Target

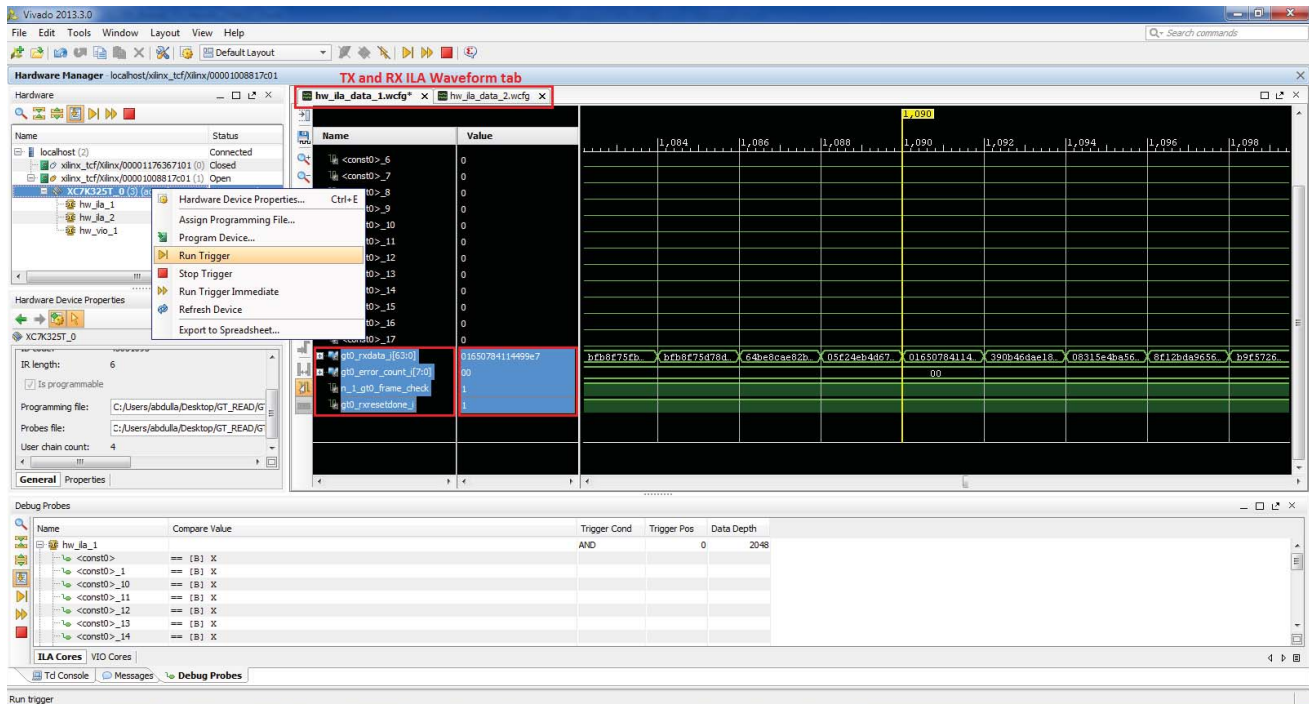
22. Repeat [step 18](#) and [step 19](#) to program board 2 with the same programming file and probe file that was downloaded on the first KC705 board. After programming is complete, right-click the device and select **Refresh Device**, as shown in [Figure C-19](#). You should be able to see all nets that are added to the ILA cores.



PG168_aC_19_090613

Figure C-19: Refresh Device

23. Right-click the device again and select **Run Trigger**. You will be able to see all the ILA cores debug signals in the waveform, as shown in [Figure C-20](#).



PG168_aC_20_090613

Figure C-20: Run Trigger

Notes:

- Make sure that the signals have the values shown in the RX ILA waveform window ([Figure C-20](#)):
 - `gt0_rxdata_i[63:0]` = some random 64-bit value
 - `gt0_error_count` = 00
 - `gt0_frame_check` = 1
 - `gt0_rxreset_done` = 1
 - `track_data_out_i` = 1
- Make sure that the signal has the following value as shown in the TX ILA waveform window ([Figure C-21](#)):
 - `gt0_txresetdone_i` = 1

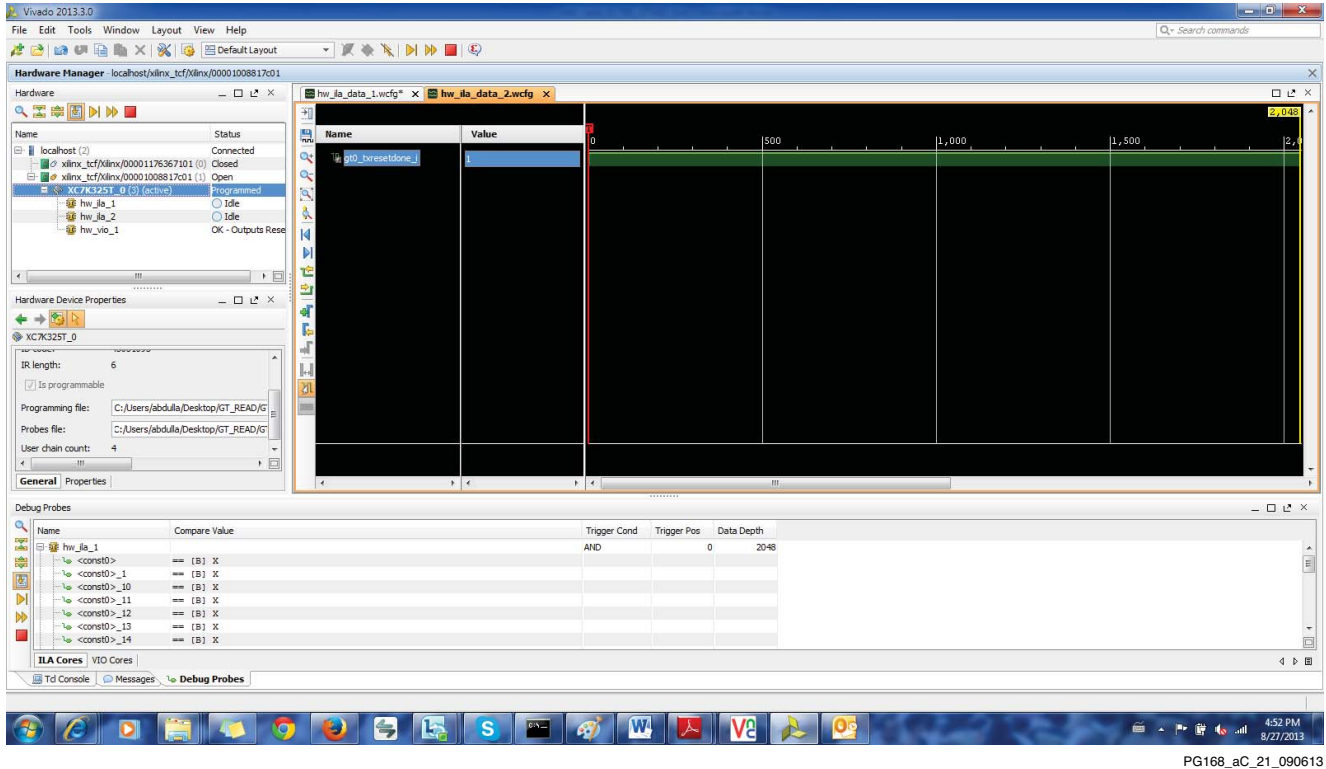


Figure C-21: TX ILA Waveform

24. Similarly, you can monitor the GTX transceiver transaction for the first KC705 board by repeating [step 21](#) and [step 22](#) to validate board 1.

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see [Xilinx Support](#).

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open the Xilinx Documentation Navigator (DocNav):

- From the Vivado® IDE, select **Help > Documentation and Tutorials**.
- On Windows, select **Start > All Programs > Xilinx Design Tools > DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In the Xilinx Documentation Navigator, click the **Design Hubs View** tab.
- On the Xilinx website, see the [Design Hubs](#) page.

Note: For more information on Documentation Navigator, see the [Documentation Navigator](#) page on the Xilinx website.

References

These documents provide supplemental material useful with this product guide:

1. *Virtex-7 FPGAs Data Sheet: DC and Switching Characteristics* ([DS183](#))
2. *Kintex-7 FPGAs Data Sheet: DC and Switching Characteristics* ([DS182](#))

3. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* ([UG994](#))
4. *Vivado Design Suite User Guide: Logic Simulation* ([UG900](#))
5. *Artix-7 FPGAs Data Sheet: DC and Switching Characteristics* ([DS181](#))
6. *Zynq-7000 SoC Data Sheet: DC and Switching Characteristics* ([DS191](#))
7. *7 Series FPGAs GTX/GTH Transceivers User Guide* ([UG476](#))
8. *7 Series FPGAs GTP Transceivers User Guide* ([UG482](#))
9. *Vivado Design Suite User Guide: Designing with IP* ([UG896](#))
10. *Vivado Design Suite User Guide: Getting Started* ([UG910](#))
11. *7 Series FPGAs Overview* ([DS180](#))
12. *7 Series FPGAs GTZ Transceivers User Guide* ([UG478](#))
13. *Synthesis and Simulation Design Guide* ([UG626](#))
14. *ISE to Vivado Design Suite Migration Guide* ([UG911](#))
15. *Vivado Design Suite User Guide: Programming and Debugging* ([UG908](#))
16. *Kintex-7 FPGA KC705 Evaluation Kit* ([UG883](#))

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
05/19/2022	3.6	General updates.
11/30/2016	3.6	Chapter 2: Added port direction and clock domain information.
09/30/2015	3.6	<ul style="list-style-type: none"> • Removed the GREFCLK reference clock source option, and added further information related to REFCLK0 and REFCLK1. • Example Design chapter: <ul style="list-style-type: none"> ◦ Changed example design to include Reset with CTLE tuning. ◦ Added new CTLE Tuning Module Ports and details. ◦ Added new CTLE Tuning Enable State Machine diagrams. ◦ Added steps for using the example design in hardware. ◦ Updated the Known Limitations of the GTZ Wizard section.
04/01/2015	3.5	<ul style="list-style-type: none"> • Updated for Wizard v3.5 and Vivado Design Suite 2015.1. • Added additional ports to the Optional Ports table. • Added more details about resetting the Finite State Machine (FSM) in the Example Design chapter.

Date	Version	Revision
10/01/2014	3.4	<p>Wizard 3.4 release. Updated to Vivado Design Suite 2014.3 throughout. Removed support for PCI Express throughout.</p> <p>IP Facts: Updated Example Design and Test Bench rows in LogiCORE IP Facts Table.</p> <p>Chapter 1: Added SATA: 6.0 to list of supported standards for GTX and GTP transceivers in Feature Summary. Updated Unsupported Features.</p> <p>Chapter 4: Updated Figure 4-1, and Figure 4-5 to Figure 4-15.</p> <p>Appendix B: Updated Figure B-4 and Figure B-5.</p>
06/04/2014	3.3	<p>Wizard 3.3 release. Updated to Vivado Design Suite 2014.2 throughout.</p> <p>Chapter 4: Updated Figure 4-1, and Figure 4-5 to Figure 4-16.</p>
04/02/2014	3.2	<p>Wizard 3.2 release. Updated to Vivado Design Suite 2014.1 throughout. Added Zynq SoCs to LogiCORE IP Facts Table.</p> <p>Chapter 2: Added I/O column to Table 2-2, Table 2-3, and Table 2-4. In Table 2-5, updated column heading to SRIO Gen2. Updated DRP port in Table 2-6.</p> <p>Chapter 4: Updated Figure 4-1, and Figure 4-3 to Figure 4-16. Removed Figure 4-5: "Starting a New Project," Figure 4-6: "Locating the Transceiver Wizard," and Figure 4-7: "Locating the Transceiver Wizard (Vivado Tools)." Updated Configuring and Generating the Wrapper.</p> <p>Appendix B: Updated Overview of Major Changes. Removed Figure B-1: Include Shared Logic in Core, and Figure B-2: Include Shared Logic in Example Design.</p> <p>Appendix C: Updated Debugging Using Embedded BERT.</p>
12/18/2013	3.1	<p>Wizard 3.1 release. Updated to Vivado Design Suite 2013.4 throughout.</p> <p>Chapter 1: Updated GTZ support list in Feature Summary.</p> <p>Chapter 2: In Table 2-1, replaced DRP_CLK_IN with DRP_CLK_IN_P/DRP_CLK_IN_N and updated description. Added Table 2-6 and Table 2-7.</p> <p>Chapter 3: In Table 3-1, replaced DRPCLK with DRP_CLK_IN_P/DRP_CLK_IN_N and updated description.</p> <p>Chapter 4: Updated Figure 4-1, and Figure 4-6 to Figure 4-16.</p> <p>Chapter 5: Updated XDC file output in Example Design.</p> <p>Appendix B: Updated Figure B-1, Figure B-2, and Figure B-4.</p> <p>Appendix C: Updated filename in Hardware Debug. Added bullet after Figure C-3. Added step 3 to GTX Transceiver Validation. Updated Figure C-7.</p>
10/02/2013	3.0	<p>Initial release of this core as a product guide. This new guide is based on UG769.</p>

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage

suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at <https://www.xilinx.com/legal.htm#tos>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at <https://www.xilinx.com/legal.htm#tos>.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

©Copyright 2013-2022 Xilinx, Inc. Xilinx, the Xilinx logo, Alveo, Artix, Kintex, Kria, Spartan, Versal, Virtex, Vitis, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. PCI, PCIe and PCI Express are trademarks of PCI-SIG and used under license. V-by-One HS is a trademark of THine Electronics, Inc. CPRI is a trademark of Siemens AG. All other trademarks are the property of their respective owners.