UltraScale FPGAs Transceivers Wizard v1.7

LogiCORE IP Product Guide

Vivado Design Suite

PG182 (v1.7) December 4, 2020

XILINX

Table of Contents

Introduction

The UltraScale™ FPGAs Transceivers Wizard IP core helps configure one or more serial transceivers. You can target an industry standard using provided configuration presets, or start from scratch. The flexible Transceivers Wizard generates a customized IP core for the transceivers, configuration options, and enabled ports you have selected, optionally including a variety of helper blocks to simplify common functionality. In addition, the wizard can produce an example design for simple simulation and hardware usage demonstration.

Features

- Transceiver configuration presets for industry standards
- Simple and intuitive feature selection flow
- Automatically sets transceiver parameters
- Advanced options to tune performance
- Transceiver site and reference clock selection interface
- Available helper blocks to simplify common or complex transceiver usage
- Optional exposure of any transceiver port depending upon the selected configuration
- Example design with configurable PRBS generator, checker, and link status indicator to demonstrate functionality in simulation and hardware
- Flexible placement of each helper block: within core for simplicity, or within example design for user customization
- Support for UltraScale and UltraScale+™ architectures

LogiCORE™ IP Facts Table	
Core Specifics	
Supported Device Family(1)	UltraScale+™ Families Kintex® UltraScale FPGA Virtex® UltraScale FPGA
Supported User Interfaces	Not Applicable
Resources	See Table 2-2.
Provided with Core	
Design Files	RTL
Example Design	Verilog
Test Bench	Verilog
Constraints File	Xilinx Design Constraints (XDC)
Simulation Model	Source HDL with SecureIP transceiver simulation models
Supported S/W Driver	Not Provided
Tested Design Flows	
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 Record: 57487
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.

Overview

The UltraScale™ FPGAs Transceivers Wizard is used to configure and simplify the use of one or more serial transceivers in a Xilinx® UltraScale or UltraScale+™ device. See Chapter 2, Product Specification for a detailed description of the core.

This document describes the Wizard IP core. See the *UltraScale Architecture GTH Transceivers User Guide* (UG576) [Ref 1] or *UltraScale Architecture GTY Transceivers User Guide* (UG578) [Ref 2] for details on the specific use and behavior of the serial transceivers.

Feature Summary

The wizard provides the following features:

- Customization flow driven by the Vivado Integrated Design Environment (IDE), providing high-level choices that configure supported transceiver features and automatically set primitive parameters, as appropriate
- Variety of transceiver configuration preset selections to target industry standards
- Advanced configuration options to tune transceiver performance
- Transceiver site, reference clock, and recovered clock selection interface for enabling one or more transceiver channels and adherence to clock routing restrictions
- Optional feature configuration interface for comma detection and alignment, channel bonding, clock correction, buffer control, advanced clocking, and some protocol-specific features
- Available helper blocks to simplify common or complex transceiver usage, and the choice to either include or exclude each helper block from the core
    - Helper blocks excluded from the core are delivered as user-customizable starting points within the example design
- Ability to locate enabled transceiver common primitives either within the core or in the example design, and connectivity to simplify resource sharing across multiple cores
- Optional port enablement interface provides the ability to expose any transceiver primitive port as a top-level core port. However, these ports should not be in conflict with any dependent helper core location and configuration of the wizard.

- Synthesizable example design with configurable pseudo-random binary sequence (PRBS) data generator, checker, and link status indicator logic to quickly demonstrate core and transceiver functionality in simulation and hardware:
  - Simulation test bench that monitors example design PRBS lock in loopback, and indicates resulting link status
  - Virtual input/output (VIO) core instance that simplifies basic example design hardware bring-up, and key debug signal probing
  - Additional convenience features, including differential reference clock buffer instantiation and wiring, and per-channel vector slicing
- Core and example design level Xilinx design constraints (XDC) files with timing, location, and other constraints as necessary for the selected configuration

Applications

The wizard is the supported method of configuring and using one or more serial transceivers in a Xilinx UltraScale FPGA.

Licensing and Ordering Information

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

Product Specification

The UltraScale™ FPGAs Transceivers Wizard core is the supported method of configuring and using one or more serial transceivers in a Xilinx® UltraScale or UltraScale+™ device. In addition to automatically setting primitive parameters as appropriate for your application, the wizard simplifies serial transceiver usage by providing a variety of port enablement and helper block convenience functions. These concepts, as well as technical specifications, are described in this chapter.

Wizard Basic Concepts

Transceiver primitives. Fundamentally, the wizard instantiates, configures, and connects one or more serial transceiver primitives to provide a simplified user interface to those resources. The core instance configures the channel and common primitives by applying HDL parameter values derived from the Vivado® Integrated Design Environment (IDE)-driven customization of that instance.

Transceiver configuration presets. During Vivado IDE-driven customization, you can choose from a variety of transceiver configuration presets to target an industry standard. If required, customization settings can be further modified to suit your application.

Optional port enablement. Xilinx serial transceiver primitives have many ports, and most ports are usually not required for any one use mode. The wizard provides access to all transceiver primitive ports using an optional port enablement interface, but by default offers a compact user interface by exposing only those ports likely to be necessary for the core as customized. Some of the ports might not be applicable to be exposed from GT wizard core due to the customization and optional enablement of some of the helper cores.

Helper blocks. The wizard provides optional modules called helper blocks that abstract or automate certain common or complex transceiver usage procedures. Each helper block can be located either within the core or outside it. They are delivered with the example design as a user-modifiable starting point. Helper blocks in this release include:

- *Reset controller*. Controls and abstracts the transceiver reset sequence.
- *Transmitter user clocking network*. Contains resources to drive the transmitter user clocking network.

- *Receiver user clocking network*. Contains resources to drive the receiver user clocking network.
- *User data width sizing*. Sizes the transmitter and receiver data vectors to the specified user widths.
- *Transmitter buffer bypass controller*. Controls and abstracts the transmitter buffer bypass procedure, if required.
- *Receiver buffer bypass controller*. Controls and abstracts the receiver buffer bypass procedure, if required.

The wizard is intended to simplify the use of the serial transceivers. However, it is still important to understand the behavior, usage, and any limitations of the transceivers. See the *UltraScale Architecture GTH Transceivers User Guide* (UG576) [Ref 1] or *UltraScale Architecture GTY Transceivers User Guide* (UG578) [Ref 2] for details.

Figure 2-1 and its description illustrate the basic concepts of the wizard in the context of the core hierarchy.

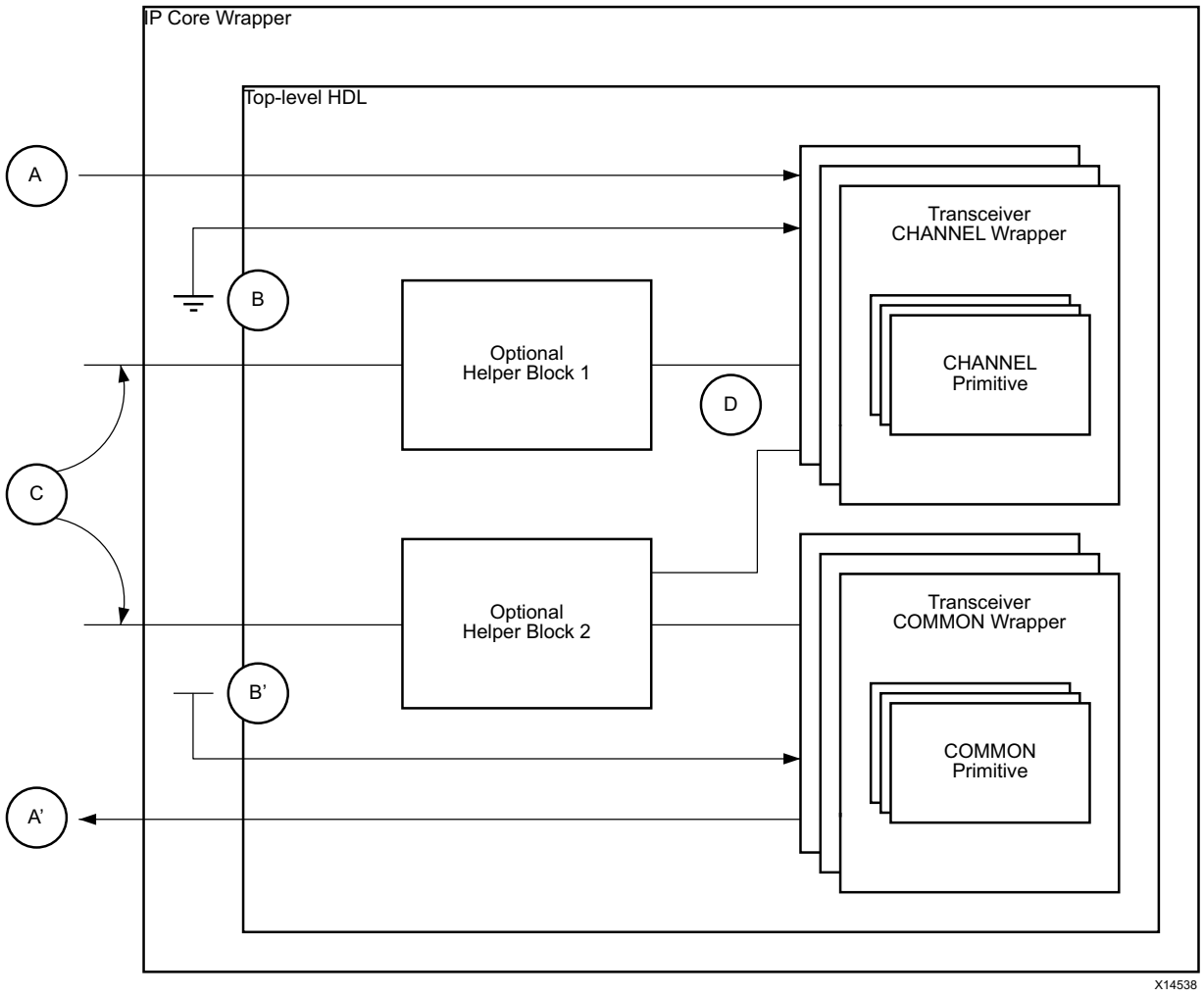

Figure 2-1: **Wizard IP Core Block Diagram**

Transceiver channel primitives and transceiver common primitives are instantiated by the transceiver channel wrapper and transceiver common wrapper modules, respectively. One or more wrapper modules can be used to instantiate those transceiver primitives as required for your application. The wrapper modules apply appropriate parameter values to the underlying transceiver primitives based on the choices made during IP customization, or according to the selected transceiver configuration preset. These wrappers, like the rest of the core hierarchy, should not be user-modified.

To provide a compact user interface, only those transceiver primitive ports that are likely needed for the selected configuration are exposed as wizard IP core-level ports by default. Input vector **A** represents an enabled core port that drives a corresponding input port of one or more transceiver channel primitives. Likewise, output vector **A'** is driven by a corresponding output port of one or more transceiver common primitives. If not enabled by default, user-required ports can be individually enabled during IP customization for maximum flexibility.

Transceiver primitive input ports that are not exposed through the core boundary are tied off to their appropriate values (per the core customization) within the Vivado Design Suite IP core wrapper. Net **B** represents an input port of one or more transceiver channel primitives that is not enabled as a core port and is automatically tied Low by the wizard. Net **B'** represents an analogous transceiver common primitive input, tied High.

The wizard provides optional helper blocks to simplify common or complex transceiver usage, and each helper block can be located either within the core or within the user-modifiable example design. Vector **C** represents the simple user interface of the optional helper blocks when located within the core, while nets **D** represents the more complex interface between those helper blocks and the transceiver channel and/or common primitives to which they connect.

Performance

The wizard is designed to operate in coordination with the performance characteristics of the transceiver primitives it instantiates.

Maximum Frequencies

For the serial transceiver switching characteristics and the serial transceiver user clock switching characteristics, see the applicable data sheet for your device:

- *Kintex UltraScale Architecture Data Sheet: DC and AC Switching Characteristics* (DS892) [Ref 3]
- *Virtex UltraScale Architecture Data Sheet: DC and AC Switching Characteristics* (DS893) [Ref 4]
- *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925) [Ref 5]
- *Kintex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics* (DS922) [Ref 6]
- *Virtex UltraScale+ FPGA Data Sheet: DC and AC Switching Characteristics* (DS923) [Ref 7]

You must use the frequency ranges specified by these documents for proper transceiver and core operation.

IMPORTANT: *A free-running clock input, `gtwiz_reset_clk_freerun_in`, is required by the reset controller helper block to reset the transceiver primitives. In GTH transceiver core configurations targeting engineering sample (ES1 or ES2) UltraScale devices and GTHE4, GTYE4 UltraScale+ devices where the CPLL is used, this clock must also drive each bit of the `drpclk_in` port. As shown in Table 2-1, the maximum frequency of this clock must not exceed either an upper bound or the slowest of the transceiver channels' user clock frequencies for the core as customized. The precise frequency of the free-running clock which is specified during IP customization should not change because of the dependencies in the CPLL calibration block. For more details, see Chapter 4, Customizing and Generating the Core. The free-running clock must not be derived from user clocks or their sources.*

Table 2-1: **Free-Running Clock Maximum Frequency**

Transceiver User Clock Frequency Relationship	Maximum Frequency of gtwiz_reset_clk_freerun_in
$F_{RXUSRCLK2} \leq F_{TXUSRCLK2}$	The lower of F_{UPPER} [(1)] or $F_{RXUSRCLK2}$
$F_{RXUSRCLK2} > F_{TXUSRCLK2}$	The lower of F_{UPPER} [(1)] or $F_{TXUSRCLK2}$

Notes:

1. F_{UPPER} is 200 MHz for GTH transceiver core configurations targeting engineering sample (ES1 or ES2) UltraScale devices where the CPLL is used; or 250 MHz for other configurations. For UltraScale+ devices, this is 250 MHz.

Other Performance Characteristics

See the *UltraScale Architecture GTH Transceivers User Guide* (UG576) [Ref 1] or *UltraScale Architecture GTY Transceivers User Guide* (UG578) [Ref 2] for other performance characteristics of the transceiver primitives. Also, see the following datasheets for any other device level details:

- Kintex UltraScale Architecture Data Sheet: DC and AC Switching Characteristics (DS892) [Ref 3]
- Virtex UltraScale Architecture Data Sheet: DC and AC Switching Characteristics (DS893) [Ref 4]
- Kintex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics (DS922) [Ref 6]
- Virtex UltraScale+ FPGA Data Sheet: DC and AC Switching Characteristics (DS923) [Ref 7]
- Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics (DS925) [Ref 5]

Resource Utilization

The basic Wizard HDL is highly structural and uses a negligible amount of device resources to instantiate and wire the transceiver primitives for use. In GTH transceiver configurations targeting engineering sample (ES1 or ES2) UltraScale devices where the CPLL is used as either the transmitter PLL type, receiver PLL type, or as the source of a selectable TXOUTCLK frequency, CPLL calibration logic is included. One BUFG_GT and approximately 280 LUTs and 285 flip-flops are utilized per enabled transceiver channel in these configurations.

The device utilization of the optional helper blocks is shown in Table 2-2. These resources are only consumed when the relevant helper block is enabled and used within the core, or otherwise included in your design. Resources are shown per helper block instance, although most configurations that enable a helper block use only one instance.

Table 2-2: **Resource Utilization of Helper Blocks**

Helper Block		Device Resources (per Helper Block Instance)		
Type	**Configuration**	**LUTs**	**Flip-Flops**	**Clock Buffers**
Reset controller	Any	120	195	0(1)
Transmitter user clocking network	$F_{TXUSRCLK} = F_{TXUSRCLK2}$	0	2	1 (BUFG_GT)
	$F_{TXUSRCLK} \neq F_{TXUSRCLK2}$	0	2	2 (BUFG_GT)
Receiver user clocking network	$F_{RXUSRCLK} = F_{RXUSRCLK2}$	0	2	1 (BUFG_GT)
	$F_{RXUSRCLK} \neq F_{RXUSRCLK2}$	0	2	2 (BUFG_GT)
Transmitter buffer bypass controller	Single-lane	8	25	0
	Multi-lane	<20	25	0
Receiver buffer bypass controller	Single-lane	8	25	0
	Multi-lane	<20	25	0
User data width sizing	Any	0	0	0

Notes:
1. A shareable BUFG for the free-running clock is not included in the helper block HDL.

Resources required are derived from post-synthesis reports and may change during implementation.

Port Descriptions

The wizard enables access to underlying transceiver primitive ports as needed, as well as providing a user interface to enable the helper blocks that are included within the core instance. As such, the wizard user interface can vary significantly between different customizations.

To provide a compact interface, only those transceiver primitive ports that are likely needed for the selected customization are exposed as Wizard IP core-level ports. Additional user-required ports can be individually enabled during IP customization using a flexible optional port enablement interface. See Chapter 4, Customizing and Generating the Core, for details on optional port enablement.

The presence and location of helper blocks also affects the core user interface. When a helper block is enabled and located within the core, a simple user interface is available at the core boundary instead of at the transceiver primitive ports to which it connects. When the helper block is located within the example design, the more complex transceiver primitive ports it connects to are necessarily enabled at the core boundary. Figure 2-2 illustrates how helper block location affects core port enablement.

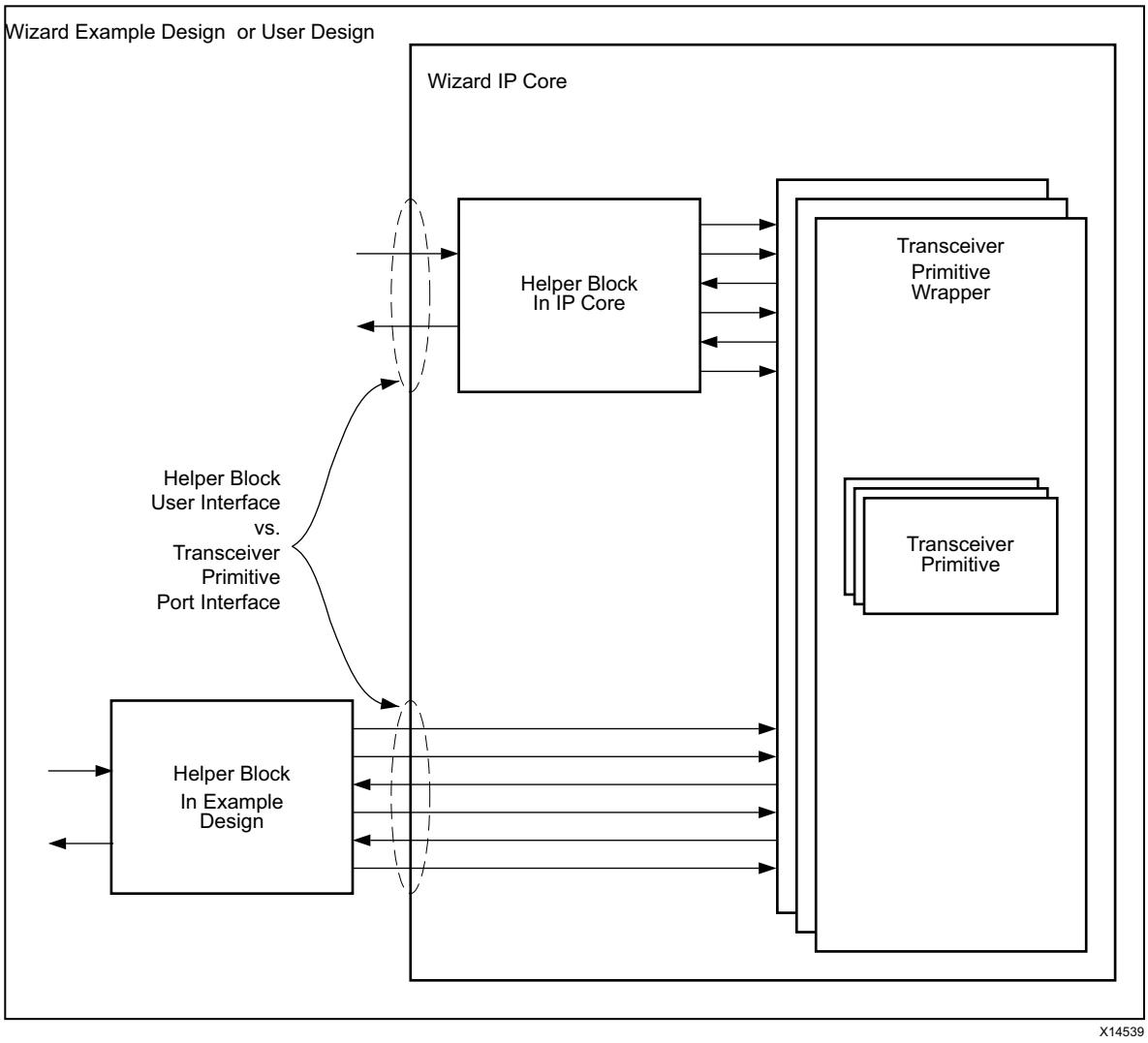

Figure 2-2: **Effect of Port Helper Block Location on Port Enablement**

Reset Controller Helper Block Ports

The reset controller helper block contains a user interface and a transceiver interface. The user interface provides a simple means of initiating and monitoring the completion of transceiver reset procedures. The transceiver interface implements the signaling required to control the various transceiver primitive reset sequences.

Reset controller helper block user interface ports can be identified by the prefix *gtwiz_reset_*. For guidance on the usage of the reset controller helper block, see Chapter 3, Designing with the Core.

The reset controller helper block user interface ports described in Table 2-3 are present on the Wizard IP core instance when it is configured to locate the reset controller helper block in the core. They are also present on the helper block itself, directly accessible when the helper block is located in the example design.

Table 2-3: **Reset Controller Helper Block User Interface Ports on Core (Helper Block in Core)**

Name	Direction	Width	Clock Domain	Description
gtwiz_reset_clk_freerun_in	Input	1		Free-running clock used to reset transceiver primitives. Must be toggling prior to device configuration. See Performance, page 10 for maximum frequency guidance.
gtwiz_reset_all_in	Input	1	Async	User signal to reset the phase-locked loops (PLLs) and active data directions of transceiver primitives. The falling edge of an active-High, asynchronous pulse of at least one gtwiz_reset_clk_freerun_in period in duration initializes the process.
gtwiz_reset_tx_pll_and_datapath_in	Input	1	Async	User signal to reset the transmit data direction and associated PLLs of transceiver primitives. An active-High, asynchronous pulse of at least one gtwiz_reset_clk_freerun_in period in duration initializes the process.
gtwiz_reset_tx_datapath_in	Input	1	Async	User signal to reset the transmit data direction of transceiver primitives. An active-High, asynchronous pulse of at least one gtwiz_reset_clk_freerun_in period in duration initializes the process.

Table 2-3: **Reset Controller Helper Block User Interface Ports on Core (Helper Block in Core)**

Name	Direction	Width	Clock Domain	Description
gtwiz_reset_rx_pll_and_datapath_in	Input	1	Async	User signal to reset the receive data direction and associated PLLs of transceiver primitives. An active-High, asynchronous pulse of at least one gtwiz_reset_clk_freerun_in period in duration initializes the process.
gtwiz_reset_rx_datapath_in	Input	1	Async	User signal to reset the receive data direction of transceiver primitives. An active-High, asynchronous pulse of at least one gtwiz_reset_clk_freerun_in period in duration initializes the process.
gtwiz_reset_rx_cdr_stable_out	Output	1	gtwiz_reset_clk_freerun_in	Active-High indication that the clock and data recovery (CDR) circuits of the transceiver primitives are stable. Reserved; do not use.
gtwiz_reset_qpll0lock_in	Input	1 × Num. commons	Async	QPLL0 lock signal, present when the transceiver common is located in the example design and QPLL0 is used as either the transmitter or receiver PLL type.
gtwiz_reset_qpll1lock_in	Input	1 × Num. commons	Async	QPLL1 lock signal, present when the transceiver common is located in the example design and QPLL1 is used as either the transmitter or receiver PLL type.
gtwiz_reset_tx_done_out	Output	1	TXUSRCLK2 of TX master channel	Active-High indication that the transmitter reset sequence of transceiver primitives as initiated by the reset controller helper block has completed.
gtwiz_reset_rx_done_out	Output	1	RXUSRCLK2 of RX master channel	Active-High indication that the receiver reset sequence of transceiver primitives as initiated by the reset controller helper block has completed.

Table 2-3: **Reset Controller Helper Block User Interface Ports on Core (Helper Block in Core)**

Name	Direction	Width	Clock Domain	Description
gtwiz_reset_qpll0reset_out	Output	1 × Num. commons	gtwiz_reset_clk_freerun_in	QPLL0 reset signal, present when the transceiver common is located in the example design and QPLL0 is used as either the transmitter or receiver PLL type.
gtwiz_reset_qpll1reset_out	Output	1 × Num. commons	gtwiz_reset_clk_freerun_in	QPLL1 reset signal, present when the transceiver common is located in the example design and QPLL1 is used as either the transmitter or receiver PLL type.

The reset controller helper block user interface ports described in Table 2-4 are present on the core instance when it is configured to locate the reset controller helper block in the example design.

Table 2-4: **Reset Controller Helper Block User Interface Ports on Core (Helper Block in Example Design)**

Name	Direction	Width	Clock Domain	Description
gtwiz_reset_tx_done_in	Input	1	Async	Upon successful completion of the transmitter reset sequence, this active-High port must be asserted to allow dependent helper blocks within the core to operate. The reset controller helper block drives this port by default.
gtwiz_reset_rx_done_in	Input	1	Async	Upon successful completion of the receiver reset sequence, this active-High port must be asserted to allow dependent helper blocks within the core to operate. The reset controller helper block drives this port by default.

The reset controller helper block user interface ports described in Table 2-5 are not present on the core instance, but are present on the reset controller helper block itself when it is included in the example design.

Table 2-5: **Other Reset Controller Helper Block User Interface Ports (Helper Block in Example Design)**

Name	Direction	Width	Clock Domain	Description
gtwiz_reset_userclk_tx_active_in	Input	1	Async	When the TXUSRCLK and TXUSRCLK2 signals that drive transceiver primitives are active and stable, this active-High port must be asserted for the transmitter reset sequence to complete. The transmitter user clocking network helper block drives this port by default.
gtwiz_reset_userclk_rx_active_in	Input	1	Async	When the RXUSRCLK and RXUSRCLK2 signals that drive transceiver primitives are active and stable, this active-High port must be asserted to allow the receiver reset sequence to complete. The receiver user clocking network helper block drives this port by default.

The reset controller helper block transceiver interface ports described in Table 2-6 connect the reset controller helper block to transceiver primitives. When the helper block is located within the core, these connections are internal and the transceiver primitive inputs that are driven by helper block outputs cannot be enabled as optional ports on the core instance. Inversely, when the helper block is located in the example design, the connections cross the core boundary so the transceiver primitive ports that connect to the helper block are enabled by necessity.

Table 2-6: **Reset Controller Helper Block Transceiver Interface Ports**

Name	Direction	Width	Clock Domain	Description
gtpowergood_in	Input	1	Async	Logical AND of all GTPOWERGOOD signals produced by transceiver channel logic.
txusrclk2_in	Input	1		TXUSRCLK2 of master transceiver channel.
plllock_tx_in	Input	1	Async	Logical AND of all lock signals produced by PLLs that clock the transmit datapath of transceiver channel primitives.
txresetdone_in	Input	1	Async	Logical AND of all TXRESETDONE signals produced by transceiver channel primitives.
rxusrclk2_in	Input	1		RXUSRCLK2 of master transceiver channel.
plllock_rx_in	Input	1	Async	Logical AND of all lock signals produced by PLLs that clock the receive datapath of transceiver channel primitives.

Table 2-6: **Reset Controller Helper Block Transceiver Interface Ports *(Cont'd)***

Name	Direction	Width	Clock Domain	Description
rxcdrlock_in	Input	1	Async	Logical AND of all RXCDRLOCK signals produced by transceiver channel primitives.
rxresetdone_in	Input	1	Async	Logical AND of all RXRESETDONE signals produced by transceiver channel primitives.
pllreset_tx_out	Output	1	gtwiz_reset_clk_freerun_in (used asynchronously)	Active-High signal fanned out to the reset ports of all PLLs that clock the transmit datapath of transceiver channel primitives.
txprogdivreset_out	Output	1	gtwiz_reset_clk_freerun_in (used asynchronously)	Active-High signal fanned out to TXPROGDIVRESET port of all transceiver channel primitives.
gttxreset_out	Output	1	gtwiz_reset_clk_freerun_in (used asynchronously)	Active-High signal fanned out to GTTXRESET port of all transceiver channel primitives.
txuserrdy_out	Output	1	gtwiz_reset_clk_freerun_in (used asynchronously)	Active-High signal fanned out to TXUSERRDY port of all transceiver channel primitives.
pllreset_rx_out	Output	1	gtwiz_reset_clk_freerun_in (used asynchronously)	Active-High signal fanned out to the reset ports of all PLLs that clock the receive datapath of transceiver channel primitives.
rxprogdivreset_out	Output	1	gtwiz_reset_clk_freerun_in (used asynchronously)	Active-High signal fanned out to RXPROGDIVRESET port of all transceiver channel primitives.
gtrxreset_out	Output	1	gtwiz_reset_clk_freerun_in (used asynchronously)	Active-High signal fanned out to GTRXRESET port of all transceiver channel primitives.
rxuserrdy_out	Output	1	gtwiz_reset_clk_freerun_in (used asynchronously)	Active-High signal fanned out to RXUSERRDY port of all transceiver channel primitives.

Note: All input/output ports which are described as async, are synchronized to `gtwiz_reset_clk_freerun_in` in the example design. In user designs, all asynchronous signals coming as inputs to the IP, should be asserted for sufficient time. This ensures that the synchronizers present inside the IP sampling on the `gtwiz_reset_clk_freerun_in` identify the toggles on these ports.

The reset controller helper block ports described in Table 2-7 must be tied off. By default, appropriate tie-offs are provided for each core customization.

Table 2-7: **Reset Controller Helper Block Tie-off Ports**

Name	Direction	Width	Clock Domain	Description
tx_enabled_tie_in	Input	1	gtwiz_reset_clk_freerun_in	When tied High, transmitter resources are reset as part of the sequence in response to gtwiz_reset_all_in.
rx_enabled_tie_in	Input	1	gtwiz_reset_clk_freerun_in	When tied High, receiver resources are reset as part of the sequence in response to gtwiz_reset_all_in.
shared_pll_tie_in	Input	1	gtwiz_reset_clk_freerun_in	When tied High, the shared PLL is reset only once as part of the sequence in response to gtwiz_reset_all_in.

Transmitter User Clocking Network Helper Block Ports

The transmitter user clocking network helper block provides a single interface with a source clock input port driven by a transceiver primitive-based output clock. Transmitter user clocking network helper block ports can be identified by the prefix *gtwiz_userclk_tx_*. For guidance on the usage of the transmitter user clocking network helper block, see Chapter 3, Designing with the Core.

The transmitter user clocking network helper block ports described in Table 2-8 are present on the Wizard IP core instance when it is configured to locate the transmitter user clocking network helper block in the core.

Table 2-8: **Transmitter User Clocking Network Helper Block Ports on Core (Helper Block in Core)**

Name	Direction	Width	Clock Domain	Description
gtwiz_userclk_tx_reset_in	Input	1	Async	User signal to reset the clocking resources within the helper block. The active-High assertion should remain until gtwiz_userclk_tx_srcclk_in/out is stable.
gtwiz_userclk_tx_srcclk_out	Output	1		Transceiver primitive-based clock source used to derive and buffer TXUSRCLK and TXUSRCLK2 outputs.
gtwiz_userclk_tx_usrclk_out	Output	1		Drives TXUSRCLK of transceiver channel primitives. Derived from gtwiz_userclk_tx_srcclk_in/out, buffered and divided as necessary by BUFG_GT primitive.

Table 2-8: **Transmitter User Clocking Network Helper Block Ports on Core (Helper Block in Core)**

Name	Direction	Width	Clock Domain	Description
gtwiz_userclk_tx_usrclk2_out	Output	1		Drives TXUSRCLK2 of transceiver channel primitives. Derived from gtwiz_userclk_tx_srcclk_in/out, buffered and divided as necessary by BUFG_GT primitive if required.
gtwiz_userclk_tx_active_out	Output	1	gtwiz_userclk_tx_usrclk2_out	Active-High indication that the clocking resources within the helper block are not held in reset.

The transmitter user clocking network helper block ports described in Table 2-9 are present on the core instance when it is configured to locate the transmitter user clocking network helper block in the example design.

Table 2-9: **Transmitter User Clocking Network Helper Block User Interface Ports on Core (Helper Block in Example Design)**

Name	Direction	Width	Clock Domain	Description
gtwiz_userclk_tx_active_in	Input	1	Async	When the clocks produced by the transmitter user clocking network helper block are active, this active-High port must be asserted to allow dependent helper blocks within the core to operate. The transmitter user clocking network helper block drives this port by default.
gtwiz_userclk_tx_reset_in	Input	1	Async	This core port is present in GTH transceiver configurations targeting engineering sample (ES1 or ES2) UltraScale devices where the CPLL is used. It must be driven identically to the gtwiz_userclk_tx_reset_in port on the transmitter user clocking network helper block, present in the example design.

The transmitter user clocking network helper block ports described in Table 2-10 are not present on the core instance but are present on the transmitter user clocking network helper block itself when it is included in the example design.

Table 2-10: **Other Transmitter User Clocking Network Helper Block User Interface Ports (Helper Block in Example Design)**

Name	Direction	Width	Clock Domain	Description
gtwiz_userclk_tx_srcclk_in	Input	1		Transceiver primitive-based clock source used to derive and buffer TXUSRCLK and TXUSRCLK2 outputs.

Receiver User Clocking Network Helper Block Ports

The receiver user clocking network helper block provides a single interface with a source clock input port driven by a transceiver primitive-based output clock. Receiver user clocking network helper block ports can be identified by the prefix *gtwiz_userclk_rx_*. For guidance on the usage of the receiver user clocking network helper block, see Chapter 3, Designing with the Core.

The receiver user clocking network helper block ports described in Table 2-11 are present on the wizard core instance when it is configured to locate the receiver user clocking network helper block in the core.

Table 2-11: **Receiver User Clocking Network Helper Block Ports on Core (Helper Block in Core)**

Name	Direction	Width	Clock Domain	Description
gtwiz_userclk_rx_reset_in	Input	1	Async	User signal to reset the clocking resources within the helper block. The active-High assertion should remain until gtwiz_userclk_rx_srcclk_in/out is stable.
gtwiz_userclk_rx_srcclk_out	Output	1		Transceiver primitive-based clock source used to derive and buffer the RXUSRCLK and RXUSRCLK2 outputs.
gtwiz_userclk_rx_usrclk_out	Output	1		Drives RXUSRCLK of transceiver channel primitives. Derived from gtwiz_userclk_rx_srcclk_in/out, buffered and divided as necessary by BUFG_GT primitive.
gtwiz_userclk_rx_usrclk2_out	Output	1		Drives RXUSRCLK2 of transceiver channel primitives. Derived from gtwiz_userclk_rx_srcclk_in/out, buffered and divided as necessary by BUFG_GT primitive if required.
gtwiz_userclk_rx_active_out	Output	1	gtwiz_userclk_rx_usrclk2_out	Active-High indication that the clocking resources within the helper block are not held in reset.

The receiver user clocking network helper block ports described in Table 2-12 are present on the core instance when it is configured to locate the receiver user clocking network helper block in the example design.

Table 2-12: **Receiver User Clocking Network Helper Block User Interface Ports on Core (Helper Block in Example Design)**

Name	Direction	Width	Clock Domain	Description
gtwiz_userclk_rx_active_in	Input	1	Async	When the clocks produced by the receiver user clocking network helper block are active, this active-High port must be asserted to allow dependent helper blocks within the core to operate. The receiver user clocking network helper block drives this port by default.

The receiver user clocking network helper block ports described in Table 2-13 are not present on the core instance, but are present on the receiver user clocking network helper block itself when it is included in the example design.

Table 2-13: **Other Receiver User Clocking Network Helper Block User Interface Ports (Helper Block in Example Design)**

Name	Direction	Width	Clock Domain	Description
gtwiz_userclk_rx_srcclk_in	Input	1		Transceiver primitive-based clock source used to derive and buffer RXUSRCLK and RXUSRCLK2 outputs.

User Data Width Sizing Helper Block Ports

The user data width sizing helper block consists of two modules: one for the transmitter user data interface, and the other for the receiver user data interface. Each module contains a user interface and a transceiver interface. The user interface provides a single vector sized to user data width and scaled by the number of transceiver channels. The transceiver interface implements the bit assignments and any interleaving or de-interleaving required for interfacing to the data transmission and reception ports on the transceiver channel primitives.

User data width sizing helper block user interface ports can be identified by the prefix *gtwiz_userdata_*. See Chapter 3, Designing with the Core, for information about the user data width sizing helper block. The user and transceiver interfaces of the helper block transmitter and receiver modules are described in Table 2-14 through Table 2-17.

Table 2-14: **User Data Width Sizing Helper Block User Interface Ports (Transmitter Module)**

Name	Direction	Width	Clock Domain	Description
gtwiz_userdata_tx_in	Input	TX user data width × Num. channels	TXUSRCLK2, per channel	User interface for data to be transmitted by transceiver channels

Table 2-15: **User Data Width Sizing Helper Block User Interface Ports (Receiver Module)**

Name	Direction	Width	Clock Domain	Description
gtwiz_userdata_rx_out	Output	RX user data width × Num. channels	RXUSRCLK2, per channel	User interface for data received by transceiver channels

Table 2-16: **User Data Width Sizing Helper Block Transceiver Interface Ports (Transmitter Module)**

Name	Direction	Width	Clock Domain	Description
txdata_out	Output	128 × Num. channels	TXUSRCLK2, per channel	Connects to TXDATA on transceiver channel primitives
txctrl0_out	Output	16 × Num. channels	TXUSRCLK2, per channel	Connects to TXCTRL0 on transceiver channel primitives
txctrl1_out	Output	16 × Num. channels	TXUSRCLK2, per channel	Connects to TXCTRL1 on transceiver channel primitives

Table 2-17: **User Data Width Sizing Helper Block Transceiver Interface Ports (Receiver Module)**

Name	Direction	Width	Clock Domain	Description
rxdata_in	Input	128 × Num. channels	RXUSRCLK2, per channel	Connects to RXDATA on transceiver channel primitives
rxctrl0_out	Input	16 × Num. channels	RXUSRCLK2, per channel	Connects to RXCTRL0 on transceiver channel primitives
rxctrl1_out	Input	16 × Num. channels	RXUSRCLK2, per channel	Connects to RXCTRL1 on transceiver channel primitives

Transmitter Buffer Bypass Controller Helper Block Ports

The transmitter buffer bypass controller helper block contains a user interface and a transceiver interface. The user interface provides a simple means of initiating and monitoring the status of the transceiver transmitter buffer bypass procedure. The

transceiver interface implements the signaling required to control the transceiver primitive buffer bypass sequence.

Transmitter buffer bypass helper block user interface ports can be identified by the prefix *gtwiz_buffbypass_tx_*. For guidance on the usage of the transmitter buffer bypass controller helper block, see Chapter 3, Designing with the Core.

The transmitter buffer bypass controller helper block user interface ports described in Table 2-18 are present on the Wizard IP core instance when it is configured to locate the transmitter buffer bypass controller helper block in the core. The ports are also present on the helper block itself, directly accessible when the helper block is located in the example design.

In this configuration, the helper block clock port `gtwiz_buffbypass_tx_clk_in` is driven internal to the core by the same source that drives `TXUSRCLK2` of the transmitter master channel, and is not exposed.

Table 2-18: **Transmitter Buffer Bypass Controller Helper Block User Interface Ports on Core (Helper Block in Core)**

Name	Direction	Width	Clock Domain	Description
gtwiz_buffbypass_tx_reset_in	Input	1	gtwiz_buffbypass_tx_clk_in	User signal to reset the logic within the helper block. An active-High, synchronous pulse should be provided immediately after TXUSRCLK2 stabilizes for all transceiver channels.
gtwiz_buffbypass_tx_start_user_in	Input	1	gtwiz_buffbypass_tx_clk_in	Active-High user signal that is synchronously pulsed to force the transmitter buffer bypass procedure to restart. Hold Low when not used.
gtwiz_buffbypass_tx_done_out	Output	1	gtwiz_buffbypass_tx_clk_in	Active-High indicates that the transmitter buffer bypass procedure has completed.
gtwiz_buffbypass_tx_error_out	Output	1	gtwiz_buffbypass_tx_clk_in	Active-High indicates that the transmitter buffer bypass helper block encountered an error condition.

The transmitter buffer bypass controller helper block user interface ports described in Table 2-19 are not present on the core instance but are present on the transmitter buffer bypass controller helper block when it is included in the example design.

Table 2-19: **Other Transmitter Buffer Bypass Controller Helper Block User Interface Ports (Helper Block in Example Design)**

Name	Direction	Width	Clock Domain	Description
gtwiz_buffbypass_tx_clk_in	Input	1		Transceiver primitive-based clock used to control the transmitter buffer bypass controller helper block. Must be driven by the same source that drives TXUSRCLK2 of the transmitter master channel.
gtwiz_buffbypass_tx_resetdone_in	Input	1	Async	Active-High indicates that transmitter reset sequence has completed, which allows the buffer bypass procedure to begin.

The transmitter buffer bypass controller helper block transceiver interface ports (described in Table 2-20) connect the transmitter buffer bypass controller helper block to transceiver primitives. When the helper block is located within the core, these connections are internal, and the transceiver primitive inputs that are driven by helper block outputs cannot be enabled as optional ports on the core instance. Inversely, when the helper block is located in the example design, the connections cross the core boundary, and the transceiver primitive ports that connect to the helper block are enabled by necessity.

To implement the multi-lane buffer bypass procedure, the port width of each signal scales with the number of transceiver channels that the transmitter buffer bypass controller helper block interfaces to.

Table 2-20: **Transmitter Buffer Bypass Controller Helper Block Transceiver Interface Port**

Name	Direction	Width	Clock Domain	Description
txphaligndone_in	Input	1 × Num. channels	Async	Connects to TXPHALIGNDONE of transceiver channel primitives
txphinitdone_in	Input	1 × Num. channels	Async	Connects to TXPHINITDONE of transceiver channel primitives
txdlysresetdone_in	Input	1 × Num. channels	Async	Connects to TXDLYSRESETDONE of transceiver channel primitives
txsyncout_in	Input	1 × Num. channels	Async	Connects to TXSYNCOUT of transceiver channel primitives

Table 2-20: **Transmitter Buffer Bypass Controller Helper Block Transceiver Interface Port**

Name	Direction	Width	Clock Domain	Description
txsyncdone_in	Input	1 × Num. channels	Async	Connects to TXSYNCDONE of transceiver channel primitives
txphdlyreset_out	Output	1 × Num. channels	Tied off	Connects to TXPHDLYRESET of transceiver channel primitives
txphalign_out	Output	1 × Num. channels	Tied off	Connects to TXPHALIGN of transceiver channel primitives
txphalignen_out	Output	1 × Num. channels	Tied off	Connects to TXPHALIGNEN of transceiver channel primitives
txphdlypd_out	Output	1 × Num. channels	Tied off	Connects to TXPHDLYPD of transceiver channel primitives
txphinit_out	Output	1 × Num. channels	Tied off	Connects to TXPHINIT of transceiver channel primitives
txphovrden_out	Output	1 × Num. channels	Tied off	Connects to TXPHOVRDEN of transceiver channel primitives
txdlysreset_out	Output	1 × Num. channels	gtwiz_buffbypass_tx_clk_in (used asynchronously)	Connects to TXDLYSRESET of transceiver channel primitives
txdlybypass_out	Output	1 × Num. channels	Tied off	Connects to TXDLYBYPASS of transceiver channel primitives
txdlyen_out	Output	1 × Num. channels	Tied off	Connects to TXDLYEN of transceiver channel primitives
txdlyovrden_out	Output	1 × Num. channels	Tied off	Connects to TXDLYOVRDEN of transceiver channel primitives
txphdlytstclk_out	Output	1 × Num. channels	Tied off	Connects to TXPHDLYTSTCLK of transceiver channel primitives

Table 2-20: **Transmitter Buffer Bypass Controller Helper Block Transceiver Interface Port**

Name	Direction	Width	Clock Domain	Description
txdlyhold_out	Output	1 × Num. channels	Tied off	Connects to TXDLYHOLD of transceiver channel primitives
txdlyupdown_out	Output	1 × Num. channels	Tied off	Connects to TXDLYUPDOWN of transceiver channel primitives
txsyncmode_out	Output	1 × Num. channels	Tied off	Connects to TXSYNCMODE of transceiver channel primitives
txsyncallin_out	Output	1 × Num. channels	Async	Connects to TXSYNCALLIN of transceiver channel primitives
txsyncin_out	Output	1 × Num. channels	Async	Connects to TXSYNCIN of transceiver channel primitives

Receiver Buffer Bypass Controller Helper Block Ports

The receiver buffer bypass controller helper block contains a user interface and a transceiver interface. The user interface provides a simple means of initiating and monitoring the status of the transceiver receiver buffer bypass procedure. The transceiver interface implements the signaling required to control the transceiver primitive buffer bypass sequence.

Receiver buffer bypass helper block user interface ports can be identified by the prefix *gtwiz_buffbypass_rx_*. For guidance on the usage of the receiver buffer bypass controller helper block, see Chapter 3, Designing with the Core.

The receiver buffer bypass controller helper block user interface ports described in Table 2-21 are present on the Wizard IP core instance when it is configured to locate the receiver buffer bypass controller helper block in the core. The ports are also present on the helper block itself, directly accessible when the helper block is located in the example design.

In this configuration, the helper block clock port `gtwiz_buffbypass_rx_clk_in` is driven internal to the core by the same source that drives `RXUSRCLK2` of the receiver master channel, and is therefore not exposed. When the single-lane buffer bypass procedure is used, one instance of the helper block exists per transceiver channel. The width of each port scales by this factor, and each helper block instance is clocked by the same source that drives `RXUSRCLK2` of the associated channel.

Table 2-21: **Receiver Buffer Bypass Controller Helper Block User Interface Ports on Core (Helper Block in Core)**

Name	Direction	Width	Clock Domain	Description
gtwiz_buffbypass_rx_reset_in	Input	1	gtwiz_buffbypass_rx_clk_in	User signal to reset the logic within the helper block. An active-High, synchronous pulse should be provided immediately after RXUSRCLK2 stabilizes for all transceiver channels.
gtwiz_buffbypass_rx_start_user_in	Input	1	gtwiz_buffbypass_rx_clk_in	Active-High user signal that is synchronously pulsed to force the receiver buffer bypass procedure to restart. Hold Low when not used.
gtwiz_buffbypass_rx_done_out	Output	1	gtwiz_buffbypass_rx_clk_in	Active-High indicates that the receiver buffer bypass procedure has completed.
gtwiz_buffbypass_rx_error_out	Output	1	gtwiz_buffbypass_rx_clk_in	Active-High indicates that the receiver buffer bypass helper block encountered an error condition.

The receiver buffer bypass controller helper block user interface ports described in Table 2-22 are not present on the core instance but are present on the receiver buffer bypass controller helper block itself when it is included in the example design.

Table 2-22: **Other Receiver Buffer Bypass Controller Helper Block User Interface Ports (Helper Block in Example Design)**

Name	Direction	Width	Clock Domain	Description
gtwiz_buffbypass_rx_clk_in	Input	1		Transceiver primitive-based clock used to control the receiver buffer bypass controller helper block. Must be driven by the same source that drives RXUSRCLK2 of the receiver master channel.
gtwiz_buffbypass_rx_resetdone_in	Input	1	Async	Active-High indication that receiver reset sequence has completed, which allows the buffer bypass procedure to begin.

The receiver buffer bypass controller helper block transceiver interface ports described in Table 2-23 connect the receiver buffer bypass controller helper block to transceiver primitives. When the helper block is located within the core, these connections are internal,

and the transceiver primitive inputs that are driven by helper block outputs cannot be enabled as optional ports on the core instance. Inversely, when the helper block is located in the example design, the connections cross the core boundary, so the transceiver primitive ports that connect to the helper block are enabled by necessity.

To implement the multi-lane buffer bypass procedure, the width of each port scales with the number of transceiver channels that the receiver buffer bypass controller helper block interfaces to. When the single-lane buffer bypass procedure is used, one instance of the helper block exists per transceiver channel, so the scaling factor is 1.

Table 2-23: **Receiver Buffer Bypass Controller Helper Block Transceiver Interface Ports**

Name	Direction	Width	Clock Domain	Description
rxphaligndone_in	Input	1 × Num. channels	Async	Connects to RXPHALIGNDONE of transceiver channel primitives
rxdlysresetdone_in	Input	1 × Num. channels	Async	Connects to RXDLYSRESETDONE of transceiver channel primitives
rxsyncout_in	Input	1 × Num. channels	Async	Connects to RXSYNCOUT of transceiver channel primitives
rxsyncdone_in	Input	1 × Num. channels	Async	Connects to RXSYNCDONE of transceiver channel primitives
rxphdlyreset_out	Output	1 × Num. channels	Tied off	Connects to RXPHDLYRESET of transceiver channel primitives
rxphalign_out	Output	1 × Num. channels	Tied off	Connects to RXPHALIGN of transceiver channel primitives
rxphalignen_out	Output	1 × Num. channels	Tied off	Connects to RXPHALIGNEN of transceiver channel primitives
rxphdlypd_out	Output	1 × Num. channels	Tied off	Connects to RXPHDLYPD of transceiver channel primitives
rxphovrden_out	Output	1 × Num. channels	Tied off	Connects to RXPHOVRDEN of transceiver channel primitives
rxdlysreset_out	Output	1 × Num. channels	gtwiz_buffbypass_rx_clk_in (used asynchronously)	Connects to RXDLYSRESET of transceiver channel primitives

Table 2-23: **Receiver Buffer Bypass Controller Helper Block Transceiver Interface Ports *(Cont'd)***

Name	Direction	Width	Clock Domain	Description
rxdlybypass_out	Output	1 × Num. channels	Tied off	Connects to RXDLYBYPASS of transceiver channel primitives
rxdlyen_out	Output	1 × Num. channels	Tied off	Connects to RXDLYEN of transceiver channel primitives
rxdlyovrden_out	Output	1 × Num. channels	Tied off	Connects to RXDLYOVRDEN of transceiver channel primitives
rxsyncmode_out	Output	1 × Num. channels	Tied off	Connects to RXSYNCMODE of transceiver channel primitives
rxsyncallin_out	Output	1 × Num. channels	Async	Connects to RXSYNCALLIN of transceiver channel primitives
rxsyncin_out	Output	1 × Num. channels	Tied off	Connects to RXSYNCIN of transceiver channel primitives

Transceiver Common Ports

A subset of the ports (described in Table 2-24) is present on the wizard core instance when it is configured with one or more active transceiver common primitives, and when those primitives are located within the core. These ports connect through the core hierarchy to the corresponding transceiver common primitive ports. Because only a subset is required for a given core customization, most are not exposed as ports on the core interface by default. See Chapter 4, Customizing and Generating the Core, for details on optional port enablement.

The width of each port scales with the number of transceiver common primitives instantiated within the core instance. The least significant bit(s) correspond to the first enabled transceiver common primitive in increasing grid order, where the Y axis increments before X. For example, the `QPLL0REFCLKSEL` port on a transceiver common primitive is 3 bits in size. In a hypothetical Wizard IP core customization that instantiates three GTH transceiver common primitives in physical positions GTHE3_COMMON_X0Y2, GTHE3_COMMON_X0Y5, and GTHE3_COMMON_X1Y3, the `qpll0refclksel_in` port on the core instance is sized [8:0], where:

- `qpll0refclksel_in[2:0]` connects to the transceiver common instance at location GTHE3_COMMON_X0Y2.
- `qpll0refclksel_in[5:3]` connects to the transceiver common instance at location GTHE3_COMMON_X0Y5.

- `qpll0refclksel_in[8:6]` connects to the transceiver common instance at location GTHE3_COMMON_X1Y3.

By vectoring in this manner, the user interface of the core is compact and predictable. As a convenience, the example design also provides per-primitive signals that are assigned the relevant bit slices of the concatenated vectors. See Chapter 5, Example Design, for more details on example design features.

This document does not provide guidance on the usage of the transceiver primitive ports. See the *UltraScale Architecture GTH Transceivers User Guide* (UG576) [Ref 1] or *UltraScale Architecture GTY Transceivers User Guide* (UG578) [Ref 2] for relevant details.

Table 2-24: **Transceiver Common Ports**

Name	Direction	Width	Description
bgbypassb_in	Input	1 × Num. commons	Connects to BGBYPASSB on transceiver common primitives
bgmonitorenb_in	Input	1 × Num. commons	Connects to BGMONITORENB on transceiver common primitives
bgpdb_in	Input	1 × Num. commons	Connects to BGPDB on transceiver common primitives
bgrcalovrd_in	Input	5 × Num. commons	Connects to BGRCALOVRD on transceiver common primitives
bgrcalovrdenb_in	Input	1 × Num. commons	Connects to BGRCALOVRDENB on transceiver common primitives
drpaddr_common_in	Input	9 × Num. commons (GTHE3) 10 × Num. commons (GTYE3) 16 × Num. commons (GTHE4) 16 × Num. commons (GTYE4)	Connects to DRPADDR on transceiver common primitives
drpclk_common_in	Input	1 × Num. commons	Connects to DRPCLK on transceiver common primitives
drpdi_common_in	Input	16 × Num. commons	Connects to DRPDI on transceiver common primitives
drpen_common_in	Input	1 × Num. commons	Connects to DRPEN on transceiver common primitives
drpwe_common_in	Input	1 × Num. commons	Connects to DRPWE on transceiver common primitives
gtgrefclk0_in	Input	1 × Num. commons	Connects to GTGREFCLK0 on transceiver common primitives and the use of this port is reserved for test purposes only.
gtgrefclk1_in	Input	1 × Num. commons	Connects to GTGREFCLK1 on transceiver common primitives and the use of this port is reserved for test purposes only.
gtnorthrefclk00_in	Input	1 × Num. commons	Connects to GTNORTHREFCLK00 on transceiver common primitives

Table 2-24: **Transceiver Common Ports *(Cont'd)***

Name	Direction	Width	Description
gtnorthrefclk01_in	Input	1 × Num. commons	Connects to GTNORTHREFCLK01 on transceiver common primitives
gtnorthrefclk10_in	Input	1 × Num. commons	Connects to GTNORTHREFCLK10 on transceiver common primitives
gtnorthrefclk11_in	Input	1 × Num. commons	Connects to GTNORTHREFCLK11 on transceiver common primitives
gtrefclk00_in	Input	1 × Num. commons	Connects to GTREFCLK00 on transceiver common primitives
gtrefclk01_in	Input	1 × Num. commons	Connects to GTREFCLK01 on transceiver common primitives
gtrefclk10_in	Input	1 × Num. commons	Connects to GTREFCLK10 on transceiver common primitives
gtrefclk11_in	Input	1 × Num. commons	Connects to GTREFCLK11 on transceiver common primitives
gtsouthrefclk00_in	Input	1 × Num. commons	Connects to GTSOUTHREFCLK00 on transceiver common primitives
gtsouthrefclk01_in	Input	1 × Num. commons	Connects to GTSOUTHREFCLK01 on transceiver common primitives
gtsouthrefclk10_in	Input	1 × Num. commons	Connects to GTSOUTHREFCLK10 on transceiver common primitives
gtsouthrefclk11_in	Input	1 × Num. commons	Connects to GTSOUTHREFCLK11 on transceiver common primitives
pcierateqpll0_in	Input	3 × Num. commons	Connects to PCIERATEQPLL0 on transceiver common primitives (GTHE4 and GTYE4 only)
pcierateqpll1_in	Input	3 × Num. commons	Connects to PCIERATEQPLL1 on transceiver common primitives (GTHE4 and GTYE4 only)
pmarsvd0_in	Input	8 × Num. commons	Connects to PMARSVD0 on transceiver common primitives
pmarsvd1_in	Input	8 × Num. commons	Connects to PMARSVD1 on transceiver common primitives
qpll0clkrsvd0_in	Input	1 × Num. commons	Connects to QPLL0CLKRSVD0 on transceiver common primitives
qpll0clkrsvd1_in	Input	1 × Num. commons	Connects to QPLL0CLKRSVD1 on transceiver common primitives (GTHE3, GTHE4, and GTYE4 only)
qpll0fbdiv_in	Input	8 × Num. commons	Connects to QPLL0FBDIV on transceiver common primitives (GTHE4 and GTYE4 only)
qpll0lockdetclk_in	Input	1 × Num. commons	Connects to QPLL0LOCKDETCLK on transceiver common primitives

Table 2-24: **Transceiver Common Ports *(Cont'd)***

Name	Direction	Width	Description
qpll0locken_in	Input	1 × Num. commons	Connects to QPLL0LOCKEN on transceiver common primitives
qpll0pd_in	Input	1 × Num. commons	Connects to QPLL0PD on transceiver common primitives
qpll0refclksel_in	Input	3 × Num. commons	Connects to QPLL0REFCLKSEL on transceiver common primitives
qpll0reset_in	Input	1 × Num. commons	Connects to QPLL0RESET on transceiver common primitives
qpll1clkrsvd0_in	Input	1 × Num. commons	Connects to QPLL1CLKRSVD0 on transceiver common primitives
qpll1clkrsvd1_in	Input	1 × Num. commons	Connects to QPLL1CLKRSVD1 on transceiver common primitives (GTHE3, GTHE4, and GTYE4 only)
qpll1fbdiv_in	Input	8 × Num. commons	Connects to QPLL1FBDIV on transceiver common primitives (GTHE4 and GTYE4 only)
qpll1lockdetclk_in	Input	1 × Num. commons	Connects to QPLL1LOCKDETCLK on transceiver common primitives
qpll1locken_in	Input	1 × Num. commons	Connects to QPLL1LOCKEN on transceiver common primitives
qpll1pd_in	Input	1 × Num. commons	Connects to QPLL1PD on transceiver common primitives
qpll1refclksel_in	Input	3 × Num. commons	Connects to QPLL1REFCLKSEL on transceiver common primitives
qpll1reset_in	Input	1 × Num. commons	Connects to QPLL1RESET on transceiver common primitives
qpllrsvd1_in	Input	8 × Num. commons	Connects to QPLLRSVD1 on transceiver common primitives
qpllrsvd2_in	Input	5 × Num. commons	Connects to QPLLRSVD2 on transceiver common primitives
qpllrsvd3_in	Input	5 × Num. commons	Connects to QPLLRSVD3 on transceiver common primitives
qpllrsvd4_in	Input	8 × Num. commons	Connects to QPLLRSVD4 on transceiver common primitives
rcalenb_in	Input	1 × Num. commons	Connects to RCALENB on transceiver common primitives
sdm0data_in	Input	25 × Num. commons	Connects to SDM0DATA on transceiver common primitives (GTYE3, GTHE4, and GTYE4 only)
sdm0reset_in	Input	1 × Num. commons	Connects to SDM0RESET on transceiver common primitives (GTYE3, GTHE4, and GTYE4 only)

Table 2-24: **Transceiver Common Ports *(Cont'd)***

Name	Direction	Width	Description
sdm0toggle_in		1 × Num. commons	Connects to SDM0TOGGLE on transceiver common primitives (GTHE4 and GTYE4 only)
sdm0width_in	Input	2 × Num. commons	Connects to SDM0WIDTH on transceiver common primitives (GTYE3, GTHE4, and GTYE4 only)
sdm1data_in	Input	25 × Num. commons	Connects to SDM1DATA on transceiver common primitives (GTYE3, GTHE4, and GTYE4 only)
sdm1reset_in	Input	1 × Num. commons	Connects to SDM1RESET on transceiver common primitives (GTYE3, GTHE4, and GTYE4 only)
sdm1toggle_in	Input	1 × Num. commons	Connects to SDM1TOGGLE on transceiver common primitives (GTHE4 and GTYE4 only)
sdm1width_in	Input	2 × Num. commons	Connects to SDM1WIDTH on transceiver common primitives (GTYE3, GTHE4, and GTYE4 only)
tcongpi_in	Input	10 × Num. commons	Connects to TCONGPI on transceiver common primitives (GTHE4 only)
tconpowerup_in	Input	1 × Num. commons	Connects to TCONPOWERUP on transceiver common primitives (GTHE4 only)
tconreset_in	Input	2 × Num. commons	Connects to TCONRESET on transceiver common primitives (GTHE4 only)
tconrsvdin1_in	Input	2 × Num. commons	Connects to TCONRSVDIN1 on transceiver common primitives (GTHE4 only)
ubcfgstreamen_in	Input	1 × Num. commons	Connects to UBCFGSTREAMEN on transceiver common primitives (GTYE4 only)
ubdo_in	Input	16 × Num. commons	Connects to UBDO on transceiver common primitives (GTYE4 only)
ubdrdy_in	Input	1 × Num. commons	Connects to UBDRDY on transceiver common primitives (GTYE4 only)
ubenable_in	Input	1 × Num. commons	Connects to UBENABLE on transceiver common primitives (GTYE4 only)
ubgpi_in	Input	2 × Num. commons	Connects to UBGPI on transceiver common primitives (GTYE4 only)
ubintr_in	Input	2 × Num. commons	Connects to UBINTR on transceiver common primitives (GTYE4 only)

Table 2-24: **Transceiver Common Ports *(Cont'd)***

Name	Direction	Width	Description
ubiolmbrst_in	Input	1 × Num. commons	Connects to UBIOLMBRST on transceiver common primitives (GTYE4 only)
ubmbrst_in	Input	1 × Num. commons	Connects to UBMBRST on transceiver common primitives (GTYE4 only)
ubmdmcapture_in	Input	1 × Num. commons	Connects to UBMDMCAPTURE on transceiver common primitives (GTYE4 only)
ubmdmdbgrst_in	Input	1 × Num. commons	Connects to UBMDMDBGRST on transceiver common primitives (GTYE4 only)
ubmdmdbgupdate_in	Input	1 × Num. commons	Connects to UBMDMDBGUPDATE on transceiver common primitives (GTYE4 only)
ubmdmregen_in	Input	4 × Num. commons	Connects to UBMDMREGEN on transceiver common primitives (GTYE4 only)
ubmdmshift_in	Input	1 × Num. commons	Connects to UBMDMSHIFT on transceiver common primitives (GTYE4 only)
ubmdmsysrst_in	Input	1 × Num. commons	Connects to UBMDMSYSRST on transceiver common primitives (GTYE4 only)
ubmdmtck_in	Input	1 × Num. commons	Connects to UBMDMTCK on transceiver common primitives (GTYE4 only)
ubmdmtdi_in	Input	1 × Num. commons	Connects to UBMDMTDI on transceiver common primitives (GTYE4 only)
drpdo_common_out	Output	16 × Num. commons	Connects to DRPDO on transceiver common primitives
drprdy_common_out	Output	1 × Num. commons	Connects to DRPRDY on transceiver common primitives
pmarsvdout0_out	Output	8 × Num. commons	Connects to PMARSVDOUT0 on transceiver common primitives
pmarsvdout1_out	Output	8 × Num. commons	Connects to PMARSVDOUT1 on transceiver common primitives
qpll0fbclklost_out	Output	1 × Num. commons	Connects to QPLL0FBCLKLOST on transceiver common primitives
qpll0lock_out	Output	1 × Num. commons	Connects to QPLL0LOCK on transceiver common primitives
qpll0outclk_out	Output	1 × Num. commons	Connects to QPLL0OUTCLK on transceiver common primitives

Table 2-24: **Transceiver Common Ports *(Cont'd)***

Name	Direction	Width	Description
qpll0outrefclk_out	Output	1 × Num. commons	Connects to QPLL0OUTREFCLK on transceiver common primitives
qpll0refclklost_out	Output	1 × Num. commons	Connects to QPLL0REFCLKLOST on transceiver common primitives
qpll1fbclklost_out	Output	1 × Num. commons	Connects to QPLL1FBCLKLOST on transceiver common primitives
qpll1lock_out	Output	1 × Num. commons	Connects to QPLL1LOCK on transceiver common primitives
qpll1outclk_out	Output	1 × Num. commons	Connects to QPLL1OUTCLK on transceiver common primitives
qpll1outrefclk_out	Output	1 × Num. commons	Connects to QPLL1OUTREFCLK on transceiver common primitives
qpll1refclklost_out	Output	1 × Num. commons	Connects to QPLL1REFCLKLOST on transceiver common primitives
qplldmonitor0_out	Output	8 × Num. commons	Connects to QPLLDMONITOR0 on transceiver common primitives
qplldmonitor1_out	Output	8 × Num. commons	Connects to QPLLDMONITOR1 on transceiver common primitives
refclkoutmonitor0_out	Output	1 × Num. commons	Connects to REFCLKOUTMONITOR0 on transceiver common primitives
refclkoutmonitor1_out	Output	1 × Num. commons	Connects to REFCLKOUTMONITOR1 on transceiver common primitives
rxrecclk0_sel_out	Output	2 × Num. commons	Connects to RXRECCLK0_SEL on transceiver common primitives (GTHE3 and GTYE3 only)
rxrecclk1_sel_out	Output	2 × Num. commons	Connects to RXRECCLK1_SEL on transceiver common primitives (GTHE3 and GTYE3 only)
sdm0finalout_out	Output	4 × Num. commons	Connects to SDM0FINALOUT on transceiver common primitives (GTE3, GTHE4, and GTYE4Y only)
sdm0testdata_out	Output	15 × Num. commons	Connects to SDM0TESTDATA on transceiver common primitives (GTYE3, GTHE4, and GTYE4 only)
rxrecclk0sel_out	Output	2 × Num. commons	Connects to RXRECCLK0SEL on transceiver common primitives (GTHE4 and GTYE4 only)
rxrecclk1sel_out	Output	2 × Num. commons	Connects to RXRECCLK1SEL on transceiver common primitives (GTHE4 and GTYE4 only)
sdm1finalout_out	Output	4 × Num. commons	Connects to SDM1FINALOUT on transceiver common primitives (GTYE3, GTHE4, and GTYE4 only)

Table 2-24: **Transceiver Common Ports *(Cont'd)***

Name	Direction	Width	Description
sdm1testdata_out	Output	15 × Num. commons	Connects to SDM1TESTDATA on transceiver common primitives (GTYE3, GTHE4, and GTYE4 only)
tcongpo_out	Output	10 × Num. commons	Connects to TCONGPO on transceiver common primitives (GTHE4 only)
tconrsvdout0_out	Output	1 × Num. commons	Connects to TCONRSVDOUT0 on transceiver common primitives (GTHE4 only)
ubdaddr_out	Output	16 × Num. commons	Connects to UBDADDR on transceiver common primitives (GTYE4 only)
ubden_out	Output	1 × Num. commons	Connects to UBDEN on transceiver common primitives (GTYE4 only)
ubdi_out	Output	16 × Num. commons	Connects to UBDI on transceiver common primitives (GTYE4 only)
ubdwe_out	Output	1 × Num. commons	Connects to UBDWE on transceiver common primitives (GTYE4 only)
ubmdmtdo_out	Output	1 × Num. commons	Connects to UBMDMTDO on transceiver common primitives (GTYE4 only)
ubrsvdout_out	Output	1 × Num. commons	Connects to UBRSVDOUT on transceiver common primitives (GTYE4 only)
ubtxuart_out	Output	1 × Num. commons	Connects to UBTXUART on transceiver common primitives (GTYE4 only)

Transceiver Channel Ports

A subset of the ports described in Table 2-25 is present on the wizard core instance. These ports connect through the core hierarchy to the corresponding transceiver channel primitive ports. Because only a subset is required for a given core customization, most are not exposed as ports on the core interface by default. See Chapter 4, Customizing and Generating the Core, for details on optional port enablement.

The width of each port scales with the number of transceiver channel primitives instantiated within the core instance. The least significant bit(s) correspond to the first enabled transceiver channel primitive in increasing grid order, where the Y axis increments before X. For example, the `TXDIFFCTRL` port on a transceiver channel primitive is 4 bits in size. In a hypothetical Wizard IP core customization which instantiates three GTH transceiver channel primitives in physical positions GTHE3_CHANNEL_X0Y3, GTHE3_CHANNEL_X0Y10, and GTHE3_CHANNEL_X1Y0, the `txdiffctrl_in` port on the core instance will be sized [11:0], where:

- `txdiffctrl_in[3:0]` connects to the transceiver channel instance at location GTHE3_CHANNEL_X0Y3.
- `txdiffctrl_in[7:4]` connects to the transceiver channel instance at location GTHE3_CHANNEL_X0Y10.
- `txdiffctrl_in[11:8]` connects to the transceiver channel instance at location GTHE3_CHANNEL_X1Y0.

By vectoring in this manner, the user interface of the core is compact and predictable. As a convenience, the example design also provides per-primitive signals, which are assigned the relevant bit slices of the concatenated vectors. See Chapter 5, Example Design, for more details on example design features.

This document does not provide guidance on the usage of the transceiver primitive ports. See the *UltraScale Architecture GTH Transceivers User Guide* (UG576) [Ref 1] or *UltraScale Architecture GTY Transceivers User Guide* (UG578) [Ref 2] for relevant details.

Table 2-25: **Transceiver Channel Ports**

Name	Direction	Width	Description
cdrstepdir_in	Input	1 × Num. channels	Connects to CDRSTEPDIR on transceiver channel primitives (GTYE3, GTHE4, and GTYE4 only)
cdrstepsq_in	Input	1 × Num. channels	Connects to CDRSTEPSQ on transceiver channel primitives (GTYE3, GTHE4, and GTYE4 only)
cdrstepsx_in	Input	1 × Num. channels	Connects to CDRSTEPSX on transceiver channel primitives (GTYE3, GTHE4, and GTYE4 only)
cfgreset_in	Input	1 × Num. channels	Connects to CFGRESET on transceiver channel primitives
clkrsvd0_in	Input	1 × Num. channels	Connects to CLKRSVD0 on transceiver channel primitives
clkrsvd1_in	Input	1 × Num. channels	Connects to CLKRSVD1 on transceiver channel primitives
cpllfreqlock_in	Input	1 × Num. channels	Connects to CPLLFREQLOCK on transceiver channel primitives (GTHE4 and GTYE4 only)
cplllockdetclk_in	Input	1 × Num. channels	Connects to CPLLLOCKDETCLK on transceiver channel primitives
cplllocken_in	Input	1 × Num. channels	Connects to CPLLLOCKEN on transceiver channel primitives
cpllpd_in	Input	1 × Num. channels	Connects to CPLLPD on transceiver channel primitives and for configurations that require CPLL Calibration block, this port resets the CPLL calibration logic.

Table 2-25: **Transceiver Channel Ports *(Cont'd)***

Name	Direction	Width	Description
cpllrefclksel_in	Input	3 × Num. channels	Connects to CPLLREFCLKSEL on transceiver channel primitives
cpllreset_in	Input	1 × Num. channels	Connects to CPLLRESET on transceiver channel primitives
dmonfiforeset_in	Input	1 × Num. channels	Connects to DMONFIFORESET on transceiver channel primitives
dmonitorclk_in	Input	1 × Num. channels	Connects to DMONITORCLK on transceiver channel primitives
drpaddr_in	Input	drp_addr_width x Num. Channels	Connects to DRPADDR on transceiver channel primitives See *UltraScale Architecture GTH Transceivers User Guide* (UG576) [Ref 1] and *UltraScale Architecture GTY Transceivers User Guide* (UG578) [Ref 2] for more information on `drp_addr_width`.
drpclk_in	Input	1 × Num. channels	Connects to DRPCLK on transceiver channel primitives. It is recommended to connect this port to the same stable source as `gtwiz_reset_clk_freerun_in`.
drpdi_in	Input	16 × Num. channels	Connects to DRPDI on transceiver channel primitives
drpen_in	Input	1 × Num. channels	Connects to DRPEN on transceiver channel primitives
drprst_in	Input	1 × Num. channels	Connects to DRPRST on transceiver channel primitives (GTHE4 and GTYE4 only)
drpwe_in	Input	1 × Num. channels	Connects to DRPWE on transceiver channel primitives
elpcaldvorwren_in	Input	1 × Num. channels	Connects to ELPCALDVORWREN on transceiver channel primitives (GTYE3 only)
elpcalpaorwren_in	Input	1 × Num. channels	Connects to ELPCALPAORWREN on transceiver channel primitives (GTYE3 only)
evoddphicaldone_in	Input	1 × Num. channels	Connects to EVODDPHICALDONE on transceiver channel primitives (GTHE3 and GTYE3 only)
evoddphicalstart_in	Input	1 × Num. channels	Connects to EVODDPHICALSTART on transceiver channel primitives (GTHE3 and GTYE3 only)
evoddphidrden_in	Input	1 × Num. channels	Connects to EVODDPHIDRDEN on transceiver channel primitives (GTHE3 and GTYE3 only)

Table 2-25: **Transceiver Channel Ports *(Cont'd)***

Name	Direction	Width	Description
evoddphidwren_in	Input	1 × Num. channels	Connects to EVODDPHIDWREN on transceiver channel primitives (GTHE3 and GTYE3 only)
evoddphixrden_in	Input	1 × Num. channels	Connects to EVODDPHIXRDEN on transceiver channel primitives (GTHE3 and GTYE3 only)
evoddphixwren_in	Input	1 × Num. channels	Connects to EVODDPHIXWREN on transceiver channel primitives (GTHE3 and GTYE3 only)
eyescanmode_in	Input	1 × Num. channels	Connects to EYESCANMODE on transceiver channel primitives (GTHE3 and GTYE3 only)
eyescanreset_in	Input	1 × Num. channels	Connects to EYESCANRESET on transceiver channel primitives
eyescantrigger_in	Input	1 × Num. channels	Connects to EYESCANTRIGGER on transceiver channel primitives
freqos_in	Input	1 × Num. channels	Connects to FREQOS on transceiver channel primitives (GTHE4 and GTYE4 only)
gtgrefclk_in	Input	1 × Num. channels	Connects to GTGREFCLK on transceiver channel primitives and the use of this port is reserved for test purposes only.
gthrxn_in	Input	1 × Num. channels	Connects to GTHRXN on transceiver channel primitives (GTH only)
gthrxp_in	Input	1 × Num. channels	Connects to GTHRXP on transceiver channel primitives (GTH only)
gtnorthrefclk0_in	Input	1 × Num. channels	Connects to GTNORTHREFCLK0 on transceiver channel primitives
gtnorthrefclk1_in	Input	1 × Num. channels	Connects to GTNORTHREFCLK1 on transceiver channel primitives
gtrefclk0_in	Input	1 × Num. channels	Connects to GTREFCLK0 on transceiver channel primitives
gtrefclk1_in	Input	1 × Num. channels	Connects to GTREFCLK1 on transceiver channel primitives
gtresetsel_in	Input	1 × Num. channels	Connects to GTRESETSEL on transceiver channel primitives (GTHE3 and GTYE3 only)
gtrsvd_in	Input	16 × Num. channels	Connects to GTRSVD on transceiver channel primitives
gtrxreset_in	Input	1 × Num. channels	Connects to GTRXRESET on transceiver channel primitives

Table 2-25: **Transceiver Channel Ports *(Cont'd)***

Name	Direction	Width	Description
gtrxresetsel_in	Input	1 × Num. channels	Connects to GTRXRESETSEL on transceiver channel primitives (GTHE4 and GTYE4 only)
gtsouthrefclk0_in	Input	1 × Num. channels	Connects to GTSOUTHREFCLK0 on transceiver channel primitives
gtsouthrefclk1_in	Input	1 × Num. channels	Connects to GTSOUTHREFCLK1 on transceiver channel primitives
gttxreset_in	Input	1 × Num. channels	Connects to GTTXRESET on transceiver channel primitives
gttxresetsel_in	Input	1 × Num. channels	Connects to GTTXRESETSEL on transceiver channel primitives (GTHE4 and GTYE4 only)
incpctrl_in	Input	1 × Num. channels	Connects to INCPCTRL on transceiver channel primitives (GTHE4 and GTYE4 only)
gtyrxn_in	Input	1 × Num. channels	Connects to GTYRXN on transceiver channel primitives (GTY only)
gtyrxp_in	Input	1 × Num. channels	Connects to GTYRXP on transceiver channel primitives (GTY only)
loopback_in	Input	3 × Num. channels	Connects to LOOPBACK on transceiver channel primitives
looprsvd_in	Input	16 × Num. channels	Connects to LOOPRSVD on transceiver channel primitives (GTYE3 only)
lpbkrxtxseren_in	Input	1 × Num. channels	Connects to LPBKRXTXSEREN on transceiver channel primitives (GTHE3 and GTYE3 only)
lpbktxrxseren_in	Input	1 × Num. channels	Connects to LPBKTXRXSEREN on transceiver channel primitives (GTHE3 and GTYE3 only)
pcieeqrxeqadaptdone_in	Input	1 × Num. channels	Connects to PCIEEQRXEQADAPTDONE on transceiver channel primitives
pcierstidle_in	Input	1 × Num. channels	Connects to PCIERSTIDLE on transceiver channel primitives
pciersttxsyncstart_in	Input	1 × Num. channels	Connects to PCIERSTTXSYNCSTART on transceiver channel primitives
pcieuserratedone_in	Input	1 × Num. channels	Connects to PCIEUSERRATEDONE on transceiver channel primitives
pcsrsvdin_in	Input	16 × Num. channels	Connects to PCSRSVDIN on transceiver channel primitives
pcsrsvdin2_in	Input	5 × Num. channels	Connects to PCSRSVDIN2 on transceiver channel primitives (GTHE3 and GTYE3 only)

Table 2-25: **Transceiver Channel Ports *(Cont'd)***

Name	Direction	Width	Description
pmarsvdin_in	Input	5 × Num. channels	Connects to PMARSVDIN on transceiver channel primitives (GTHE3 and GTYE3 only)
qpll0clk_in	Input	1 × Num. channels	Connects to QPLL0CLK on transceiver channel primitives
qpll0freqlock_in	Input	1 × Num. channels	Connects to QPLL0FREQLOCK on transceiver channel primitives (GTHE4 and GTYE4 only)
qpll0refclk_in	Input	1 × Num. channels	Connects to QPLL0REFCLK on transceiver channel primitives
qpll1clk_in	Input	1 × Num. channels	Connects to QPLL1CLK on transceiver channel primitives
qpll1freqlock_in	Input	1 × Num. channels	Connects to QPLL1FREQLOCK on transceiver channel primitives (GTHE4 and GTYE4 only)
qpll1refclk_in	Input	1 × Num. channels	Connects to QPLL1REFCLK on transceiver channel primitives
resetovrd_in	Input	1 × Num. channels	Connects to RESETOVRD on transceiver channel primitives
rstclkentx_in	Input	1 × Num. channels	Connects to RSTCLKENTX on transceiver channel primitives (GTHE3 and GTYE3 only)
rx8b10ben_in	Input	1 × Num. channels	Connects to RX8B10BEN on transceiver channel primitives
rxafecfoken_in	Input	1 × Num. channels	Connects to RXAFECFOKEN on transceiver channel primitives (GTHE4 and GTYE4 only)
rxbufreset_in	Input	1 × Num. channels	Connects to RXBUFRESET on transceiver channel primitives
rxcdrfreqreset_in	Input	1 × Num. channels	Connects to RXCDRFREQRESET on transceiver channel primitives
rxcdrhold_in	Input	1 × Num. channels	Connects to RXCDRHOLD on transceiver channel primitives
rxcdrovrden_in	Input	1 × Num. channels	Connects to RXCDROVRDEN on transceiver channel primitives
rxcdrreset_in	Input	1 × Num. channels	Connects to RXCDRRESET on transceiver channel primitives
rxcdrresetrsv_in	Input	1 × Num. channels	Connects to RXCDRRESETRSV on transceiver channel primitives (GTHE3 and GTYE3 only)
rxchbonden_in	Input	1 × Num. channels	Connects to RXCHBONDEN on transceiver channel primitives

Table 2-25: **Transceiver Channel Ports *(Cont'd)***

Name	Direction	Width	Description
rxchbondi_in	Input	5 × Num. channels	Connects to RXCHBONDI on transceiver channel primitives
rxchbondlevel_in	Input	3 × Num. channels	Connects to RXCHBONDLEVEL on transceiver channel primitives
rxchbondmaster_in	Input	1 × Num. channels	Connects to RXCHBONDMASTER on transceiver channel primitives
rxchbondslave_in	Input	1 × Num. channels	Connects to RXCHBONDSLAVE on transceiver channel primitives
rxckcalreset_in	Input	1 × Num. channels	Connects to RXCKCALRESET on transceiver channel primitives (GTYE3, GTHE4, and GTYE4 only)
rxckcalstart_in	Input	7 × Num. channels	Connects to RXCKCALSTART on transceiver channel primitives (GTHE4 and GTYE4 only)
rxcommadeten_in	Input	1 × Num. channels	Connects to RXCOMMADETEN on transceiver channel primitives
rxdfeagcctrl_in	Input	2 × Num. channels	Connects to RXDFEAGCCTRL on transceiver channel primitives (GTH only)
rxdccforcestart_in	Input	1 × Num. channels	Connects to RXDCCFORCESTART on transceiver channel primitives (GTYE3 only)
rxdfeagchold_in	Input	1 × Num. channels	Connects to RXDFEAGCHOLD on transceiver channel primitives
rxdfeagcovrden_in	Input	1 × Num. channels	Connects to RXDFEAGCOVRDEN on transceiver channel primitives
rxdfecfokfcnum_in	Input	4 × Num. channels	Connects to RXDFECFOKFCNUM on transceiver channel primitives (GTHE4 and GTYE4 only)
rxdfecfokfen_in	Input	1 × Num. channels	Connects to RXDFECFOKFEN on transceiver channel primitives (GTHE4 and GTYE4 only)
rxdfecfokfpulse_in	Input	1 × Num. channels	Connects to RXDFECFOKFPULSE on transceiver channel primitives (GTHE4 and GTYE4 only)
rxdfecfokhold_in	Input	1 × Num. channels	Connects to RXDFECFOKHOLD on transceiver channel primitives (GTHE4 and GTYE4 only)
rxdfecfokovren_in	Input	1 × Num. channels	Connects to RXDFECFOKOVREN on transceiver channel primitives (GTHE4 and GTYE4 only)
rxdfekhhold_in	Input	1 × Num. channels	Connects to RXDFEKHHOLD on transceiver channel primitives (GTHE4 and GTYE4 only)

Table 2-25: **Transceiver Channel Ports *(Cont'd)***

Name	Direction	Width	Description
rxdfekhovrden_in	Input	1 × Num. channels	Connects to RXDFEKHOVRDEN on transceiver channel primitives (GTHE4 and GTYE4 only)
rxdfelfhold_in	Input	1 × Num. channels	Connects to RXDFELFHOLD on transceiver channel primitives
rxdfelfovrden_in	Input	1 × Num. channels	Connects to RXDFELFOVRDEN on transceiver channel primitives
rxdfelpmreset_in	Input	1 × Num. channels	Connects to RXDFELPMRESET on transceiver channel primitives
rxdfetap10hold_in	Input	1 × Num. channels	Connects to RXDFETAP10HOLD on transceiver channel primitives
rxdfetap10ovrden_in	Input	1 × Num. channels	Connects to RXDFETAP10OVRDEN on transceiver channel primitives
rxdfetap11hold_in	Input	1 × Num. channels	Connects to RXDFETAP11HOLD on transceiver channel primitives
rxdfetap11ovrden_in	Input	1 × Num. channels	Connects to RXDFETAP11OVRDEN on transceiver channel primitives
rxdfetap12hold_in	Input	1 × Num. channels	Connects to RXDFETAP12HOLD on transceiver channel primitives
rxdfetap12ovrden_in	Input	1 × Num. channels	Connects to RXDFETAP12OVRDEN on transceiver channel primitives
rxdfetap13hold_in	Input	1 × Num. channels	Connects to RXDFETAP13HOLD on transceiver channel primitives
rxdfetap13ovrden_in	Input	1 × Num. channels	Connects to RXDFETAP13OVRDEN on transceiver channel primitives
rxdfetap14hold_in	Input	1 × Num. channels	Connects to RXDFETAP14HOLD on transceiver channel primitives
rxdfetap14ovrden_in	Input	1 × Num. channels	Connects to RXDFETAP14OVRDEN on transceiver channel primitives
rxdfetap15hold_in	Input	1 × Num. channels	Connects to RXDFETAP15HOLD on transceiver channel primitives
rxdfetap15ovrden_in	Input	1 × Num. channels	Connects to RXDFETAP15OVRDEN on transceiver channel primitives
rxdfetap2hold_in	Input	1 × Num. channels	Connects to RXDFETAP2HOLD on transceiver channel primitives
rxdfetap2ovrden_in	Input	1 × Num. channels	Connects to RXDFETAP2OVRDEN on transceiver channel primitives
rxdfetap3hold_in	Input	1 × Num. channels	Connects to RXDFETAP3HOLD on transceiver channel primitives
rxdfetap3ovrden_in	Input	1 × Num. channels	Connects to RXDFETAP3OVRDEN on transceiver channel primitives

Table 2-25: **Transceiver Channel Ports *(Cont'd)***

Name	Direction	Width	Description
rxdfetap4hold_in	Input	1 × Num. channels	Connects to RXDFETAP4HOLD on transceiver channel primitives
rxdfetap4ovrden_in	Input	1 × Num. channels	Connects to RXDFETAP4OVRDEN on transceiver channel primitives
rxdfetap5hold_in	Input	1 × Num. channels	Connects to RXDFETAP5HOLD on transceiver channel primitives
rxdfetap5ovrden_in	Input	1 × Num. channels	Connects to RXDFETAP5OVRDEN on transceiver channel primitives
rxdfetap6hold_in	Input	1 × Num. channels	Connects to RXDFETAP6HOLD on transceiver channel primitives
rxdfetap6ovrden_in	Input	1 × Num. channels	Connects to RXDFETAP6OVRDEN on transceiver channel primitives
rxdfetap7hold_in	Input	1 × Num. channels	Connects to RXDFETAP7HOLD on transceiver channel primitives
rxdfetap7ovrden_in	Input	1 × Num. channels	Connects to RXDFETAP7OVRDEN on transceiver channel primitives
rxdfetap8hold_in	Input	1 × Num. channels	Connects to RXDFETAP8HOLD on transceiver channel primitives
rxdfetap8ovrden_in	Input	1 × Num. channels	Connects to RXDFETAP8OVRDEN on transceiver channel primitives
rxdfetap9hold_in	Input	1 × Num. channels	Connects to RXDFETAP9HOLD on transceiver channel primitives
rxdfetap9ovrden_in	Input	1 × Num. channels	Connects to RXDFETAP9OVRDEN on transceiver channel primitives
rxdfeuthold_in	Input	1 × Num. channels	Connects to RXDFEUTHOLD on transceiver channel primitives
rxdfeutovrden_in	Input	1 × Num. channels	Connects to RXDFEUTOVRDEN on transceiver channel primitives
rxdfevphold_in	Input	1 × Num. channels	Connects to RXDFEVPHOLD on transceiver channel primitives
rxdfevpovrden_in	Input	1 × Num. channels	Connects to RXDFEVPOVRDEN on transceiver channel primitives
rxdfevsen_in	Input	1 × Num. channels	Connects to RXDFEVSEN on transceiver channel primitives (GTHE3 and GTYE3 only)
rxdfexyden_in	Input	1 × Num. channels	Connects to RXDFEXYDEN on transceiver channel primitives
rxdlybypass_in	Input	1 × Num. channels	Connects to RXDLYBYPASS on transceiver channel primitives
rxdlyen_in	Input	1 × Num. channels	Connects to RXDLYEN on transceiver channel primitives

Table 2-25: **Transceiver Channel Ports *(Cont'd)***

Name	Direction	Width	Description
rxdlyovrden_in	Input	1 × Num. channels	Connects to RXDLYOVRDEN on transceiver channel primitives
rxdlysreset_in	Input	1 × Num. channels	Connects to RXDLYSRESET on transceiver channel primitives
rxelecidlemode_in	Input	2 × Num. channels	Connects to RXELECIDLEMODE on transceiver channel primitives
rxeqtraining_in	Input	1 × Num. channels	Connects to RXEQTRAINING on transceiver channel primitives (GTHE4 and GTYE4 only)
rxgearboxslip_in	Input	1 × Num. channels	Connects to RXGEARBOXSLIP on transceiver channel primitives
rxlatclk_in	Input	1 × Num. channels	Connects to RXLATCLK on transceiver channel primitives
rxlpmen_in	Input	1 × Num. channels	Connects to RXLPMEN on transceiver channel primitives
rxlpmgchold_in	Input	1 × Num. channels	Connects to RXLPMGCHOLD on transceiver channel primitives
rxlpmgcovrden_in	Input	1 × Num. channels	Connects to RXLPMGCOVRDEN on transceiver channel primitives
rxlpmhfhold_in	Input	1 × Num. channels	Connects to RXLPMHFHOLD on transceiver channel primitives
rxlpmhfovrden_in	Input	1 × Num. channels	Connects to RXLPMHFOVRDEN on transceiver channel primitives
rxlpmlfhold_in	Input	1 × Num. channels	Connects to RXLPMLFHOLD on transceiver channel primitives
rxlpmlfklovrden_in	Input	1 × Num. channels	Connects to RXLPMLFKLOVRDEN on transceiver channel primitives
rxlpmoshold_in	Input	1 × Num. channels	Connects to RXLPMOSHOLD on transceiver channel primitives
rxlpmosovrden_in	Input	1 × Num. channels	Connects to RXLPMOSOVRDEN on transceiver channel primitives
rxmcommaalignen_in	Input	1 × Num. channels	Connects to RXMCOMMAALIGNEN on transceiver channel primitives
rxmonitorsel_in	Input	2 × Num. channels	Connects to RXMONITORSEL on transceiver channel primitives
rxoobreset_in	Input	1 × Num. channels	Connects to RXOOBRESET on transceiver channel primitives
rxoscalreset_in	Input	1 × Num. channels	Connects to RXOSCALRESET on transceiver channel primitives
rxoshold_in	Input	1 × Num. channels	Connects to RXOSHOLD on transceiver channel primitives

Table 2-25: **Transceiver Channel Ports *(Cont'd)***

Name	Direction	Width	Description
rxosintcfg_in	Input	4 × Num. channels	Connects to RXOSINTCFG on transceiver channel primitives (GTHE3 and GTYE3 only)
rxosinten_in	Input	1 × Num. channels	Connects to RXOSINTEN on transceiver channel primitives (GTHE3 and GTYE3 only)
rxosinthold_in	Input	1 × Num. channels	Connects to RXOSINTHOLD on transceiver channel primitives (GTHE3 and GTYE3 only)
rxosintovrden_in	Input	1 × Num. channels	Connects to RXOSINTOVRDEN on transceiver channel primitives (GTHE3 and GTYE3 only)
rxosintstrobe_in	Input	1 × Num. channels	Connects to RXOSINTSTROBE on transceiver channel primitives (GTHE3 and GTYE3 only)
rxosinttestovrden_in	Input	1 × Num. channels	Connects to RXOSINTTESTOVRDEN on transceiver channel primitives (GTHE3 and GTYE3 only)
rxosovrden_in	Input	1 × Num. channels	Connects to RXOSOVRDEN on transceiver channel primitives
rxoutclksel_in	Input	3 × Num. channels	Connects to RXOUTCLKSEL on transceiver channel primitives
rxpcommaalignen_in	Input	1 × Num. channels	Connects to RXPCOMMAALIGNEN on transceiver channel primitives
rxpcsreset_in	Input	1 × Num. channels	Connects to RXPCSRESET on transceiver channel primitives
rxpd_in	Input	2 × Num. channels	Connects to RXPD on transceiver channel primitives
rxphalign_in	Input	1 × Num. channels	Connects to RXPHALIGN on transceiver channel primitives
rxphalignen_in	Input	1 × Num. channels	Connects to RXPHALIGNEN on transceiver channel primitives
rxphdlypd_in	Input	1 × Num. channels	Connects to RXPHDLYPD on transceiver channel primitives
rxphdlyreset_in	Input	1 × Num. channels	Connects to RXPHDLYRESET on transceiver channel primitives
rxphovrden_in	Input	1 × Num. channels	Connects to RXPHOVRDEN on transceiver channel primitives (GTE3, GTHE4, and GTYE4 only)
rxpllclksel_in	Input	2 × Num. channels	Connects to RXPLLCLKSEL on transceiver channel primitives. For more information on using this port, see Enabling CPLL Calibration block for UltraScale+ Devices.

Table 2-25: **Transceiver Channel Ports *(Cont'd)***

Name	Direction	Width	Description
rxpmareset_in	Input	1 × Num. channels	Connects to RXPMARESET on transceiver channel primitives
rxpolarity_in	Input	1 × Num. channels	Connects to RXPOLARITY on transceiver channel primitives
rxprbscntreset_in	Input	1 × Num. channels	Connects to RXPRBSCNTRESET on transceiver channel primitives
rxprbssel_in	Input	4 × Num. channels	Connects to RXPRBSSEL on transceiver channel primitives
rxprogdivreset_in	Input	1 × Num. channels	Connects to RXPROGDIVRESET on transceiver channel primitives
rxqpien_in	Input	1 × Num. channels	Connects to RXQPIEN on transceiver channel primitives (GTH only)
rxrate_in	Input	3 × Num. channels	Connects to RXRATE on transceiver channel primitives
rxratemode_in	Input	1 × Num. channels	Connects to RXRATEMODE on transceiver channel primitives
rxslide_in	Input	1 × Num. channels	Connects to RXSLIDE on transceiver channel primitives
rxslipoutclk_in	Input	1 × Num. channels	Connects to RXSLIPOUTCLK on transceiver channel primitives
rxslippma_in	Input	1 × Num. channels	Connects to RXSLIPPMA on transceiver channel primitives
rxsyncallin_in	Input	1 × Num. channels	Connects to RXSYNCALLIN on transceiver channel primitives
rxsyncin_in	Input	1 × Num. channels	Connects to RXSYNCIN on transceiver channel primitives
rxsyncmode_in	Input	1 × Num. channels	Connects to RXSYNCMODE on transceiver channel primitives
rxsysclksel_in	Input	2 × Num. channels	Connects to RXSYSCLKSEL on transceiver channel primitives
rxtermination_in	Input	1 × Num. channels	Connects to RXTERMINATION on transceiver channel primitives (GTHE4 and GTYE4 only)
rxuserrdy_in	Input	1 × Num. channels	Connects to RXUSERRDY on transceiver channel primitives
rxusrclk_in	Input	1 × Num. channels	Connects to RXUSRCLK on transceiver channel primitives
rxusrclk2_in	Input	1 × Num. channels	Connects to RXUSRCLK2 on transceiver channel primitives
sigvalidclk_in	Input	1 × Num. channels	Connects to SIGVALIDCLK on transceiver channel primitives

Table 2-25: **Transceiver Channel Ports *(Cont'd)***

Name	Direction	Width	Description
tstin_in	Input	20 × Num. channels	Connects to TSTIN on transceiver channel primitives
tx8b10bbypass_in	Input	8 × Num. channels	Connects to TX8B10BBYPASS on transceiver channel primitives
tx8b10ben_in	Input	1 × Num. channels	Connects to TX8B10BEN on transceiver channel primitives
txbufdiffctrl_in	Input	3 × Num. channels	Connects to TXBUFDIFFCTRL on transceiver channel primitives (GTHE3 and GTYE3 only)
txcominit_in	Input	1 × Num. channels	Connects to TXCOMINIT on transceiver channel primitives
txcomsas_in	Input	1 × Num. channels	Connects to TXCOMSAS on transceiver channel primitives
txcomwake_in	Input	1 × Num. channels	Connects to TXCOMWAKE on transceiver channel primitives
txctrl0_in	Input	16 × Num. channels	Connects to TXCTRL0 on transceiver channel primitives
txctrl1_in	Input	16 × Num. channels	Connects to TXCTRL1 on transceiver channel primitives
txctrl2_in	Input	8 × Num. channels	Connects to TXCTRL2 on transceiver channel primitives
txdata_in	Input	128 × Num. channels	Connects to TXDATA on transceiver channel primitives
txdataextendrsvd_in	Input	8 × Num. channels	Connects to TXDATAEXTENDRSVD on transceiver channel primitives
txdccforcestart_in	Input	1 × Num. channels	Connects to TXDCCFORCESTART on transceiver channel primitives (GTYE3, GTHE4, and GTYE4 only)
txdccreset_in	Input	1 × Num. channels	Connects to TXDCCRESET on transceiver channel primitives (GTYE3, GTHE4, and GTYE4 only)
txdeemph_in	Input	1 × Num. channels (GTHE3 and GTYE3) 2 × Num. channels (GTHE4 and GTYE4)	Connects to TXDEEMPH on transceiver channel primitives
txdetectrx_in	Input	1 × Num. channels	Connects to TXDETECTRX on transceiver channel primitives
txdiffctrl_in	Input	4 × Num. channels (GTHE3) 5 × Num. channels (GTYE3, GTHE4, and GTYE4)	Connects to TXDIFFCTRL on transceiver channel primitives

Table 2-25: **Transceiver Channel Ports *(Cont'd)***

Name	Direction	Width	Description
txdiffpd_in	Input	1 × Num. channels	Connects to TXDIFFPD on transceiver channel primitives (GTHE3 and GTYE3 only)
txdlybypass_in	Input	1 × Num. channels	Connects to TXDLYBYPASS on transceiver channel primitives
txdlyen_in	Input	1 × Num. channels	Connects to TXDLYEN on transceiver channel primitives
txdlyhold_in	Input	1 × Num. channels	Connects to TXDLYHOLD on transceiver channel primitives
txdlyovrden_in	Input	1 × Num. channels	Connects to TXDLYOVRDEN on transceiver channel primitives
txdlysreset_in	Input	1 × Num. channels	Connects to TXDLYSRESET on transceiver channel primitives
txdlyupdown_in	Input	1 × Num. channels	Connects to TXDLYUPDOWN on transceiver channel primitives
txelecidle_in	Input	1 × Num. channels	Connects to TXELECIDLE on transceiver channel primitives
txelforcestart_in	Input	1 × Num. channels	Connects to TXELFORCESTART on transceiver channel primitives (GTYE3 only)
txheader_in	Input	6 × Num. channels	Connects to TXHEADER on transceiver channel primitives
txinhibit_in	Input	1 × Num. channels	Connects to TXINHIBIT on transceiver channel primitives
txlatclk_in	Input	1 × Num. channels	Connects to TXLATCLK on transceiver channel primitives
txlfpstreset_in	Input	1 × Num. channels	Connects to TXLFPSTRESET on transceiver channel primitives (GTHE4 and GTYE4 only)
txlfpsu2lpexit_in	Input	1 × Num. channels	Connects to TXLFPSU2LPEXIT on transceiver channel primitives (GTHE4 and GTYE4 only)
txlfpsu3wake_in	Input	1 × Num. channels	Connects to TXLFPSU3WAKE on transceiver channel primitives (GTHE4 and GTYE4 only)
txmaincursor_in	Input	7 × Num. channels	Connects to TXMAINCURSOR on transceiver channel primitives
txmargin_in	Input	3 × Num. channels	Connects to TXMARGIN on transceiver channel primitives
txmuxdcdexhold_in	Input	1 × Num. channels	Connects to TXMUXDCDEXHOLD on transceiver channel primitives (GTHE4 and GTYE4 only)

Table 2-25: **Transceiver Channel Ports *(Cont'd)***

Name	Direction	Width	Description
txmuxdcdorwren_in	Input	1 × Num. channels	Connects to TXMUXDCDORWREN on transceiver channel primitives (GTHE4 and GTYE4 only)
txoneszeros_in	Input	1 × Num. channels	Connects to TXONESZEROS on transceiver channel primitives (GTHE4 and GTYE4 only)
txoutclksel_in	Input	3 × Num. channels	Connects to TXOUTCLKSEL on transceiver channel primitives
txpcsreset_in	Input	1 × Num. channels	Connects to TXPCSRESET on transceiver channel primitives
txpd_in	Input	2 × Num. channels	Connects to TXPD on transceiver channel primitives
txpdelecidlemode_in	Input	1 × Num. channels	Connects to TXPDELECIDLEMODE on transceiver channel primitives
txphalign_in	Input	1 × Num. channels	Connects to TXPHALIGN on transceiver channel primitives
txphalignen_in	Input	1 × Num. channels	Connects to TXPHALIGNEN on transceiver channel primitives
txphdlypd_in	Input	1 × Num. channels	Connects to TXPHDLYPD on transceiver channel primitives
txphdlyreset_in	Input	1 × Num. channels	Connects to TXPHDLYRESET on transceiver channel primitives
txphdlytstclk_in	Input	1 × Num. channels	Connects to TXPHDLYTSTCLK on transceiver channel primitives
txphinit_in	Input	1 × Num. channels	Connects to TXPHINIT on transceiver channel primitives
txphovrden_in	Input	1 × Num. channels	Connects to TXPHOVRDEN on transceiver channel primitives
txpippmen_in	Input	1 × Num. channels	Connects to TXPIPPMEN on transceiver channel primitives
txpippmovrden_in	Input	1 × Num. channels	Connects to TXPIPPMOVRDEN on transceiver channel primitives
txpippmpd_in	Input	1 × Num. channels	Connects to TXPIPPMPD on transceiver channel primitives
txpippmsel_in	Input	1 × Num. channels	Connects to TXPIPPMSEL on transceiver channel primitives
txpippmstepsize_in	Input	5 × Num. channels	Connects to TXPIPPMSTEPSIZE on transceiver channel primitives
txpisopd_in	Input	1 × Num. channels	Connects to TXPISOPD on transceiver channel primitives

Table 2-25: **Transceiver Channel Ports *(Cont'd)***

Name	Direction	Width	Description
txpllclksel_in	Input	2 × Num. channels	Connects to TXPLLCLKSEL on transceiver channel primitives and for more information on using this port, see Enabling CPLL Calibration block for UltraScale+ Devices.
txpmareset_in	Input	1 × Num. channels	Connects to TXPMARESET on transceiver channel primitives
txpolarity_in	Input	1 × Num. channels	Connects to TXPOLARITY on transceiver channel primitives
txpostcursor_in	Input	5 × Num. channels	Connects to TXPOSTCURSOR on transceiver channel primitives
txpostcursorinv_in	Input	1 × Num. channels	Connects to TXPOSTCURSORINV on transceiver channel primitives (GTHE3 only)
txprbsforceerr_in	Input	1 × Num. channels	Connects to TXPRBSFORCEERR on transceiver channel primitives
txprbssel_in	Input	4 × Num. channels	Connects to TXPRBSSEL on transceiver channel primitives
txprecursor_in	Input	5 × Num. channels	Connects to TXPRECURSOR on transceiver channel primitives
txprecursorinv_in	Input	1 × Num. channels	Connects to TXPRECURSORINV on transceiver channel primitives (GTHE3 only)
txprogdivreset_in	Input	1 × Num. channels	Connects to TXPROGDIVRESET on transceiver channel primitives
txqpibiasen_in	Input	1 × Num. channels	Connects to TXQPIBIASEN on transceiver channel primitives (GTH only)
txqpistrongpdown_in	Input	1 × Num. channels	Connects to TXQPISTRONGPDOWN on transceiver channel primitives (GTHE3 only)
txqpiweakpup_in	Input	1 × Num. channels	Connects to TXQPIWEAKPUP on transceiver channel primitives (GTH only)
txrate_in	Input	3 × Num. channels	Connects to TXRATE on transceiver channel primitives
txratemode_in	Input	1 × Num. channels	Connects to TXRATEMODE on transceiver channel primitives
txsequence_in	Input	7 × Num. channels	Connects to TXSEQUENCE on transceiver channel primitives
txswing_in	Input	1 × Num. channels	Connects to TXSWING on transceiver channel primitives
txsyncallin_in	Input	1 × Num. channels	Connects to TXSYNCALLIN on transceiver channel primitives

Table 2-25: **Transceiver Channel Ports *(Cont'd)***

Name	Direction	Width	Description
txsyncin_in	Input	1 × Num. channels	Connects to TXSYNCIN on transceiver channel primitives
txsyncmode_in	Input	1 × Num. channels	Connects to TXSYNCMODE on transceiver channel primitives
txsysclksel_in	Input	2 × Num. channels	Connects to TXSYSCLKSEL on transceiver channel primitives
txuserrdy_in	Input	1 × Num. channels	Connects to TXUSERRDY on transceiver channel primitives
txusrclk_in	Input	1 × Num. channels	Connects to TXUSRCLK on transceiver channel primitives
txusrclk2_in	Input	1 × Num. channels	Connects to TXUSRCLK2 on transceiver channel primitives
bufgtce_out	Output	3 × Num. channels (GTHE3 and GTYE3) 1 × Num. channels (GTHE4 and GTYE4)	Connects to BUFGTCE on transceiver channel primitives
bufgtcemask_out	Output	3 × Num. channels	Connects to BUFGTCEMASK on transceiver channel primitives
bufgtdiv_out	Output	9 × Num. channels	Connects to BUFGTDIV on transceiver channel primitives
bufgtreset_out	Output	3 × Num. channels (GTHE3 and GTYE3) 1 × Num. channels (GTHE4 and GTYE4)	Connects to BUFGTRESET on transceiver channel primitives
bufgtrstmask_out	Output	3 × Num. channels	Connects to BUFGTRSTMASK on transceiver channel primitives
cpllfbclklost_out	Output	1 × Num. channels	Connects to CPLLFBCLKLOST on transceiver channel primitives
cplllock_out	Output	1 × Num. channels	Connects to CPLLLOCK on transceiver channel primitives
cpllrefclklost_out	Output	1 × Num. channels	Connects to CPLLREFCLKLOST on transceiver channel primitives
dmonitorout_out	Output	17 × Num. channels (GTHE3 and GTYE3) 16 × Num. channels (GTHE4 and GTYE4)	Connects to DMONITOROUT on transceiver channel primitives
dmonitoroutclk_out	Output	1 × Num. channels	Connects to DMONITOROUTCLK on transceiver channel primitives (GTHE4 and GTYE4 only)
drpdo_out	Output	16 × Num. channels	Connects to DRPDO on transceiver channel primitives
drprdy_out	Output	1 × Num. channels	Connects to DRPRDY on transceiver channel primitives

Table 2-25: **Transceiver Channel Ports *(Cont'd)***

Name	Direction	Width	Description
eyescandataerror_out	Output	1 × Num. channels	Connects to EYESCANDATAERROR on transceiver channel primitives
gthtxn_out	Output	1 × Num. channels	Connects to GTHTXN on transceiver channel primitives (GTH only)
gthtxp_out	Output	1 × Num. channels	Connects to GTHTXP on transceiver channel primitives (GTH only)
gtpowergood_out	Output	1 × Num. channels	Connects to GTPOWERGOOD on transceiver channel primitives
gtrefclkmonitor_out	Output	1 × Num. channels	Connects to GTREFCLKMONITOR on transceiver channel primitives
gtytxn_out	Output	1 × Num. channels	Connects to GTYTXN on transceiver channel primitives (GTY only)
gtytxp_out	Output	1 × Num. channels	Connects to GTYTXP on transceiver channel primitives (GTY only)
pcierategen3_out	Output	1 × Num. channels	Connects to PCIERATEGEN3 on transceiver channel primitives
pcierateidle_out	Output	1 × Num. channels	Connects to PCIERATEIDLE on transceiver channel primitives
pcierateqpllpd_out	Output	2 × Num. channels	Connects to PCIERATEQPLLPD on transceiver channel primitives
pcierateqpllreset_out	Output	2 × Num. channels	Connects to PCIERATEQPLLRESET on transceiver channel primitives
pciesynctxsyncdone_out	Output	1 × Num. channels	Connects to PCIESYNCTXSYNCDONE on transceiver channel primitives
pcieusergen3rdy_out	Output	1 × Num. channels	Connects to PCIEUSERGEN3RDY on transceiver channel primitives
pcieuserphystatusrst_out	Output	1 × Num. channels	Connects to PCIEUSERPHYSTATUSRST on transceiver channel primitives
pcieuserratestart_out	Output	1 × Num. channels	Connects to PCIEUSERRATESTART on transceiver channel primitives
pcsrsvdout_out	Output	12 × Num. channels (GTHE3) 16 × Num. channels (GTYE3, GTHE4, and GTYE4))	Connects to PCSRSVDOUT on transceiver channel primitives
phystatus_out	Output	1 × Num. channels	Connects to PHYSTATUS on transceiver channel primitives
pinrsrvdas_out	Output	8 × Num. channels (GTHE3 and GTYE3) 16 × Num. channels (GTHE4 and GTYE4)	Connects to PINRSRVDAS on transceiver channel primitives

Table 2-25: **Transceiver Channel Ports *(Cont'd)***

Name	Direction	Width	Description
powerpresent_out	Output	1 × Num. channels	Connects to POWERPRESENT on transceiver channel primitives (GTHE4 and GTYE4 only)
resetexception_out	Output	1 × Num. channels	Connects to RESETEXCEPTION on transceiver channel primitives
rxbufstatus_out	Output	3 × Num. channels	Connects to RXBUFSTATUS on transceiver channel primitives
rxbyteisaligned_out	Output	1 × Num. channels	Connects to RXBYTEISALIGNED on transceiver channel primitives
rxbyterealign_out	Output	1 × Num. channels	Connects to RXBYTEREALIGN on transceiver channel primitives
rxcdrlock_out	Output	1 × Num. channels	Connects to RXCDRLOCK on transceiver channel primitives
rxcdrphdone_out	Output	1 × Num. channels	Connects to RXCDRPHDONE on transceiver channel primitives
rxchanbondseq_out	Output	1 × Num. channels	Connects to RXCHANBONDSEQ on transceiver channel primitives
rxchanisaligned_out	Output	1 × Num. channels	Connects to RXCHANISALIGNED on transceiver channel primitives
rxchanrealign_out	Output	1 × Num. channels	Connects to RXCHANREALIGN on transceiver channel primitives
rxchbondo_out	Output	5 × Num. channels	Connects to RXCHBONDO on transceiver channel primitives
rxckcaldone_out	Output	1 × Num. channels	Connects to RXCKCALDONE on transceiver channel primitives (GTYE3, GTHE4, and GTYE4 only)
rxclkcorcnt_out	Output	2 × Num. channels	Connects to RXCLKCORCNT on transceiver channel primitives
rxcominitdet_out	Output	1 × Num. channels	Connects to RXCOMINITDET on transceiver channel primitives
rxcommadet_out	Output	1 × Num. channels	Connects to RXCOMMADET on transceiver channel primitives
rxcomsasdet_out	Output	1 × Num. channels	Connects to RXCOMSASDET on transceiver channel primitives
rxcomwakedet_out	Output	1 × Num. channels	Connects to RXCOMWAKEDET on transceiver channel primitives
rxctrl0_out	Output	16 × Num. channels	Connects to RXCTRL0 on transceiver channel primitives
rxctrl1_out	Output	16 × Num. channels	Connects to RXCTRL1 on transceiver channel primitives
rxctrl2_out	Output	8 × Num. channels	Connects to RXCTRL2 on transceiver channel primitives

Table 2-25: **Transceiver Channel Ports *(Cont'd)***

Name	Direction	Width	Description
rxctrl3_out	Output	8 × Num. channels	Connects to RXCTRL3 on transceiver channel primitives
rxdata_out	Output	128 × Num. channels	Connects to RXDATA on transceiver channel primitives
rxdataextendrsvd_out	Output	8 × Num. channels	Connects to RXDATAEXTENDRSVD on transceiver channel primitives
rxdatavalid_out	Output	2 × Num. channels	Connects to RXDATAVALID on transceiver channel primitives
rxdlysresetdone_out	Output	1 × Num. channels	Connects to RXDLYSRESETDONE on transceiver channel primitives
rxelecidle_out	Output	1 × Num. channels	Connects to RXELECIDLE on transceiver channel primitives
rxheader_out	Output	6 × Num. channels	Connects to RXHEADER on transceiver channel primitives
rxheadervalid_out	Output	2 × Num. channels	Connects to RXHEADERVALID on transceiver channel primitives
rxlfpstresetdet_out	Output	1 × Num. channels	Connects to RXLFPSTRESETDET on transceiver channel primitives (GTHE4 and GTYE4 only)
rxlfpsu2lpexitdet_out	Output	1 × Num. channels	Connects to RXLFPSU2LPEXITDET on transceiver channel primitives (GTHE4 and GTYE4 only)
rxlfpsu3wakedet_out	Output	1 × Num. channels	Connects to RXLFPSU3WAKEDET on transceiver channel primitives (GTHE4 and GTYE4 only)
rxmonitorout_out	Output	7 × Num. channels (GTHE3 and GTYE3 only) 8 × Num. channels (GTHE4 and GTYE4 only)	Connects to RXMONITOROUT on transceiver channel primitives
rxosintdone_out	Output	1 × Num. channels	Connects to RXOSINTDONE on transceiver channel primitives
rxosintstarted_out	Output	1 × Num. channels	Connects to RXOSINTSTARTED on transceiver channel primitives
rxosintstrobedone_out	Output	1 × Num. channels	Connects to RXOSINTSTROBEDONE on transceiver channel primitives
rxosintstrobestarted_out	Output	1 × Num. channels	Connects to RXOSINTSTROBESTARTED on transceiver channel primitives
rxoutclk_out	Output	1 × Num. channels	Connects to RXOUTCLK on transceiver channel primitives
rxoutclkfabric_out	Output	1 × Num. channels	Connects to RXOUTCLKFABRIC on transceiver channel primitives

Table 2-25: **Transceiver Channel Ports *(Cont'd)***

Name	Direction	Width	Description
rxoutclkpcs_out	Output	1 × Num. channels	Connects to RXOUTCLKPCS on transceiver channel primitives
rxphaligndone_out	Output	1 × Num. channels	Connects to RXPHALIGNDONE on transceiver channel primitives
rxphalignerr_out	Output	1 × Num. channels	Connects to RXPHALIGNERR on transceiver channel primitives
rxpmaresetdone_out	Output	1 × Num. channels	Connects to RXPMARESETDONE on transceiver channel primitives
rxprbserr_out	Output	1 × Num. channels	Connects to RXPRBSERR on transceiver channel primitives
rxprbslocked_out	Output	1 × Num. channels	Connects to RXPRBSLOCKED on transceiver channel primitives
rxprgdivresetdone_out	Output	1 × Num. channels	Connects to RXPRGDIVRESETDONE on transceiver channel primitives
rxqpisenn_out	Output	1 × Num. channels	Connects to RXQPISENN on transceiver channel primitives (GTH only)
rxqpisenp_out	Output	1 × Num. channels	Connects to RXQPISENP on transceiver channel primitives (GTH only)
rxratedone_out	Output	1 × Num. channels	Connects to RXRATEDONE on transceiver channel primitives
rxrecclkout_out	Output	1 × Num. channels	Connects to RXRECCLKOUT on transceiver channel primitives
rxresetdone_out	Output	1 × Num. channels	Connects to RXRESETDONE on transceiver channel primitives
rxsliderdy_out	Output	1 × Num. channels	Connects to RXSLIDERDY on transceiver channel primitives
rxslipdone_out	Output	1 × Num. channels	Connects to RXSLIPDONE on transceiver channel primitives
rxslipoutclkrdy_out	Output	1 × Num. channels	Connects to RXSLIPOUTCLKRDY on transceiver channel primitives
rxslippmardy_out	Output	1 × Num. channels	Connects to RXSLIPPMARDY on transceiver channel primitives
rxstartofseq_out	Output	2 × Num. channels	Connects to RXSTARTOFSEQ on transceiver channel primitives
rxstatus_out	Output	3 × Num. channels	Connects to RXSTATUS on transceiver channel primitives
rxsyncdone_out	Output	1 × Num. channels	Connects to RXSYNCDONE on transceiver channel primitives
rxsyncout_out	Output	1 × Num. channels	Connects to RXSYNCOUT on transceiver channel primitives

Table 2-25: **Transceiver Channel Ports *(Cont'd)***

Name	Direction	Width	Description
rxvalid_out	Output	1 × Num. channels	Connects to RXVALID on transceiver channel primitives
txbufstatus_out	Output	2 × Num. channels	Connects to TXBUFSTATUS on transceiver channel primitives
txcomfinish_out	Output	1 × Num. channels	Connects to TXCOMFINISH on transceiver channel primitives
txdccdone_out	Output	1 × Num. channels	Connects to TXDCCDONE on transceiver channel primitives (GTYE3, GTHE4, and GTYE4 only)
txdlysresetdone_out	Output	1 × Num. channels	Connects to TXDLYSRESETDONE on transceiver channel primitives
txoutclk_out	Output	1 × Num. channels	Connects to TXOUTCLK on transceiver channel primitives
txoutclkfabric_out	Output	1 × Num. channels	Connects to TXOUTCLKFABRIC on transceiver channel primitives
txoutclkpcs_out	Output	1 × Num. channels	Connects to TXOUTCLKPCS on transceiver channel primitives
txphaligndone_out	Output	1 × Num. channels	Connects to TXPHALIGNDONE on transceiver channel primitives
txphinitdone_out	Output	1 × Num. channels	Connects to TXPHINITDONE on transceiver channel primitives
txpmaresetdone_out	Output	1 × Num. channels	Connects to TXPMARESETDONE on transceiver channel primitives
txprgdivresetdone_out	Output	1 × Num. channels	Connects to TXPRGDIVRESETDONE on transceiver channel primitives
txqpisenn_out	Output	1 × Num. channels	Connects to TXQPISENN on transceiver channel primitives (GTH only)
txqpisenp_out	Output	1 × Num. channels	Connects to TXQPISENP on transceiver channel primitives (GTH only)
txratedone_out	Output	1 × Num. channels	Connects to TXRATEDONE on transceiver channel primitives
txresetdone_out	Output	1 × Num. channels	Connects to TXRESETDONE on transceiver channel primitives
txsyncdone_out	Output	1 × Num. channels	Connects to TXSYNCDONE on transceiver channel primitives
txsyncout_out	Output	1 × Num. channels	Connects to TXSYNCOUT on transceiver channel primitives

Designing with the Core

This chapter includes guidelines and additional information to facilitate designing with the UltraScale™ FPGAs Transceivers Wizard IP core.

General Design Guidelines

The design guidelines for the wizard core largely reflect those of the serial transceivers instantiated by the Wizard. It is important to understand the general usage and specific procedures that are required for correct operation of serial transceivers in your system. See the *UltraScale Architecture GTH Transceivers User Guide* (UG576) [Ref 1] or *UltraScale Architecture GTY Transceivers User Guide* (UG578) [Ref 2] for details.

The UltraScale™ FPGAs Transceivers Wizard provides a highly flexible Vivado® Integrated Design Environment (IDE)-driven customization flow, which in addition to basic customization of transceiver use modes, also includes a physical resource site selection interface, an optional port enablement interface, and helper block location choices. The result is a core instance that addresses the specific needs of your application. As such, Wizard IP core instances do not require manual modification and should not be edited. Xilinx cannot guarantee timing, functionality, or support if modifications are made to any output products of the generated core.

Designing with the Helper Blocks

The helper block modules provided with the Wizard simplify common or complex transceiver usage. Design and usage guidelines of these helper blocks are presented in the following sections.

Consider the benefits and drawbacks of each choice when deciding whether to locate each helper block within the core or in the example design. The primary benefits of locating a helper block within the core are a simpler, more abstracted interface, and that as part of the core, the helper block is also updated if you upgrade the core to a new version. However, the helper block is not accessible for manual modification if different behavior is required for your use case.

The primary benefit of locating a helper block within the example design is that you gain the ability to use it as an example starting point, should connectivity or contents require modification to suit your specific needs. However, because it is not part of the core, the

example design must be regenerated and any manual edits must be performed again if you upgrade the core to a new version. Xilinx cannot guarantee support for modifications made to the example design contents as they are delivered.

Designing with the Example Design

An example design can be generated for any instance of the Wizard IP core. The example design instantiates the core instance, any helper blocks that you have chosen to locate in the example design, and the requisite reference clock and recovered clock buffers. It also provides various convenience functions such as per-channel vector slicing. The contents of the example design are customized to support the specific core customization. Use of the example design as a demonstration and as a starting point for integration into your system is suggested.

Reset Controller Helper Block

The reset controller helper block simplifies the process of resetting and initializing the serial transceiver primitives. To operate, the helper block must be provided the free-running clock `gtwiz_reset_clk_freerun_in` that is toggling at the frequency specified during IP customization, prior to device configuration.

A single instance of the helper block is delivered with each instance of the Wizard IP core. Its user interface provides you with a simple means of initiating and monitoring the completion of transceiver reset procedures. Its transceiver interface connects to each transceiver primitive resource within the core instance.

The helper block contains three finite state machines:

- **Transmitter reset state machine**: Resets the transmitter PLL and/or the transmitter datapath of all transceiver primitives, and indicates their completion
- **Receiver reset state machine**: Resets the receiver PLL and/or the receiver datapath of all transceiver primitives, and indicates their completion
- **"Reset all" state machine**: Controls the transmitter and receiver reset state machines and sequences them appropriately to reset all of the necessary transceiver primitives without redundant operations

The transmitter and receiver reset state machines are independent of one another, and each can be initiated either directly through the user interface if needed, or they can be controlled by the "reset all" state machine if you initiate a reset all command. The reset all state machine is provided as a convenience and is useful for initial bring-up. However, it is not necessary to use if only independent transmitter and reset sequences are desired.

Reset State Machines

The transmitter and receiver reset state machines each have two entry points: one which causes the associated PLL(s) to be reset, followed by a reset of the datapath, and a second in which only the datapath is reset. Figure 3-1 illustrates the three reset controller helper block finite state machines and the reset sequences they control.

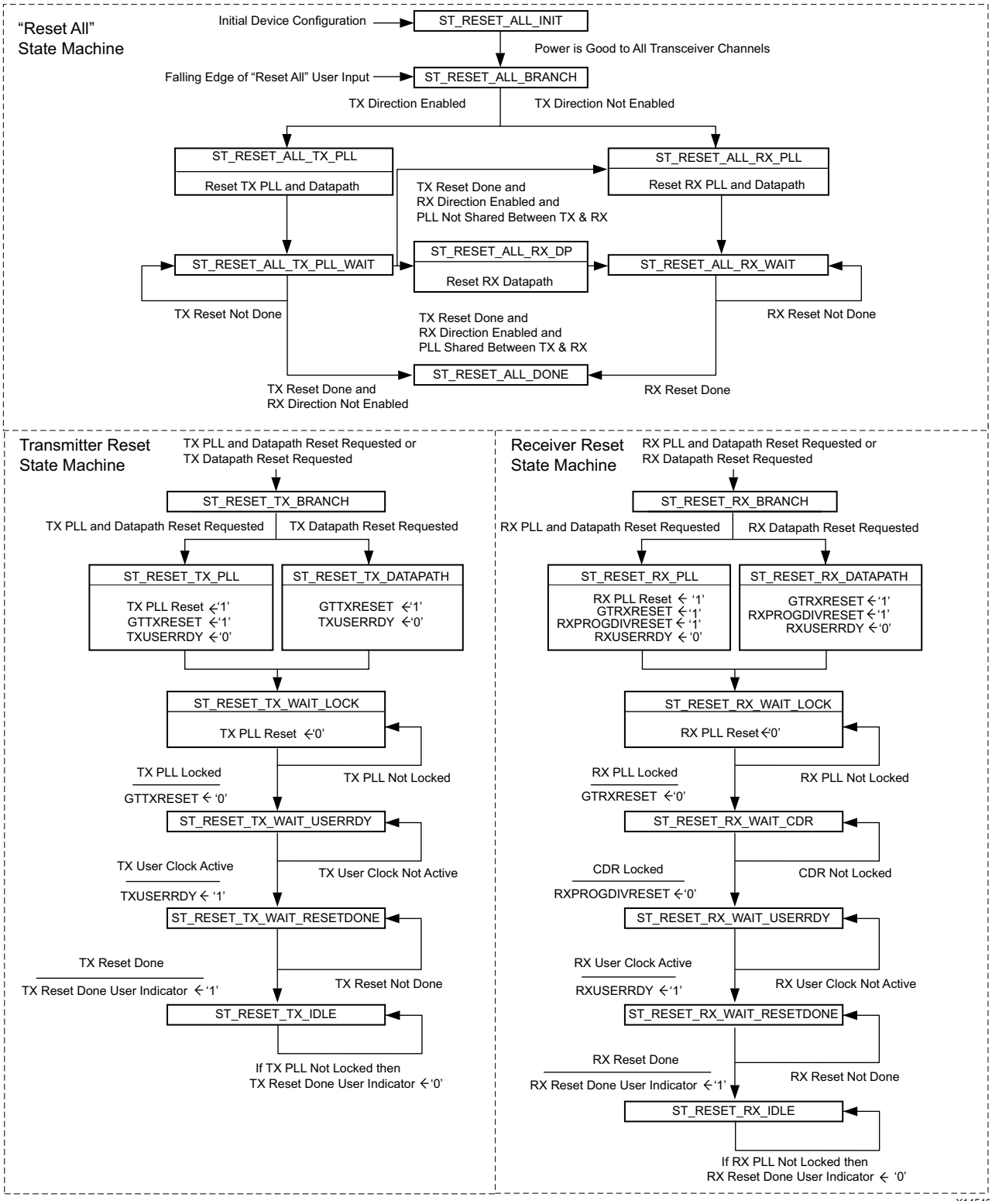

Figure 3-1: **Reset Controller Helper Block Finite State Machines**

The transmitter reset state machine initiates a PLL reset followed by a transmitter datapath reset when the `gtwiz_reset_tx_pll_and_datapath_in` input is pulsed. All PLLs (either QPLL or CPLL type) instantiated by the core instance that are used to clock the transmitter datapath are reset in response to this input. After all these PLLs lock, the transmitter programmable dividers and datapaths of all transceiver primitives are reset. If a PLL reset is not needed, a transmitter datapath-only reset is initiated when the `gtwiz_reset_tx_datapath_in` input is pulsed. Regardless of the reset entry point, the `gtwiz_reset_tx_done_out` indicator is asserted synchronous to transmitter master channel `TXUSRCLK2` upon completion of the transmitter reset sequence for all transceiver primitives.

Likewise, the receiver reset state machine initiates a PLL reset followed by a receiver datapath reset when the `gtwiz_reset_rx_pll_and_datapath_in` input is pulsed. All PLLs (either QPLL or CPLL type) instantiated by the core instance that are used to clock the receiver datapath are reset in response to this input. When all these PLLs lock, the receiver datapaths of all transceiver primitives are reset. If a PLL reset is not needed, a receiver datapath-only reset is initiated when the `gtwiz_reset_rx_datapath_in` input is pulsed. Regardless of the reset entry point, the `gtwiz_reset_rx_done_out` indicator is asserted synchronous to receiver master channel `RXUSRCLK2` upon completion of the receiver reset sequence for all transceiver primitives.

IMPORTANT: *The independent transmitter and receiver reset state machines are simple and useful. However, because PLLs can be shared between transmitter and receiver datapaths, it is important to understand the potential system impacts when using the* `gtwiz_reset_tx_pll_and_datapath_in` *and* `gtwiz_reset_rx_pll_and_datapath_in` *inputs. For example, if both transmitter and receiver datapaths are clocked by QPLL0 resources, assertion of either of those two inputs would reset the shared QPLL0 of each transceiver Quad, causing potentially-unintended link loss in the other data direction. Use these inputs with caution, especially if PLL resources are shared with other core instances.*

The reset all state machine can be used to avoid just such redundant PLL reset sequences. In addition, it resets the transmitter data direction before the receiver data direction (which can improve data integrity in loopback or some other circumstances) and is triggered by a simple one-input interface. The reset all state machine does not sequence transceiver primitive reset signals itself. Rather, it controls the transmitter and receiver reset state machines in the appropriate fashion for your core customization—effectively controlling some sequence of `gtwiz_reset_tx_pll_and_datapath_in`, `gtwiz_reset_rx_pll_and_datapath_in`, and `gtwiz_reset_rx_datapath_in` assertions. See Figure 3-1 to visualize the specific effects of the reset all state machine for your core customization, noting that the reset all state machine is initialized by the falling edge of the synchronized `gtwiz_reset_all_in` input.

Special GTH Transceiver CPLL Reset Requirements for Engineering Sample (ES1 or ES2) UltraScale Devices

In GTH transceiver configurations targeting engineering sample (ES1 or ES2) UltraScale devices where the CPLL is used as either the transmitter PLL type, receiver PLL type, or as the source of a selectable TXOUTCLK frequency, a special CPLL calibration procedure runs as part of the PLL reset sequence. GTH transceiver core configurations that target engineering sample (ES1 or ES2) UltraScale devices and that use the CPLL have these additional reset requirements and characteristics:

- Do not attempt to perform transceiver channel DRP transactions in the time period between initializing a CPLL reset and assertion of the CPLL lock indicator, or in the time period between releasing the CPLL from power-down mode and initializing a CPLL reset. During these times, transceiver channel DRP transactions are ignored.
- Do not attempt to assert `txprogdivreset_in` or change the value of the `txoutclksel_in` port in the time period between initializing a CPLL reset and assertion of the CPLL lock indicator, or in the time period between releasing the CPLL from power-down mode and initializing a CPLL reset. During this time, inputs on these ports are ignored.
- Each bit of the `drpclk_in` port must be driven by the free-running clock, operating at exactly the frequency specified during IP customization. See Performance, page 10, for more details.
- If the transmitter user clocking network helper block is located in the example design, then the `gtwiz_userclk_tx_reset_in` port on the helper block and the `gtwiz_userclk_tx_reset_in` port on the core must be driven by the same source. See Transmitter User Clocking Network Helper Block Ports, page 19, for more details.
- The time required to achieve CPLL lock in hardware operation can vary between CPLL resets.
- Resetting the CPLL temporarily disrupts the `TXOUTCLK` signal, and therefore also the clocks produced by the transmitter user clocking network helper block, even when the CPLL is used exclusively for the receiver data path. This is because the CPLL calibration procedure takes control of `TXOUTCLK` during CPLL reset, irrespective of which resources the CPLL drives. If runtime disruption to transmitter user clocks are not tolerable in configurations where the CPLL drives only receiver resources, take care to reset and achieve lock on the CPLL prior to, or separate from bringing up transmitter resources.

Initialization of a CPLL reset can include pulses of `gtwiz_reset_all_in`, `gtwiz_reset_tx_pll_and_datapath_in` (when the CPLL is used for the transmitter datapath), `gtwiz_reset_rx_pll_and_datapath_in` (when the CPLL is used for the receiver datapath), or `cpllreset_in` (when the reset controller helper block is not used).

CPLL lock indicators can include `gtwiz_reset_tx_done_out` (when the CPLL is used for the transmitter data path), `gtwiz_reset_rx_done_out` (when the CPLL is used for the receiver datapath), or `cplllock_out` (when the reset controller helper block is not used, or if a direct CPLL lock indicator is desired).

Enabling CPLL Calibration block for UltraScale+ Devices

In UltraScale+ GTH/GTY transceiver CPLL may not be able to reliably lock in the following circumstances:

- After configuration
- Removing/re-applying reference clock
- Asserting/de-asserting CPLLPD

In the failure state, the CPLL may freeze at an invalid output frequency and the CPLLLOCK signal may incorrectly be set high. The solution for this issue is to include the CPLL Calibration block. The following user parameters are added to the UltraScale GT Wizard IP without the GUI presence for this purpose:

- `INCLUDE_CPLL_CAL`:
  - Default Value => 2: For CPLL configured IPs, the CPLL Calibration block will be enabled internally by default for GTHE4 and GTYE4.
  - setting 1 => Includes the CPLL Calibration block,
  - setting 0 => Excludes the CPLL Calibration block.
- `SIM_CPLL_CAL_BYPASS`:
  - Default Value => 1
  - setting 0 => Continues to use the large counters in CPLL Calibration block and mimics the hardware behavior
  - setting 1 => Bypasses some of the counters in CPLL Calibration block through usage of synthesis translate on/off pragmas and improve functional simulation time.

To enable the CPLL Calibration block for GTYE4/GTHE4 UltraScale+ devices, set the value of `INCLUDE_CPLL_CAL` to 1 while generating the IP through Tcl customization. This additional block could result in higher simulation times. This is because CPLL calibration block evaluates the frequency at which the CPLL lock is achieved. To bypass this block to reduce functional simulation time, set the `SIM_CPLL_CAL_BYPASS` user parameter to 1 during IP customization.

Note: This will not have any effect on post-synthesis simulations and hardware functionality.

The CPLL Calibration block is now enhanced to work with both TX alone or RX alone for advanced use cases. When Both TX alone or TX + RX are using CPLL, then only the TX CPLL

Calibration block would be applicable, while in the use case where TX is not CPLL and RX is CPLL (txpllclksel_in != 0 and rxpllclksel_in ==0), then RX CPLL Calibration block is used. The choice of TX/RX CPLL calibration block to be used is done based on the user configuration entered in IP customization GUI. The below CPLL calibration block ports are exposed only when the INCLUDE_CPLL_CAL user parameter is set to 1, the default values corresponding to the line-rate are obtained by open-example design step or by following the equation specified in the description field. If you set the value of INCLUDE_CPLL_CAL to 2, the HDL logic inside the wizard drives the relevant ports internally for configuration during IP customization. Xilinx recommends that the INCLUDE_CPLL_CAL parameter be set to 1 and appropriate values be driven on the ports shown in Table 3-1 when the GT parent IP intends to perform dynamic line rate switching. For more information and guidance on rate usage of these ports, see Xilinx Answer: AR# 70485.

Table 3-1: **CPLL Calibration Block Additional Ports**

Name	Direction	Description
USER_TXOUTCLK_BUFG_CE_IN gtwiz_gthe4_cpll_cal_bufg_ce_in gtwiz_gtye4_cpll_cal_bufg_ce_in	Input	CE for BUFG_GT for TX clocking
TXOUTCLK_PERIOD_IN[17:0] gtwiz_gthe4_cpll_cal_txoutclk_period_in gtwiz_gtye4_cpll_cal_txoutclk_period_in	Input	Calculated as: $\frac{CPLL\_VCO\_RATE}{20} \times \frac{FREQ\_COUNT\_WINDOW}{4 * FREQ\_CLKIN}$ Set the value of FREQ_COUNT_WINDOW to 16,000. FREQ_CLKIN is the frequency of gtwiz_reset_clk_freerun_in, referred to as the FREERUN_FREQUENCY user parameter, on the Physical Resources tab of the UltraScale GT Wizard as shown in Figure 4-2. This frequency is referred to as the free-running and DRP clock frequency. CPLL_VCO_RATE is the fPLLClkout, which is the PLL output frequency in MHz as described in the *UltraScale Architecture GTH Transceivers User Guide* (UG576) [Ref 1] and *UltraScale Architecture GTY Transceivers User Guide* (UG578) [Ref 2]. For CPLL based rate switching requirements, the easiest way to get the expected values is to generate example designs for each of the target line rate combinations, as the GT Wizard IP would calculate the target values based on the project parameters and the data-sheet ranges.
CNT_TOL_IN[17:0] gtwiz_gthe4_cpll_cal_cnt_tol_in gtwiz_gtye4_cpll_cal_cnt_tol_in	Input	Set to ROUND(0.01*TXOUTCLK_PERIOD_IN)

Reset Sequencing and Other Services

The transmitter and receiver reset state machines implement the relevant reset sequences as specified in the *UltraScale Architecture GTH Transceivers User Guide* (UG576) [Ref 1] or *UltraScale Architecture GTY Transceivers User Guide* (UG578) [Ref 2]. The reset controller helper block transceiver interface connects to the transceiver primitives. To implement proper `TXUSERRDY` and `RXUSERRDY` signaling, wiring exists between the reset controller helper block and both transmitter and receiver user clocking network helper blocks. Also, if the transmitter or receiver buffers are bypassed, wiring exists between the reset helper block and the relevant buffer bypass controller helper blocks to initiate the buffer bypass sequences upon completion of the reset sequences. This wiring exists regardless of the location of each helper block.

Following device configuration, no reset helper block reset inputs should be asserted until transceiver power is reported as good. The reset controller helper block internally holds all PLL and datapath resources in reset until `GTPOWERGOOD` is High from all transceiver channels, then subsequently resets all transceiver resources by transitioning once through the reset all state machine. As a result, you should wait for either the initial assertion of all bits of the `gtpowergood_out` port (if you have exposed the port), or of both `gtwiz_reset_tx_done_out` and `gtwiz_reset_rx_done_out` before attempting subsequent resets of any kind. However with UltraScale+ devices, it has been observed that if the JTAG frequency with which the FPGA is programmed is greater than 6MHz,then there could be some initial instability on the GT Reference clock output from `IBUFDS_GTE4` followed by `BUFG_GT`. To avoid this, a user delay powergood logic is added by default inside the GT Wizard IP to hold the `gtpowergood_out` so that the logic is kept in reset initially. `USER_GTPOWERGOOD_DELAY_EN` user parameter has been added for enabling this. It is recommended that the value of this parameter should always be 1, regardless of the actual JTAG frequency used to program.

When the transmitter reset state machine is complete, if one or more PLLs that clock transceiver primitive transmitter datapaths lose lock, the `gtwiz_reset_tx_done_out` user indicator is de-asserted. A reset sequence does not automatically restart in this circumstance; user intervention is required.

Similarly, when the receiver reset state machine has completed, if one or more PLLs that clock transceiver primitive receiver datapaths lose lock, the `gtwiz_reset_rx_done_out` user indicator is de-asserted. A reset sequence does not automatically restart in this circumstance; user intervention is required.

The helper block can be located either within the core or in the example design per user selection. Depending on its location and the location of other helper blocks, the relevant ports are enabled on the core interface so that the necessary signals can cross the core boundary.

If additional reset control or related ports are required for your application, or if you wish to observe individual transceiver primitive reset status signals, you can enable the relevant ports on the core instance using the optional ports interface during IP customization.

See Chapter 2, Product Specification, for a description of all reset controller block ports. See the *UltraScale Architecture GTH Transceivers User Guide* (UG576) [Ref 1] or *UltraScale Architecture GTY Transceivers User Guide* (UG578) [Ref 2] for complete documentation on resetting and initializing the transceiver primitives.

Transmitter User Clocking Network Helper Block

The transmitter user clocking network helper block is a simple module used to derive and buffer the appropriate clocks to drive the TXUSRCLK and TXUSRCLK2 inputs of one or more transceiver channel primitives.

A single instance of the helper block is delivered with each instance of the Wizard IP core. By default, its source clock input port, gtwiz_userclk_tx_srcclk_in, is driven by the TXOUTCLK port of the master transceiver channel. Within the helper block, this source drives either one or two BUFG_GT primitives, which are global clock buffers that are capable of clock division.

As shown in Figure 3-2, if the TXUSRCLK and TXUSRCLK2 frequencies are identical (which is the case when the transmitter user data width is narrower than or equal to the size of the internal data width), then only a single BUFG_GT is instantiated within the helper block. This buffer drives both gtwiz_userclk_tx_usrclk_out and gtwiz_userclk_tx_usrclk2_out helper block output ports, which are wired to the TXUSRCLK and TXUSRCLK2 input ports, respectively, of each transceiver channel primitive. The helper block configures the BUFG_GT to divide the source clock down to the correct user clock frequency as required.

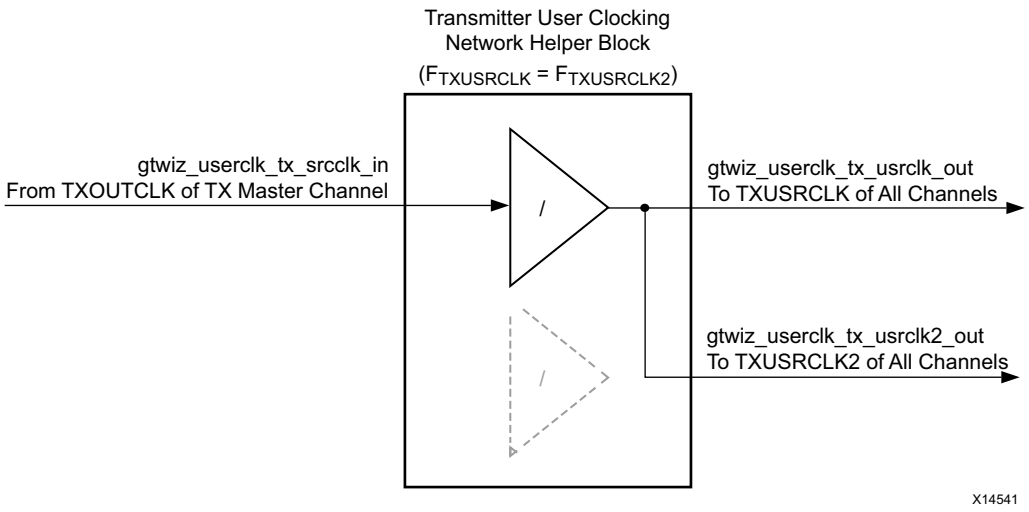

Figure 3-2: **Transmitter User Clocking Network Helper Block (with One BUFG_GT)**

As shown in Figure 3-3, if TXUSRCLK is twice the frequency of TXUSRCLK2 (which is the case when the transmitter user data width is wider than the internal data width), then two

BUFG_GT primitives are instantiated within the helper block. The helper block configures one BUFG_GT to divide the source clock down to the correct transmitter datapath frequency and drive the `gtwiz_userclk_tx_usrclk_out` helper block output port, which is wired to the `TXUSRCLK` input port of each transceiver channel primitive. The helper block configures the other BUFG_GT to divide the source clock down to the correct transmitter user interface frequency and drive the `gtwiz_userclk_tx_usrclk2_out` helper block output port, which is wired to the `TXUSRCLK2` input port of each transceiver channel primitive.

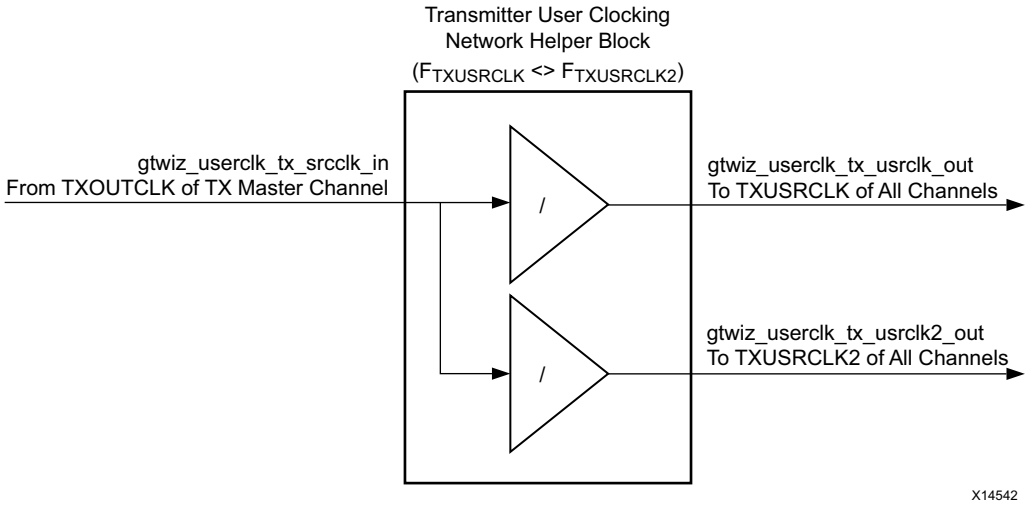

Figure 3-3: **Transmitter User Clocking Network Helper Block**

The helper block holds BUFG_GT primitive(s) in reset when the `gtwiz_userclk_tx_reset_in` user input is asserted. This reset input should be held High until the source clock input is known to be stable. When the reset input is released, the `gtwiz_userclk_tx_active_out` user indicator synchronously asserts, indicating an active user clock and allowing dependent helper blocks to proceed.

The helper block can be located either within the core, or in the example design, per user selection. If included within the core, wiring from the master transceiver channel primitive `TXOUTCLK` output port to the helper block `gtwiz_userclk_tx_srcclk_in` input port is also internal to the core, but that clock signal is presented on the core interface as `gtwiz_userclk_tx_srcclk_out`. Similarly, wiring between the helper block `gtwiz_userclk_tx_usrclk_out` and `gtwiz_userclk_tx_usrclk2_out` output ports and the transceiver channel primitives is internal to the core but those helper block outputs are also presented on the core interface.

If the helper block is located within the example design, then by necessity, the relevant transceiver channel clock ports are enabled on the core interface so that the necessary signals can cross the core boundary.

If additional clock signals or related ports are required for your application, you can enable the relevant ports on the core instance through the optional ports interface during IP

customization. See Chapter 2, Product Specification, for a description of all transmitter user clocking network helper block ports. See the *UltraScale Architecture GTH Transceivers User Guide* (UG576) [Ref 1] or *UltraScale Architecture GTY Transceivers User Guide* (UG578) [Ref 2] for complete documentation on clocking the transceiver primitives.

Receiver User Clocking Network Helper Block

The receiver user clocking network helper block is a simple module used to derive and buffer the appropriate clocks to drive `RXUSRCLK` and `RXUSRCLK2` inputs of one or more transceiver channel primitives.

A single instance of the helper block is usually delivered with each instance of the Wizard IP core. Alternatively, when the receiver elastic buffer is bypassed and single-lane buffer bypass mode is enabled, one instance of the helper block is delivered for, and wired to, each independently clocked transceiver channel primitive instance.

By default, the helper block source clock input port `gtwiz_userclk_rx_srcclk_in` is driven by either the `RXOUTCLK` port of the master transceiver channel (in most configurations) or by the `RXOUTCLK` port of the corresponding transceiver channel (when single-lane buffer bypass is used). Within the helper block, this source drives either one or two BUFG_GT primitives, which are global clock buffers that are capable of clock division.

As shown in Figure 3-4, if `RXUSRCLK` and `RXUSRCLK2` frequencies are identical (which is the case when the receiver user data width is narrower than or equal to the size of the internal data width), then only a single BUFG_GT is instantiated within the helper block. This buffer drives both `gtwiz_userclk_rx_usrclk_out` and `gtwiz_userclk_rx_usrclk2_out` helper block output ports, which are wired to the `RXUSRCLK` and `RXUSRCLK2` input ports, respectively, of the appropriate transceiver channel primitive(s). The helper block configures the BUFG_GT to divide the source clock down to the correct user clock frequency as required.

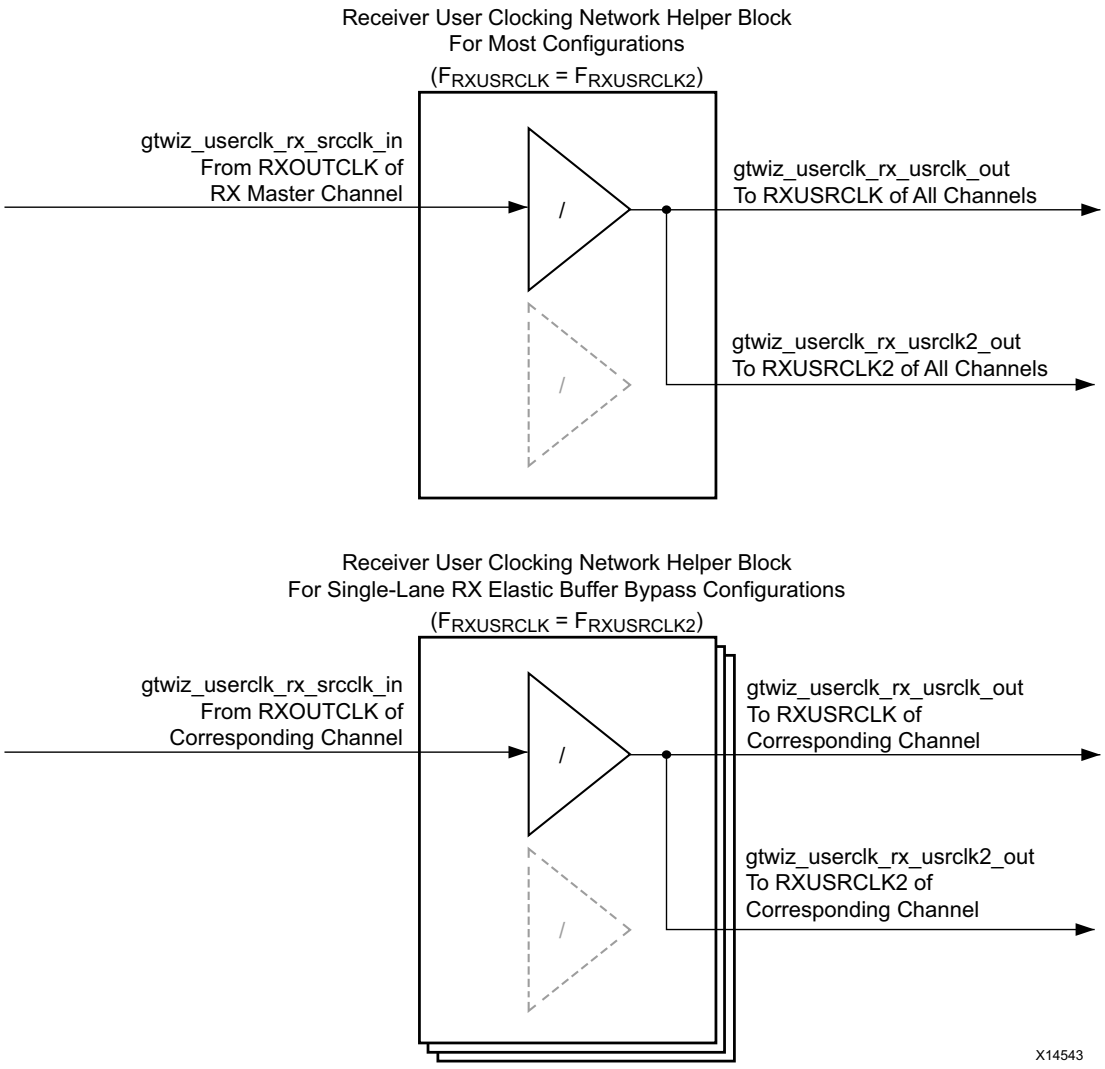

Figure 3-4: **Receiver User Clocking Network Helper Block (with One BUFG_GT)**

As shown in Figure 3-5, if RXUSRCLK is twice the frequency of RXUSRCLK2 (which is the case when the receiver user data width is wider than the internal data width), then two BUFG_GT primitives are instantiated within the helper block. The helper block configures one BUFG_GT to divide the source clock down to the correct receiver datapath frequency and drive the gtwiz_userclk_rx_usrclk_out helper block output port, which is wired to the RXUSRCLK input port of the appropriate transceiver channel primitive(s). The helper block configures the other BUFG_GT to divide the source clock down to the correct receiver user interface frequency and drive the gtwiz_userclk_rx_usrclk2_out helper block output port, which is wired to the RXUSRCLK2 input port of the appropriate transceiver channel primitive(s).

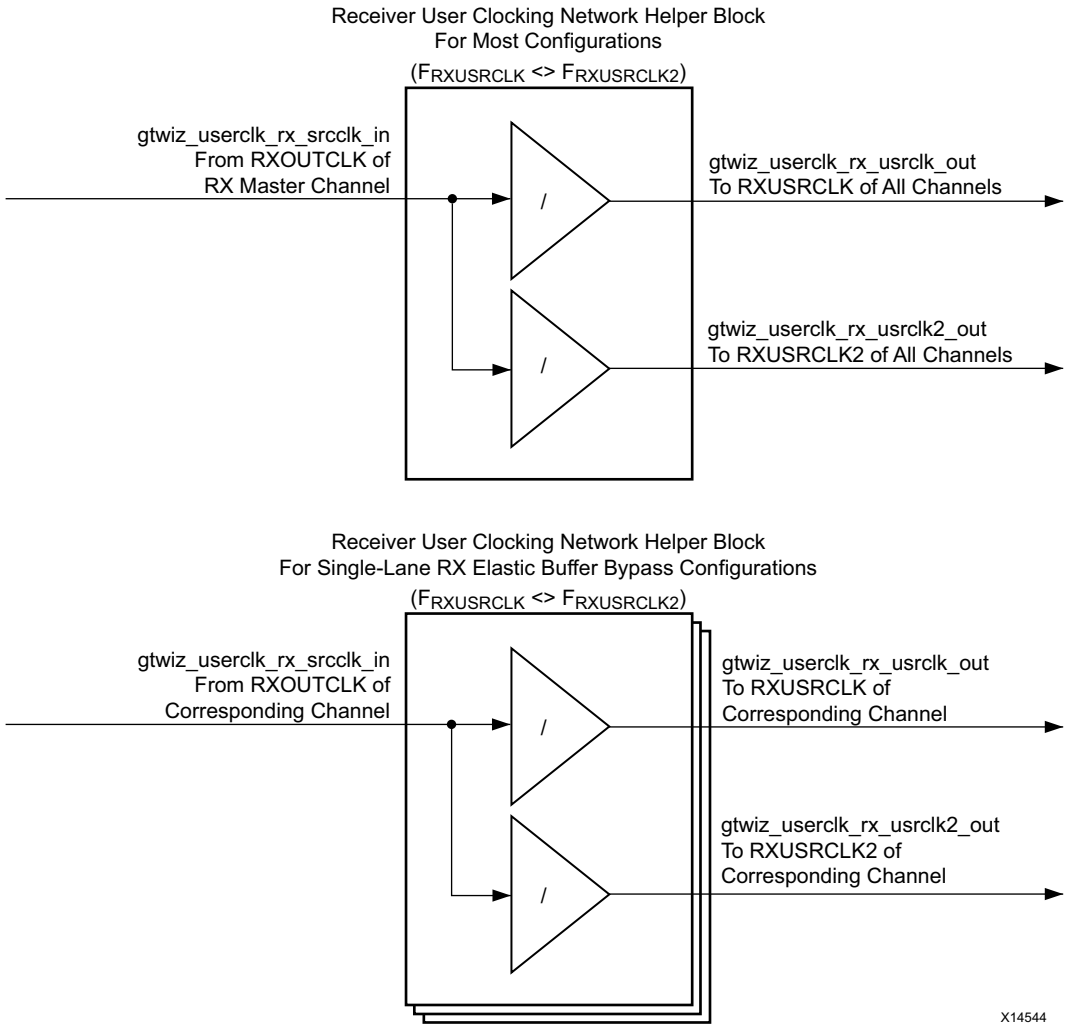

Figure 3-5: **Receiver User Clocking Network Helper Block (with Two BUFG_GT Primitives)**

The helper block holds BUFG_GT primitive(s) in reset when the `gtwiz_userclk_rx_reset_in` user input is asserted. This reset input should be held High until the source clock input is known to be stable. When the reset input is released, the `gtwiz_userclk_rx_active_out` user indicator synchronously asserts, indicating an active user clock and allowing dependent helper blocks to proceed.

The helper block can be located either within the core or in the example design per user selection. If included within the core, wiring from the appropriate transceiver channel primitive `RXOUTCLK` output port(s) to the helper block `gtwiz_userclk_rx_srcclk_in` input port(s) is also internal to the core, but that clock signal is presented on the core interface as `gtwiz_userclk_rx_srcclk_out`.

Similarly, wiring between the helper block `gtwiz_userclk_rx_usrclk_out` and `gtwiz_userclk_rx_usrclk2_out` output ports and the transceiver channel primitives is internal to the core, but those helper block outputs are also presented on the core interface. If the helper block is located within the example design, then the relevant

transceiver channel clock ports are enabled on the core interface so that the necessary signals can cross the core boundary.

If additional clock signals or related ports are required for your application, you can enable the relevant ports on the core instance through the optional ports interface during IP customization. For a description of all receiver user clocking network helper block ports, see Chapter 2, Product Specification. See the *UltraScale Architecture GTH Transceivers User Guide* (UG576) [Ref 1] or *UltraScale Architecture GTY Transceivers User Guide* (UG578) [Ref 2] for complete documentation on clocking the transceiver primitives.

User Data Width Sizing Helper Block

The user data width sizing helper block simplifies the process of interfacing to the transmitter and receiver data ports of the transceiver channel primitives.

The `TXDATA` and `RXDATA` ports of each transceiver channel are 128 bits, but only those bits within the range of the configured transmitter and receiver user data widths are used; other bits are to be tied off or left unconnected, respectively. When multiple channels are enabled, it can be inconvenient to identify the active bits of the `txdata_in` and `rxdata_out` core port vectors. Furthermore, for user data widths of 20, 40, 80, or 160 bits, portions of `TXCTRL0` and `TXCTRL1` are interleaved with `TXDATA`, and portions of `RXCTRL0` and `RXCTRL1` are interleaved with `RXDATA`.

The helper block handles this transceiver-facing complexity while providing a simple user interface sized to the chosen user data width utilizing wiring only. The helper block is divided into two independent modules: a transmitter module and a receiver module. Both modules use generated HDL wire assignments. Because no combinatorial or sequential logic is used, there is no impact on the datapath.

Transmitter Module

The helper block transmitter module port `gtwiz_userdata_tx_in` is a vector sized to the chosen transmitter user data width, multiplied by the number of enabled transceiver channels. By core convention, its least significant bits correspond to the least significant bits of the transceiver channel in the lowest enabled XY grid position.

Figure 3-6 shows the helper block configuration for an example core configuration using a 32-bit transmitter user data width and four enabled transceiver channels. With the resulting packed 128-bit `gtwiz_userdata_tx_in` vector, the helper block drives the appropriate bits of each transceiver channel's `TXDATA` port. For 20-, 40-, 80-, and 160-bit transmitter user data widths, it also drives the appropriate bits of each transceiver channel's `TXCTRL0` and `TXCTRL1` ports, handling the required de-interleaving. In other configurations, the `TXCTRL0` and `TXCTRL1` ports are not driven by the helper block and are available for user access.

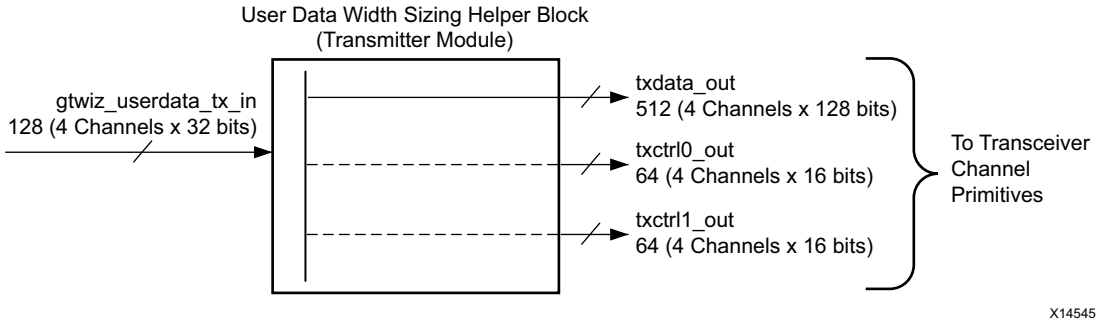

Figure 3-6: **User Data Width Sizing Helper Block (Transmitter Module) Example Configuration**

Receiver Module

The helper block receiver module port `gtwiz_userdata_rx_out` is a vector sized to the chosen receiver user data width multiplied by the number of enabled transceiver channels. By core convention, its least significant bits correspond to the least significant bits of the transceiver channel in the lowest enabled XY grid position.

Figure 3-7 shows the helper block configuration for an example core configuration using a 32-bit receiver user data width and four enabled transceiver channels. The resulting packed 128-bit `gtwiz_userdata_rx_out` helper block vector provides received data from the appropriate bits from each transceiver channel's `RXDATA` port. For 20-, 40-, 80-, and 160-bit receiver user data widths, it also provides the appropriate bits from each transceiver channel's `RXCTRL0` and `RXCTRL1` ports, handling the required interleaving. The `RXCTRL0` and `RXCTRL1` ports are available for user access.

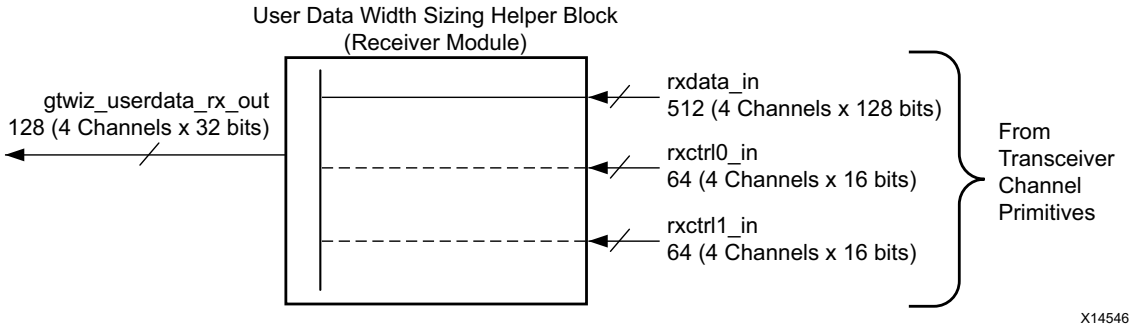

Figure 3-7: **User Data Width Sizing Helper Block (Receiver Module) Example Configuration**

Transmitter Buffer Bypass Controller Helper Block

The transmitter buffer bypass controller helper block automates the buffer bypass procedure that must be performed when the serial transceiver transmitter buffer is not used. The helper block implements the auto-mode buffer bypass sequence.

A single instance of the helper block is delivered with each instance of the Wizard IP core that is configured to bypass the transmitter buffer. Its user interface provides you with a simple means of initiating and monitoring the status of the transmitter buffer bypass procedure. Its transceiver interface connects to each transceiver channel primitive within the core.

For core configurations that contain multiple serial transceiver primitives, the helper block implements the multi-lane buffer bypass procedure. The transmitter master channel is specified during IP customization.

The helper block is synchronously reset when the `gtwiz_buffbypass_tx_reset_in` user input is asserted. This signal should be released as soon as `TXUSRCLK2` is stable, and before the transmitter datapath reset sequence completes for all channels. By default, the reset helper block `gtwiz_reset_tx_done_out` output is wired to the transmitter buffer bypass controller helper block `gtwiz_buffbypass_tx_resetdone_in` input. A rising edge on this port automatically initiates the transmitter buffer bypass procedure.

When the transmitter buffer bypass procedure completes, the `gtwiz_buffbypass_tx_done_out` user indicator is asserted and the `gtwiz_buffbypass_tx_error_out` indicator is set. The two user interface outputs should be considered together to decode the result of the buffer bypass procedure, as shown in Table 3-2.

Table 3-2: **Transmitter Buffer Bypass Controller Helper Block Completion Result Encoding**

gtwiz_buffbypass_tx_done_out	gtwiz_buffbypass_tx_error_out	Buffer Bypass Procedure Result
0	Any	Not complete
1	0	Completed successfully
1	1	Completed with error

By pulsing the `gtwiz_buffbypass_tx_start_user_in` user input, you can also force the transmitter buffer bypass controller helper block to initiate the buffer bypass procedure at any time after the helper block has been reset and the initial procedure has completed.

The helper block can be located either within the core or in the example design per user selection. Depending on its location and the location of other helper blocks, the relevant ports are enabled on the core interface so that the necessary signals can cross the core boundary. If you choose to locate the helper block within the core but also wish to observe individual transceiver primitive buffer bypass status signals, you can enable the relevant ports on the core instance through the optional ports interface during IP customization.

See Chapter 2, Product Specification, for a description of all transmitter buffer bypass controller helper block ports. See the *UltraScale Architecture GTH Transceivers User Guide* (UG576) [Ref 1] or *UltraScale Architecture GTY Transceivers User Guide* (UG578) [Ref 2] for complete information about bypassing the transmitter buffer in transceiver primitives.

Receiver Buffer Bypass Controller Helper Block

The receiver buffer bypass controller helper block automates the buffer bypass procedure, which must be performed when the serial transceiver receiver elastic buffer is not used. The helper block implements the auto-mode buffer bypass sequence. When the wizard is configured to bypass the receiver elastic buffer, the following takes place:

- If multi-lane buffer bypass mode is enabled, one instance of the helper block is delivered and implements the multi-lane buffer bypass procedure. In this case, the user interface port width is not scaled, while the transceiver interface is scaled to the number of channel primitives in the core.
- If single-lane buffer bypass mode is enabled, an instance of the helper block is delivered for, and wired to each transceiver channel primitive instance. In this case, the user interface port width scales with the number of helper blocks, while the transceiver interface connects to its corresponding channel primitive only.

The helper block user interface provides you with a simple means of initiating and monitoring the status of the receiver buffer bypass procedure. Its transceiver interface connects to transceiver channel primitive(s) within the core.

The helper block is synchronously reset when the `gtwiz_buffbypass_rx_reset_in` user input is asserted. This signal should be released as soon as `RXUSRCLK2` of the receiver master channel (for multi-lane buffer bypass configurations), or `RXUSRCLK2` of the corresponding channel (for single-lane buffer bypass configurations) is stable, and before the receiver datapath reset sequence completes for all channels. By default, the reset helper block `gtwiz_reset_rx_done_out` output is wired to the receiver buffer bypass controller helper block `gtwiz_buffbypass_rx_resetdone_in` input. A rising edge on this port automatically initiates the receiver buffer bypass procedure.

When the receiver buffer bypass procedure completes, the `gtwiz_buffbypass_rx_done_out` user indicator is asserted and the `gtwiz_buffbypass_rx_error_out` indicator is set. The two user interface outputs should be considered together to decode the result of the buffer bypass procedure, as shown in Table 3-3.

Table 3-3: **Transmitter Buffer Bypass Controller Helper Block Completion Result Encoding**

gtwiz_buffbypass_rx_done_out	gtwiz_buffbypass_rx_error_out	Buffer Bypass Procedure Result
0	Any	Not complete
1	0	Completed successfully
1	1	Completed with error

You can force the receiver buffer bypass controller helper block to initiate the buffer bypass procedure at any time after the helper block has been reset and the initial procedure has completed by pulsing the `gtwiz_buffbypass_rx_start_user_in` user input.

The helper block can be located either within the core or in the example design per user selection. Depending on its location and the location of other helper blocks, the relevant ports are enabled on the core interface so that the necessary signals can cross the core boundary.

If you choose to locate the helper block within the core but also wish to observe individual transceiver primitive buffer bypass status signals, you can enable the relevant ports on the core instance through the optional ports interface during IP customization.

See Chapter 2, Product Specification, for a description of all receiver buffer bypass controller helper block ports. See the *UltraScale Architecture GTH Transceivers User Guide* (UG576) [Ref 1] or *UltraScale Architecture GTY Transceivers User Guide* (UG578) [Ref 2] for complete documentation on bypassing the receiver elastic buffer in transceiver primitives.

Transceiver Common Primitive

The transceiver common primitive is required and instantiated for core configurations where QPLL0 or QPLL1 clocking resources are used. Although it is a transceiver primitive, its logical location can be specified during IP customization. Like a helper block, it can be located either within the core or in the example design.

By providing this flexibility, it might be possible to share a single transceiver common between multiple Wizard IP core instances when the following conditions are met:

- You attempt to place the physical transceiver resources of those core instances into a single transceiver Quad, and
- The transceiver common configuration is identical or otherwise safe to share between the two core instances.

Transceiver common sharing between core instances is an advanced use mode and should only be performed when restrictions and limitations are fully understood. See the *UltraScale Architecture GTH Transceivers User Guide* (UG576) [Ref 1] or *UltraScale Architecture GTY Transceivers User Guide* (UG578) [Ref 2] for details on the use of the transceiver common primitive.

Figure 3-8 illustrates the case where one or more transceiver common primitives are enabled and located within the core instance, when user-specified. In this case, the `QPLL#OUTCLK` and `QPLL#OUTREFCLK` ports of the transceiver common primitives internally drive the `QPLL#CLK` and `QPLL#REFCLK` ports of the associated transceiver channel primitives, as required (where # is 0 or 1 for `QPLL0` or `QPLL1`, respectively). However, those same signals are also provided as outputs on the core interface as `qpll#outclk_out` and `qpll#outrefclk_out`.

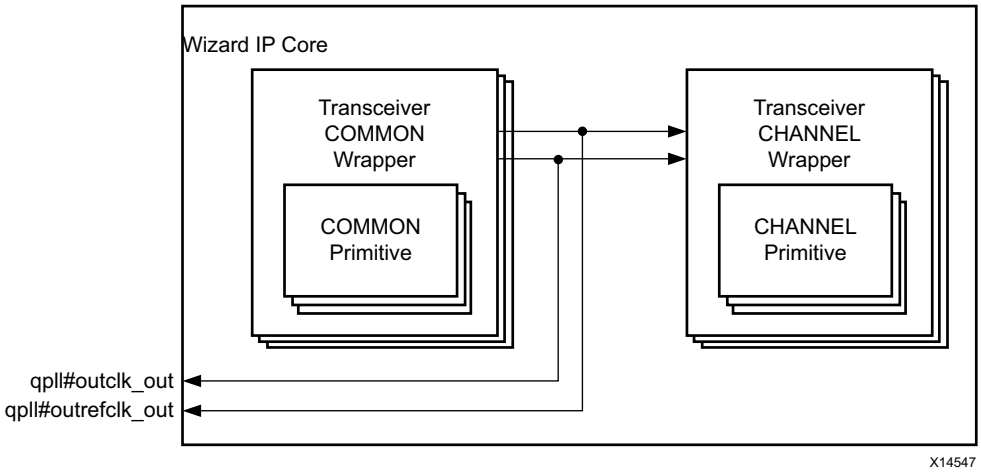

Figure 3-8: **Transceiver Common Located in the Core**

Figure 3-9 illustrates the case where one or more transceiver common primitives are enabled but located within the example design, when user-specified. In this case, the `QPLL#OUTCLK` and `QPLL#OUTREFCLK` ports of the transceiver common primitives drive the `qpll#clk_in` and `qpll#refclk_in` input ports on the core interface, which in turn are connected to the `QPLL#CLK` and `QPLL#REFCLK` ports of the associated transceiver channel primitives.

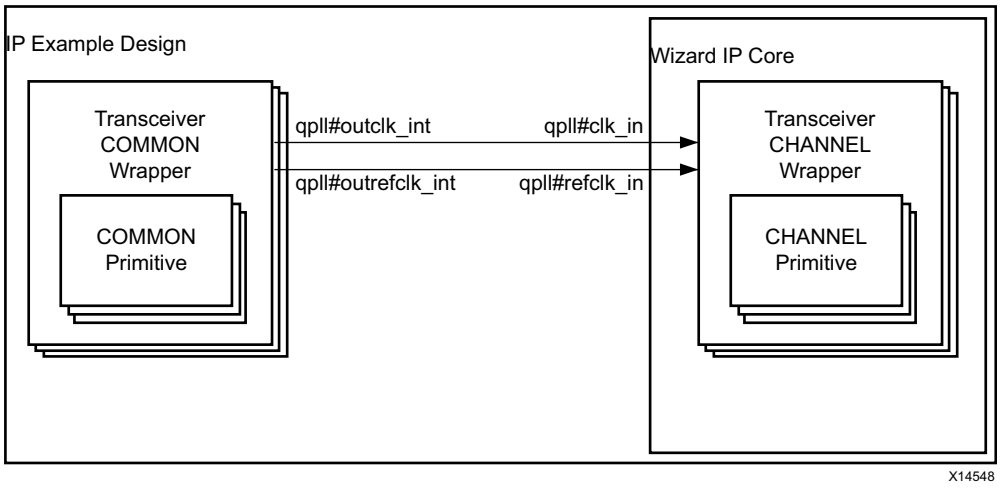

Figure 3-9: **Transceiver Common Located in the Example Design**

When integrating multiple core instances into your system, the two types of customizations described earlier can be combined to share the transceiver common resources that are present within the instance that contains them. Figure 3-10 illustrates how the output ports on the core instance that contains the transceiver common resources can be easily connected to the associated input ports on the core instance which does not. Essentially, the two instances are connected together when integrating the cores, excluding the example design wrappers.

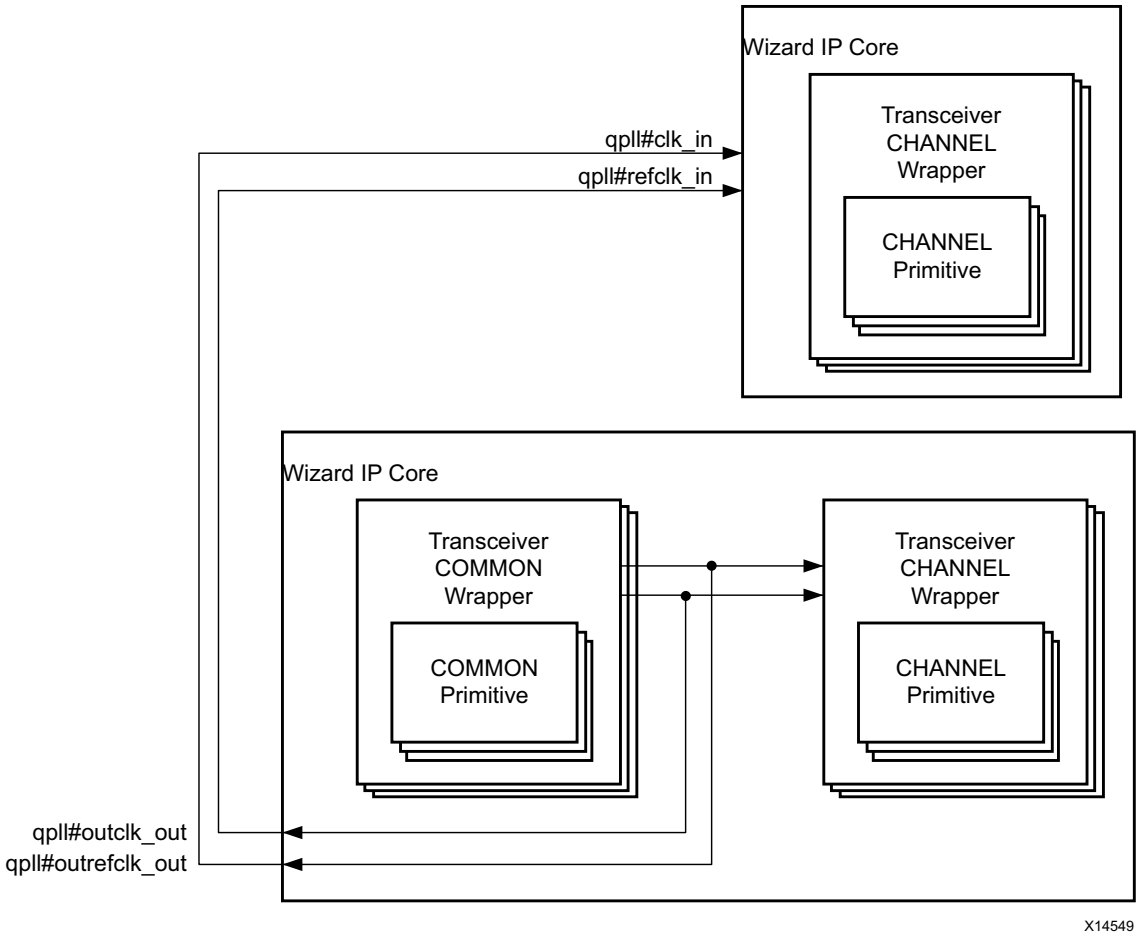

Figure 3-10: **Transceiver Common Sharing**

See AR#65228 for information on how to share a COMMON block using GTH transceivers.

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 can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 8]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 9]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 10]

Customizing and Generating the Core

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

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

1. In the Vivado Design Suite, create a new project or open an existing project that is configured to target one of the supported UltraScale™ or UltraScale+™ devices.

IMPORTANT: *It is important to choose the exact part because characteristics such as speed grade, temperature grade, and silicon level affect the available features and performance limits of the serial transceivers. Limitations based on device characteristics are represented by the available choices when customizing the Wizard IP in the Vivado Integrated Design Environment (IDE).*

2. Open the IP catalog and select the IP at **FPGA Features and Design > I/O Interfaces > UltraScale FPGAs Transceivers Wizard**.
3. Double-click the IP or select the **Customize IP** command from the toolbar or right-click menu to display the Wizard Customize IP dialog box.

   Note: This core is not available in the Vivado IP integrator.

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

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

Vivado IDE Customization Parameters

The Wizard IP parameters is organized in four tabs:

- **Basic**: Provides customization options for fundamental transceiver features, including transceiver type, and transmitter and receiver settings. Also provides the ability to select a transceiver configuration preset. See Basic Tab.
- **Physical Resources**: Provides table and graphical interfaces to select specific transceiver channel sites to enable, as well as reference clock routing options. See Physical Resources Tab.
- **Optional Features**: Provides extensive configuration options for optional or advanced features, if appropriate for your application. See Optional Features Tab.
- **Structural Options**: Provides location choices for each available helper block, and the optional port enablement interface. See Structural Options Tab.

Review each of the available options and modify them as desired so that the resulting core instance meets your system requirements. For a full understanding of transceiver primitive features and available use modes, see the *UltraScale Architecture GTH Transceivers User Guide* (UG576) [Ref 1] or *UltraScale Architecture GTY Transceivers User Guide* (UG578) [Ref 2].

Component Name and Symbol

The name of the generated IP is set in the Component Name field. The default name is **gtwizard_ultrascale_0**. This must be set to a name that is unique within your project.

The IP symbol is shown on the left-hand side of the Customize IP dialog box, and displays only enabled ports by default. It organizes input ports on the left-hand side of the symbol and output ports on the right-hand side. The IP symbol is updated as you make customization choices and use the optional ports enablement interface. See Chapter 2, Product Specification for a detailed description of ports and their use.

Basic Tab

IP customization options in the Basic tab (Figure 4-1) are described in the following subsections. Selections apply to each enabled transceiver channel in the core instance. See the *UltraScale Architecture GTH Transceivers User Guide* (UG576) [Ref 1] or *UltraScale Architecture GTY Transceivers User Guide* (UG578) [Ref 2] for full details on available choices for each customization option.

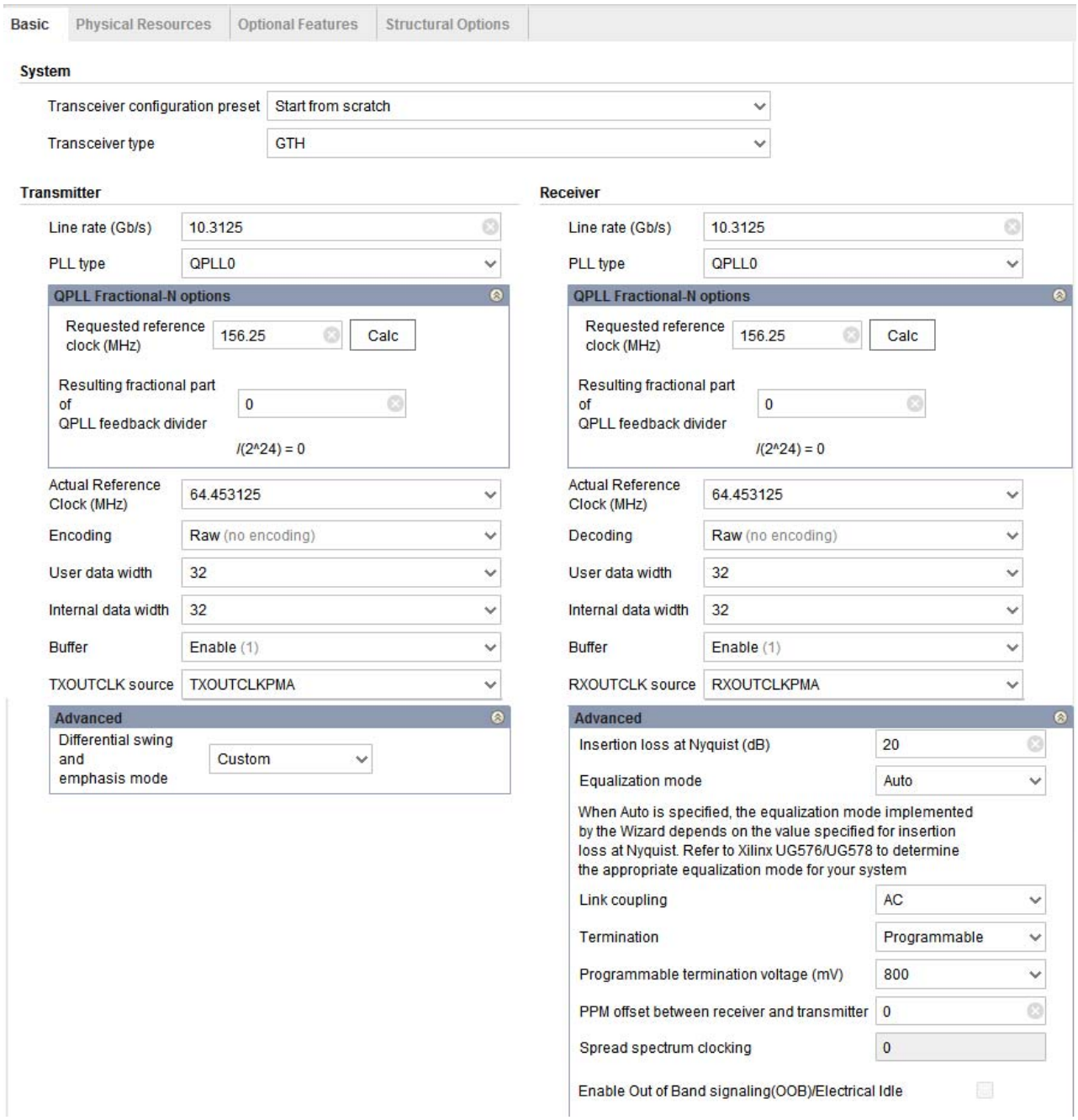

Figure 4-1: **Basic Tab**

System Frame

Overall system settings are customized by options in the system frame. Note that the XGUI design is top-down priority. The transmitter updates would have higher priority and could update the dependent receiver customizations.

- **Transceiver configuration preset.** Several industry-standard configuration presets are available for selection. Choosing a preset configures options as appropriate for that

industry standard. After applying the preset, you can further modify options as needed for your specific system and its protocol IP. Leave the selection set to "Start from scratch" if you wish to make fully custom selections. You can also click "Switch to Defaults" at the top of the screen to return all options to the "Start from scratch" defaults.

- **Transceiver type.** Select the type of serial transceiver to configure. Available choices are limited to the transceiver types present within the selected device.

Transmitter Frame

Serial transceiver transmitter settings are customized by options in the transmitter frame.

- **Line rate (Gb/s).** Enter the transmitter line rate in gigabits per second. The available range depends on transceiver type and can be limited by the selected device.
- **PLL type.** Select the desired PLL type used to clock the transmitter of each enabled serial transceiver channel. Possible options are QPLL0, QPLL1, and CPLL, but available choices can be limited by the selected device and transmitter line rate. When QPLL0 or QPLL1 is chosen, one or more transceiver common primitives is instantiated. If the transmitter and receiver line rates differ, different PLL types might be required for each data direction.
- **QPLL Fractional-N options**. For configurations targeting a QPLL in a device which supports the fractional-N feedback divider, enter a **Requested reference clock (MHz)** value and click the **Calc** button. This action populates the **Actual reference clock (MHz)** field with a variety of supported transmitter reference clock frequencies based on the requested value. In most cases, the requested frequency is available for selection. The calculation also populates the **Fractional part of QPLL feedback divider** field with the numerator of the fractional part of the QPLL feedback divider used to clock the transmitter datapath. This value can be manually tweaked to fine-tune the available **Actual reference clock (MHz)** selections for advanced use cases. Possible values are 0 through 16777215, where 0 disables fractional-N operation. Changing this field updates the available **Actual reference clock (MHz)** selections.
- **Actual reference clock (MHz)**. Select the desired frequency, from among all compatible frequencies, for the reference clock that will be provided to the selected PLL type to achieve the selected transmitter line rate.
- **Encoding.** Select the type of encoding or data format handling you want the transceiver to apply when data is transmitted. The options and their characteristics are as follows, but choices can be limited by selected device, transceiver type, and line rate:
  - *Raw (no encoding)*. Data is transmitted as provided.
  - *8B/10B*. Data is encoded in 8B/10B format before being transmitted.
  - *Sync. gearbox for 64B/66B*. Data is transmitted using the TX Synchronous Gearbox in normal mode for 64B/66B applications.

- *Sync. gearbox for 64B/66B (CAUI mode).* Data is transmitted using the TX Synchronous Gearbox in CAUI (dual data stream) mode for 64B/66B applications.
- *Async. gearbox for 64B/66B.* Data is transmitted using the TX Asynchronous Gearbox in normal mode for 64B/66B applications.
- *Async. gearbox for 64B/66B (CAUI mode).* Data is transmitted using the TX Asynchronous Gearbox in CAUI (dual data stream) mode for 64B/66B applications.
- *Sync. gearbox for 64B/67B.* Data is transmitted using the TX Synchronous Gearbox in normal mode for 64B/67B applications.
- *Sync. gearbox for 64B/67B (CAUI mode).* Data is transmitted using the TX Synchronous Gearbox in CAUI (dual data stream) mode for 64B/67B applications.

- **User data width.** Also known as external data width. Select the desired bit width for the transmitter user data interface of each serial transceiver channel. Possible options are 16, 20, 32, 40, 64, 80, 128, and 160 but available choices can be limited by selected device, transceiver type, line rate, and encoding. This selection sets the active portion of the transmitter data vector, which is presented at full size on the core interface unless the user data width sizing helper block is located within the core. The active portion of the transmitter data vector is least significant bit-aligned. Inactive bits should be tied Low.
- **Internal data width.** Select the desired bit width for the internal transmitter datapath of each serial transceiver channel. Possible options are 16, 20, 32, 40, 64, and 80, but available choices are limited by selected device, transceiver type, line rate, encoding, and user data width.
- **Buffer.** Choose whether to enable or bypass the transmitter buffer. The ability to bypass the buffer might be limited by the selected encoding. When the buffer is bypassed, the transmitter buffer bypass controller helper block is provided.
- **TXOUTCLK source.** Select the internal clock source for the `TXOUTCLK` port of each serial transceiver primitive. Possible options are TXPLLREFCLK_DIV1, TXPLLREFCLK_DIV2, TXOUTCLKPCS, TXOUTCLKPMA, and TXPROGDIVCLK, but available choices can be limited by selected device, line rate, reference clock frequency, encoding, internal data width, and buffer usage. The transmitter user clocking network helper block is driven by the `TXOUTCLK` port, and thus by this clock source, of the master transceiver channel.

Transmitter Frame (Advanced Section)

Advanced serial transceiver transmitter settings are customized by options in the collapsible portion of the transmitter frame. Click the title to expand the section.

- **Differential swing and emphasis mode.** Select the transmitter driver mode. Selection determines the set of ports that control the transmitter driver swing and cursors. This option is mainly used only by PCIe™ or Ethernet IP configurations. It is recommended that you use the values generated by GT Wizard IP as is. Manual interaction and change of values for this might be needed only for QPI setup.

Receiver Frame

Serial transceiver receiver settings are customized by options in the receiver frame.

- **Line rate (Gb/s).** Enter the receiver line rate in gigabits per second. The available range depends on transceiver type and can be limited by the selected device.
- **PLL type.** Select the desired PLL type used to clock the receiver of each enabled serial transceiver channel. Possible options are QPLL0, QPLL1, and CPLL, but available choices can be limited by the selected device and receiver line rate. When QPLL0 or QPLL1 is chosen, one or more transceiver common primitives will be instantiated. If the receiver and transmitter line rates differ, different PLL types might be required for each data direction.
- **QPLL Fractional-N options**. For configurations targeting a QPLL in a device which supports the fractional-N feedback divider, enter a **Requested reference clock (MHz)** value and click the **Calc** button. This action populates the **Actual reference clock (MHz)** field with a variety of supported receiver reference clock frequencies based on the requested value. In most cases, the requested frequency is available for selection, but must be equivalent to the analogous transmitter value if the same QPLL is used for both transmitter and receiver. The calculation also populates the **Fractional part of QPLL feedback divider** field with the numerator of the fractional part of the QPLL feedback divider used to clock the receiver datapath. This value can be manually tweaked to fine-tune the available **Actual reference clock (MHz)** selections for advanced use cases. Possible values are 0 through 16777215, where 0 disables fractional-N operation. Changing this field updates the available **Actual reference clock (MHz)** selections.
- **Actual Reference Clock (MHz).** Select the desired frequency from among all compatible frequencies for the reference clock that will be provided to the selected PLL type to achieve the selected receiver line rate.
- **Decoding.** Select the type of decoding or data format handling you want the transceiver to apply when data is received. The options and their characteristics are as follows, but choices can be limited by selected device, transceiver type, line rate, and transmitter data encoding:
  - *Raw (no decoding)*. Data is provided as received.
  - *8B/10B*. Received data is decoded from 8B/10B format.
  - *Sync. gearbox for 64B/66B*. Data is received using the RX Synchronous Gearbox in normal mode for 64B/66B applications.
  - *Sync. gearbox for 64B/66B (CAUI mode)*. Data is received using the RX Synchronous Gearbox in CAUI (dual data stream) mode for 64B/66B applications.
  - *Async. gearbox for 64B/66B*. Data is received using the RX Asynchronous Gearbox in normal mode for 64B/66B applications.

- *Async. gearbox for 64B/66B (CAUI mode).* Data is received using the RX Asynchronous Gearbox in CAUI (dual data stream) mode for 64B/66B applications.
- *Sync. gearbox for 64B/67B.* Data is received using the RX Synchronous Gearbox in normal mode for 64B/67B applications.
- *Sync. gearbox for 64B/67B gearbox (CAUI mode).* Data is received using the RX Synchronous Gearbox in CAUI (dual data stream) mode for 64B/67B applications.

- **User data width.** Also known as external data width. Select the desired bit width for the receiver user data interface of each serial transceiver channel. Possible options are 16, 20, 32, 40, 64, 80, 128, and 160 but available choices can be limited by selected device, transceiver type, line rate, and decoding. This selection sets the active portion of the receiver data vector, which is presented at full size on the core interface unless the user data width sizing helper block is located within the core. The active portion of the receiver data vector is least significant bit-aligned. Inactive bits should be ignored.
- **Internal data width.** Select the desired bit width for the internal receiver datapath of each serial transceiver channel. Possible options are 16, 20, 32, 40, 64, and 80, but available choices are limited by selected device, transceiver type, line rate, decoding, and user data width.
- **Buffer.** Choose whether to enable or bypass the receiver elastic buffer. The ability to bypass the buffer can be limited by the selected decoding. When the buffer is bypassed, the receiver buffer bypass controller helper block is provided.
- **RXOUTCLK source.** Select the internal clock source for the `RXOUTCLK` port of each serial transceiver primitive. Possible options are RXPLLREFCLK_DIV1, RXPLLREFCLK_DIV2, RXOUTCLKPCS, RXOUTCLKPMA, and RXPROGDIVCLK, but available choices can be limited by selected device, line rate, reference clock frequency, decoding, internal data width, and buffer usage. The receiver user clocking network helper block is driven by the `RXOUTCLK` port, and thus by this clock source, of either the master transceiver channel or of each transceiver channel (depending on buffer usage options).

Receiver Frame (Advanced Section)

Advanced serial transceiver receiver settings are customized by options in the collapsible portion of the receiver frame. Click the title to expand the section.

- **Insertion loss at Nyquist (dB).** Specify the insertion loss of the channel between the transmitter and receiver at the Nyquist frequency in dB.
- **Equalization mode.** Select between decision feedback equalization (DFE) mode and low-power mode (LPM) for the receiver equalization. When the Auto option is selected, the mode is set automatically based on the channel insertion loss, where a value greater than 14 dB causes DFE to be used, otherwise LPM is used. Refer to the *UltraScale Architecture GTH Transceivers User Guide* (UG576) [Ref 1] or the *UltraScale Architecture GTY Transceivers User Guide* (UG578) [Ref 2] for further guidance.

- **Link coupling.** Options are AC and DC. Select AC if external AC coupling is enabled in the application, and DC otherwise.
- **Termination.** Select the receiver termination voltage. Your choice of termination should depend on the protocol and its link coupling.
- **Programmable termination voltage (mV).** When termination is set to programmable, select the termination voltage in mV.
- **PPM offset between receiver and transmitter.** Specify the offset between received data and transmitter data in PPM. For example, if your protocol specifies ±100 ppm, you would enter 200 in this field. This offset affects the receiver CDR settings.
- **Spread spectrum clocking.** Specify the spread spectrum clocking (SSC) modulation in PPM. SSC affects the receiver CDR settings.
- **Enable Out of Band signaling (OOB)/Electrical Idle.** Select this option to enable Out of Band (OOB) signaling/Electrical Idle. Availability is subject to the supported receiver line rate, data decoding, reference clock frequency, termination, programmable termination value, and link coupling selections.

Physical Resources Tab

IP customization options in the Physical Resources tab are described in the following subsections. When customizing options on this tab, it is important to understand that choices you make affect generated HDL and constraints. Select the options that are appropriate for your project and system. See the *UltraScale Architecture GTH Transceivers User Guide* (UG576) [Ref 1] or *UltraScale Architecture GTY Transceivers User Guide* (UG578) [Ref 2] for details on the transceiver Quad architecture and reference clocking options. The layout of the Physical Resources tab is shown in Figure 4-2.

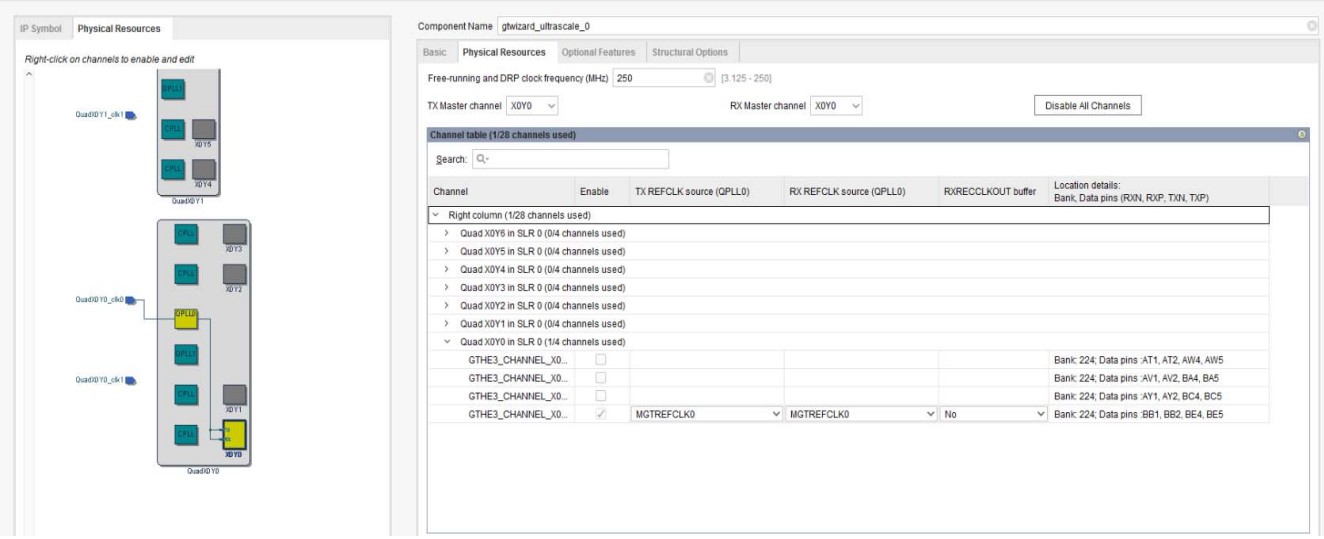

Figure 4-2: **Physical Resources Tab**

Free-Running and DRP Clock Frequency (MHz)

Specify the frequency of the required free-running clock that will be provided to bring up the core and to clock various helper blocks. An accurate frequency is required to construct clock constraints and parameterize certain design modules. For GTH transceiver configurations that target engineering sample (ES1 or ES2) UltraScale devices and that use the CPLL, this clock must also be used for the transceiver channel DRP interface. See Performance, page 10 for maximum frequency guidance.

TX Master Channel and RX Master Channel

Independently select the master transmitter and receiver channels from among all enabled transceiver channels. Enabled channels are identified by their coordinates on the transceiver channel grid. In the generated core instance, the TX master channel drives the source clock input of the transmitter user clocking network helper block, and the RX master channel drives the source clock input of the receiver user clocking network helper block. The TX and RX master channel designations are also used to configure the buffer bypass master lane when the transmitter buffer or receiver elastic buffer is bypassed, respectively.

Channel Table and Channel Graphic

Transceiver channel enablement, reference clock source, and recovered clock selections are customized by options in the channel table or channel graphic. The channel graphic is shown on the left side of the Customization dialog box in place of the IP symbol when interacting with Physical Resources tab options. The channel table contains interactive Channel Enable, TX REFCLK source, RX REFCLK source, and RXRECCLKOUT buffer column options, and an informative Location details column. Available channels are organized by their column and Quad locations. The interactive channel graphic provides the same customization options and aids in visualizing transceiver primitive and reference clocking topology.

- **Enabling a channel.** To enable a particular transceiver channel for use, either click the corresponding checkbox in the channel table or right-click the channel in the graphic and select **Enable**. Enabling a channel causes that physical transceiver site to be instantiated, connected, and appropriately constrained in the generated core instance. At least one channel must be enabled at all times; to disable the default channel, first enable the desired channel(s), then disable the default. Click the **Disable All Channels** button to return transceiver channel enablement to its default state. Transceiver channels are organized by column and Quad, and are named according to their coordinates on the transceiver channel grid. Channels can also be identified by their serial data pins as shown in the Data pins column.
- **Choosing a transmitter reference clock source.** A valid transmitter reference clock source must be chosen for each enabled channel. You can choose a source from the TX REFCLK column of the channel table or by right-clicking the channel in the graphic. The transmitter PLL type selected during Basic tab customization is shown for reference. Choosing a reference clock source for a channel causes that buffered input to be

routed to the channel, and for appropriate constraints to be generated. Each unique reference clock source selected requires a differential clock input to the device.

- **Choosing a receiver reference clock source.** A valid receiver reference clock source must be chosen for each enabled channel. You can choose a source from the RX REFCLK column of the channel table or by right-clicking the channel in the graphic. The receiver PLL type selected during Basic tab customization is shown for reference. Choosing a reference clock source for a channel causes that buffered input to be routed to the channel, and for appropriate constraints to be generated. Each unique reference clock source selected requires a differential clock input to the device.

Note: The wizard always connects the buffered reference clock signal to the "GTREFCLK0" position of the appropriate PLL clock multiplexer(s) even if MGTREFCLK1 or a reference clock requiring north or south routing is selected. This is a supported simplification use mode, and the Vivado design tools handle the required routing complexity. GTGREFCLK should be used only for testing purposes. For more information on the use of GTGREFCLK, see *UltraScale Architecture GTH Transceivers User Guide* (UG576) [Ref 1] or *UltraScale Architecture GTY Transceivers User Guide* (UG578) [Ref 2].

- **Choosing a recovered clock source and buffer.** The recovered clock of a transceiver channel can be buffered and driven out of the device. For an enabled transceiver channel, you can choose an available output buffer from the RXRECCLKOUT buffer column of the channel table, or by right-clicking the channel in the graphic. This feature requires using one of the available differential clock buffers within that transceiver's Quad as an output, preventing that same resource from being used as a reference clock input buffer. Choosing a recovered clock source and buffer causes the channel's recovered clock output to be routed to an instantiated output buffer primitive and the appropriate constraints to be generated.

Optional Features Tab

IP customization options in the Optional Features tab are described in the following subsections. The use of each of these features is optional. You need not customize the options for a given feature if your application does not use that feature. See the *UltraScale Architecture GTH Transceivers User Guide* (UG576) [Ref 1] or *UltraScale Architecture GTY Transceivers User Guide* (UG578) [Ref 2] for details on the relevant serial transceiver features. The layout of the Optional Features tab with the Receiver comma detection and alignment section expanded is shown in Figure 4-3.

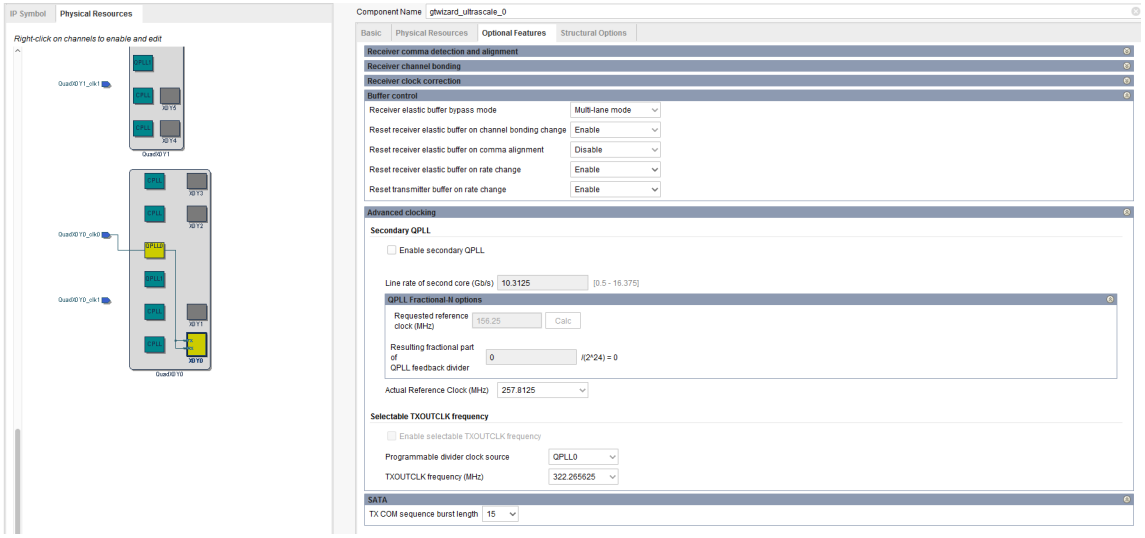

Figure 4-3: **Optional Features Tab**

Receiver Comma Detection and Alignment Section

Various customization options relating to the detection of received comma characters and data alignment to those characters are presented in this collapsible section. For more information, see *UltraScale Architecture GTH Transceivers User Guide* (UG576) [Ref 1] or *UltraScale Architecture GTY Transceivers User Guide* (UG578) [Ref 2]. Click the title to expand the section.

- **Valid comma values for 8B/10B decoding.** Select whether all 8B/10B commas or just IEEE Std 802.3-specified comma characters are decoded as comma characters.
- **Plus comma.** Mark the checkbox under "Detect" to enable detection of the provided bit pattern as a plus comma. You can enter a pattern directly in the text box under "Value" or choose an option in the comma preset box to specify the standard plus comma pattern.
- **Minus comma.** Mark the checkbox under "Detect" to enable detection of the provided bit pattern as a minus comma. You can enter a pattern directly in the text box under "Value" or choose an option in the comma preset box to specify the standard minus comma pattern.
- **Mask.** Enter a comma mask bit pattern. Any bit set to "0" makes the comma detection block treat corresponding plus and minus comma values as a "don't care".
- **Detect combined plus/minus (double length) comma.** Mark the checkbox to enable the transceiver to search for the two commas in a row.
- **Alignment boundary.** Select which data byte boundaries are allowed for comma alignment. Possible options are any byte boundary, two byte boundaries, four byte boundaries, and eight byte boundaries, but available selections can be limited by receiver internal data width.

- **Show realign comma.** Mark the checkbox to cause commas which cause realignment to be shown at the receiver interface.
- **Manual alignment (RXSLIDE) mode.** If RXSLIDE will be used to implement manual alignment, select which mode to enable. Possible options are Off (to disable manual alignment), PCS, PMA, and Automated PMA, but available selections can be limited by receiver data decoding and receiver elastic buffer usage.

Receiver Channel Bonding Section

Various customization options relating to receiver channel bonding are presented in this collapsible section. Click the title to expand the section.

- **Enable and select number of sequences to use.** Select whether to enable receiver channel bonding, and if enabled, how many channel bonding sequences to use. Possible options are No channel bonding, 1, or 2 sequences, but available selections can be limited by the number of enabled channels, receiver data decoding, receiver internal data width, and receiver elastic buffer usage.
- **Length of each sequence.** When channel bonding is used, select the length of each channel bonding sequence. Options are 1, 2, and 4 patterns.
- **Sequence maximum skew.** When channel bonding is used, select a channel bonding maximum skew value that is less than half the minimum distance between instances of the channel bonding sequence. Options are 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, and 14 characters.
- **Maximum channel bonding level to be used.** When channel bonding is used, select the maximum channel bonding level that will be used in the channel bonding topology of the system. Options are 1, 2, 3, 4, 5, 6, and 7 levels.
- **Don't care.** For each pattern of each enabled channel bonding sequence, mark the checkbox to indicate that it should be treated as a “don’t care” and always considered as a match within a channel bonding sequence.
- **Value.** For each pattern of each enabled channel bonding sequence, specify its bit value.
- **K character.** For each pattern of each enabled channel bonding sequence, mark the checkbox to indicate that it is a K character.
- **Inverted disparity.** For each pattern of each enabled channel bonding sequence, mark the checkbox to indicate that it uses inverted disparity to signify control of a character by deliberate error.

Receiver Clock Correction Section

Various customization options relating to receiver clock correction are presented in this collapsible section. Click the title to expand the section. Note that, if `RXOUTCLK` source is selected as `RXOUTCLKPMA` which in turn drives `RXUSRCLK/RXUSRCLK2`, then clock correction would do nothing. For more information, see the *UltraScale Architecture GTH*

Transceivers User Guide (UG576) [Ref 1] or the *UltraScale Architecture GTY Transceivers User Guide* (UG578) [Ref 2].

- **Enable and select number of sequences to use.** Select whether to enable receiver clock correction, and if enabled, how many clock correction sequences to use. Possible options are No clock correction, 1, or 2 sequences, but available selections can be limited by receiver data decoding, receiver internal data width, and receiver elastic buffer usage.
- **Length of each sequence.** When clock correction is used, select the length of each clock correction sequence. Options are 1, 2, and 4 patterns.
- **Don't care.** For each pattern of each enabled clock correction sequence, mark the checkbox to indicate that it should be treated as a "don't care" and always considered as a match within a clock correction sequence.
- **Value.** For each pattern of each enabled clock correction sequence, specify its bit value.
- **K character.** For each pattern of each enabled clock correction sequence, mark the checkbox to indicate that it is a K character.
- **Inverted disparity.** For each pattern of each enabled clock correction sequence, mark the checkbox to indicate that it uses inverted disparity to signify a control character by deliberate error.
- **Periodicity of the sequence (in bytes).** Specify the separation between clock correction sequences, in bytes.
- **Keep idle.** Specify whether at least one clock correction sequence is kept in the data stream for every continuous stream of clock correction sequences received. Options are Enable or Disable.
- **Precedence.** Specify whether clock correction takes precedence over channel bonding when both operations are triggered at the same time. Options are Enable or Disable.
- **Minimum repetition.** Specify the number of `RXUSRCLK` cycles following a clock correction during which the elastic buffer is not permitted to execute another clock correction. Legal options are 0 (for no limit), or 1 to 31 cycles.

Buffer Control Section

Various customization options relating to transmitter and receiver elastic buffer control and behaviors are presented in this collapsible section. Click the title to expand the section.

- **Receiver elastic buffer bypass mode.** When the receiver elastic buffer is bypassed and two or more transceiver channels are enabled, specify whether multi-lane buffer bypass mode or single-lane buffer bypass mode is used. When multi-lane mode is selected, the designated receiver master channel acts as the buffer bypass master lane and clock source for the one receiver user clocking network helper block instance that provides user clocks to the master and all slave lanes. When single-lane mode is selected, each transceiver channel receiver elastic buffer is bypassed individually, resulting in an

instance of the receiver buffer bypass controller helper block and an instance of the receiver user clocking network helper block for each transceiver channel. The default is multi-lane mode.

- **Reset receiver elastic buffer on channel bonding change.** Specify whether the receiver elastic buffer is reset on change to `RXCHANBONDMASTER`, `RXCHANBONDSLAVE`, or `RXCHANBONDLEVEL`. When the receiver elastic buffer is used, options are Enable or Disable.
- **Reset receiver elastic buffer on comma alignment.** Specify whether the receiver elastic buffer is reset on comma alignment. When the receiver elastic buffer is used, options are Enable or Disable.
- **Reset receiver elastic buffer on rate change.** Specify whether the receiver elastic buffer is reset on rate change. When the receiver elastic buffer is used, options are Enable or Disable.
- **Reset transmitter buffer on rate change.** Specify whether the transmitter buffer is reset on rate change. When the transmitter buffer is used, options are Enable or Disable.

Advanced Clocking Section

Various customization options relating to advanced clocking methodologies and frequencies are present in this collapsible section. Click the title to expand the section.

- **Enable secondary QPLL.** When either QPLL0 or QPLL1 (but not both) are used to clock the transmitter and/or receiver, the remaining QPLL is unused in the core as configured. Enable this option to allow customization of the secondary QPLL in the transceiver common for use in a different core. Refer to Transceiver Common Primitive, page 77 for transceiver common sharing approaches.
- **Line rate of second core (Gb/s).** Enter the transmitter and/or receiver line rate, in gigabits per second, for the core that will utilize the secondary QPLL instantiated by this core. The available range depends on transceiver type and can be limited by the selected device.
- **QPLL Fractional-N options**. For configurations targeting a secondary QPLL in a device which supports the fractional-N feedback divider, enter a **Requested reference clock (MHz)** value and click the **Calc** button. This action populates the **Actual reference clock (MHz)** field with a variety of supported reference clock frequencies based on the requested value. In most cases, the requested frequency is available for selection. The calculation also populates the **Fractional part of QPLL feedback divider** field with the numerator of the fractional part of the QPLL feedback divider used to clock the secondary QPLL-driven datapath. This value can be manually tweaked to fine-tune the available **Actual reference clock (MHz)** selections for advanced use cases. Possible values are 0 through 16777215, where 0 disables fractional-N operation. Changing this field updates the available **Actual reference clock (MHz)** selections.

- **Actual reference clock frequency (MHz).** Select the desired frequency from among all compatible frequencies for the reference clock that will be provided to the secondary QPLL to achieve the selected transmitter and/or line rate used in the second core.
- **Enable selectable TXOUTCLK frequency.** When the TX programmable divider (TXPROGDIVCLK) is selected as the TXOUTCLK source, it might be possible to choose a non-default frequency for that clock, or to choose a different clock source for the TX programmable divider. Enable this option to select from among the available choices that are compatible with the core as configured.
- **Programmable divider clock source.** Select the PLL clock source for the TX programmable divider. Options include the PLL type chosen for the transmitter, and the CPLL if certain frequency relationships are met.

  Note: If a PLL type is selected that is different than the PLL type chosen for the transmitter, you must take care to properly reset and ensure a locked TX programmable divider clock source in coordination with the remainder of the core and system reset sequencing.
- **TXOUTCLK frequency (MHz).** Select from among the TXOUTCLK frequencies that can be generated by the TX programmable divider and are compatible with the core configuration and selected device. Options are divided to the required user clock frequencies by the transmitter user clocking network helper block.

SATA Section

Customization options relating to SATA configurations are present in this collapsible section. Click the title to expand the section.

- **TX COM sequence burst length.** Select the number of bursts that make up a SATA COM sequence. Options are 6, 7, 8, 9, 10, 11, 12, 13, 14, and 15.

Structural Options Tab

IP customization options in the Structural Options tab are described in the following subsections. The layout of the Structural Options tab is shown in Figure 4-4.

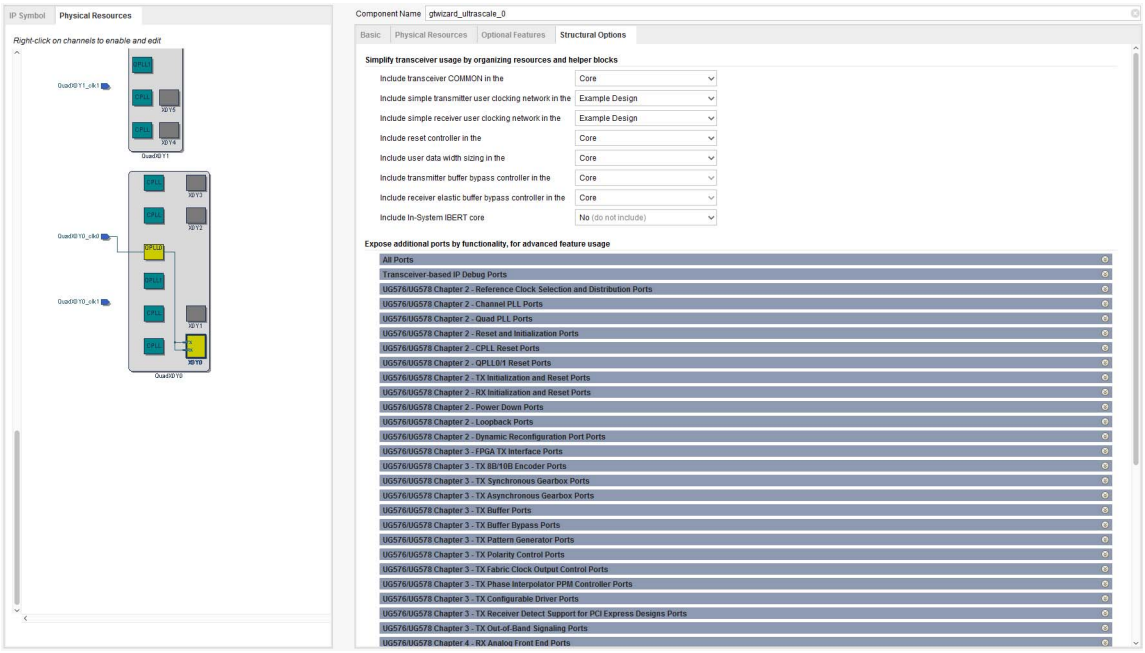

Figure 4-4: **Structural Options Tab**

Helper Block Location Frame

This frame is labeled as "Simplify transceiver usage by organizing resources and helper blocks" in the Customize IP dialog box. The location of the transceiver common primitive and each available helper block can be selected here. See Chapter 3, Designing with the Core, for guidance on the usage of each helper block and trade-offs to consider when deciding where to locate the transceiver common and each available helper block.

- **Include the transceiver COMMON in the...** Specify whether enabled transceiver common primitives are instantiated in the core or in the example design. The default is in the core.
- **Include simple transmitter user clocking network in the...** Specify whether the transmitter user clocking network helper block is instantiated in the core or in the example design. The default is in the example design.
- **Include simple receiver user clocking network in the...** Specify whether the receiver user clocking network helper block is instantiated in the core or in the example design. The default is in the example design.
- **Include reset controller in the...** Specify whether the reset controller helper block is instantiated in the core or in the example design. The default is in the core.
- **Include user width data sizing in the...** Specify whether the user data width sizing helper block is instantiated in the core or in the example design. The default is in the core.

- **Include transmitter buffer bypass controller in the...** If the transmitter buffer is bypassed, specify whether the transmitter buffer bypass controller helper block is instantiated in the core or in the example design. The default is in the core.
- **Include receiver elastic buffer bypass controller in the...** If the receiver elastic buffer is bypassed, specify whether the receiver buffer bypass controller helper block is instantiated in the core or in the example design. The default is in the core.
- **Include In-System IBERT core...** Specify the optional inclusion of In-System IBERT core in the wizard example design. The default is No (do not include).

Optional Port Enablement Interface

This frame is labeled as "Expose additional ports by Functionality, for advanced feature usage" in the Customize IP dialog box. The optional port enablement interface allows additional transceiver channel and transceiver common primitive ports to be exposed on the core interface. See the *UltraScale Architecture GTH Transceivers User Guide* (UG576) [Ref 1] or *UltraScale Architecture GTY Transceivers User Guide* (UG578) [Ref 2] for details on each available transceiver primitive port.

- **All Ports.** All wizard ports are presented for optional enablement in this highlighted, collapsible section. Click the title to expand the section and display the All Ports table. Ordering is alphabetical by port name, and organized according to port direction. The Name column of the table shows the Wizard IP core interface name for each port, while the Information column indicates the transceiver primitive type and mapping of that port (for non-helper block ports). Mark the checkbox for a given port in the Enable column to expose that port on the core interface. The IP symbol is updated to reflect the enabled ports. The search field can be used to search text within the All Ports collapsible section. Port enablement restrictions of the All Ports table are as follows:
  - Helper block port enablement cannot be directly controlled and is a function of helper block availability and location only.
  - Transceiver primitive input ports that are driven within the core instance, usually as a result of locating a helper block within the core, cannot be enabled.
  - Ports corresponding to transceiver common primitives cannot be enabled if the core instance does not instantiate any transceiver common primitives.
  - Ports unique to transceiver types that are different from the selected transceiver type cannot be enabled.
- **Other groups.** Collapsible groups other than All Ports categorize the wizard ports according to specific transceiver functionality. Click the title of a group to expand its section. To easily identify and enable the ports required for your application, groups are organized and named according to chapters within the *UltraScale Architecture GTH Transceivers User Guide* (UG576) [Ref 1] or *UltraScale Architecture GTY Transceivers User Guide* (UG578) [Ref 2]. In addition, a Transceiver-based IP Debug Ports group is provided to organize frequently used debug ports. Mark the checkbox for a given port

to expose that port on the core interface. The IP symbol is updated to reflect the enabled ports. Port enablement restrictions in other groups are as follows:

  - Transceiver primitive input ports that are driven within the core instance, usually as a result of locating a helper block within the core, cannot be enabled.
  - Ports corresponding to transceiver common primitives cannot be enabled if the core instance does not instantiate any transceiver common primitives.
  - Ports unique to transceiver types that are different from the selected transceiver type cannot be enabled.

- `ENABLE_COMMON_USRCLK`:
  - Default value =>0
  - Setting 0 => `TXUSRCLK/TXUSRCLK2` source is `TXOUTCLK` and `RXUSRCLK/RXUSRCLK2` source is `RXOUTCLK`
  - Setting 1 => `TXUSRCLK/TXUSRCLK2/RXUSRCLK/RXUSRCLK2` source is `RXOUTCLK`
  - Setting 2 => `TXUSRCLK/TXUSRCLK2/RXUSRCLK/RXUSRCLK2` source is `TXOUTCLK`
- This advanced option is valid only if both the TX and the RX clocking helper blocks are configured as being inside the CORE option, and the BufferBypass controllers are also inside the CORE. If any of these helper blocks are configured as EXAMPLE_DESIGN, this parameter takes a default value of 0. In cases where clocking helper blocks are outside the Wizard CORE, you could manually instantiate TX alone or RX alone as per your clocking requirement.

Constraining the Core

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

Required Constraints

Core-level Constraints

Each instance of the UltraScale FPGAs Transceivers Wizard core includes a core-level Xilinx design constraints (XDC) file customized for that instance. The core-level XDC file contains:

- **Transceiver location constraints** that reflect the transceiver primitive site locations selected during customization of the Physical Resources tab of the Customize IP dialog box.
- **Case analysis constraints**, as necessary, that direct the Vivado design tools to propagate the correct constraint for the operational TXOUTCLK frequency.

- **False path constraints**, as necessary, if synchronizer modules or other false paths are included in the core.

The constraints provided in the core-level XDC file are required for proper operation of the core instance. This file is managed by the Vivado design tools and can change to reflect core customization changes or core version upgrades. Do not modify this file. For advanced use cases, if you want to manage GT locations manually, set `DISABLE_LOC_XDC` user parameter to '1' during IP customization.

Note: GT parent IP could be driving the value of this user parameter as part of the hierarchical instantiation, and using this parameter may not be always possible in a locked IP scenario. If you need to manage the GT locations, Xilinx recommends that you configure the GT parent IP as GT outside and then customize the UltraScale GT wizard IP based on your design requirement.

Example Design Constraints

When an example design is generated for an instance of the Wizard IP core, an XDC file is generated for that example project. The example design XDC file contains the required top-level constraints for the full design, which include:

- **I/O location constraints** for each instantiated transceiver differential reference clock buffer.
- **Placeholder I/O location constraints**, which serve as commented templates to constrain the location of top-level I/O as appropriate for your system.
- **System-level clock period constraints** on the free-running clock used for system bring-up, and on the transceiver reference clocks differential inputs.
- **False path constraints** for synchronizer modules or other false paths that are included in the example design.

The example design XDC is necessary to properly constrain elements within the example design, but it can also be used as a starting point for the development of your system-level constraints. The constraints in the example design XDC do not overlap with those in the core-level XDC.

Out-of-Context Constraints

When an out-of-context (OOC) design flow such as OOC synthesis or hierarchical design is used, the Wizard also uses a special OOC XDC file customized for that instance. The OOC XDC file provides default period constraints on clock ports that would otherwise be constrained by the example design XDC file. This file is managed by the Vivado design tools and can change to reflect core customization changes or core version upgrades. Do not modify this file.

Device, Package, and Speed Grade Selections

The core and example design constraints generated for a given wizard instance reflect the choices made during IP customization for the selected device. If you wish to use a different device, package, or speed grade, use the Vivado IDE to select the desired part and re-customize the core rather than modifying an XDC file.

Clock Frequencies

The example design XDC file creates period constraints for the clock inputs that drive the dedicated differential reference clock buffers, which in turn drive the various PLL resources in the core and propagate to the `TXOUTCLK` and `RXOUTCLK` pins of each transceiver channel primitive. Where a transceiver channel drives a user clocking network helper block, those derived constraints then propagate through the helper block resources to constrain synchronous paths at the relevant clock frequencies of that user clock network. The correct clock period for each constraint is automatically determined by the wizard. For example, when using a single reference clock buffer in the X0Y0 grid position, a command similar to the following will exist in the example design XDC file:

```
create_clock -period 6.400 [get_ports mgtrefclk0_x0y0_p]
```

IMPORTANT: *It is important to retain the create_clock command that exists in the example design XDC and is applied to the input of the differential reference clock buffers. This constraint is used to derive the user clocking network constraints.*

For GTH transceiver configurations targeting engineering sample (ES1 or ES2) UltraScale devices, and in which the CPLL is used as the PLL type for a given data direction or as the source of the selectable TXOUTCLK frequency, `set_case_analysis` commands exist within the core-level XDC. For example:

```
set_case_analysis 0 [get_pins -hierarchical -filter {NAME =~
*gen_channel_container[0].*gen_gthe3_channel_inst[0].GTHE3_CHANNEL_PRIM_INST/
TXOUTCLKSEL[2]}]

set_case_analysis 1 [get_pins -hierarchical -filter {NAME =~
*gen_channel_container[0].*gen_gthe3_channel_inst[0].GTHE3_CHANNEL_PRIM_INST/
TXOUTCLKSEL[1]}]

set_case_analysis 0 [get_pins -hierarchical -filter {NAME =~
*gen_channel_container[0].*gen_gthe3_channel_inst[0].GTHE3_CHANNEL_PRIM_INST/
TXOUTCLKSEL[0]}]
```

Using the frequency provided during core customization, the example design XDC file creates a period constraint for the free-running clock used for system bring-up. For example:

```
create_clock -period 10 [get_ports hb_gtwiz_reset_clk_freerun_in]
```

Additional clock period constraints can be necessary when integrating the Wizard IP core into your system. For example:

- If you enable any optional clock ports on the Wizard IP core interface (for example, the DRP clock), you must appropriately constrain those clocks.

  Note: As described in Chapter 4, Customizing and Generating the Core, the free-running clock must also be used for the transceiver channel DRP interface clock in GTH transceiver configurations targeting architecture engineering sample (ES1 or ES2) UltraScale devices that use the CPLL.

If you choose not to use the provided user clocking network helper blocks and thus break the path from `TXOUTCLK` or `RXOUTCLK` through the appropriate user clocking network helper blocks and to the appropriate transceiver user clocks, you must appropriately constrain the new source of the transceiver user clocks.

Clock Management

This section is not applicable for this IP core.

Clock Placement

The example design XDC file creates package pin constraints for each instantiated transceiver differential reference clock buffer primitive as well as each instantiated differential recovered clock output buffer primitive, if utilized. The constraints reflect the transceiver primitive site locations selected during customization of the Physical Resources tab of the Customize IP dialog box. If you wish to use different physical locations, the correct way to adjust both the location constraints and the wiring between clock buffers, transceiver channel and common primitives is to re-customize the core and select different clock buffer locations, rather than modify the example design XDC file.

Within the example design XDC, two `set_property package_pin` commands exist per transceiver differential reference clock buffer in the following format. The specific locations and ports are examples only:

```
set_property package_pin Y5 [get_ports mgtrefclk0_x0y0_n]
set_property package_pin Y6 [get_ports mgtrefclk0_x0y0_p]
```

Within the example design XDC, two `set_property package_pin` commands exist per transceiver differential recovered clock output buffer in the following format. The specific locations and ports are examples only:

```
set_property package_pin T5 [get_ports rxrecclkout_chx0y4_n]
set_property package_pin T6 [get_ports rxrecclkout_chx0y4_p]
```

Banking

This section is not applicable for this IP core. For transceiver channel primitive location constraints, see Transceiver Placement.

Transceiver Placement

The core-level XDC file creates a location constraint for each enabled transceiver channel primitive. The constraints reflect the transceiver primitive site locations selected during customization of the Physical Resources tab of the Customize IP dialog box. To use different physical locations, re-customize the core and choose different transceiver channel primitive locations rather than modifying the core-level XDC file.

Within the core-level XDC, one set_property LOC command exists per transceiver channel in the following format. The specific hierarchical path and location are examples only:

```
set_property LOC GTHE3_CHANNEL_X0Y0 [get_cells -hierarchical -filter {NAME =~
*gen_channel_container[0].*gen_gthe3_channel_inst[0].GTHE3_CHANNEL_PRIM_INST}]
```

I/O Standard and Placement

The example design XDC file creates the following placeholder constraints for general example design top-level I/O. The set_property package_pin and set_property iostandard constraints should be uncommented and assigned to package pins and I/O standards, (replacing "<>") respectively, that are appropriate for your system:

```
#set_property package_pin <> [get_ports hb_gtwiz_reset_clk_freerun_in]
#set_property iostandard  <> [get_ports hb_gtwiz_reset_clk_freerun_in]

#set_property package_pin <> [get_ports hb_gtwiz_reset_all_in]
#set_property iostandard  <> [get_ports hb_gtwiz_reset_all_in]

#set_property package_pin <> [get_ports link_down_latched_reset_in]
#set_property iostandard  <> [get_ports link_down_latched_reset_in]

#set_property package_pin <> [get_ports link_status_out]
#set_property iostandard  <> [get_ports link_status_out]
```

Other Constraints

Depending on IP customization choices including helper block locations, synchronizers can be instantiated within the core or the example design in order to facilitate clock domain crossing of individual signals. When synchronizers or other ignorable asynchronous paths are present, false path constraints are included in the core-level XDC file or in the example design XDC file (as appropriate) for those arbitrary latency paths. Some possible commands for each XDC file are as follows:

```
set_false_path -to [get_cells -hierarchical -filter {NAME =~ *bit_synchronizer*inst/
i_in_meta_reg}]

set_false_path -to [get_cells -hierarchical -filter {NAME =~
*reset_synchronizer*inst/rst_in_*_reg}

set_false_path -to [get_cells -hierarchical -filter {NAME =~ *gtwiz_userclk_tx_inst/
*gtwiz_userclk_tx_active_*_reg}]
```

```
set_false_path -to [get_cells -hierarchical -filter {NAME =~ *gtwiz_userclk_rx_inst/
*gtwiz_userclk_rx_active_*_reg}]
```

Simulation

The wizard example design test bench can be simulated to quickly demonstrate core and transceiver functionality. For more information, see Chapter 6, Test Bench.

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 10].

Synthesis and Implementation

The wizard example design can be synthesized and implemented to quickly demonstrate core and transceiver functionality in hardware. For more information, see Chapter 5, Example Design.

For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 8].

Example Design

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

Purpose of the Example Design

An example design can be generated for any customization of the UltraScale™ FPGAs Transceivers Wizard IP core. After you customize and generate a core instance, choose the **Open IP Example Design** Vivado Integrated Design Environment (IDE) option for that instance. A separate Vivado project opens with the wizard example design as the top-level module. The example design instantiates the customized core.

The purpose of the Wizard IP example design is to:

- **Provide a simple demonstration** of the customized core instance operating in simulation or in hardware through the use of a link status indicator based on PRBS generators and checkers.
- **Provide a starting point for integrating** the customized core into your system, including reference clock buffers and example system-level constraints.
- **Simplify hardware bring-up and debug** through the inclusion of a virtual input/output (VIO) core instance that probes key debug signals and can be re-customized as needed.
- **Provide a variety of convenience features** including instantiation and use of helper blocks that were not located in the core, and per-channel vector-slicing.

The example design contains configurable PRBS generator and checker modules per transceiver channel that enable simple data integrity testing, and resulting link status reporting. As described in Chapter 6, Test Bench, an included self-checking test bench simulates the example design in loopback, checking for link maintenance. The example design is also synthesizable so it can be used to check for data integrity and resulting link in hardware, either through loopback or connection to a suitable link partner. A VIO core instance probes key status signals, drives basic control signals, and reduces reliance on hardware I/O interaction.

RECOMMENDED: *As the primary means of demonstrating the customized core, Xilinx recommends that you use the example design to familiarize yourself with the basic usage and behavior of the Wizard IP core.*

Hierarchy and Structure

The hierarchy and simplified representation of the structure of the Wizard IP example design is illustrated in Figure 5-1.

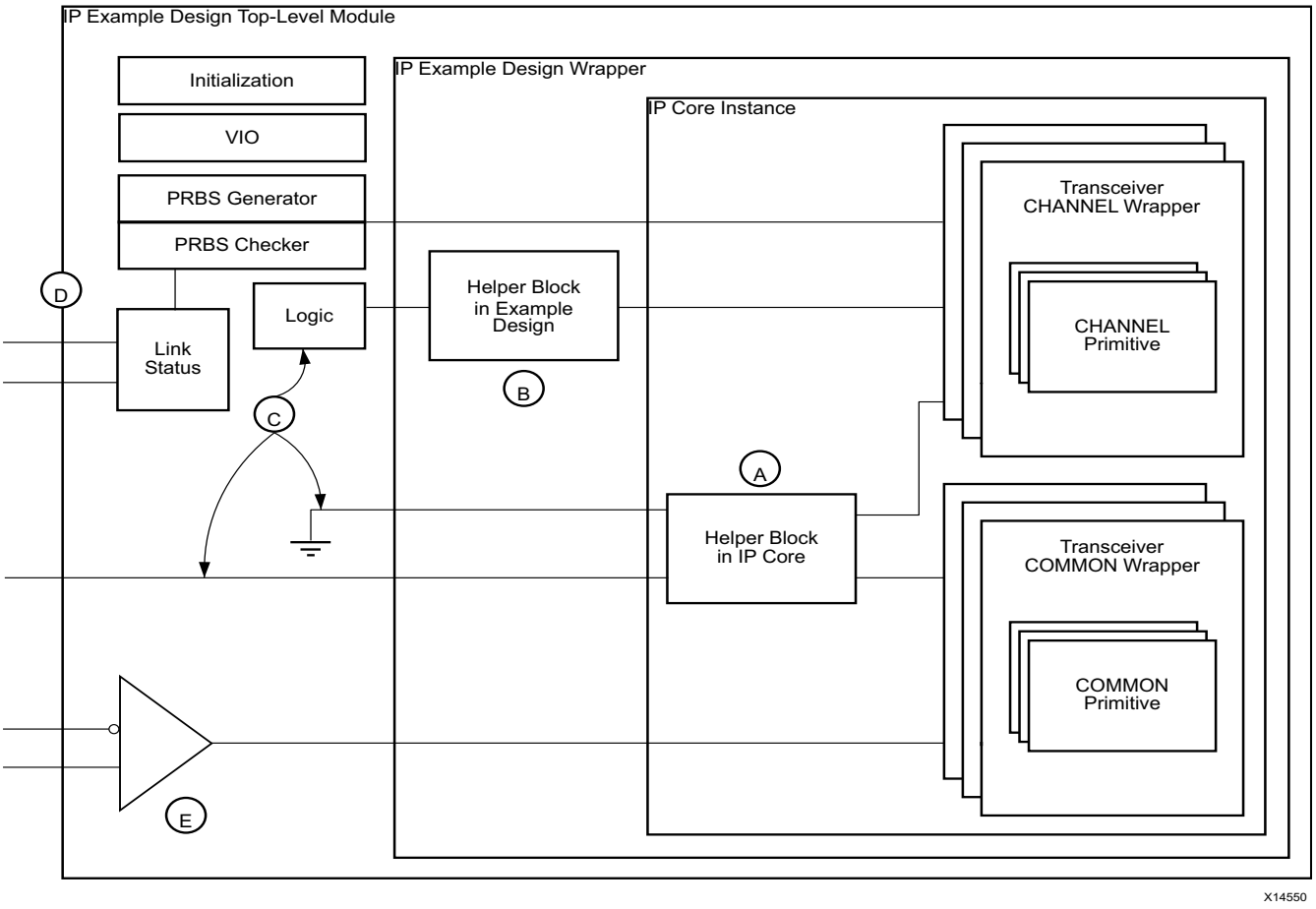

Figure 5-1: **Wizard Example Design Block Diagram**

The example design instantiates the customized core instance. The core instance contains one or more transceiver channel primitives and, depending on customization choices, can instantiate one or more transceiver common primitives and one or more helper blocks. Portion **A** of the figure represents a helper block included within the core instance.

The lowest level of example design hierarchy is the example design wrapper. The purpose of the example design wrapper is to instantiate only the customized core and any helper blocks that you included in the example design. By including only those resources and no additional demonstration logic, the example design wrapper can be integrated in your project with no or minimal modification required. Portion **B** of the figure represents a helper block instantiated within the example design wrapper. Enabled ports of the core instance that do not directly interface to helper block **B** are routed through to the next level of example design hierarchy.

The example design top-level module instantiates the example design wrapper and is the top-level module of the Vivado IP example project. The top-level module serves many purposes. Portion **C** of the figure represents the different ways that ports enabled on the core and exposed through the example design wrapper are handled in the example design top-level module. Ports that were optionally enabled or otherwise enabled but are not directly used in the example design are tied off to their appropriate values (per the core customization). A small number of ports that must be driven internally to the example design for purposes of example design operation are driven by some logic function to produce the necessary value. A small number of ports are also connected to the top-level example design I/O.

An example stimulus module and an example checking module are each instantiated for each transceiver channel instance. As shown in portion **D** of the figure, these modules include a PRBS block that is customized for data generation or checking as appropriate, as well as a minimal amount of additional logic sufficient to interface to the selected transmitter data encoding and receiver data decoding formats of the transceiver channels.

Note: The example stimulus and example checking modules fundamentally transmit and check raw PRBS data, and do not implement higher-level protocols to encapsulate that raw data.

Because independent example stimulus and checking modules exist per channel, an aggregate pattern match status across all modules is needed. A small amount of logic in the example design top-level module combines the match status from all example checking modules into a single “match all” signal. This signal is then used in a simple link status state machine to indicate the health of the PRBS checker-based link while remaining resilient to occasional bit errors. A sticky link down indicator is set any time the link status signal deasserts, and an accompanying reset signal clears that sticky link down indicator. Together, these three signals are sufficient to monitor data integrity-based link status across all transceiver channels in an abstracted fashion, including mapping the top-level I/O to two LEDs and one pushbutton on a PCB. To reduce reliance on hardware I/O interaction and simplify example design bring-up and debug, a VIO core instance also connects to these top-level signals and other key control and status signals. The VIO core can be re-customized and connected to other signals as needed.

Additionally, an initialization state machine provides demonstration logic that interacts with and enhances the reset controller helper block to assist with successful system bring-up. See Link Status and Initialization, page 107 for more details on these features.

As shown in portion **E** of the figure, a dedicated transceiver reference clock differential buffer (IBUFDS_GTE3 or IBUFDS_GTE4 primitive) is instantiated for each reference clock source that was specified in the Physical Resources tab of the Customize IP dialog box. Differential clock input ports drive each buffer instance, which in turn is wired to all the transceiver channel or transceiver common primitives it was specified to drive. If you wish to use different connectivity in your system, re-customize the core and select different transceiver reference clock locations in the Customize IP dialog box, rather than modifying the clock connectivity, to properly adjust both the wiring and the transceiver primitive location constraints.

The ports shown in Table 5-1 are present on the example design top-level module, and are therefore package pins in the example project.

Table 5-1: **Example Design Top-Level Ports**

Name	Direction	Width	Clock Domain	Description
mgtrefclk<i>_<j>_p	Input	1		Positive and negative inputs of the differential reference clock where: <i>: 0 (corresponds to MGTREFCLK0 source) or 1 (corresponds to MGTREFCLK1 source) <j>: The coordinate of the transceiver common primitive where the IBUFDS_GTE3 or IBUFDS_GTE4 instance resides. For example, the input “mgtrefclk0_x0y0_p” is the positive clock input of MGTREFCLK0 within the Quad that contains the transceiver common at the X0Y0 coordinate.
mgtrefclk<i>_<j>_n	Input	1		
rxrecclkout_ch<j>_p	Output	1		Positive and negative outputs of the differential recovered clock where: <j>: The coordinate of the transceiver channel primitive that drives the OBUFDS_GTE3 or OBUFDS_GTE4 primitive which produces the differential outputs. For example, the output "rxrecclkout_chx0y4_n" is the negative clock output of the OBUFDS_GTE3 or OBUFDS_GTE4 within the Quad that contains the transceiver channel primitive at the X0Y4 coordinate.
rxrecclkout_ch<j>_n	Output	1		
ch<i>_gt[h|y]rxn_in	Input	1	Serial	Positive and negative inputs of the transceiver channel differential serial data receiver, where: <i>: Corresponds to the index of the transceiver channel among all enabled transceiver channels in the core.
ch<i>_gt[h|y]rxp_in	Input	1	Serial	
ch<i>_gt[h|y]txn_out	Output	1	Serial	Positive and negative outputs of the transceiver channel differential serial data transmitter, where: <i>: Corresponds to the index of the transceiver channel among all enabled transceiver channels in the core.
ch<i>_gt[h|y]txp_out	Output	1	Serial	

Table 5-1: **Example Design Top-Level Ports *(Cont'd)***

Name	Direction	Width	Clock Domain	Description
hb_gtwiz_reset_clk_freerun_in	Input	1		Free-running clock, used by the example design and reset controller helper block for various system bring-up tasks. The example design top-level module globally buffers this single-ended clock input. ***Note:*** To alternatively use a differential clock input: - add a second input port, - instantiate an IBUFDS primitive driven by both the existing hb_gtwiz_reset_clk_freerun_in and that new port, and - drive the input of the existing BUFG primitive with the output of that IBUFDS primitive instead of the hb_gtwiz_reset_clk_freerun_in port.
hb_gtwiz_reset_all_in	Input	1	Async	Falling edge-triggered, active-High "reset all" input used by the reset controller helper block to initiate a full system reset sequence. Assumed to be de-bounced external to the device.
link_down_latched_reset_in	Input	1	Async	Active-High signal used to reset the sticky link down indicator. Assumed to be de-bounced external to the device.
link_status_out	Output	1	hb_gtwiz_reset_clk_freerun_in	Active-High, live indicator of link status based on combined PRBS match status across all example checking modules.
link_down_latched_out	Output	1	hb_gtwiz_reset_clk_freerun_in	Active-High, sticky link down indicator. Set when link_status_out is Low and cleared when link_down_latched_reset_in is High.

Link Status and Initialization

The wizard example design contains link status logic that indicates the current state of the PRBS checkers across all transceiver channels while remaining tolerant of occasional mismatches such as infrequent bit errors. The example design also includes an initialization module that is a demonstration of how logic can be constructed to interact with and enhance the reset controller helper block to assist with successful system bring-up. Together, the link status logic and the initialization module provide a robust demonstration of example design system bring-up, and work in coordination to both indicate link status and regain the link if it is lost.

Link Status Logic

The wizard example design instantiates an independent PRBS data checker module for each enabled transceiver channel. The combined and synchronized match signal is used by the link status logic, which produces a link status indicator using a simple state machine within the wizard example design. To best represent the link health of the example design system, the link status indicator follows the combined PRBS match value but is resilient to occasional mismatches such as infrequent bit errors.

The link status state machine uses a leaky bucket algorithm to accumulate multiple consecutive clock cycles of combined PRBS matches, incrementing a link counter to its prescribed maximum before reporting that the link is up (indicated by `link_status_out` = 1). After the link is up, any PRBS mismatches cause a more rapid decrease in the link counter, such that bursts of mismatches or independent mismatches in close proximity quickly reduce the link counter to its prescribed minimum where the link is reported as down (indicated by `link_status_out` = 0). The logic operates continually, and therefore automatically attempts to recover from transient mismatches or regain link upon its loss. Figure 5-2 illustrates the behavior of the link counter and resulting link status in response to various PRBS checker conditions.

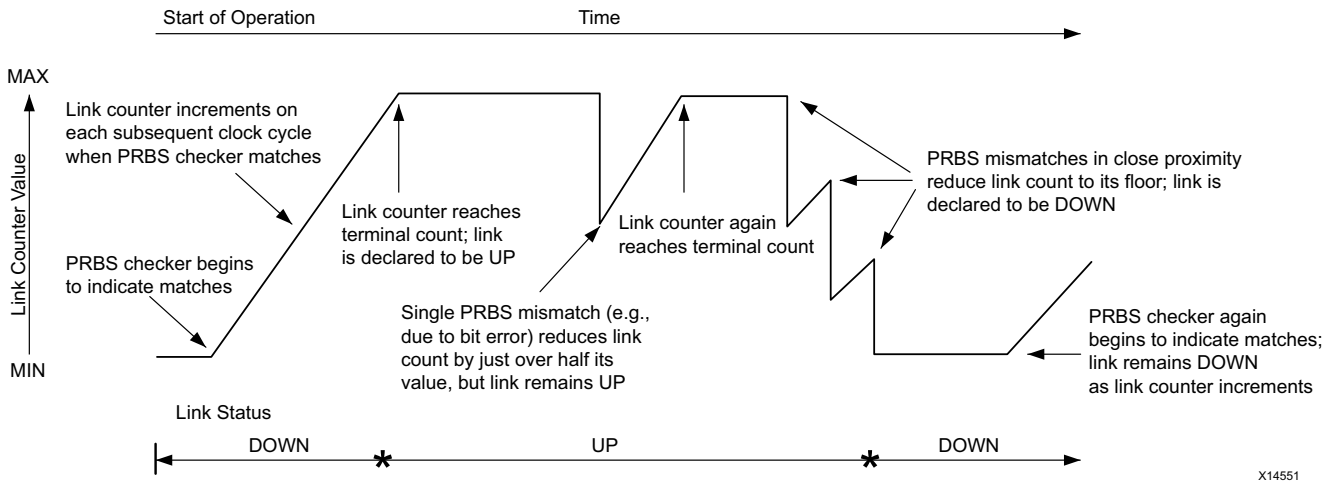

Figure 5-2: **Link Counter and Link Status in Response to Various PRBS Checker Conditions**

Whenever the link is down, including at the start of operation, the sticky link down indicator `link_down_latched_out` is set to 1. It can only be reset by assertion of the `link_down_latched_reset_in` input.

A simple use of the link status interface in hardware is to map `link_status_out` and `link_down_latched_out` to active-High LEDs, and `link_down_latched_reset_in` to an active-High pushbutton. The `link_status_out` LED gives a rough estimation of link behavior, while even momentary loss of link is visible due to the sticky behavior of `link_down_latched_out` lighting the LED. The `link_down_latched_reset_in` pushbutton clears `link_down_latched_out`, turning off its LED as long as the link

remains up. These link status interface signals are also connected to the VIO core instance by default.

Initialization Module

The wizard example design contains a module that demonstrates how initialization logic can be constructed to interact with and enhance the reset controller helper block to assist with successful system bring-up. The example initialization logic monitors for timely transceiver resource reset completion, retrying appropriate resets as necessary to mitigate problems with system bring-up such as clock or data connection readiness. It also optionally monitors data quality after the system is operational, resetting the receiver if the data is not considered to be "good." The initialization module is an example and can be modified as necessary to suit your needs.

The example initialization module is implemented as a finite state machine that is activated with the first user-provided "reset all" pulse following device configuration. The module first monitors for timely completion of the transmitter PLL and datapath transceiver resources, pulsing an internal "reset all" signal to the reset controller helper block in the event that the transmitter resets do not complete in a reasonable time. Upon transmitter reset completion, the example initialization module similarly waits for timely completion of receiver PLL and datapath transceiver resources, pulsing an internal receiver PLL and datapath reset (or receiver datapath reset if a single PLL is used for both data directions) to the reset controller helper block in the event that the receiver resets do not complete in a reasonable time. For debug purposes, each reset assertion increments a retry counter up to a specified saturation point, and the retry counter is only cleared upon device configuration. The initialization done and retry counter signals are connected to the VIO core instance by default.

The example initialization module also contains a receive data good input. If an active-High indication of data quality drives this port, the initialization module automatically pulses the appropriate receiver reset to the reset controller helper block if the design has been successfully initialized but the receiver data good input is Low. In this way, the initialization module repeatedly attempts to re-establish good data reception in the event of its loss; for example, due to cable pull effects on the receiver. Figure 5-3 illustrates the initialization module state machine.

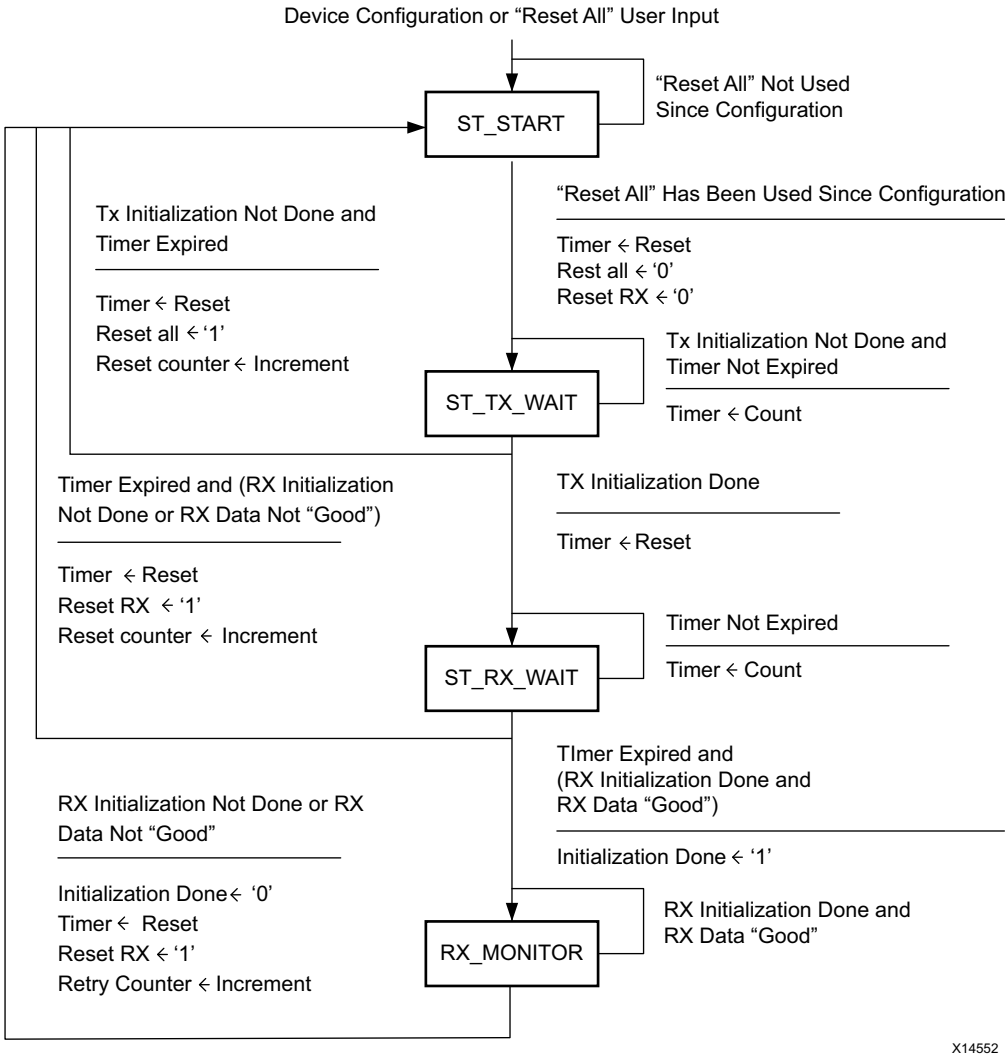

Figure 5-3: **Example Initialization Module Finite State Machine**

In the example design as delivered, the link status indicator signal directly drives the initialization module’s receive data good input port. Therefore, any loss of link causes repeated receiver reset attempts until the link is again established. This approach is useful for demonstrating link robustness in the face of system disruptions such as cable pull tests. If it is not desired, this optional behavior can be disabled by simply tying the initialization module’s receive data good port High.

VIO Core Instance

To simplify example design hardware bring-up and debug, a VIO core instance is included in the example design top-level module. By default, the instance is adapted to the customized Wizard core to probe key status and debug signals, and to drive key control signals. It can be re-customized and connected to other signals as needed.

The default VIO core instance always probes the following:

- the link status indicator signals `link_status_out` and `link_down_latched_out`.
- the initialization module retry counter signals `init_done_int` and `init_retry_ctr_int`.
- synchronized versions of reset helper block signals `gtwiz_reset_tx_done_out` and `gtwiz_reset_rx_done_out`.

The VIO core always drives the following:

- internal versions (using a logical OR where appropriate) of link status indicator signal `link_down_latched_reset_in`.
- reset helper block signals `gtwiz_reset_all_in`, `gtwiz_reset_tx_pll_and_datapath_in`, `gtwiz_reset_tx_datapath_in`, `gtwiz_reset_rx_pll_and_datapath_in`, and `gtwiz_reset_rx_datapath_in`.

By interacting with these key signals, using the VIO core can reduce reliance on hardware I/O interaction and provide rapid insight into basic system behavior.

The default VIO core instance probes synchronized versions of the following signals if they are available in the example design top-level module for the customized Wizard core:

`gtpowergood_out`	`cplllock_out`
`qpll0lock_out`	`qpll1lock_out`
`txprgdivresetdone_out`	`rxprgdivresetdone_out`
`txpmaresetdone_out`	`rxpmaresetdone_out`
`gtwiz_buffbypass_tx_done_out`	`gtwiz_buffbypass_rx_done_out`
`gtwiz_buffbypass_tx_error_out`	`gtwiz_buffbypass_rx_error_out`
`rxelecidle_out`	`rxstatus_out`
`rxbufstatus_out`	`rxprbserr_out`
`rxprbslocked_out`	

TIP: *To probe these signals, use the optional ports interface during IP customization to expose the relevant ports through the IP core boundary.*

One VIO probe port is used per signal, and each signal is vectored across all enabled transceiver primitives.

The default VIO core instance drives the following signals if they are available in the example design top-level module for the customized Wizard core. Synchronizers are used where appropriate:

txpmareset_in	rxpmareset_in
txpcsreset_in	rxpcsreset_in
rxcdrreset_in	rxdfelpmreset_in
txelecidle_in	txpd_in
rxpd_in	txprecursor_in
txpostcursor_in	loopback_in
txprbssel_in	rxprbssel_in
txprbsforceerr_in	rxprbscntreset_in

TIP: *To interactively drive these signals, use the optional ports interface during IP customization to expose the relevant ports through the IP core boundary.*

One VIO probe port is used per signal, and each signal is vectored across all enabled transceiver primitives.

To probe additional wizard signals available in the example design top-level module, re-customize the VIO core instance to add further probe ports. For more details on using the VIO core and other Vivado Design Suite debug features, see *Vivado Design Suite User Guide: Programming and Debugging (UG908)* [Ref 12].

In-System IBERT Core Instance

In-System IBERT core instance is included in the example design top-level module optionally when you enable it through the GUI customization in Structural Options tab. This instance is included to simplify example design hardware bring-up, to help in GT tuning and get the eye scans by using the Serial I/O analyzer run time software. For more information on how to use this instance, see *In-System IBERT v1.0 LogiCORE Product Guide* (PG246) [Ref 13].

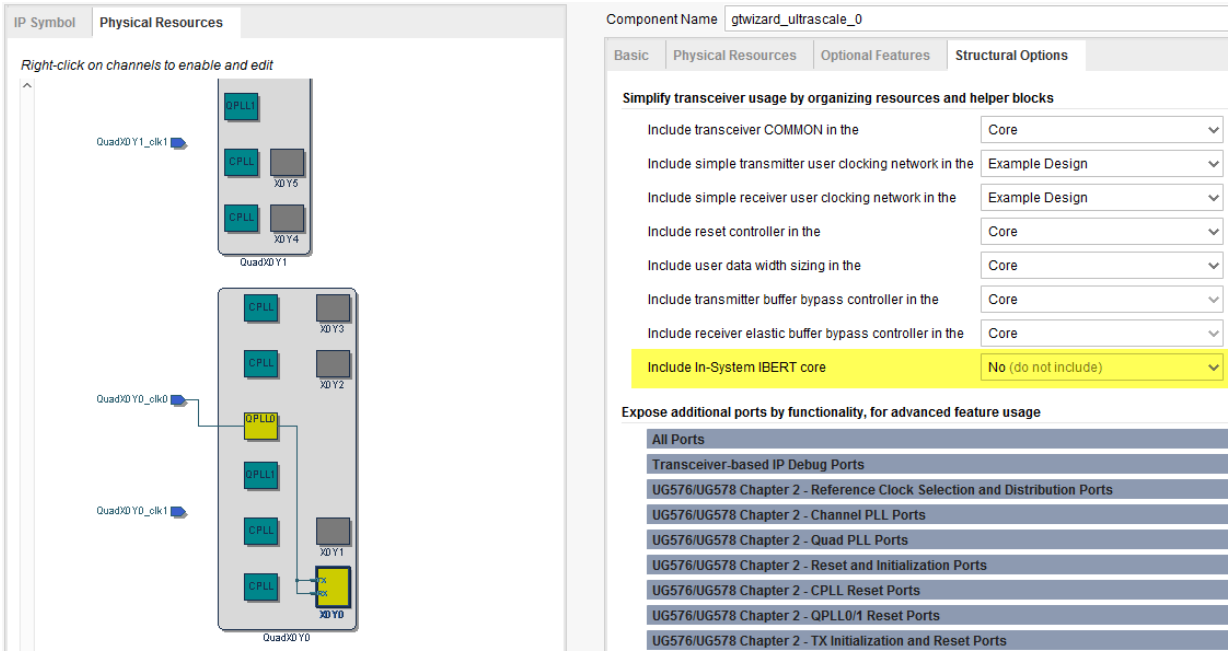

Figure 5-4: **Optional enablement of In-System IBERT Core in GT Wizard Example Design**

Convenience Features

The wizard example design provides several convenience features that can be useful when integrating the core instance into your system.

- Any helper blocks that were specified to be included in the example design are instantiated within the example wrapper level of hierarchy. By including only those resources and no additional demonstration logic, the example design wrapper can be useful to integrate in your own project with no, or minimal modification required.
- If the transceiver common was specified to be included in the example design, all enabled transceiver common instances are also localized within the example wrapper level of hierarchy.
- As described in Chapter 2, Product Specification, the core provides vectored ports that are concatenations of the corresponding port across enabled transceiver primitives. While this provides a compact and predictable user interface, users might prefer individual signals per transceiver primitive. The example design top-level module provides vector slicing for each enabled port, assigning each slice as appropriate for the signal type. This feature can be used for reference or integrated into your system as desired. Three examples follow, for illustration:

a. If the core instance contains four enabled GTH transceiver channels, their GTHTXP serial data output pins are vectored as gthtxp_out[3:0] on the core interface and mapped to gthtxp_int[3:0] within the example design top-level module. The four bits of the vector are sliced into four per-channel assignments that also map to top-level outputs. The "ch" prefix of each signal indicates a transceiver channel signal type, and the number that follows indicates its index among all enabled transceiver channel primitives:

```
wire [3:0] gthtxp_int;
assign ch0_gthtxp_out = gthtxp_int[0:0];
assign ch1_gthtxp_out = gthtxp_int[1:1];
assign ch2_gthtxp_out = gthtxp_int[2:2];
assign ch3_gthtxp_out = gthtxp_int[3:3];
```

b. If the core instance contains three enabled transceiver channels and optionally enables the drpaddr_in port, the 9-bit DRPADDR transceiver channel ports are vectored as drpaddr_in[26:0] on the core interface and mapped to drpaddr_int[26:0] within the example design top-level module. The 27-bits of the vector are sliced into three per-channel assignments that are each set to the same default driver value that the corresponding transceiver primitive port would have been assigned to internally if the port were not exposed on the core interface. If you choose to integrate the vector slicing convenience code into your project, assign the signals as appropriate for your system:

```
wire [26:0] drpaddr_int;
wire [8:0] ch0_drpaddr_int = 9'b000000000;
wire [8:0] ch1_drpaddr_int = 9'b000000000;
wire [8:0] ch2_drpaddr_int = 9'b000000000;
assign drpaddr_int[8:0] = ch0_drpaddr_int;
assign drpaddr_int[17:9] = ch1_drpaddr_int;
assign drpaddr_int[26:18] = ch2_drpaddr_int;
```

c. If the core instance contains one enabled transceiver common and optionally enables the qpll0lock_out port, the one-bit QPLL0LOCK transceiver common port is provided as qpll0lock_out[0:0] on the core interface and mapped to qpll0lock_int[0:0] within the example design top-level module. For this single primitive case, the provided vector slicing is equivalent to a renamed signal. The "cm" prefix of each signal indicates a transceiver common signal type, and the number that follows indicates its index among all enabled transceiver common primitives:

```
wire [0:0] qpll0lock_int;
wire [0:0] cm0_qpll0lock_int;
assign cm0_qpll0lock_int = qpll0lock_int [0:0];
```

d. Multiple instances of a helper block also results in similar assignments. The "hb" prefix of each signal indicates a helper block signal type, and the number that follows indicates its index among all included helper blocks of that type.

Note: A very small number of uncommonly used transceiver primitive ports are tied off to safe values within the core. If you use the optional port enablement interface to enable access to such a port on the core interface, a warning message with usage instructions is included as a code comment with its vector slicing assignments in the example design top-level module.

- The example design top-level module provides easy access to the reset inputs of the user clocking network helper blocks, and by default, drives them with appropriate signals that indicate clock source stability.

Adapting the Example Design

The example design is provided as a means of Wizard IP core demonstration, and it can also prove useful as a starting point for integrating the core into your system. While you should not modify core files themselves, modification of the example design can be a useful part of this adaptation.

IMPORTANT: *Xilinx cannot guarantee support for modifications made to the example design contents as they are delivered, so be sure to understand the effects of your changes and follow any recommendations in this document and in the example design code.*

It can be useful to use the example wrapper level of the example design hierarchy in your system because it instantiates the core and contains the example helper blocks and transceiver common instances if those resources were specified to be located in the example design during IP customization. Helper blocks and transceiver common instances delivered within the example wrapper are provided as examples and can be modified as necessary to suit your system requirements.

Note: The same parameter overrides exist on transceiver common instances for a given core customization, regardless of their instantiated location.

One or more IBUFDS_GTE3 or IBUFDS_GTE4 transceiver differential reference clock buffer primitives are instantiated in the example design top-level module to drive transceiver PLLs as appropriate for your core instance. These buffers as well as any OBUFDS_GTE3 or OBUFDS_GTE4 differential recovered clock output buffers are included in the example design rather than the core to facilitate sharing and for general clocking flexibility. However, they are necessary components of the wizard solution, so the buffer primitives and the nets they connect to should be included in your system. If you wish to use different connectivity in your system, then to properly adjust both the wiring and the transceiver primitive location constraints, re-customize the core, and choose different transceiver reference clock and/or recovered clock buffer locations rather than modifying the clock connectivity.

As described in Constraining the Core, the example design XDC file contains top-level design constraints, some of which will be necessary to include in your system when the Wizard IP core is integrated into it. For example, differential reference clock period constraints and buffer location constraints must be included in your own project XDC file.

The core-level XDC file that constrains individual transceiver channel locations automatically remains associated with each instantiation of the core.

Limitations of the Example Design

The example design is the recommended means of simulating or implementing an instance of the Wizard IP core outside the context of your own system. It can also prove useful as a starting point for integrating the core into your system. However, it is quite simplistic, and the following limitations should be understood:

- The example design does not implement specific protocols to generate or check data. For example, while the example stimulus module does support TX Gearbox data encoding configurations and the example checking module does support RX Gearbox data decoding configurations to interface to the transceiver channel primitives, they do not implement true 64B/66B or 64B/67B data coding. Fundamentally, raw PRBS data is generated and checked.
- When the example design is simulated using the provided test bench, each transceiver channel is looped back from the serial data transmitter to the receiver. As such, data integrity can only be properly checked if the transmitter and receiver are configured for the same line rate and to use the same data coding. No transcoding or rate adjustment schemes are used. If the transmitter and receiver line rates or data coding are configured differently from one another in your system, you might wish to cross-couple two appropriately-customized core instances and check for data integrity in hardware or in your own test bench. In such a setup, the transmitter of core instance A is rate- and coding-matched to the receiver of core instance B, and vice versa.

Test Bench

This chapter contains information about the provided test bench in the Vivado® Design Suite.

The UltraScale™ FPGAs Transceivers Wizard includes a simple self-checking test bench module that provides basic stimulus to the example design and interacts with its link status interface to check for data integrity across all enabled transceiver channels.

Simulating the Example Design

To simulate an instance of the Wizard IP core, first open its example design as described in Chapter 5, Example Design. In the example project, start a behavioral simulation by clicking **Run Simulation > Run Behavioral Simulation** in the Vivado Integrated Design Environment (IDE). The **Simulation Settings** selection can be used to choose the supported simulator of your choice.

The example design instantiates an example stimulus module to drive the transmitter user interface and an example checking module that is driven by the receiver user interface of each transceiver channel. The example design combines the individual PRBS match indicators from each channel into an overall match signal. The combined match signal is the basis of a link status indicator with corresponding sticky link down indicator and dedicated reset input. See Chapter 5, Example Design for more details on the data stimulus, checking, and link status functions of the example design.

The provided test bench instantiates the example design top-level module and loops back each enabled transceiver channel in the core instance from the serial data transmitter to the receiver. This enables the example stimulus, checking, and link status logic within the example design to operate as part of a self-checking system under the stimulus of the simulation test bench. For more information, refer to *Vivado Design Suite: Logic Simulation* (UG900) [Ref 10].

Simulation Behavior

The example design simulation test bench provides the requisite free-running clock and transceiver reference clock signals, as well as a "reset all" pulse to the example design logic and reset controller helper block input ports. This stimulus is sufficient to allow the helper blocks to bring up the remainder of the system. After some time, the transceiver PLL(s) will achieve lock, allowing the reset controller helper block finite state machines to complete the full reset sequence. After the reset sequence is complete, you can begin to observe the example stimulus module transmitting data. A short time later, the example checking module begins to search for data alignment and checks for data integrity, which is in turn used by the link status logic to drive the link status indicator.

Note: To quickly demonstrate operation of the entire example design, the simulation test bench asserts "reset all" from the beginning of operation. In hardware operation, initial reset inputs should not be provided until transceiver power is good. See Reset Sequencing and Other Services for relevant guidelines.

The example design output port `link_status_out` indicates a PRBS match-based link across all channels. The test bench uses a counter to detect a level `link_status_out` assertion, and deassertions reset the counter. When the counter saturates, the test bench prints this message to the transcript:

```
Initial link achieved across all transceiver channels.
```

The test bench then pulses `link_down_latched_reset_in` to reset the example design sticky link down indicator, and allows the simulation to run for a prescribed period of time to ensure that the link is maintained. These messages are printed to the transcript:

```
Resetting latched link down indicator.
Continuing simulation for 50us to check for maintenance of link.
```

At the end of the prescribed wait period, the test bench checks whether the link has been maintained. If so, the following messages are printed to the transcript and the test is considered to have passed. The simulation then finishes:

```
PASS: simulation completed with maintained link.
** Test completed successfully
```

Figure 6-1 shows the characteristic waveform of a passing test, demonstrating initial link, a saturating link up counter leading to the link stable indicator, a pulse to reset the sticky link down indicator, and the beginning of the wait period where the test bench is run while the sticky link down indicator remains deasserted. The signals shown are those from the test bench level of hierarchy only, and are the default set when loading a simulation from the Vivado design tools. You might wish to add additional signals to the waveform window for more visibility into the operation of the example design or the core instance.

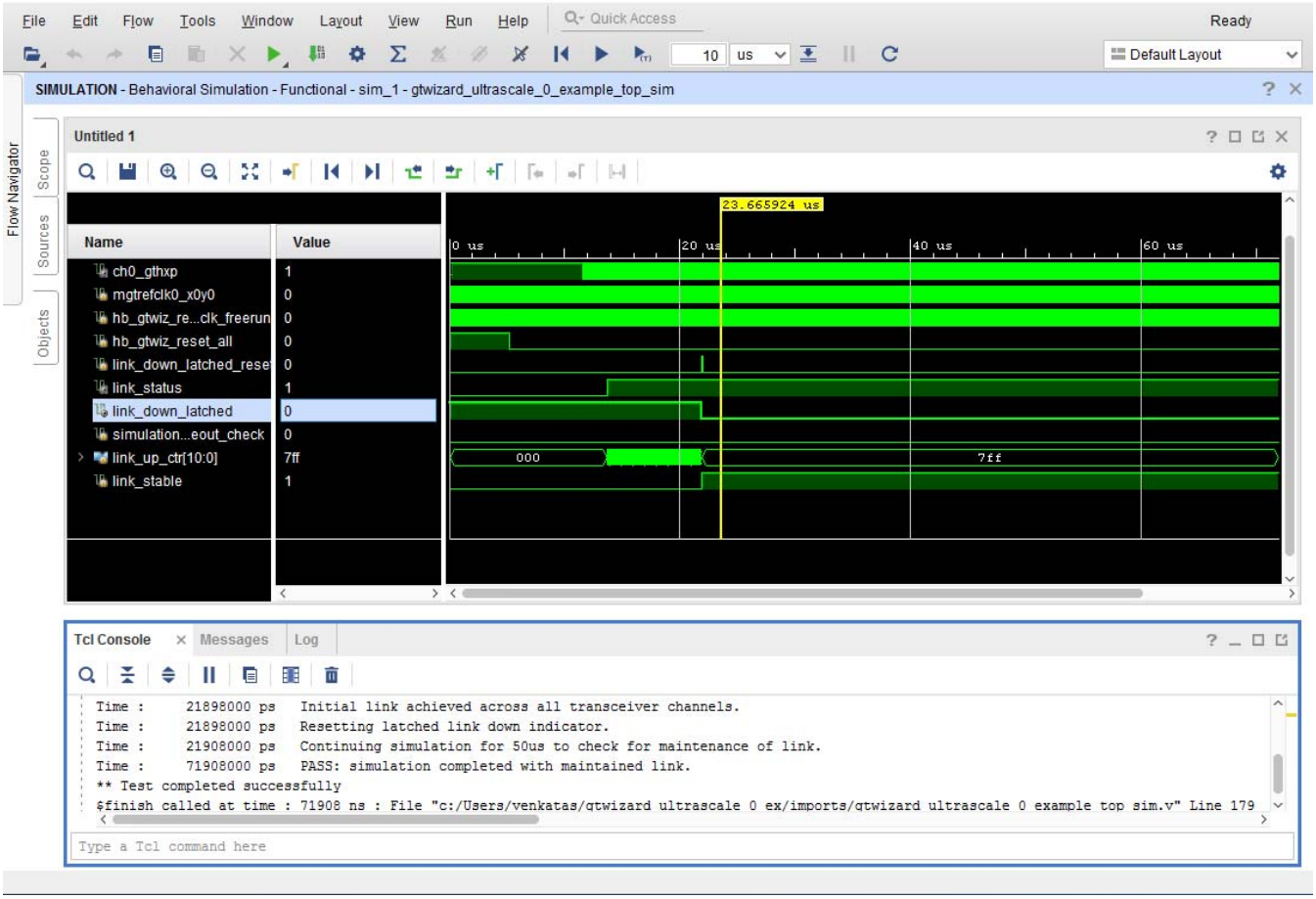

Figure 6-1: **Test Bench Simulation Waveform of a Passing Test**

If the link is lost after it was first achieved, the following messages are printed to the transcript and the test is considered to have failed. The simulation then finishes:

```
FAIL: simulation completed with subsequent link loss after initial link.
** Error: Test did not complete successfully
```

Use the "run all" feature of your simulator to allow the simulation to run for an unbounded period of time. The provided test bench includes a timeout process that, should the time limit be reached before a stable link is first achieved, prints the following message to the transcript before exiting the simulation. This behavior is considered a test failure and is not expected:

```
FAIL: simulation timeout. Link never achieved.
** Error: Test did not complete successfully
```

Appendix A

Migrating and Upgrading

This appendix contains information about 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 11].

Upgrading from a Previous Version

You are encouraged to upgrade from a prior version of the UltraScale™ FPGAs Transceivers Wizard IP to the newest version for compatibility and the benefit of any feature enhancements or bug fixes. Refer to the wizard core changelog for a record of versions and their specific changes, and make note of any messages produced during the IP upgrade process.

Migrating across Devices

When the IP is migrated across devices, there is no guaranteed direct upgrade path from Transceivers Wizard IP cores. Xilinx recommends you to review their GT locations and other settings when the device has been changed and IP upgrade is performed. The upgrade of GT Wizard IP is based on channel locations and not on PACKAGE_PIN assignment. Hence, ensure that the GT locations are provided for the target device.

Migrating from a Previous Device Family

There is no direct upgrade path from Transceivers Wizard IP cores for previous Xilinx device families to the UltraScale FPGAs Transceivers Wizard, which supports only UltraScale and UltraScale+™ device families. It is necessary to customize and generate a new core instance when targeting an UltraScale or UltraScale+ device for the first time.

The wizard provides many of the same options present in previous Xilinx® Wizards IP cores, as well as significant new features and flexibility.

Appendix B

Debugging

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

Finding Help on Xilinx.com

To help in the design and debug process when using the UltraScale™ FPGAs Transceivers Wizard, 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.

Documentation

This product guide is the main document associated with the Wizard. 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 from the Downloads page. For more information about this tool and the features available, open the online help after installation.

Answer Records

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

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

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

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

Master Answer Record for the UltraScale FPGAs Transceivers Wizard

AR: 57487

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.

Vivado Design Suite Debug Feature

The Vivado® Design Suite debug feature inserts logic analyzer and virtual I/O cores directly into your design. The debug feature also 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 6.2 (and later versions)
- VIO 3.0 (and later versions)

See the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 12].

Appendix C

Additional Resources and Legal Notices

Xilinx Resources

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

References

These documents provide supplemental material useful with this product guide:

1. *UltraScale Architecture GTH Transceivers User Guide* (UG576)
2. *UltraScale Architecture GTY Transceivers User Guide* (UG578)
3. *Kintex UltraScale Architecture Data Sheet: DC and AC Switching Characteristics* (DS892)
4. *Virtex UltraScale Architecture Data Sheet: DC and AC Switching Characteristics* (DS893)
5. *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925)
6. *Kintex UltraScale+ FPGAs Data Sheet: DC and AC Switching Characteristics* (DS922)
7. *Virtex UltraScale+ FPGA Data Sheet: DC and AC Switching Characteristics* (DS923)
8. *Vivado Design Suite User Guide: Designing with IP* (UG896)
9. *Vivado Design Suite User Guide: Getting Started* (UG910)
10. *Vivado Design Suite: Logic Simulation* (UG900)
11. *ISE to Vivado Design Suite Migration Guide* (UG911)
12. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
13. *In-System IBERT v1.0 LogiCORE Product Guide* (PG246)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
12/04/2020	1.7	• Updated Table 3-1.
10/30/2019	1.7	• Updated Enabling CPLL Calibration block for UltraScale+ Devices section. • Updated USER_GTPOWERGOOD_DELAY_EN parameter value.
06/21/2019	1.7	• Updated description for USER_GTPOWERGOOD_DELAY_EN
04/04/2018	1.7	• Updated descriptions of gtgrefclk0_in, gtgrefclk1_in, and gtgrefclk_in. • Updated descriptions of rxpllclksel_in and txpllclksel_in • Reference to Xilinx Answer added in Enabling CPLL Calibration block for UltraScale+ devices section • Note about the usage of DISABLE_LOC_XDC added under Required Constraints in Design Flow Steps chapter
10/04/2017	1.7	• Updated description and guidance for usage of `SIM_CPLL_CAL_BYPASS`
06/07/2017	1.7	• Updated core to v1.7 • CPLL Calibration block is enabled by default for CPLL configurations of GTHE4 and GTYE4 • `gtpowergood_out` is added as a mandatory port
04/05/2017	1.6	• Updated Simulation guidance for CPLL Calibration block • Updated Spread Spectrum related guidance
11/30/2016	1.6	• Updated for CPLL Calibration block for UltraScale+ devices
10/05/2016	1.6	• Updated for 2016.3 release • Added In-System IBERT Core instantiation in Example design with 2016.3 • Added reference to PG246 - In-System IBERT LogiCORE IP Product Guide
11/18/2015	1.6	• Added support for UltraScale+™ families
09/30/2015	1.6	• Updated the maximum frequency in the Free-Running Clock Maximum Frequency table • Updated the resource utilization data • Clarified that the reset controller helper block's reset all input is falling-edge triggered • Wizard options: ◦ Added QPLL Fractional-N ◦ Removed Fractional part of QPLL feedback divider (Advanced option) ◦ Updated descriptions for Enabling a channel (Physical Resources tab)

Date	Version	Revision
02/23/2015	1.5	• Updated for the core v1.5 • Specified engineering sample (ES1 or ES2) devices in all references to GTH transceiver configurations which use the CPLL • Added Virtual Input/Output (VIO) core instance to the example design • In Chapter 4, clarified that the core is not available in Vivado IP integrator, and further clarified the constraints support details
10/01/2014	1.4	• Updated for 2014.3 release • IP Facts: Updated eighth bullet in Features • Chapter 1, updated ninth bullet in Feature Summary • Chapter 2: ◦ Updated number of LUTs and flip-flops in Resource Utilization, including Table 2-2 ◦ In Table 2-3, removed gtwiz_reset_rx_data_good_in and updated description of gtwiz_reset_rx_cdr_stable_out ◦ Added gtpowergood_in to Table 2-6 • Chapter 3: Updated second paragraph in Reset Sequencing and Other Services • Chapter 4: ◦ Updated Enabling a channel bullet in Channel Table and Channel Graphic ◦ Updated Figure 4-1 to Figure 4-4 • Chapter 5: ◦ Added second bullet to Core-level Constraints ◦ In Clock Frequencies, replaced create_clock commands with set_case_analysis commands ◦ Updated placeholder package pin constraints in I/O Standard and Placement ◦ Updated XDC file commands in Other Constraints ◦ Updated first bullet and last paragraph in Purpose of the Example Design ◦ Updated Hierarchy and Structure ◦ In Figure 5-1, added Initialization block and renamed PRBS Lock I/O block to Link Status ◦ In Table 5-1, added rxrecclkout_ch<j>_p/n and updated link_down_latched_reset_in, link_status_out, and link_down_latched_out ports ◦ Added Link Status and Initialization ◦ Removed third bullet in Limitations of the Example Design • Chapter 6: ◦ Updated second and third paragraphs in Simulating the Example Design ◦ Updated Simulation Behavior, including Figure 6-1 • Appendix B: Updated ILA and VIO versions in Vivado Lab Edition

Date	Version	Revision
06/04/2014	1.3	Updated for 2014.2 release • Corrected titles for UltraScale FPGAs GTH Transceivers User Guide (UG576) and UltraScale FPGAs GTY Transceivers User Guide (UG578) throughout • Chapter 2: ◦ In Table 2-2, updated reset controller, transmitter buffer bypass controller utilization, and receiver buffer bypass controller numbers ◦ Added a row for user data width sizing ◦ In Table 2-3, updated description of gtwiz_reset_rx_data_good_in • Chapter 3: ◦ Added recovered clock buffers to Designing with the Example Design ◦ Updated first paragraph of Reset Controller Helper Block ◦ Updated receiver reset state machine in Figure 3-1 ◦ Updated third paragraph of Reset Sequencing and Other Services • Chapter 4: ◦ Updated first bullet in System Frame ◦ Updated Encoding bullet in Transmitter Frame ◦ Updated decoding bullet in Receiver Frame ◦ Updated first paragraph in Channel Table and Channel Graphic ◦ Updated fourth bullet in Helper Block Location Frame ◦ Updated Figure 4-1 to Figure 4-4 • Chapter 5: ◦ Removed core-level clock period constraints bullet from Core-level Constraints ◦ Updated Out-of-Context Constraints ◦ Updated Clock Frequencies ◦ Chapter 5: Updated last paragraph of Adapting the Example Design • Chapter 6: Updated Figure 6-1

Date	Version	Revision
04/02/2014	1.2	Updated for 2014.1 release • Chapter 1: Updated fifth bullet in Feature Summary • Chapter 2: ◦ Updated note in Maximum Frequencies ◦ In Table 2-3, added gtwiz_reset_qpll0lock_in, gtwiz_reset_qpll1lock_in, gtwiz_reset_qpll0reset_out, and gtwiz_reset_qpll1reset_out ports ◦ Updated description of gtwiz_reset_rx_cdr_stable_out port ◦ Added gtwiz_userclk_tx_reset_in port to Table 2-9 ◦ In Table 2-25, corrected rxckokreset_in and rxckokdone_out to rxckcalreset_in and rxckcaldone_out, respectively • Chapter 3: ◦ Updated paragraphs after Figure 3-1 ◦ Added Special GTH Transceiver CPLL Reset Requirements for Engineering Sample (ES1 or ES2) Devices ◦ Updated description of CDR stability in second paragraph of Reset Sequencing and Other Services • Chapter 4: ◦ Updated all figures ◦ Added "Fractional part of QPLL feedback divider" to Transmitter Frame (Advanced Section) ◦ In Receiver Frame (Advanced Section), added Fractional part of QPLL feedback divider and Enable Out of Band signaling (OOB)/Electrical Idle and removed Jitter tolerance mask: Mask corner frequency (MHz) and Jitter tolerance mask: Mask low frequency slope (dB/decade) ◦ Added Free-Running and DRP Clock Frequency (MHz) ◦ Updated Manual alignment (RXSLIDE) mode bullet in Receiver Comma Detection and Alignment Section ◦ Updated Enable and select number of sequences to use bullet in Receiver Channel Bonding Section ◦ Updated Enable and select number of sequences to use bullet in Receiver Clock Correction Section ◦ Added Advanced Clocking Section and SATA Section ◦ Removed note from Include reset controller in the... bullet in Helper Block Location Frame • Chapter 5: ◦ Updated first bullet in Core-level Constraints ◦ Updated Clock Frequencies • Appendix C: Added UltraScale FPGAs GTY Transceivers User Guide (UG578) and Virtex UltraScale Architecture Data Sheet: DC and AC Switching Characteristics (DS893) to References
12/18/2013	1.1	Initial Xilinx release

Please Read: Important Legal Notices