Virtex UltraScale+ FPGAs GTM Transceivers Wizard v1.0

LogiCORE IP Product Guide

Vivado Design Suite

PG315 (v1.0) November 24, 2020

XILINX®

Table of Contents

Introduction

The Virtex® UltraScale+™ FPGAs GTM Tranceivers Wizard IP core helps configure one or more serial transceivers. You can start from scratch, input your requirements, and generate valid configurations, or chose to start from one of the existing presets applicable to design requirements. The flexible Wizard generates a customized IP core for the transceivers, configuration options, and enabled ports you have selected, 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

- Simple and intuitive feature selection flow.
- Automatically sets transceiver parameters.
- Available helper blocks to simplify common or complex transceiver usage.
- Example design with configurable PRBS generator, checker, and link status indicator to demonstrate functionality in simulation and hardware.
- Support for GTM transceivers in Virtex UltraScale+ devices.
- 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.
- Advanced configuration options to tune transceiver performance.
- 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.
- 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.
    - Simulation test bench that monitors example design PRBS lock in external loopback, and indicates resulting link status.

- Additional convenience features, including differential reference clock buffer instantiation and wiring, and dual vector slicing.

IP Facts

LogiCORE™ IP Facts Table	
Core Specifics	
Supported Device Family[1]	Virtex® UltraScale+™ devices
Supported User Interfaces	Not applicable
Resources	Performance
Provided with Core	
Design Files	RTL
Example Design	System Verilog
Test Bench	System Verilog
Simulation Model	For supported simulators, see the Xilinx Design Tools: Release Notes Guide
Supported S/W Driver	N/A
Tested Design Flows[3]	
Simulation	For supported simulators, see the Xilinx Design Tools: Release Notes Guide
Support	
Release Notes and Known Issues	Master Answer Record: 72071
All Vivado® IP Change Logs	Master Vivado IP Change Logs: 72775
Provided by Xilinx® at the Xilinx Support web page	

Notes:

1. For a complete list of supported devices, see the Vivado IP catalog.
2. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.
3. In this version of the GTM Wizard IP, SIP_GTM_DUAL instantiates the transceiver serial ports as integers to showcase the PAM4 encoding levels. While this is masked as logic wires at GTM_DUAL, the GTM Wizard parent IP will have to perform a hierarchical access of these ports for simulation purposes. So the legal values for transitions on these ports while in PAM4 mode are: 0/1/2/3, while in NRZ mode they are: 0/3.
4. Refer to the Simulation section for mixed language simulation options.

Related Information

VHDL GTM Transceiver Parent IP Simulation Workarounds

Chapter 2

Overview

The Virtex® UltraScale+™ FPGAs GTM transceivers Wizard IP core is used to configure and simplify the use of one or more GTM serial transceivers in a Virtex UltraScale+ device. See Chapter 3: Product Specification for a detailed description of the core. This document describes the Wizard IP core. See the *UltraScale FPGAs GTM Transceivers User Guide* (UG581) for details on the specific use and behavior of the serial transceivers.

Navigating Content by Design Processes

Xilinx® documentation is organized around a set of standard design processes to help you find relevant content for your current development task. This document covers the following design processes:

- **Hardware, IP, and Platform Development:** Creating the PL IP blocks for the hardware platform, creating PL kernels, subsystem functional simulation, and evaluating the Vivado® timing, resource use, and power closure. Also involves developing the hardware platform for system integration.
    - Port Descriptions
    - Reset Controller Helper Block Ports
    - Transmitter User Clocking Network Helper Block Ports
    - Receiver User Clocking Network Helper Block Ports
    - Customizing and Generating the Core
    - Chapter 6: Example Design
- **Board System Design:** Designing a PCB through schematics and board layout. Also involves power, thermal, and signal integrity considerations.
    - Sampled Eye Scan Functionality
    - Reset Controller Helper Block Ports
    - Simulating the Example Design

Applications

The GTM Wizard is the supported method of configuring and using one or more serial GTM transceivers in a Xilinx® Virtex® UltraScale+™ FPGA.

Licensing and Ordering

This Xilinx® LogiCORE™ IP module is provided at no additional cost with the Xilinx Vivado® Design Suite under the terms of the Xilinx End User License.

Information about 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.

Chapter 3

Product Specification

The Virtex® UltraScale+™ FPGAs GTM transceivers Wizard IP core is the supported method of configuring and using one or more serial GTM transceivers in a Virtex 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 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 `GTM_DUAL`transceiver primitives to provide a simplified user interface to those resources. The core instance configures the dual primitives by applying HDL parameter values derived from the Vivado® Integrated Design Environment (IDE)-driven customization of that instance.
- **Transceiver Configuration:** During Vivado IDE-driven customization, you can customize transceiver configuration settings to suit your application.
- **Helper Blocks:** The Wizard provides helper blocks that are abstract or automate certain common or complex transceiver usage procedures. Each helper block can be located either within the core or outside it, 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:** Controls and abstracts the transceiver reset sequence.
  - **Receiver user clocking network:** Contains resources to drive the receiver user clocking network.
  - **Transcoder:** This block implements the transmit and receive transcode and alignment marker removal, mapping and insertion functions for 100G and 50G KP4 Ethernet.

The Wizard is intended to simplify the use of the serial GTM transceivers. However, it is still important to understand the behavior, usage, and any limitations of the transceivers. See the *UltraScale FPGAs GTM Transceivers User Guide* (UG581) for details.

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.

Table 1: **Maximum Frequencies**

Transceiver User Clock Frequency Relationship	Maximum Frequency of gtwiz_reset_clk_freerun_in
$F_{RXUSRCLK2} \le F_{TXUSRCLK2}$	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 250 MHz for UltraScale+™ devices.

Other Performance Characteristics

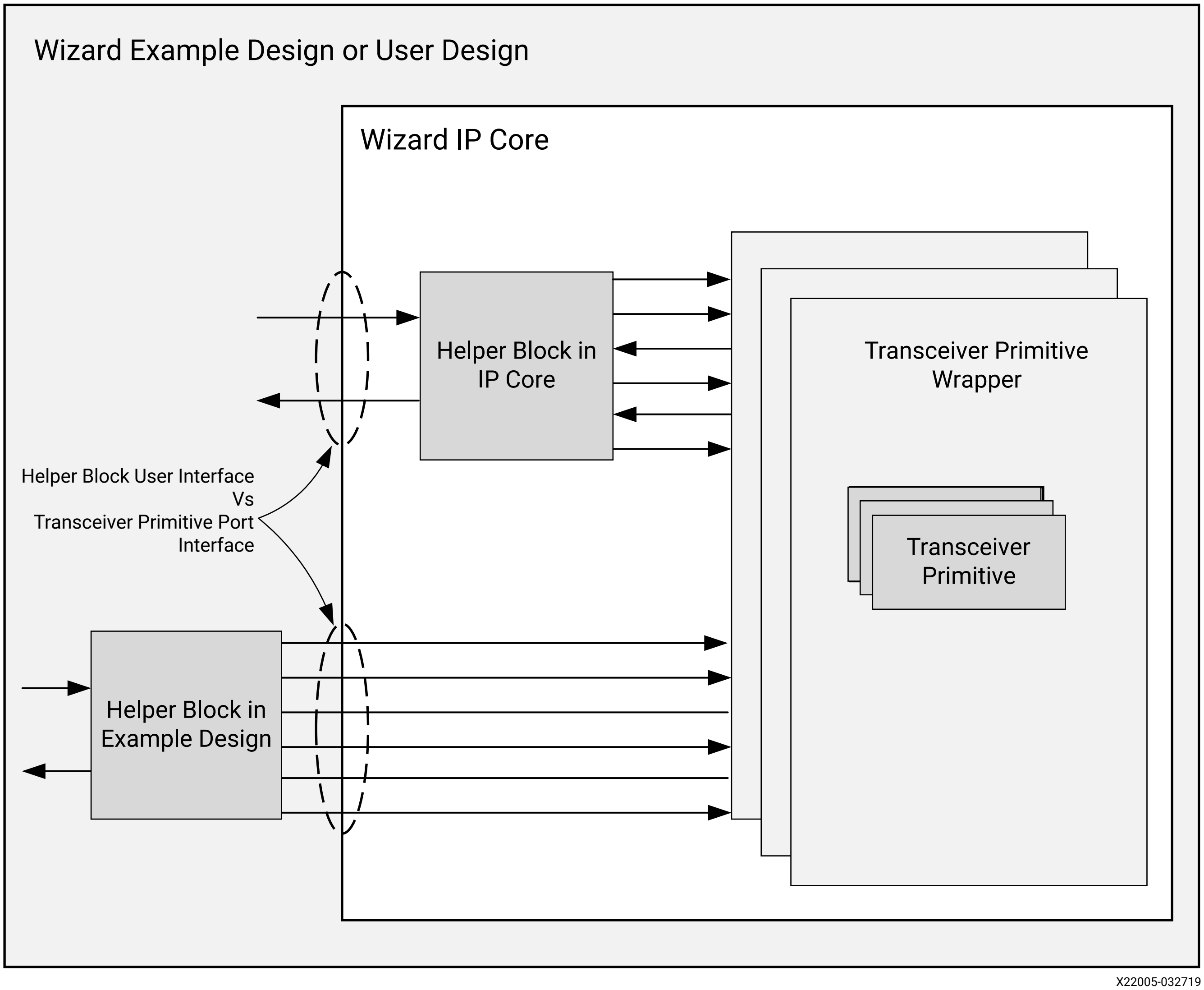
See the *UltraScale FPGAs GTM Transceivers User Guide* (UG581) for other performance characteristics of the transceiver primitives and valid line rate ranges.

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. For the applicable tentative port tie off values, refer to Chapter 6: Example Design.

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. The FEC Transcode helper block is always located within the core when enabled, refer to the Transcode helper block section for a description. The following figure illustrates how the location of the Helper block affects core port enablement.

Figure 1: **Effect of Port Helper Block Location on Port Enablement**



Related Information

Transcode Helper Block

GTM Controller Helper Logic

The GTM Controller IP instantiates a MicroBlaze™ processor to control the sequencing of RXRESET and enhance link stability. gtm_cntrl_v1_0 IP is instantiated as a hierarchical IP inside the GTM Wizard IP to handle internal sequencing required over DRP ports. It is recommended that the default False option for BYPASS_GTM_CNTRL remains unchanged, except for advanced use cases where GTM Wizard is used for FEC only use-cases, that are typically used in GT loopback mode.

The GTM controller logic serves up to four duals within the same SLR region. The number of duals selected in the GTM Wizard should be sequential and it should not cross SLR boundaries. It is recommended to not perform any DRP operations while the RX reset sequence is in progress.

Refer to gtwiz_sol_gpo port description.

Table 2: **GTM Controller Logic Port Descriptions**

Port Name	I/O	Clock	Description
ubclockper_rxcdr_lock_time[15:0]	Input	gtwiz_reset_clk_freerun_in (sync)	Configurable wait time for the GTM controller; Tie off to value 16'h0100.
ch0_resetsol_en [3:0]	Input	gtwiz_reset_clk_freerun_in (sync)	Enables CH0 for DUALS [3:0] to use the GTM controller; each bit enables the GTM controller for `Dual3_Ch0`, `Dual2_Ch0`, `Dual1_Ch0`, `Dual0_Ch0`. ***Note***: Drive used DualX_Ch0 to 1 and unused to 0. Make sure that this port value is not changed when a reset request is in progress. If there is a reason to change the port value, pulse the datapath reset input of the helper logic.
ch1_resetsol_en [3:0]	Input	gtwiz_reset_clk_freerun_in (sync)	Enables CH1 for DUALS [3:0] to use the GTM controller. Each bit enables the GTM controller for `Dual3_Ch1`, `Dual2_Ch1`, `Dual1_Ch1`, `Dual0_Ch1`. ***Note***: Drive used DualX_Ch1 to 1 and unused to 0. Make sure that this port value is not changed when a reset request is in progress. If there is a reason to change the port value, pulse the datapath reset input of the helper logic.
gtrxreset_req_user	Input	gtwiz_reset_clk_freerun_in (sync)	Requests the GTM controller to initiate an RX reset sequence for channels specified by `ch0_resetsol_en / ch1_resetsol_en`. The recommended connection for this is from the reset controller helper logic output.

Table 2: **GTM Controller Logic Port Descriptions** *(cont'd)*

Port Name	I/O	Clock	Description
gtwiz_sol_gpo[3:0]	Output	gtwiz_reset_clk_freerun_in (sync)	Status indicator logic from the GTM controller. Monitor this for the GTM controller status to determine if the RX reset was successful: 0x0 = GTM controller in reset 0x1 = GTM controller initialized/ idle 0x3 = GTM controller completed an RX reset successfully 0x7 = GTM controller completed an RX Reset unsuccessfully ***Note***: 1. When the GTM controller is enabled, ensure that the status RX reset is successful. 2. The DRP operations are not expected to be performed when the status of gtwiz_sol_gpo[3:0] is either 0 or 1, that is when the GTM controller reset sequence is in progress. The link behavior may not be reliable if any DRP operations are performed during this stage. You may require an additional reset pulse to get a clean link again.
es_fifo_request [7:0]	Input	ASYNC	Enables sampled Eye Scan for selected channel {D3Ch1, D3Ch0, D2Ch1, D2Ch0, D1Ch1, D1Ch0, D0Ch1, D0Ch0} ***Note***: 1. If multiple bits are simultaneously asserted, only the LSB channel is serviced. 2. De-assert es_fifo_request when the es_fifo_full flag asserts.
es_fifo_rclk	Input		ES FIFO Read clk It is recommended that you use the same source as freerun_clk in the GTM Wizard example design.
es_fifo_axis_0_tready	Input	es_fifo_rclk	AXI4-Stream Interface: Indicates that the slave can accept a transfer in the current cycle.

Table 2: **GTM Controller Logic Port Descriptions** *(cont'd)*

Port Name	I/O	Clock	Description
es_fifo_axis_0_tvalid	Output	es_fifo_rclk	AXI4-Stream Interface: Indicates that the master is driving a valid transfer. A transfer takes place when both tvalid and tready are asserted. ***Note*:** tvalid also serves as the Empty flag for the FIFO.
es_fifo_axis_0_tdata [15:0]	Output	es_fifo_rclk	AXI4-Stream Interface: Read Dataout from the FIFO.
es_fifo_full	Output	es_fifo_rclk	Status flag that indicates sampled Eye Scan for selected channel has completed and FIFO is filled.
es_fifo_axis_0_tlast	Output	es_fifo_rclk	Reserved.
gtm_cntrl_ch0_rxclkrdy[3:0]	Output	RXUSRCLK2	Status signal that indicates the stability of the rxusrclk2 when the GTM Controller logic performs internal DRP operations.
gtm_cntrl_ch1_rxclkrdy[3:0]	Output	RXUSRCLK2	Status signal that indicates the stability of the rxusrclk2 when the GTM Controller logic performs internal DRP operations.
gtm_cntrl_in_fecrx0cwinc	Input	RXUSRCLK2	Slice 0 codeword count increment.
gtm_cntrl_in_fecrx0uncorrcwinc	Input	RXUSRCLK2	Slice 0 uncorrected codeword count increment.
gtm_cntrl_in_fecrx1cwinc	Input	RXUSRCLK2	Slice 1 codeword count increment.
gtm_cntrl_in_fecrx1uncorrcwinc	Input	RXUSRCLK2	Slice 1 uncorrected codeword count increment.
gtm_cntrl_in_fecrxln0biterr0to1inc	Input	RXUSRCLK2	Lane0 bit error count increment (0 corrected to 1).
gtm_cntrl_in_fecrxln0biterr1to0inc	Input	RXUSRCLK2	Lane0 bit error count increment (1 corrected to 0).
gtm_cntrl_in_fecrxln1biterr0to1inc	Input	RXUSRCLK2	Lane1 bit error count increment (0 corrected to 1).
gtm_cntrl_in_fecrxln1biterr1to0inc	Input	RXUSRCLK2	Lane1 bit error count increment (1 corrected to 0).
gtm_cntrl_in_fecrxln2biterr0to1inc	Input	RXUSRCLK2	Lane2 bit error count increment (0 corrected to 1).
gtm_cntrl_in_fecrxln2biterr1to0inc	Input	RXUSRCLK2	Lane2 bit error count increment (1 corrected to 0).
gtm_cntrl_in_fecrxln3biterr0to1inc	Input	RXUSRCLK2	Lane3 bit error count increment (0 corrected to 1).
gtm_cntrl_in_fecrxln3biterr1to0inc	Input	RXUSRCLK2	Lane3 bit error count increment (1 corrected to 0).
gtm_cntrl_in_fectrxln0lock	Input	RXUSRCLK2	Lane 0 lock status.
gtm_cntrl_in_fectrxln1lock	Input	RXUSRCLK2	Lane 1 lock status
gtm_cntrl_in_fectrxln2lock	Input	RXUSRCLK2	Lane 2 lock status

Table 2: **GTM Controller Logic Port Descriptions** *(cont'd)*

Port Name	I/O	Clock	Description
gtm_cntrl_in_fectrxln3lock	Input	RXUSRCLK2	Lane 3 lock status
Temperature [9:0]	Input	gtwiz_reset_clk_freerun_in (sync)	10-bit ADC code from the SYSMON temperature sensor. Valid values must always be assigned. An example implementation of System Management Wizard IP is instantiated in the example design. If you are designing with the example designs, you do not need custom connections to temperature [9:0] ports. Designs using PAM4 modulation either with less than 12 dB insertion loss at Nyquist or with line rate greater than 53 Gb/s must integrate the SYSMON instantiation and provide valid values to the temperature [9:0] ports at all times; they cannot leave the port to be undriven or tie-off to 0's.

GTM Wizard IP when in PAM4 enabled configurations, in general require RS-FEC implementations as part of the design, either using some custom parent IP implementations or the choice of using the integrated KP4 RS-FEC inside GTM_DUAL

The GTM Wizard IP requires the integrated KP4 RS-FEC for designs with PAM4 modulation and insertion loss of less than 12 dB (as set in the GTM Wizard Receiver Advanced Options) to be enabled. Designs that do not utilize the integrated KP4 RS-FEC for specified use mode must implement their own KP4 RS-FEC logic to provide equivalent statistics information as described in the RS FEC section in *UltraScale FPGAs GTM Transceivers User Guide* (UG581). Note that the above ports are vectorized for each dual enabled in user design.

Sampled Eye Scan Functionality

The Sampled Eye Scan functionality has been added to the GTM Control IP and populates an ES FIFO with equalized ADC samples and loop coefficients used for SNR calculation. Each sample of `es_fifo_full` from an `es_fifo_request` contains 1750 data samples with the following data structure as shown in the following table. The Sampled Eye Scan function is shared between 4 GTM_DUALs and only one channel can be serviced at a time.

Table 3: **Data Structure**

FIFO Location	Data Content
1	DUAL/Channel ID
2	Coefficient (H0_P3X)
3	Coefficient (H0_P2X)

Table 3: **Data Structure** *(cont'd)*

FIFO Location	Data Content
4	Coefficient (H0_P1X)
5	Coefficient (H0_0)
6	Coefficient (H0_M1X)
7	Coefficient (H0_M2X)
8	Coefficient (H0_M3X)
9	YK_DATA2[7:0],YKDATA1[7:0]
0	YK_DATA4[7:0],YKDATA3[7:0]
883	YK_DATA1750[7:0],YKDATA1749[7:0]

Instructions on how to use the GTM Sampled Eye Scan feature are as follows:

1. After GT RXRESET is complete, assert `es_fifo_request` at any time to initiate a sampled eye scan for a specific DUAL.

   `es_fifo_request` should stay asserted while monitoring the `es_fifo_full` flag.
2. Once the `es_fifo_full` flag is asserted, de-assert the `es_fifo_request` input.
3. Begin reading the data through the added `es_fifo_* AXI4S` interface.
4. Once the `es_fifo` is completely drained, the `es_fifo_axis_0_tvalid` will go low, thus preventing further reads.

   YK_DATAx are signed values (2s complement); they are not continuous samples. It is just a data set of equalized sampled data and can be plotted directly.

   H0_* - are not necessary to plot the data.

You can acquire additional sampled eye data by repeating the above steps on each of the target required GTs.

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 Designing with the Core.

The reset controller helper block user interface ports described in the following table are present on the helper block itself. It is directly accessible as the helper block is located in the example design.

Table 4: **Port Descriptions**

Port Name	I/O	Clock	Description
gtwiz_reset_clk_freerun_in	Input	N/A	Free-running clock used to reset transceiver primitives. Must be toggling prior to device configuration. See Performance for maximum frequency guidance. Width = 1
gtwiz_reset_all_in	Input	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. This also works as the master reset for the entire gtm_cntrl helper logic. Width = 1
gtwiz_reset_tx_pll_and_datapath_in	Input	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. Width = 1
gtwiz_reset_tx_datapath_in	Input	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. Width = 1
gtwiz_reset_rx_pll_and_datapath_in	Input	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. Width = 1
gtwiz_reset_rx_datapath_in	Input	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. Width = 1
gtwiz_reset_tx_done_out	Output	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. Width = 1
gtwiz_reset_rx_done_out	Output	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. ***Note***: You must monitor the gtwiz_sol_gpo status for complete status when the GTM Controller helper block is present. Width = 1

Table 4: **Port Descriptions** *(cont'd)*

Port Name	I/O	Clock	Description
gtwiz_reset_userclk_rx_active_in	Input	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. Width = 1
gtwiz_reset_userclk_tx_active_in	Input	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. Width = 1

Related Information

Designing with the Core

Reset Controller Helper Block Transceiver Interface Ports

The reset controller helper block transceiver interface ports described in the following table 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 5: **Reset Controller Helper Block Transceiver Interface Ports**

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

Table 5: **Reset Controller Helper Block Transceiver Interface Ports** *(cont'd)*

Port Name	I/O	Clock	Description
rxresetdone_in	Input	Async	Logical AND of all RXRESETDONE signals produced by transceiver dual primitives. Width = 1
pllreset_tx_out	Output	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 dual primitives. Width = 1
txprogdivreset_out	Output	gtwiz_reset_clk_freerun_in (used asynchronously)	Active-High signal fanned out to TXPROGDIVRESET port of all transceiver dual primitives. Width = 1
gttxreset_out	Output	gtwiz_reset_clk_freerun_in (used asynchronously)	Active-High signal fanned out to GTTXRESET port of all transceiver dual primitives. Width = 1
txuserrdy_out	Output	gtwiz_reset_clk_freerun_in (used asynchronously)	Active-High signal fanned out to TXUSERRDY port of all transceiver dual primitives. Width = 1
pllreset_rx_out	Output	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. Width = 1
rxprogdivreset_out	Output	gtwiz_reset_clk_freerun_in (used asynchronously)	Active-High signal fanned out to RXPROGDIVRESET port of all transceiver dual primitives. Width = 1
gtrxreset_out	Output	gtwiz_reset_clk_freerun_in (used asynchronously)	Active-High signal fanned out to GTRXRESET port of all transceiver dual primitives. Width = 1
rxuserrdy_out	Output	gtwiz_reset_clk_freerun_in (used asynchronously)	Active-High signal fanned out to RXUSERRDY port of all transceiver dual primitives. Width = 1

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.

Reset Controller Helper Block Tie-off Ports

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

Table 6: **Reset Controller Helper Block Tie-off Ports**

Port Name	I/O	Clock	Description
tx_enabled_tie_in	Input	gtwiz_reset_clk_freerun_in	When tied High, transmitter resources are reset as part of the sequence in response to gtwiz_reset_all_in. Width = 1
rx_enabled_tie_in	Input	gtwiz_reset_clk_freerun_in	When tied High, receiver resources are reset as part of the sequence in response to gtwiz_reset_all_in. Width = 1
shared_pll_tie_in	Input	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. Width = 1

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 Designing with the Core.

The transmitter user clocking network helper block ports described in the following table 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 7: **Transmitter User Clocking Network Helper Block Ports**

Port Name	I/O	Clock	Description
gtwiz_userclk_tx_reset_in	Input	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. Width = 1
gtwiz_userclk_tx_srcclk_out	Output	N/A	Transceiver primitive-based clock source used to derive and buffer TXUSRCLK and TXUSRCLK2 outputs. Width = 1
gtwiz_userclk_tx_usrclk_out	Output	N/A	Drives TXUSRCLK of transceiver channel primitives. Derived from `gtwiz_userclk_tx_srcclk_in/out`, buffered and divided as necessary by BUFG_GT primitive. Width = 1
gtwiz_userclk_tx_usrclk2_out			Drives TXUSRCLK2 of transceiver dual primitives. Derived from `gtwiz_userclk_tx_srcclk_in/out`, buffered and divided as necessary by BUFG_GT primitive if required. Width = 1

Table 7: **Transmitter User Clocking Network Helper Block Ports** *(cont'd)*

Port Name	I/O	Clock	Description
gtwiz_userclk_tx_active_out		gtwiz_userclk_ tx_usrclk2_out	Active-High indication that the clocking resources within the helper block are not held in reset. Width = 1

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

Table 8: **Transmitter User Clocking Network Helper Block User Interface Ports on Core (Helper Block in Example Design)**

Port Name	I/O	Clock	Description
gtwiz_userclk_tx_active_in	Input	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. Width = 1
gtwiz_userclk_tx_reset_in	Input	Async	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. Width = 1
gtwiz_userclk_tx_srcclk_in	Input	Async	Transceiver primitive-based clock source used to derive and buffer TXUSRCLK and TXUSRCLK2 outputs. Width = 1

Related Information

Designing with the Core

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 Designing with the Core.

The receiver user clocking network helper block ports described in the following are present on the Wizard core instance when it is configured to locate the receiver user clocking network helper block in the core.

Table 9: **Receiver User Clocking Network Helper Block Ports**

Port Name	I/O	Clock	Description
gtwiz_userclk_rx_reset_in	Input	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		Transceiver primitive-based clock source used to derive and buffer the RXUSRCLK and RXUSRCLK2 outputs.
gtwiz_userclk_rx_usrclk_out	Output		Drives RXUSRCLK of transceiver dual primitives. Derived from gtwiz_userclk_rx_srcclk_in/ out, buffered and divided as necessary by BUFG_GT primitive.
gtwiz_userclk_rx_usrclk2_out	Output		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	gtwiz_userclk_rx_usrclk2_out	Active-High indication that the clocking resources within the helper block are not held in reset.
gtwiz_userclk_rx_active_in	Input	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.
gtwiz_userclk_rx_srcclk_in	Input	N/A	Transceiver primitive-based clock source used to derive and buffer RXUSRCLK and RXUSRCLK2 outputs.

Related Information

Designing with the Core

Chapter 4

Designing with the Core

This chapter includes guidelines and additional information to facilitate designing with the Virtex® UltraScale+™ FPGAs GTM 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. For more information see *UltraScale FPGAs GTM Transceivers User Guide* (UG581).

The 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 helper block 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.

Note: For this release version of GTM Wizard IP, some static location selections of the helper blocks have been made which align with the most common use cases. These selections will be enhanced to provide user input choice in future release of the GTM Wizard IP.

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.

Use the Example Design

Each instance of the GTM Wizard core created by the Vivado design tool is delivered with an example design that can be implemented in a device and then simulated. This design can be used as a starting point for your own design or can be used to sanity-check your application in the event of difficulty. See the Example Design content for information about using and customizing the example designs for the core.

Registering Signals

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

Recognize Timing Critical Signals

The constraints provided with the example design identify the critical signals and timing constraints that should be applied.

Make Only Allowed Modifications

You should not modify the core. Any modifications can have adverse effects on system timing and protocol compliance. Supported user configurations of the core can only be made by selecting the options in the customization IP dialog box when the core is generated.

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:

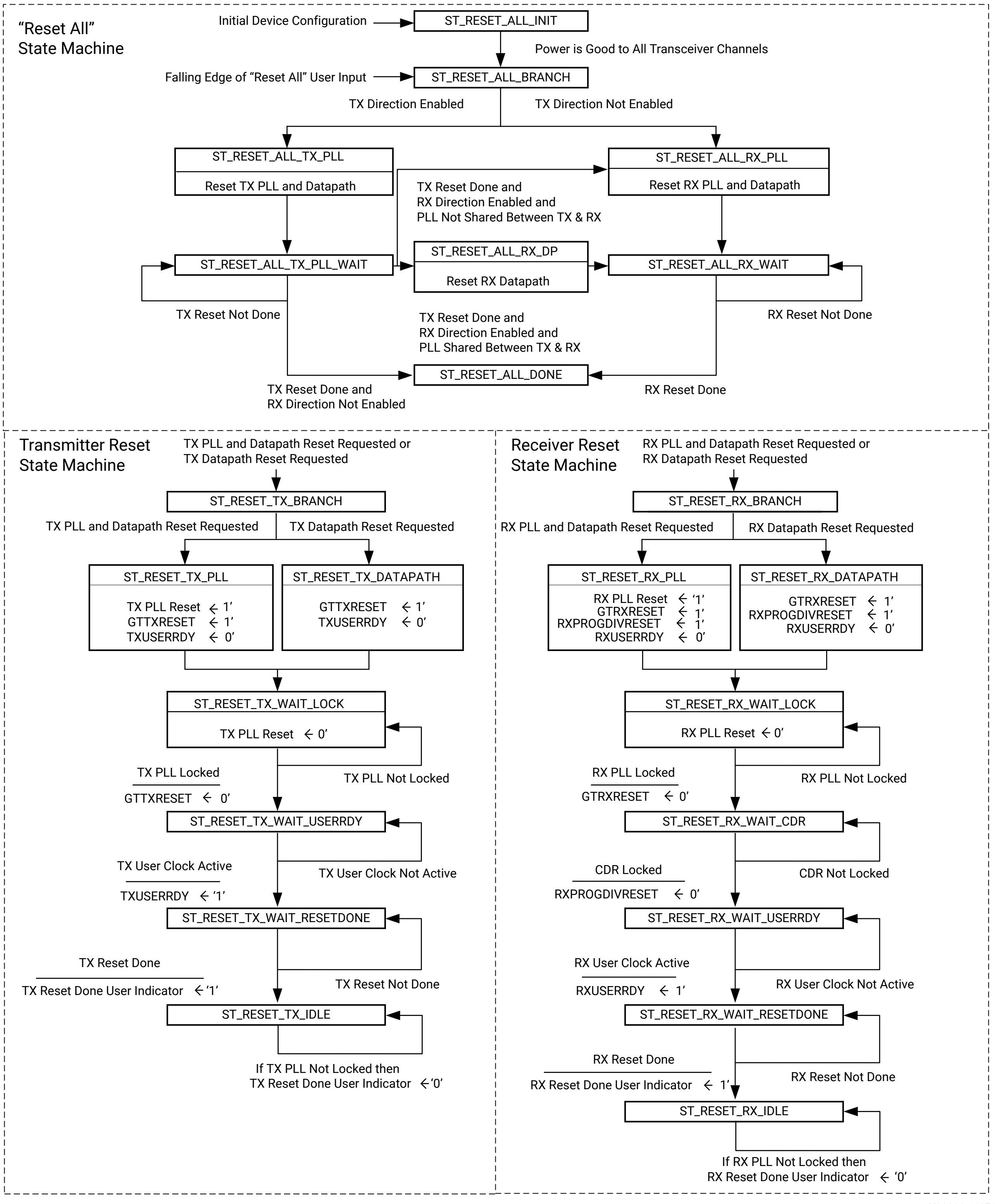
- **Transmitter reset state machine:** Resets the transmitter PLL and/or the transmitter datapath of all transceiver primitives, and indicates their completion.
- **Receiver reset state machine:** Resets the receiver PLL and/or the receiver datapath of all transceiver primitives, and indicates their 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 or by the "reset all" state machine using the 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 required.

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. The following figure illustrates the three reset controller helper block finite state machines and the reset sequences they control.

Figure 2: **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 instantiated by the core instance that are used to clock the transmitter datapath are reset in response to this input. After all of 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 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 LCPLL is 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. As both transmitter and receiver datapaths are clocked by LCPLL resources, assertion of either of those two inputs would reset the shared LCPLL of each transceiver Dual, 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 the previous figure 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.

***Note*:** A normal reset sequence would be TX and PLL reset followed by RX data path reset. If the TX and PLL is reset, it needs to be followed by the RX data path if the RX is to be used. If the RX and PLL is reset, it needs to be followed by the TX reset if the TX is to be used. In case of TX looping back to RX kind of designs, note that TX reset should be done before RX reset.

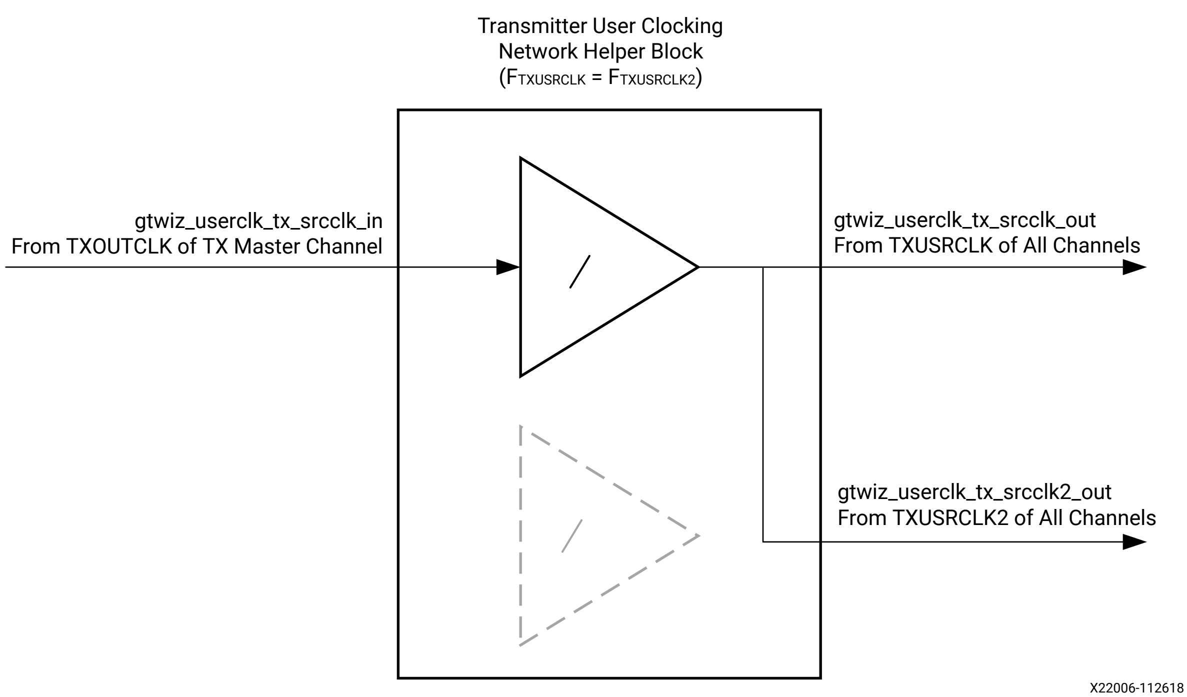
Transmitter User Clocking Network Helper Block

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 channels.

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.

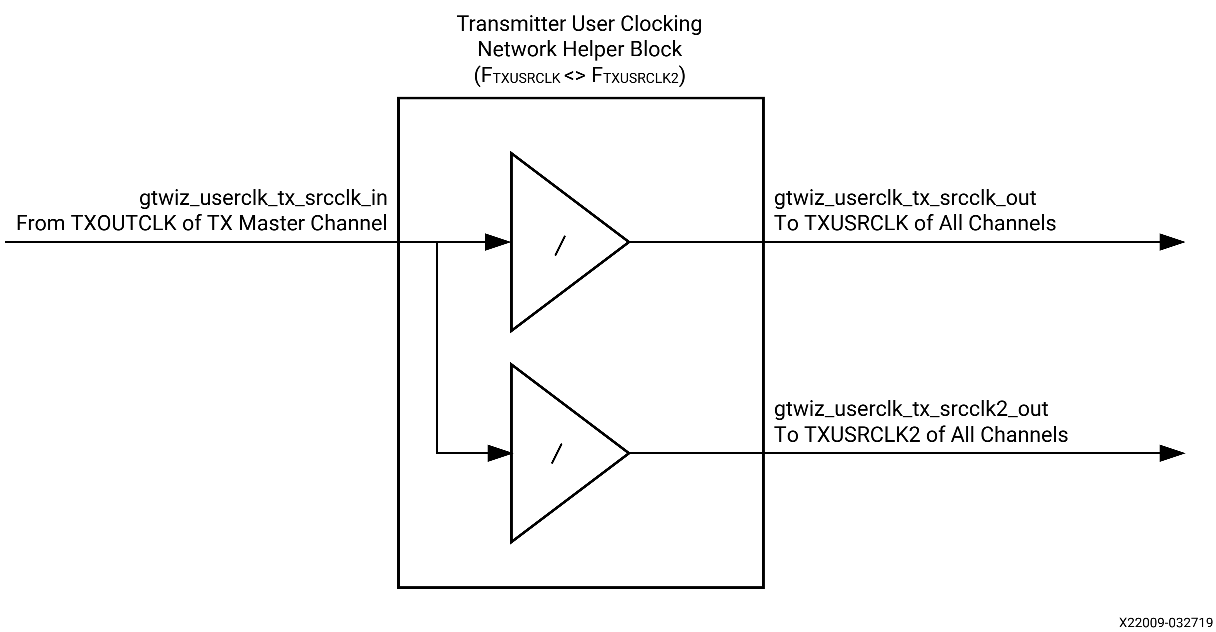
As shown in the following figure, if the TXUSRCLK and TXUSRCLK2 frequencies are identical (which is the case when the transmitter user data width is narrower than or equal to the size of the internal data width), then only a single BUFG_GT is instantiated within the helper block. This buffer drives both 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: **Transmitter User Clocking Network Helper Block (with One BUFG_GT)**



As shown in the above figure, 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. 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.

Figure 4: **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 `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 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.

For complete documentation on clocking the transceiver primitives, see *UltraScale FPGAs GTM Transceivers User Guide* (UG581).

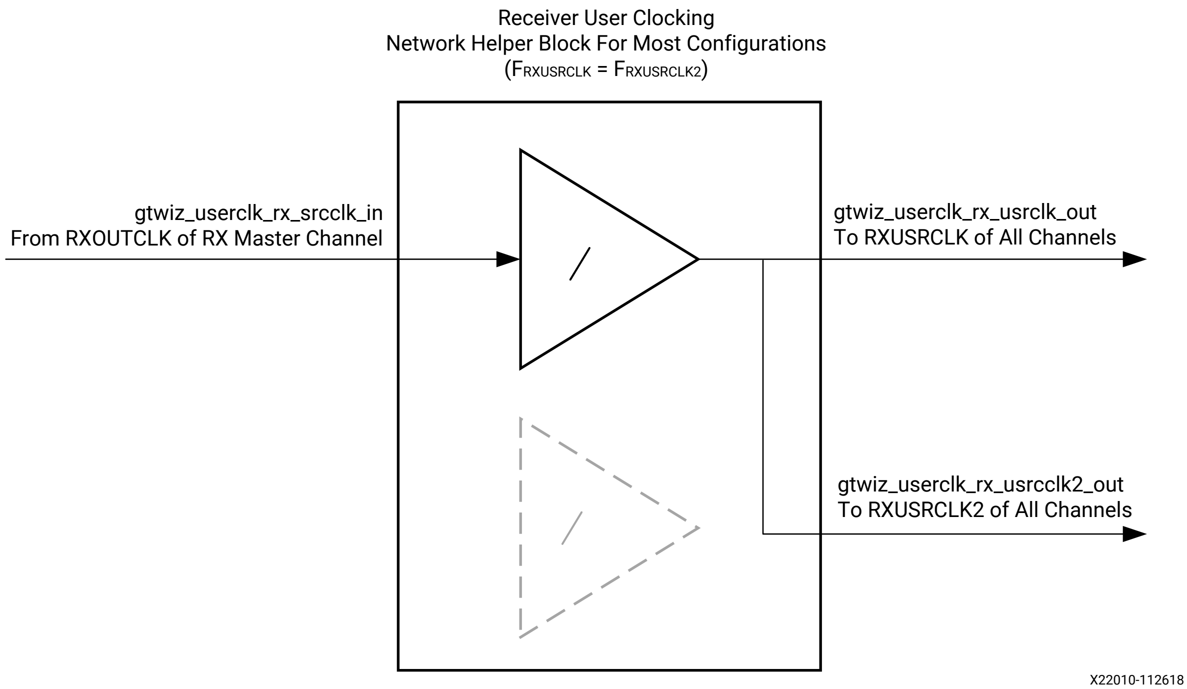
Receiver User Clocking Network Helper Block

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

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.

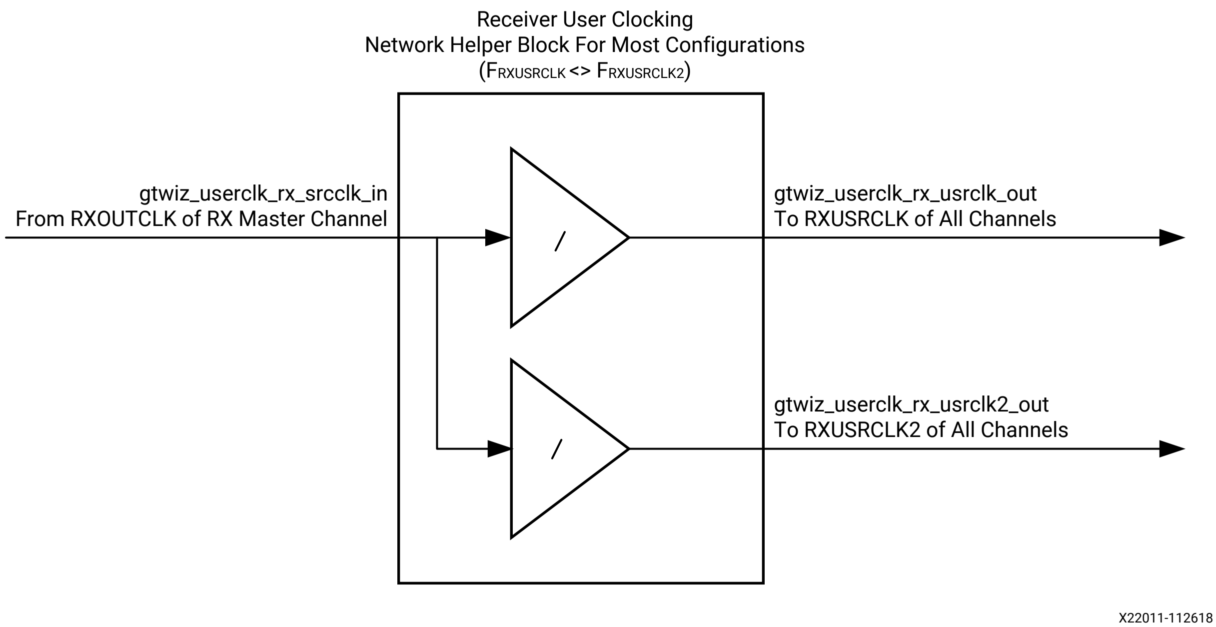
As shown in the following figure, if `RXUSRCLK` and `RXUSRCLK2` frequencies are identical (which is the case when the receiver user data width is narrower than or equal to the size of the internal data width), then only a single BUFG_GT is instantiated within the helper block. This buffer drives both `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 channels. The helper block configures the BUFG_GT to divide the source clock down to the correct user clock frequency as required.

Figure 5: **Receiver User Clocking Network Helper Block (with One BUFG_GT)**



As shown in the following figure, 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 channels. 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 channels.

Figure 6: **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.

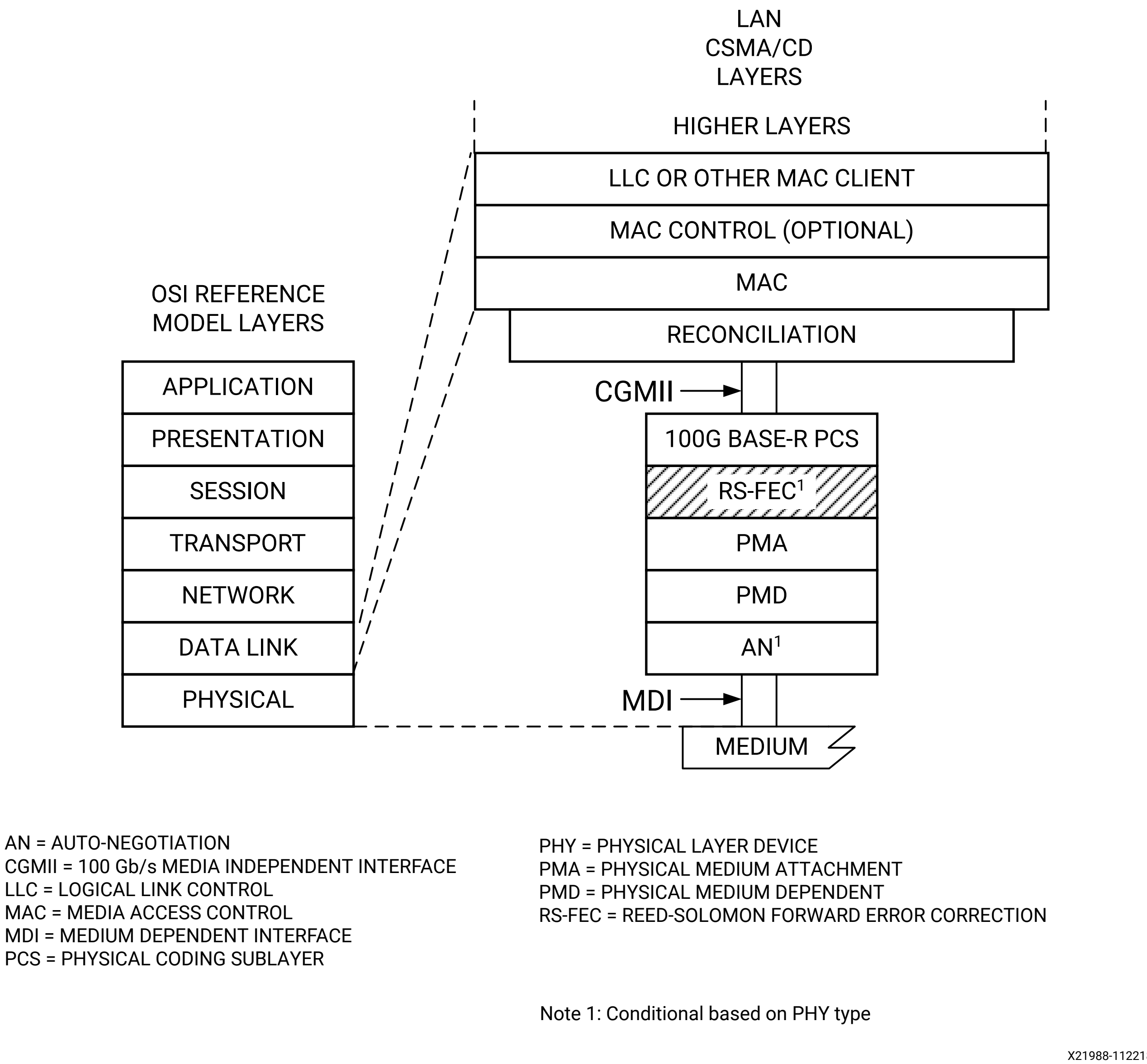
If additional clock signals or related ports are required for your application, you can enable the relevant ports on the core instance through the optional ports interface during IP customization. For a description of all receiver user clocking network helper block ports, see Chapter 3: Product Specification. For complete documentation on clocking the transceiver primitives, see *UltraScale FPGAs GTM Transceivers User Guide* (UG581).

Transcode Helper Block

This block implements the transmit and receive transcode and alignment marker, the removal/insertion, and mapping functions for 100G and 50G Ethernet with KP4 FEC. It also implements a simplified transmit alignment lock function like that found in the existing Xilinx® soft RS-FEC IP for 100G and 50G Ethernet. The transcode block supports the latest version of the standards listed in: *IEEE Standard for Ethernet* (IEEE Std 802.3-2015).

RS-FEC for 100G Ethernet is defined in Clause 91 of *IEEE Standard for Ethernet* (IEEE Std 802.3-2015). The hard RS-FEC inside the GTM_DUAL supports the RS (544,514) *KP4* variant only. The transcoding functions are not included in the hard block, and hence will be implemented in the FPGA logic by this block. A conceptual overview of the RS-FEC datapath in the context of the GTM_DUAL and Transcode helper block is shown in the following figure.

Figure 7: **Relationship of RS-FEC Layer to the ISO/IES Open System Interconnection Reference Model**



Port Descriptions

The transcode IP block's port definitions are shown in the following table. The width of the data buses varies depending on whether the block is configured in 50GE mode or 100GE mode. The `stat_rx_err_count_inc` input is driven by the GTM_DUAL's FEC hard block and allows the counting of the symbol errors in order to detect and implement the hi-SER state and associated logic.

Table 10: **Transcoder Helper Block Port Descriptions**

Port Name	Width (50G)	Width (100G)	I/O	Description
tx_din	132	320	Input	Transmit path data input This signal is handled internal to the GTM Wizard IP and it interfaces the GTM_DUAL UNISIM.
tx_din_start	1	1	Input	Start of codeword on tx_din This signal is handled internal to the GTM Wizard IP and it interfaces the GTM_DUAL UNISIM.
tx_dout	160	320	Output	Transmit path data output This signal is handled internal to the GTM Wizard IP and it interfaces the GTM_DUAL UNISIM.
rx_din	160	320	Input	Receive path data input This signal is handled internal to the GTM Wizard IP and it interfaces the GTM_DUAL UNISIM.
rx_din_start	1	1	Input	Start of codeword on rx_din This signal is handled internal to the GTM Wizard IP and it interfaces the GTM_DUAL UNISIM.
rx_din_is_am	1	1	Input	Indicates alignment markers present on rx_din.
rx_din_flags	4	4	Input	Indicates status of data on rx_din This signal is handled internal to the GTM Wizard IP and it interfaces the GTM_DUAL UNISIM. [0]: No error exists at RX FEC output [1]: Error exists at RX FEC output [2]: No error corrected at chien search [3]: Error corrected at chien search
rx_dout	132	320	Output	Receive path data output This signal is handled internal to the GTM Wizard IP and it interfaces the GTM_DUAL UNISIM.
stat_rx_err_count_inc	4	4	Input	Symbol error count from RS decoder This signal is handled internal to the GTM Wizard IP and it interfaces the GTM_DUAL UNISIM.

Table 10: **Transcoder Helper Block Port Descriptions** *(cont'd)*

Port Name	Width (50G)	Width (100G)	I/O	Description
ctl_rx_bypass_correction	1	1	Input	Correction bypass configuration This is exposed as the top-level port as "gtm_transcode_ctl_rx_bypass_correction"
ctl_rx_bypass_indication	1	1	Input	Indication bypass configuration This is exposed as the top-level port as "gtm_transcode_ctl_rx_bypass_indication"
stat_rx_hi_ser	1	1	Output	Hi-SER status This signal is handled internal to the GTM Wizard IP and it interfaces the GTM_DUAL UNISIM.
stat_rx_tcd_lock	1	1	Output	TX block lock status (100G mode only) This is exposed as the top-level port as "gtm_transcode_stat_rx_tcd_lock"
stat_tx_pcs_am_lock	1	1	Output	TX AM lock status This is exposed as the top-level port as "gtm_transcode_stat_tx_pcs_am_lock"
gtm_transcode_bypass_rx_core	1	1	Input reserved port, tie-off to 0	Future enhancement for GTM Switchable Ethernet configurations
gtm_transcode_bypass_tx_core	1	1	Input reserved port, tie-off to 0	Future enhancement for GTM Switchable Ethernet configurations

For more information, see AR 72110.

Supported Modes

The supported operating sub-modes for 100GE and 50GE are summarized in the following table. In all modes where virtual lane alignment markers and frame lengths are defined, these are configurable via core parameters. Some restrictions are imposed by the FEC layer, for example, frame length must be such that the alignment markers always fall at the start of a FEC codeword. This means that in 100GE mode, the length should be a multiple of 4, and in 50GE mode, the length should be a multiple of 20. A full set of status flags is provided by the RS-FEC block to enable the erroring of 66b sync headers by the RX transcoder in the case of an uncorrected codeword.

Table 11: **100GE and 50GE Supported Sub-Modes**

Rate	Indication Bypass	Correction Bypass	Indication and Correction Bypass
100G	Full support as per standard	Full support as per standard	Full support (although not required by standard)

Table 11: **100GE and 50GE Supported Sub-Modes** *(cont'd)*

Rate	Indication Bypass	Correction Bypass	Indication and Correction Bypass
50G	Full support as per standard	Full support as per standard	Full support (although not required by standard)

The nominal operating mode against standard line rates is given in the following table. However, the GTM Wizard IP GUI provides the option to modify the line rates for over/under sampling modes. If this option is selected, ensure the standard ratio of GTM clock to MAC clock rates are matched as shown in the following table. To illustrate this, the GTM Wizard IP example design instantiates a clocking wizard block for reference.

Table 12: **Nominal Operating Mode -v- Standard Line Rates**

FEC Mode	Nominal SERDES Clock Rate (MHz)	Nominal MAC Clock Rate (MHz)
100GE KP4 (53.125 Gb/s X2)	332.03125	322.265625
50GE KP4(53.125 Gb/s X1)	332.03125	390.625

N PCS lanes are multiplexed to form the data stream that is fed in to the Transcoder, where N=20 in a 100GE system and N=4 in a 50GE system. Each PCS lane contains an alignment marker once in every FEC_VL_LENGTH 66b blocks. This is because the PCS layer processes 66b blocks of data. The units used to specify GUI input for FEC_VL_LENGTH_CWS and FEC_VL_LENGTH_CWS_RX is the number of codewords - whose typical configuration values are 4096@100GE and 1024@50GE.

The functionality and high-level architecture required for the soft Ethernet transcoding IP block is described in the following figures. The functions managed in the Transcode block include Lane Block Lock, Alignment Lock, Alignment Marker Removal, 64b/66b to 256b/257b Transcoding, and related Alignment marker mapping and insertion. The Transcoding logic helper block has a fixed, deterministic latency, excluding the latency of the CDC crossing FIFO.

Figure 8: **High-Level Architecture for 100G Datapath**

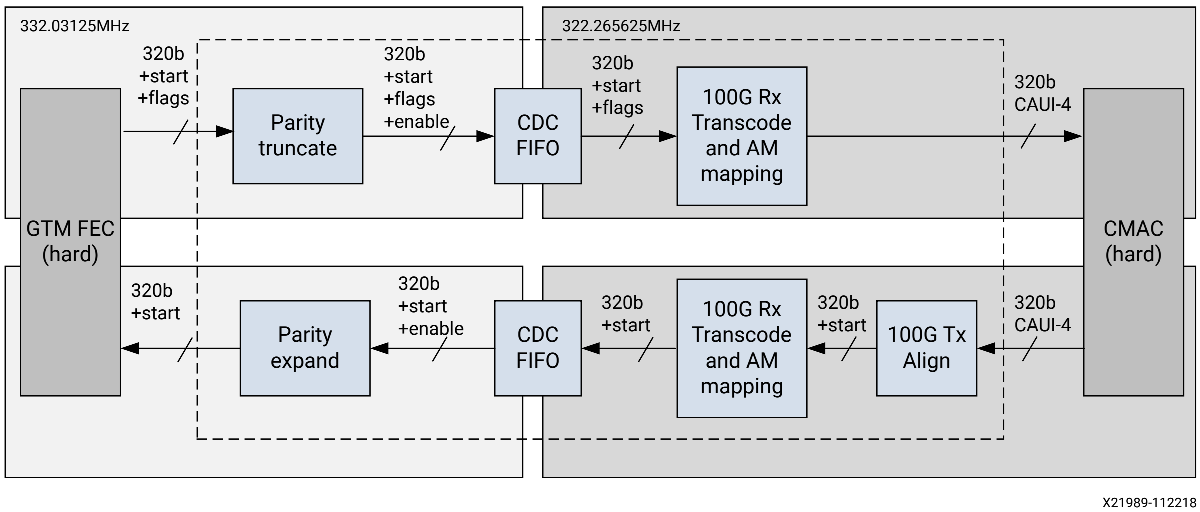
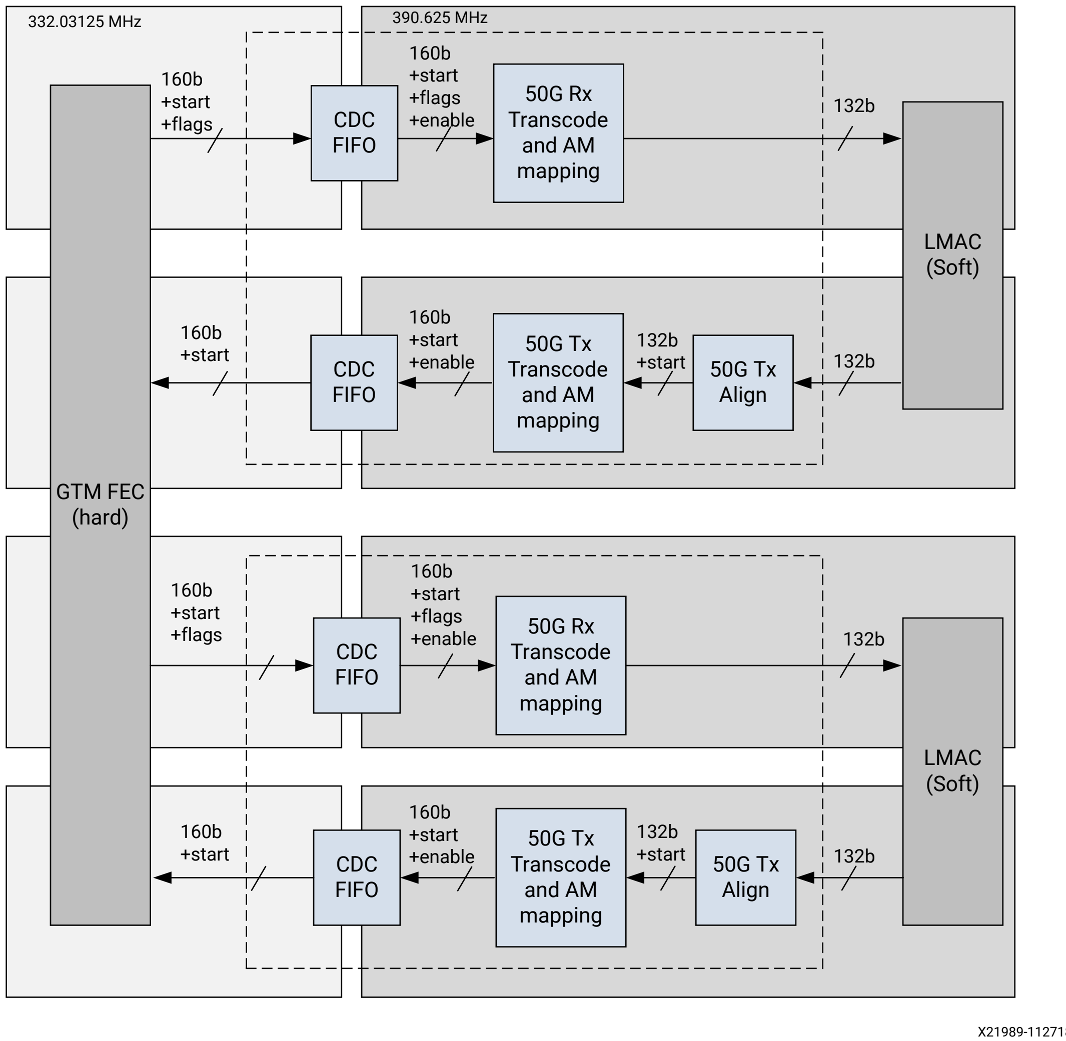


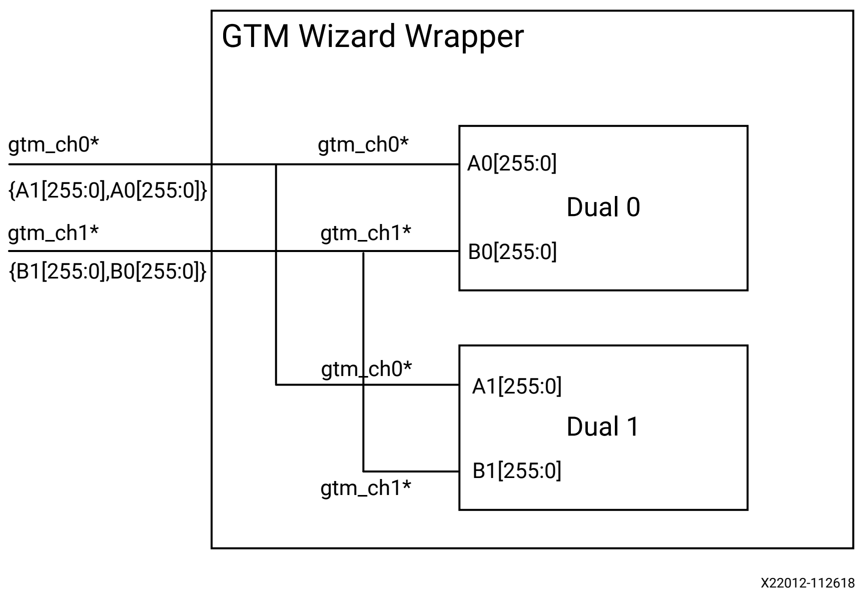
Figure 9: **High-Level Architecture for 50G Datapath**



Designing With Multi-Duals

When the GTM Wizard core is generated with one dual, it generates two datapath signals - one for Channel 0 (`gtm_ch0*`) and another for Channel 1 (`gtm_ch1*`). Datapath signals are categorized as channels to suit most use cases. For multi-dual designs, the wizard generates two sets of datapath signals (`gtm_ch0*` and `gtm_ch1*`) similar to single dual design where `Channel 0` signals of all duals are concatenated and `Channel 1` signals of all duals are concatenated. A multi-dual wizard wrapper datapath connection is shown in the following figure. Care must be taken to map these datapath signals to a multi-dual user design.

Figure 10: **Two Duals GTM Wizard Wrapper**



Chapter 5

Design Flow Steps

This section 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 IP Subsystems using IP Integrator* (UG994)
- *Vivado Design Suite User Guide: Designing with IP* (UG896)
- *Vivado Design Suite User Guide: Getting Started* (UG910)
- *Vivado Design Suite User Guide: Logic Simulation* (UG900)

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

> **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).*

1. In the Vivado Design Suite, create a new project or open an existing project that is configured to target one of the supported Virtex® UltraScale+™ devices which has a GTM transceiver.
2. Double-click the selected IP or select the Customize IP command from the toolbar or right click menu.

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

Figures in this chapter are illustrations of the Vivado IDE. The layout depicted here might vary from the current version.

Vivado IDE Customization Parameters

The Wizard IP parameters are organized in six tabs:

- **Basic:** Provides customization options for fundamental transceiver features, including transceiver type, and transmitter and receiver settings.
- **Physical Resources:** Provides reference clock source, Master channel selection and Duals selection.
- **Optional Features:** Provides extensive configuration options for optional or advanced features, if appropriate for your application.
- **FEC Options:** Provides FEC related options for customization if PAM4 and valid 160/80 or 80/80 selections for data widths are made.
- **AM_50G:** Provides customization options for the Alignment markers used for 50GE configuration for each of the slices (lanes).
- **AM_100G:** Provides customization option for the Alignment markers used for 100GE configuration.

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 *UltraScale FPGAs GTM Transceivers User Guide* (UG581).

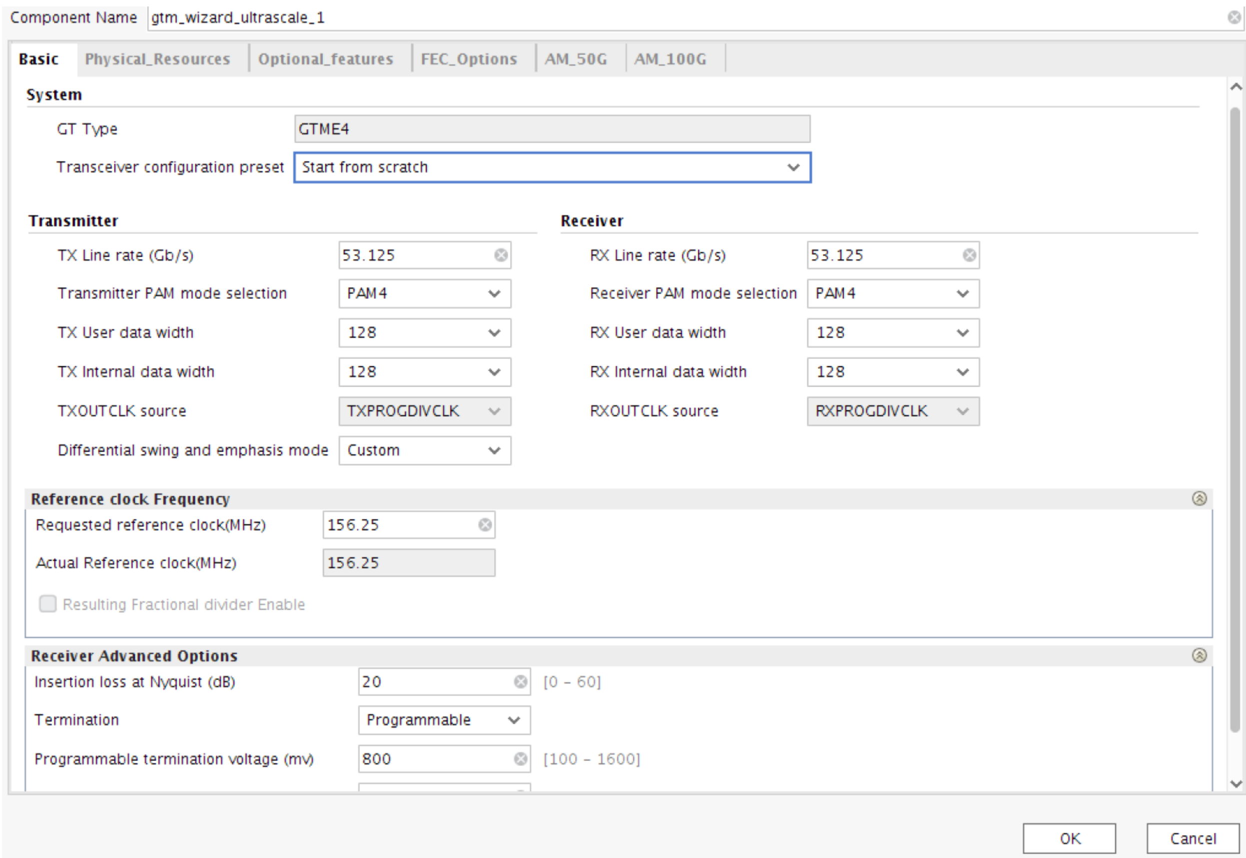
Component Name

The name of the generated IP is set in the Component Name field. The default name is gtmwizard_ultrascale_0. This must be set to a name that is unique within your project.

Basic Tab

IP customization options in the Basic tab are described in the following subsections. Selections apply to each enabled transceiver channel in the core instance. For full details on available choices for each customization option, see *UltraScale FPGAs GTM Transceivers User Guide* (UG581).

Figure 11: **Basic Tab**



Overall system settings are customized using the options in this section. Note that the GUI design is top-down priority.

- **Transceiver type:** Only GTME4 is supported. This is for display purposes only to differentiate with the UltraScale™ GT Wizard IP; For GTY or GTH transceivers, users need to use the `gtwizard_ultrascale` IP. Refer to the *UltraScale FPGAs Transceivers Wizard LogiCORE IP Product Guide* (PG182).
- **Trasnsceiver configuration preset:** This provides a default start from scratch option wherein you can select the target configurations manually, or alternatively select a list of pre-validated presets available in the drop down menu. Note that you would still have to review the configuration as per their target system requirements, as the GTM parent IP's could be changing few additional user parameters. For example they could be changing INS_LOSS_NYQ parameter as per the target physical medium being used.
- **TX Line rate (Gb/s):** Enter the transmitter line rate in gigabits per second. The available range is limited by the selected device. Refer to the relevant FPGA data sheet for more information on valid line rate ranges.
- **Transmitter PAM mode selection:** Enter the transmitter modulation value from drop down options. These values have dependency on line rate selected.

- **RX line rate (Gb/s):** Enter the receiver line rate in gigabits per second. It should be same as TX Line rate. Other options are not supported in this release. The available range is limited by the selected device.
- **Receiver PAM mode selection:** Enter the receiver modulation value from drop down options. These values have dependency on line rate selected.

Data width selection

- **TX User data width:** Also known as external data width. Select the desired bit width for the transmitter user data interface of each serial transceiver dual. Possible options are 64, 80,128, 160 and 256 but available choices can be limited by selected line rate and PAM mode selections.
- **RX User data width:** Also known as external data width. Select the desired bit width for the receiver user data interface of each serial transceiver dual. Possible options are 64, 80,128, 160 and 256 but available choices can be limited by selected line rate and PAM mode selections.
- **TX Internal data width:** Select the desired bit width for the internal transmitter datapath of each serial transceiver dual. Possible options are 64, 80 and 128 but available choices can be limited by selected line rate, PAM mode selections and TX user data width.
- **RX Internal data width:** Select the desired bit width for the internal receiver datapath of each serial transceiver dual. Possible options are 64, 80 and 128 but available choices can be limited by selected line rate, PAM mode selections and RX user data width.
- **TXOUTCLK source:** Select the internal clock source for the TXOUTCLK port of each serial transceiver primitive. TXPROGDIVCLK is the OUTCLK source for this release.
- **RXOUTCLK source:** Select the internal clock source for the RXOUTCLK port of each serial transceiver primitive. RXPROGDIVCLK is the OUTCLK source for this release.
- **Differential swing and emphasis mode:** Select the transmitter driver mode. Selection determines the set of ports that control the transmitter driver swing and cursors.
- **Reference clock Configurations:** Enter a requested reference clock (MHz) value and it auto-infers fractional N feedback divider options (if required) and populates the Actual Reference Clock (MHz) field that should be applied on GT Wizard's reference clock pin. In most cases, the requested frequency is available for selection. The calculation also populates the Fractional part of feedback divider field with the numerator of the fractional part of the feedback divider used to clock the datapath.
- **Actual Reference clock (MHz):** This is a read-only field. The frequency displayed here should be applied to achieve the selected line rate, both on hardware and for simulation purposes.

Receiver Advanced Options

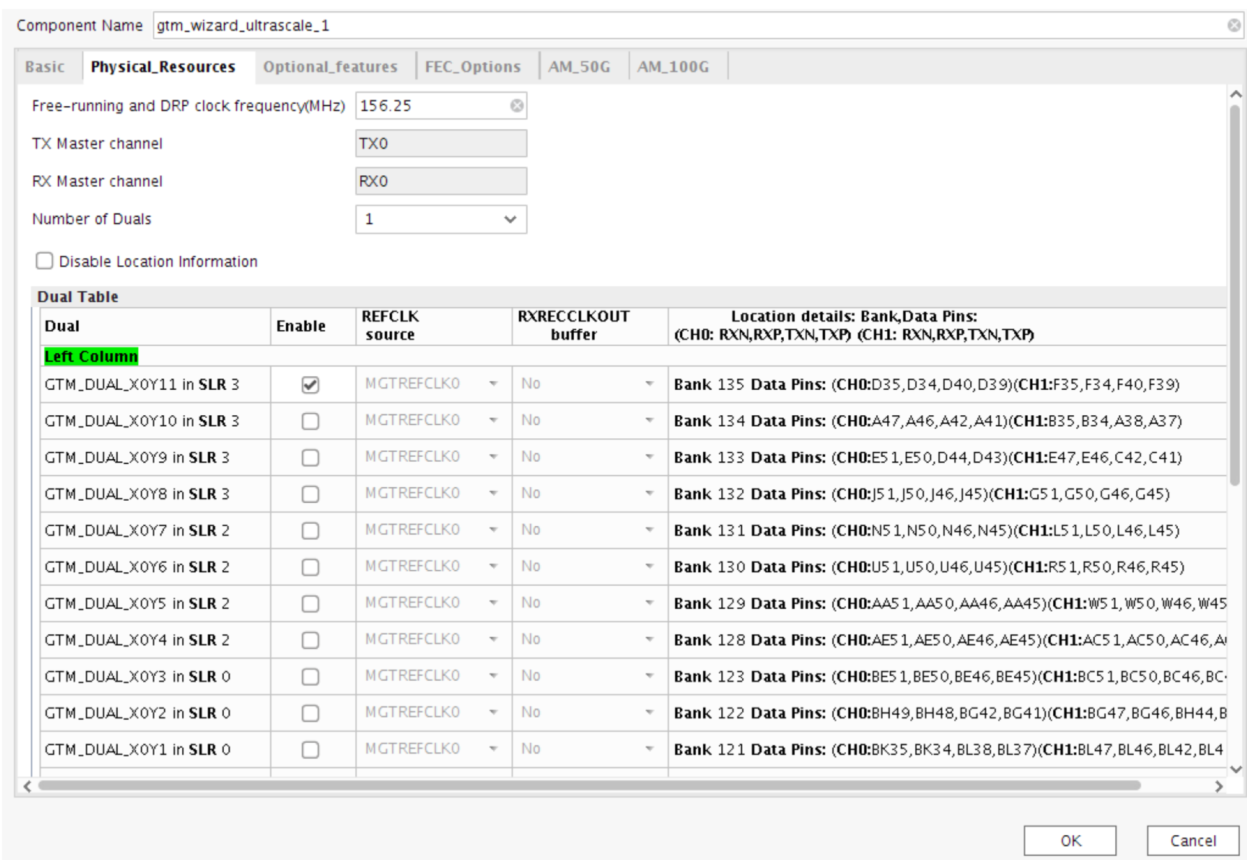
- **Insertion loss at Nyquist (dB):** Specify the insertion loss of the channel between the transmitter and receiver at the Nyquist frequency in dB.

- **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.

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. For more information, see *UltraScale FPGAs GTM Transceivers User Guide* (UG581). Users have the option to enable or disable location information in xdc by making the appropriate selection in the **Disable Location Information** field. The default selection is to not select location information.

Figure 12: **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.

> **RECOMMENDED:** *An accurate frequency is required to construct clock constraints and parameterize certain design modules.*

- **TX Master Channel and RX Master Channel:** Independently select the master transmitter and receiver channels from among all enabled transceiver channels. 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.

  ***Note*:** These options are unavailable in this release because the clocking helper block is present in the example design, and this can be configured as required.

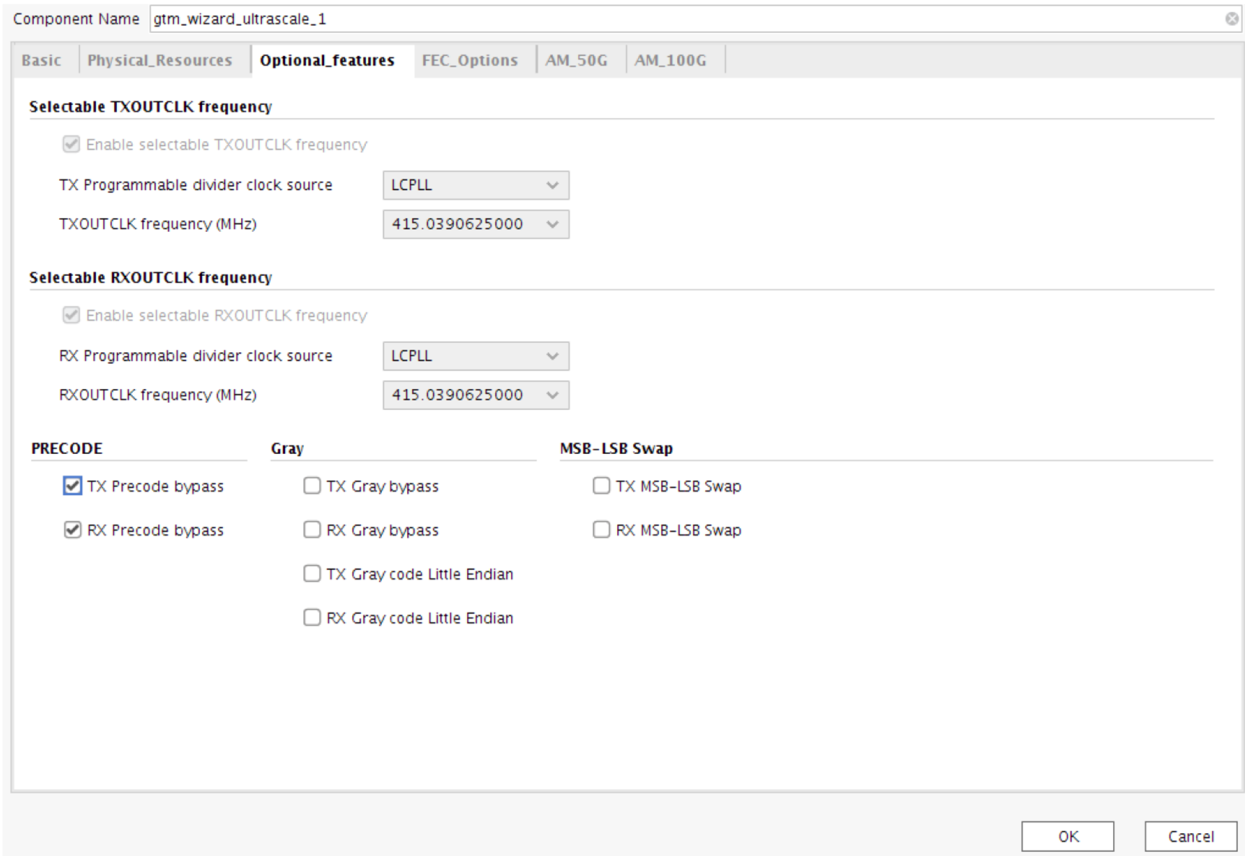
- **Number of Duals:** This allows you to select the number of duals. Refer to the following table for location specific information. Uncheck the option **Disable Location Information**, and see the list of available GTM_DUAL locations in the device and select them. Note that the MGT reference clock IBUFDS_GTM is instantiated outside the IP in theChapter 6: Example Design, hence the MGT reference clock selection in GUI is not provided unlike the one's from gtwizard_ultrascale_v1_7 IP. Note that for few line rates (refer *UltraScale FPGAs GTM Transceivers User Guide* (UG581)), sharing of MGT reference clock is not allowed, and it is recommended to use local MGT reference clock from same clock region as the GTM_DUAL.

- **BYPASS_GTM_CNTRL:** From 2018.3.1 version of the GTM Wizard IP, a new GTM controller block has been added to enable additional features in future releases. The default GTM Wizard behavior is to enable this block and is not expected to be bypassed.

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. For more information, see *UltraScale FPGAs GTM Transceivers User Guide* (UG581). The layout of the Optional Features tab is shown in the following figure.

Figure 13: **Optional Features Tab**



Selectable TXOUTCLK Frequency

- **Enable selectable TXOUTCLK frequency:** For this version of the GTM Wizard IP, this is always enabled, as `TXPROGDIVCLK` is the only source.
- **Programmable divider clock source:** LCPLL is the clock source used for the TX programmable divider.
- **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.

Selectable RXOUTCLK Frequency

- **Enable selectable RXOUTCLK frequency:** For this version of the GTM Wizard IP, this is always enabled, as `RXPROGDIVCLK` is the only source.
- **Programmable divider clock source:** LCPLL is the clock source for the RX programmable divider.

- **RXOUTCLK frequency (MHz):** Select from among the `RXOUTCLK` frequencies that can be generated by the RX programmable divider and are compatible with the core configuration and selected device.

PRECODE

- **TX Precode Bypass:** Enable this option to bypass TX Pre-Coder.
- **TX Precode Little Endian mode:** Enable this option to select TX Precode little endian mode. This is shown in the GUI as TX MSB-LSB Swap.
- **RX Precode Bypass:** Enable this option to bypass RX Pre-Coder.
- **RX Precode Little Endian mode:** Enable this option to select RX Precode little endian mode. This is shown in the GUI as RX MSB-LSB Swap.

Gray

- **TX Gray Bypass:** Enable this option to bypass TX Gray code.
- **TX Gray code Little Endian mode:** Enable this option to select TX Gray code little endian mode.
- **RX Gray Bypass:** Enable this option to bypass RX Gray code.
- **RX Gray code Little Endian mode:** Enable this option to select RX Gray code little endian mode.

FEC Options Tab

IP customization options in the FEC Options tab are described in the following subsections.

- **TX GT FEC Mode Enable:** Enable this option to select TX FEC block inside the `GTM_DUAL`.
- **TX GT FEC Mode:** Select the desired option amongst RAW, 50G, 100G, 50G BP, and 50G BP SCRAM.
- **TX FEC Transcode Logic Enable:** Enable this option to select the TX logic in the Transcode IP helper block instantiation.
- **RX GT FEC Mode Enable:** Enable this option to select RX FEC block inside the `GTM_DUAL`.
- **RX GT FEC Mode:** Select the desired option amongst RAW, 50G, 100G, 50G BP, and 50G BP SCRAM.
- **RX FEC Transcode Logic Enable:** Enable this option to select the RX logic in the Transcode IP helper block instantiation.

***Note*:** See the *UltraScale FPGAs GTM Transceivers User Guide* (UG581) for a detailed description of the various FEC modes and functionality.

Output Generation

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

Simulation

The Wizard example design test bench can be simulated to quickly demonstrate core and transceiver functionality. For more information, see 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).

See the *Vivado Design Suite Tutorial: Logic Simulation* (UG937) for additional references of using `xil_dut_bypass` module, for automated hierarchical reference port access in PAM4 Signal Inference in GTM Transceivers top for simulation.

Related Information

Test Bench

Chapter 6

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.

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 Test Bench, an included self-checking test bench simulates the example design in loopback, checking for link maintenance. The example design is also synthesizable.

***Note*:** Some asymmetric configurations are not directly supported for loopback testing, an equivalent partner instance may be required to be generated for testing.

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.*

RECOMMENDED: *As part of the Vivado version update/migration, Xilinx recommends that you manually upgrade the example design files.*

Related Information

Test Bench

Example Design Ports

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

Table 13: **Example Design Top-level Ports**

Name	Direction	Clock Domain	Description
refclk<i>_<j>_p	Input	N/A	Positive and negative inputs of the differential reference clock
refclk<i>_<j>_n	Input	N/A	
gtm_ch<i><j>_rxn_in	Input	Serial	Positive and negative inputs of the transceiver channel differential serial data receiver, where: <i>:0/1, <j>:dual* corresponding to the enabled duals.
gtm_ch<i><j>_rxp_in	Input	Serial	
gtm_ch<i><j>_txn_out	Output	Serial	Positive and negative inputs of the transceiver channel differential serial data transmitter, where: <i>:0/1, <j>:dual* corresponding to the enabled duals.
gtm_ch<i><j>_txp_out	Output	Serial	
hb_gtwiz_reset_clk_fr eerun_in	Input		Free-running clock, used by the example design and reset controller helper block. ***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	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_re set_in	Input	Async	Active-High signal used to reset the sticky link down indicator. Assumed to be de-bounced external to the device. 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.
link_status_out	Output	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	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

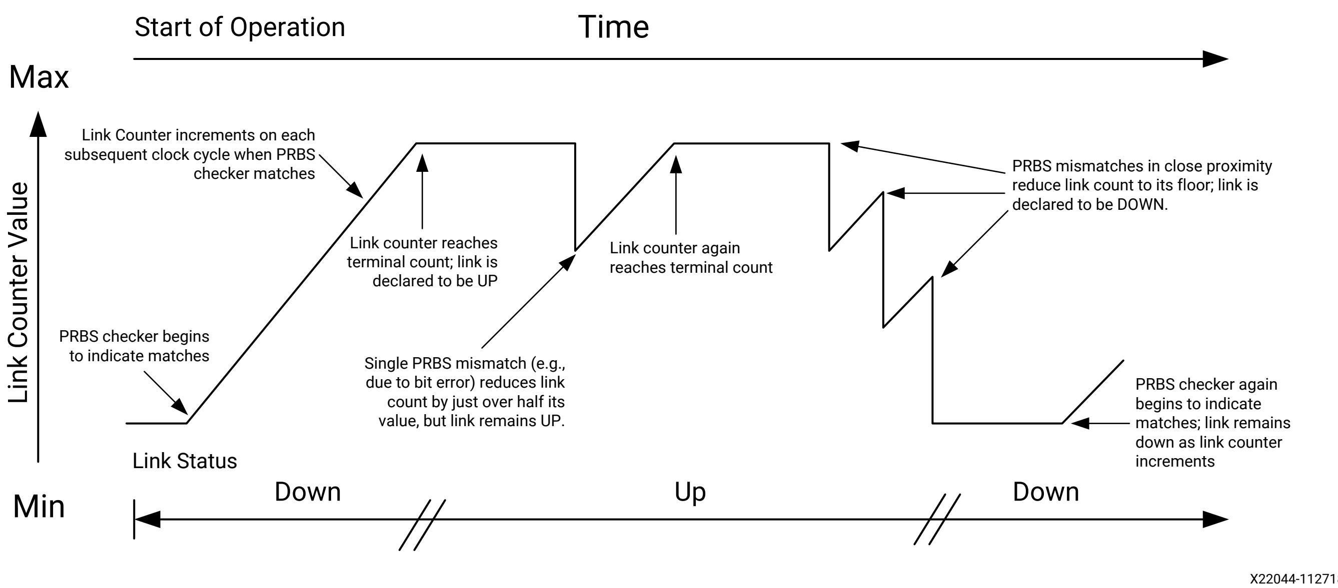
The Wizard example design contains link status logic that indicates the current state of the PRBS checkers across all transceiver channels while remaining tolerant of occasional mismatches such as infrequent bit errors. The example design also includes an initialization module that is a demonstration 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. The following figure illustrates the behavior of the link counter and resulting link status in response to various PRBS checker conditions.

Figure 14: **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.

Initialization Module

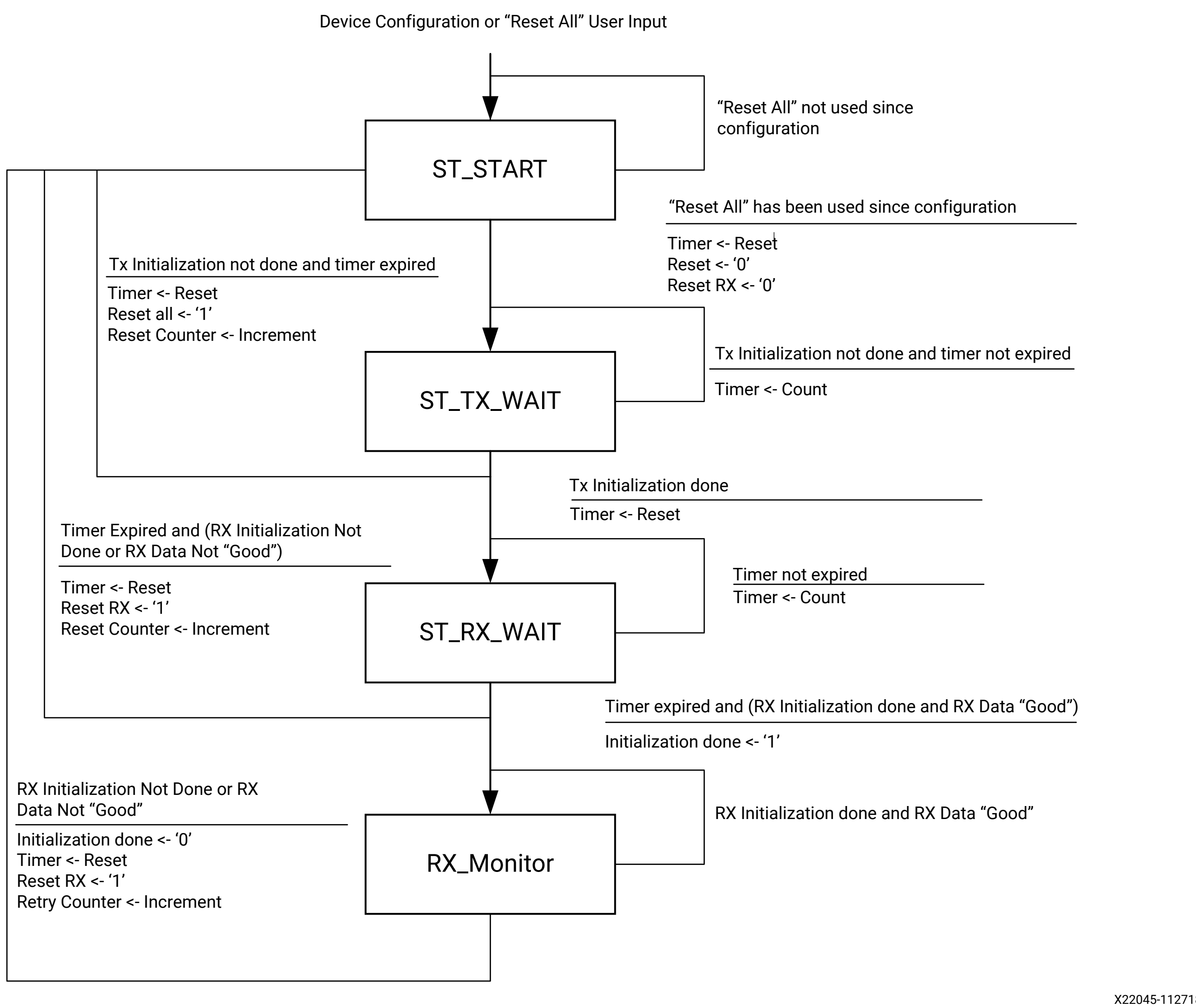
The Wizard example design contains a module that demonstrates how initialization logic can be constructed to interact with and enhance the reset controller helper block to assist with successful system bring-up. The example initialization logic monitors for timely transceiver resource reset completion, 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 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. The following figure illustrates the initialization module state machine.

Figure 15: **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.

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.*

Note: 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 if those resources were specified to be located in the example design during IP customization.

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

One or more 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_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.

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. Fundamentally, raw PRBS data is generated and checked. For FEC enabled use cases, a default PCS pattern is sent which does not test all the IEEE 802.3 specified patterns. Also only FEC lock from the GTM_DUAL is checked for arrival for test pass/fail criteria.
- 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 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.
- Example design needs an update for giving the port maps before bitstream generation, you need to consider the target board and update the location constraints accordingly.
- There are chances that the Chapter 6: Example Design may not meet the timing in all configurations depending on the speed grade and IP configuration, please refer to *UltraFast Design Methodology Guide for Xilinx FPGAs and SoCs* (UG949) and try different implementation/ synthesis strategies

Test Bench

This chapter contains information about the provided test bench in the Vivado® Design Suite. The Virtex UltraScale+ FPGAs GTM transceivers Wizard IP core 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 duals.

Simulating the Example Design

To simulate an instance of the GTM Wizard IP core, first open its example design as described in 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 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, see *Vivado Design Suite User Guide: Logic Simulation* (UG900). Also refer to Hierarchical access simulation tutorial from *Vivado Design Suite Tutorial: Logic Simulation* (UG937).

Related Information

Example Design

PAM4 Signal Inference in GTM Transceivers

The GTM_DUAL UNISIM models the Secure IP ports as integer declarations, while the GTM_DUAL UNISIM abstracts this as a logic wire for synthesis and placement tools. Because the GTM transceiver is targeted to have PAM4 signaling, which needs to model analog values of 0/1/2/3 (in case of PAM4) OR 0/3 (in case of NRZ), you need to hierarchically probe the internal variable in the UNISIM. These are modeled as `CH{0/1} _GTM{T/R}X{P/N}_integer` (for example, `CH0_GTMTXN_integer`). Refer to the hierarchical force mechanism used in the example design simulation top for references on how to access these for simulation purposes.With 2020.1 and later Vivado releases, a new feature is introduced where the tool generates a `xil_dut_bypass` module which takes care of the hierarchical access of `GTM_DUAL PAM4` serial ports for simulation purpose. If the default Vivado IP settings option is used, then when `launch_simulation` is clicked, the `xil_dut_bypass` module gets generated by querying the latest hierarchical reference of the `GTM_DUAL` unisim and gives the same for user view, this module can be instantiated in a system verilog based top file in simulation sources for port mapping of user logic. If you decides to avoid this and continue with the workarounds as shown below, then this option should be disabled in project settings before IP/Example design generation, so as to mimic the behavior of existing designs, where the requirement would be that the user designs manually query the hierarchical paths and use them. A code excerpt from the simulation top file is as shown below:

```
  integer gtm_ch0_n;
  integer gtm_ch0_p;
  always @(*)
  begin
    force gtm_ch0_n =
u_exdes_top.u_gtm_wiz_ip_top.inst.dual0.gtm_dual_inst.CH0_GTMTXN_integer;
    force gtm_ch0_p =
u_exdes_top.u_gtm_wiz_ip_top.inst.dual0.gtm_dual_inst.CH0_GTMTXP_integer;
    force
u_exdes_top.u_gtm_wiz_ip_top.inst.dual0.gtm_dual_inst.CH0_GTMRXN_integer =
gtm_ch0_n ;
    force
u_exdes_top.u_gtm_wiz_ip_top.inst.dual0.gtm_dual_inst.CH0_GTMRXP_integer =
gtm_ch0_p ;
  end
```

Figure 16: **Code Excerpt from Example Design Top File**

```
// ==========================================================================================
// Since GTM_DUAL models PAM4 signalling and top level ports are declared as logic wires,
// Hierarchical access and forcing of the internal signals which are modelled as integer is required
// ==========================================================================================
  integer gtm_ch0_n;
  integer gtm_ch0_p;
  integer gtm_ch1_n;
  integer gtm_ch1_p;

  always @(*)
  begin
    force gtm_ch0_n = u_exdes_top.u_gtm_wiz_ip_top.inst.dual0.gtm_dual_inst.CH0_GTMTXN_integer;
    force gtm_ch0_p = u_exdes_top.u_gtm_wiz_ip_top.inst.dual0.gtm_dual_inst.CH0_GTMTXP_integer;
    force gtm_ch1_n = u_exdes_top.u_gtm_wiz_ip_top.inst.dual0.gtm_dual_inst.CH1_GTMTXN_integer;
    force gtm_ch1_p = u_exdes_top.u_gtm_wiz_ip_top.inst.dual0.gtm_dual_inst.CH1_GTMTXP_integer;

    force u_exdes_top.u_gtm_wiz_ip_top.inst.dual0.gtm_dual_inst.CH0_GTMRXN_integer = gtm_ch0_n ;
    force u_exdes_top.u_gtm_wiz_ip_top.inst.dual0.gtm_dual_inst.CH0_GTMRXP_integer = gtm_ch0_p ;
    force u_exdes_top.u_gtm_wiz_ip_top.inst.dual0.gtm_dual_inst.CH1_GTMRXN_integer = gtm_ch1_n ;
    force u_exdes_top.u_gtm_wiz_ip_top.inst.dual0.gtm_dual_inst.CH1_GTMRXP_integer = gtm_ch1_p ;
  end
```

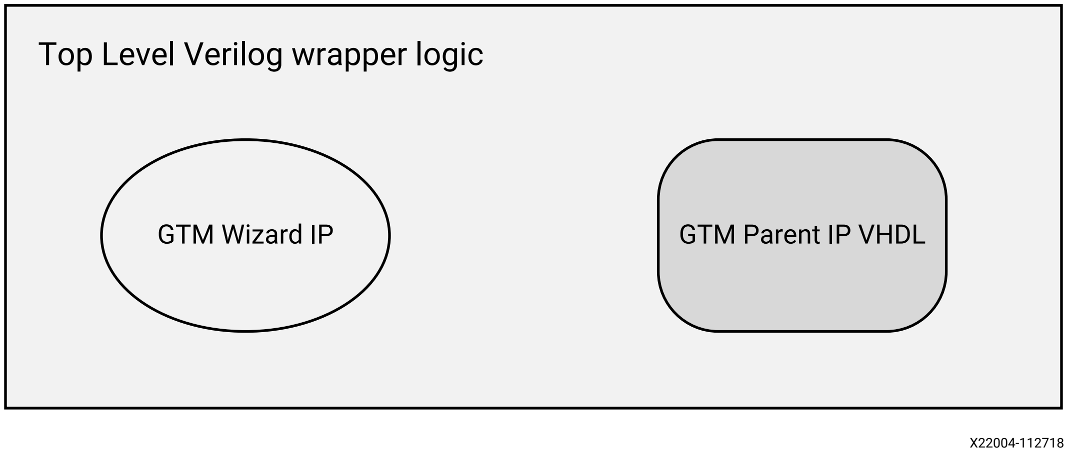
VHDL GTM Transceiver Parent IP Simulation Workarounds

The integer internal ports in `GTM_DUAL.sv` require hierarchical access. Some simulators do not support hierarchical access across language boundaries. Because the GTM Wizard IP is a Verilog-only deliverable, it is not feasible to run simulations by instantiating directly in VHDL designs. Following are two suggested workarounds for getting around this problem.

Method 1

Instantiate the GTM Wizard IP in the top-level Verilog and continue using the same model of hierarchical referencing as used in the wizard IP example design. The top-level stimulus and checker logic will be replaced by GTM parent IP VHDL designs. This method avoids accessing across hierarchical boundaries.

Figure 17: **Method 1**



Method 2

Create an additional Verilog wrapper logic on top of the GTM Wizard IP. Then create a dummy port with integer in the declaration as shown below, and abstract the requirement to have hierarchical access from this level.

***Note*:** This additional logic is to be controlled under synthesis translate on/off pragma so that the synthesis behavior remains the same.

Following is an example code snippet:

```
//--Port declaration of the GTM Wizard IP  wrapper top in user design
input  gtm_ch0_rxp_in ,
output gtm_ch0_txp_in,
………….
………….
//pragma translate_off
input integer gtm_ch0_rxp_integer,
output integer gtm_ch0_txp_integer,
……
……
//pragma translate_on
………….
………….
//--Logic inside the module definition
//pragma translate_off
always @(*)
begin
   force  u_gtm_wiz_ip_top.inst.dual0.gtm_dual_inst.CH0_GTMRXP_integer =
gtm_ch0_rxp_integer;
   force  gtm_ch0_txp_integer =
u_gtm_wiz_ip_top.inst.dual0.gtm_dual_inst.CH0_GTMTXP_integer;
end
//pragma translate_on
```

Figure 18: **Example Code**

```
output          gtm_ch0_txn_out,
output          gtm_ch1_txp_out,
output          gtm_ch1_txn_out,
//pragma translate_off
input   integer    gtm_ch0_rxp_integer,
input   integer    gtm_ch0_rxn_integer,
input   integer    gtm_ch1_rxp_integer,
input   integer    gtm_ch1_rxn_integer,
output  integer    gtm_ch0_txp_integer,
output  integer    gtm_ch0_txn_integer,
output  integer    gtm_ch1_txp_integer,
output  integer    gtm_ch1_txn_integer,
//pragma translate_on
```

```
//pragma translate_off
always @(*)
begin
  force gtm_ch0_txn_integer = u_gtm_wiz_ip_top.inst.dual0.gtm_dual_inst.CH0_GTMTXN_integer;
  force gtm_ch0_txp_integer = u_gtm_wiz_ip_top.inst.dual0.gtm_dual_inst.CH0_GTMTXP_integer;
  force gtm_ch1_txn_integer = u_gtm_wiz_ip_top.inst.dual0.gtm_dual_inst.CH1_GTMTXN_integer;
  force gtm_ch1_txp_integer = u_gtm_wiz_ip_top.inst.dual0.gtm_dual_inst.CH1_GTMTXP_integer;

  force u_gtm_wiz_ip_top.inst.dual0.gtm_dual_inst.CH0_GTMRXN_integer = gtm_ch0_rxn_integer ;
  force u_gtm_wiz_ip_top.inst.dual0.gtm_dual_inst.CH0_GTMRXP_integer = gtm_ch0_rxp_integer ;
  force u_gtm_wiz_ip_top.inst.dual0.gtm_dual_inst.CH1_GTMRXN_integer = gtm_ch1_rxn_integer ;
  force u_gtm_wiz_ip_top.inst.dual0.gtm_dual_inst.CH1_GTMRXP_integer = gtm_ch1_rxp_integer ;
end
//pragma translate_on
```

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.

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:

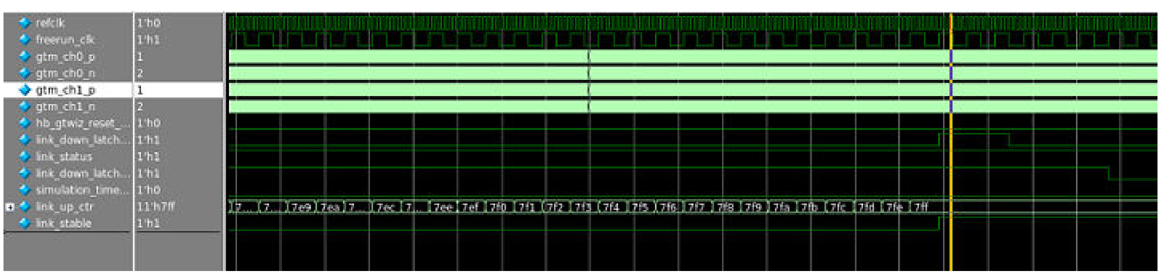
```
Resetting latched link down indicator.
Continuing simulation for 50us to check for maintenance of link.
```

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.
```

The following figure 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 19: **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

Additional Resources and Legal Notices

Xilinx Resources

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

Documentation Navigator and Design Hubs

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

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

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

- In DocNav, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

***Note*:** For more information on DocNav, see the Documentation Navigator page on the Xilinx website.

References

These documents provide supplemental material useful with this guide:

1. *UltraScale FPGAs GTM Transceivers User Guide* (UG581)
2. *Vivado Design Suite User Guide: Designing with IP* (UG896)
3. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
4. *Vivado Design Suite User Guide: Getting Started* (UG910)
5. *Vivado Design Suite Tutorial: Logic Simulation* (UG937)
6. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
7. *UltraScale FPGAs Transceivers Wizard LogiCORE IP Product Guide* (PG182)
8. *UltraScale Devices Integrated Block for 100G Ethernet LogiCORE IP Product Guide* (PG165)
9. *100G IEEE 802.3bj Reed-Solomon Forward Error Correction LogiCORE IP Product Guide* (PG197) (registration required)
10. *50G IEEE 802.3 Reed-Solomon Forward Error Correction LogiCORE IP Product Guide* (PG234) (registration required)
11. *IEEE Standard for Ethernet* (IEEE Std 802.3-2015)
12. *Vivado Design Suite Tutorial: Logic Simulation* (UG937)
13. *UltraFast Design Methodology Guide for Xilinx FPGAs and SoCs* (UG949)

Training Resources

1. Designing FPGAs Using the Vivado Design Suite 1
2. Designing FPGAs Using the Vivado Design Suite 2
3. Designing FPGAs Using the Vivado Design Suite 3
4. Designing with the UltraScale and UltraScale+ Architectures

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
11/24/2020 Version 1.0	
Port Descriptions	Added additional guidance for DRP operations while GTM Controller Logic is enabled.
06/03/2020 Version 1.0	
Simulation	Added reference for UG937.

Section	Revision Summary
10/30/2019 Version 1.0	
GTM Controller Helper Logic	Updated GTM Controller Logic Port Descriptions.
Sampled Eye Scan Functionality	Added new section.
07/02/2019 Version 1.0	
GTM Controller Helper Logic	Added a section on GTM controller logic.
03/28/2019 Version 1.0	
Physical Resources Tab	Added bullet for BYPASS_GTM_CNTRL.
12/05/2018 Version 1.0	
Initial release.	N/A.

Please Read: Important Legal Notices

AUTOMOTIVE APPLICATIONS DISCLAIMER

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

Copyright