JESD204C v4.3

LogiCORE IP Product Guide

Vivado Design Suite

PG242 (v4.3) December 11, 2024

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

Table of Contents

Introduction

The AMD LogiCORE™ IP JESD204C core implements a JESD204C compatible interface supporting line rates from 1 Gbps to 32.5 Gbps (the maximum line rate supported is dependent on the transceiver type and speed grade of the selected device). The JESD204C core can be configured to transmit or receive using either a 64B66B or 8B10B link layer. The JESD204C core is fully backwards compatible with JESD204B.

Features

- Designed to JEDEC® JESD204C.1 Standard
- Supports GTY, GTYP, and GTM (NRZ only) transceivers on AMD Versal™ adaptive SoCs
- Supports GTH and GTY transceivers on AMD UltraScale+™ and AMD UltraScale™ devices
- Supports up to eight lanes per core and greater number of lanes using multiple cores
- Supports 64B66B and 8B10B link layers
- Supports FEC Encoding (TX) and Decoding (RX) on the 64B66B link layer
- Supports CRC-12, CMD and FEC meta data modes on the 64B66B link layer
- Supports subclass 0 and 1 on the 64B66B link layer and Subclass 0,1, and 2 on the 8B10B link layer
- Provides physical and data link layer functions when used with the Versal Adaptive SoC Transceiver Wizard or GT Wizard subsystem and the JESD204_PHY core for UltraScale and UltraScale+ devices

  ***Note*:** The Versal Adaptive SoC Transceiver Wizard/GT Wizard subsystem is used directly by the JESD204C core and the JESD204_PHY is no longer required.
- AXI4-Lite configuration interface
- AXI4-Stream Data and Command interfaces
- Supports Transceiver sharing between TX and RX cores using the JESD204_PHY core or Versal Adaptive SoC Transceiver Wizard/GT Wizard subsystem

IP Facts

AMD LogiCORE™ IP Facts Table	
Core Specifics	
Supported Device Family[1]	Versal adaptive SoCs, UltraScale+, UltraScale
Supported User Interfaces	AXI4-Lite, AXI4-Stream
Resources	Performance and Resource Utilization web page
Provided with Core	
Design Files	Encrypted RTL
Example Design	Verilog
Test Bench	Verilog
Constraints File	XDC
Simulation Model	Verilog
Supported S/W Driver	N/A
Tested Design Flows[2]	
Design Entry	AMD Vivado™ Design Suite
Simulation	For supported simulators, see the *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing* (UG973).
Synthesis	Vivado Synthesis
Support	
Release Notes and Known Issues	Master Answer Record: 68804
All Vivado IP Change Logs	Master Vivado IP Change Logs: 72775
Xilinx Support web page	

Notes:

1. For a complete list of supported devices, see the Vivado IP catalog.
2. For the supported versions of third-party tools, see the *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing* (UG973).

Overview

Navigating Content by Design Process

AMD Adaptive Computing documentation is organized around a set of standard design processes to help you find relevant content for your current development task. You can access the AMD Versal™ adaptive SoC design processes on the Design Hubs page. You can also use the Design Flow Assistant to better understand the design flows and find content that is specific to your intended design needs. This document covers the following design processes:

- **Hardware, IP, and Platform Development:** Creating the PL IP blocks for the hardware platform, creating PL kernels, functional simulation, and evaluating the AMD Vivado™ timing, resource use, and power closure. Also involves developing the hardware platform for system integration. Topics in this document that apply to this design process include:
  - Port Descriptions
  - Register Space
  - Clocking
  - Resets
  - Customizing and Generating the Core
  - Chapter 6: Example Design

Core Overview

The AMD LogiCORE™ IP JESD204C core implements a JESD204C link layer. When used with the LogiCORE IP Versal Adaptive SoC Transceiver Wizard or JESD204_PHY core (to provide the physical layer), a JESD204C system can be created supporting line rates between 1 and 32.5 Gbps on one to eight lanes using Versal adaptive SoC GTY, GTYP, and GTM transceivers or AMD UltraScale+™ and AMD UltraScale™ transceivers (both GTH and GTY).

Note: Versal adaptive SoC GTM designs are currently limited in line rate to between 9.5 Gbps and the maximum line rate for the chosen speed grade, see the GTM data sheet for details. JESD204C only supports NRZ mode and not PAM4 on the Versal adaptive SoC GTM transceiver.

See the device data sheets for maximum line rates supported by each device and family. The JESD204C core can be configured as transmit or receive, using either 64B66B or 8B10B linecoding, and multiple cores can be used to realize links requiring more than eight lanes.

The JESD204C core is delivered by using the AMD Vivado™ Design Suite. In addition, an example design is provided in Verilog.

Unsupported Features

Data Converter sample data mapping is not supported by this IP as the data mapping varies for different devices. For more information see applicable converter data sheets. A simple example sample data mapper and demapper is provided for reference in the example design that can be generated for the core.

Licensing and Ordering

This AMD LogiCORE™ IP module is provided under the terms of the Core License Agreement. The module is shipped as part of the AMD Vivado™ Design Suite. For full access to all core functionalities in simulation and in hardware, you must purchase a license for the core. To generate a full license, visit the product licensing web page. Evaluation licenses and hardware timeout licenses might be available for this core. Contact your local sales representative for information about pricing and availability.

***Note*:** To verify that you need a license, check the License column of the IP catalog. Included means that a license is included with the AMD Vivado™ Design Suite; Purchase means that you have to purchase a license to use the core.

For more information about this core, visit the JESD204C product web page.

A free evaluation version of the core is provided with the AMD Vivado Design Suite which lets you assess the core functionality and demonstrates the various interfaces of the core in simulation. To access the evaluation version, visit the JESD204 IP Evaluation page.

License Checkers

If the IP requires a license key, the key must be verified. The AMD Vivado™ design tools have several license checkpoints for gating licensed IP through the flow. If the license check succeeds, the IP can continue generation. Otherwise, generation halts with an error. License checkpoints are enforced by the following tools:

- Vivado Synthesis

- Vivado Implementation
- write_bitstream (Tcl command)

IMPORTANT! *IP license level is ignored at checkpoints. The test confirms a valid license exists. It does not check IP license level.*

Product Specification

The JESD204C core is used with the AMD Versal™ Adaptive SoC Transceiver Wizard/GT Wizard subsystem or JESD204_PHY core to support the JESD204C physical layer link layer specification.

Standards

JEDEC® Serial interface for Data Converters JESD204C.

Performance

For full details about performance and resource use, visit the Performance and Resource Use web page.

Port Descriptions

The port descriptions for the JESD204C core are described in the following sections.

TX Core

Table 1: **TX Core: System Signals**

Port Name	Interface	I/O	Description
tx_core_clk	System s_axis_tx s_axis_tx_cmd	I	Core logic clock input. Frequency: Serial line rate / 66 (for 64B66B linecoding) Serial line rate / 40 (for 8B10B linecoding)
tx_core_reset	System	I	Core asynchronous logic reset active-High.
tx_aresetn	s_axis_tx s_axis_tx_cmd	O	AXI4-Stream interface reset. Active-Low. Associated with both data and command interfaces.
s_axi_aclk	s_axi	I	AXI4-Lite clock input.

Table 1: **TX Core: System Signals** *(cont'd)*

Port Name	Interface	I/O	Description
s_axi_aresetn	s_axi	I	AXI4-Lite reset input. Active-Low.
s_axi*	s_axi	I	See Appendix A of the *Vivado Design Suite: AXI Reference Guide* (UG1037) for a description of AXI4 signals.
irq	System	O	System interrupt output.
tx_sysref	System	I	SYSREF input. When Subclass 1 mode is selected, this signal is required and used by the core to set the phase of the local extended multi-block clock. This SYSREF signal must be generated synchronous to the core clock. This input should be driven from an external device generating SYSREF for both TX and RX on a link.
tx_sync	System	I	Sync signal. The sync signal is defined as an active-Low sync request signal by JESD204 so this signal is Low until comma alignment is completed and High to request ILA and normal data. This signal is only available when the core is generated with 8B10B linecoding selected.
Versal Adaptive SoCs			
gt_powergood[1]	System	I	This active-High signal from Versal Adaptive SoC Transceiver indicates when the GT clocking resources have completed power up.
gt_loopback[2:0][1]	System	O	This signal to the Versal Adaptive SoC Transceiver controls the various loopback modes. This must be connected to the chN_loopback[2:0] port in the CHn_DEBUG interface of the GT Quad/GT Wizard subsystem.
txusrclk[1]	System	I	Versal Adaptive SoC GTM 66-bit clock used for 64b66b encoder gearbox (GTM only).
reset_all[2]	System	O	Reset output to the `gtwiz_versal` subsystem reset controller. Resets TX PLL and datapath.
reset_tx_pll_and_datapath[2]	System	O	Reset output to the `gtwiz_versal` subsystem reset controller. Resets TX PLL and datapath.
reset_tx_datapath[2]	System	O	Reset output to the `gtwiz_versal` subsystem reset controller. Resets TX datapath only.
reset_tx_done[2]	System	I	Reset Done input from the `gtwiz_versal` subsystem reset controller. Active-High when TX reset sequence is complete.
AMD UltraScale+™/AMD UltraScale™ Devices			
tx_reset_gt	System	O	JESD204_PHY TX datapath reset. Core output to reset the transmit datapath in a connected JESD204_PHY. This must be connected to a JESD204_PHY.

Table 1: **TX Core: System Signals** *(cont'd)*

Port Name	Interface	I/O	Description
tx_reset_done	System	I	JESD204_PHY TX reset done input. Indicates the JESD204_PHY has completed the transmit reset process.

Notes:

1. For an example of how to connect this port to the Versal Adaptive SoC GT Quad/GT Wizard subsystem, generate the example design.
2. These ports are present only when you select the GT Wizard subsystem in JESD204C GUI.

Table 2: **TX Core: Versal Adaptive SoC GT TX Interface Ports**

Signal Name	Interface	I/O	Description
chN_txctrl0[15:0]	GT_TX	O	TX Disparity control to the Versal Adaptive SoC Transceiver. N = Lanes -1
chN_txctrl1[15:0]	GT_TX	O	TX Disparity polarity control to the Versal Adaptive SoC Transceiver. N = Lanes -1
chN_txctrl2[7:0]	GT_TX	O	TX Char is K to the Versal Adaptive SoC Transceiver. N = Lanes -1
chN_txdata[127:0] [3]	GT_TX	O	TX Data to the Versal Adaptive SoC Transceiver. N = Lanes - 1
chN_txdiffctrl[4:0]	GT_TX	O	TX diffctrl to the Versal Adaptive SoC Transceiver. N = Lanes - 1
chN_txelecidle	GT_TX	O	TX txelecidle to the Versal Adaptive SoC Transceiver. N = Lanes - 1
chN_txheader[5:0]	GT_TX	O	TX Header to the Versal Adaptive SoC Transceiver. N = Lanes - 1
chN_txinhibit	GT_TX	O	TX Inhibit to the Versal Adaptive SoC Transceiver. N = Lanes - 1
chN_txmaincursor[6:0]	GT_TX	O	TX Main-cursor pre-emphasis control to the Versal Adaptive SoC Transceiver. N = Lanes - 1
chN_txmstdatapathreset[4]	GT_TX	O	TX Master Datapath only Reset sequence start to the Versal Adaptive SoC Transceiver. N = Lanes - 1
chN_txmstreset[4]	GT_TX	O	TX Master Reset sequence start to the Versal Adaptive SoC Transceiver. N = Lanes - 1
chN_txmstresetdone[4]	GT_TX	I	TX Master Reset sequence Done to the Versal Adaptive SoC Transceiver. N = Lanes - 1
chN_txpd[1:0]	GT_TX	O	TX Power Down to the Versal Adaptive SoC Transceiver. N = Lanes - 1
chN_txpmaresetdone[4]	GT_TX	I	TX PMA Reset Done from the Versal Adaptive SoC Transceiver. N = Lanes - 1

Table 2: **TX Core: Versal Adaptive SoC GT TX Interface Ports** *(cont'd)*

Signal Name	Interface	I/O	Description
chN_txpolarity	GT_TX	O	TX Polarity control to the Versal Adaptive SoC Transceiver. N = Lanes - 1
chN_txpostcursor[5:0]	GT_TX	O	TX Post-cursor pre-emphasis control to the Versal Adaptive SoC Transceiver. N = Lanes - 1
chN_txprbssel[3:0]	GT_TX	O	TX PRBS Select to the Versal Adaptive SoC Transceiver. N = Lanes - 1
chN_txprecursor[5:0]	GT_TX	O	TX Pre-cursor pre-emphasis control to the Versal Adaptive SoC Transceiver. N = Lanes - 1
chN_txprecursor2[5:0]	GT_TX	O	TX Pre-cursor pre-emphasis control to the Versal Adaptive SoC Transceiver. N = Lanes - 1
chN_txprecursor3[5:0]	GT_TX	O	TX Pre-cursor pre-emphasis control to the Versal Adaptive SoC Transceiver. N = Lanes - 1
chN_txrate[7:0]	GT_TX	O	TX Line Rate Control to the Versal Adaptive SoC Transceiver. N = Lanes - 1
chN_txsequence[6:0]	GT_TX	O	TX Sequence Counter to the Versal Adaptive SoC Transceiver. N = Lanes - 1
chN_txuserrdy[4]	GT_TX	O	TX User clocks stable to the Versal Adaptive SoC Transceiver. N = Lanes - 1

Notes:

1. JESD204C only uses the above subset of the available signals on the Versal Adaptive SoC Transceiver TX_GT_IP_Interface. For further details, see *Versal Adaptive SoC Transceivers Wizard LogiCORE IP Product Guide* (PG331).
2. The ports mentioned here are connected between the JESD204 and the Versal Adaptive SoC Transceiver using Block Automation when legacy GT Wizard is selected in JESD204C GUI. For more information, see Chapter 5: Design Flow Steps. When you select the GT Wizard subsystem in JESD204C GUI, Block Automation is not available. The TX_GT_IP_Interface must be manually connected to the GT Wizard subsystem.
3. In Versal Adaptive SoC GTM designs this port is [255:0].
4. These ports are present only when you select the GT Wizard subsystem in JESD204C GUI.

Table 3: **TX Core: JESD204_PHY Interface Ports for UltraScale+/UltraScale Devices**

Signal Name	Interface	I/O	Description
gtN_txdata[63:0]	PHY	O	TX data to JESD204 PHY. N = Lanes - 1
gtN_txheader[1:0]	PHY	O	TX header to JESD204 PHY. N = Lanes - 1
gtN_txcharisk[3:0]	PHY	O	TX Char is K to JESD204 PHY. N = Lanes -1

Table 4: **TX Core: Transmit Interface (64B66B Linecoding Only)**

Signal Name	Interface	I/O	Description
tx_tdata [(64*N)-1:0]	s_axis_tx	I	Transmit data input. N = Lanes
tx_tready	s_axis_tx	O	AXI4-Stream tready
tx_soemb	s_axis_tx	O	Start of extended multi-block boundary indication. Set to 1 to indicate tx_tdata in the following clock cycle is the start of an extended multi-block.
tx_cmd_tdata[(32*N)-1:0]	s_axis_tx_cmd	I	Transmit Cmd interface. N = Lanes For Meta mode = CRC, Cmd payload is bits [6:0] with bits [31:7] set to zero. For Meta mode = CMD, Cmd payload is [18:0] with bits [31:19] set to zero
tx_cmd_tvalid	s_axis_tx_cmd	I	AXI4-Stream tvalid.
tx_cmd_tready	s_axis_tx_cmd	O	AXI4-Stream tready. tx_cmd_tready will be set for one cycle every multi-block to control the Cmd word flow.

Table 5: **TX Core: Transmit Interface (8B10B Linecoding Only)**

Signal Name	Interface	I/O	Description
tx_tdata [(32*N)-1:0]	s_axis_tx	I	Transmit data input. N = Lanes
tx_tready	s_axis_tx	O	AXI4-Stream tready
tx_sof [3:0]	s_axis_tx	O	Start of frame boundary indication. The signal is four bits to indicate the byte position of the first byte of a frame in tdata in the following clock cycle. • When tx_sof = 0001, the first byte of a frame is in bits [7:0] of the tdata word with the next three bytes in bits[31:8]. • When tx_sof = 0010, the first byte is in bits [15:8] of the tdata word with the next two bytes in bits[31:16]; bits [7:0] contain the end of the previous frame. • When tx_sof = 0100, the first byte is in bits [23:16] of the tdata word with the next byte in bits[31:24]; bits [15:0] contain the end of the previous frame. • When tx_sof = 1000, tdata contains the last three bytes of the previous frame in bits [23:0] and the first byte of a new frame in bits [31:24]. ***Note*:** Multiple bits of tx_sof can be asserted in the same cycle, depending on the number of octets per frame. For example, for a frame size of two octets and tx_sof = 0101, the first and third bytes (bits[7:0] and bits[23:16]) of the tdata word contain the first bytes of frames.
tx_somf [3:0]	s_axis_tx	O	Start of multiframe boundary indication. The position of the first byte of each multiframe is encoded in the same way as tx_sof.

RX Core

Table 6: **RX Core: System Signals**

Signal Name	Interface	I/O	Description
rx_core_clk	System s_axis_rx s_axis_rx_cmd	I	Core logic clock input. Frequency: Serial line rate / 66 (for 64B66B linecoding) Serial line rate / 40 (for 8B10B linecoding)
rx_core_reset	System	I	Core asynchronous logic reset active-High.
rx_aresetn	s_axis_rx s_axis_rx_cmd	O	AXI4-Stream interface reset. Active-Low. Associated with both data and command interfaces.
s_axi_aclk	s_axi	I	AXI4-Lite clock input.
s_axi_aresetn	s_axi	I	AXI4-Lite reset input. Active-Low.
s_axi*	s_axi	I	See Appendix A of the *Vivado Design Suite: AXI Reference Guide* (UG1037) for a description of AXI4 signals.
irq	System	O	System interrupt output.
rx_sysref	System	I	SYSREF input. When Subclass 1 mode is selected, this signal is required and used by the core to set the phase of the local extended multi-block clock. This SYSREF signal must be generated synchronous to the core clock. This input should be driven from an external device generating SYSREF for both TX and RX on a link.
rx_sync	System	O	Sync signal. The sync signal is defined as an active-Low sync request signal by JESD204, so this signal is Low until comma alignment is completed and High to indicate the receiver is ready for ILA and normal data. This signal is only available when the core is generated with 8B10B linecoding selected.
encommaalign	System	O	Enable/disable 8B10B comma alignment logic within the JESD204_PHY or the Versal adaptive SoC (excluding GTM). This port should be connected to the Versal Adaptive SoC GT Quad/GT Wizard subsystem as follows: JESD Channel 0: GPI[8] JESD Channel 1: GPI[9] JESD Channel 2: GPI[10] JESD Channel 3: GPI[11] This signal is only available when the core is generated with 8B10B linecoding selected and is N/A for Versal Adaptive SoC GTM designs.
Versal Adaptive SoCs			
gt_powergood[1]	System	I	This active-High signal from Versal Adaptive SoC Transceiver indicates when the GT clocking resources have completed power up.
rxusrclk[1]	System	I	Versal Adaptive SoC GTM 66-bit clock used for 64b66b decoder gearbox (GTM only).
reset_all[2]	System	O	Reset output to the `gtwiz_versal` subsystem reset controller. Resets RX PLL and datapath.

Table 6: **RX Core: System Signals** *(cont'd)*

Signal Name	Interface	I/O	Description
reset_rx_pll_and_datapath[2]	System	O	Reset output to the `gtwiz_versal` subsystem reset controller. Resets RX PLL and datapath.
reset_rx_datapath[2]	System	O	Reset output to the `gtwiz_versal` subsystem reset controller. Resets RX datapath only.
reset_rx_done[2]	System	I	Reset Done input from the `gtwiz_versal` subsystem reset controller. Active-High when RX reset sequence is complete.
UltraScale+/UltraScale Devices			
rx_reset_gt	System	O	JESD204_PHY RX datapath reset. Core output to reset the receive datapath in a connected JESD204_PHY. This must be connected to a JESD204_PHY.
rx_reset_done	System	I	JESD204_PHY RX reset done input. Indicates the JESD204_PHY has completed the receive reset process. For an example of how to connect this port to the JESD204_PHY or Versal adaptive SoC GT Quad/GT Wizard subsystem, generate the example design.

Notes:

1. For an example of how to connect this port to the Versal Adaptive SoC GT Quad/GT Wizard subsystem, generate the example design.
2. These ports are present only when you select the GT Wizard subsystem in JESD204C GUI.

Table 7: **RX Core: Versal Adaptive SoC GT RX Interface Ports**

Signal Name	Interface	I/O	Description
chN_rxctrl0[15:0]	GT_RX	I	RX Char is K from the Versal Adaptive SoC Transceiver. N = Lanes -1
chN_rxctrl1[15:0]	GT_RX	I	RX Disparity Error from the Versal Adaptive SoC Transceiver. N = Lanes -1
chN_rxctrl2[7:0]	GT_RX	I	RX Char is Comma from the Versal Adaptive SoC Transceiver. N = Lanes -1
chN_rxctrl3[7:0]	GT_RX	I	RX Not In Table Error from the Versal Adaptive SoC Transceiver. N = Lanes -1
chN_rxdata[127:0] [3]	GT_RX	I	RX Data from the Versal Adaptive SoC Transceiver. N = Lanes - 1
chN_rxdatavalid[1:0]	GT_RX	I	RX Data Valid from the Versal Adaptive SoC Transceiver. N = Lanes - 1
chN_rxgearboxslip	GT_RX	O	RX Gearbox Slip to the Versal Adaptive SoC Transceiver. N = Lanes - 1
chN_rxheader[5:0]	GT_RX	I	RX Header from the Versal Adaptive SoC Transceiver. N = Lanes - 1

Table 7: **RX Core: Versal Adaptive SoC GT RX Interface Ports** *(cont'd)*

Signal Name	Interface	I/O	Description
chN_rxheadervalid[1:0]	GT_RX	I	RX Header Valid from the Versal Adaptive SoC Transceiver. N = Lanes - 1
chN_rxlpmen	GT_RX	O	RX LPM Mode Enable to the Versal Adaptive SoC Transceiver. N = Lanes - 1
chN_rxmstdatapathreset[4]	GT_RX	O	RX Master Datapath only Reset sequence start to the Versal Adaptive SoC Transceiver. N = Lanes - 1
chN_rxmstreset[4]	GT_RX	O	RX Master Reset sequence start to the Versal Adaptive SoC Transceiver. N = Lanes - 1
chN_rxmstresetdone[4]	GT_RX	I	RX Master Reset sequence Done to the Versal Adaptive SoC Transceiver. N = Lanes - 1
chN_rxpd[1:0]	GT_RX	O	RX Power Down to the Versal Adaptive SoC Transceiver. N = Lanes - 1
chN_rxpmaresetdone[4]	GT_RX	I	RX PMA Reset Done from the Versal Adaptive SoC Transceiver. N = Lanes - 1
chN_rxpolarity	GT_RX	O	RX Polarity control to the Versal Adaptive SoC Transceiver. N = Lanes - 1
chN_rxprbscntreset	GT_RX	O	RX PRBS Error Count Reset to the Versal Adaptive SoC Transceiver. N = Lanes - 1
chN_rxprbserr	GT_RX	I	RX PRBS Error from the Versal Adaptive SoC Transceiver. N = Lanes - 1
chN_rxprbslocked	GT_RX	I	RX PRBS Locked from the Versal Adaptive SoC Transceiver. N = Lanes - 1
chN_rxprbssel[3:0]	GT_RX	O	RX PRBS Select to the Versal Adaptive SoC Transceiver. N = Lanes - 1
chN_rxrate[7:0]	GT_RX	O	RX Line Rate Control to the Versal Adaptive SoC Transceiver. N = Lanes - 1

Table 7: **RX Core: Versal Adaptive SoC GT RX Interface Ports** *(cont'd)*

Signal Name	Interface	I/O	Description
chN_rxuserrdy[4]	GT_RX	O	RX User clocks stable to the Versal Adaptive SoC Transceiver. N = Lanes - 1

Notes:

1. JESD204C only uses the above subset of the available signals on the Versal Adaptive SoC Transceiver Rx_GT_IP_Interface. For further details, see *Versal Adaptive SoC Transceivers Wizard LogiCORE IP Product Guide* (PG331).
2. The ports mentioned here are connected between the JESD204 core and the Versal Adaptive SoC Transceiver using Block Automation when legacy GT Wizard is selected in JESD204C GUI. For more information, see Chapter 5: Design Flow Steps. When you select the GT Wizard subsystem in JESD204C GUI, Block Automation is not available. The RX_GT_IP_Interface must be manually connected to the GT Wizard subsystem.
3. In Versal Adaptive SoC GTM designs this port is [255:0].
4. These ports are present only when you select the GT Wizard subsystem in JESD204C GUI.

Table 8: **RX Core: JESD204_PHY Interface Ports for UltraScale+/UltraScaleDevices**

Signal Name	Interface	I/O	Description
gtN_rxdata[63:0]	PHY	I	RX data from JESD204 PHY. N = Lanes - 1
gtN_rxheader[1:0]	PHY	I	RX header from JESD204 PHY. N = Lanes - 1
gtN_misalign	PHY	I	Signal from JESD204 PHY to indicate a misaligned sync header was detected. N = Lanes - 1
gtN_block_sync	PHY	I	Signal from JESD204 PHY to indicate block sync status. N = Lanes -1
gtN_rxcharisk[3:0]	PHY	I	RX Char is K from JESD204 PHY. N = Lanes -1
gtN_rxdisperr[3:0]	PHY	I	RX Disparity Error from JESD204 PHY. N = Lanes -1
gtN_notintable[3:0]	PHY		RX Not In Table Error from JESD204 PHY. N = Lanes -1

Table 9: **RX Core: Receive Interface (64B66B Linecoding Only)**

Signal Name	Interface	I/O	Description
rx_tdata [(64*N)-1:0]	m_axis_rx	O	Receive data output. N = Lanes
rx_tvalid	m_axis_rx	O	AXI4-Stream tvalid.
rx_soemb	m_axis_rx	O	Start of extended multi-block boundary indication. Set to 1 to indicate tx_tdata in the following clock cycle is the start of an extended multi-block.
rx_emb_err	m_axis_rx	O	Extended Multi-block Error. Set to 1 on the last block of an extended multi-block if a multi-block alignment error was detected.

Table 9: **RX Core: Receive Interface (64B66B Linecoding Only)** *(cont'd)*

Signal Name	Interface	I/O	Description
rx_crc_err	m_axis_rx	O	CRC error. Set to 1 on the last block of an multi-block if a CRC or Uncorrectable FEC error was detected within the multi-block.
rx_cmd_tdata[(32*N)-1:0]	m_axis_rx_cmd	O	Transmit Cmd interface. N = Lanes For Meta mode = CRC, Cmd payload is bits [6:0] with bits [31:7] set to Zero. For Meta mode = CMD, Cmd payload is [18:0] with bits [31:19] set to Zero
rx_cmd_tvalid	m_axis_rx_cmd	O	AXI4-Stream tvalid. rx_cmd_tvalid will be set for one cycle every multi-block to control the Cmd word flow.
rx_cmd_tready	m_axis_rx_cmd	I	AXI4-Stream tready.
rx_cmd_tuser[N:0]	m_axis_rx_cmd	O	AXI4-Stream tuser. N = Lanes - 1. The tuser data bits are used to signal that a Multiblock Alignment error was detected in the previous multiblock of the associated lane. This might mean the CMD data is invalid.

Table 10: **RX Core: Receive Interface (8B10B Linecoding Only)**

Signal Name	Interface	I/O	Description
rx_tdata [(32*N)-1:0]	m_axis_rx	O	Receive data output. N = Lanes
rx_tvalid	m_axis_rx	O	AXI4-Stream tvalid.
rx_sof[3:0]	m_axis_rx	O	Start of frame boundary indication. The position of the first byte in a frame is encoded in the same way as tx_sof [3:0]. This signal is asserted one cycle before the AXI4-Stream data. The alignment of the first valid byte is always in byte 0 if the multiframe size is a multiple of 4 and rx_buffer_delay is a multiple of 4.
rx_somf[3:0]	m_axis_rx	O	Start of multiframe boundary indication. The position of the first byte of each multiframe is encoded in the same way as rx_sof.

Table 10: **RX Core: Receive Interface (8B10B Linecoding Only)** *(cont'd)*

Signal Name	Interface	I/O	Description
rx_frm_err[3:0]	m_axis_rx	O	Error in byte. JESD204 specifies that data must be replicated from the previous frame if certain errors occur. The core does not buffer the previous frame. You can choose to implement a frame buffer or use a buffer elsewhere in the system to perform this function if required. The rx_frm_err signal indicates that a single byte error exists in the data stream. There is one bit for each byte of each AXI4-Stream. For example, a four lane interface has four 32-bit AXI4-Streams, the error signal is 16 bits wide with bit 15 of the error signal corresponding to the most significant byte of lane 4 and bit 0 of the error signal corresponding to the least significant byte of lane 1. This signal is synchronous to rx_core_clk and output in the cycle before the data in the same way as rx_sof.

Register Space

The JESD204C core is configured using an AXI4-Lite Register Interface. The register map is shown in the following table.

***Note*:** The JESD204C AXI4-Lite interface currently does not support outstanding transactions. The ongoing transaction data access must end completely before the next transaction starts. Otherwise, the new transaction data might get incorrectly stored at the address of the previous transaction data.

The RX and TX cores share a common address map and register definitions where possible, exceptions are highlighted.

RECOMMENDED: *AMD recommends that if significant configuration changes are made using the control registers (in particular, changes to framing parameters), the core should be reset to ensure that the link is resynchronized using the updated parameters.*

Table 11: **Register Address Space**

AXI4-Lite Address	Register Name	64B66B		8B10B	
		TX Access Type	RX Access Type	TX Access Type	RX Access Type
0x000	IP_VERSION	R	R	R	R
0x004	IP_CONFIG	R	R	R	R
0x020	RESET	RW	RW	RW	RW
0x024	CTRL_ENABLE	RW	RW	N/A	N/A
0x028	CTRL_TX_SYNC	N/A	N/A	RW	N/A
0x030	CTRL_MB_IN_EMB	RW	RW	N/A	N/A
0x034	CTRL_SUB_CLASS	RW	RW	RW	RW

Table 11: **Register Address Space** *(cont'd)*

AXI4-Lite Address	Register Name	64B66B		8B10B	
		TX Access Type	RX Access Type	TX Access Type	RX Access Type
0x038	CTRL_META_MODE	RW	RW	N/A	N/A
0x03C	CTRL_8B10B_CFG	N/A	N/A	RW	RW
0x040	CTRL_LANE_ENA	RW	RW	RW	RW
0x044	CTRL_RX_BUF_ADV	N/A	RW	N/A	RW
0x048	CTRL_TEST_MODE	RW	RW	RW	RW
0x04C	CTRL_RX_MBLOCK_TH	N/A	RW	N/A	N/A
0x050	CTRL_SYSREF	RW	RW	RW	RW
0x054	STAT_LOCK_DEBUG	N/A	R	N/A	N/A
0x058	STAT_RX_ERR	N/A	N/A	N/A	R
0x05C	STAT_RX_DEBUG	N/A	N/A	N/A	R
0x060	STAT_STATUS	R	R	R	R
0x064	CTRL_IRQ	RW	RW	RW	RW
0x068	STAT_IRQ	R	R	R	R
0x070	CTRL_TX_ILA_CFG0	N/A	N/A	RW	N/A
0x074	CTRL_TX_ILA_CFG1	N/A	N/A	RW	N/A
0x078	CTRL_TX_ILA_CFG2	N/A	N/A	RW	N/A
0x07C	CTRL_TX_ILA_CFG3	N/A	N/A	RW	N/A
0x080	CTRL_TX_ILA_CFG4	N/A	N/A	RW	N/A
0x400[1]	(Lane 0) STAT_RX_BUF_LVL	N/A	R	N/A	R
0x404[1]	(Lane 0) CTRL_TX_ILA_LID	N/A	N/A	RW	N/A
0x410[1]	(Lane 0) STAT_RX_ERROR_CNT0	N/A	R	N/A	N/A
0x414[1]	(Lane 0) STAT_RX_ERROR_CNT1	N/A	R	N/A	N/A
0x420[1]	(Lane 0) STAT_LINK_ERR_CNT	N/A	N/A	N/A	R
0x424[1]	(Lane 0) STAT_TEST_ERR_CNT	N/A	N/A	N/A	R
0x428[1]	(Lane 0) STAT_TEST_ILA_CNT	N/A	N/A	N/A	R
0x42C[1]	(Lane 0) STAT_TEST_MF_CNT	N/A	N/A	N/A	R
0x430[1]	(Lane 0) CTRL_RX_ILA_CFG0	N/A	N/A	N/A	R
0x434[1]	(Lane 0) CTRL_RX_ILA_CFG1	N/A	N/A	N/A	R
0x438[1]	(Lane 0) CTRL_RX_ILA_CFG2	N/A	N/A	N/A	R
0x43C[1]	(Lane 0) CTRL_RX_ILA_CFG3	N/A	N/A	N/A	R

Table 11: **Register Address Space** *(cont'd)*

AXI4-Lite Address	Register Name	64B66B		8B10B	
		TX Access Type	RX Access Type	TX Access Type	RX Access Type
0x440[1]	(Lane 0) CTRL_RX_ILA_CFG4	N/A	N/A	N/A	R
0x444[1]	(Lane 0) CTRL_RX_ILA_CFG5	N/A	N/A	N/A	R
0x448[1]	(Lane 0) CTRL_RX_ILA_CFG6	N/A	N/A	N/A	R
0x44C[1]	(Lane 0) CTRL_RX_ILA_CFG7	N/A	N/A	N/A	R
0x460[1]	(Lane 0) CTRL_TX_GT	RW	RW	RW	RW
0X464[1]	(Lane 0) CTRL_RX_GT	RW	RW	RW	RW
0X468[1]	(Lane 0) CTRL_TX_VERSAL_GTY/ GTYP	RW	RW	RW	RW
0X46C[1]	(Lane 0) CTRL_TX_VERSAL_GTM	RW	RW	RW	RW

Notes:

1. As shown, lane 0 registers start at 0x400. Lane 1 registers occupy the equivalent space starting at 0x480, lanes 2-7 follow the same pattern (that is, lane 2 = 0x500, lane 3 = 0x580, etc.).

Table 12: **IP_VERSION**

Bits	Default Value	Description
31:24	–	Version: Major
23:16	–	Version: Minor
15:8	–	Version: Revision
7:0	–	Reserved (read 0x00)

Table 13: **IP_CONFIG**

Bits	Default Value	Description
18	–	1= FEC Included 0=FEC not included (FEC is only available when configured for 64B66B)
17	–	1 = Core is 64B66B 0 = Core is 8B10B
16	–	1 = Core is TX 0 = Core is RX
3:0	–	Number of lanes in core.

Table 14: **RESET**

Bits	Default Value	Description
31:24[1]	–	gt_mst_reset_busy[7:0] 1-bit per enabled GT lane. These bits show the inversion of the mstresetdone inputs from the Versal adaptive SoC GTs.
23:16[1]	–	gt_pma_reset_busy[7:0] 1-bit per enabled GT lane. These bits show the inversion of the pmaresetdone inputs from the Versal adaptive SoC GTs.
7	–	gt_reset_busy 1 = GTs are still in reset 0 = GTs have finished reset
6[1]	–	gt_powergood input state This bit shows the inversion of the gt_powergood input from the Versal adaptive SoC GT. 1 = gt_powergood not completed 0 = gt_powergood completed
5	–	Core Reset Register State 1 = tx/rx_reset asserted 0 = tx/rx_reset deasserted
4	–	Core External Reset Pin State 1 = tx/rx_core_reset asserted 0 = tx/rx_core_reset deasserted
1[1]	0	Reset type 1= Initiates a Datapath only reset 0 = Initiates a PLL+ Datapath reset Reset type should be asserted first followed by the reset bit 0.
0	0	Reset. (not self-clearing) 1 = put core into reset 0 = Release core from reset After setting this bit to 1 and clearing back to 0, this bit will read back 1 until the reset process has completed.

Notes:

1. For Versal adaptive SoCs only.

Table 15: **CTRL_ENABLE**

Bits	Default Value	Description
1	0	Enable Data Interface. 1 = Enables the AXI4-Stream Data interface and transmits/receives data on the link. 0 = The link will be transmitting/receiving scrambled 0s
0	0	Enable Cmd interface. 1 = Enables the AXI4-Stream Cmd interface and the associated processing of the sync header meta data. 0 = Cmd words will be zeroed.

Table 16: **CTRL_TX_SYNC**

Bits	Default Value	Description
0	0	tx_sync_force. Force on 8B10B transmitter. When set to 1, this register overrides the value on the tx_sync pin.

Table 17: **CTRL_MB_IN_EMB**

Bits	Default Value	Description
7:0	0x1	Number of multiblocks in an extended multiblock. Program this register with the actual value. ***Note*:** 0 is not valid.

Table 18: **CTRL_SUB_CLASS**

Bits	Default Value	Description
1:0	0x1	Sub Class: 0 = Subclass 0 1 = Subclass 1 2 = Subclass 2 (8B10B only)

Table 19: **CTRL_META_MODE**

Bits	Default Value	Description
1:0	0x0	Meta Mode: 0 = CRC12 1 = N/A 2 = CMD 3 = FEC

Table 20: **CTRL_8B10B_CFG**

Bits	Default Value	Description
31:24	0x3	ILA multiframes. Multiframes in the Transmitted Initial Lane Alignment Sequence. Parameter Range: 4–256; program the register with required value minus 1.
21:20	0x0	Reserved. Must be set to zero.
19	0	1 = Enable Link Error counters (Link errors are counted and reported using Link Error Count registers per lane) 0 = Disable Link Error counters
18	0	Error Reporting via sync: 1 = Error reporting using SYNC interface Enabled 0 = Error reporting using SYNC interface Disabled

Table 20: **CTRL_8B10B_CFG** *(cont'd)*

Bits	Default Value	Description
17	1	ILA Required: 1 = Enable ILA Support 0 = Disable ILA Support
16	1	Scrambling: 1 = Enable Scrambling 0 = Disable Scrambling
12:8	0xF	Frames per Multiframe (K) Parameter range 1–32; Program register with required value minus 1 (for example, for K = 16, 0x0F should be programmed)
7:0	0x1	Octets per Frame (F) Parameter range 1–256; Program register with required value minus 1 (for example, for F = 4, 0x03 should be programmed)

Table 21: **CTRL_LANE_ENA**

Bits	Default Value	Description
7:0	–	Lane enable register. Default is all lanes enabled. Set one bit per lane (bit 0 = lane 0, bit 1 = lane 1 etc.) ***Note*:** If any lanes are disabled, the lane ID and the number of lanes per link registers should be reprogrammed accordingly.

Table 22: **CTRL_RX_BUF_ADV**

Bits	Default Value	Description
9:0	0x0	Advance the release of the receiver buffer: For 64B66B linecoding, advance the release of the buffer by N 64 bit words. The range of N is between 0 and 31 words. For 8B10B linecoding, advance the release of the buffer by N octets. The range of N is between 0 and 127 octets.

Table 23: **CTRL_TEST_MODE**

Bits	Default Value	Description
30:28	0x0	GT loopback: (TX only) 00: Normal operation 001: Near-End PCS Loopback 010: Near-End PMA Loopback 011: Reserved 100: Far-End PMA Loopback 101: Reserved 110: Far-End PCS Loopback ***Note*:** The above bits maps directly to the GT loopback input ports. Refer to the specific GT user guide for more details.
27:12	–	Reserved
11:8	0x0	PRBS mode select (Versal adaptive SoCs only): 4'b0000: Standard operation mode. (PRBS off) 4'b0001: PRBS-7 4'b0010: PRBS-9 4'b0011: PRBS-15 4'b0100: PRBS-23 4'b0101: PRBS-31 4'b0110: PRBS-13
2:0	0x0	Test mode select (8B10B mode): 000 = Normal operation 001 = Transmit receive /K28.5/ indefinitely[2] 010 = Synchronize as normal then transmit/receive repeated ILA sequences.[2] 101 = Transmit Modified Random Pattern RPAT (TX only)[1] 111 = Transmit Scrambled Jitter Pattern JSPAT (TX only)[1]

Notes:

1. These test modes are only applicable to the JESD204C 8B10B transmitter IP. They are used to set the transceiver to output specific patterns that might be used to evaluate the electrical characteristics of a link using tools such as IBERT. A JESD204C 8B10B receiver core will not synchronize or function if these test patterns are received.
2. Physical layer test modes are made available through the Versal Adaptive SoC Transceiver or JESD204 PHY register interface.

Table 24: **CTRL_RX_MBLOCK_TH**

Bits	Default Value	Description
2:0	0x0	MB lock threshold. How many correct/incorrect multiblock alignment markers are required to achieve/lose multiblock lock. The actual value used is 1 plus the number in this register.

Table 25: **CTRL_SYSREF**

Bits	Default Value	Description
23:16	0x0	(For 64b66b cores only) SYSREF Delay: Add additional delay to SYSREF alignment LEMC. 0xFF = 255 core_clk cycles delay .... 0x00 = 0 core_clk cycles delay. This register is used to retard the phase of the LEMC.
19:16	0x0	SYSREF Delay: Add additional delay to SYSREF alignment of LMFC. 1111 = 15 core_clk cycles delay .... 0000 = 0 core_clk cycles delay This register is used to retard the phase of the LMFC.
10:8	0x0	SYSREF Tolerance: Specify a tolerance value in core_clk cycles for SYSREF detection when in SYSREF Always mode. If a SYSREF event is detected within +/- tolerance core_clk cycles from its expected position, no error signal will be issued.
1	0	SYSREF Required on Re-Sync 1 = Following a Link Re-Sync event, a SYSREF event is required to re-align the local LMFC/LEMC before the link will operate. 0 = No SYSREF is required to restart a link after a Re-sync event.
0	0	SYSREF Always 1 = The core will align the LMFC/LEMC counter on all SYSREF events. 0 = The core will only align the LMFC/LEMC counter on the first SYSREF event following a reset, all subsequent SYSREF events will be ignored.

Table 26: **STAT_LOCK_DEBUG**

Bits	Default Value	Description
23:16	–	Lane indicator multiblock aligned. 1 bit per lane. Set to 1 when multiblock alignment is achieved. 0 otherwise.
7:0	–	Lane indicator 64B66B sync header aligned. 1 bit per lane. Set to 1 when sync header alignment is achieved. 0 otherwise.

Table 27: **STAT_RX_ERR**

Bits	Default Value	Description
31:28	–	RX Error status lane 7
27:24	–	RX Error status lane 6
23:20	–	RX Error status lane 5
19:16	–	RX Error status lane 4
15:12	–	RX Error status lane 3
11:8	–	RX Error status lane 2

Table 27: **STAT_RX_ERR** *(cont'd)*

Bits	Default Value	Description
7:4	–	RX Error status lane 1
3:0	–	RX Error status lane 0 Bit 3: unused Bit 2: Unexpected K-character(s) received Bit 1: Disparity Error(s) received Bit 0: Not in table Error(s) received Each bit indicates that 1 or more errors of that type have been received in Lane 0 because the register was last read. All status bits are cleared to 0 on read of this register.

Table 28: **STAT_RX_DEBUG**

Bits	Default Value	Description
31:28	–	Link Debug status Lane 7 as per lane 0
27:24	–	Link Debug status Lane 6 as per lane 0
23:20	–	Link Debug status Lane 5 as per lane 0
19:16	–	Link Debug status Lane 4 as per lane 0
15:12	–	Link Debug status Lane 3 as per lane 0
11:8	–	Link Debug status Lane 2 as per lane 0
7:4	–	Link Debug status Lane 1 as per lane 0
3:0	–	Link Debug status Lane 0 Bit 3: 1 = Start of Data was Detected[1] Bit 2: 1 = Start of ILA was Detected[1] Bit 1: 1 = Lane has Code Group Sync[2] Bit 0: 1 = Lane is currently receiving K28.5's (BC alignment characters)[2]

Notes:

1. The status bits 3:2 latch when set and are cleared on read or when the core is reset. If the core is streaming data when these bits are cleared, they are instantly set again. The purpose of these bits is to detect whether these conditions have occurred because SYNC was asserted.
2. The status bits 1:0 show instantaneous status.

Table 29: **STAT_STATUS**

Bits	Default Value	Description
15	–	8B10B Alignment Error: 1= An 8B10B RX misalignment has been detected. Misalignment is determined by monitoring the Multiframe framing characters. If eight consecutive framing characters are detected in misaligned positions, then this bit is asserted.
14	–	8B10B RX started: 1 = The link has started outputting data on the AXI4-Stream port. This bit is applicable to an 8B10B RX only.
13	–	8B10B CGS status: 1 = The link has achieved Code Group Sync. This bit is applicable to an 8B10B RX only.

Table 29: **STAT_STATUS** *(cont'd)*

Bits	Default Value	Description
12	–	8B10B SYNC status: 1 = The receiver has signaled SYNC has been achieved. This bit is applicable to an 8B10B link only.
10	–	Buffer Overflow error. 1 = The receiver buffer has overflowed.
5	–	64B66B Multiblock Lock Status: 1 = Multiblock lock achieved on all lanes This bit is a logical AND of the individual lane status bits.
4	–	64B66B Sync Header Lock Status: 1 = Sync Header lock achieved on all lanes. This bit is a logical AND of the individual lane status bits
2	–	SYSREF error. A sysref was detected out of phase with the local extended multiblock clock.
1	–	SYSREF captured.
0	–	Interrupt pending.

Table 30: **CTRL_IRQ**

Bits	Default Value	Description
14	0	1 = Enable interrupt on 8B10B RX AXI4-Stream data start.
13	0	1 = Enable interrupt on 8B10B RX Resync request.
12	0	1 = Enable interrupt on 8B10B SYNC assertion.
10	0	1 = Enable Interrupt on overflow Error.
9	0	1 = Enable Interrupt on 64B66B FEC Error.
8	0	1 = Enable Interrupt on 64B66B CRC Error.
7	0	1 = Enable Interrupt on 64B66B Multiblock Error.
6	0	1 = Enable Interrupt on 64B66B Block Sync Error.
5	0	1 = Enable Interrupt on Loss of 64B66B Multiblock Lock.
4	0	1 = Enable Interrupt on Loss of 64B66B Sync Header Lock.
2	0	1 = Enable Interrupt on SYSREF Error.
1	0	1 = Enable Interrupt on SYSREF Received.
0	0	Global Interrupt Enable: Must be set for any interrupt to function.

Table 31: **STAT_IRQ**

Bits	Default Value	Description
14	–	1 = 8B10B RX AXI4-Stream data start interrupt triggered.
13	–	1 = 8B10B RX Resync request interrupt triggered.
12	–	1 = 8B10B SYNC assertion interrupt triggered.
10	–	1 = Overflow Error Interrupt triggered.
9	–	1 = 64B66B FEC Error detected Interrupt triggered.

Table 31: **STAT_IRQ** *(cont'd)*

Bits	Default Value	Description
8	–	1 = 64B66B CRC Error detected Interrupt triggered.
7	–	1 = 64B66B Multiblock Error detected Interrupt triggered.
6	–	1 = 64B66B Block Sync Error detected Interrupt triggered.
5	–	1 = 64B66B Multiblock Lock Status Interrupt triggered.
4	–	1 = 64B66B Sync Header Lock Status Interrupt triggered.
2	–	1 = SYSREF Error Interrupt triggered.
1	–	1 = SYSREF Received Interrupt triggered.

Table 32: **CTRL_TX_ILA_CFG0**

Bits	Default Value	Description
11:8	0x0	BID (Bank ID). Binary value.
7:0	0x0	DID (Device ID). Binary value.

Table 33: **CTRL_TX_ILA_CFG1**

Bits	Default Value	Description
31:26	–	Reserved
25:24	0x0	CS (Control bits per Sample). Binary value.
23:21	–	Reserved
20:16	0x0	N' (Totals bits per Sample). Binary value minus 1.
15:13	–	Reserved
12:8	0x0	N (Converter Resolution). Binary value minus 1.
7:0	0x0	M (Converters per Device). Binary value minus 1.

Table 34: **CTRL_TX_ILA_CFG2**

Bits	Default Value	Description
28:24	0x0	CF (Control Words per Frame). Binary value.
16	0	HD (High Density format)
12:8	0x0	S (Samples per Converter per Frame). Binary value minus 1.

Table 35: **CTRL_TX_ILA_CFG3**

Bits	Default Value	Description
31:17	–	Reserved
16	0	ADJDIR (Adjust Direction) [Subclass 2 Only]. Binary value.
15:9	–	Reserved
8	0	PHADJ (Phase Adjust Request) [Subclass 2 Only]. Binary value.
7:4	–	Reserved

Table 35: **CTRL_TX_ILA_CFG3** *(cont'd)*

Bits	Default Value	Description
3:0	0x0	ADJCNT (Phase Adjust Count) [Subclass 2 Only]. Binary value. RX: captured configuration data from the ILA sequence (per lane). TX: Sets the values to be transmitted in the ILA sequence for all lanes.

Table 36: **CTRL_TX_ILA_CFG4**

Bits	Default Value	Description
15:8	0x0	RES2 (Reserved Field 2)
7:0	0x0	RES1 (Reserved Field 1)

Table 37: **STAT_RX_BUF_LVL**

Bits	Default Value	Description
9:0	–	Buffer fill level. The amount of data in the receiver buffer for lane 0. For 64B66B linecoding: The value returned is the number of 64-bit words in the buffer. For 8B10B Linecoding: The value returned is the number of bytes in the buffer.

Notes:

1. This is a *Per Lane* Register

Table 38: **CTRL_TX_ILA_LID**

Bits	Default Value	Description
31:21	–	Reserved
20:16	L	Number of lanes per link (binary value minus 1). The default value for all lanes is L (total number of lanes minus 1). These values should be programmed when: • Not all lanes in the generated IP core are enabled. • A single TX core is used to drive multiple DAC devices. • Multiple TX cores are combined to create links with more than eight lanes.
15:5	–	Reserved
4:0	N	ID of lane N. Value can be anywhere between 0 and 31. The default value N is set to the lane number. For interfaces using more than 8 lanes and hence multiple JESD204C cores, this register should be programmed to ensure each lane has the correct identifier.

Notes:

1. This is a *Per Lane* Register

Table 39: **STAT_RX_ERROR_CNT0**

Bits	Default Value	Description
31:16	–	CRC error counter.
15:8	–	64B66B Multiblock alignment error counter.
7:0	–	64B66B Sync Header alignment error counter.

Notes:

1. This is a *Per Lane* Register. The counts are cumulative and are cleared on read or reset. The counts should be cleared with a register read when a link has been successfully established.

Table 40: **STAT_RX_ERROR_CNT1**

Bits	Default Value	Description
31:16	–	64B66B FEC uncorrected errors counter.
15:0	–	64B66B FEC corrected errors counter.

Notes:

1. This is a *Per Lane* Register. The counts are cumulative and are cleared on read or reset. The counts should be cleared with a register read when a link has been successfully established.

Table 41: **STAT_LINK_ERR_CNT**

Bits	Default Value	Description
31:0	–	Link Error Count Count of total received link errors (per lane) when Link Error Counters is Enabled. Errors counted are Disparity or Not In Table errors indicated by the lane. The error counter can be reset by disabling and re-enabling the Link Error Counters Enable, bit 19 in the CTRL_8B10B_CFG register.

Notes:

1. This is a *Per Lane* Register.

Table 42: **STAT_TEST_ERR_CNT**

Bits	Default Value	Description
31:0	–	Test Mode Error Count Count of Errors received in Data link Layer test modes. Test Mode = 001 (Continuous K28.5): counts any non K28.5 characters received Test Mode = 010 (Continuous ILA): counts any unexpected characters received This count resets to zero on transition to an active test mode and retains any count value on transition out of an active test mode.

Notes:

1. This is a *Per Lane* Register.

Table 43: **STAT_TEST_ILA_CNT**

Bits	Default Value	Description
31:0	–	Test Mode ILA Count Count of total ILA Sequences received when Test Mode = 010 (Continuous ILA) This count resets to zero on transition to Test Mode = 010, and retains any count value on transition out of test mode.

Notes:

1. This is a *Per Lane* Register.

Table 44: **STAT_TEST_MF_CNT**

Bits	Default Value	Description
31:0	–	Test Mode Multiframe Count Count of total ILA Multiframes received when Test Mode = 010 (Continuous ILA) This count resets to zero on transition to Test Mode = 010 and retains any count value on transition out of test mode.

Notes:

1. This is a *Per Lane* Register.

Table 45: **CTRL_RX_ILA_CFG0**

Bits	Default Value	Description
31:11	–	Reserved
10:8	–	JESDV (JESD204 version): 000=JESD204A 001=JESD204B 010 = JESD204C
7:3	–	Reserved
2:0	–	SUBCLASS: 000=Subclass0 001=Subclass1 010=Subclass2

Notes:

1. This is a *Per Lane* Register.

Table 46: **CTRL_RX_ILA_CFG1**

Bits	Default Value	Description
31:8	–	Reserved
7:0	–	F (Octets per Frame). Binary value minus 1.

Notes:

1. This is a *Per Lane* Register.

Table 47: **CTRL_RX_ILA_CFG2**

Bits	Default Value	Description
31:5	–	Reserved
4:0	–	K (Frames per Multiframe). Binary value minus 1.

Notes:

1. This is a *Per Lane* Register.

Table 48: **CTRL_RX_ILA_CFG3**

Bits	Default Value	Description
31:29	–	Reserved
28:24	–	L (Lanes per Link). Binary value minus 1.
23:21	–	Reserved
20:16	0x0	LID (Lane ID). Binary value.
15:12	–	Reserved
11:8	0x0	BID (Bank ID). Binary value.
7:0	0x0	DID (Device ID). Binary value.

Notes:

1. This is a *Per Lane* Register.

Table 49: **CTRL_RX_ILA_CFG4**

Bits	Default Value	Description
31:26	–	Reserved
25:24	–	(Control bits per Sample). Binary value.
23:21	–	Reserved
20:16	–	N' (Totals bits per Sample). Binary value minus 1.
15:13	–	Reserved
12:8	–	N (Converter Resolution). Binary value minus 1.
7:0	–	M (Converters per Device). Binary value minus 1.

Notes:

1. This is a *Per Lane* Register.

Table 50: **CTRL_RX_ILA_CFG5**

Bits	Default Value	Description
31:29	–	Reserved
28:24	0x0	CF (Control Words per Frame). Binary value.
23:17	–	Reserved
16	0	HD (High Density format)
15:13	–	Reserved
12:8	0x0	S (Samples per Converter per Frame). Binary value minus 1.

Table 50: **CTRL_RX_ILA_CFG5** *(cont'd)*

Bits	Default Value	Description
0	–	SCR (Scrambling Enable) [RX only, not writeable for TX] 1 = enabled

Notes:

1. This is a *Per Lane* Register.

Table 51: **CTRL_RX_ILA_CFG6**

Bits	Default Value	Description
31:17	–	Reserved
16	0	ADJDIR (Adjust Direction) [Subclass 2 Only]. Binary value.
15:9	–	Reserved
8	–	PHADJ (Phase Adjust Request) [Subclass 2 Only]. Binary value.
7:4	–	Reserved
3:0	0x0	ADJCNT (Phase Adjust Count) [Subclass 2 Only]. Binary value. RX: captured configuration data from the ILA sequence (per lane). TX: Sets the values to be transmitted in the ILA sequence for all lanes.

Notes:

1. This is a *Per Lane* Register.

Table 52: **CTRL_RX_ILA_CFG7**

Bits	Default Value	Description
31:24	–	Reserved
23:16	0x0	FCHK (Checksum) [RX only, not writeable for TX]. Binary value.
15:8	0x0	RES2 (Reserved Field 2)
7:0	0x0	RES1 (Reserved Field 1)

Notes:

1. This is a *Per Lane* Register.

Table 53: **CTRL_TX_GT**

Bits	Default Value	Description
31:5	–	Reserved
4	0	TXINHIBIT Set High to inhibit transmission of TX data
3	0	TXELECIDLE Set High to force idle signal on transmitter output
2:1	0x0	TXPD TX power down

Table 53: **CTRL_TX_GT** *(cont'd)*

Bits	Default Value	Description
0	0	TXPOLARITY Set High to invert the polarity of the outgoing TX data

Notes:

1. This is a *Per Lane* Register for Versal adaptive SoCs only.

Table 54: **CTRL_RX_GT**

Bits	Default Value	Description
31:3	–	Reserved
2:1	0x0	RXPD RX power down
0	0	RXPOLARITY Set High to invert the polarity of the incoming RX data.

Notes:

1. This is a *Per Lane* Register for Versal adaptive SoCs only.

Table 55: **CTRL TX VERSAL GTY/GTYP**

Bits	Default Value	Description
31:15	–	Reserved
14:10	0x0	TXPOSTCURSOR Transmitter post-cursor pre-emphasis control
9:5	0x0	TXPRECURSOR Transmitter pre-cursor pre-emphasis control
4:0	0x18	TXDIFFCTRL Driver swing control

Notes:

1. This is a *Per Lane* Register for GTY/GTYP Versal adaptive SoCs only.

Table 56: **CTRL TX VERSAL GTM**

Bits	Default Value	Description
31	–	Reserved
30:25	0x0	TXPOSTCURSOR Transmitter post-cursor pre-emphasis control
24:19	0x0	TXPRECURSOR3 Transmitter pre-cursor3 pre-emphasis control
18:13	0x0	TXPRECURSOR2 Transmitter pre-cursor2 pre-emphasis control
12:7	0x0	TXPRECURSOR Transmitter pre-cursor pre-emphasis control

Table 56: **CTRL TX VERSAL GTM** *(cont'd)*

Bits	Default Value	Description
6:0	0x0	TXMAINCURSOR Driver swing control

Notes:

1. This is a *Per Lane* Register for GTM Versal adaptive SoCs only.

Chapter 4

Designing with the Core

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

General Design Guidelines

This section describes the steps required to turn a JESD204C core into a fully-functioning design with user-application logic. It is important to know that not all implementations require all of the design steps listed in this chapter. Follow the logic design guidelines in this manual carefully.

Use the Example Design

Each instance of the JESD204C 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.

Know the Degree of Difficulty

JESD204C designs are challenging to implement in any technology, and the degree of difficulty is further influenced by:

- Maximum system clock frequency
- Targeted device architecture
- Nature of your application

All JESD204C implementations require careful consideration of system performance requirements. Pipelining, logic mapping, placement constraints, and logic duplication are all methods that help boost system performance.

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 AMD 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.

Use Supported Design Flows

The core is synthesized in the Vivado IDE and is delivered as Verilog. The example implementation scripts currently provided use Vivado synthesis as the synthesis tool for the IP integrator example design that is delivered with the core. Other synthesis tools can be used.

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.

Recommended Design Experience

Although the JESD204C core is a fully-verified solution. The challenges associated with implementing a complete design vary depending on the configuration and functionality of the application. For best results, previous experience in building high-performance, pipelined FPGA designs using an AMD implementation tools and the XDC is recommended.

Contact your local representative for a closer review and estimation for your specific requirements.

Subclass Mode

The JESD204C core supports operation in two JESD204C Subclass modes (0 and 1) for 64B66B linecoding, and three Subclass modes (0, 1, and 2) for 8B10B linecoding.

This is controlled by a register setting. By default, the core operates in Subclass 1 mode.

The core pinout for 64B66B supports both subclass modes of operation, however an externally generated SYSREF is required for Subclass 1 operation. For Subclass 0, the SYSREF input signal is not required and can be tied off.

The core pinout for 8B10B supports all three subclass modes of operation, however an externally generated SYSREF is required for Subclass 1 operation. For Subclasses 0 and 2, the SYSREF input signal is not required and can be tied off.

Subclass 0

Subclass 0 is supported for both 64B66B and 8B10B linecoding. Subclass 0 does not support Deterministic Latency, and the SYSREF input is not required.

Subclass 1

Subclass 1 is supported for both 64B66B and 8B10B linecoding. Subclass 1 supports deterministic latency through the use of a common SYSREF signal between the converter and logic device. The SYSREF signal is generated external to the core, and is distributed to all devices within a system. SYSREF is permitted by the JESD204C standard to be either a *one-shot*, *periodic*, or *gapped periodic*. The JESD204C core is capable of operating with any of these selections. The timing and clocking requirements for the reliable capture of SYSREF are key to achieving reliable deterministic latency.

Subclass 2

Subclass 2 is only supported for 8B10B linecoding. Subclass 2 supports deterministic latency using only the `SYNC` signal. The timing and clocking requirements for the launch (by an RX core), and capture (by a TX core) of the SYNC signal are key to achieving reliable deterministic latency. Care must be taken to ensure the timing of this signal is met.

Programming the Core

Run time operation of the JESD204C core is configured through an AXI4-Lite register interface. See Register Space for details of the register map and available configuration registers.

For correct operation and bring-up of a JESD204C link, it is important that the major framing and link operation parameters match at both ends of the link. These parameters are determined by the configurations available in the ADC/DAC converter device to which the core is interfacing.

For 64B66B Linecoding, these are:

- Meta Mode
- Multiblocks in Extended Multiblock
- Subclass mode

- SYSREF handling (for subclass 1 mode)

For 8B10B Linecoding, these are:

- Octets per frame
- Frames per Multiframe
- Scrambling On/Off
- Subclass mode
- SYSREF handling (for subclass 1 mode)

For 8B10B transmitter cores, in addition to the above parameters, some of the additional content of the configuration data which is transmitted in the ILA sequence at link start-up is also programmed through the register interface. The data values transmitted in the ILA configuration data are not normally critical to the operation of the link, but this is dependent on the behavior of the receiving device.

For 8B10B receive cores, the configuration data received in the ILA sequence is captured for each lane and can be examined using the register interface.

After programming the link parameters, the JESD204C core must be reset to restart the link using the newly programmed values. If the JESD204C core is not reset after programming, the new parameters will not be used.

Clocking

IMPORTANT! *It is strongly recommended that you use one of the clocking schemes presented in this section. Use of alternative clocking schemes might lead to design failure.*

The JESD204C specification does not define specific serial line rates for any JESD204C link, but a valid range of line rates from 312.5 Mbps to 32.5 Gbps. The JESD204C core supports 8B10B linecoding at line rates from 1 Gbps to 16.375 Gbps (depending on the part and speed grade selection) and 64B66B linecoding at line rates from 1 Gbps to 32.5 Gbps (depending on the part and speed grade selection). In most instances, the serial line rate selection is governed by the specifications of the ADC/DAC device(s) to which the core is interfaced. The required operating serial line rate directly relates to the clock rate at which the core logic operates (core clock); the serial line rate also governs the selection of the reference clock required by the transceiver(s).

Core Clock 64B66B Linecoding

The JESD204C 64B66B core operates using a 64-bit (8-byte) datapath. The core clock frequency is always the line rate divided by 66. For example, for a serial line rate of 16.5 Gbps, the core clock frequency is 250 MHz.

The AXI4-Stream RX/TX Data and Cmd interfaces operate at this core clock frequency. TX and RX core clock should be used as the clock source for these interfaces.

Core Clock 8B10B Linecoding

The JESD204C 8B10B core operates using a 32-bit (4-byte) datapath. The core clock frequency is always the line rate divided by 40. For example, for a serial line rate of 12.5 Gbps, the core clock frequency is 312.5 MHz.

The AXI4-Stream RX and TX Data interfaces operate at this core clock frequency. TX and RX core clock should be used as the clock source for these interfaces.

Reference Clock

In AMD Versal™ adaptive SoCs, the GTY, GTYP, and GTM serial transceivers require a stable, low-jitter reference clock which has a device and speed grade-dependent range. For AMD UltraScale+™ and AMD UltraScale™ devices, the GTH/GTY serial transceivers in the JESD204_PHY require a stable, low-jitter reference clock which has a device and speed grade-dependent range.

In some circumstances, it can be advantageous to use the same clock frequency for both core clock and reference clock. However, this might not always be practical. It is important to understand the limitations imposed on the reference clock and core clock, together with system-level implications such as the synchronous capture of SYSREF for Subclass 1.

JESD204_PHY Outclocks for UltraScale+/UltraScale Devices

The JESD204_PHY IP core generates the clocks and `rxoutclk` at the correct frequency to use as `core_clk`. However, the output phase of these clocks varies from reset to reset. This means that for JESD204 interfaces, the JESD204_PHY IP core `outclock` ports might only be used to drive `core_clk` when Subclass 0 is used, and deterministic latency is not required. For systems that require deterministic latency and therefore use Subclass 1 or Subclass 2, the JESD204_PHY IP core `outclocks` should not be used.

Versal Adaptive SoC Transceiver Outclocks

The Versal Adaptive SoC Transceiver Wizard generates `txoutclk` and `rxoutclk` at the correct frequency to use as `core_clk`. However, the output phase of these clocks varies from reset to reset. This means that for JESD204 interfaces the Versal Adaptive SoC Transceiver `outclock` ports might only be used to drive `core_clk` when Subclass 0 is used and deterministic latency is not required. For systems that require deterministic latency and therefore use Subclass 1 or Subclass 2, the Versal Adaptive SoC Transceiver `outclocks` should not be used.

The rest of this section details the valid clocking architecture options.

Versal Adaptive SoC GTM Transceiver Clocking

Versal adaptive SoC GTM transceivers do not contain any internal encoding options; instead, they operate in raw mode. Both the 8b10b and the 64b66b encoding and decoding functions are instead performed by the JESD204C IP core.

In 64b66b mode the JESD204C IP core requires both a 64-bit clock and a 66-bit clock for the gearbox of the encoder and decoder. The JESD204C `core_clk` supplies the 64-bit clock. The 66-bit clock (`txusrclk/rxusrclk`) is sourced from the Versal Adaptive SoC Transceiver `outclock` port via a `bufg_gt`. Generate the example design for an illustration of how to connect these clocks.

In 8b10b mode no extra clocking is required.

AXI4-Lite Interface Clock

The JESD204C core is configured and monitored through an AXI4-Lite processor interface. The clock for this interface is separate and independent from the core and reference clocks. The AXI4-Lite clock must be a free running clock as it is also used to clock the reset circuitry. The valid range is between 10 MHz and 200 MHz.

DRP Clock for UltraScale+/UltraScale Devices

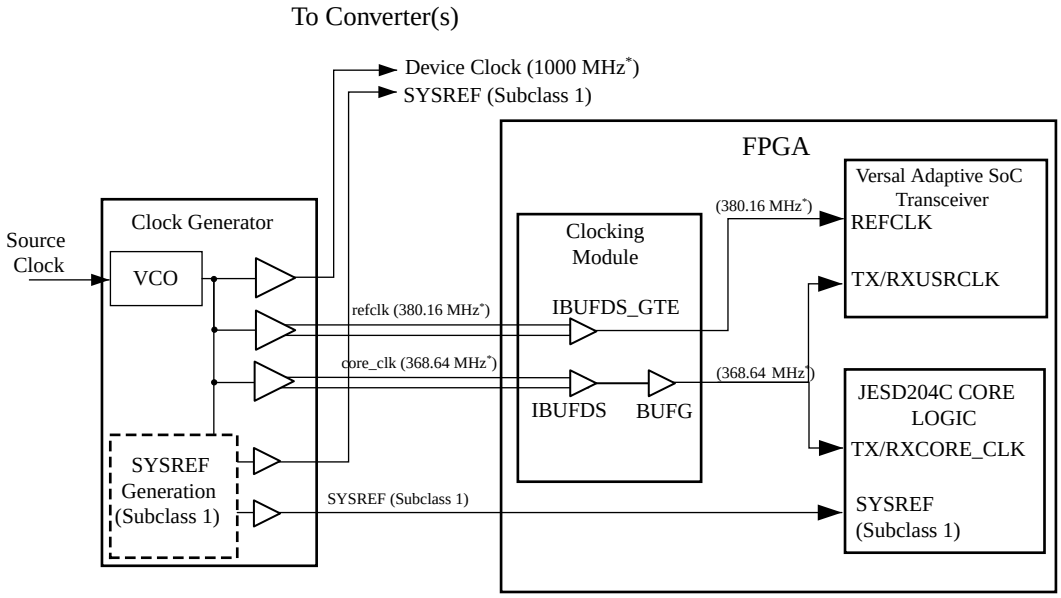
JESD204C system implementation requires the use of a JESD204 PHY core. The JESD204_PHY core must be supplied with a DRP clock (see *JESD204 PHY LogiCORE IP Product Guide* (PG198)).

Separate Transceiver Reference and Core Clocks

For JESD204C, the most generic and flexible clocking scheme uses separate transceiver reference and JESD204C core clocks supplied to the FPGA. In this configuration, the reference and core clocks are physically separate and can be run at independent, but related, frequencies, without additional constraints.

The reference clock can be run at any frequency within the limitations of the transceiver for the selected line rate. The core clock always runs at the required rate (1/66th or 1/40th of the serial line rate). This configuration is shown in the following two figures for 64B66B and 8B10B line coding.

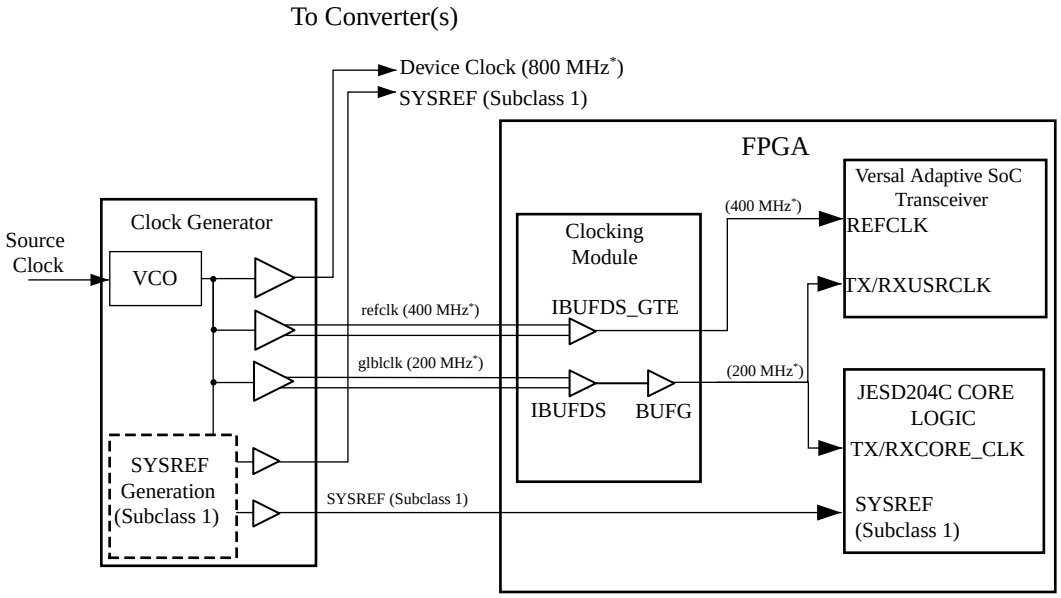
Figure 1: **Separate Transceiver Reference and Core Clocks: 64B66B Example for Versal Adaptive SoCs**



* example frequencies. 64B66B Line Rate = 24.33024 Gb/s

X24019-121024

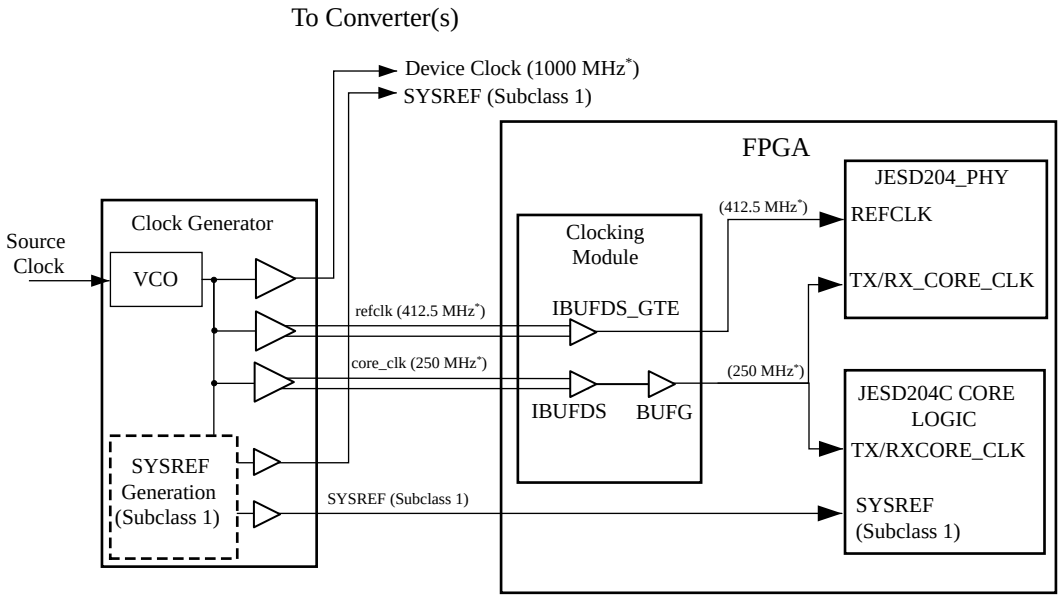
Figure 2: **Separate Transceiver Reference Clock and Core Clock: 8B10B Example for Versal Adaptive SoCs**



* example frequencies. 8B10B Line Rate = 8.0 Gb/s

X24025-121024

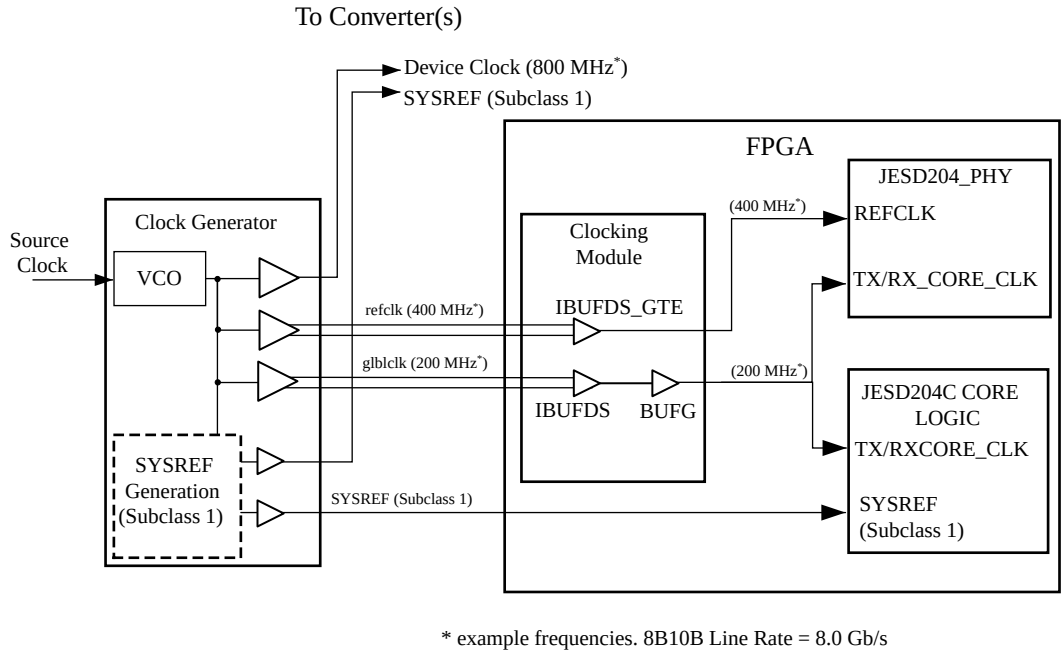
Figure 3: **Separate Transceiver Reference and Core Clocks: 64B66B Example for UltraScale+/UltraScale Devices**



* example frequencies. 64B66B Line Rate = 16.5 Gb/s

X24222-121024

Figure 4: **Separate Transceiver Reference Clock and Core Clock: 8B10B Example for UltraScale+/UltraScale Devices**



Transceiver Reference Clock Used as Core Clock

For some systems, it is possible to run a single clock input which acts as both the transceiver reference clock and the JESD204C core clock. While this configuration can sometimes simplify a system design, it is not always compatible.

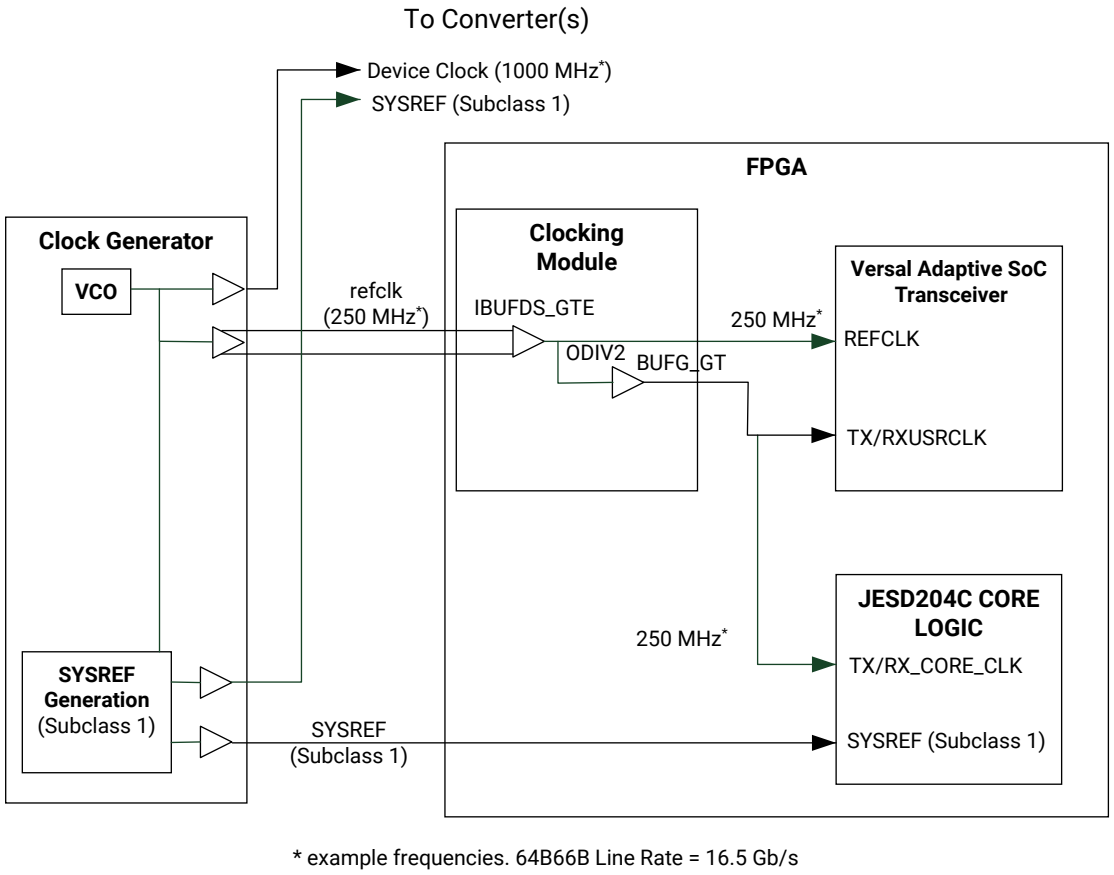
For 64B66B systems in Versal adaptive SoCs, the required core clock frequency is usually not suitable for use as a reference clock for a Versal Adaptive SoC Transceiver configured to use the RPLL. Therefore, the LCPLL should be used if a single clock is required. In this configuration, the input transceiver reference clock must always be the required rate (1/66th of the serial line rate for 64B66B systems or 1/40th of the serial line rate for 8B10B systems).

For 64B66B systems in UltraScale+/UltraScale devices, the required core clock frequency is not suitable for use as a reference clock for a JESD204 PHY configured to use the CPLL (therefore QPLL 0 or 1 must be used if a single clock is required, or two clocks must be supplied if the CPLL must be used). In this configuration, the input transceiver reference clock must always be the required rate (1/66th of the serial line rate for 64B66B systems or 1/40th of the serial line rate for 8B10B systems).

***Note*:** When using this clocking scheme in UltraScale+/UltraScale devices, the signal `GT_POWERGOOD` output from the JESD204_PHY must be connected to the CE pin on the BUFG_GT used to source `core_clk` from `refclk`.

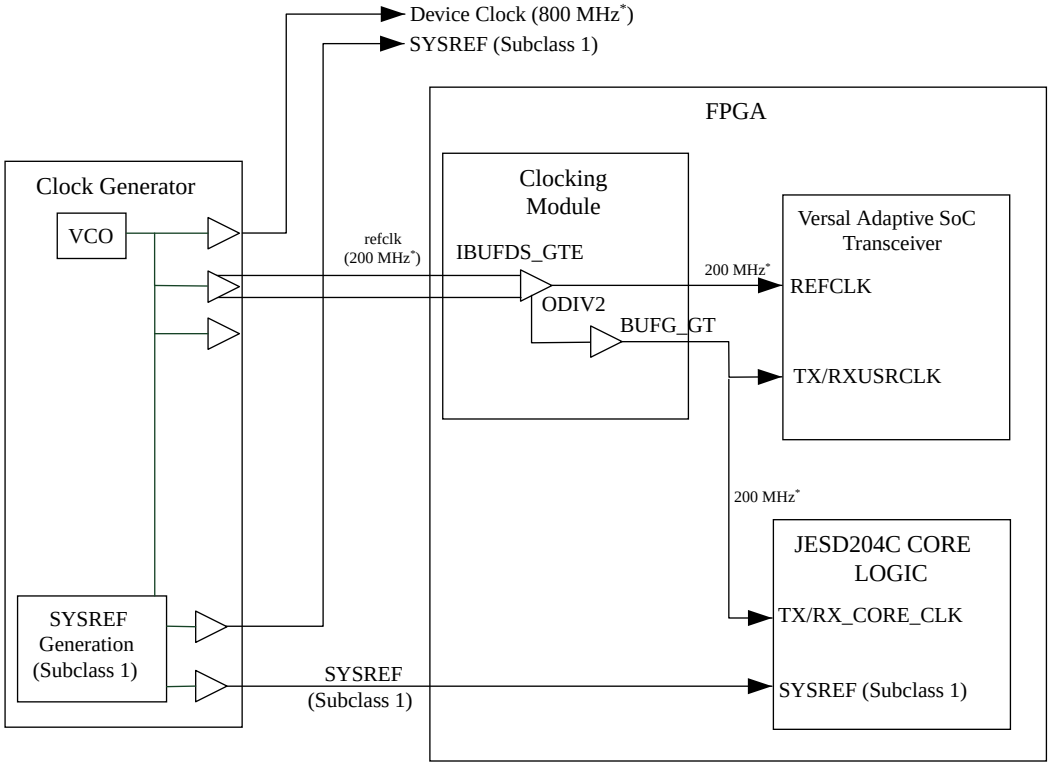
The configurations are shown in the following figures.

Figure 5: **Transceiver Reference Clock Used as Core Clock 64B66B Example for Versal Adaptive SoCs**



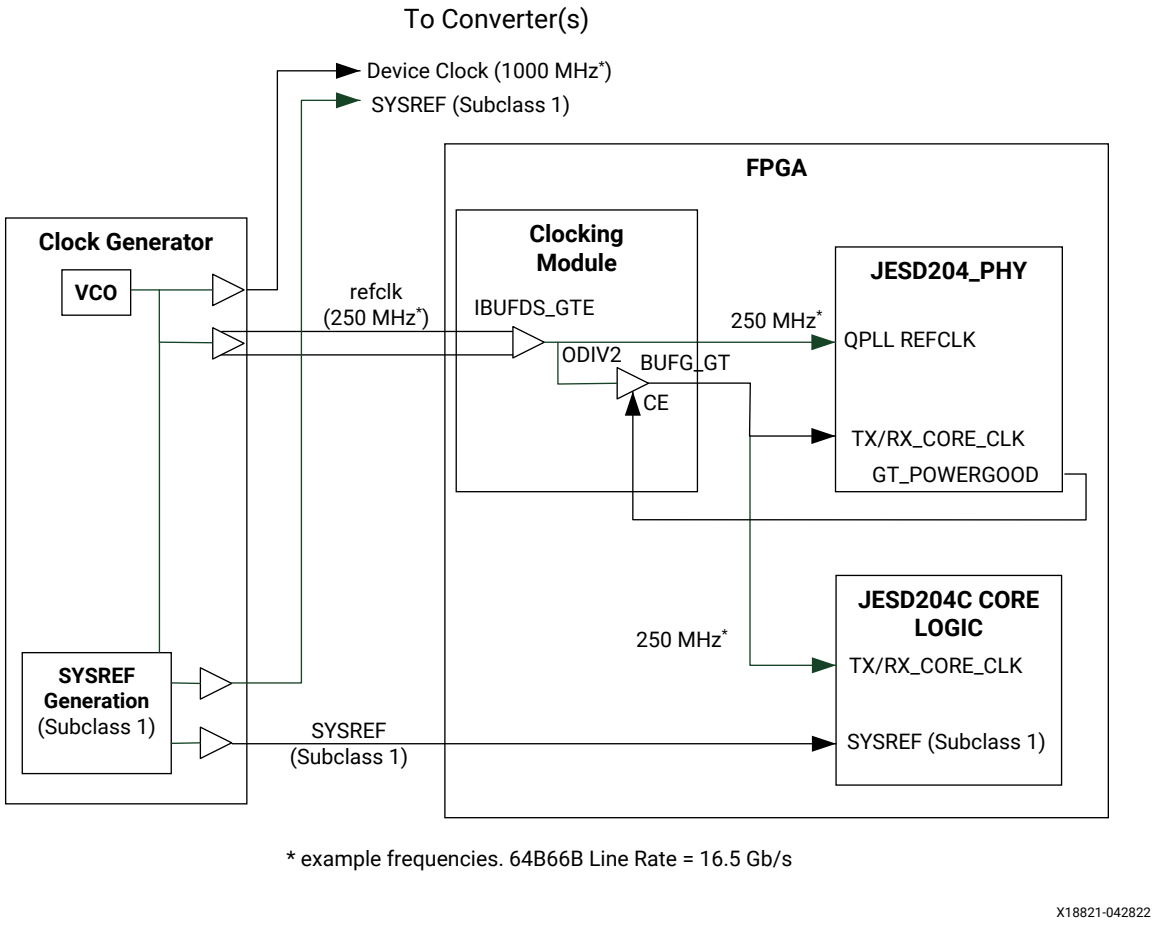
***Note*:** The ODIV2 output from the IBUFDS_GTE should be used to drive the I input of the BUFG_GT. Using the O output of the IBUFDS_GTE is not allowed and will result in a DRC error.

Figure 6: **Transceiver Reference Clock Used as Core Clock 8B10B Example for Versal Adaptive SoCs**



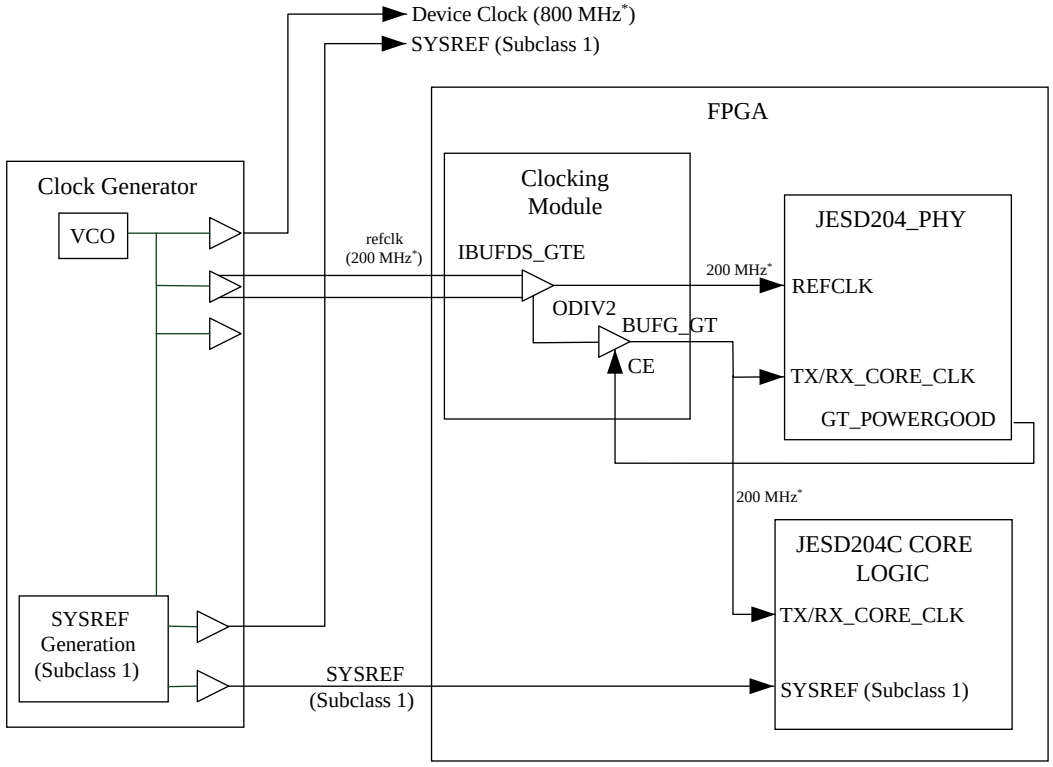
***Note*:** The ODIV2 output from the IBUFDS_GTE should be used to drive the I input of the BUFG_GT. Using the O output of IBUFDS_GTE is not allowed and will result in a DRC error.

Figure 7: **Transceiver Reference Clock Used as Core Clock 64B66B Example for UltraScale+/UltraScale Devices**



***Note*:** The ODIV2 output from the IBUFDS_GTE should be used to drive the I input of the BUFG_GT. Using the O output of IBUFDS_GTE is not allowed and will result in a DRC error.

Figure 8: **Transceiver Reference Clock Used as Core Clock 8B10B Example for UltraScale+/UltraScale Devices**



***Note*:** The ODIV2 output from the IBUFDS_GTE should be used to drive the I input of the BUFG_GT. Using the O output of IBUFDS_GTE is not allowed and will result in a DRC error.

Transceiver Reference Clock Used as Core Clock

For Subclass 0 only operation, the timing limitations imposed to support deterministic latency are removed, and a simplified clocking arrangement can be used which requires only a reference clock input. In this case, the transceiver PLL is used to generate the core clock signal. In this configuration, any clock frequency that is suitable to use as the transceiver reference cock is acceptable.

The Versal adaptive SoCs and UltraScale+/UltraScale device configurations are shown in the following figures.

This configuration is not suitable for subclass 1 or 2 operation because the output phase of the transceiver PLL is unknown, and therefore, this clock cannot be used to reliably sample SYSREF or SYNC.

Figure 9: **Transceiver Output Clock Used as Core Clock (Subclass 0) for Versal Adaptive SoCs**

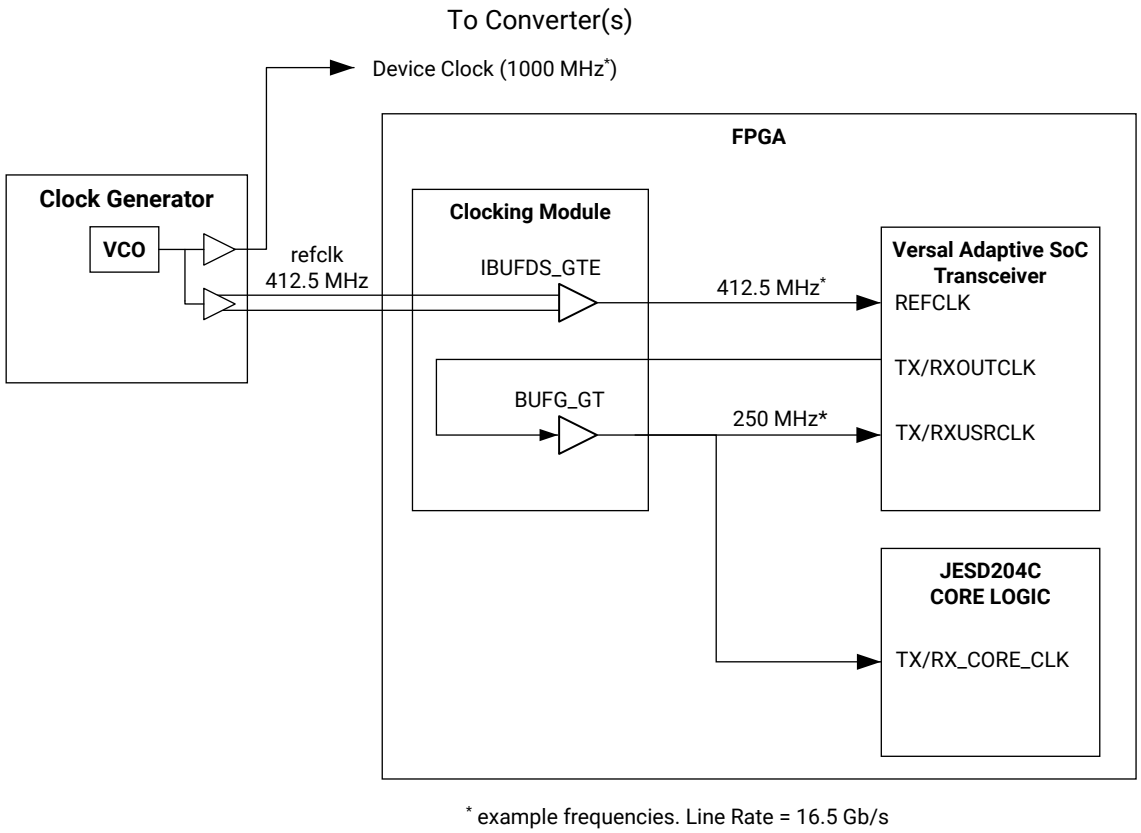
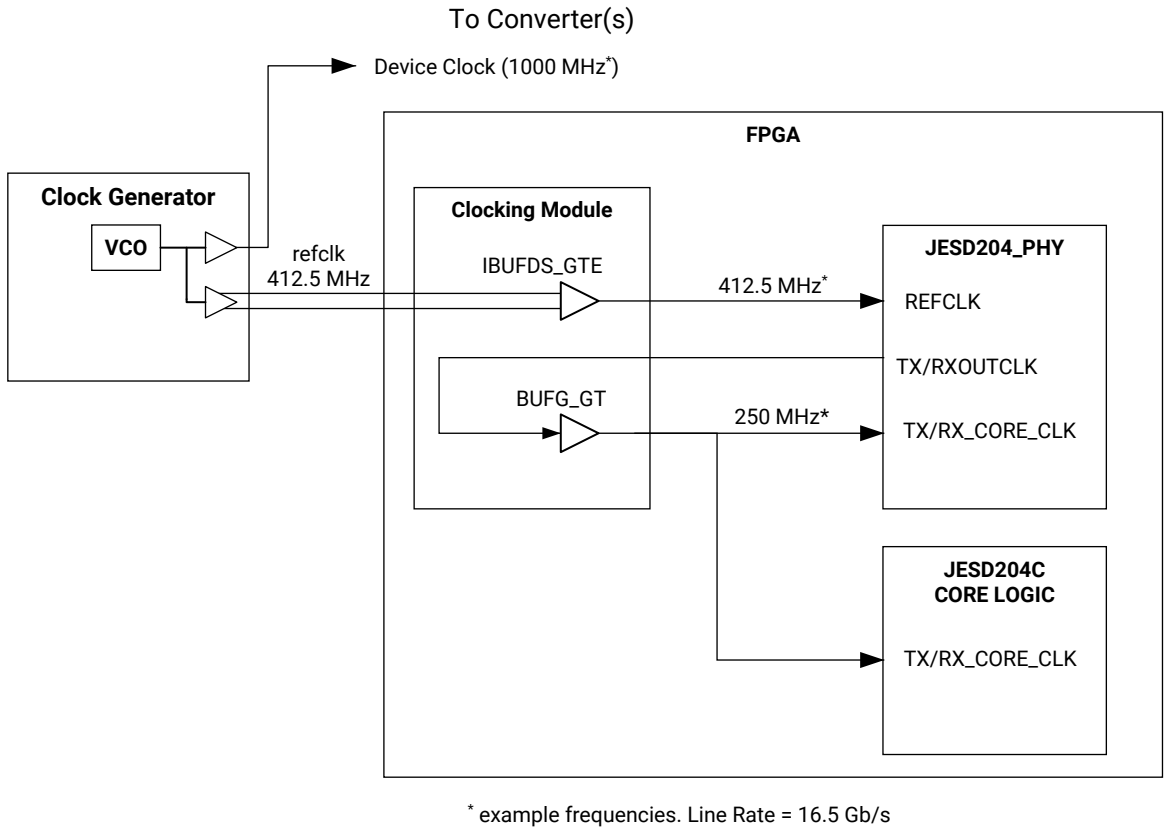


Figure 10: **Transceiver Output Clock Used as Core Clock (Subclass 0) for UltraScale+/ UltraScale Devices**



Clocking Considerations

- Always refer to the device data sheet for the chosen part and speed grade to confirm which PLLs are available for a required line rate; PLL selection for a particular rate might not be arbitrary.

  Versal adaptive SoCs:

  - If the RPLL is required, the transceiver reference clock cannot be used as the core clock when the core is configured for 64B66B linecoding because the acceptable reference clock input frequencies to the RPLL do not cover the required Line Rate/66 ratio. This restriction does not apply when the core is configured for 8B10B linecoding.

  UltraScale+/UltraScale:

- If the CPLL is required, the transceiver reference clock cannot be used as the core clock when the core is configured for 64B66B linecoding because the acceptable reference clock input frequencies to the CPLL do not cover the required Line Rate/66 ratio. This restriction does not apply when the core is configured for 8B10B linecoding.
- For Line rates above 16.375 Gbps, ensure only port `MGTREFCLK0` is used to drive QPLL0, and `MGTREFCLK1` to drive QPLL1.

Resets

The reset inputs and outputs on the JESD204C core are as shown in the following table.

Table 57: **JESD204C Resets**

Reset	Description
tx/rx_core_reset	This reset input is asynchronous and active-High. This reset input will reset the JESD204C core logic but does not reset the AXI4-Lite register interface, so all programmed register values will be maintained.
s_axi_aresetn	This reset input must be synchronized with the AXI4-Lite interface clock. This reset input will reset the AXI4-Lite register interface.
tx/rx_aresetn	This reset output is synchronous to tx/rx_core_clk. This output is an AXI4-Stream interface reset signal to be used with the AXI4-Stream RX/TX Data and Cmd interfaces.
Versal Adaptive SoCs	
chN_txmstdatapathreset[1]	This active-High reset output is connected to the Versal Adaptive SoC Transceiver using Block Automation. The output is driven by internal logic in the JESD204C core. N = Lanes - 1
chN_txmstreset[1]	This active-High reset output is connected to the Versal Adaptive SoC Transceiver using Block Automation. The output is driven by internal logic in the JESD204C core. N = Lanes - 1
chN_txpmaresetdone[1]	This active-High reset input is connected to the Versal Adaptive SoC Transceiver using Block Automation. The input holds the JESD204C core in reset until it asserts High. N = Lanes - 1
chN_txuserrdy[1]	This active-High reset output is connected to the Versal Adaptive SoC Transceiver using Block Automation. The output is driven by internal logic in the JESD204C core. N = Lanes - 1
chN_txmstresetdone[1]	This active-High reset input is connected to the Versal Adaptive SoC Transceiver using Block Automation. The input holds the JESD204C core in reset until it asserts High. N = Lanes - 1
chN_rxmstdatapathreset[1]	This active-High reset output is connected to the Versal Adaptive SoC Transceiver using Block Automation. The output is driven by internal logic in the JESD204C core. N = Lanes - 1

Table 57: **JESD204C Resets** *(cont'd)*

Reset	Description
chN_rxmstreset[1]	This active-High reset output is connected to the Versal Adaptive SoC Transceiver using Block Automation. The output is driven by internal logic in the JESD204C core. N = Lanes - 1
chN_rxpmaresetdone[1]	This active-High reset input is connected to the Versal Adaptive SoC Transceiver using Block Automation. The input holds the JESD204C core in reset until it asserts High. N = Lanes - 1
chN_rxuserrdy[1]	This active-High reset output is connected to the Versal Adaptive SoC Transceiver using Block Automation. The output is driven by internal logic in the JESD204C core. N = Lanes - 1
chN_rxmstresetdone[1]	This active-High reset input is connected to the Versal Adaptive SoC Transceiver using Block Automation. The input holds the JESD204C core in reset until it asserts High. N = Lanes - 1
reset_all[2]	Reset output to the `gtwiz_versal` subsystem reset controller. Resets TX PLL and datapath.
reset_tx_pll_and_datapath[2]	Reset output to the `gtwiz_versal` subsystem reset controller. Resets TX PLL and datapath.
reset_tx_datapath[2]	Reset output to the `gtwiz_versal` subsystem reset controller. Resets TX datapath only.
reset_tx_done[2]	TX Reset Done input from the `gtwiz_versal` subsystem reset controller. Active-High when TX reset sequence is complete.
reset_rx_pll_and_datapath[2]	Reset output to the `gtwiz_versal` subsystem reset controller. Resets RX PLL and datapath.
reset_rx_datapath[2]	Reset output to the `gtwiz_versal` subsystem reset controller. Resets RX datapath only.
reset_rx_done[2]	RX Reset Done input from the `gtwiz_versal` subsystem reset controller. Active-High when RX reset sequence is complete.
UltraScale+/UltraScale Devices	
tx/rx_reset_gt	This reset output must be connected to the JESD204_PHY core. This signal is used to initiate a JESD204_PHY GT reset sequence.
tx/rx_reset_done	This input must be connected to the JESD204_PHY core. This signal is used to hold the JESD204C core in reset until completion of the JESD204_PHY GT reset sequence. ***Note*:** A Low input on this port forces the JESD204C core into a reset state.

Notes:

1. These ports are present only when you select the Legacy GT Wizard in JESD204C GUI.
2. These ports are present only when you select the GT Wizard subsystem in JESD204C GUI.

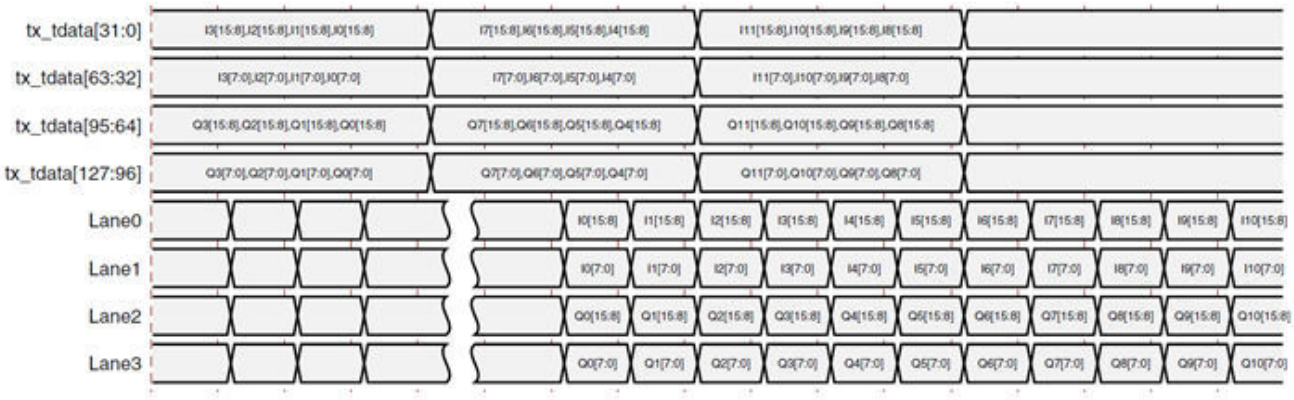
Data and Command Interfaces

The transmitter and receiver cores incorporate AXI4-Stream interfaces for data ingress and egress. These AXI4-Stream interfaces include data and flow control signals only. In addition, there are supplementary control signals that are used to signal the timing of the data on the AXI4-Stream interface.

The AXI data input and output by the core contains 4-bytes per clock cycle per lane for 8B10B and 8-bytes per clock cycle per lane for 64B66B. The least significant byte position in each 32-bit or 64-bit block holds the first byte received from the ADC or transmitted to the DAC. The following figure shows an example of how AXI4-Stream data is mapped on to JESD204C using 8B10B encoding.

***Note*:** When you use the 64B66B encoding, the command interface operates in Multi-Lane mode. The JESD204C core uses the header ports of all available lanes to form the command channel.

Figure 11: **8B10B Encoding - 4 Lane DAC, Frame size (F) = 1, 16 Bit I and Q with I Sample in Lanes 0 & 1 and Q Sample in Lanes 3 & 4**



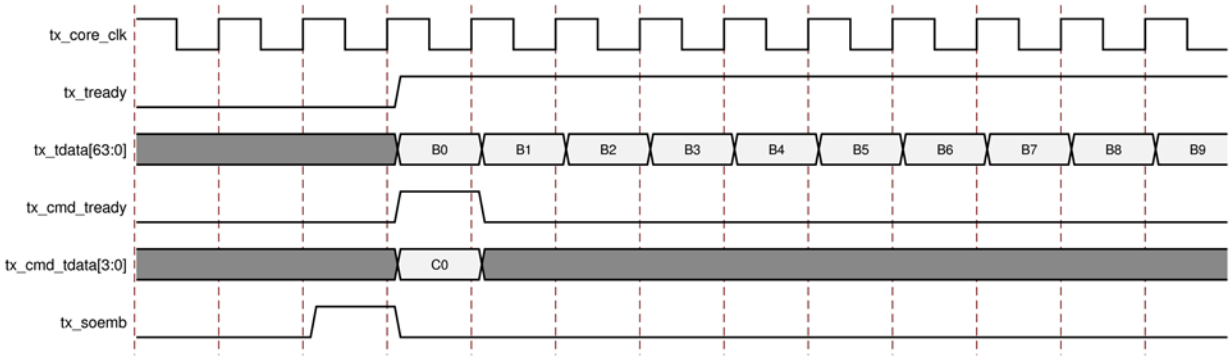
The following figure shows an example of how AXI4-Stream data is mapped on to JESD204C using 64B66B encoding.

Figure 12: **64B66B Encoding - 4 Lane DAC, Frame size (F) = 1, 16 Bit I and Q with I Sample in Lanes 0 & 1 and Q Sample in Lanes 3 & 4**

***Note*:** The AXI4-Stream interfaces transfer the JESD204C transport layer—not raw converter samples. Refer to the appropriate converter data sheet for information on correctly mapping samples into the transport layer.

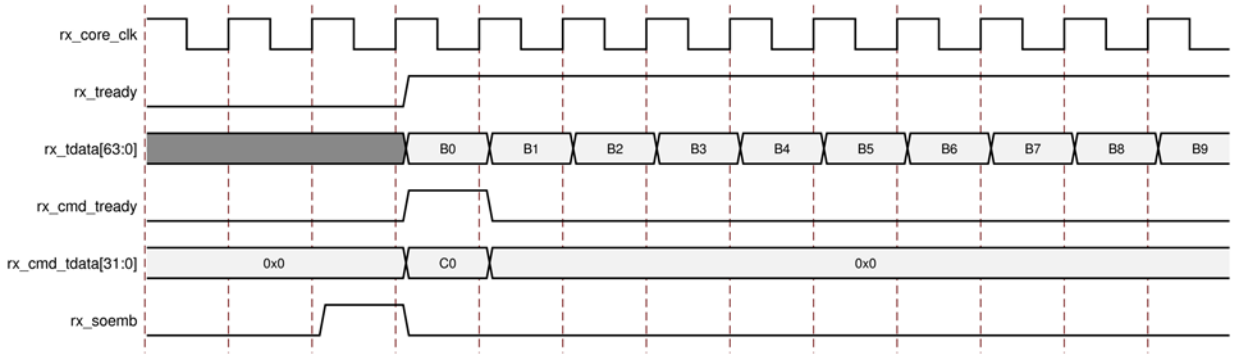
For a 64B66B transmitter, the following figure shows the timing of the `tx_soemb` (Start Of Extended Multiblock) signal relative to the AXI4-Stream data `tx_tdata` and `tx_cmd_tdata`. The `tx_soemb` signal is a single bit and it is set High in the cycle preceding the first data block of an extended multiblock. The data interface will transfer one 64-bit block B every core clock cycle. The command interface will transfer one command word (either 19-bit or 7-bit) every multiblock. If data is not available on the command interface (`tx_cmd_tvalid` = 1), then an IDLE command will be transmitted. The command interface has been padded out to 32 bits per lane. Only bits [18:0] or [6:0] are actually used, and all unused bits are set to 0.

Figure 13: **64B66B Transmit Data Interface Timing**



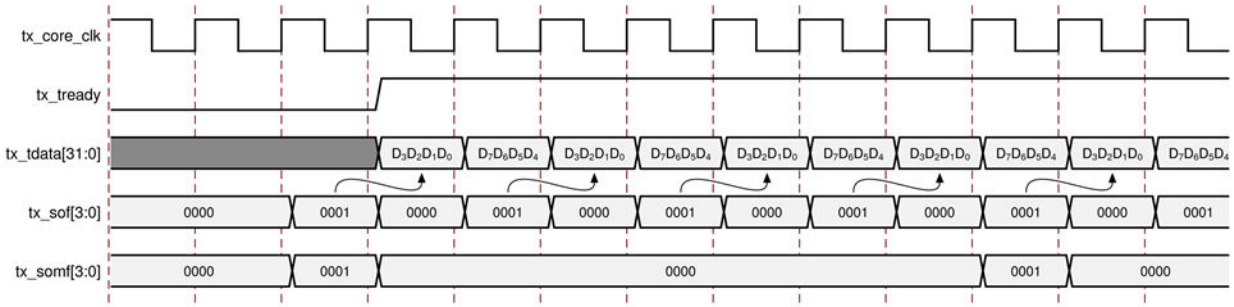
For a 64B66B receiver, the following figure shows the timing of the `rx_soemb` signal relative to the AXI4-Stream data `rx_tdata`. The `rx_soemb` signal is a single bit, and it is set High in the cycle preceding the first data block of an extended multiblock. The command interface will transfer one word every multiblock.

Figure 14: **64B66B Receive Data Interface Timing**



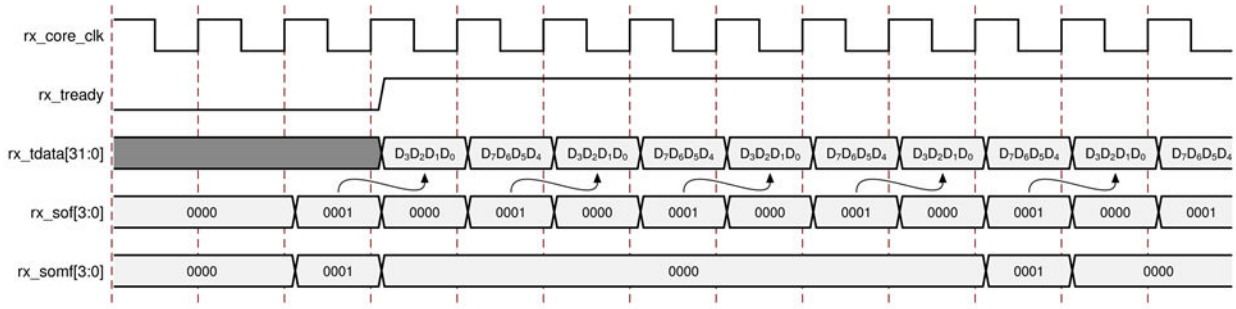
For a 8B10B transmitter, the following figure shows the timing of `tx_sof` (Start Of Frame) and `tx_somf` (Start Of Multiframe) signals relative to the AXI data `tx_tdata`. `tx_sof` and `tx_somf` are fixed at four bits wide because the internal data width of each lane is 32 bits, and the start of frame (or multiframe) can occur in any of the four byte positions of the 32-bit word. For multilane configurations, the start of frame (or multiframe) signal indicates the byte position of the first byte of a frame in `tx_tdata[31:0]`, `tx_tdata[63:32]`, `tx_tdata[95:64]`, etc. For example, in a four lane configuration when `tx_sof` = 0001, the first byte of four new frames appears in `tx_tdata` in a single cycle, `tx_tdata[7:0]`, `tx_tdata[39:32]`, `tx_tdata[71:64]`, and `tx_tdata[103:96]`.

Figure 15: **8B10B Transmit Data Interface Timing for F = 8 and K = 4**



For a 8B10B receiver, the following figure shows the timing of `rx_sof` (Start Of Frame) and `rx_somf` (Start Of Multiframe) signals relative to the AXI data `rx_tdata`. `rx_sof` and `rx_somf` are fixed at four bits wide because the internal data width of each lane is 32 bits, and the start of frame (or multiframe) can occur in any of the four byte positions of the 32-bit word. For multilane configurations, the start of frame (or multiframe) signal indicates the byte position of the first byte of a frame in `rx_tdata[31:0]`, `rx_tdata[63:32]`, `rx_tdata[95:64]`, etc. For example, in a four lane configuration when `rx_sof` = 0001, the first byte of four new frames appears in `rx_tdata` in a single cycle, `rx_tdata[7:0]`, `rx_tdata[39:32]`, `rx_tdata[71:64]`, and `rx_tdata[103:96]`.

Figure 16: **8B10B Receive Data Interface Timing for F = 8 and K = 4**



SYSREF

SYSREF Timing

When the JESD204C is used in Subclass 1, the `SYSREF` signal is the master timing reference for the system. To achieve accurate deterministic latency, the `SYSREF` signal must be captured synchronously to the core clock. To achieve this, the SYSREF period must be a multiple of 4-byte clock periods for 8B10B linecoding and 8-byte clock periods for 64B66B linecoding. This is because the core uses a 4-byte or 8-byte internal datapath for 8B10B and 64B66B, respectively.

SYSREF Type

The accurate capture of SYSREF is critical in Subclass 1 operation. The JESD204C Specification allows SYSREF to be generated in any of the following ways:

- Periodic
- One-shot
- *Gapped* Periodic

For maximum flexibility, the JESD204C core provides several options for how SYSREF is handled for Subclass 1 operation.

Because the JESD204C core operates using a 32-bit (4-byte) or 64-bit (8-byte) datapath, if a periodic or gapped periodic SYSREF is used in the system, the following conditions must be met:

- For 64B66B linecoding, the period must be an integer multiple of the Extended Multiblock period.
- For 8B10B linecoding, the period must be an integer multiple of the multiframe period. It must also be a multiple of 4-byte clocks. Care must be taken to ensure both of these conditions are met if the multiframe period is not a multiple of 4-byte clocks.

SYSREF Handling

SYSREF Delay

The SYREF Delay bits in the CTRL_SYSREF register can be used to add delay to the `SYSREF` signal after it is captured (see Table 25: CTRL_SYSREF). This allows the effective phase of the local multiframe clock (LMFC)/ local extended multiblock clock (LEMC) to be adjusted. The value programmed into the SYSREF delay register equates to the number of core clock cycles that SYSREF will be delayed by.

For 8B10B linecoding, the deterministic latency mechanism, as defined in the JESD204C standard, requires that the multiframe size be larger than the maximum possible delay across the link. In practice, this can be difficult to achieve, particularly with small frame sizes. However, as long as the multiframe size is greater than the maximum variation between lanes in delay across the link, then deterministic latency can be achieved. A potential issue occurs when the maximum lane delay variation causes the overall latency to straddle the boundary between two adjacent LMFC periods. In such a case, latency variations of exactly one LMFC period can be observed between system restarts. In this case, the SYSREF might be delayed to adjust the LMFC boundary position to alleviate the problem.

For 64B66B linecoding, the deterministic latency mechanism defined in the JESD204C standard requires that the maximum variation between lanes in delay across the link be less than the Extended Multiblock size. However, the alignment buffer in the 64B66B RX IP core is fixed at a size of two multiblocks. Care should be taken when designing systems where the number of Multiblocks in an Extended Multiblock (MB_IN_EMB) is greater than two because this can cause the buffers to overflow if more than two Multiblocks of data need to be stored before the next LEMC buffer release point. The SYSREF Delay register can be used to offset the phase of the LEMC clock from SYSREF such that even for large values of MB_IN_EMB the RX IP core LEMC can be adjusted such that the required amount of data to be buffered is less than two Multiblocks. The TX IP core also has this register to aid system latency optimization.

SYSREF Always

The SYSREF Always bit in the CTRL_SYSREF register provides the JESD204C core with a programmable option allowing the choice of how a periodic SYSREF is used internally (see Table 25: CTRL_SYSREF).

When SYSREF Always is set to 0, only an initial SYSREF event seen after reset (or on link resynchronization) is used to align the internal LMFC counter. All subsequent SYSREF events will be ignored.

When SYSREF Always is set to 1, all SYSREF events are used to (re)align the LMFC counter. This setting requires that the SYSREF period be a correct multiple of the Multiframe/Extended Multiblock periods.

SYSREF Required

The SYSREF required bit in the CTRL_SYSREF register provides the JESD204C core with a programmable option allowing the choice of whether a SYSREF event is required or not for the link to restart after a resync request (see Table 25: CTRL_SYSREF).

When SYSREF Required is set to 0, a resync request will automatically restart the link.

When SYSREF Required is set to 1, a resync request will stall until a new SYSREF event has been detected.

Capturing SYSREF

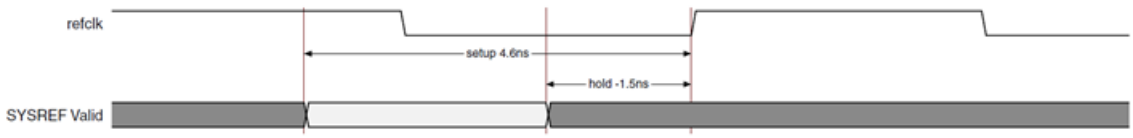
The synchronous capture of SYSREF is critical to the deterministic latency mechanism of JESD204C. By default, no constraints are applied to the SYSREF input. However, the required timing of the SYSREF input can be checked using the `report_datasheet` command in the Vivado Design Suite.

An example timing diagram is shown in the following figure. This example uses the following settings:

`core_clk` period = 6.4 ns (6.25 Gb/s line rate with 8B10B linecoding)

In this example, the `report_datasheet` command gives a setup of 4.6 ns and hold of -1.5 ns for the SYSREF pin.

Figure 17: **SYSREF Timing Example**



The easiest method to ensure a design will be able to reliably capture SYSREF is to use a programmable clock generator chip that allows fine delay adjustment of its outputs, to generate all the JESD204C clocks and SYSREF signals in the system. This will allow for the delay between core clock and SYSREF to be adjusted to meet the setup and hold requirements achieved by your design.

It is also possible to use an MMCM to adjust the phase of core clock internally to align with the setup and hold requirements.

SYSREF on Initial Link Bring-Up with 8B10B Linecoding

After a reset, a JESD204C core configured for subclass 1 operation requires at least one SYSREF event to align the internal LMFC counter, and bring up the link:

- A receive core requires an initial SYSREF event to align the LMFC, and then asserts SYNC on the next LMFC boundary when code group sync has been achieved. The core does not assert SYNC until an initial SYSREF event is detected.
- A transmit core requires a SYSREF event to align the LMFC. The core begins ILA transmission on an LMFC boundary after SYNC is asserted. The core does not begin ILA transmission until an initial SYSREF event is detected.

The system must ensure that SYSREF to the JESD204C core is generated after the core has completed reset. This is of particular importance if the system is operating a one-shot SYSREF.

SYSREF on Link Resynchronization with 8B10B Linecoding

After initial bring-up, if a link resynchronization is requested (by the deassertion of SYNC by the receiving device), the desired core behavior relative to SYSREF can be controlled using the SYSREF Required control bit in the CTRL_SYSREF Handling register.

When SYSREF Required is set to 0, no SYSREF event is required for the link to re-synchronize (the assumption is that LMFC counters continue to free-run and remain valid).

- A receive core asserts SYNC on the next LMFC boundary after code group sync.
- A transmit core transmits the ILA sequence on the next LMFC boundary after SYNC is asserted.

When SYSREF Required is set to 1, a SYSREF event is required for the link to reestablish SYNC following a resync request. In this case, the behavior is the same as the initial link bring up detailed earlier.

This setting is particularly important in systems where a One-Shot SYSREF is used, or where SYSREF is periodic, but SYSREF Always is set to 0.

SYSREF on Initial Link Bring-Up with 64B66B Linecoding

After a reset, a JESD204C core configured for subclass 1 operation requires at least one SYSREF event to align the internal LMBC counter, and bring up the link:

- A receiver core requires an initial SYSREF event to align the LMBC. The core does not start the LEMC counter until an initial SYSREF event is detected.
- A transmitter core requires a SYSREF event to align the LMBC. The core does not begin transmission of multiblocks until an initial SYSREF event is detected. Therefore, a link cannot achieve multiblock lock until after a SYSREF event has been seen by both a transmitter and a receiver.

The system must ensure that SYSREF to the JESD204C core is generated after the core has completed reset. This is of particular importance if the system is operating a One-shot SYSREF.

SYSREF on Link Resynchronization with 64B66B Linecoding

After initial bring-up, if a link resynchronization is requested by the receiving device, the desired core behavior relative to SYSREF can be controlled using the SYSREF Required control bit in the CTRL_SYSREF Handling register.

When SYSREF Required is set to 0, no SYSREF event is required for the link to resynchronize (the assumption is that LMBC counters continue to free-run and remain valid).

- A receive core will re-acquire multiblock lock and output received data on the next LMBC boundary.
- A transmit core will continue to transmit multiblock data.

When SYSREF Required is set to 1, a SYSREF event is required for the link to reestablish SYNC following a resync request. In this case, the behavior is the same as the initial link bring-up detailed earlier.

This setting is particularly important in systems where a one-shot SYSREF is used, or where SYSREF is periodic but SYSREF Always is set to 0.

Subclass 2 Operation (8B10B Line Coding Only)

The operation of the JESD204C circuit in Subclass 2 mode is similar to a device in Subclass 1 mode. In this case, deterministic latency across the link is achieved using the SYNC~ interface. When the receiver has aligned to the incoming idle characters from the transmitter, it asserts the SYNC signal. The transmitter detects this and waits until the next LMFC crossing before transmitting data or the ILA sequence (depending on the setting of the enable link synchronization control bit). The receiver buffers the data at its input until the next LMFC crossing at its side of the link, before sending the received data to the client.

Subclass 2 operation can be difficult to achieve at medium to high line rates due to the critical need to accurately capture the SYNC signal without violating the setup or hold requirements of the capturing flip flop. It is recommended that careful design validation is performed early to ensure this can be achieved.

Number of Lanes per Link

The maximum number of lanes per link is eight. For interfaces which require more than eight lanes, simply create multiple cores with a maximum of eight lanes each.

For 8B10B *transmit* interfaces, the lane ID and number of lanes per link ILAS parameters for each lane can be independently programmed using the Lane ID registers.

For 8B10B *receive* interfaces, the lane ID and number of lanes per link parameters for each lane can be read from the LID and L fields of the RX_ILA_CFG3 register for each lane.

This programmable Lane ID and number of lanes per link feature for 8B10B cores can also be used to share a single JESD204C core with multiple synchronous converters. For example, eight one lane converters might be connected to a single JESD204C core. If this is an 8B10B transmitter core, the Lane IDs can all be programmed to be lane 0 with 1 lane per link (value = 0). For 64B66B cores, there are no Lane IDs so this is not applicable.

Receive Latency

The latency variation is critical for JESD204C systems. The receive latency through the IP core is fixed, although it can be varied under user control—see Minimum Deterministic Latency Support for details, but the transceiver introduces some variation as the internal receive elastic buffer of the transceiver is included in the data path. The variation in latency is compensated automatically by the core to ensure that the overall end-to-end latency has no variation. Refer to the Latency calculations checklist (AR: 66143).

The 8B10B and 64B66B receive data path latencies are shown in the following tables. They do not include the latency through the JESD204_PHY IP core if using UltraScale/UltraScale+, or the GT Quad/GT Wizard subsystem if using a Versal adaptive SoC (GT_{delay}).

If using UltraScale or UltraScale+, the receive latency through the JESD204_PHY IP core is equal to the latency through the GT (GT_{delay}). There is no additional latency added by the JESD204_PHY IP core. Refer to the relevant GT for the PHY latency values.

Table 58: **8B10B Receive Data Path Latency Through JESD204C**

	GTME5 Transceiver	All Other Transceivers
T_{RXLMFC} (octet clks)	32	16
T_{RXIN}	T_{RXLMFC} + GT_{delay}	T_{RXLMFC} + GT_{delay}
T_{RXOUT}	0	0

Notes:

1. Latency values are given in 8b10b octet clocks. See RX End to End Latency for detailed use.
2. When using Versal adaptive SoC GTME5 transceivers, there is an additional latency in the receive data path because the 8b10b decoding function is performed external to the GT Quad/GT Wizard subsystem.
3. For more information on GT latency values, see GT documentation.

Table 59: **64B66 Receive Data Path Latency Through JESD204C**

	GTME5 Transceiver		All Other Transceivers	
	With FEC	Without FEC	With FEC	Without FEC
T_{RXLEMC} (core clks)	17 +/-1	15 +/-1	8	6
T_{RXIN}	T_{RXLMFC} + GT_{delay}	T_{RXLMFC} + GT_{delay}	T_{RXLMFC} + GT_{delay}	T_{RXLMFC} + GT_{delay}
T_{RXOUT}	0	0	0	0

Notes:

1. Latency values are given in core clocks. See RX End to End Latency for detailed use.
2. When using Versal adaptive SoC GTME5 transceivers, there is an additional latency through the receive data path because the 64b66b decoding function is performed external to the GT Quad/GT Wizard subsystem.
3. When using Versal adaptive SoC GTME5 transceivers, there is an additional +/-1 clock cycle of uncertainty due to the clock crossing from the `rxusrclk` to the `rx_core_clk` within the 64b66b decoder gearbox.
4. For more information on GT latency values, see GT documentation.

RX End to End Latency

Overall latency in a JESD204C system requires consideration of the various sources of fixed and variable latencies across the link.

8B10B Line Coding

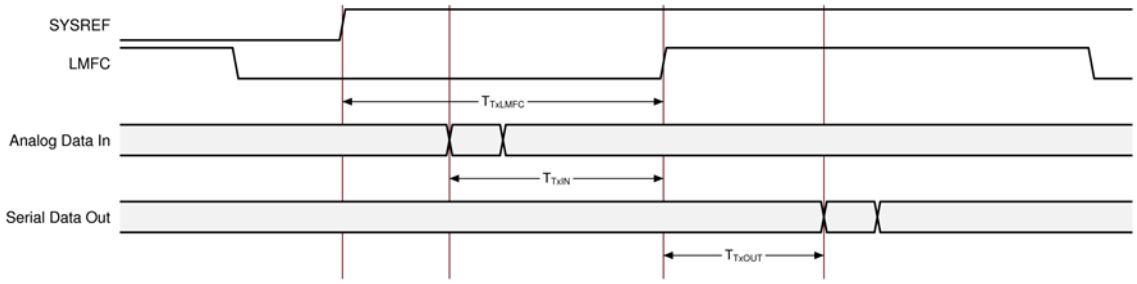
ADC Timing

The key parameters of the ADC required to calculate end to end latency are:

1. The fixed delay from SYSREF to LMFC (T_{TXLMFC})
2. The fixed delay from analog input to LMFC (T_{TXIN})
3. The delay from LMFC to JESD204C serial output (T_{TXOUT})

The delay from SYSREF to LMFC and analog data into LMFC must be fixed for a Subclass 1 device, but the delay from LMFC to JESD204C serial data out can vary because it is compensated for in the receiver during alignment to LMFC. See the following figure.

Figure 18: **ADC Timing**



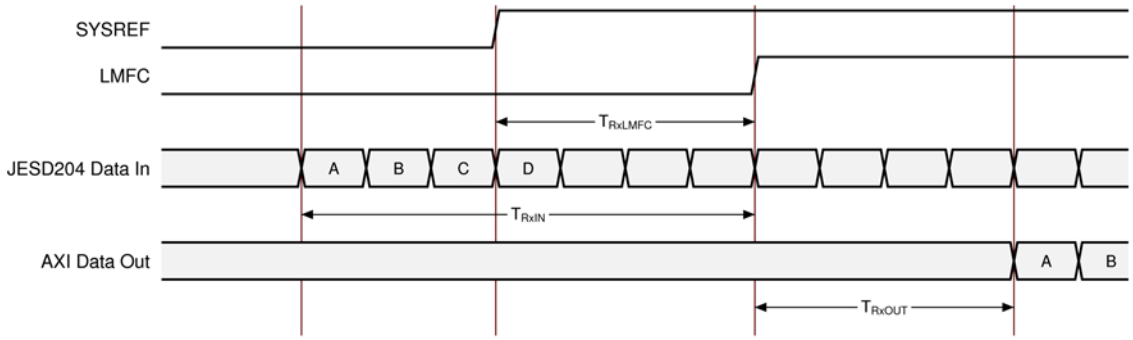
Core Timing

The key parameters of the FPGA receiver required to calculate end to end latency are:

1. The fixed delay from SYSREF to LMFC (T_{RXLMFC})
2. The delay from JESD204C input to LMFC (T_{RXIN})
3. The fixed delay from LMFC to AXI Data output (T_{RXOUT})

The delay from SYSREF to LMFC and from LMFC to AXI Data output must be fixed for a Subclass 1 device, but the delay from JESD204C serial data in to LMFC can vary because the receiver buffer compensates for variations in end to end latency. In the latency calculations, T_{RXOUT} = 0 and all the fixed delays are included in T_{RXLMFC}. See the following figure.

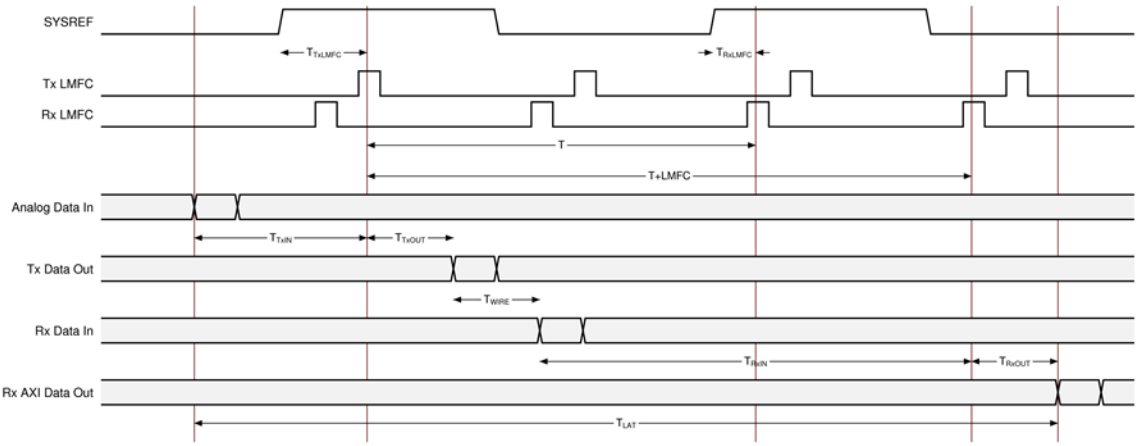
Figure 19: **FPGA Receive Timing**



Calculating End to End Latency

The end to end latency is always fixed and made up of a multiple number of LMFC periods plus the fixed TX and RX delays. To calculate the end to end latency, use the following steps, referring to the following figure.

Figure 20: **End to End Latency**



Step 1: Determine How Many LMFC Periods are Required (N)

The delay between LMFC at TX and LMFC at RX (T) is an integer number (N) of LMFC periods adjusted by the internal delays from SYSREF to LMFC of TX and RX:

$T = N^*LMFC - T_{TXLMFC} + T_{RXLMFC}$

To ensure the overall propagation delay is constant between system restarts, the maximum propagation delay must be less than T plus one LMFC period:

$(T_{TXOUT} + T_{WIRE(max)} + T_{RXIN(max)}) < T + LMFC$

The minimum propagation delay must also be greater than T:

$(T_{TXOUT} + T_{WIRE(min)} + T_{RXIN(min)}) > T$

Substituting $(N^*LMFC - T_{TXLMFC} + T_{RXLMFC})$ for T gives:

$(T_{TXOUT} + T_{WIRE(max)} + T_{RXIN(max)}) < ((N+1)^*LMFC - T_{TXLMFC} + T_{RXLMFC})$

$(T_{TXOUT} + T_{WIRE(min)} + T_{RXIN(min)}) > (N^*LMFC - T_{TXLMFC} + T_{RXLMFC})$

It is possible that no valid value for N can be found if the received data arrives close to an RX LMFC boundary and the variation in the link causes it to fall before or after the boundary after resetting the system. This would be seen as a jump in latency of exactly one LMFC period between system restarts. If no valid value can be found for N, then the `SYSREF` signals can be delayed relative to one another to move the RX LMFC boundary relative to the TX LMFC boundary. For each cycle of delay added to TX SYSREF add 4 to T_{TXLMFC}. Additional delay can be added to the `SYSREF` processing in the JESD204C core to accomplish this; see Table 25: CTRL_SYSREF.

Step 2: Calculate the End to End Latency Using N

The data is received after N LMFC periods and before N+1 LMFC periods, so the minimum deterministic latency is T plus one LMFC plus the fixed input delay at the FPGA and the fixed output delay at the ADC:

$T_{LAT} = (T + LMFC) + T_{TXIN} + T_{RXOUT}$

Substituting $(N^*LMFC - T_{TXLMFC} + T_{RXLMFC})$ for T gives:

$T_{LAT} = ((N^*LMFC - T_{TXLMFC} + T_{RXLMFC}) + LMFC) + T_{TXIN} + T_{RXOUT}$

$= (N+1)^*LMFC - T_{TXLMFC} + T_{RXLMFC} + T_{TXIN} + T_{RXOUT}$

Example

Table 60: **Example**

JESD204C + GT	Assume an ADC with Parameters	Assume
T_{RXLMFC} = 16	T_{TXIN} = 14	LMFC = 32
T_{RXIN} = 32 + 44±4 = 76±4	T_{TXLMFC} = 3	T_{WIRE} = 0
T_{RXOUT} = 0	T_{TXOUT} = 6	

Try N=2

$(T_{TXOUT(max)} + T_{WIRE(max)} + T_{RXIN(max)}) < ((N+1)*LMFC - T_{TXLMFC} + T_{RXLMFC})$

(6 + 0 + 80) < ((N+1)*32 - 3 + 16)

86 < 109 (margin = 23)

$(T_{TXOUT(min)} + T_{WIRE(min)} + T_{RXIN(min)}) > (N*LMFC - T_{TXLMFC} + T_{RXLMFC})$

(6 + 0 + 72) > (N*32 - 3 + 16)

78 > 77 (margin = 1)

N=2 is OK, no need to skew SYSREF

The data is received after two LMFCs (N) and less than three LMFCs (N+1) so the latency is 3*LMFC plus fixed delays:

$T_{LAT} = (N+1)*LMFC - T_{TXLMFC} + T_{RXLMFC} + T_{TXIN} + T_{RXOUT}$

= 96 - 3 + 16 + 14 + 0 = 123

***Note*:** In this case, the margin (how far after the RX LMFC the data arrives) is very small (1-byte period), any variation introduced by the ADC or connection between the ADC and FPGA could cause the data to be received early, causing the data to be received before the LMFC boundary. There is plenty of margin before the third LMFC so, in this case, it is advisable to delay the TX LMFC by one or two to increase the margin and therefore the reliability of the system.

If delaying the TX LMFC is not possible, then the RX LMFC can be delayed by five cycles and N reduced by one as follows:

T_{RXLMFC} becomes 16 + 5*4 = 36

$(T_{TXOUT(max)} + T_{WIRE(max)} + T_{RXIN(max)}) < ((N+1)*LMFC - T_{TXLMFC} + T_{RXLMFC})$

(6 + 0 + 80) < ((N+1)*32 - 3 + 36)

86 < 97 (margin 11 byte periods)

$(T_{TXOUT(min)} + T_{WIRE(min)} + T_{RXIN(min)}) > (N^*LMFC - T_{TXLMFC} + T_{RXLMFC})$

$(6 + 0 + 72) > (N^*32 - 3 + 36)$

78 > 65 (margin 13 byte periods)

The margin numbers at min delay and max delay are closer in value, which means the system is operating with the received multiframe boundary close to the middle of the RX LMFC period; this reduces the chances of the latency changing due to delay variations in the link. The data is received after one LMFCs (N) and less than two LMFCs (N+1) so, the latency is 2*LMFC plus fixed delays:

$T_{LAT} = (N+1)^*LMFC - T_{TXLMFC} + T_{RXLMFC} + T_{TXIN} + T_{RXOUT}$

$= 64 - 3 + 36 + 14 = 111$

***Note*:** GT latency is not accounted for these calculations.

64B66B Line Coding

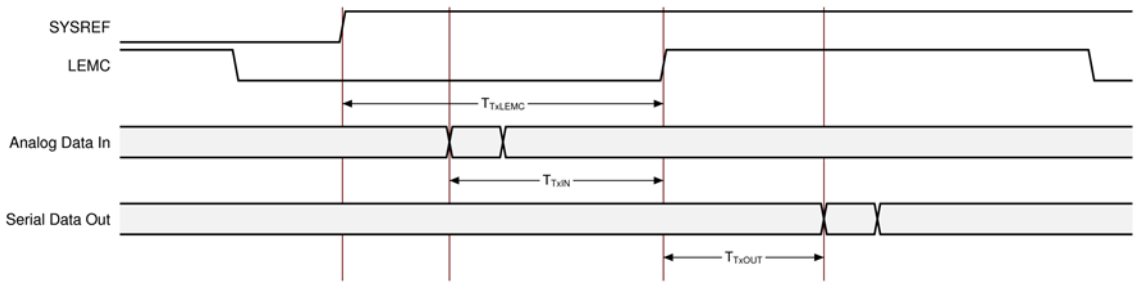
ADC Timing

The key parameters of the ADC required to calculate end to end latency are:

1. The fixed delay from SYSREF to LEMC (T_{TXLEMC})
2. The fixed delay from analogue input to LEMC (T_{TXIN})
3. The delay from LEMC to JESD204C serial output (T_{TXOUT})

The delay from SYSREF to LEMC and analogue data in to LEMC must be fixed for a Subclass 1 device but the delay from LEMC to JESD204C serial data out can vary because it is compensated for in the receiver during alignment to LEMC. See the following figure.

Figure 21: **ADC Timing**



Core Timing

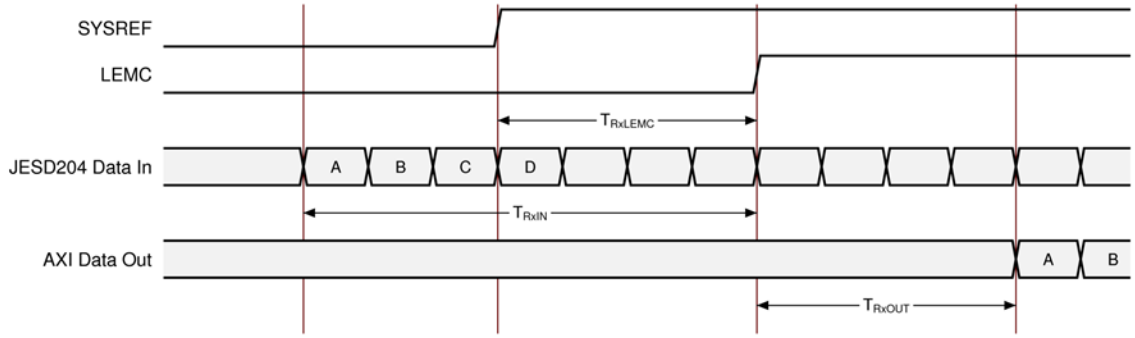
The key parameters of the FPGA receiver required to calculate end to end latency are:

1. The fixed delay from SYSREF to LEMC (T_{RXLEMC})

2. The delay from JESD204C input to LEMC (T_{RXIN})
3. The fixed delay from LEMC to AXI Data output (T_{RXOUT})

The delay from SYSREF to LEMC and from LEMC to AXI Data output must be fixed for a Subclass 1 device, but the delay from JESD204C serial data in to LEMC can vary because the receiver buffer compensates for variations in end to end latency. In the latency calculations, T_{RXOUT} = 0 and all the fixed delays are included in T_{RXLEMC}. See the following figure.

Figure 22: **FPGA Receive Timing**

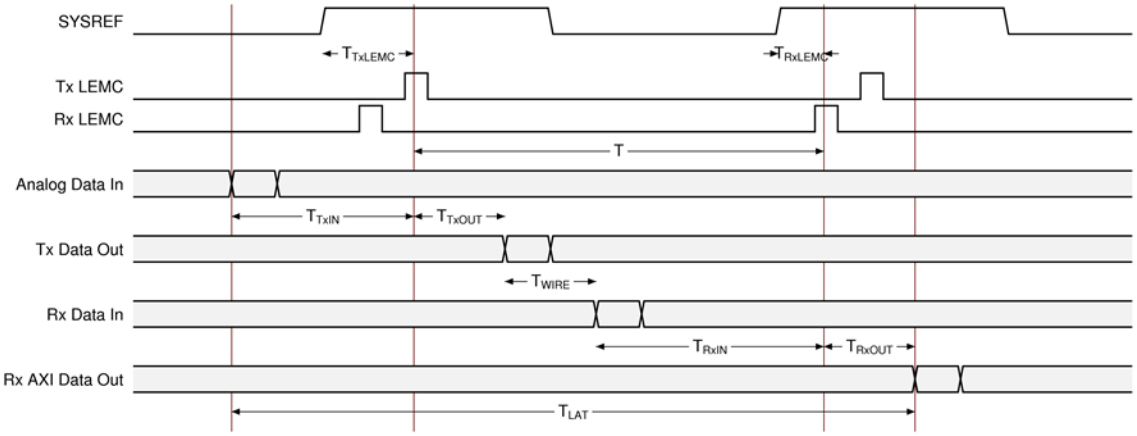


Calculating End to End Latency

The end to end latency is always fixed and made up of one LEMC period plus the fixed TX and RX delays. To calculate the end to end latency, use the following steps, referring to the following figure.

***Note*:** The following is valid for LEMCs of one and two multiblocks. For LEMCs of three or greater, see Receiver Elastic Buffer.

Figure 23: **End to End Latency**



The delay between LEMC at TX and LEMC at RX (T) is the LEMC period adjusted by the internal delays from SYSREF to LEMC of TX and RX:

$T = LEMC - T_{TXLEMC} + T_{RXLEMC}$

To ensure the overall propagation delay is constant between system restarts, the maximum propagation delay must be less than T:

$T_{TXOUT(max)} + T_{WIRE(max)} + T_{RXIN(max)} < T$

Substituting $LEMC - T_{TXLEMC} + T_{RXLEMC}$ for T gives:

$T_{TXOUT(max)} + T_{WIRE(max)} + T_{RXIN(max)} < LEMC - T_{TXLEMC} + T_{RXLEMC}$

Additional delay can be added to the SYSREF processing in the JESD204C core (see Table 25: CTRL_SYSREF). For each cycle of delay added to TX SYSREF, add 1 to T_{TXLEMC}.

The data is received after the fixed delays of TX and RX adjusted by the internal delays from SYSREF to LEMC of TX and RX. This value is also the minimum value for deterministic latency.

$T_{LAT} = T_{TXOUT(max)} + T_{WIRE(max)} + T_{RXIN(max)} - T_{TXLEMC} + T_{RXLEMC}$

Table 61: **Example**

JESD204C + GT Latency	Assume an ADC with Parameters	Assume
$T_{RXLEMC} = 6$	$T_{TXIN} = 4$	LEMC = 64
$T_{RXIN} = 19 \pm 1$	$T_{TXLEMC} = 1$	$T_{WIRE} = 0$
$T_{RXOUT} = 0$	$T_{TXOUT} = 1$	

$T_{TXOUT(max)} + T_{WIRE(max)} + T_{RXIN(max)} < LEMC - T_{TXLEMC} + T_{RXLEMC}$

$2 + 0 + 20 < 64 - 1 + 6$

$22 < 69$

The data is received after the fixed delays of TX and RX adjusted by the internal delays from SYSREF to LEMC of TX and RX.

$T_{LAT} = T_{TXOUT(max)} + T_{WIRE(max)} + T_{RXIN(max)} - T_{TXLEMC} + T_{RXLEMC}$

$T_{LAT} = 2 + 0 + 20 - 1 + 6$

$T_{LAT} = 27$ clock cycles

Note: GT latency is not accounted for these calculations.

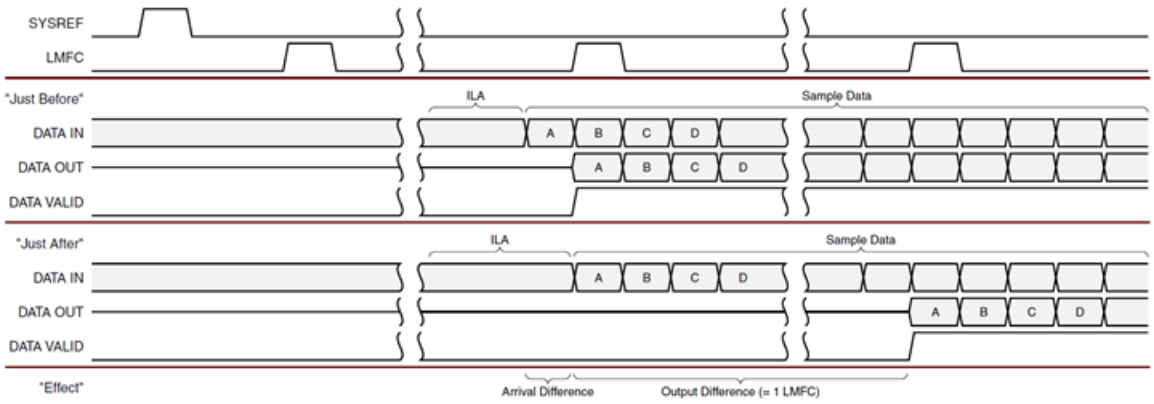
Achieving Repeatable Latency

8B10B Line Coding

In some cases, it is not necessary to know the absolute magnitude of the end-to-end latency, but it is necessary to ensure this delay is robustly repeatable. Because high-speed transceiver-based links, such as those used for the physical layer in JESD204C, do not maintain the same data path latency through a reset or a power cycle, issues can arise in systems where the start data arrival at the receive core is very close to the LMFC boundary. Such systems might be unable to manage the small variability in delay through the link that can occur between restarts.

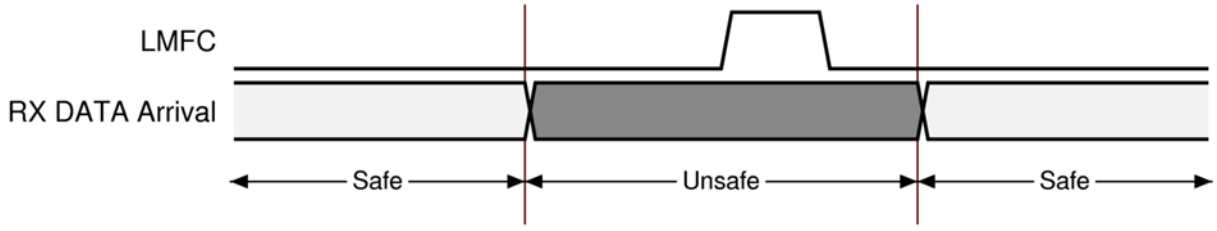
The following figure shows the start of sample data arriving "Just before" and "Just after" the LMFC boundary, illustrating the effect a restart can have. It can be seen that a very small difference in data arrival time at the core can cause a change in the timing of the output data equal to one LMFC period.

Figure 24: **Data arrives ''Just before'' vs ''Just after'' the LMFC**



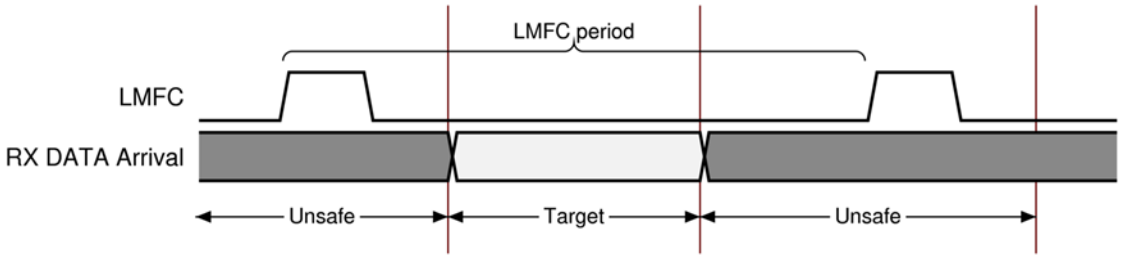
To protect against this problem, a margin must be introduced, around the LMFC boundary, which the start of data must avoid. The following figure shows this.

Figure 25: **LMFC Margin**



The LMFC is a periodic signal requiring a target window (TW) within which to aim. The following figure shows this. If data can start within this TW, link robustness is ensured.

Figure 26: **Target Window**



The TW can generally be set reliably at between two CORE_CLK cycles before and two CORE_CLK cycles after an LMFC boundary.

This assumes that any variation in the output timing of the ADC is less than four octets. If the output variation is greater than four octets, the margin can be increased and the TW decreased accordingly.

The solution is to adjust the internal LMFC relative to the fixed data arrival such that arriving data falls within the target window shown in above figure (i.e., move the window).

This is achieved by starting the link, and using core registers to determine the amount of data which was buffered, then taking steps to adjust the LMFC boundary until acceptable values are attained. This process should only be required to be performed once during development as when the system has been adjusted to ensure the data arrival is within the target window, it should not require further adjustment.

The registers used during this process are:

- **BUFFER FILL LEVEL:** Located in the STAT_RX_BUF_LVL register (see Table 37: STAT_RX_BUF_LVL). The JESD204C 8B10B core contains a readable BUFFER FILL LEVEL register for every JESD204C 8B10B lane. This register indicates how much data was in the lane alignment buffer for each lane at the LMFC boundary when the output data was released.

  ***Note*:** The value in the BUFFER FILL LEVEL register might be different for each lane due to inter-lane skew on the link. The value of interest is the smallest value of all the lanes. This marks the starting point in time when all lanes had valid data. This is the point where the core was actually ready to release data. The value in this register is a count in octets, and there are four octets per CORE CLK cycle.

- **SYSREF DELAY:** Located in the CTRL_SYSREF register (see Table 25: CTRL_SYSREF). This register delays the internal LMFC boundaries relative to SYSREF by an integer number of CORE CLK cycles. Therefore, programming a SYSREF DELAY of 0x1 will cause the LMFC to be delayed by four octets.

Procedure - Achieving Repeatable Latency

1. Calculate the Multi Frame (MF) size. MF= F * K

   (Where F= Frame Size and K= Number of Frames per Multi Frame.)

   The larger the value of MF, the greater the target window will be. It is recommended to pick a value for K that results in a MF of at least 32 octets.

2. Calculate the TW maximum value in octets based on the MF size. This is:

   Max = (MF-0x8)

   Min= 0x8. (0x8 = 2 CORE_CLK cycles)

3. Configure the system and start the link. At this point, all delays can be unknown.
4. Once the link is running, read the JESD204C 8B10B RX core BUFFER FILL LEVEL register for every JESD204C 8B10B lane in use. Select the smallest value read from the BUFFER FILL LEVEL register and call this value BUF_FILL.
5. If BUF_FILL falls between the Min and Max values (calculated at step 2), the data arrival is within the safe TW and no further action is required. If however the BUF_FILL does not fall between the Min and Max values, then the following action should be taken.
   a. Program the SYSREF DELAY register with a delay value calculated as follows:

      If the BUF_FILL value is low then the SYSREF DELAY value programmed should be increased by 1 or 2 as follows:

      If BUF_FILL < Min - 4. Then SYSREF DELAY should be increased by two.

      If BUF_FILL < Min but > Min - 4. Then SYSREF DELAY should be increased by one.

      If the buffer is almost full, then the SYSREF DELAY value programmed should be increased by three or four (i.e., 1 or 2 plus the low margin of 2) as follows:

   b. Reset the JESD204C receive core, and reinitialize the link. This step must be performed before the modified SYSREF_DELAY value will affect the LMFC.
   c. Re-read the BUFFER FILL LEVEL register for every lane to confirm the data arrival is now within the target window.

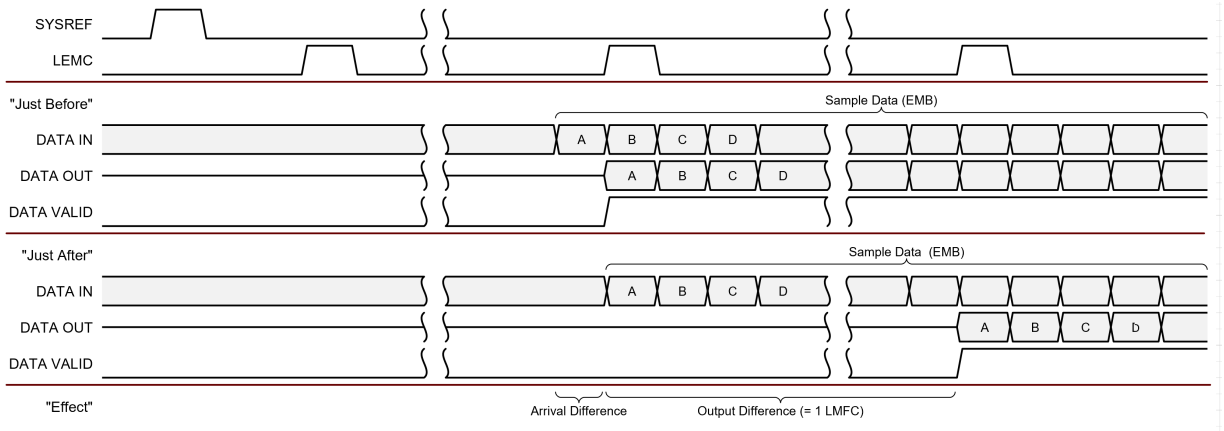
      The calculated SYSREF DELAY value should be stored for future use when configuring this link.

64B66B Line Coding

In some cases, it is not necessary to know the absolute magnitude of the end-to-end latency, but it is necessary to ensure this delay is robustly repeatable. Because high-speed transceiver-based links, such as those used for the physical layer in JESD204C, do not maintain the same data path latency through a reset or a power cycle, issues can arise in systems where the start data arrival at the receive core is very close to the LEMC boundary. Such systems might be unable to manage the small variability in delay through the link that can occur between restarts.

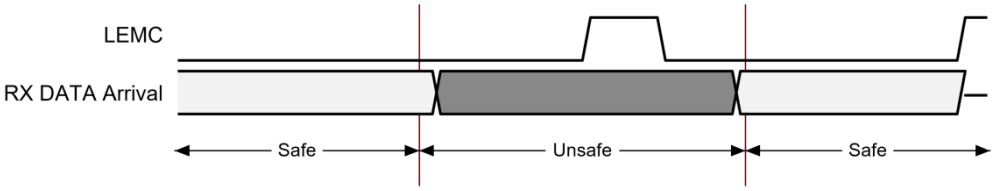
The following figure shows the start of sample data arriving "Just before" and "Just after" the LEMC boundary, illustrating the effect a restart can have. It can be seen that a very small difference in data arrival time at the core can cause a change in the timing of the output data equal to one LEMC period.

Figure 27: **Data arrives ''Just before'' vs ''Just after'' the LEMC boundary**



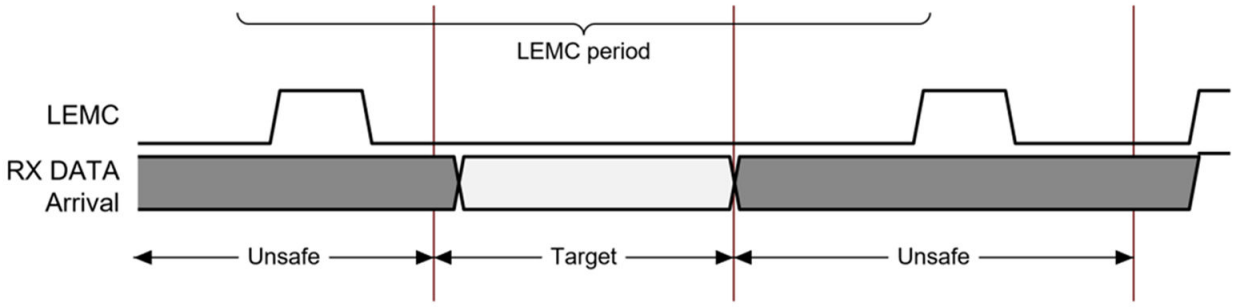
To protect against this problem, a margin must be introduced around the LEMC boundary, which the start of data must avoid. The following figure shows this.

Figure 28: **LEMC Margin**



The LEMC is a periodic signal requiring a TW within which to aim. The following figure shows this. If data can start within this TW, link robustness is ensured.

Figure 29: **Target Window**



The TW can generally be set reliably at between two CORE_CLK cycles before and two CORE_CLK cycles after an LEMC boundary.

This assumes that any variation in the output timing of the ADC is less than one 64-bit word. If the output variation is greater than one word, the margin can be increased, and the TW decreased accordingly.

The solution is to adjust the internal LEMC relative to the fixed data arrival such that arriving data falls within the target window shown in the previous figure (i.e., move the window).

This is achieved by starting the link, and using core registers to determine the amount of data which was buffered, then taking steps to adjust the LEMC boundary until acceptable values are attained. This process should only be required to be performed once during development as when the system has been adjusted to ensure the data arrival is within the target window, it should not require further adjustment.

The registers used during this process are:

- **BUFFER FILL LEVEL:** Located in the STAT_RX_BUF_LVL register (see Table 37: STAT_RX_BUF_LVL): The JESD204C 64B66B core contains a readable BUFFER FILL LEVEL register for every JESD204C 64B66B lane. This register indicates how much data was in the lane alignment buffer for each lane at the LEMC boundary when the output data was released.

  The value in the BUFFER FILL LEVEL register might be different for each lane due to inter-lane skew on the link. There are two values of interest, the smallest and the largest values of all the lanes. The smallest value impacts the beginning of the TW, and the largest impacts the end. The value in this register is a count in 64-bit words and there is one word per CORE CLK cycle.

- **SYSREF DELAY:** Located in the CTRL_SYSREF register (see Table 25: CTRL_SYSREF). This register delays the internal LEMC boundaries relative to SYSREF by an integer number of CORE CLK cycles. Therefore, programming a SYSREF DELAY of 0x1 will cause the LEMC to be delayed by one word.

Procedure - Achieving Repeatable Latency

1. Calculate the TW max and min values in words based on the receive buffer size as follows.

   TW Max = 64-2 = 62. (In the case where EMB=1, TW Max is not applicable)

   TW Min = 0+2 = 2. (two CORE_CLK cycles)

2. Configure the system and start the link. At this point, all delays can be unknown.
3. Once the link is running, read the JESD204C 64B66B RX core BUFFER FILL LEVEL register for every JESD204C 64B66B lane in use.

   Select the smallest value read from the BUFFER FILL LEVEL register and call this value BUF_FILL.

4. If BUF_FILL falls between the Min and Max values (calculated at step 2), the data arrival is within the safe TW and no further action is required. If however the BUF_FILL does not fall between the Min and Max values, then the following action should be taken.
   a. Program the SYSREF DELAY register with a delay value calculated as follows:

      If the BUF_FILL value is low, then the SYSREF DELAY value programmed should be increased by one or two as follows:

      - If BUF_FILL = 0. Then SYSREF DELAY should be increased by two.
      - If BUF_FILL = 1. Then SYSREF DELAY should be increased by one.

      If the buffer is almost full, then the SYSREF DELAY value programmed should be increased by three or four (i.e., 1 or 2 plus the low margin of 2).

   b. Reset the JESD204C receive core and reinitialize the link. This step must be performed before the modified SYSREF_DELAY value will affect the LEMC.
   c. Re-read the BUFFER FILL LEVEL register for every lane to confirm the data arrival is now within the target window.

      When configuring this link, the calculated SYSREF DELAY value should be stored for future use.

Receiver Elastic Buffer

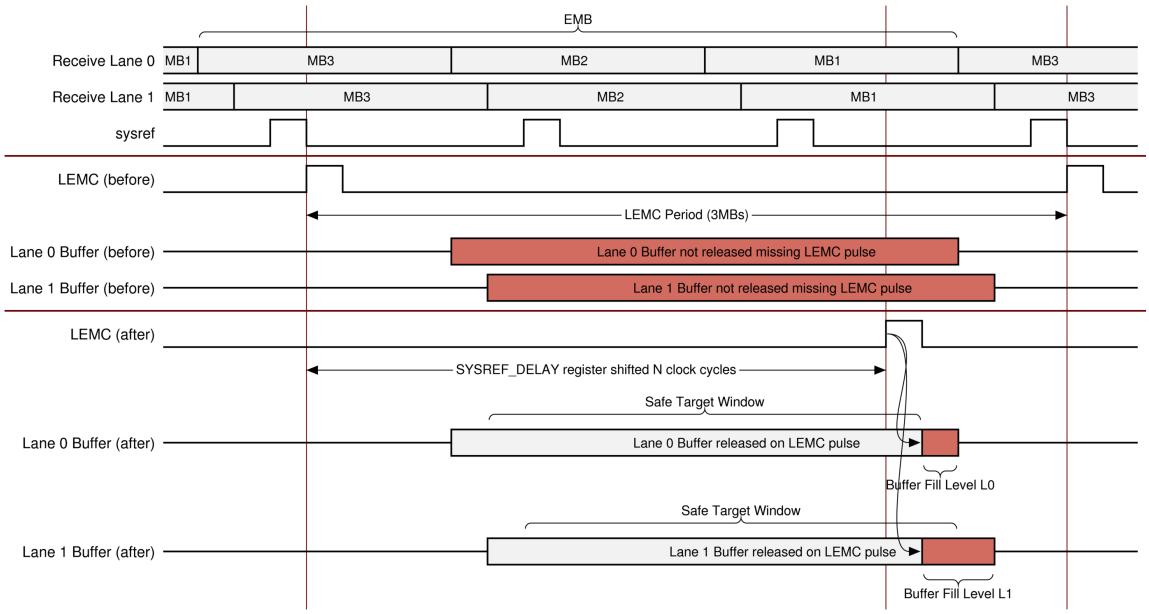
Lane alignment is achieved using 64B66B encoding in the following manner.

Each lane has a receiver buffer, which is two multiblocks deep, 2 x 32 64-bit words.

Each lane's receive buffer starts buffering data when the EOEMB is received. When the last arriving lane has received its EOEMB, all lane receive buffers are released on the next LEMC, so that all lanes are aligned and released at the right time.

For designs where EMB > 2 MB, extra care should be taken to ensure the buffer does not overflow. The following figure shows an example where E = 3, i.e., there are three multiblocks in an EMB. The figure shows before and after an adjustment to the SYSREF_DELAY register has been made. In the before case, the LEMC boundaries straddle the receive buffer, so it is not released and the buffer overflows. In the after case, the LEMC boundary has been shifted by N clock cycles to place it in the safe TW allowing the buffer to be released correctly with no overflow.

Figure 30: **Adjusting the LEMC boundary using SYSREF_DELAY (EMB = 3)**



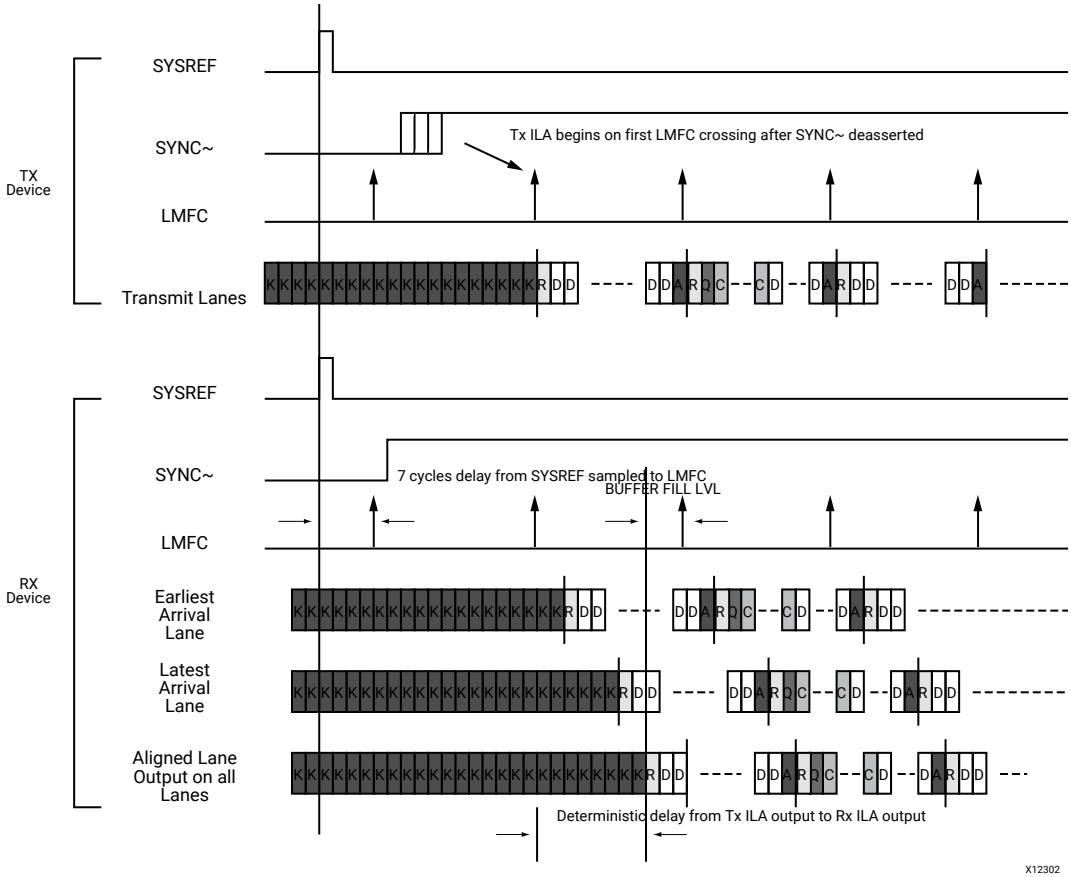
Minimum Deterministic Latency Support

When the JESD204C link is first established in Subclass 1 and Subclass 2 devices, the receiver outputs data as shown in the following figure. Data is output on the LMFC crossing after valid data is detected in all lanes. It is possible to support minimum latency by adjusting the number of octets input on the BUFFER FILL LEVEL register.

An indication of the maximum allowable reduction of the latency is output on the BUFFER FILL LEVEL register. This provides an indication of the difference between the write and read pointers of the receiver elastic buffer in each lane. The number of octets output in each 10-bit value give an indication of the buffer fill level in each lane. The lowest number given can be used to calculate a value that can be programmed to the BUFFER FILL LEVEL register to reduce the overall latency by that number of octets. The maximum value programmed into the BUFFER FILL LEVEL register must take into account the eight octet margin described in Achieving Repeatable Latency, and therefore should not be greater than the lowest BUFFER FILL LEVEL value minus eight.

A reset of the JESD204C receive core and a full link resynchronization cycle must take place before the modified latency setting is acted upon. A minimum latency example is shown in the following figure.

Figure 31: **Minimum Deterministic Latency**



Error Signaling Using the SYNC~ Interface (8B10B only)

In the JESD204C RX core, the SYNC~ signal can be deasserted for two frame periods at the end of each multiframe to indicate an error has occurred which does not require a full reinitialization of the link. This behavior is enabled by the deassertion of the disable_error_reporting control bit.

The SYNC~ signal is a 1-bit output; this enables the correct error output for a frame size of down to two octets. The signal is Low when the receiver has detected an error. An error occurs when an idle or an unexpected control character is received during normal frame transmission.

To give a frame-accurate SYNC~ output when the frame size is one octet, the SYNC~ signal should be asserted for only half a cycle of the device clock. This is not implemented in the core as it would require a dedicated clock (either twice the device clock speed or an inverted device clock). If accurate timing of SYNC~ assertion for error reporting is required, logic must be added externally to generate a half cycle pulse on the external SYNC~ pin if the core SYNC~ signal is asserted for a single cycle and F = 0.

Link Reinitialization (8B10B only)

In the JESD204C RX core, errors that require a full link re-initialization are signaled by deasserting SYNC~ for a period 16 frames. When this occurs, the transmitter should fall back out of data transmission mode and renegotiate the link as described in the previous device subclass sections.

The following error conditions require a full link reinitialization:

- Four consecutive 8B10B decoder errors (disparity or not in table).
- Four consecutive unexpected Kx.y characters (rxcharisk is expected only at the end of the frame).
- Eight consecutive alignment errors (multiframes not aligned to LMFC).

A TX might initiate a re-initialization using the data interface by sending four consecutive Kx.y characters. This causes the error condition above to be triggered in the receiver. K28.5 (0xBC) would normally be used for this.

Transmit Latency

The latency variation is critical for JESD204C systems. The transmit latency through the IP core is fixed but the transceiver introduces some variation as the internal transmit elastic buffer of the transceiver is included in the data path.

The 8B10B and 64B66B transmit data path latencies are shown in the following tables, respectively. They do not include the latency through the JESD204_PHY IP core if using UltraScale/UltraScale+, or the GT Quad/GT Wizard subsystem if using Versal Adaptive SoCs (GT_{delay}).

If using UltraScale or UltraScale+ the transmit latency through the JESD204_PHY IP core is equal to the latency through the GT (GT_{delay}). There is no additional latency added by the JESD204_PHY IP core. Refer to the relevant GT for the PHY latency values.

Table 62: **8B10B Transmit Data Path Latency Through JESD204C**

	GTME5 Transceiver	All Other Transceivers
T_{TXLEMC} (octet clks)	28	24
T_{TXOUT}	T_{TXLEMC} + GT_{delay}	T_{TXLEMC} + GT_{delay}
T_{TXIN}	0	0

Notes:

1. Latency values are given in 8b10b octet clocks. See TX End to End Latency for detailed use.
2. When using Versal adaptive SoC GTME5 transceivers, there is an additional latency through the receive data path because the 8B10B encoding function is performed external to the GT Quad/GT Wizard subsystem.
3. For more information on GT latency, see GT documentation.

Table 63: **64B66B Transmit Data Path Latency Through JESD204C**

	GTME5 Transceiver		All Other Transceivers	
	With FEC	Without FEC	With FEC	Without FEC
T_{TXLEMC} (core clks)	16 +/-1	14 +/-1	8	6
T_{TXOUT}	T_{TXLEMC} + GT_{delay}	T_{TXLEMC} + GT_{delay}	T_{TXLEMC} + GT_{delay}	T_{TXLEMC} + GT_{delay}
T_{TXIN}	0	0	0	0

Notes:

1. Latency values are given in core clock cycles. See TX End to End Latency for detailed use.
2. When using Versal adaptive SoC GTME5 transceivers there is an additional latency through the receive data path because the 64b66b encoding function is performed external to the GT Quad/GT Wizard subsystem.
3. When using Versal adaptive SoC GTME5 transceivers, there is an additional +/-1 clock cycle of uncertainty due to the clock crossing from the `tx_core_clk` to the `txusrclk` within the 64B66B encoder gearbox.
4. For more information on GT latency, see GT documentation.

TX End to End Latency

Overall latency in a JESD204C system requires consideration of the various sources of fixed and variable latencies across the link.

8B10B Line Coding

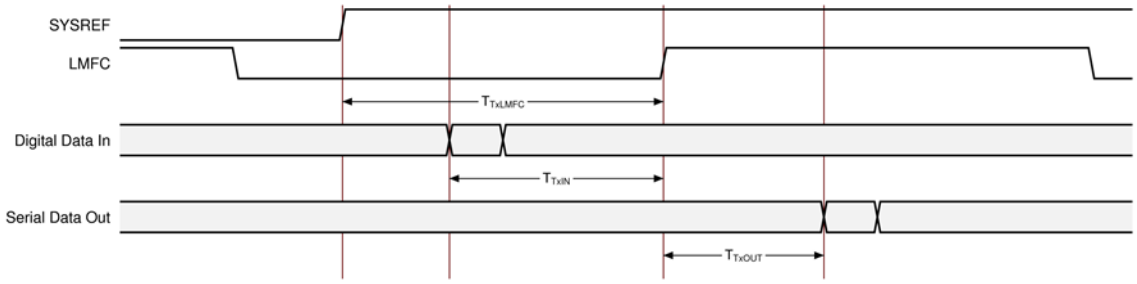
Core Timing

The key parameters of the FPGA transmitter required to calculate end to end latency are:

1. The fixed delay from SYSREF to LMFC (T_{TXLMFC})
2. The fixed delay from AXI input to LMFC (T_{TXIN})
3. The delay from LMFC to JESD204C serial output (T_{TXOUT})

   The delay from SYSREF to LMFC and AXI data in to LMFC must be fixed for a Subclass 1 device but the delay from LMFC to JESD204C serial data out can vary because it is compensated for in the receiver during alignment to LMFC. See the following figure.

Figure 32: **FPGA TX Timing**



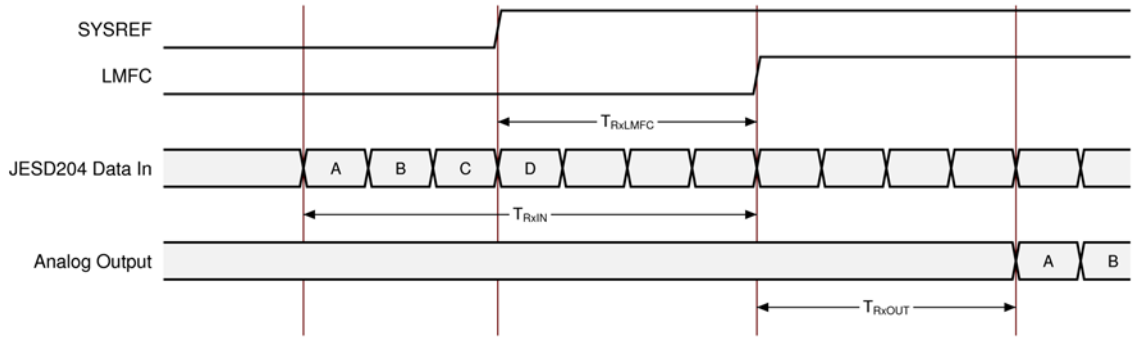
DAC Timing

The key parameters of the DAC receiver required to calculate end to end latency are:

1. The fixed delay from SYSREF to LMFC (T_{RXLMFC})
2. The delay from JESD204C input to LMFC (T_{RXIN})
3. The fixed delay from LMFC to analog output (T_{RXOUT})

   The delay from SYSREF to LMFC and from LMFC to output must be fixed for a Subclass 1device but the delay from JESD204C serial data in to LMFC can vary as the receiver buffer compensates for variations in end to end latency. See the following figure.

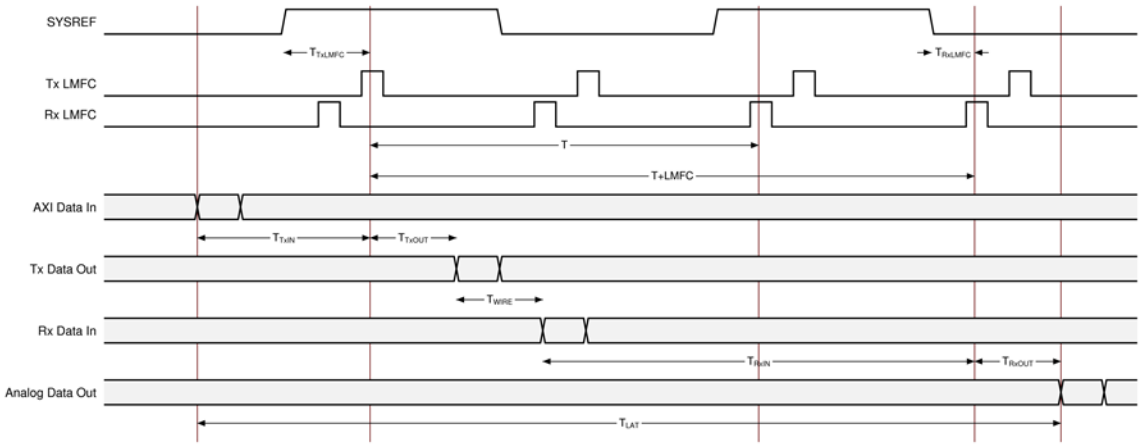
Figure 33: **DAC RX Timing**



Calculating End to End Latency

The end to end latency is always fixed and made up of a multiple number of LMFC periods plus the fixed TX and RX delays. To calculate the end to end latency, use the following steps, referring to the following figure.

Figure 34: **End to End Latency**



Step 1: Determine How Many LMFC Periods are Required (N)

The delay between LMFC at TX and LMFC at RX (T) is an integer number (N) of LMFC periods adjusted by the internal delays from SYSREF to LMFC of TX and RX:

$T = N*LMFC - T_{TXLMFC} + T_{RXLMFC}$

To ensure the overall propagation delay is constant between system restarts, the maximum propagation delay must be less than T plus one LMFC period:

$(T_{WIRE(max)} + T_{RXIN(max)} + T_{TXOUT(max)}) < T + LMFC$

The minimum propagation delay must also be > T:

$(T_{WIRE(min)} + T_{RXIN(min)} + T_{TXOUT(min)}) > T$

Substituting $(N*LMFC - T_{TXLMFC} + T_{RXLMFC})$ for T gives:

$(T_{WIRE(max)} + T_{RXIN(max)} + T_{TXOUT(max)}) < ((N+1)*LMFC - T_{TXLMFC} + T_{RXLMFC})$

$(T_{WIRE(min)} + T_{RXIN(min)} + T_{TXOUT(min)}) > (N*LMFC - T_{TXLMFC} + T_{RXLMFC})$

It is possible that no valid value for N can be found if the received data arrives close to an RX LMFC boundary and the variation in the link causes it to fall just before or just after the boundary after resetting the system. This would be seen as a jump in latency of exactly one LMFC period between system restarts. If no valid value can be found for N then the SYSREF signals can be delayed relative to one another to move the RX LMFC boundary relative to the TX LMFC boundary. For each cycle of delay added to TX SYSREF add four to T_{TXLMFC}. Additional delay can be added to the SYSREF processing in the JESD204C core to accomplish this; see Table 25: CTRL_SYSREF.

Step 2: Calculate the End to End Latency Using N

The data is received after N LMFC periods and before N+1 LMFC periods so the min deterministic latency is T plus one LMFC plus the fixed input delay at the FPGA and the fixed output delay at the DAC:

$T_{LAT} = (T + LMFC) + T_{TXIN} + T_{RXOUT}$

Substituting $(N*LMFC - T_{TXLMFC} + T_{RXLMFC})$ for T gives:

$T_{LAT} = ((N*LMFC - T_{TXLMFC} + T_{RXLMFC}) + LMFC) + T_{RXOUT} + T_{TXIN}$

$= (N+1)*LMFC - T_{TXLMFC} + T_{RXLMFC} + T_{TXIN} + T_{RXOUT}$

Example

Table 64: **Example**

JESD204C + GT Latency	Assume a DAC with Parameters	Assume
T_{TXLMFC} = 24 bytes	T_{RXIN} = 50±2 bytes	T_{WIRE} = 0
T_{TXOUT} = 40±2 bytes	T_{RXLMFC} = 4 bytes	LMFC = 32 bytes
T_{TXIN} = 0	T_{RXOUT} = 20 bytes	

$(T_{TXOUT(max)} + T_{WIRE(max)} + T_{RXIN(max)}) < ((N+1)*LMFC - T_{TXLMFC} + T_{RXLMFC})$

$(T_{TXOUT(min)} + T_{WIRE(min)} + T_{RXIN(min)}) > (N*LMFC - T_{TXLMFC} + T_{RXLMFC})$

$(42 + 0 + 52) < ((N+1)*32 - 24 + 4)$ and $(38 + 0 + 48) > (N*32 - 24 + 4)$

Try N = 3

94 < 108 and 86 > 76

N = 3 is OK, no need to skew SYSREF

The data is received after three LMFCs (N) and less than four LMFCs (N+1), so the latency is 4*LMFC plus fixed delays:

$T_{LAT} = T + T_{TXIN}$

T_{LAT} = (N+1)*LMFC - T_{TXLMFC} + T_{RXLMFC} + T_{TXIN}

= 128 - 24 + 4 + 20 = 128 byte clock periods

Note: GT latency is not accounted for these calculations.

Transmitter Phase Adjustment for Subclass 2

The core provides ports to enable the alignment of the frame clock and LMFC internal to the FPGA to those of the DAC receiver. The required clock and LMFC phase adjustments are communicated to the DAC using the ADJCNT, ADJDIR and PHADJ values in the CTRL_RX_ILA_CFG6 register as detailed in Table 51: CTRL_RX_ILA_CFG6.

In Subclass 2 devices, the DAC receiver deasserts the SYNC~ signal on its internal LMFC boundary. At the FPGA transmitter it is the responsibility of the client logic to detect the deassertion of the SYNC~ signal and to compare its timing to that of the transmitter LMFC. If adjustment of the DAC LMFC phase is required, the client inputs the required adjustment step count to the ADJCNT register. The direction of the phase adjustment is input on ADJDIR and an adjustment request is signaled by the assertion of the PHADJ register.

The values on the phase adjustment ports are embedded in bytes 1 and 2 of the ILA sequence that is sent to the DAC during link initialization. On the reception of the ILA, the DAC adjusts its LMFC phase by the step count value and sends back an error report with the new LMFC phase information. This process can be repeated until the LMFC at the DAC and the FPGA are aligned.

64B66B Line Coding

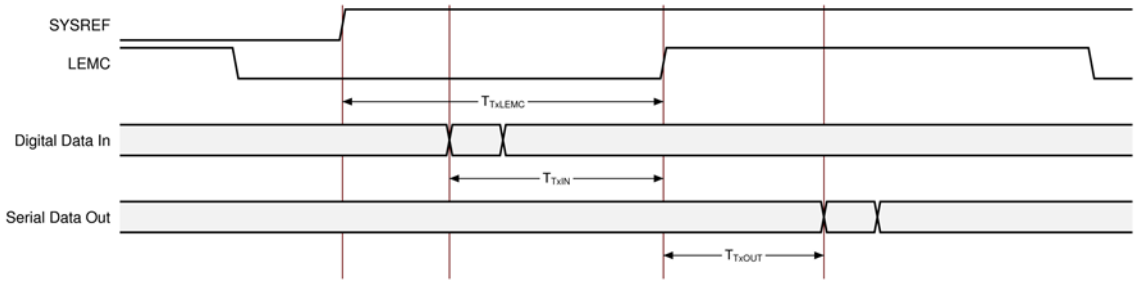
Core Timing

The key parameters of the FPGA transmitter required to calculate end to end latency are:

1. The fixed delay from SYSREF to LEMC (T_{TXLEMC})
2. The fixed delay from AXI input to LEMC (T_{TXIN})
3. The delay from LEMC to JESD204C serial output (T_{TXOUT})

   The delay from SYSREF to LEMC and AXI data into LEMC must be fixed for a Subclass 1 device, but the delay from LEMC to JESD204C serial data out can vary because it is compensated for in the receiver during alignment to LEMC. See the following figure.

Figure 35: **FPGA TX Timing**



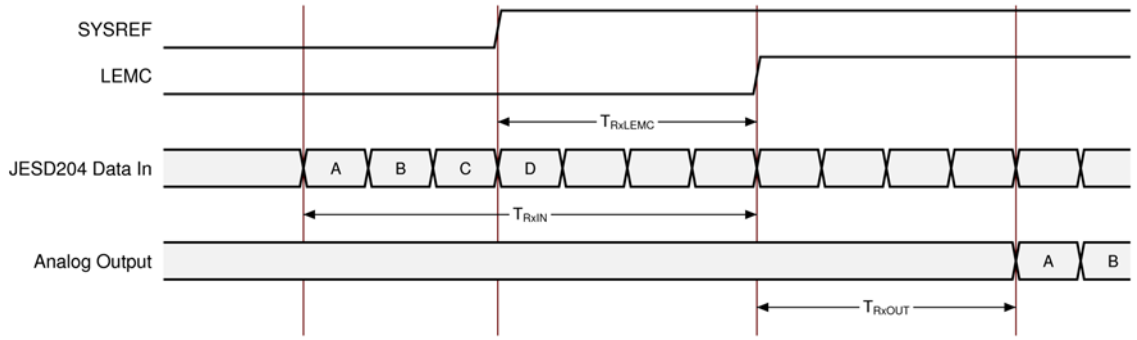
DAC Timing

The key parameters of the DAC receiver required to calculate end to end latency are:

1. The fixed delay from SYSREF to LEMC (T_{RXLEMC})
2. The delay from JESD204C input to LEMC (T_{RXIN})
3. The fixed delay from LEMC to analog output (T_{RXOUT})

   The delay from SYSREF to LEMC and from LEMC to output must be fixed for a Subclass 1 device, but the delay from JESD204C serial data into LEMC can vary because the receiver buffer compensates for variations in end to end latency. See the following figure.

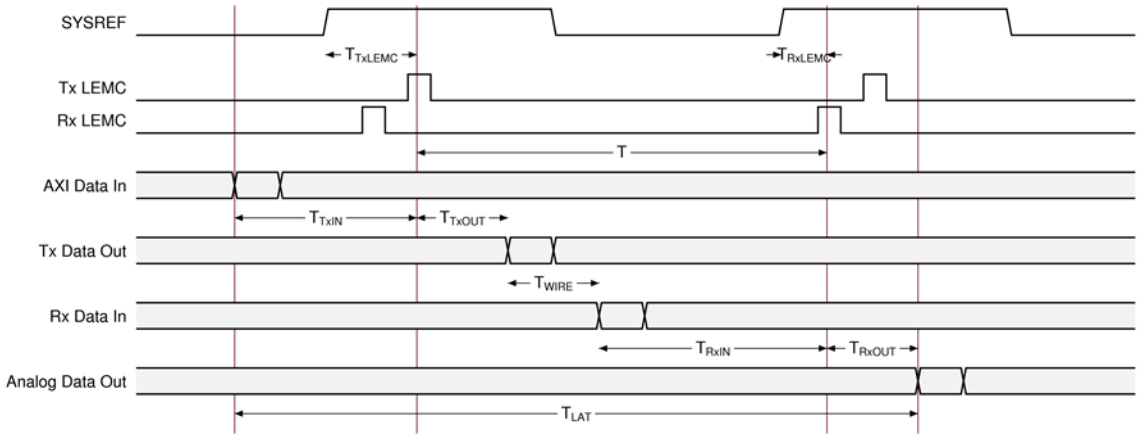
Figure 36: **DAC RX Timing**



Calculating End to End Latency

The end to end latency is always fixed and made up of one LEMC period plus the fixed TX and RX delays. To calculate the end to end latency, use the following steps, referring to the following figure.

Figure 37: **End to End Latency**



The delay between LEMC at TX and LEMC at RX (T) is the LEMC period adjusted by the internal delays from SYSREF to LEMC of TX and RX:

$T = LEMC - T_{TXLEMC} + T_{RXLEMC}$

To ensure the overall propagation delay is constant between system restarts, the maximum propagation delay must be less than T:

$T_{TXOUT(max)} + T_{WIRE(max)} + T_{RXIN(max)} < T$

Substituting $LEMC - T_{TXLEMC} + T_{RXLEMC}$ for T gives:

$T_{TXOUT(max)} + T_{WIRE(max)} + T_{RXIN(max)} < LEMC - T_{TXLEMC} + T_{RXLEMC}$

Additional delay can be added to the SYSREF processing in the JESD204C core (see Table 25: CTRL_SYSREF). For each cycle of delay added to TX SYSREF, add 1 to T_{TXLEMC}.

The data is received after the fixed delays of TX and RX adjusted by the internal delays from SYSREF to LEMC of TX and RX. This value is also the minimum value for deterministic latency.

$T_{LAT} = T_{TXOUT(max)} + T_{WIRE(max)} + T_{RXIN(max)} - T_{TXLEMC} + T_{RXLEMC}$

Table 65: **Example**

JESD204C + GT Latency	Assume a DAC with Parameters	Assume
T_{TXLEMC} = 6 words	T_{RXIN} = 12±1 words	T_{WIRE} = 0
T_{TXOUT} = 14±1 words	T_{RXLEMC} = 1 word	LEMC = 64 words
T_{TXIN} = 0	T_{RXOUT} = 5 words	

$T_{TXOUT(max)} + T_{WIRE(max)} + T_{RXIN(max)} < LEMC - T_{TXLEMC} + T_{RXLEMC}$

$15 + 0 + 13 < 64 - 6 + 1$

28 < 59

The data is received after the fixed delays of TX and RX adjusted by the internal delays from SYSREF to LEMC of TX and RX.

$T_{LAT} = T_{TXOUT(max)} + T_{WIRE(max)} + T_{RXIN(max)} - T_{TXLEMC} + T_{RXLEMC}$

$T_{LAT} = 15 + 0 + 13 - 6 + 1$

$T_{LAT} = 23$ clock cycles

***Note*:** GT latency is not accounted for these calculations.

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 AMD Vivado™ design flows and the IP integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
- *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 AMD tools to customize and generate the core in the AMD Vivado™ Design Suite.

For AMD Versal™ Adaptive SoCs, the core can be used in Vivado IP integrator or the Vivado Integrated Design Environment flow (RTL flow). However, the choice of GT Wizard IP in the JESD204C GUI defines the flow to use for a core.

When configuring the JESD204C core with the Legacy GT Wizard the IP integrator is the only supported flow using Block Automation to generate and configure Versal Adaptive SoC Transceiver cores.

When configuring the JESD204C core with the GT Wizard subsystem both the IP integrator and the Vivado Integrated Design Environment flow (RTL flow) can be used.

For AMD UltraScale+™/AMD UltraScale™ devices, you can either select the IP from the Vivado IP catalog or place the JESD204C IP core on the IP integrator canvas depending on the Vivado flow you choose to use. If you are customizing and generating the core in the IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) for detailed information.

IP integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values do change, see the description of the parameter in this chapter. To view the parameter value, you can run the `validate_bd_design` command in the Tcl Console.

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

1. Select the IP from the Vivado IP catalog or place the JESD204C IP core on the IP integrator canvas depending on the Vivado flow you select.
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 JESD204C configuration screens in the Vivado IDE. The layout depicted here might vary from the current version.

Configuration Tab

Figure 38: **Configuration Tab for Versal Adaptive SoCs**

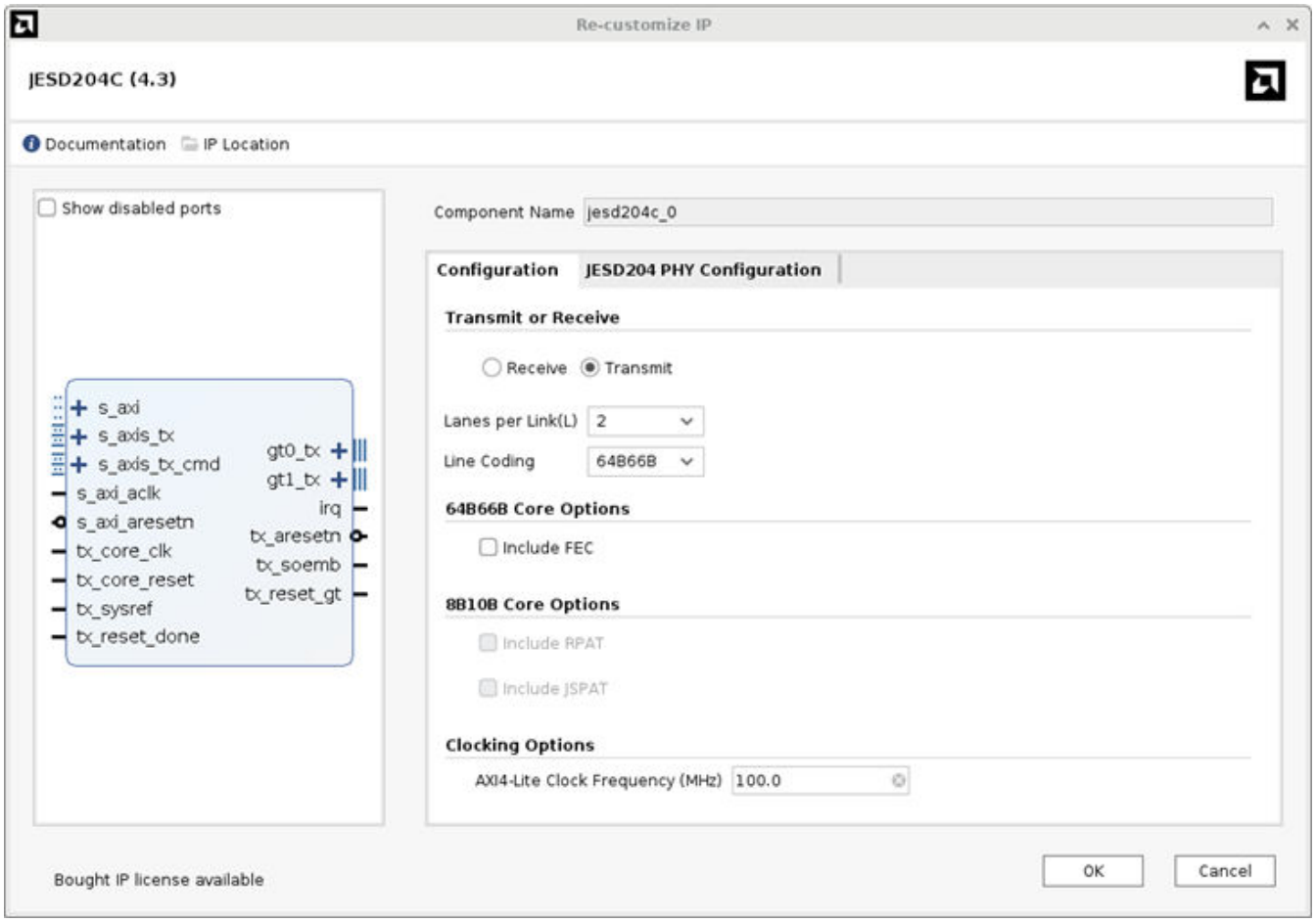
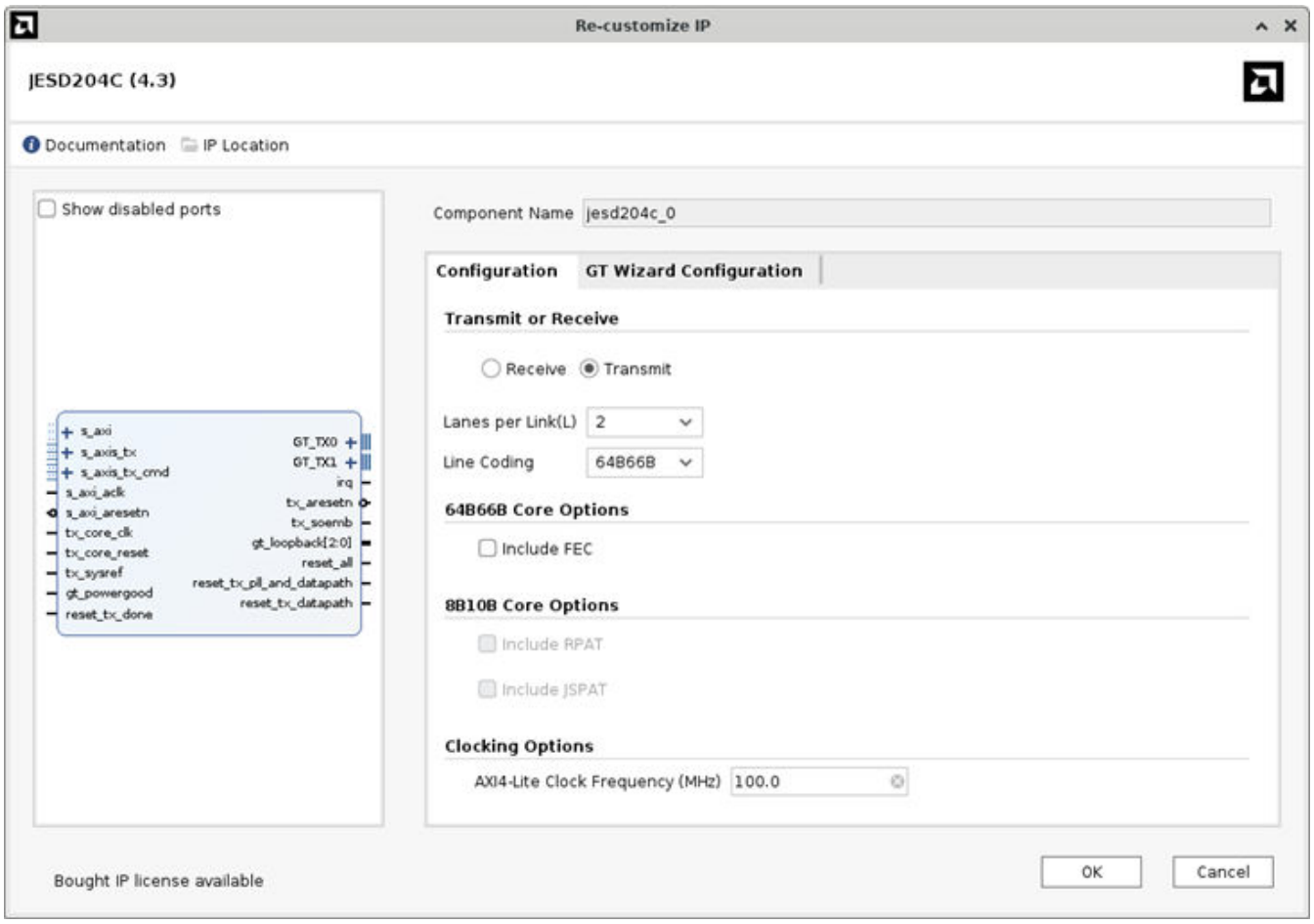


Figure 39: **Configuration Tab for UltraScale+/UltraScale Devices**

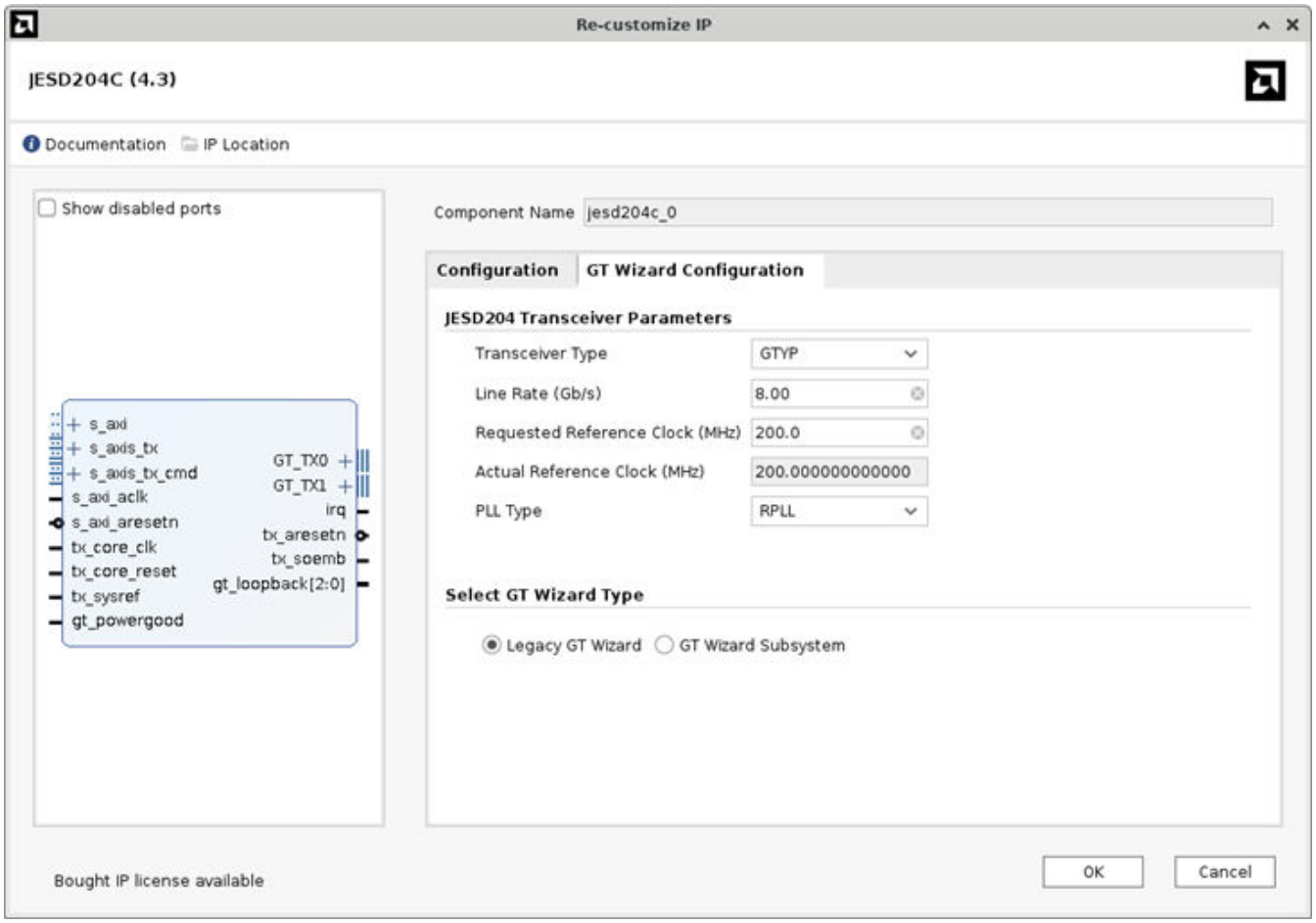


- **Component Name:** The component name is used as the base name of the output files generated for the core. Names must begin with a letter and must be composed from these characters: a through z, 0 through 9, and "_" (underscore).
- **Transmit or Receive:** The core can be configured as a transmitter, for connection to DAC devices, or receiver, for connection to ADC devices.
- **Lanes per Link (L):** The core supports one to eight lanes. For interfaces requiring more than eight lanes, multiple cores must be used.
- **Line Coding:** The core supports 64B66B and 8B10B linecoding modes.
- **Include FEC:** (64B66B only.) Check this option to include the FEC Encoder (TX) or Decoder (RX) in the core. Including the FEC core will increase the resources required by the core.
- **AXI4-Lite Clock Frequency:** The frequency of the clock connected to the AXI4-Lite Management Interface.

- **Include RPAT:** (8B10B Transmitter only.) Check this option to include the logic required to generate a Modified Random test pattern. This option will increase the resources required by the core.
- **Include JSPAT:** (8B10B Transmitter only.) Check this option to include the logic required to generate a Scrambled Jitter test pattern. This option will increase the resources required by the core.

GT Wizard Configuration Tab for Versal Adaptive SoCs

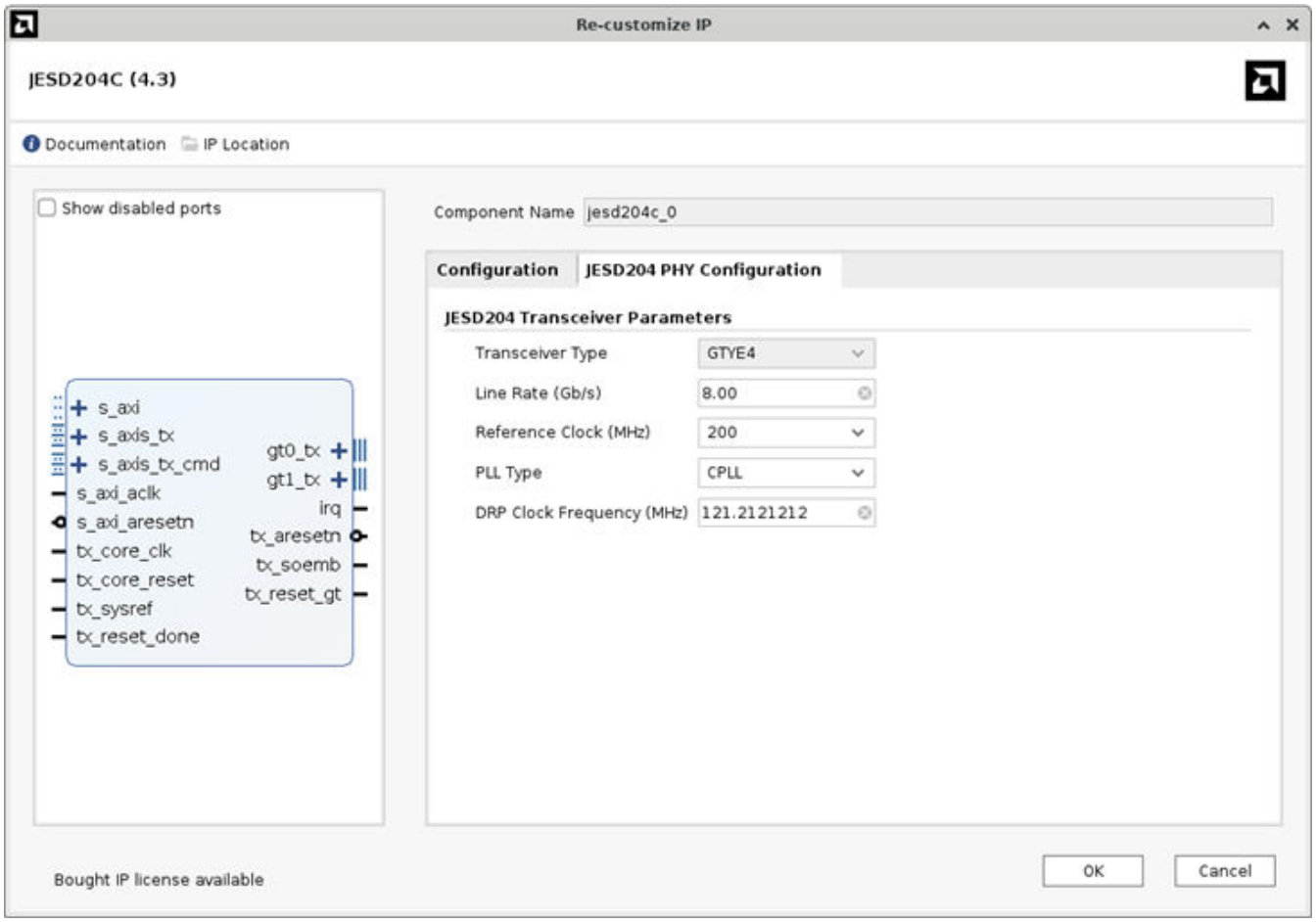
Figure 40: **GT Wizard Configuration Tab**



- **Transceiver Parameters:** For any selected Line Rate and PLL Type, valid Reference Clock frequencies can be calculated by first entering the requested reference clock frequency.

JESD204 PHY Configuration Tab for UltraScale+/ UltraScale Devices

Figure 41: **JESD204 PHY Tab**



- **Transceiver Parameters:** For any selected Line Rate and PLL Type, valid Reference Clock frequencies can be selected from a drop-down list. A free-running DRP clock must be supplied, and the frequency (within the displayed valid range) must be entered in the DRP Clock Frequency box.

User Parameters

The following table shows the relationship between the fields in the AMD Vivado™ IDE and the user parameters (which can be viewed in the Tcl Console).

Table 66: **User Parameters**

Vivado IDE Parameter/Value[1]	User Parameter/Value[1]	Default Value
Transmit or Receive	C_NODE_IS_TRANSMIT	1 (= Transmit)
Lanes per Link	C_LANES	2
AXI4-Lite Clock Frequency	AXICLK_FREQ	100.00
Transceiver Parameters		
Transceiver Type[3]	Transceiver	GTYE3
Line Rate[2]	GT_Line_Rate	8.0
Ref Clock Frequency[2, 4]	GT_REFCLK_FREQ	200.0
DRP Clock Frequency[4]	DRPCLK_FREQ	121.2121212
Requested Reference Clock[3]	GT_REFCLK_FREQ_REQUEST	200.0
Actual Reference Clock[3]	GT_REFCLK_FREQ_ACTUAL	200.0
PLL Type	C_PLL_SELECTION	0 (=CPLL)[4] 3 (=RPLL)[3]
Line Coding	C_ENCODING	1 (1 = 64B66B. 0 = 8B10B)
Select GT Wizard Type[3]	C_GT_WIZARD_TYPE	0 = Legacy_GT_Wizard
Include FEC	C_USE_FEC	0 (= not included)
Include RPAT	C_USE_RPAT	0 (= not included)
Include JSPAT	C_USE_JSPAT	0 (= not included)

Notes:

1. Parameter values are listed in the table where the Vivado IDE parameter value differs from the user parameter value. Such values are shown in this table as indented below the associated parameter.
2. Varies depending on device.
3. For Versal adaptive SoCs only.
4. For UltraScale+/UltraScale devices only.

Output Generation

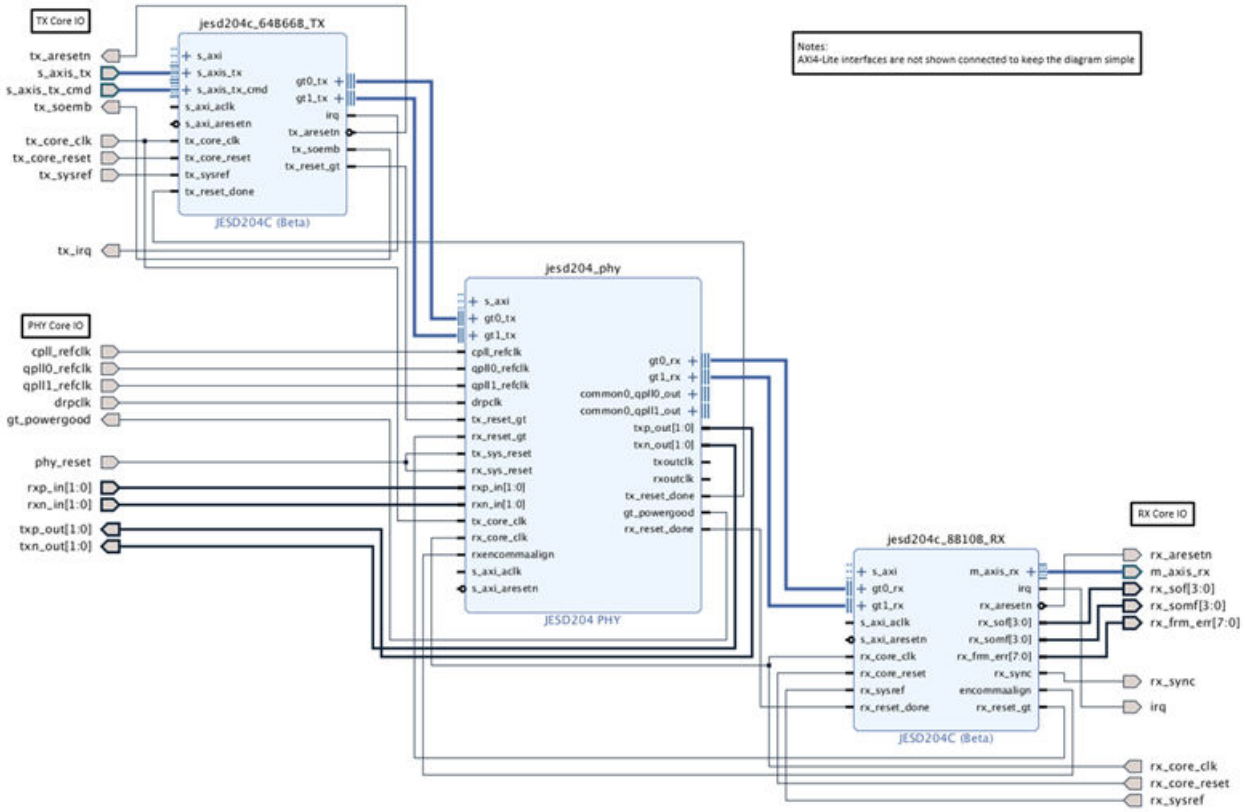
For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896).

Transceiver Sharing for UltraScale+/UltraScale Devices

The JESD204_PHY core (see *JESD204 PHY LogiCORE IP Product Guide* (PG198)) provides a simple way to share transceivers between JESD204C cores. Any number of JESD204_PHY cores can be connected to any number of JESD204C cores to cater for any combination of ADCs and DACs using different line rates, lane counts, linecoding, and versions of the JESD204 standard.

An example of a two lane 64B66B TX and two lane 8B10B RX sharing a JESD204 PHY is shown in the following figure. The transmitter and the receiver are configured for different line rates. Separate `refclk` inputs are provided for each PLL and separate core clocks are provided for TX and RX to support subclass 1 .

Figure 42: **Transceiver Sharing for UltraScale+/UltraScale Devices**



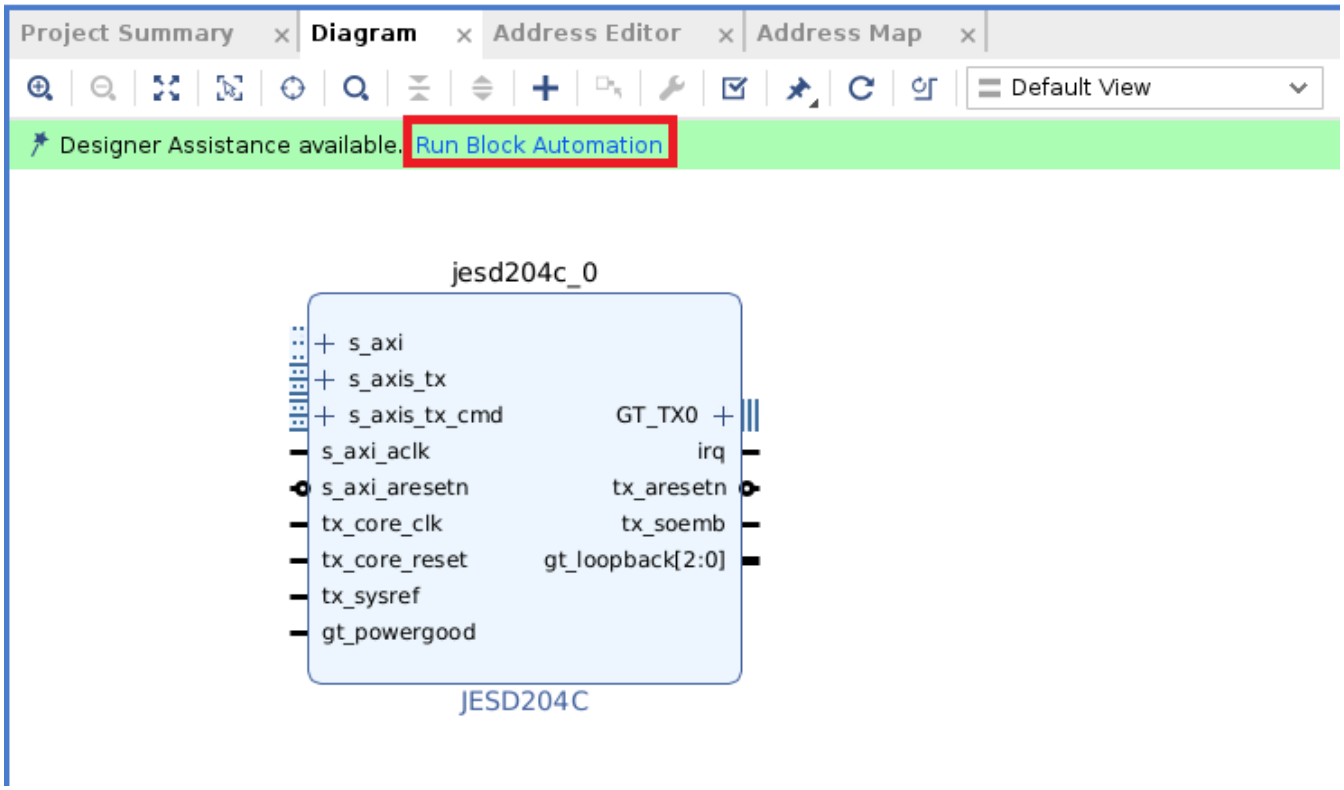
Block Automation (For Versal Adaptive SoC Only)

Connecting to Transceivers Using Block Automation for Versal Adaptive SoCs

Note: Block Automation is available only when you select Legacy GT Wizard in the JESD204C GUI.

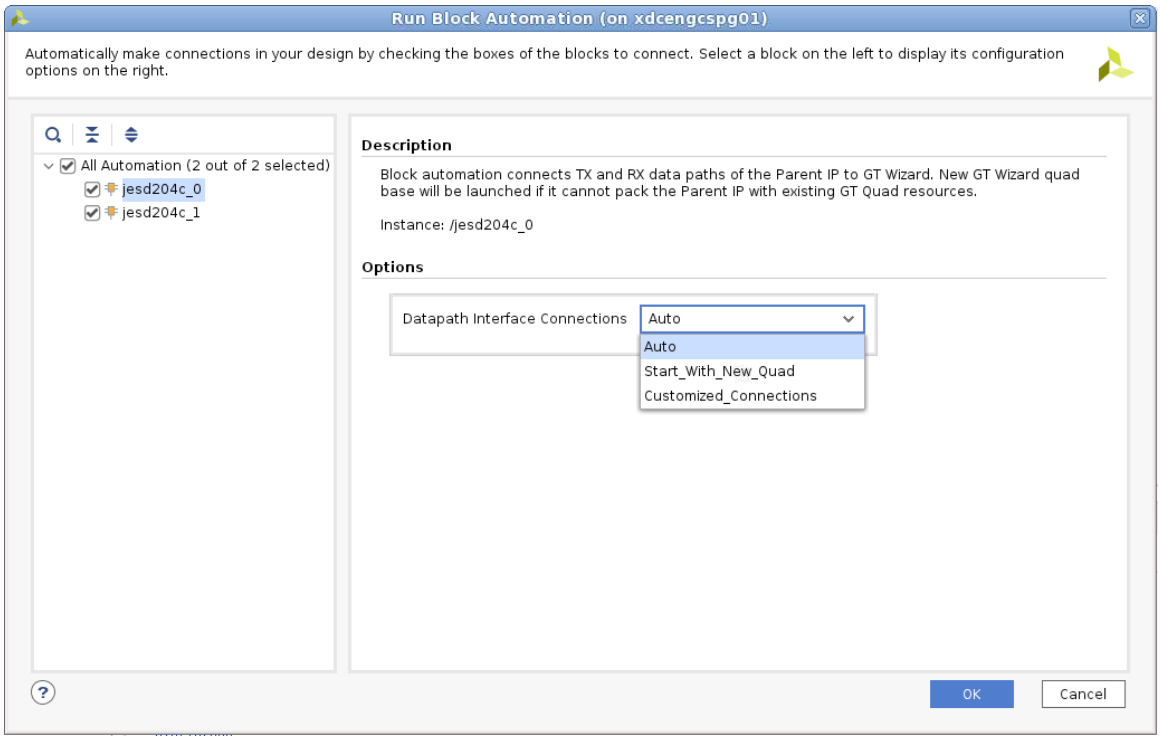
Block automation is the methodology in IP integrator which is used to connect the JESD204C core to a Legacy GT Wizard Transceiver. This is the only supported flow to correctly generate and configure the Legacy Transceiver. To use the new GT Wizard subsystem IP, see GT Wizard Subsystem (For Versal Adaptive SoC only). Block automation is also used to enable transceiver sharing with other JESD cores (TX and RX) and other IPs. The following figure shows the Run Block Automation link in the IP integrator window.

Figure 43: **Block Automation Link for Versal Adaptive SoCs**



The following figure shows the Block Automation dialog box. In this case, two JESD204C IP cores have been placed onto the IP integrator canvas. The dialog box gives you the option to select one or both cores for the application of Block Automation. There is also the option of Auto, Start_With_New_Quad, or Customized Connections.

Figure 44: **Block Automation Dialog Box**



If Auto is selected, Block Automation sequentially places the selected cores in the first available suitable GT Quad location. If no existing Versal Adaptive SoC Transceiver GT Quad is available or suitable, then a new GT Quad is created. Block Automation attempts to pack TX and RX cores (and other IP) into as few GT resources as possible. TX and RX cores share PLL resources when they have common settings (line rate, ref clock, and PLL) or use different resources when they do not. This is shown in Example 2.

If Start_With_New_Quad is selected, Block Automation sequentially places the selected cores in new Versal Adaptive SoC Transceiver GT Quad IPs.

If Customized Connections is selected, Block Automation allows you to choose the specific GT locations to use for each IP core lane. This option does not add new GT Quads to the IP integrator canvas, so it requires the GT Quads to be added to the IP integrator canvas prior to running Block Automation. This option gives the maximum flexibility around the wiring of JESD lanes to GT quad locations.

Block Automation Output

The following examples show what results from running Block Automation.

Versal Adaptive SoC Transceiver GT Quad will be created with the correct settings required for the selected JESD204C IP cores.

- JESD204C IP cores will be connected to the Versal Adaptive SoC Transceiver GT Quad.

- The Versal Adaptive SoC Transceiver Register Configuration Interface is automatically configured as AXI in Block Automation. To change this to an APB3 interface, run the following Tcl command in the Tcl Console.

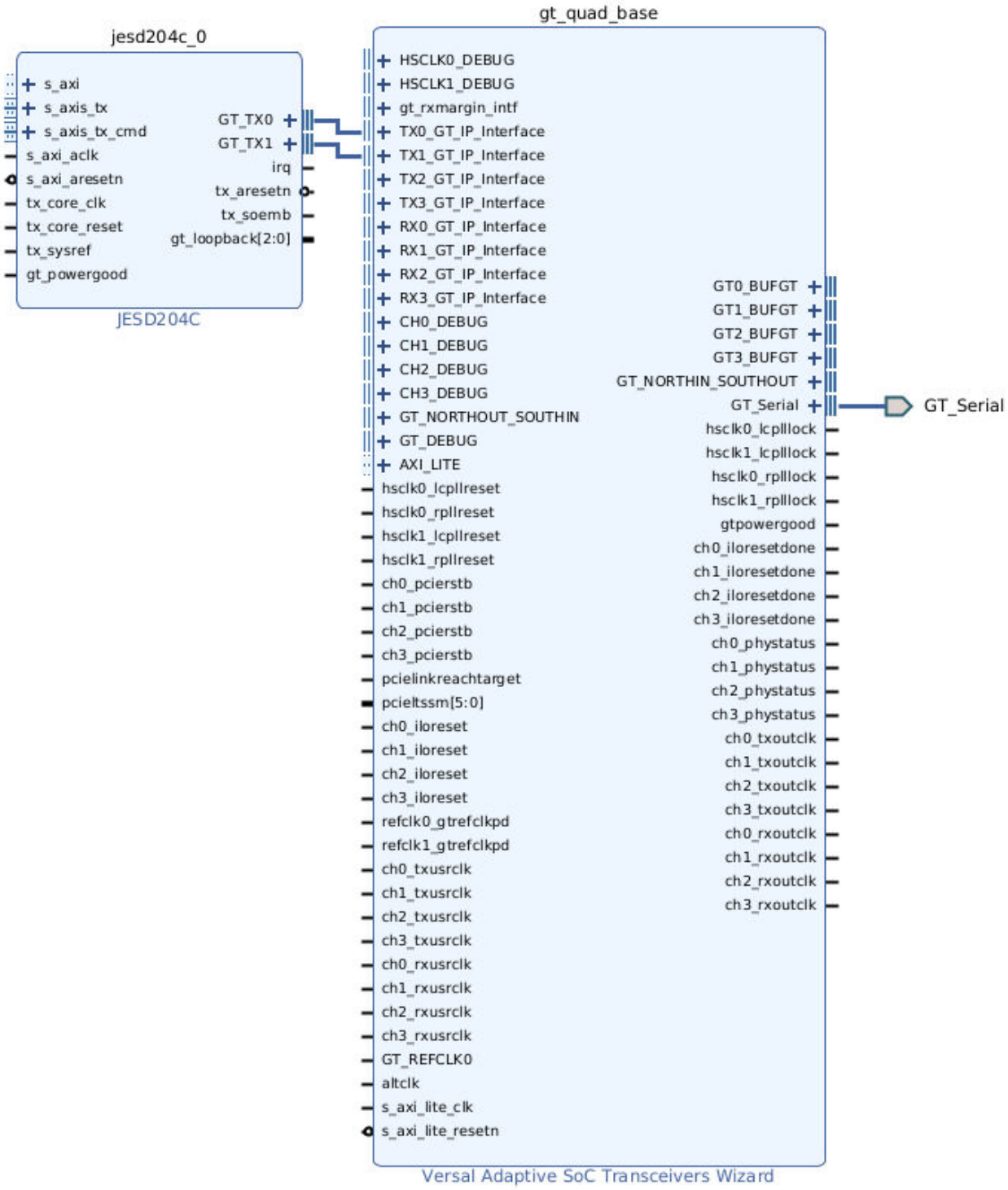
```
set_property -dict [list CONFIG.REG_CONF_INTF {APB3_INTF} ] [get_bd_cells
gt_quad_base_0]
```

The remaining unconnected ports such as `ref_clock`, `core_clk`, resets and `sysref`, etc. then need to be connected up manually. For an example of how to connect up the remaining ports, generate the IP example design (see Chapter 6: Example Design) in Vivado and use this as a reference.

Example 1

The following figure shows the result of running Block Automation on a single JESD204C core with default settings.

Figure 45: **Block Automation on a Single JESD204C Core with Default Settings**

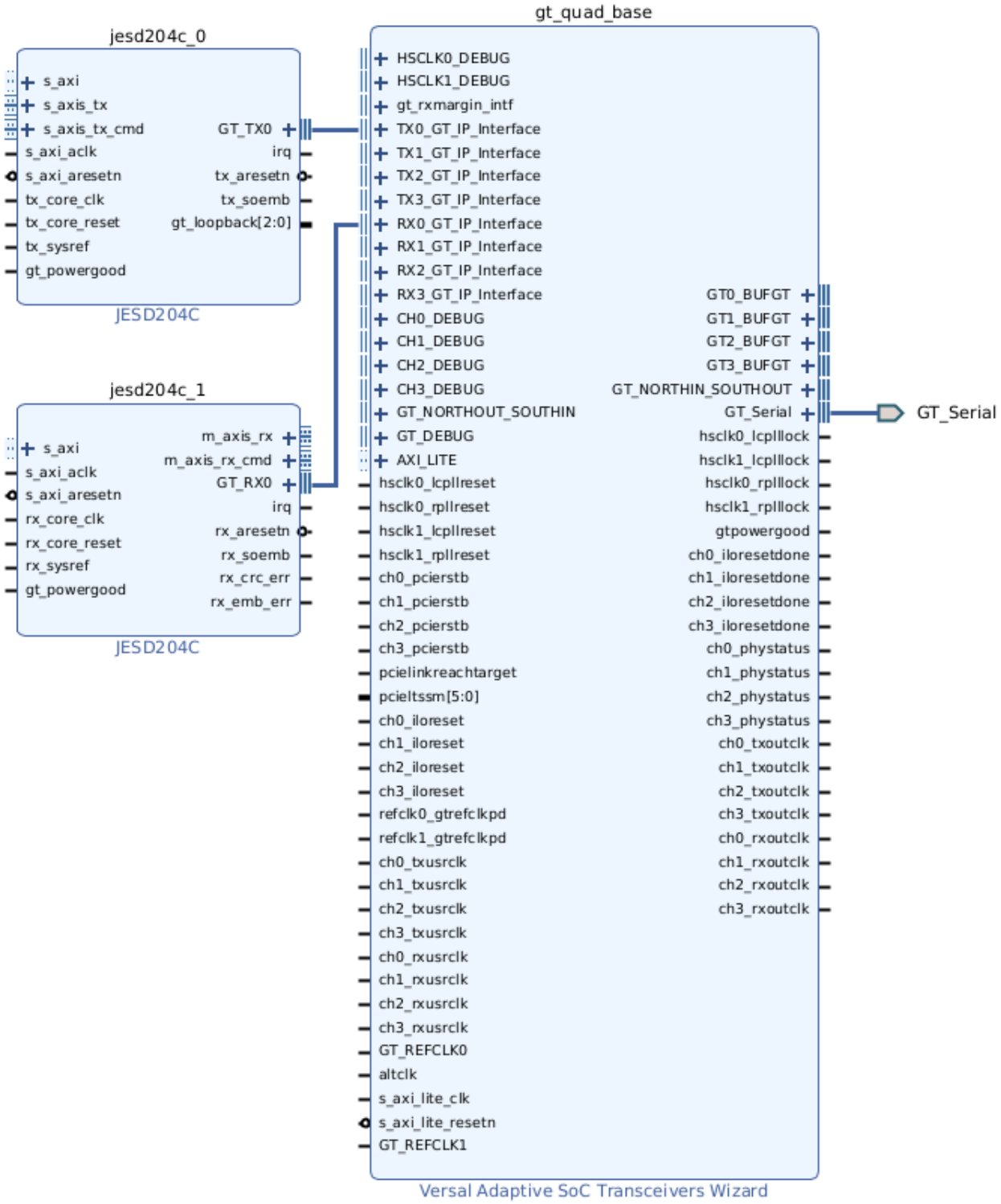


Example 2

The following figure shows the result of running Block Automation on two JESD204C cores, one TX and one RX. The JESD204C cores are running at different line rates and therefore have been configured so that one uses the RPLL and the other the LCPLL.

***Note*:** To see the case where both TX and RX cores sharing a Transceiver are running at the same line rate, generate the IP example design (see Chapter 6: Example Design) in the Vivado IDE.

Figure 46: **Block Automation on a TX JESD204C Core and RX JESD204C Core Running at Different Line Rates**



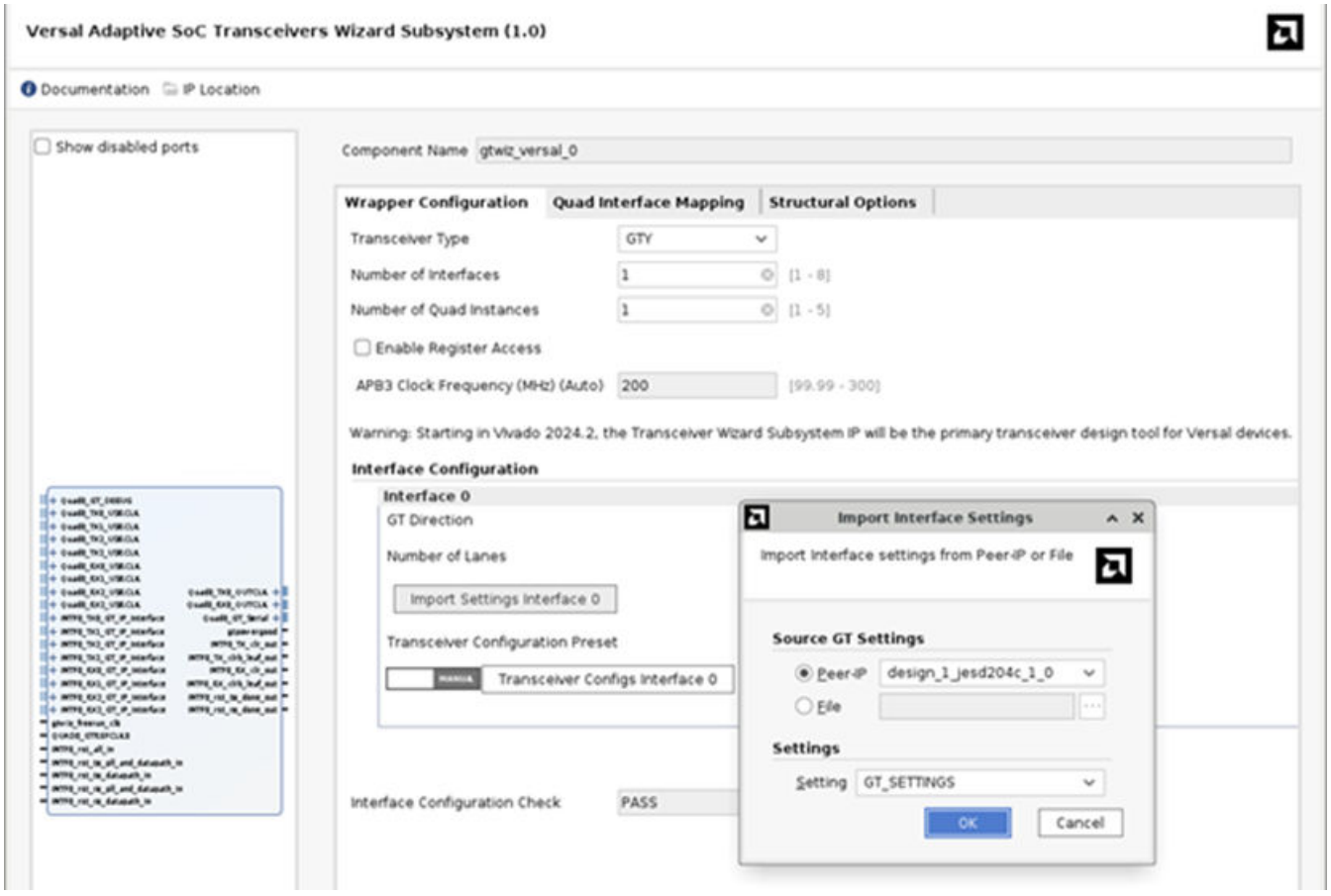
GT Wizard Subsystem (For Versal Adaptive SoC only)

The GT Wizard subsystem is a new Versal Adaptive SoC GT Wizard IP. It supports both the IP integrator flow and the Vivado IDE (RTL) flow.

IP integrator designs that use the GT Wizard subsystem do not support Block Automation, in such cases, the JESD204C IP and the GT Wizard subsystem IP must be connected manually. The JESD204C IP GT interface settings values are automatically populated to the GT Wizard subsystem when you click the **Validate Design** button in JESD204C GUI.

For Vivado IDE (RTL) flow designs, you must first generate the JESD204C IP and the GT Wizard subsystem IP from the IP catalog. Thereafter, you can configure the JESD204C IP and the GT Wizard subsystem IP, including the number of interfaces, number of quads, channels to be used by each interface, and other options related to optional ports. The JESD204C IP GT interface settings can then be automatically populated to the GT Wizard subsystem IP using the **Import Interface Settings** button in the GT Wizard subsystem GUI as shown in the following figure.

Figure 47: **Import GT Interface Settings**



The JESD204C IP example design supports both the Legacy GT Wizard and the new GT Wizard subsystem in the IP integrator flow.

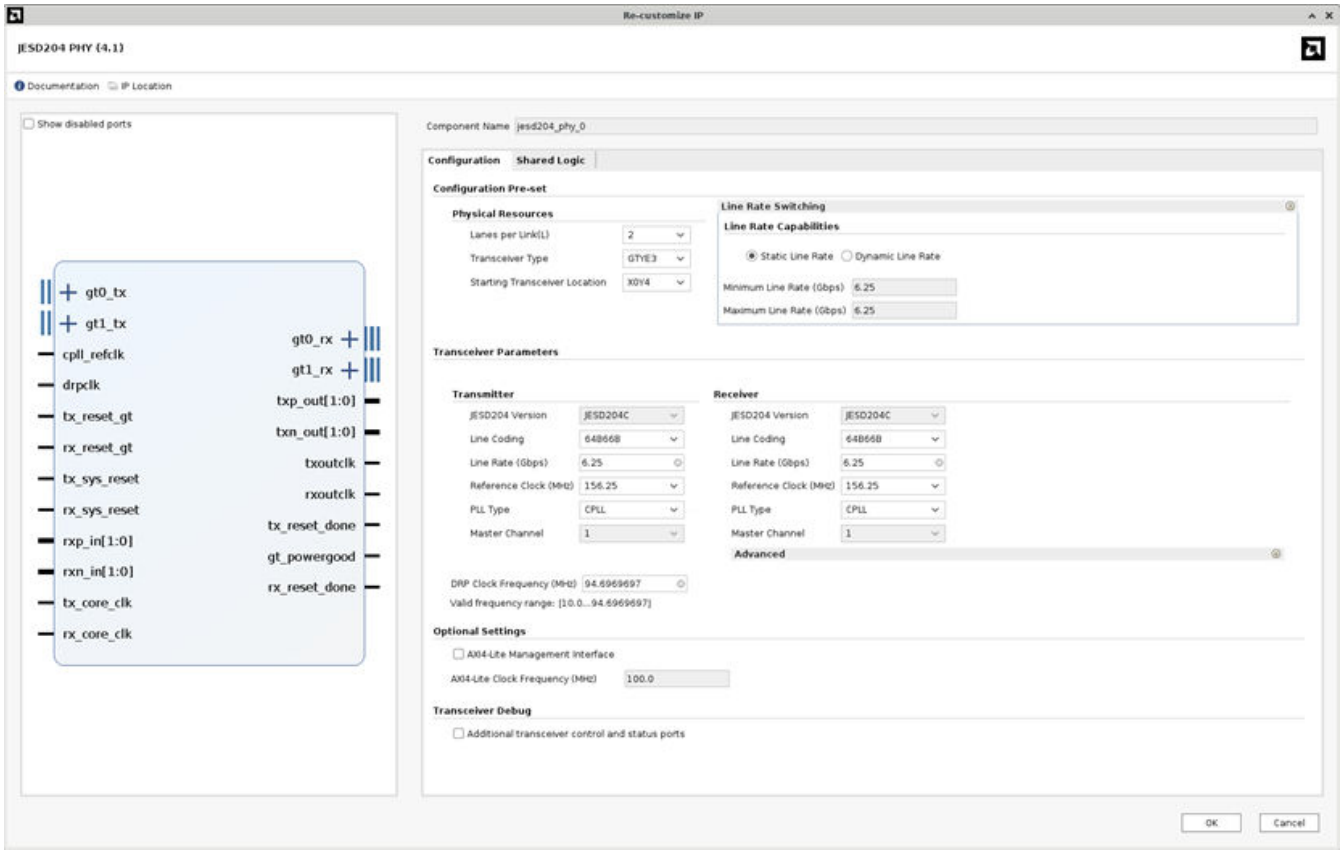
For more information on GT Wizard subsystem, see *Versal Adaptive SoC Transceiver Subsystem Product Guide* (PG442).

Configuring the JESD204 PHY in IP Integrator for UltraScale+/UltraScale Devices

The example design that can be generated for the JESD204C core in Vivado (see Chapter 6: Example Design) delivers a JESD204 PHY core with the settings used from the JESD204C GUI. When configuring a JESD204 PHY core for use with a JESD204C core, the subsequent values must be set as shows in the following image:

- The transceiver type must be set to GTHE3, GTHE4, GTYE3, or GTYE4.
- The JESD204 Version is hard-coded to JESD204C as the JESD204 IP is discontinued.
- The line coding must be set to `64B66B` or `8B10B`.
- The line rate must be set to the same value as defined in the JESD204C GUI.

Figure 48: **JESD204 PHY GUI**

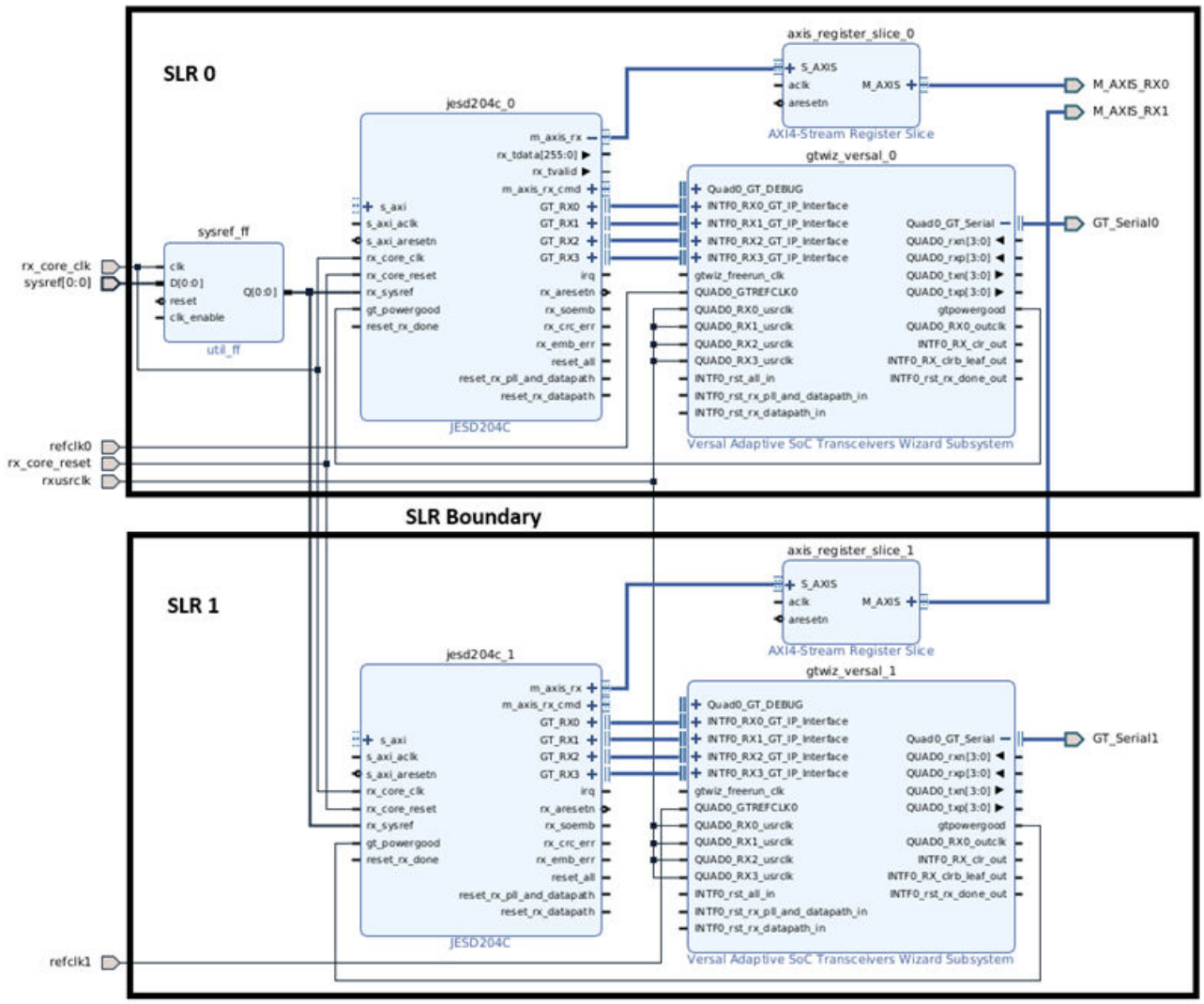


It is possible to share a JESD204 PHY between different instances of JESD204C cores. The TX and RX are configured separately and can therefore run at different line rates and line coding standards.

Using Multiple JESD204C Cores to Connect to One or More ADCs Across SLR Boundaries

In some cases, it might be necessary to connect multiple JESD204C cores to more than one ADC across SLR boundaries, while making sure that there is no latency between them. The following figure shows an example of Multiple RX cores in separate SLRs.

Figure 49: **Two JESD204C RX cores spanning separate SLRs**



Note: Only key ports are connected in this figure for clarity.

Some important points are listed here:

- The whole system is synchronous and as such all clocks should be generated from a common source.
- Each GT quad or GT Wizard subsystem requires its own `refclk`.
- All JESD204C RX cores share a single common `core_clk`:
  - Both separate `refclk` and `core_clk` and `refclk` as `core_clk` clocking schemes are supported. For more information, see Clocking.
  - However if `refclk` as `core_clk` is used then `core_clk` is generated from only one of the `refclk`.
- core_clk is used as the AXI4-Stream clock for both JESD204C RX AXI4-Stream data interfaces.
- To ensure successful capture of SYSREF by both JESD204C RX cores, a D type flip flop should be placed on the input SYSREF signal.
  - The output of this should then be fed into both JESD204C RX cores.
  - The T_{su} and T_{hd} requirements of this flip flop with respect to the `core_clk` must be met.
  - Should there be issues with closing timing for SYSREF over the SLR boundary, additional FFs are needed for the SLR crossing. This additional delay can be compensated for in the SYSREF delay settings between the cores.

The output of both JESD204C RX cores are aligned with no latency differences if the above recommendation are implemented correctly.

To confirm this:

- Monitor the AXI4-Stream `rx_tvalid` signals. They should both be asserted HIGH on the same `core_clk` cycle.
- Monitor the start of EMB signals (64b66b) or start of frame and multiframe signals (8b10b) on each JESD204C RX core. They should be identical.

There are two options to process the received data from each JESD204C RX core:

- The first option is to use the output data separately in their respective SLRs.
- The second option is shown in the previous image, an example of how you can combine the output data into one SLR using AXIS Register Slice IPs configured for SLR crossing.

Constraining the Core

This section describes how to constrain a design containing the JESD204C core. This is accomplished by using the XDC delivered with the core at generation time. An additional XDC file is generated with the IP example design; only the core XDC file should be used in user designs.

Required Constraints

This section defines the constraint requirements for the core. Constraints are provided in several XDC files which are delivered with the core and the example design to give a starting point for constraints for the user design.

There are four XDC constraint files associated with this core:

- `<corename>_example_design.xdc`
- `<corename>_ooc.xdc`
- `<corename>.xdc`
- `<corename>_clocks.xdc`

The first file is used only by the example design; the second file is used for Out-Of-Context support where this core can be synthesized without any wrappers; the third file is the main XDC file for this core. The last file defines constraints which depend on clock period definition, either those defined by other XDC files or those generated automatically by the AMD tools, and this XDC file is marked for automatic late processing within the Vivado design tools to ensure that definitions exist.

Device, Package, and Speed Grade Selections

See the appropriate device data sheet listed in References to determine the maximum line rate supported. Not all devices, packages, and speed grades can operate at the maximum line rate supported by the IP.

Clock Frequencies

The reference clock and core clock frequency constraints vary depending on the selected line rate and reference clock when generating the core. See the generated XDC for details.

Clock Domains

There are also several paths where clock domains are crossed. These include the management interface. See the generated XDC file for details.

Clock Management

Reference clock and core clock resources require location constraints appropriate to your top-level design.

Clock Placement

Reference clock input should be given location constraints appropriate to your top-level design and to the placement of the transceivers.

Core clock input (if required) should be given location constraints appropriate to your top-level design.

Banking

All ports should be given location constraints appropriate to your top-level design within banking limits.

Transceiver Placement

Transceivers should be given location constraints appropriate to your design.

I/O Standard and Placement

All ports should be given I/O standard and location constraints appropriate to your top-level design.

Simulation

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

Synthesis and Implementation

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

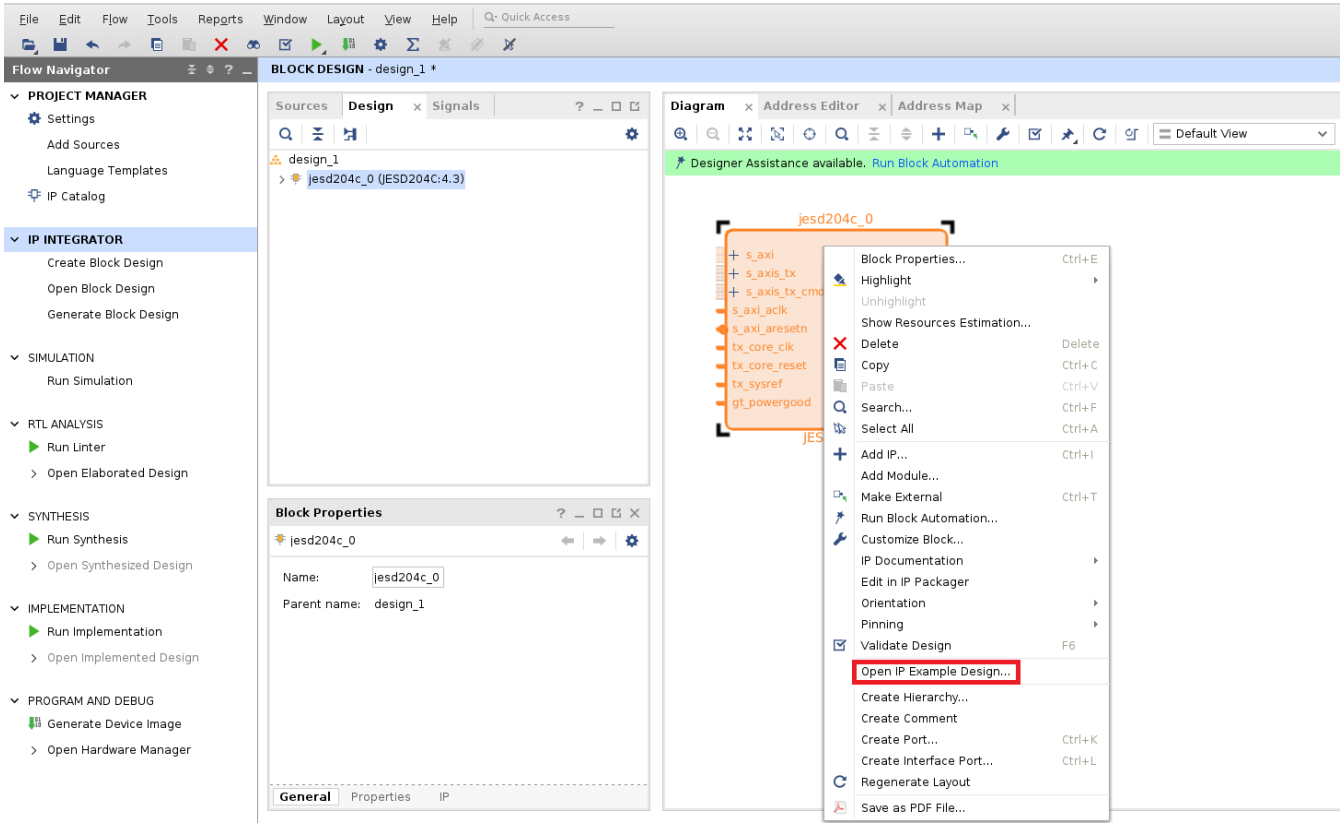
Example Design

The JESD204C IP can be generated as a TX or RX configuration with either 64B66B or 8B10B linecoding. All selections include a lightweight test harness to enable familiarization with the design and signal interface.

To create the example design for AMD Versal™ adaptive SoCs:

1. In the AMD Vivado™ Design Suite, create a new block design in IP integrator.
2. Add the JESD204C IP core to the canvas.
3. Select the JESD204C IP core and configure exactly as required.
4. Right-click the JESD204C IP, and select Open IP Example Design, from the drop-down menu as shown in the following figure. This opens a new Vivado project containing the complete RX or TX design example.

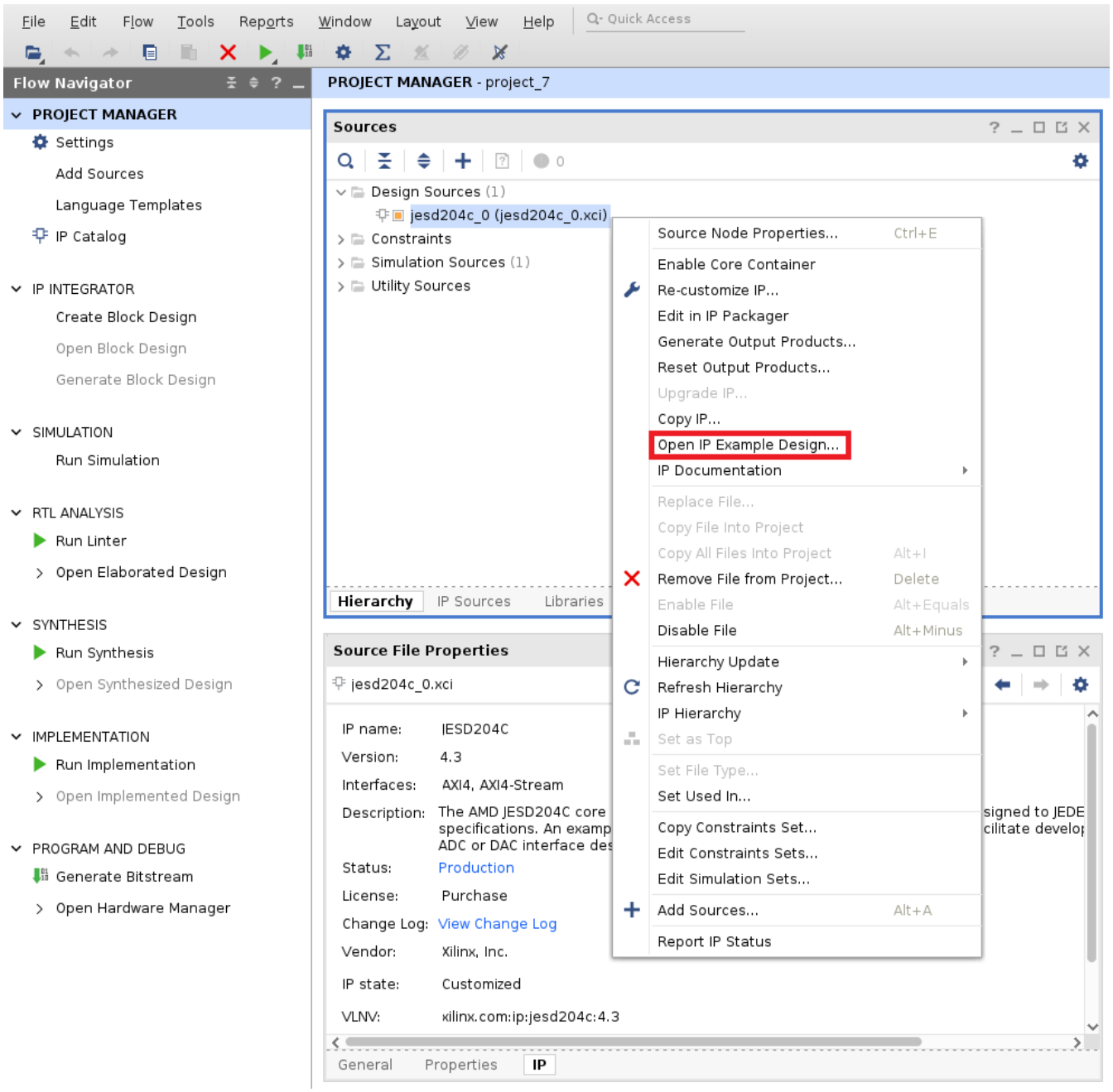
Figure 50: **Opening the Example Design**



To create the example design for AMD UltraScale+™ and AMD UltraScale™ devices:

1. In Vivado, create a new empty project.
2. Select the FPGA part that you wish to use.
3. Using the Vivado IP catalog, select the JESD204C IP core and configure exactly as required.
4. Right-click the block under Design Sources, and select Open IP Example Design, from the drop-down menu as shown in the following figure. This opens a new Vivado project containing the complete RX or TX design example.

Figure 51: **Opening the Example Design**



The following figures show an overview of the example design created for both a JESD204C TX and an JESD204C RX core. The design contains a full transmit and receive path example sharing a Versal Adaptive SoC Transceiver or JESD204_PHY core. The design will generate the JESD204C TX or RX core as configured in the IP catalog. A matching JESD204C TX or RX core will also be generated with settings to complement.

The design is composed of the following main blocks:

- An AXI4-Lite interconnect block to provide multiplexed access to the TX, RX and PHY AXI4-Lite interfaces.
- A simple pattern generator that generates analog sample data and control bits.
- An example mapper that demonstrates mapping the analog sample data and control words into the JESD204C transport layer on the AXI4-Stream interface to drive the TX core.
- (Only in 64B66B cores.) A simple command word generator that passes commands to the TX core. These commands are aligned with the values from the data generator.
- A JESD204C TX core (configuration set in the JESD204C core IP catalog).
- A Versal Adaptive SoC Transceiver core, either GT Quad or GT Wizard subsystem. Configuration is set in the JESD204C core GUI GT Wizard Configuration tab. (Figure 52: Core Example Design for Versal Adaptive SoCs).
- A JESD204_PHY core (configuration set in the JESD204C core GUI PHY Configuration tab) (Figure 53: Core Example Design for UltraScale+/UltraScale Devices).
- A JESD204C RX core (configuration set in the JESD204C core IP catalog).
- An example demapper that demonstrates the AXI4-Stream interface and the JESD204C transport layer back to analog samples and control words.
- A simple pattern checker that checks the received sample and control words for correctness.
- (Only in 64B66B cores.) A simple command word checker that checks the received command word for correctness.

Figure 52: **Core Example Design for Versal Adaptive SoCs**

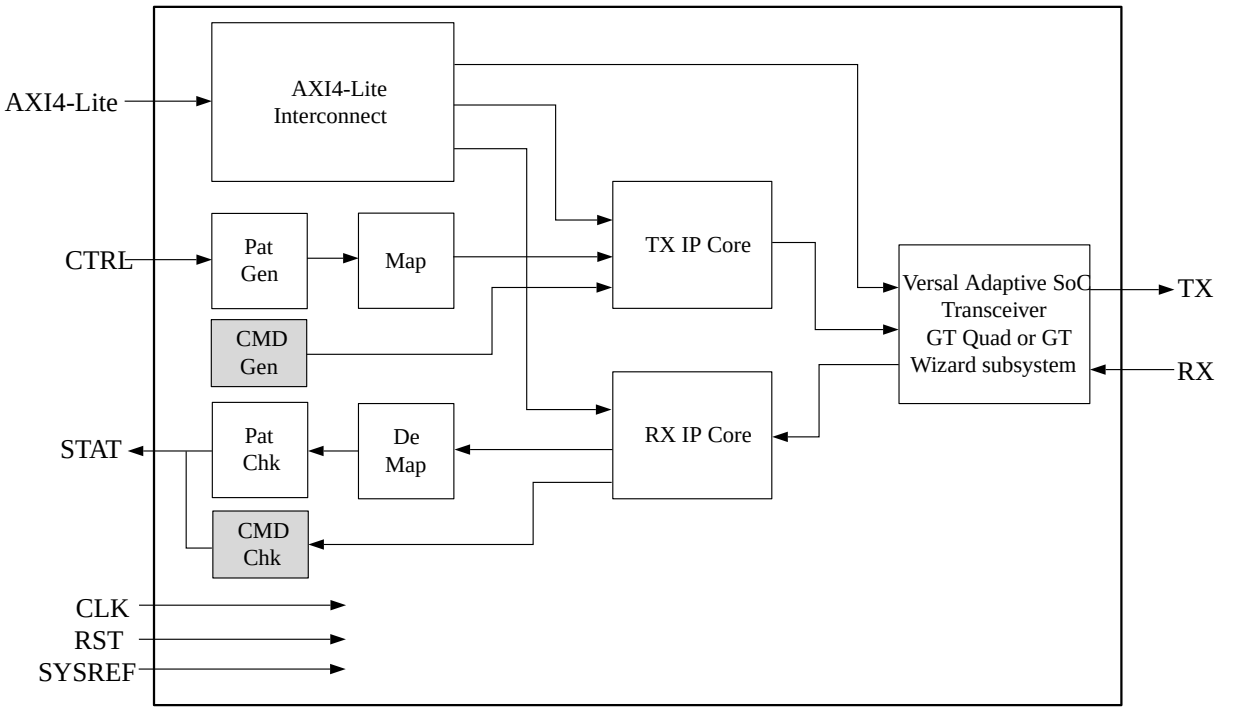
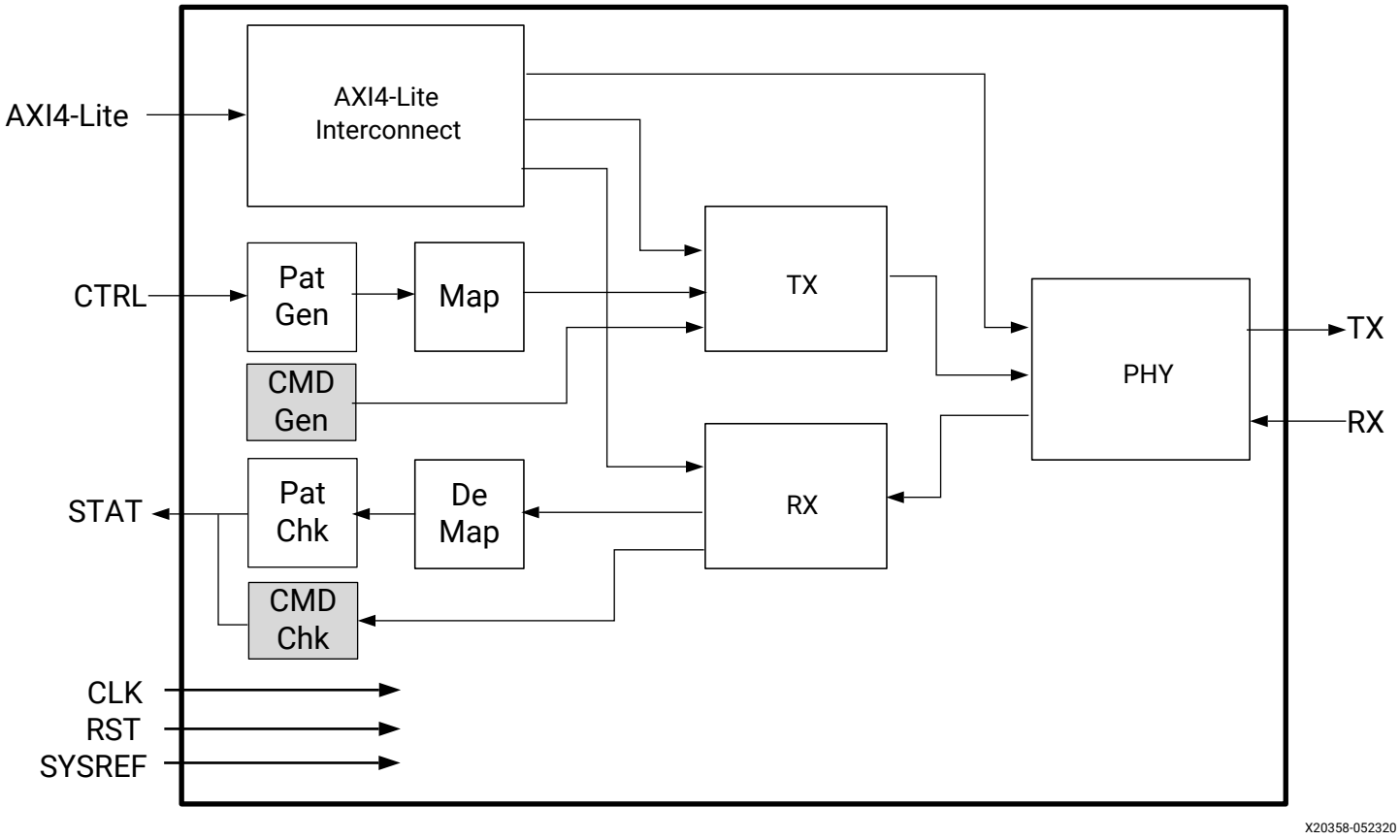


Figure 53: **Core Example Design for UltraScale+/UltraScale Devices**



The example design is intended to be used as a starting point for your custom design.

***Note*:** The I/O clock buffers necessary for the example design to function as a complete FPGA design are instantiated in the top level wrapper that encapsulates this example design.

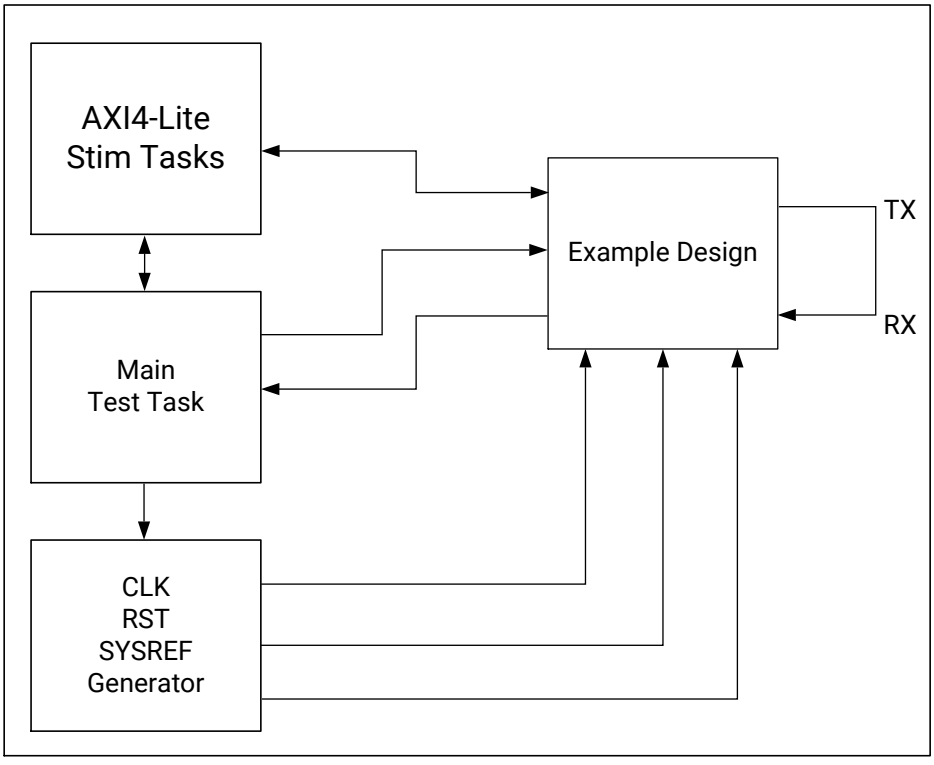
Test Bench

The example design supplied with the JESD204C core provides a complete simulation environment including a demonstration test bench that allows you to simulate the core, and view the inputs and outputs using the AMD Vivado™ Design Suite.

The test bench instantiates the example design described in Chapter 5: Design Flow Steps, and provides the necessary stimulus to show the example design functioning. The test bench can be run at all stages of the design process from behavioral simulation of the RTL code through full post-implementation timing simulation.

The following figure shows an overview of the test bench delivered with the example design.

Figure 54: **Core Demo Test Bench**



Upgrading

Upgrading from v3.0 to v4.0

The TX and RX Command interface AXI4-Stream `tdata` ports have been increased in width to 32 bits per lane. Bits [31:19] are unused.

Upgrading from v2.0 to v3.0

No action is required.

Upgrading from v1.0 to v2.0

New ports have been added to the JESD204 PHY interface for TX and RX as follows:

- `gtN_txcharisk[3:0]`
- `gtN_rxcharisk[3:0]`
- `gtN_rxdisperr[3:0]`
- `gtN_rxnotintable[3:0]`

These ports must be wired to the corresponding ports on the JESD204 PHY core.

Debugging

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

If the IP requires a license key, the key must be verified. The AMD Vivado™ design tools have several license checkpoints for gating licensed IP through the flow. If the license check succeeds, the IP can continue generation. Otherwise, generation halts with an error. License checkpoints are enforced by the following tools:

- Vivado Synthesis
- Vivado Implementation
- write_bitstream (Tcl command)

IMPORTANT! *IP license level is ignored at checkpoints. The test confirms a valid license exists. It does not check IP license level.*

Finding Help with AMD Adaptive Computing Solutions

To help in the design and debug process when using the core, the Support web page contains key resources such as product documentation, release notes, answer records, information about known issues, and links for obtaining further product support. The Community Forums are also available where members can learn, participate, share, and ask questions about AMD Adaptive Computing solutions.

Documentation

This product guide is the main document associated with the core. This guide, along with documentation related to all products that aid in the design process, can be found on the AMD Adaptive Support web page or by using the AMD Adaptive Computing Documentation Navigator. Download the Documentation Navigator from the Downloads page. For more information about this tool and the features available, open the online help after installation.

Answer Records

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

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

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

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

Master Answer Record for the Core

AR 68804.

Technical Support

AMD Adaptive Computing provides technical support on the Community Forums for this AMD LogiCORE™ IP product when used as described in the product documentation. AMD Adaptive Computing cannot guarantee timing, functionality, or support if you do any of the following:

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

To ask questions, navigate to the Community Forums.

Debug Tools

There are many tools available to address JESD204C design issues. It is important to know which tools are useful for debugging various situations.

Vivado Design Suite Debug Feature

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

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

- ILA 2.0 (and later versions)
- VIO 2.0 (and later versions)

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

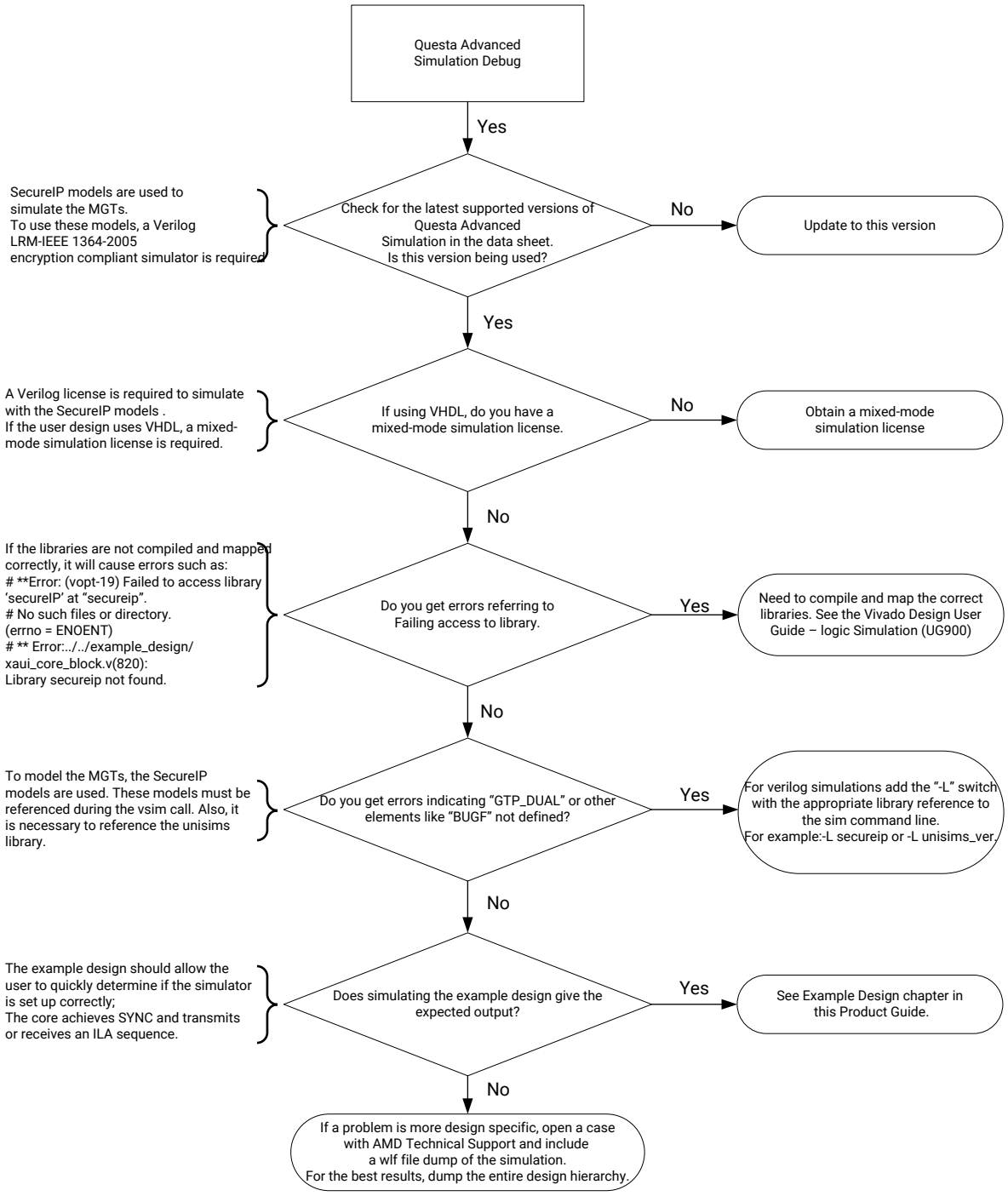
Reference Boards

The AMD VCU118 evaluation board boards support the JESD204CC. This board can be used to prototype designs and establish that the core can communicate with the system.

Simulation Debug

The simulation debug flow for Mentor Graphics Questa Advanced Simulator is illustrated in the following figure. A similar approach can be used with other simulators.

Figure 55: **Simulator Flow Diagram**



X18827-060320

Hardware Debug

Hardware issues can range from link bring-up to problems seen after hours of testing. This section provides debug steps for common issues. The AMD Vivado™ debug feature is a valuable resource to use in hardware debug. The signal names mentioned in the following individual sections can be probed using the debug feature for debugging the specific problems.

General Checks

- For AMD Versal™ Adaptive SoCs, ensure that the core is correctly wired up and that Block Automation has been used to connect up the Versal Adaptive SoC Transceiver when using the Legacy GT Wizard.
- For AMD UltraScale+™ and AMD UltraScale™ devices, ensure that the core is correctly wired up and that the lane based signals are wired to the correct location on the JESD204_PHY.
- Ensure that all the timing constraints for the core were met during implementation.
- Ensure that all clock sources are clean and in particular that the transceiver reference clocks meet the transceiver requirements from the appropriate FPGA Data Sheet.
- Ensure all clock sources are stable before deasserting the external reset signal to the core.
- For UltraScale+ and UltraScale devices, ensure that all transceiver PLLs have obtained lock by monitoring the `QPLLLOCK_OUT` and/or `CPLLLOCK_OUT` port either using the debug feature or by routing the signals to a spare pin.

Issues Obtaining Lane Synchronization

- Ensure that the AXI4-Lite registers have been programmed with the correct value for multi-blocks per extended multi-block.

Interface Debug

AXI4-Lite Interfaces

Read from a register that does not have all 0s as a default to verify that the interface is functional. Output `s_axi_arready` asserts when the read address is valid, and output `s_axi_rvalid` asserts when the read data/response is valid. If the interface is unresponsive, ensure that the following conditions are met:

- The `S_AXI_ACLK` and `ACLK` inputs are connected and toggling.
- The interface is not being held in reset, and `S_AXI_ARESET` is an active-Low reset.

- The interface is enabled, and `s_axi_aclken` is active-High (if used).
- The main core clocks are toggling and that the enables are also asserted.
- If the simulation has been run, verify in simulation and/or the Vivado Design Suite debug feature capture that the waveform is correct for accessing the AXI4-Lite interface.

AXI4-Stream Interfaces

If data is not being transmitted or received, check the following conditions:

- If transmit `tready` is stuck Low following the `tvalid` input being asserted, the transmit core cannot send data.
- If the receive `tvalid` is stuck Low following the `tready` input being asserted, the core is not receiving data.
- Check that the `core_clk` signals are connected to the TX core AXI4-Stream data source or the RX core AXI4-Stream data sink.
- Check that the AXI4-Stream waveforms are being followed (see *Vivado Design Suite: AXI Reference Guide* (UG1037)).
- Check core configuration.

Appendix C

Additional Resources and Legal Notices

Finding Additional Documentation

Technical Information Portal

The AMD Technical Information Portal is an online tool that provides robust search and navigation for documentation using your web browser. To access the Technical Information Portal, go to https://docs.amd.com.

Documentation Navigator

Documentation Navigator (DocNav) is an installed tool that provides access to AMD Adaptive Computing documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

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

***Note*:** For more information on DocNav, refer to the *Documentation Navigator User Guide* (UG968).

Design Hubs

AMD 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.
- Go to the Design Hubs web page.

Support Resources

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

References

These documents provide supplemental material useful with this guide:

1. *Versal Adaptive SoC GTY and GTYP Transceivers Architecture Manual* (AM002)
2. *Versal Adaptive SoC GTM Transceivers Architecture Manual* (AM017)
3. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
4. *Vivado Design Suite User Guide: Designing with IP* (UG896)
5. *Vivado Design Suite User Guide: Getting Started* (UG910)
6. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
7. *ISE to Vivado Design Suite Migration Guide* (UG911)
8. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
9. *Vivado Design Suite User Guide: Implementation* (UG904)
10. *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing* (UG973)
11. *AXI Interconnect LogiCORE IP Product Guide* (PG059)
12. JESD204C Standard (www.jedec.org)
13. *Vivado Design Suite: AXI Reference Guide* (UG1037)
14. *JESD204 PHY LogiCORE IP Product Guide* (PG198)
15. *Versal Adaptive SoC Transceivers Wizard LogiCORE IP Product Guide* (PG331)
16. *Versal Adaptive SoC Transceiver Subsystem Product Guide* (PG442)

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
12/11/2024 Version 4.3	
General updates	Updated to include the new GT Wizard subsystem.

Section	Revision Summary
Register Space	• CTRL_PRBS_MODE to CTRL_TEST_MODE register • Added Table 55: CTRL TX VERSAL GTY/GTYP
Using Multiple JESD204C Cores to Connect to One or More ADCs Across SLR Boundaries	Added this section.
11/06/2023 Version 4.2	
General updates	Added images.
Register Space	Added Table 56: CTRL TX VERSAL GTM
02/01/2023 Version 4.2	
TX Core	Added encoder_rst port for Versal GTM 64B66B designs.
RX Core	Added decoder_rst port for Versal GTM 64B66B designs.
Separate Transceiver Reference and Core Clocks	Changed clock frequencies to reflect higher Versal line rates.
Receive Latency	Added topic.
RX End to End Latency	Added topic.
ADC Timing	Added topic for 8B10B RX latency.
Core Timing	Added topic for 8B10B RX latency.
Calculating End to End Latency	Added topic for 8B10B RX latency.
ADC Timing	Added topic for 64B66B RX latency.
Core Timing	Added topic for 64B66B RX latency.
Calculating End to End Latency	Added topic for 64B66B RX latency.
Transmit Latency	Added topic.
TX End to End Latency	Added topic for 8B10B TX latency.
Core Timing	Added topic for 8B10B TX latency.
DAC Timing	Added topic for 8B10B TX latency.
Calculating End to End Latency	Added topic for 8B10B TX latency.
Core Timing	Added topic for 64B66B TX latency.
DAC Timing	Added topic for 64B66B TX latency.
Calculating End to End Latency	Added topic for 64B66B TX latency.
06/10/2022 Version 4.2	
Clocking	Added 8B10B and 64B66B support with Versal adaptive SoC GTM transceivers.
03/25/2021 Version 4.2	
Chapter 6: Example Designand Connecting to Transceivers Using Block Automation for Versal Adaptive SoCs	• Removed the encommaalign_AXI4l block from the Versal adaptive SoC example designs. • Added Customized Connections option information for block automation.
07/16/2020 Version 4.2	
General updates	Added support for Versal devices.
06/03/2020 Version 4.2	
Configuring the JESD204 PHY in IP Integrator for UltraScale +/UltraScale Devices	• Added support for GTHE3 and GTHE4 transceivers.

Section	Revision Summary
05/22/2019 Version 4.1	
SYSREF	• Added tolerance radius for SYSREF in SYSREF always mode. • Added configurable lanes per link register.
11/14/2018 Version 4.0	
Chapter 6: Example Design	• Modified TX and RX CMD AXI4-Stream tdata ports to be 32-bits per lane. • Modified example design. • Added configuration bits to identify if core was generated with FEC included.
04/04/2018 Version 3.0	
Core Clock 8B10B Linecoding	Added 8B10B linecoding mode.
10/4/2017 Version 2.0	
General updates	• Added ports: gtN_txcharisk[3:0], gtN_rxcharisk[3:0], gtN_rxdisperr[3:0], gtN_rxnotintable[3:0] • Removed reference to individual parts of JESD204C specification.
06/07/2017 Version 1.0	
Clocking	Added GT_POWERGOOD from JESD204_PHY to clocking example description and figure 3-2.
04/05/2017 Version 1.0	
Initial release.	N/A

Please Read: Important Legal Notices

DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

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